Sub-Polyhedral Compilation using (Unit-)Two-Variables-Per-Inequality Polyhedra

Ramakrishna Upadrasta

Comprendre le monde,
construire l'avenir®

THÈSE de DOCTORAT

Université Paris-Sud

école doctorale Informatique de Paris-Sud

Spécialité: Informatique

présentée par

Ramakrishna UPADRASTA

pour obtenir le titre de

Docteur de l'Université Paris-Sud

Sub-Polyhedral Compilation using (Unit-)Two-Variables-Per-Inequality Polyhedra

---

Soutenue le 13 mars 2013 devant le jury composé de :

Président :	Prof. Florent HIVERT	*Université Paris-Sud*
Rapporteurs :	Prof. François IRIGOIN	*MINES ParisTech*
	Prof. Christian LENGAUER	*Universität Passau*
Examinateurs :	Prof. Cédric BASTOUL	*Université Paris-Sud*
	Dr. Antoine MINÉ	*CNRS*
	Prof. Sanjay RAJOPADHYE	*Colorado State University*
Directeur de thèse :	Prof. Albert COHEN	*INRIA*

Thèse préparée à

L'Institut National de Recherche en Informatique et en Automatique (INRIA)

et au

Laboratoire de Recherche en Informatique (LRI) de l'Université Paris-Sud

Acknowledgements

I am sincerely thankful to my advisor Prof. Albert Cohen for all the advise, encouragement and support over the years. I am very happy that his belief in me has led to this work. I consider myself extremely fortunate to have worked under an advisor with a keen intellect along with so many complementary skills, and I am constantly amazed at his productivity as well as amount of simultaneous multi-tasking he is able to do. In spite of all his extremely busy schedule, I am thankful for the time he provided when I needed it. I am also happy that I could find a way of regularly keeping in touch with him, through my weekly ("Brief Report") mails, and get constant feedbacks, some of them which improved my understanding of theoretical concepts, while some kept me more on the practical plane, while others, simply just encouraged me that someone supportive was listening.

I am also very thankful to him for the freedom he gave me in doing research, as well as his respect for my points-of-view, whether in research, official, or personal matters. His constant efforts to support me in many ways increased my productivity by a large amount and pushed me try for the very best: when I opted to stay at INRIA-Saclay when the rest of his team moved to ENS, when I opted to target for the high risk POPL conference, or when I opted to move back to India due to bronchitis during the peak thesis writing time, etc.

It was indeed a life-saving as well as career-saving event when I said yes to him when he asked me to join first as a research-engineer, and later as a doctoral student in Paris. *Merci beaucoup* Albert, for all the years. I am sure many people are thankful to you, and blessing you for helping me cross the barrier of PhD.

I am thankful to Prof. François Irigoin and Prof. Christian Lengauer, two well respected researchers, as well as heads of successful teams at the forefront of polyhedral community research, for gladly and kindly accepting to be reviewers of my thesis. Since a part of the work began with my trip to Passau in early-Spring 2011, it seems just right that Prof. Christian Lengauer is a reviewer of this thesis. I wish to thank Prof. François Irigoin for the detailed and extensive comments on the initial version of manuscript which lead to the current exposition to be clearer and more precise, as well as more emphatic as he wanted it to be.

I am also thankful to Prof. Cédric Bastoul, Prof. Florent Hivert, Dr. Antoine Miné, and Prof. Sanjay Rajopadhye, for kindly accepting to be on my committee.

I wish to sincerely thank Prof. Cédric Bastoul for his help at many times. His encouragement to show practical evidence on many of my ideas, including taking a written promise from me which says that I

will do empirical work as part of my thesis was instrumental in me doing so. In times of frustration, his practical knowledge of polyhedral tools like PIP, PolyLib, and CLooG was a life-saver. I am also happy that this dissertation contains the code-generation chapter which extends his work in a small way. This chapter owes a lot to many extended ("two minute") discussions with him, till the date when he had to devote his entire time for his own HDR dissertation.

I am also thankful to Dr. Antoine Miné for accepting to be on the committee as this dissertation owes much to the success of his path-breaking work on Octagons in static analysis. The exposition of his work in his various papers, as well as his excellently written dissertation definitely had a very strong influence on my papers as well as this dissertation, for which I am very thankful for. I am also thankful to him for his detailed comments on the dissertation.

I am very thankful to Prof. Florent Hivert for accepting to be an examinateur of the dissertation as well as the president of the committee.

I am very glad that I could have a dissertation with Prof. Sanjay Rajopadhye, an inventor of the polyhedral model, on my committee. In spite of the misfortune of not being able to complete my dissertation under him, the way events play with people in an unfathomable way that I could submit a little work like this with him on my committee is a proof of cyclical nature of time and divine providence. The patience and kindness with which he dealt with me when I left Colorado State will stay with me forever.

I am very thankful to Prof. Paul Feautrier, the father of affine scheduling frameworks in polyhedral compilation for his constant guidance and criticism throughout his thesis. His insights and precise feedback at many times, while answering my questions over email, helped both in my theoretical as well as firm practical understanding.

In a very similar way, my discussions with Armin Größlinger beginning from his stay in INRIA during our daily "water breaks", followed by my memorable trip to Passau in 2011, helped a lot in fine-tuning of my ideas. I am also thankful that we could continue exchanging ideas when he joined in Passau later. *Danke* Armin, for helping in fine-tuning my ideas, explaining things, and being patient in acting as sounding board for some of my newer ideas.

I am thankful to all my team members in INRIA: Konrad Trifunovic, Boubacar Diouf, Michael Kruse, Anna Beletska, Mohamed-Walid Benabderrahmane et al. from the ALCHEMY from the INRIA Saclay-Île-de-France group; Tobias Grosser, Feng Li, Riyadh Baghdadi, Léonard Gérard, Adrien Guatto, Antoniu Pop, Cédric Pasteur et al. from the PARKAS group at Ecole Normale Supérieure at rue d'Ulm; and members of the new Archi group at PCRI, LRI, Université Paris-Sud-11 for all their help and support. In particular, I should mention Konrad, Tobias, Boubacar, Michael, Léonard and Adrien as well as Benoît Pradelle from the ICPS group at Strasbourg for their particular interest in my work with a hope that it will work. Thanks as well to Prof. Mark Pouzet, Dr. Louis Mandel, and Prof. Christine Eisenbeis for various kinds of support and interest in my work.

I am also extremely thankful to Lakshminarayanan Renganarayana, Gautam Gupta and Dae Gon Kim from Colorado State University for their support, understanding and co-learnings. I am glad that all of them have been very successful in their own different ways. Lakshminarayanan ("Lakshmi"), not only being my senior from IISc, was indeed like a big brother to me in many ways and I am happy that he is now a

successful researcher in IBM. I am very glad to have found a good friend in Gautam, who is now a manager of successful startup in the polyhedral model. Thanks to Dae Gon as well. The support they provided to me when I was at CSU was simply invaluable, and the sendoff their families gave me was unforgettable. I consider myself fortunate to have learned from them in my little own ways.

Thanks also to my friends and past colleagues: Sandya S. Mannarswamy, Nandivada Venkata Krishna, Manjunath Kudlur, Rajarshi Bhattacharya, Anmol Paralkar, Surya Jakhotia, and Ravi Parimi for their belief and trust in me.

I am very thankful to the numerous people in INRIA who made my life easier through their special efforts. Special mention to Valérie Berthou for going out of her way to help me in numerous mundane matters in France. Indeed, the bureaucratic hurdles could only be surmounted only with her help. Thanks also to the various library managers at INRIA-Saclay and INRIA-Rocquencourt for letting me keep a huge volume of books for an extended amount of time.

I am very thankful to Dr. Dibyendu Das for acting as a very patient mentor during our days at Hewlett Packard (HP) Compiler optimization group in Bangalore, and keeping faith in me in many ways afterwards. I am very glad that we could continue to work together in a fruitful ways primarily due to his efforts even after we continued to work at different places and far away geographical locations.

I am also sincerely thankful to Dr. Dan Quinlan who was my mentor in Lawrence Livermore National Laboratories (LLNL), Livermore, California for a warm extended stay in the ROSE compiler team. I joining LLNL in Livermore, California was a career, and life changing point in multiple ways.

I am also extremely thankful to Axel Simon both for his inspiring work on TVPI polyhedra in static analysis as well as his personal interest and feedback on our IMPACT-11 paper.

Furthermore, I am also thankful to Aditya Vithal Nori, my senior from Andhra University as well as IISc, and now a successful researcher in Microsoft Research, for guidances in many ways at various times and for inspiration to tread to do bold things the hard way in a honest manner. Without his inspiration, I could not have changed my fields so many times and been successful in my little ways.

I am glad that I came to France ("*La France*") to do PhD, which was the most perfect place in anywhere in the world to work on polyhedral compilation. This was advantageous in multiple ways: Firstly I consider it a divine coincidence for my PhD work to coincide with the new International Workshop on Polyhedral Compilation Techniques ("IMPACT") which began with 2011, where I could publish the initial versions of my work. In spite of the polyhedral community being a well knit community in Europe sticking to the highest standards of research in compilers, this particular field was relatively under-represented in the general compiler community. So, having the opportunity of publishing in IMPACT-workshop meant that I had excellent anonymous and objective feedback on my initial works, which saved me from the rather difficult effort of selling these ideas to outside compiler community.

Secondly, I am also thankful to the polyhedral community for the support through their tools. In this, I should particularly mention Uday Bondhugula for his state-of-the-art and successful PLuTo source-to-source compiler from which this work benefitted a lot, and Louis-Noël Pouchet for his work on many tools

including FMLib and PolyBench. I am also thankful to Prof. Vincent Loechner for the various help given on the PolyLib library, as well as members of the polymake community for many little tips with their software.

Finally, thanks to all the anonymous reviewers of our papers in IMPACT-11 and IMPACT-12, especially the later work, which gave us sufficient confidence that our work was mature enough for a prestigious conference like POPL.

I am very thankful to Prof. Dorit Hochbaum for her personal feedback as well as her various papers on TVPI polyhedra. I am also sincerely thankful to Prof. Alain Darte, Prof. Gérard Cornuéjols, Prof. Kevin Wayne and Prof. Günter Ziegler, for their works, valuable feedback, as well as explanations at much needed times. Thanks also to the reviewers of ACM-POPL for their feedback on our work, as well as its acceptance.

I would also like to thank Prof. Don Knuth many things, including his morale-boosting checks recorded in my account in his *Bank of San Serriffe*.

Most of my research life owes a lot to the initial belief Prof. Priti Shankar ("Madam"), my advisor at the Indian Institute of Science (IISc) had in me. Her encouraging words through emails, including her last one in spite of her life-taking illness were morale boosters. I am sure she is happy at my current progress, while teaching formal methods and compilers to well deserving students in other worlds.

I also thank many of my past teachers at alma meters: Prof. Ross McConnell, Prof. Edwin K. P. Chong, Prof. Darrell Whitley and Prof. Chuck Anderson at Colorado State University in Fort Collins; Prof. Y.N. Srikant, Prof. Ramesh Hariharan, Prof. V. Vinay, Prof. Vijay Chandru at IISc in Bangalore; and Prof. P.V. Ratnam, Prof. V. Ramamurty, and many professors at Andhra University in Visakhapatnam.

I also thank Shri. V. Subramanian ("Subbu-ji"), Prof. V. Krishnamurty ("Prof. VK"), Dr. Vidyasankar Sundaresan and Shri. Ramesh Krishnamurty for inspiring me with their writings on Advaita Vedanta as well as being guides in various vedanta related matters, in the spirit of *satsaNgatve nissN^gatvaM*. I am eternally thankful to Shri Subbu-ji for that eventful trip to Sringeri, during which I could personally meet the jagadguru and *sha.nkaraachaarya of the daxiNAMnAya-shaarada-maTh shrii shrii shrii bhaaratii tIrtha svaaminaH* and obtain His blessings.

Reflecting the upaniShadic *yato-vaacho-nivartante*, no words can even my thanks to the member of my family and gurus. I am thankful to Brahma Shri Veda Murti Pandit Narendra Kapre-ji, a traditional scholar of *rig-veda* for teaching me veda, or at least enough to the capacity of my interest (*shraddha*) and learning capacity (*grahaNa shakti*). Convincing me that being born in a vedic family was not sufficient, and age is never a deterrent to becoming a student of *rig-veda* (*svashaakha*) which should be learned and thereby pay the *RRiShi-RRiNa*, as well as constant support of my readings of scriptures of *bhaarata-desha*, with an emphasis on Adi Shankaracharya's advaita-vedanta works, in their *original,* meaning in *samskRita*, made me a student of *sanaatana-dharma* to the level of my competence.

I am very thankful to my grandparents for implanting the seeds for following *sanaatana-dharma* in me.

I also thank my sister ("Akka") Madhavi Upadrasta, a well qualified Mechanical Engineer, and my brother-in-law Venkat Bhamidi, who holds a PhD in Chemical Engineering, for supporting me through many ups and downs. Having a sister like the one I have is a divine blessing, for she not only acts the part of elder son of my parents, but also as my sister as well as supporting me in many mundane matters, while working through the ups and downs of her own life. I am sure that the result of her hard work is also quite near.

Though any such attempt would fall short, I am sincerely thankful to my parents my mother ("Amma") Yasoda Upadrasta, my father ("Daddy") Suryanarayana Upadrasta for all the patience they have shown with me, along with giving me extraordinary encouragement to aim for the best, while all along continually and silently making sacrifices for their children, and for their all giving nature for the sake of larger *dharma*.

I am always surprised at how their curious blend of idealism in my mother, and down-to-earth policy of my father has worked so much for my personal benefit. I am very happy that I could sit in my home under their care in the final crucial months of my thesis and work in an uninterrupted way. Just like my previous achievements, I hope the success of this little work would make them happy as well, once more validating their unwavering belief in me.

–

dhanyo.smi

nandana-naama-samvatsara-maagha-bahuLa-chaturdashi
(mahA-shivaraatri)

Dedication

To my parents and all my teachers

tasmai shrii gurumUrtaye nama idam shrii daxiNAmUrtaye
(shivaarpaNaM)

Abstract

The goal of this thesis is to design algorithms that run with better complexity when compiling or parallelizing loop programs. The framework within which our algorithms operate is the *polyhedral model of compilation* which has been successful in the design and implementation of complex loop nest optimizers and parallelizing compilers. The algorithmic complexity and scalability limitations of the above framework remain one important weakness. We address it by introducing *sub-polyhedral compilation* by using *(Unit-)Two-Variable-Per-Inequality* or (U)TVPI Polyhedra, namely polyhedra with restricted constraints of the type $ax_i + bx_j \leq c$ ($\pm x_i \pm x_j \leq c$).

A major focus of our sub-polyhedral compilation is the introduction of *sub-polyhedral scheduling*, where we propose a technique for scheduling using (U)TVPI polyhedra. As part of this, we introduce algorithms that can be used to construct under-aproximations of the systems of constraints resulting from affine scheduling problems. This technique relies on simple polynomial time algorithms to under-approximate a general polyhedron into (U)TVPI polyhedra. The above under-approximation algorithms are generic enough that they can be used for many kinds of loop parallelization scheduling problems, reducing each of their complexities to asymptotically polynomial time.

We also introduce *sub-polyhedral code-generation* where we propose algorithms to use the improved complexities of (U)TVPI sub-polyhedra in polyhedral code generation. In this problem, we show that the exponentialities associated with the widely used polyhedral code generators could be reduced to polynomial time using the improved complexities of (U)TVPI sub-polyhedra.

The above presented sub-polyhedral scheduling techniques are evaluated in an experimental framework. For this, we modify the state-of-the-art PLuTo compiler which can parallelize for multi-core architectures using permutation and tiling transformations. We show that using our scheduling technique, the above under-approximations yield polyhedra that are non-empty for 10 out of 16 benchmarks from the Polybench (2.0) kernels. Solving the under-approximated system leads to asymptotic gains in complexity, and shows practically significant improvements when compared to a traditional LP solver. We also verify that code generated by our sub-polyhedral parallelization prototype matches the performance of PLuTo-optimized code when the under-approximation preserves feasibility.

Résumé

Notre étude de la compilation sous-polyédrique est dominée par l'introduction de la notion l'ordonnancement affine sous-polyédrique, pour laquelle nous proposons une technique utilisant des sous-polyèdres (U)TVPI. Dans ce cadre, nous introduisons des algorithmes capables de construire des sous-approximations de systèmes de contraintes résultant de problèmes d'ordonnancement affine. Cette technique repose sur des algorithmes polynomiaux simples pour approcher un polyèdre quelconque par un polyèdre (U)TVPI. Nos algorithmes sont suffisamment génériques pour s'appliquer à de nombreux problèmes d'ordonnancement, de parallélisation, et d'optimisation de boucles, réduisant leur complexité temporelle à des fonctions polynomiales.

Nous introduisons également une méthode pour la génération de code utilisant des algorithmes sous-polyédriques, tirant parti de la faible complexité des sous-polyèdres (U)TVPI. Dans ce cadre, nous montrons comment réduire la complexité associée aux générateurs de code les plus populaires, ramenant la complexité de plusieurs facteurs exponentiels à des fonctions polynomiales.

Nombre de ces techniques sont évaluées expérimentalement. Pour cela, nous avons réalisé une version modifiée du compilateur PLuTo, capable de paralléliser et d'optimiser des nids de boucles pour des architectures multi-cœurs à l'aide de transformations affines, et notamment de partitionnement (tiling). Nous montrons qu'une majorité des noyaux de calcul de la suite Polybench (2.0) peut être manipulée à l'aide de notre technique d'ordonnancement, en préservant la faisabilité des polyèdres lors des sous-approximations. L'utilisation des systèmes approchés par des sous-polyèdres conduit à des gains asymptotiques en complexité, qui se traduit par des réduction significatives en temps de compilation, par rapport à un solveur de programmation linéaire de référence. Nous vérifions également que le code généré par notre prototype de parallélisation sous-polyédrique est compétitif par rapport à la performance du code généré par Pluto.

Table of Contents

Notations and Abbreviations

(U)TVPI	(Unit-)Two-Variables-Per-Inequality constraint/polyhedron
$\mathbb{Q}^n, \mathbb{Z}^n$	Rationals/Integers in n-dimension
$\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{Q}^n$	n-dimensional (column) vectors
λ, μ	Variables of the Farkas polyhedron
$A, B \in \mathbb{Q}^{m \times n}$	$m \times n$-size matrices
$A^T, B^T \in \mathbb{Q}^{n \times m}$	$n \times m$-size (transpose) matrices
$P, \mathcal{P}, \mathcal{D} \subseteq \mathbb{Q}^n$	n-dimensional (rational or integer) polyhedra
$K \subseteq \mathbb{Q}^n$	n-dimensional (homogeneous) cone ($\mathbf{0} \in K$)
$C^*, K^* \subseteq (\mathbb{Q}^n)^*$	n-dimensional polar cones
m, n	Number of constraints and variables/dimensions (respectively) of a polyhedron
(U)TCPV	(Unit-)Two-Column-Per-Vector: A column vector with at most two non-zero elements.
UA/OA	Under/Over-Approximation
TU	Totally Unimodular
V, E	Vertices and Edges of the (usually dependence) graph
$\|V\|, \|E\|$	Cardinality of the Vertex and Edge sets
$Z(m, n)$	Running time of Simplex with m constraints and n variables
conv, cone	Convex and Conical Sums
$\mathcal{H}, \mathcal{V}$	Halfspace (constraint) and Generator (Vertex) form of a Polyhedron
$\|\mathbf{a}_l\|$	Number of non-zero elements in vector $\mathbf{a}$
$\widehat{\|\mathcal{S}\|}$	Average number of non-zero coefficients/elements in the non-TVPI constraints in $\mathcal{S}(\mathbf{x})$.

Chapter 1

Introduction

1.1 Motivation

Automatic compilation and translation of programs to newer architectures so that the program runs efficiently exploiting all the resources of the machine has been an important problem since the advent of programming languages. In particular, the recognition of the difference between a language which a programmer "sees and thinks in" and a language which is particularly suited for the architecture has led to the understanding of a program transformations and the so called program optimizations. While the concept of program transformations is well understood using a concept of equivalence between the input and output programs, the concept of optimizations is harder to define with many times such definition being architecture dependent. With the modern day architectural challenges like decline of Moore's law and the rise of multi-core machines—which have large computing powers equivalent to earlier era super-computers—which demand proper exploitation, defining optimizations has become a demanding, specialized and hard to solve task, even after decades of research.

The concept of these program optimizations has led to reasoning that iterative computation, usually expressed as equivalents of `for` loops are best cases for optimization. Though many techniques have been introduced and evolved over decades, they have relatively been adhoc, lacking in formalism and completeness. The formal way to understand loop optimizations is now agreed upon to be a model termed as polyhedral compilation. This formalism, though extremely powerful to encode many interesting program transformations, is quite costly for large input programs making the overall optimization process and hence the compiler unscalable. In this thesis we contribute to polyhedral compilation by introducing an algorithmic analysis and scalability driven approximations so that a precision vs. cost tradeoff can be made by the optimizing compiler.

1.2 Key Concepts

Compilers. *Compilation* is a branch of computer science that is devoted to transforming a computer program in a source language to execute on a target architecture, with *compilers* being the associated systems software. Though compilers usually perform several analysis and verifications on the input program,

they are primarily used for the process of translation of a program from a source language—usually, like C/C++/FORTRAN—to a prespecified target language. Sometimes, the source and target languages are the same, in which case it is called a source-to-source compiler, with its output usually being linked to a regular out-of-the-box compiler. The process of compilation usually involves an intermediate language, or an intermediate form like an abstract syntax tree, which can represent the input program correctly.

Iterative regular computation. The seminal work of Karp Miller and Winograd [KMW67] introduced a model—called Systems of Uniform Recurrence Equations (SUREs)—which can be used to describe a set of regular computations as mathematical equations. Many key concepts in the reasoning of loop programs can be traced back to the above paper. The model of SUREs has been extended to Systems of Affine Recurrence Equations (SAREs) so that a restricted variety of `for` loop programs expressed in a language like C—with restrictions on the kinds of ranges of the iterator variables, the kinds of array accesses, the memory model of the variables etc.—can be represented, analyzed and transformed.

Loop program optimization. An *optimization* of an input program is understood to be its improvement on a pre-defined metric, like execution time, memory footprint, registers needed etc. Most compiler optimizations are inherently linked to a concrete measure of improvement, usually involving the architectural characteristics. In earlier times, advanced compiler optimizations were reserved for supercomputers [Wol89]. But, with the widespread use of new powerful architectures, they have become a core part of systems research with many challenging problems to be solved. There have been many compiler optimizations, like software pipelining, that have been designed to improve the performance of the output program when compared to the input program, but most of them bring about only a constant fold improvement. It has however been well recognized that optimization of iterative or regular computation expressed as loop programs is the best way to bring about an *asymptotic improvement* in the running time of the input program. So, many individual optimizations, like loop distribution, permutation, unimodular transformations, shifting, etc. have been developed over the history of loop optimization. But, most of them are lacking in formalism and completeness.

Polyhedral compilation. *Polyhedral compilation* is the compilation of loop programs using polyhedral (geometric) representations, and subsequent transformations obtained by rational and integer linear programming. It is generally accepted that the *formalism* in polyhedral compilation which is rooted in the above cited Systems of Recurrence Equations work of Karp et al. is complete as well as being powerful enough to encode many varieties of which are powerful affine transformations [DRV00]. One traditional view is the polyhedral compiler should be designed to directly take in SAREs expressed in a high level language, but the more practical view is that the input pieces of code—also called Static Control Parts of a larger program (or SCoPs)—are usually expressed in a language similar to C, but have restrictions so that equivalent SAREs could be derived from them. In this latter method, polyhedral compilation follows a three phase process. The first phase is a *dependence analysis* phase where the input program is analyzed and an equivalent polyhedral representation for it is constructed where the domains and dependences of the input program are representable by rational parametrized polyhedra [Fea91]. This is followed by a *schedul-*

ing or transformation finding phase [DRV00] where a proper optimization is found using the model defined by the architectural and other constraints. This is further followed by a *code generation* phase where the schedules are applied to the input program model to generate the transformed program [QRW00].

Affine scheduling. In polyhedral compilation, following the pioneering work of Feautrier [Fea92a, Fea92b] the constraints of the input program are translated to a new space defined by application of a LP duality transformation called *affine form of Farkas lemma*, and this new polyhedron ("Farkas polyhedron") searched for the proper transfomations. Feautrier also defined *affine scheduling and affine transformations*, in which the transfomations are encoded by an affine function of the Farkas polyhedron. Affine transformations not only have the advantage that searching for them simply involves solving a linear program, but also they can be linked to a conceptually simple code generation scheme which leads to code that could even be directly translated to hardware if needed to. Because of the fact that affine scheduling is both a scheduling algorithm as well as a scheduling framework, the work of Feautrier's model has been extended by Griebl et al. [GFG02]. The latter defined permutability conditions and subsequent space-time transformational model named *Forward Communications Only (FCO)*, which is particularly suited for tiling [GFL04]. This FCO transformation has been re-defined with a cost function particularly amenable for multi-core machines and implemented in the *PLuTo* source-to-source compiler by Bondhugula et al. [BHRS08]. The model of PLuTo has been very successful with variations of its algorithm having been implemened in various compilers across industry: IBM's XL, Reservoir Lab's R-Stream, GCC's GRAPHITE and more recently in LLVM's Polly compilers.

Code generation. After finding an optimizing transformation for the input program, the final step in polyhedral compilation is *polyhedral code generation*, which is defined to be finding a scanning order for the transformed program by touching each integer point in each of the domains exactly once, while respecting the dependences between the statements. Though there have been many scheduling algorithms, and there will be more in the coming times, the *algorithm for scanning polyhedra* by Quilleré et al. [QRW00], is the only one which can generate exact code. The above algorithm's implementation in CLooG code generator [Bas04a] has comprehensively solved the code generation problem and is arguably the first module that is put in when designing a new polyhedral compiler.

Sub-polyhedral compilation. The practical affine scheduling algorithms involve solving large (rational) linear programs whose size is dependent on the size of the input program. And, the polyhedral code generator algorithm involves polyhedral operations which take exponential time. Hence, both the scheduling and the code generation modules of polyhedral compilation are well understood to be asymptotically quite costly and hence unscalable when the input programs are large. The subject of this thesis is the introduction of *sub-polyhedral compilation* so that the asymptotic complexity of these two critical modules can be reduced by using approximations of polyhedra or *sub-polyhedra*. The particular variety of sub-polyhedra that we focus on is *(Unit-)Two-Variables-Per-Inequality sub-polyhedra* which are specialized or restricted form of polyhedra, where each of the constraints could be only of the form $ax_i + bx_j \leq c$ ($\pm x_i \pm x_j \leq c$). These varieties of polyhedra, because of the binary nature of their constraint relations, have the advantage

that they can be represented using a graph theoretic encoding. Optimization and feasibility problems on systems of constraints involving (U)TVPI polyhedra can be solved using problems like min-cost-flow and Bellman-Ford algorithms, with the time complexity of these algorithms being asymptotically polynomial time. Though these sub-polyhedral methods are powerful enough to encode interesting program transformations, they are lesser in power and precision than ones using general polyhedra, with a consequent gain in time complexity. So, use of sub-polyhedral compilation methods gives the compiler writer a unique *precision vs. cost tradeoff*, which she can factor in while designing polyhedral scheduling and polyhedral code-generation problems.

1.3 Overview of this Thesis

This thesis is organized as follows.

In Chapter 2, we first give some basics of polyhedral compilation and affine scheduling. We then show a theoretical analysis of the asymptotic complexity of the unscalability problem in scheduling, followed by an empirical measurement of the affine scheduling methods currently implemented in PLuTo; we show that they do not scale for large versions of typical input programs.

In Chapter 3, we study various classes of sub-polyhedra, the time complexities of their operators and predicates, along with their use in static analysis. After reviewing the many varieties of sub-polyhedra that have been used by the static analysis community for abstract interpretation purposes, and the complexity theory community for algorithmic improvements, we focus on (Unit-)Two-Variables-Per-Inequality polyhedra; namely, ones with constraints of the form $ax_i + bx_j \leq c$; $a, b, c \in \mathbb{Q}$ (and $\pm x_i \pm x_j \leq c$; $c \in \mathbb{Q}$ respectively). After the above study of sub-polyhedra purely on an asymptotic analysis purposes, in Chapter 4, we study whether any approximations have already been used in polyhedral scheduling. After understanding the limitations of some of the major varieties of polyhedral approximations, we introduce a new variety of affine scheduling: *polyhedral scheduling using (U)TVPI sub-polyhedra.*

The above scheduling technique needs under-approximation algorithms, and hence, the next two chapters focus on developing new algorithms which *under-approximate a general polyhedron into (U)TVPI sub-polyhedra.* To do this, in Chapter 5 we introduce a duality based framework that is particularly suited for our purposes. In Chapter 6, we design Under-Approximation algorithms. These algorithms run in polynomial time and provide a method by which the UA algorithms could be applied on a per-dependence basis, or on the overall Farkas polyhedron.

In Chapter 7, we present *experimental results* showing the effectiveness of our mentioned methods by implementing them in PLuTo, a state-of-the-art compiler for tiling and parallelization purposes. This experimentation is based both on a polyhedral theoretical sense as well as from the perspective of compilation, namely, feasibility study of the above suggested under-approximation algorithms, improvements in the execution time of the scheduler, and execution time of the generated code.

In Chapter 8, we study various *loop optimization applications* that use the above UA framework to improve their worst-case asymptotic complexity. The complexities of the original problems range from weakly polynomial time to NP-Complete, while the complexities of the algorithms that we suggest are always of

fixed weakly or strongly polynomial time. In Chapter 9, we introduce *sub-polyhedral code-generation;* a preliminary approach to see how the lower complexities of (U)TVPI sub-polyhedra can be used to obtain better worst-case bounds for polyhedral code generation, albeit for loss of precision in the form of reduction of quality of the generated code, and increased control overhead.

In Chapter 10, we provide some conclusions and perspectives.

The problems being solved in this thesis, the approaches and results by which it was influenced, and the contributing factors therein, are briefly summarized in Figure 1.1.

1.4 Our Contributions

One important goal during this thesis was to introduce the scalability analysis of parallelization of loop programs that is firmly rooted in complexity theoretic asymptotic analysis. In particular, our contributions are theoretical and algorithmic and are of practical interest as well. Our work takes in ideas from complexity theoretic community, static analysis community, and the polyhedral loop optimization community. Here are our contributions:

- We show, both theoretically and empirically, that the current methods used in polyhedral compilation are unscalable. The above leads to proposal of sub-polyhedral compilation strategies. They introduce the concept of trading precision for execution time in a complexity guided manner, with a compilation-time scalability motivation.

- Of the many sub-polyhedra used in static analysis, we show that Two-Variable-Per-Inequality (TVPI) and Unit-Two-Variable-Per-Inequality (UTVPI) sub-polyhedra can provide good alternatives to be used in polyhedral scheduling, limiting its worst-case complexity. We show that state-of-the-art parallelization and affine scheduling heuristics such as PLuTo can be adapted to (U)TVPI sub-polyhedra, thereby reducing their algorithmic complexity. We propose two alternative algorithms—either with a direct Under-Approximation of all the feasible schedules, or as an intersection of Under-Approximations of Farkas polyhedra per each dependence edge—which can be used to solve feasibility problems of large polyhedra arising in affine scheduling.

- Using elementary polyhedral concepts, we present a simple and powerful framework (an approximation scheme) which can be used for designing Under-Approximation (UA) algorithms of general convex polyhedra, linearizing the UA problem. Using the above framework, we present five simple algorithms that under-approximate a general polyhedra expressed in constraint representation ($\mathcal{H}$-form) into (U)TVPI sub-polyhedra.

- We evaluate these methods by integrating them into the PLuTo polyhedral compiler. We show that for 29 out of 45 Farkas-polyhedra arising from PolyBench 2.0, the (U)TVPI-UAs proposed above are precise enough to preserve feasibility. We show that our approximations when solved with a Bellman-Ford algorithm show theoretically asymptotic, and practically considerable improvement in running time over a well established Simplex implementation; more particularly, the theoretical improvements are a reduction from $|V|^5$ to $|V|^3$, where $|V|$ is the number of statements in the input program, while

the practical improvements are more than 40 times for an unrolled matrix multiplication program with around thousand dependences. Further, we show that a preliminary integration of the above UA polyhedra into PLuTo yields code for 3 out of 16 PolyBench 2.0 benchmark cases that does not suffer significant increase in execution time.

- We show how our framework is general enough to be used for various other problems in compiler scheduling. These include problems which could be expressed within the affine transformation framework, and ones which are more powerful: either because of the type of transformations, or because of their computational complexity. In particular, we show that a small modification of the same framework that was developed for approximation of affine scheduling can be used to approximate NP-Complete loop transformation problems; this leads to a theoretical improvement in the asymptotic complexity of these problems to worst-case polynomial time.
- We show a preliminary way in which the polynomial complexities of operations on (U)TVPI polyhedra can be applied to reduce the exponential complexiy of polyhedral code generation. We introduce multiple sub-polyhedral code generation strategies which reduce the exponential complexity of the widely used algorithm in CLooG to polynomial time.

Some of the above results have been published in various workshops and conferences. Sections of Chapters 3–4 were first published in IMPACT-11 [UC11], and Sections of Chapters 5–7 in IMPACT-12 [UC12]. Some of the aforementioned sections, as well as the initial Sections of Chapter 8 are published in POPL-13 [UC13].

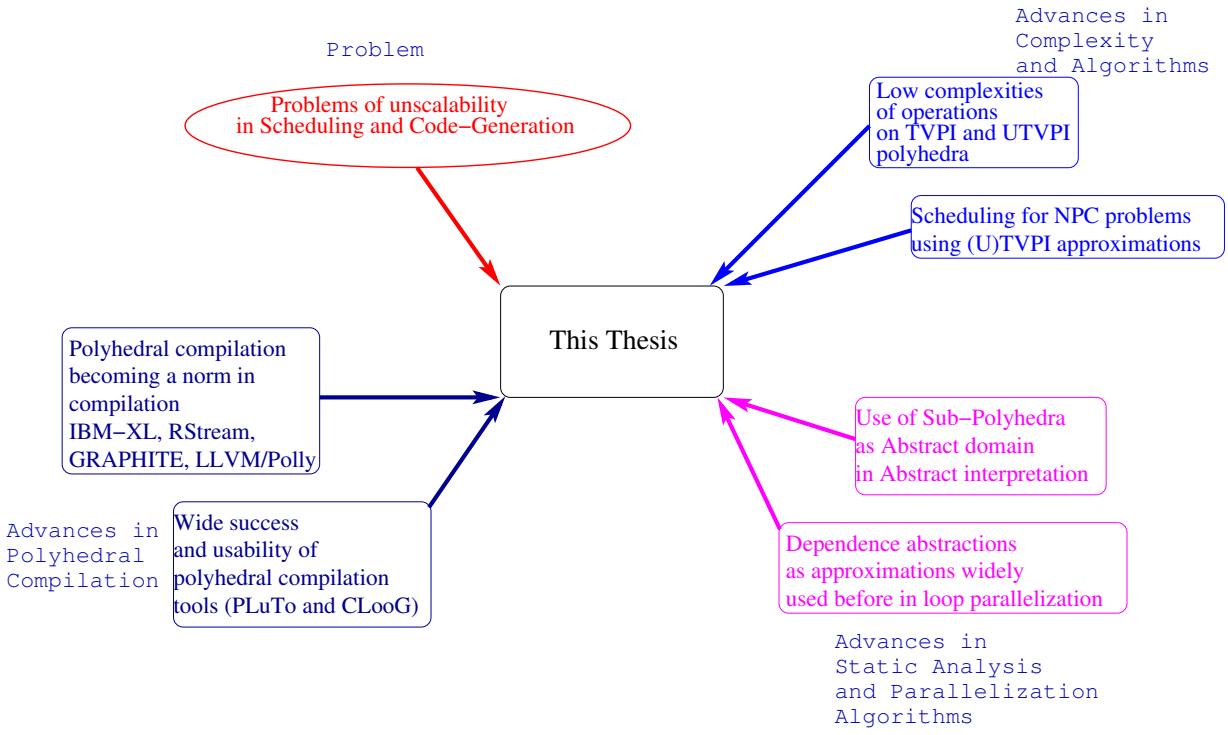


Figure 1.1: Influences on this thesis and its contributing factors

Chapter 2

Polyhedral Scheduling and the Scalability Challenge

In this chapter, we summarize polyhedral compilation, affine scheduling, and provide a motivation for solving the scalability problem in it.

2.1 Introduction

Affine scheduling [DRV00] now is a part and parcel of many compilers which aspire to compile efficiently for parallel architectures (GCC's GRAPHITE, IBM's XL, Reservoir Lab's R-Stream, LLVM's Polly). The seminal work of Feautrier [Fea92a] opened the avenue of constraint-based affine transformation methods, building on the affine form of the Farkas lemma. This approach has been refined, extended and applied in many directions. To cite only two recent achievements at the two extremes of the complexity spectrum: the tiling-centric PLuTo algorithm of Bondhugula et al. [BHRS08] extending the Forward Communication Only (FCO) principle of Griebl et al. [GFG02, GFL04] for coarse-grain parallelization, and the complete, convex characterization of Vasilache [Vas07] and decoupled exploration heuristic of Pouchet et al. [PBB+11]. Much progress has been made in the understanding of the theoretical and practical complexity of polyhedral compilation problems. Nevertheless, when considering multidimensional affine transformations, none of these are strongly polynomial in the size of the program. The lowest complexity heuristics such as PLuTo are reducible to linear programming, which is only weakly polynomial, its traditional Simplex implementation being associated with large memory requirements and having a worst-case exponential complexity.

In this chapter, we make a case for solving the scalability problem in polyhedral scheduling. First, in Section 2.2, we give a very brief and quick introduction of affine scheduling. This is followed in Section 2.3 where a case is made from different perspectives for solving the scalability problem in affine scheduling. Then in Section 2.4, we conclude this chapter giving a brief overview of the approaches that are developed in the next chapters.

2.2 Affine Scheduling: A Quick Introduction

In this section we quickly introduce some of the terms needed for this dissertation. As there are resources like the book by Darte et al. [DRV00] which do a good introduction to the highly specialized topic of polyhedral scheduling, we will only recall some essential notations and results about polyhedral compilation and in particular, affine scheduling.

Mathematical preliminaries. The mathematics needed for affine scheduling is linear programming, convexity and duality [Sch86, Zie06]. An affine transformation is an image function of the type $\mathbf{f}(\mathbf{x}) = \{\mathbf{y} \mid \mathbf{y} = A\mathbf{x} + \mathbf{b}\}$. A polyhedron is a set of rational points enclosed by affine inequalities involving the variables $\mathbf{x}$: $\mathcal{P} = \{\mathbf{x} \in \mathbb{Z}^n \mid A\mathbf{x} \leq \mathbf{b}\}$, while a parametric polyhedron $\mathcal{P}_\mathbf{y} = \{\mathbf{x} \in \mathbb{Z}^n \mid A\mathbf{x} + M\mathbf{y} \leq \mathbf{b}\}$ is defined to be a set of rational points enclosed by affine inequalities involving the variables $\mathbf{x}$ and additional symbolic constants ("parameters") $\mathbf{y}$. A polyhedron can be in either of its representations, the constraint form ($\mathcal{H}$-form) and its generator form ($\mathcal{V}$-form). The conversion from one form to the other can be performed by the Chernikova algorithm [Ver92], which is available in libraries like PolyLib [Wil93].

Input programs. The input programs for polyhedral compilation are called Static Control Parts (SCoPs). A SCoP [Fea91] in a program like C can have only `for` loops along with `if` conditionals, where the iterators of the former and conditionals of the latter define convex shapes, and are defined as affine functions of the outer loop indices and the symbolic constants ("parameters"). Also, the array variables have indices which can be described using affine functions. For these reasons, SCoPs have also been called as Affine Control Loops (ACLs) in literature. There are some other assumptions about SCoPs, like the array variables are mapped to linear memories that do not intersect with each other and the index variables are unaliased with each other, but such detection is left to the static analyzer of the compiler, and not to the polyhedral compiler itself. The more powerful the static analyzer is, along with its modules like alias and pointer analysis, the larger and more numerous are the SCoPs which are input to the polyhedral compiler. In some source-to-source compilers, SCoPs are marked by a set of pragmas. SCoPs may seem to be pieces of code of a restricted variety, but are powerful enough to encode many interesting applications, like dense matrix applications, linear algebra applications or stencil computations. In fact they are Turing complete [SQ93] and equivalent Systems of Affine Recurrence Equations (SAREs) can be derived from them.

Dependences and dependence analysis. In a SCoP, a dependence is said to exist between two memory accesses if one of the accesses is a write. Data dependence analysis is a well-formulated and well-solved problem; Banerjee's book [Ban92], and Zima and Chapman [ZC90] are good references. Computationally, it involves solving finite number of parametric integer linear programs with tools like PIP [Fea88] or by Omega [Pug91]. Polyhedral dependence analysis is the primary means of analyzing the input SCoPs and returning a dependence graph whose edges are annotated by parametrized polyhedra which indicate the regions of dependence between the two variables or statements in context.

Dependence graph. The output of a dependence analysis, and input to any polyhedral scheduling algorithm is a polyhedral dependence graph G, and is defined to be a multi-graph $G = (V, E)$, where V is the set of statements, and E is the set of dependence edges. Both kinds of entities are annotated by parametrized rational polyhedra; the nodes $v \in V$ are by rational polyhedral approximations of the iteration domains (or plainly domains) $\mathcal{P}_v$ and the edges $e \in E$ by dependence polyhedra $\mathcal{D}_e$. Each of the constraints of $\mathcal{P}_v$ and $\mathcal{D}_e$ is affine and involves (I, N) where vectors I and N are the iteration and parameter vectors, respectively.

Dependence satisfaction. The primary constraints inherent to the program that have to be satisfied are the dependence edges as formulated by Feautrier [Fea92a]. The above, usually called strong satisfaction of dependences, is simple to formulate in a LP formulation, but may seem overly restrictive because of the reduction of the number of interesting transformations. A related notion is called weak satisfaction of dependences arises from the lexicographic positivity nature of dependences and from the intuition that a dependence satisfied in a lower dimension may be enough to leave as unconstrained the dependence satisfaction of the higher dimensions. This latter notion leads to more freedom by exposing more interesting transformations, though searching for the proper schedules becomes a hard combinatorial problem because the question "which dimension of which dependence is to be satisfied at what level?" needs to be answered.

Affine transformations and Farkas lemma. An affine transformations finds an affine function in (I, N) to transform the input SCoP. These have also been called affine mappings, space-time mappings, and even plainly affine schedules. Because of the fact that every computable SARE, and hence every SCoP in a language like C, has a multi-dimensional affine schedule [Fea92b]—with the unit schedule being a trivial example of multi-dimensional schedule—these transformations are extremely powerful. They are practical as well, as most common and useful loop transformations can be expressed as affine transformations; though common simple examples are single loop nest transformations like reversal, skewing, shifting, unimodular transformations etc., the result on complete convex characterization of semantic preserving transformations by Vasilache [Vas07] proves their generality. Though these transformations happen in the input space, with the (I, N) vector as variables, the transformations are searched for in a transformed affine space obtained by application of affine form of Farkas lemma. All the affine transformations in polyhedral compilation could be said to come under the framework of affine scheduling defined by Feautrier in his classic works [Fea92a, Fea92b]. The most popular of these are the latency minimization algorithm by Feautrier [Fea92a] and the Forward Communications Only space-time mapping of Griebl et al. [GFG02, GFL04] and its extension by Bondhugula et al. [BHRS08]. This latter algorithm is the basic engine in PLuTo.

In Feautrier's algorithm [Fea92a], the input dependence constraints are converted into a per-dependence edge polyhedron $P_e(\mu, \lambda)$, with μ-variables being the Farkas multipliers that come from domain constraints and λ-variables being the Farkas multipliers that come from dependence constraints. This conversion is done by application of the affine form of the Farkas lemma [Sch86, Corollary 7.1h] given as:

Lemma 2.2 [Affine Form of Farkas's Lemma]. Let $\mathcal{D}$ be a nonempty polyhedron defined by p inequalities $\mathbf{a}_k\mathbf{x} + b_k \geq 0$, for any $k \in \{1, \ldots, p\}$. An affine form Φ is non-negative over $\mathcal{D}$ if and only if it is a non-negative affine combination of the affine forms used to define $\mathcal{D}$, meaning:

$$\Phi(\mathbf{x}) \equiv \lambda_0 + \sum_{k=1}^{p} \lambda_k(\mathbf{a}_k\mathbf{x} + b_k);\ \forall k \in [0, p]\ \lambda_k \geq 0$$

The nonnegative values λ_k are called Farkas's multipliers.

Feautrier's scheduler uses the above version of Farkas lemma so as to linearize the polyhedra in the presence of iterator vectors and parameter vectors. In the per-edge Farkas polyhedron $P_e(\mu, \lambda)$, both of the newly created λ and μ variables are called the Farkas multipliers. By putting together all the per-edge Farkas polyhedra, one obtains an overall Farkas polyhedron $P = \cap_{e \in E} P_e$, which is amenable to Linear Programming. Any rational point that satisfies P is considered a valid schedule. Hence, even Fourier-Motzkin elimination could be used to obtain a feasible point.

Two models of affine scheduling. The above application of the Farkas lemma results in all the constraints in the Feautrier's scheduler, with some additional variables to model the strong/strict satisfaction of dependences at a given dimension of the affine schedule. In PLuTo, a different but conceptually similar method results in a majority of dependence constraints of the same form as Feautrier's.

2.3 Motivation: Unscalability in Affine Scheduling

Before we attempt to solve it, we first attempt to define the problem of scalability.

2.3.1 Introduction: A definition of unscalability

Difficulties of defining the problem. The problem of scalability is difficult to define because of the following reasons:

a. Relying purely on asymptotic analysis may provide only a partial picture of the *usefulness* of the particular algorithm. The asymptotic analysis says that an algorithm with worst-case complexity $\mathcal{O}(|V|^5)$ is cheaper than one with $\mathcal{O}(|V|^6)$ worst-case complexity, while for small-size practical inputs, the reverse may be true; an algorithm with $|V|^6$ complexity may be better than one with $10^6|V|^5$ complexity. Furthermore, algorithmic and data-structure improvements that take $\mathcal{O}(|V| \log^* |V|)$ or $\mathcal{O}(|V|\alpha(|V|))$ worst-case time have remained as theoretical curiosities, while theoretically slower ones that instead take $|V| \log |V|$ have remained as the most widely used ones.

b. Algorithms which rely on constructing huge size tableaus have *limited* scope. In spite of memory becoming cheaper by drastic amounts, the amount of memory that a compilers can devote to one of its phases is limited; the problem is exacerbated for JIT compilers.

c. Use of standard tools like LP-solvers, ILP-solvers and SAT-solvers does not *solve* the problem. This is because, many times, the complexity of the tools does not match that of the algorithm. Furthermore,

many of these tools have heuristics that are tuned for some applications which may not be useful for polyhedral compilation applications.

In spite of the apparent *theoretical nature* of the above questions, they have to be answered because of the way they manifest in practical polyhedral compilation.

Beginning to solve the problem. Our analysis begins with opening the boxes of the transformations *along* with their tools and measuring the precise complexities as parametrized by input sizes: for example, the number of statements $|V|$ and dependence edges $|E|$ in the input SCoP, the maximum depth d of the input loop-nest etc. This is despite the fact that they rely on well written and well used tools like PIP, PolyLib, etc.

View of this dissertation. The view taken by this dissertation is to prefer asymptotic time complexity analysis as the *primary* means of measuring the scalability or unscalability of an algorithm, and to evaluate different competing algorithms. By this measure, any solution which uses an NP-Complete formulation, would clearly be unscalable. Even among polynomial algorithms, we prefer algorithms with strongly polynomial time complexity, rather than ones which have weakly polynomial time complexity.

Furthermore, will also rely on memory complexity analysis as the *secondary* means of measuring scalability. Algorithms which rely on construction of huge-size tableau's—like simplex does—are likely to be heavy in compilation frameworks, like for JIT compilation.

The above leaves linear and quadratic time complexities as scalable complexities, which should be the eventual and perhaps ideal goal. Even after after all these, we will measure how the algorithms performs for practical inputs.

Overview of this section. In this section, we show an example of unscalability of current affine scheduling methods in an empirical sense using PLuTo, followed by a theoretical evaluation. First in Section 2.3.2, we do an experimental evaluation of affine scheduling. Then in Section 2.3.3, we see the practical ways in which the unscalability problem can arise in practical inputs. Then, in Section 2.3.4, we see some empirical characteristics of the above LP programs. Finally, in Section 2.3.5, making some reasonable estimates, we estimate the complexity of affine scheduling.

2.3.2 Unscalability in affine scheduling: an experimental evaluation

Input programs. We began with two typical kernels `matmul` and `seidel` of PolyBench [PB$^+$] as shown in Figure 2.1, and artificially unrolled them by a variable number of times so as to increase the number of dependences in the loop nests. As shown in Figure 2.2, we have also enclosed the unrolled loops in two or three "time loops", mimicking the behavior of a scientific computing kernel. The above transformations induce thousands of dependences in the input programs—up to 4000 dependences for `matmul`, and up to 7000 dependences for `seidel`—and thereby make the scheduler solve large size linear programs. In a sense, the two programs that have been selected form the extreme end of affine transformations; `matmul` has only affine dependences, while `seidel` has only uniform dependences. This difference is crucial because

```
for (i=0; i<M; i++)
  for (j=0; j<N; j++)
    for (k=0; k<K; k++)
       C[i][j] = beta*C[i][j] + alpha*A[i][k] * B[k][j];
```

`matmul` kernel

```
for (t=0; t<=T-1; t++){
  for (i=1; i<=N-2; i++){
    for (j=1; j<=N-2; j++){
      a[i][j] = (a[i-1][j-1] + a[i-1][j] + a[i-1][j+1]
                             + a[i][j-1] + a[i][j] + a[i][j+1]
                             + a[i+1][j-1] + a[i+1][j] + a[i+1][j+1])/9.0;
    }
  }
}
```

`seidel` (9 point) kernel

Figure 2.1: Sample Loop Kernel Examples from Polyhedral Compilation (from PLuTo/PolyBench)

```
for (t1=0; t1<N1; t1++){
  for (t2=0; t2<N2; t2++){
    for (t3=0; t3<N3; t3++){
      matmul1
      matmul2
      ...
      ...
      matmulN
    }
  }
}
```

ITR-MATMUL

```
for (t1=0; t1<=N1; t1++){
  for (t2=0; t2<=N2; t2++){
    for (t3=0; t3<=N3; t3++){
      seidel1
      seidel2
      ...
      ...
      seidelN
    }
  }
}
```

ITR-SEIDEL

Figure 2.2: Iterated/Unrolled Examples to induce Unscalability for Affine Scheduling. The variables of the programs (`matmul` or `seidel`) have been changed so that there is a dependence between individual chunks of programs and a non-trivial transformation occurs.

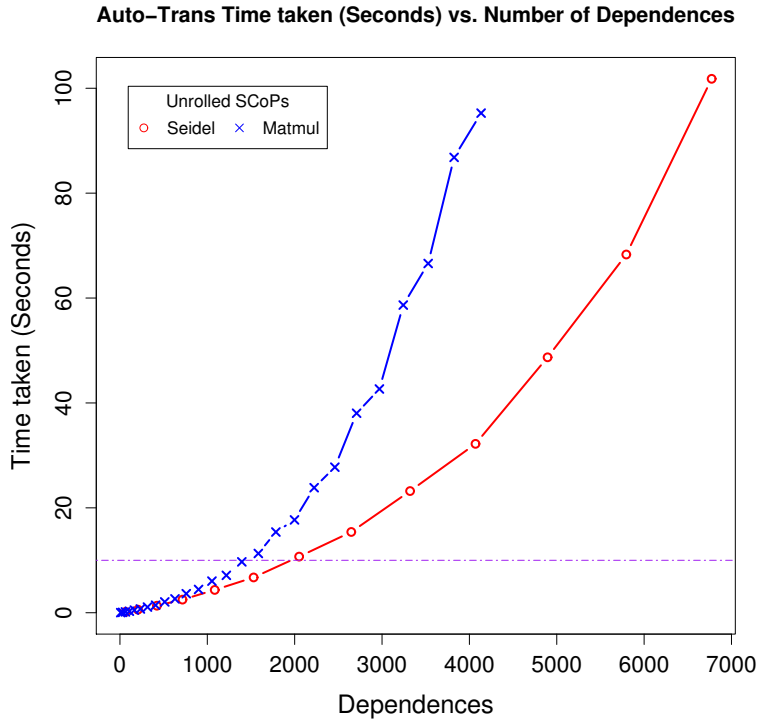


Figure 2.3: Unscalability for Large Loop Programs

the programs with affine dependences may induce Farkas systems which are more complex than the one with uniform dependences, thereby making the simplex algorithm in the LP solver of PIP [Fea88] converge with different rates, namely higher rate for the ones with affine dependences.

Compilation times. For measuring the execution time of the scheduler, we focussed on `auto_transform` function of PLuTo, which encodes its top-level automatic transformation search engine. The above function is preceeded by a call to the `clan_scop_extract` function which does dependence analysis of the input pieces of code, and succeeded by the `pluto_codegen` function which does the code-generation by making a call to the popular code-generator CLooG. More particularly, the above function `auto_transform` iteratively constructs the Farkas polyhedra according to its formulation of the FCO algorithm, makes a call to PIP's linear programming solver to find a `lexmin` of the Farkas systems, and interprets the resultant solution to obtain permutable hyperplanes.

The execution times of `auto_transform` are shown in Figure 2.3, with `matmul` in blue (crosses) and `seidel` in red (circles). We also measured independently the time taken in execution of rest of the modules of PLuTo—in particular, the dependence analysis and code generation phases—and found that they took significantly less time than the above measured times.

Rise of the curves. Using linear regression tools from *The R Project for Statistical Computing* [R], we checked that the execution times of `auto_transform` increases in a quintic complexity: $|V|^5$, where $|V|$ is

the number of statements in the system. This is the high complexity that polyhedral compilation has to pay for using LP based techniques.

Unscalability in different modules of affine scheduling. Within affine scheduling itself, there are various algorithms like Farkas multipliers elimination by gaussian elimination and a limited preprocessing with Fourier-Motzkin elimination. Even though these could be costly in themselves, these are not major sources of unscalability. This is primarily because they can be done on a per-dependence basis and the per-dependence Farkas polyhedra are considerably smaller to afford high-complexity algorithms.

Unscalability in different modules of polyhedral compiler. From the curves in Figure 2.3, it is clear that there is a strong case of solving the scalability problem for affine scheduling in polyhedral compilers using affine scheduling. From these experiments, we have also observed that dependence analysis and code generation are not major sources of unscalability when compared to affine scheduling.

The former is not much surprising since the dependence polyhedra, being mainly per-dependence entities, are considerably smaller than the overall Farkas polyhedra. This will be empirically shown in this dissertation in Section 7.2 on page 90. Furthermore, dependence analysis is a relatively one-time process whose cost does not vary much asymptotically with the input program size, with its cost being limited to $\mathcal{O}(|E|)$ number of linear programs whose size is bounded.

The latter is a little surprising because polyhedral code generation has its own scalability problems definitely arising from use of exponential time algorithms and polyhedral libraries with exponential complexity. But, it can only be reconciled that for these particular examples, the scalability problem in code generation is not manifested, and there exist other examples for which there is a perceptible unscalability problem, for example like programs which do multi-level tiling. The aspect of scalability in code-generation is studied further in this dissertation in Chapter 9 on page 127.

2.3.3 Practical causes of unscalability

It may seem that the above unrolling based method is an artificial way to induce unscalability, with inlining being a better candidate for the same in real world benchmarks. Also, in current benchmarks for loop nest optimization—like the PolyBench 2.0 [PB$^+$]—the range of number of dependences is in tens, and it can arguably be said that presently there exists no scalability problem like in the above artificial examples.

Unrolling vs. inlining. While unrolling is much simpler to simulate, limitations of the research prototype infrastructures we have used—unlike more powerful ones like PIPS [PIP]—do not provide a platform to study the asymptotic time complexity associated with code size increases associated with inlining. Hence, the above examples could only be taken as representatives of the unscalability problem. But it should also be remembered that when discussing the solution time with respect to the input code size increase, the number of constraints in the overall LP problem is linear in the number of dependences in the input code. So, a method like the above which gives a representative sample to increase the size of the LP program is not a limitation.

Longer source codes. Longer source codes in the input programs, which lead to longer SCoPs and hence larger LP programs that the affine scheduler has to solve, could arise not just from inlining or unrolling. They could arise from difficult and more expressive source languages. The current input language of PLuTo is a variation of C, which does not have powerful features like templates of C++. Polyhedral compilers will soon face such large problems when the Front End of polyhedral compilers is improved. And, aggressive interprocedural optimization, domain-specific program generation along with the iteraction of C++ program features like templates with inlining could exponentially increase the size of SCoPs. Also, sources in medium/low-level languages and preceded by more powerful static analysis like alias/pointer analysis as a part of compiler framework like in GCC-GRAPHITE [TCE+10] make these issues a very possible reality.

Expanding the definition of SCoPs. Longer SCoPs could also arise from the removal of restrictions on the definition of SCoPs in the input program and thereby increase the range of programs that could come under the category of affine analysis. In this trait are the works of Griebl et al. [Gri96, GL94] on `while` loop parallelization, and analyses like Maximal Static Expansion (MSE) by Barthou et al. [BCC00] who provide an expansion framework for general (possibly non-affine) data structures. In the same category is the work of Benabderrahmane et al. [BPCB10] who add more categories of programs for polyhedral analysis, by removing the affinity limitation in the input SCoPs by conservatively approximating the non-affine domains, and by converting control dependences into data dependences.

JIT compilation. In addition, there could be a restriction of the time limit in just-in-time compilers that would further exacerbate the scalability problem. Just-In-Time (JIT) polyhedral compiler of LLVM called Polly [GGL12] is such an example which has a scheduler, which is very similar to the one in PLuTo.

Existing limitations on unscalability. Also, industrial compilers like IBM's XL, and Reservoir's R-Stream compiler have schedulers very similar to PLuTo, and the IBM-XL compiler is known to limit the compilation time of kernels of using its polyhedral compilation module to 10 seconds. One can see the particular time limit being crossed by PLuTo in the above plot for both the examples. For an initial range estimate, our experiments indicate that the 10 second compilation limit could be crossed for around 1000 dependences which is possible with the above mentioned reasons.

In the following, we aim for lower complexity feasibility and optimization algorithms, with worst-case strongly polynomial bounds, and closer to n^2 time complexity for large-scale and/or just-in-time compilation applications.

2.3.4 LP program characteristics

So as to understand the empirical complexity a little better, in Figure 2.4, we plot the characteristics of the ITR-MATMUL, using `the pairs` function of R [R].

We denote $|V|$ to be the number of statements in the input dependence graph, $|E|$ to be the number of dependences, m to be the number of constraints in the LP overall formulation, and τ to be the time

taken for the LP solver—here *rational* Simplex of PIP for a *feasible* point—to complete.

It can be seen from the plots in Figure 2.4 that:

- The number of dependences is quadratic in the number of statements: $|E| = \mathcal{O}(|V|^2)$;
- the number of constraints is linear in the number of dependences: $m = \mathcal{O}(|E|)$;
- the time taken is almost cubic in the number of dependences: $\tau = \mathcal{O}(|E|^2|V|) \approx o(|E|^3)$;
- the time taken is quintic in the number of statements: $\tau = \mathcal{O}(|V|^5)$.

It should be noted that these asymptotic estimates are representative of this particular input program, here unrolled `matmul`, and not a general and exact characterization of the simplex algorithm of PIP. But, it is a safe assumption that they indicate a dependence of the running time of simplex on large powers, like quintic powers, of the input programs which leads to scalability issues.

In the next section, we make some reasonable assumptions and attempt to obtain a theoretical estimate of the sizes of linear programs to be solved, and time taken to solve them.

2.3.5 A theoretical estimate of complexity

As explained earlier, in Feautrier's algorithm [Fea92a], the overall constraint system P, which is assembled as $P = \cap_{e \in E} P_e(\mu, \lambda)$ is *not* a parametric polyhedron: classical non-parametric algorithms can be used. This is an encouraging observation for the effectiveness of any scalability based algorithmic approaches. Feautrier suggests preprocessing P with an initial Gaussian elimination step to remove some Farkas multipliers. This is possible because the Farkas multipliers occur in equations and the elimination can be done on a per-dependence basis. But, this step will *not* remove all the Farkas multipliers as P_e is too under-constrained at least with respect to the λ-variables. This is complicated by the fact that the systems P_e and $P_{e'}$ on different edges e and e' are not independent systems with respect to the μ-variables. The assumption in this thesis is that P has too many variables and constraints, and hence solving it will be asymptotically unscalable.

An estimate. Consider a dependence multigraph (V, E). Let $|V|$ be the number of statements in the program and $|E|$ be the number of edges. We can assume that all statements have the same dimension and that the maximum dimension of any statement in the program is d, counting both iteration and parameter dimensions.

Let n_k and m_k be the respective number of variables and constraints of the domain polyhedron for statement k. The polyhedron of a dependence $S_i \to S_j$ between two statements i and j has dimension $2d$.[1] Let n_e and m_e be the respective number of variables and constraints of the polyhedron of dependence edge e.

Applying Feautrier's one-dimensional scheduling, we have two kinds of Farkas variables or Farkas multipliers:

[1] This is an upper bound: $2d$ minus the number of parameters to be precise.

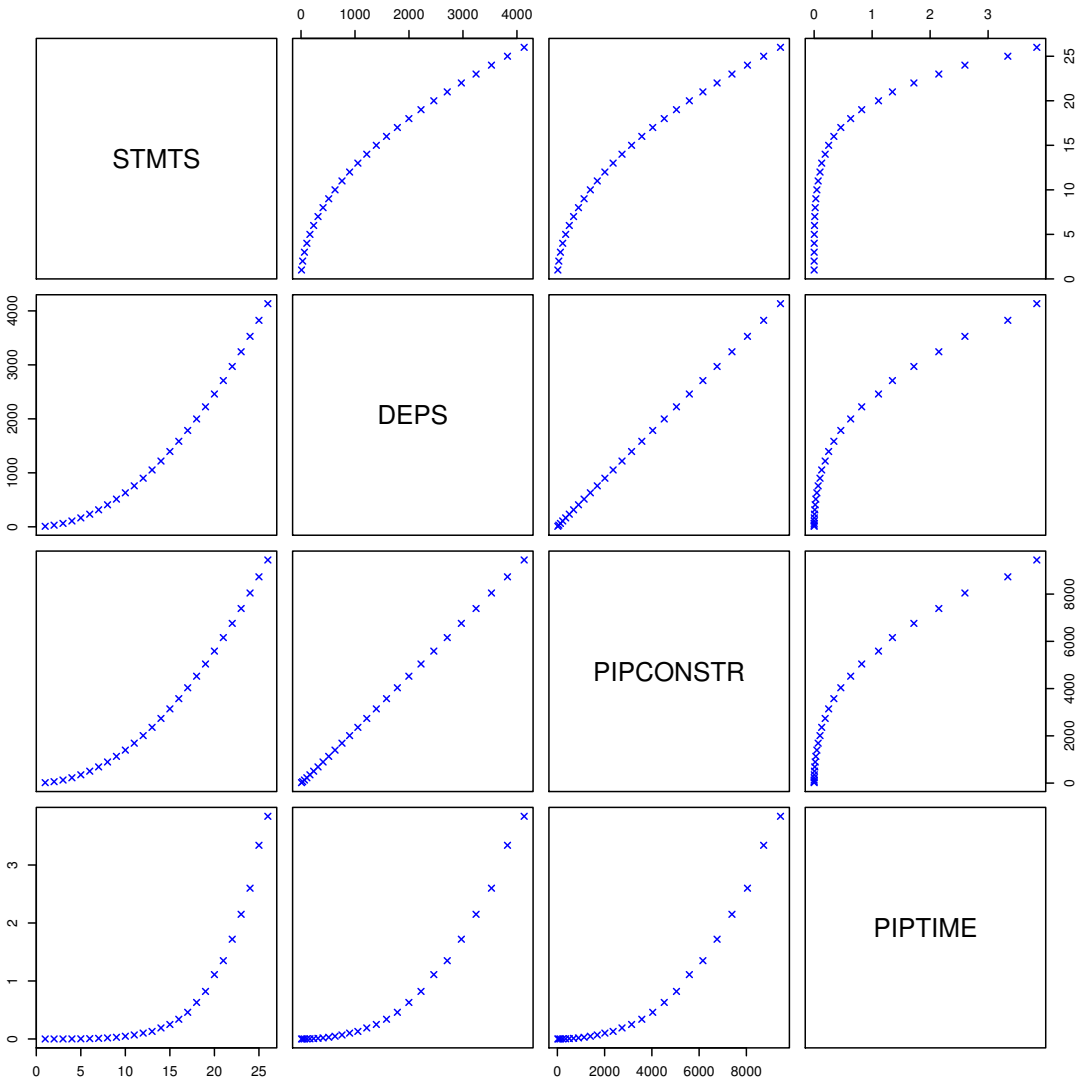


Figure 2.4: Characteristics of LP programs of ITR-MATMUL for Unscalability: #Statements (`STMTS`) vs. #Dependences (`DEPS`) vs. #Constraints (`CONSTR`) vs. Time Taken per PIP call (`PIPTIME`)

- one λ variable per dependence and per dependence constraint. So, the total number of λ variables is $2d\sum_{e\in E} m_e = 2d|E|\widehat{m_E}$, where $\widehat{m_E}$ is the average number of constraints per dependence polyhedron;

- one μ variable per statement and per domain constraint. So the total number of μ variables is $d\sum_{k\in V} m_k = d|V|\widehat{m_V}$, where $\widehat{m_V}$ is the average number of constraints per domain polyhedron.

Also, the total number of constraints in the complete system is $d|E|$.

The above estimates lead to a linear programming problem with $d|E|$ constraints and around $2d|E|\widehat{m_E}+d|V|\widehat{m_V}$ variables. As suggested by Feautrier [Fea92a], we can assume that a gaussian elimination step runs as a preprocessing step to remove λ-variables which occur only in equalities, and is able to remove a significant number of them, making the number of variables as $d|V|\widehat{m_V}$. This elimination step would be inexpensive as it can run on a per-dependence edge basis. We can also validly assume that $\widehat{m_V}$ is a very small bounded value, making the number of variables in the overall system to be $d|V|$.

Assuming a usual complexity of linear programming solved with a normal simplex algorithm (the best algorithm for state-of-the-art Farkas-based methods)[2], this brings the total complexity of the scheduling algorithm to order of:

$$(d|V| + d|E|) \cdot d|E| \cdot d|V| \approx d^3|E|^2|V| = \mathcal{O}(|E|^2|V|)$$

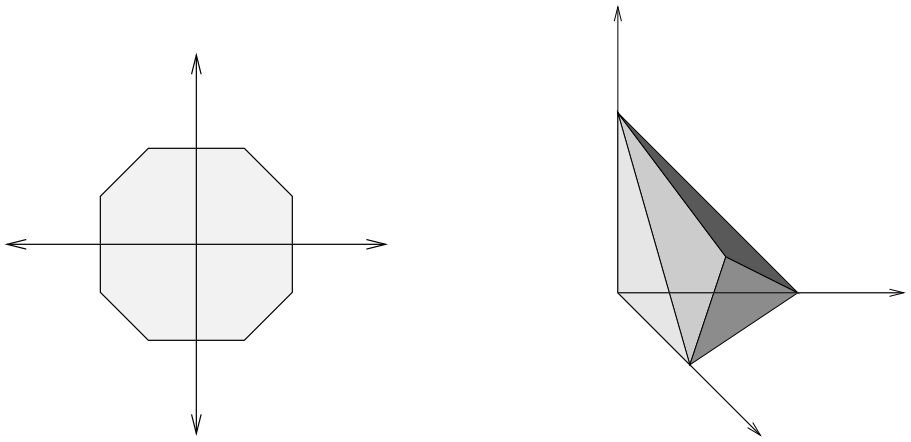
Assuming that the dependence graph is a complete graph, we have $|E| \approx |V|^2$ which makes the above complexity to as $\mathcal{O}(|V|^5)$. This complexity could be worse when the dependence graph is a more dense multi-graph, or in presence of piecewise-affine dependence relations (e.g., in some cases after array data-flow analysis).

The above complexities are clearly not scalable. Analyzing the PLuTo algorithm leads to a similar computation and the same non-scalability result. Practical evidence shows that it scales well on all benchmarks considered so far. But no benchmark with thousands of loops and statements has been processed either, which will no doubt happen with three-address code representations [TCE+10] and extensions to irregular, data-dependent control flow [BPCB10]. Considering Feautrier multi-dimensional scheduling [Fea92b] only makes things worse, as it could involve solving a mixed-integer linear programming problem.

2.4 Conclusion and Brief Overview of Our Approaches

Conclusion. In this chapter, after a quick summary of polyhedral scheduling, we showed using both theoretical and empirical ways that existing methods result in an LP problem of size $m \times n \approx (d \cdot |E|) \times (d \cdot |V|)$, where d is the mean depth of the loop nests. Assuming a usual simplex method—whose complexity will be detailed further in Section 3.3.1 on page 28—for solving systems with bounded d leads to a close to $\mathcal{O}(d^3|E|^2|V|) \approx \mathcal{O}(d^3|V|^5) \approx \mathcal{O}(|V|^5)$ asymptotic complexity (not counting the bit-size complexity, and assuming $|E| = \mathcal{O}(|V|^2)$), leading to unscalability problems. The estimates focus on Feautrier's methods but are general enough to be extended to any affine scheduling methods like the FCO implementation in PLuTo.

[2]This algorithmic analysis will be further developed in Section 3.3.1 on page 28.



Unit-Two-Variables Per-Inequality (UTVPI)
$x_i \pm x_j \leq c;\ c \in \mathbb{Q}$
[UC11, UC12, UC13]

Two-Variables Per-Inequality (TVPI)
$ax_i + bx_j \leq c;\ a, b, c \in \mathbb{Q}$
[UC11, UC12, UC13]

Figure 2.5: Sub-Polyhedra used in this dissertation in polyhedral compilation.

Approximations of polyhedra. After pointing out the problem of unscalability of current methods that use convex polyhedra, the main contributions of this thesis is to alleviate this problem by the use of sub-polyhedra in polyhedral compilation. In particular, we focus on TVPI and UTVPI sub-polyhedral approximations shown in Figure 2.5. This notion is explored in the next chapter. Also, we concentrate on (U)TVPI under-approximations of general polyhedra. The above polyhedra, along with a sub-variety DBM polyhedra are shown in Table 2.1. The above hierarchy could also be considered as different levels of optimization offered by the affine scheduler offering a precision vs. cost choice to the compiler writer.

Sub-polyhedra	Approximation (nature of constraint) $(a, b, c \in \mathbb{Q})$
Intervals	$a \leq x_i \leq b$
DBM (or monotone UTVPI or m-UTVPI)	$x_i - x_j \leq c; x_i, x_j \geq 0$
Octagons (UTVPI)	$\pm x_i \pm x_j \leq c$
TVPI	$ax_i + bx_j \leq c$
Convex Polyhedra	$\sum a_i x_i \leq c$

Table 2.1: Sub-Polyhedra and their Variations used in this Dissertation

Convex polyhedra and approximations of polyhedra, like the restricted ones in Table 2.1 could be defined as elements of lattices, or Complete Partial Orders, with appropriate operators like meet and join defined on them. The resultant semantic structures are used as *abstract domains* in static analysis beginning from the work of Cousot and Cousot [CC77]. For most of this dissertation, however, we use the above restrictions in affine-scheduling. Algorithmically, the problem reduces to constructing a *single* large polyhedron which encodes the feasible solution points; and searching for a valid solution reduces to solving a linear programming problem. In the latter part of the thesis, however when applying the above approximations to the problem of code-generation, we will be concerned with a using *sets* of the above

varieties with some operators—like image, intersection and difference—defined over them. Furthermore, in this thesis, when we use the term sub-polyhedra without any qualification, it would refer to the restricted ones in Table 2.1

Hierarchies of precison and cost. The above varieties of sub-polyhedra and their inclusiveness also give rise to a hierarchy of classes of lattices–with the above overloading of notation, approximations of polyhedra—and to different tradeoffs of precision and time complexities:

$$\xrightarrow[\text{Intervals} \subset \text{Octagons (UTVPI)} \subset \text{TVPI} \subset \text{Conv.Poly}]{\text{Precision}}$$
$$\xrightarrow[\text{Complexity}]{}$$

Sub-polyhedral affine scheduling. The main ideas that will be developed in the next couple of chapters is the concept of affine scheduling using (U)TVPI sub-polyhedra. This is also tied to the concept of Over/Under-Approximations of polyhedra and approximates the input from within the affine scheduling model, and is hence unique when compared to previous approaches.

Chapter 3

Sub-Polyhedra: TVPI and UTVPI

In this chapter, we briefly cover some basics of Two-Variables-Per-Inequality (TVPI) and (Unit-)Two-Variables-Per-Inequality (UTVPI) approximations of polyhedra.

$$\overrightarrow{\text{Interval} \subset \text{UTVPI} \subset \text{TVPI} \subset \text{Poly}}$$

Figure 3.1: A Hierarchy of Sub Polyhedral Classes

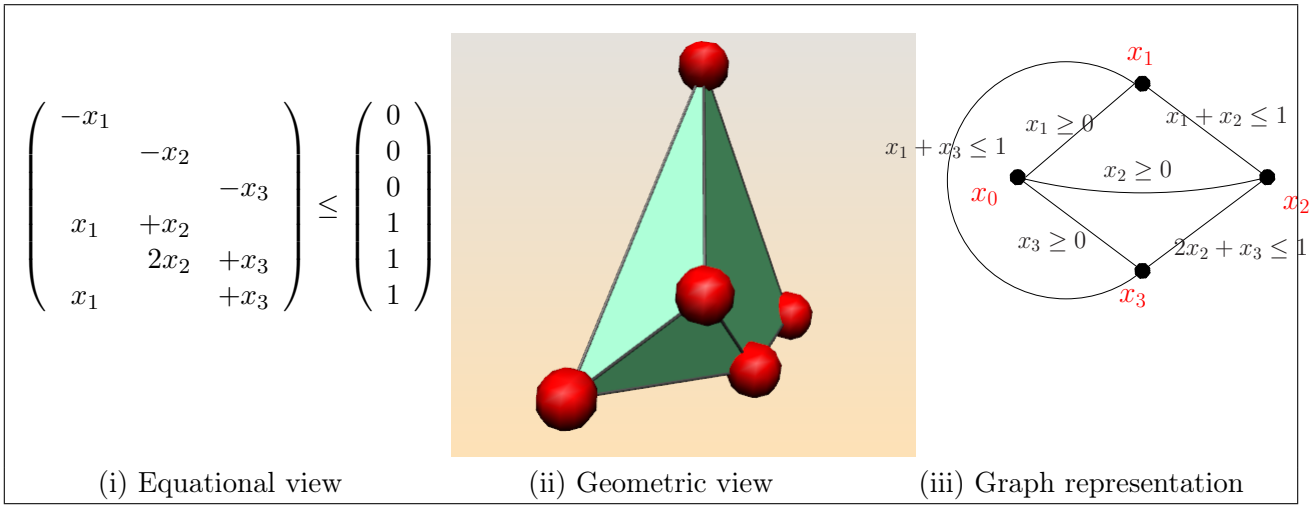
3.1 Introduction

In Section 3.2, we cover some complexities of (U)TVPI Sub-Polyhedra. In Section 3.3, we cover some basics of Linear programming and Simplex complexity. This is followed by Section 3.4, where we study how (U)TVPI Sub-polyhedra fare when compared with general convex polyhedra in a complexity theoretic sense.

Prior to our work in loop parallelization, the static analysis field in recent times has seen a widespread usage of sub-polyhedra. In Section 3.5, we do a brief summary of their use, and then differentiate on how their requirements are different from our proposed use of sub-polyhedra. A similar summary of the use of sub-polyhedral abstractions by the loop parallelization community is done in the next chapter.

This will be followed with some conclusions and related work in Section 3.7.

Notation. For a polyhedron described in constraint form, let m be the number of inequalities, n be the number of variables and B the upper bound on the absolute value of the coefficients describing the system.



(i) Equational view (ii) Geometric view (iii) Graph representation

Figure 3.2: Different representations of TVPI Sub-polyhedra

3.2 (U)TVPI Sub-Polyhedra

In this section, first we study the complexities of certain operations on TVPI and UTVPI sub-polyhedra, along with monotone variants of both of them. This is followed by study of the monotonizing transformation, which is used to construct graph theoretic formulations from a general TVPI or UTVPI sub-polyhedra.

3.2.1 TVPI Sub-Polyhedra

In Two Variables Per Inequality (TVPI) polyhedra, each constraint is of the form: $ax_i + bx_j \leq c$; $a, b, c \in \mathbb{R}$. TVPI are closed under projection, and hence many algorithms on geometric operations that are developed for planar polyhedra (polygons) are directly applicable to general TVPI, giving rise to simple algorithms with low complexity. Furthermore, the dual of a TVPI linear program is a generalized min-cost flow problem, which can be solved using graph algorithms. So, the linear programming community has been interested in TVPI polyhedra because of the strongly polynomial time algorithms for the emptiness of TVPI constraint systems.

Application of graph theory to linear programming using TVPI systems was pioneered by Shostak [Sho81], who showed that TVPI systems can be represented using undirected multi-graphs: one vertex for each variable and one edge for each constraint. Since each inequality contains two non-zero variables, an inequality involving variables x_i and x_j is represented as an edge between the two corresponding vertices. Upper ($x_i \leq c$) or lower ($x_i \geq c$) bound type of inequalities can be represented as edges between x_i and x_0, where x_0 is an additional vertex to represent source or sink vertex.

Example 3.2.1 [TVPI representations]. In Figure 3.2, we show different representations of the TVPI constraint system

$$\{x_1 \geq 0;\ x_2 \geq 0;\ x_3 \geq 0;\ x_1 + x_2 \leq 1;\ 2x_2 + x_3 \leq 1;\ x_1 + x_3 \leq 1;\}$$

the equational view, the geometric view[3] and the graph representation.

Aspvall and Shiloach [AS80] showed the polynomiality of the feasibility problem of TVPI-LP formulations by introducing a unique strongly polynomial time procedure ("*rumor in Grapevine algorithm*") that can be used to decide the range of a particular variable x_i with respect to a given constant ξ. This latter procedure is a Bellman-Ford style propagation of values assigned to variables through inequalities in the system, and is the heart of all subsequent algorithms in the TVPI literature.

Briefly, in the above algorithm, Aspvall and Shiloach make the assignment $x_i = \xi$. As the inequalities in which x_i participates are encoded as edges in the graph, the assignments to the neighbours of x_i in the graphical representation changes and they find corresponding assignments. These assignments percolate in the graph exchanging mutual assignments between neighbours, until a contradiction is found or all inequalities are satisfied. Shostak had earlier suggested an algorithm which enumerates simple cycles in the graph for the same purpose and hence takes exponential time for such percolation, but Aspvall and Shiloach show that the percolation can proceed in a controlled manner just similar to Bellman-Ford algorithm. The complexity of Aspvall and Shiloach [AS80]'s percolation algorithm is the following:

Lemma 3.2.2 [Deciding a range of a variable in a TVPI system]. In a non-trivial TVPI system, given a variable x_i and a numerical value (constant) ξ, it can be decided in $\mathcal{O}(mn)$ worst-case time whether (i) $\xi < x_i^{\min}$, (i) $x_i^{\min} \leq \xi \leq x_i^{\max}$, or (iii) $x_i^{\max} < \xi$.

The above complexity measure should be contrasted with the corresponding complexity measure of the analogous problem on general polyhedra which involves formulating an LP optimization problem and hence is as hard as LP-optimization.

The following result by Wayne [Way99] is the best to date for the TVPI optimization problem:

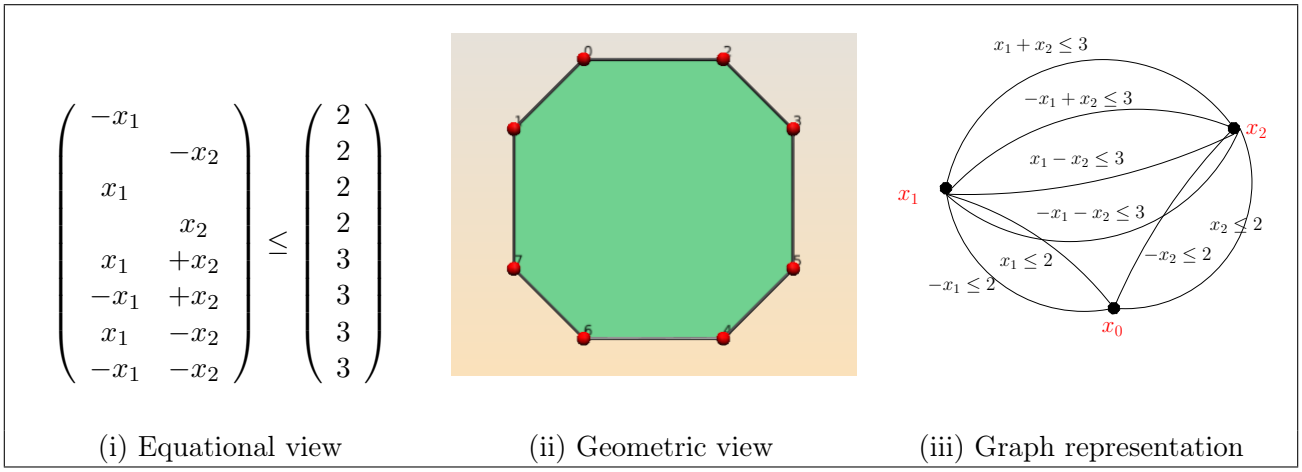
Lemma 3.2.3 [LP optimization on TVPI]. Linear programming optimization on TVPI systems can be solved in $\mathcal{O}(m^3n^2 \log m \log B)$ worst-case time.

It well known that for general polyhedra, the optimization and the feasibility problems have the same weakly-polynomial time hardness. But it is interesting to note that up to now, they have different complexities on TVPI systems. The feasibility problem on TVPI systems has lower complexity that the above weakly polynomial time result by Wayne [Way99] on the optimization problem. If we denote A as the coefficient matrix of the system, $B = \text{bitsize}(A)$, the upper-bound on the absolute value of the coefficients describing the system, and the dependence of the above measure on B makes it weakly polynomial time.

The above complexity measure by Wayne is a general result with no restrictions on the type of objective function. There exist improved complexity results for TVPI systems when the objective-function is of a restricted variety; for example, systems with bounded number of non-zero elements in the objective function by Hochbaum and Naor [HN94] and systems with all coefficients of the objective function having the same sign by Hochbaum et al. [HMNT93].

The first strongly polynomial time algorithm for TVPI feasibility was given by Megiddo [Meg83]. Network flow based ("combinatorial") strongly polynomial time algorithms for the feasibility problem

[3]The 3-dimensional polyhedra in this chapter are drawn using polymake [GJ00] software.



(i) Equational view (ii) Geometric view (iii) Graph representation

Figure 3.3: Different representations of UTVPI Sub-Polyhedra

were given by Cohen and Megiddo [CM94]. Hochbaum and Naor [HN94] showed that feasibility of TVPI polyhedra can be determined in strongly polynomial time:

Lemma 3.2.4 [Feasibility on TVPI]. Feasibility of TVPI systems can be solved in $\mathcal{O}(mn^2 \log m)$ worst-case time.

The above algorithm by Hochbaum and Naor is surprisingly simple and we give a quick summary. The algorithm begins with the observation that the set of inequalities between two vertices x_i and x_j induces a piecewise linear function on the $x_i - x_j$ plane. The vertices of this polygon ("polyhedral-vertices") can be found in polynomial time using any standard geometric algorithm [Gra72, SK04, dBKS00]. Hochbaum and Noar call these polyhedral-vertices as *breakpoints* and prove that the number of breakpoints on a particular edge is $\mathcal{O}(m)$. Next, they observe that the above mentioned decision procedure of Aspvall and Shiloach can be used to run a binary search in the sorted list of all breakpoints corresponding to a particular polyhedral-vertex. In polynomial time of $\mathcal{O}(mn \log m)$, this binary search returns either with an assignment to x_i, or with a contradiction of assignments. In the former case, the vertex x_i is collapsed with vertex x_0 and the procedure continues with the next vertex x_{i+1}, while in the latter case, the overall procedure stops with the result that the system is infeasible. The overall cost of the algorithm is thus $\mathcal{O}(mn^2 \log m)$.

It can be seen that the above result of Hochbaum and Naor can as well be used to derive strongly polynomial time bounds for Fourier-Motzkin elimination, and for projection of variables from TVPI systems. For general convex polyhedra these are known to be more costly.

TVPI systems have successfully been used for various problems in the abstract interpretation and verification by Simon, King and Howe [SKH10].

3.2.2 UTVPI Sub-Polyhedra (octagons)

Unit Two Variables Per Inequality (UTVPI) have constraints of the form $ax_i + bx_j \leq c; a, b \in \{0, \pm 1\}; c \in \mathbb{Q}$. They are also called Octagons because in 2-dimensions, their geometric shape is octagonal. Since UTVPI $\subset$ TVPI, the above complexity bounds of optimization and feasibility on TVPI polyhedra apply

to UTVPI polyhedra as well. But, as the dual of the LP formulation of shortest-paths problem has just difference constraints [Pra77] with the form $x_i - x_j \leq c$, general UTVPI systems can be solved with same quadratic complexity as Bellman-Ford [AMO93, CSRL01], giving the following lemma:

Lemma 3.2.5 [Feasibility on UTVPI]. Feasibility of UTVPI systems can be solved in $\mathcal{O}(mn)$ worst-case time and in $\mathcal{O}(m+n)$ space.

The above decision algorithm can also trivially return a *feasibility certificate* of the UTVPI polyhedron.

Example 3.2.6 [UTVPI representations]. In Figure 3.3, we show different representations of the UTVPI system

$$\{-2 \leq x_1, x_2 \leq 2;\ x_1 + x_2 \leq 3;\ +x_1 - x_2 \leq 3;\ -x_1 + x_2 \leq 3;\ -x_1 - x_2 \leq 3;\}$$

the equational view, the geometric view and the graph representations.

UTVPI polyhedra have successfully been used for various problems in abstract interpretation and program verification [Min06, BHZ09, Min04]. Also, they have well supported libraries like Apron [JM09], and in the Parma Polyhedra Library (PPL) [BHZ08, PPL]. These are used with success in static-analysis tools like the Astrée analyzer [BCC$^+$03, CCF$^+$09].

Monotone-UTVPI Sub-polyhedra (Diffference constraints). It can be noticed that difference constraints of the form $x_i - x_j \leq c$ are a specialization of UTVPI polyhedra. As mentioned, the polynomiality of solving difference constraints using shortest paths algorithms of graph theory originates from the theory of Pratt [Pra77]. They have been called *monotone constraints* [Hoc04, HN94, CM94] because of the monotone nature of the sign of the coefficients, and *constraint graph* [CSRL01, AMO93] because they can be represented using weighed directed multi-graphs: one vertex per variable and one edge per constraint with the directed edge for constraint $x_i - x_j \leq c$ being $x_j \xrightarrow{c} x_i$.

Their use as abstract domains is mainly by Bagnara et al. [Bag97], Miné et al. [Min01a] and Shaham et al. [SKS00] who call them Difference Bound Matrices (DBM), *potential constraints*, and *zones.*

It can noted that monotone-UTVPI constraints are quite similar to UTVPI constraints in that both of these allow only two non-zero variables with only unit coefficients. But, UTVPIs can even encode constraints of the form $x_i + x_j \leq c$ and $-x_i - x_j \leq c$ as well, and even the upper-bound and lower-bound constraints $-x_i \leq c$ and $x_i \leq c$. Also, it can be noted that for both TVPI as well as UTVPI constraints, the graphs in their graphical representation are undirected and the constraints are just annotations on the (undirected) edges. Both these restrictions can be removed for either of (U)TVPI systems using a *monotonizing* transformation.

3.2.3 Monotonizing transformation, Integer polyhedra and Feasibility

The monotonizing transformation takes in a normal (non-monotone) (U)TVPI constraint system, and creates an equivalent monotone (U)TVPI constraint system having $2n$ variables: each variable x_i in the

original system would be represented by two variables x_i^+ and x_i^- in the new system with $x_i = x_i^+ = -x_i^-$. The constraints in the new system are defined as in in Table 3.1.

TVPI monotonizing transformation. Given a TVPI system, the monotonizing transformation converts TVPI constraints to m-TVPI constraints (as in Table 3.1)

	TVPI constraint $(a, b \in \mathbb{Q}^+)$		Monotone representation		
(++ case)	$ax_i + bx_j \leq c$	$i \neq j$	$ax_i^+ - bx_j^- \leq c$	$\wedge$	$-ax_i^- + bx_j^+ \leq c$
(−− case)	$-ax_i - bx_j \leq c$	$i \neq j$	$-ax_i^+ + bx_j^- \leq c$	$\wedge$	$ax_i^- - bx_j^+ \leq c$
(+− case)	$ax_i - bx_j \leq c$	$i \neq j$	$ax_i^+ - bx_j^+ \leq c$	$\wedge$	$-ax_i^- + bx_j^- \leq c$
(−+ case)	$-ax_i + bx_j \leq c$	$i \neq j$	$-ax_i^+ + bx_j^+ \leq c$	$\wedge$	$ax_i^- - bx_j^- \leq c$
(+0 case)	$ax_i \leq c$		$ax_i^+ - ax_i^- \leq 2c$		
(−0 case)	$-ax_i \leq c$		$-ax_i^+ + ax_i^- \leq 2c$		

Table 3.1: Monotonizing for converting TVPIs into monotone-TVPIs

It can be seen that in the new monotonized system is TVPI as well, with each constraint having exactly one positive coefficient element and exactly one negative coefficient element with no other types of constraints (like, for example between two elements with positive coefficients). Hence, the system can be encoded as a *directed graph* with the direction of edges from the negative coefficient vertices to the positive coefficient vertices. We call the monotone versions of TVPI as MTVPI constraints. The monotone versions of UTVPI of course are MUTVPI or DBM constraints. The monotonizing transformation has been invented and used many times, mostly in transforming the UTVPIs to MUTVPIs, but the above is in its most general form, where it is used to transform TVPI to MTVPI constraints.

Hidden behind the above monotonizing transformation is a rounding scheme of rationals into integers. This is because, *integral* general (non-monotone) TVPI and UTVPI systems are computationally more complex than their corresponding monotonized variations; these integral systems have NP-Complete reductions. As an example for the same, the constraints in the formulation of vertex cover problem (or its complement, the maximum independent set) have just two non-zero elements per row. Here is the ILP formulation of the same.

Vertex cover ILP formulation. Let $G = (V, E)$ denote an undirected graph. Each vertex $i \in V$ is labelled by a cost $c(i) \geq 0$. The vertex cover problem is to find a *cover*, namely a collection of weights $x_i \in \mathbb{Z}$ for the vertices, so that the weighed sum $\sum_{i \in V} c(i)x_i$ is minimized,

$$\text{subject to} \begin{cases} e = (i, j) \in E : & x_i + x_j \geq 1 \\ 1 \leq i \leq N \in V : & x_i \in \{0, 1\} \end{cases}$$

The vertex cover problem is strongly NP-Complete, being one of Karp's 21 classic problems [GJ79, prob. GT1, p. 190]. It can be seen that the constraints of the vertex cover problem are UTVPI constraints with only the additional restriction that $x_i \in \mathbb{Z}$. But, it can be noticed that the constraints are non-monotone UTVPI. Enforcing them to be monotone and making a rational relaxation for each x_i, namely by $x_i \in \mathbb{Q}$; $0 \leq x_i \leq 1$ and solving the resultant MUTVPI constraints by graph theory methods would be an approximation algorithm [GLS93, GJ79] of the original NP-Complete algorithm.

When the original system is UTVPI, the above monotonizing transformation converts it into a Totally Unimodular system, while the original may not be. Any MUTVPI (or DBM) system is Totally-Unimodular [Sch86, Chapter 19]. This is another illustration that solving MUTVPI system is computationally simpler than solving general UTVPI systems.

3.3 Linear Programming on Convex Polyhedra

Linear Programming is one of the most powerful tools in the algorithmician's toolkit and its history dates back to 1963 with Dantzig's classic work [Dan63], coinciding with that of algorithms, and algorithmic analysis itself. In 1979, Khachiyan (Hačijan) [Hač79] proved that Linear Programming can be solved in polynomial time using *ellipsoid method*. This was followed by Karmarkar [Kar84], who introduced *interior point methods* for the same. Though both the above algorithms have weakly polynomial time complexity, the former has only been a theoretical interest, while the latter have proved to have much practical value beyond linear programming. They have been applied with much success to non-linear programming problems like quadratic programming, convex programming and semi-definite programming [BV04].

From Megiddo's result [Meg84], it is known that when the dimension is fixed, the complexity of LP is strongly polynomial, but no specific algorithm is known for the general case, i.e., when the dimension is also considered a variable. Whether there exists a strongly polynomial time algorithm for simplex is still an open question and it has been named in Smale's 18 unsolved problems [Sma98] listed in 2000[4].

In spite of all the above, the most widely used algorithm for linear programming is the simplex algorithm itself, whose basic structure has not changed much since its original discovery by Dantzig [Dan63]. It has been extremely successful and academia, research as well as industry would consider that a problem is solved when it is reduced to a LP which uses simplex algorithm. There are three views which are equivalent about simplex algorithm:

- [Algorithmic] Simplex algorithm traverses the polytope[5] from a vertex to its adjacent vertex.
- [Complexity] For any fixed pivoting rule, there exists a particular Klee-Minty cube, for which the simplex algorithm traverses a *hamiltonian path* in the graph of the cube.
- [Combinatorial] The maximum path length of a traversal of simplex algorithm is upper-bounded by the *diameter of the graph of the polytope.*

[4] in the spirit of Hilber't 23 problems posed at the beginning of previous century

[5] Though a polytope, a bounded polyhedron, is geometrically different from a polyhedron, algorithmically and combinatorially restricting the discussion to just polytopes does not pose any limitation; equivalence can be obtained by transformations like homogenization and de-homogenization [Zie06]. Even tools like polymake [GJ00] support these transformations.

3.3.1 Algorithmic and complexity theoretic view

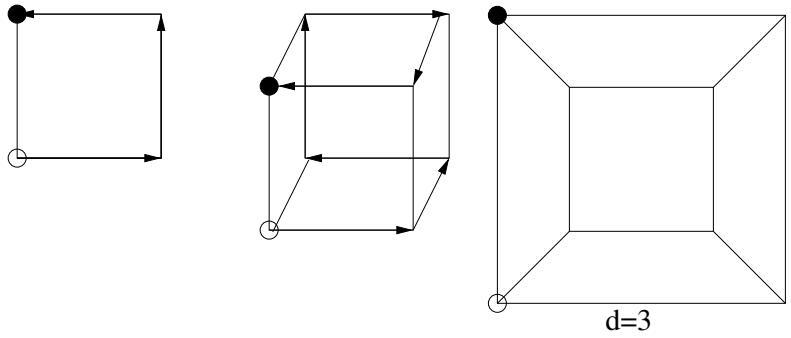
The algorithmic view is well understood by basic linear programming. It is well known that simplex algorithm achieves an optimal value at a vertex. To exploit this, Dantzig's simplex algorithm, picks a particular vertex in its first phase, and in the subsequent phases follows vertices adjacent to each other in an iterative fashion, until further improvement in the cost function is not possible. This algorithmic characterization gave rise to many kinds of pivoting rules, ways in which the next adjacent vertex can be picked. The *bland's rule*, a trick to avoid degeneracy (or cycling), is used generally universally and is known to work extremely well in practice [Bix02]. Bland's rule, along with dual simplex strategy is used as the engine of Feautrier's famed PIP [Fea88] as well.

The Simplex algorithm's success is a classic case of a local search strategy working extremely well on the average case. Spielman and Teng [ST04] even proposed *smoothened analysis*, a new method of algorithmic average case analysis, where the hard instances of the input are perturbed using some noise—gaussian with zero mean and variance—so that an easy instance is obtained which runs in guaranteed polynomial time with high probability. This method of Spielman and Teng is as much about average case analysis of simplex, as it is about perturbation analysis of algorithms.

Observation 3.3.1 [Experimental complexity of (LP using) simplex algorithm]. Let $Z(m, n)$ be the complexity of LP using the Simplex algorithm with m constraints and n variables. Determining typical value of Z is not an easy task, as it depends on many details of the algorithm, like the pivoting rule, the method to exploit sparsity, and even many implementation details, like ratio of m and n, the type of arithmetic used: fixed vs. arbitrary-precision, integers vs. rationals and so on. In this dissertation however, and only for *some* empirical comparisons, we use the following folklore result which does not count the bit-size complexity: On a *typical* input, $Z(m, n) = \mathcal{O}((m + n)mn)$ on *average*.

Even though we have added the above caveats, we should note that the above estimate is practically very accurate for the LP programs we observe. In the above empirical estimate, which is obviously not strongly polynomial time (in which we are not counting the bits), we assume that the gaussian elimination steps of the simplex are very fast, exploiting sparsity, and assuming a linear number $\mathcal{O}(m+n)$ of pivoting steps. Similar conclusions have been made in the literature as well. For example, Todd [Tod02] says: "*... remarkable fact: the primal simplex method typically requires at most* $2m$ *to* $3m$ *pivots to attain optimality*".

Complexity theoretic view. The complexity theoretic or worst-case view is best explained by the Klee-Minty traversal, where all the vertices are traversed by the simplex algorithm as in Figure 3.4 on the facing page.(i). The exponentiality of the simplex method was proved by Klee and Minty [KM72]. For every fixed pivot rule, there exists a version of Klee-Minty cube. Also, the exponentiality of Klee-Minty illustrates that the initial local search choice spoils the whole search mechanism. In either of the cases in Figures 3.4 on the next page.(i), the simplex algorithm could have reached the optimal value in one step instead of traversing a path that touches all the vertices before returning to the final optimum point. The path of Simplex on the Klee-Minty cubes could be considered as a hamiltonian path on the graph of the polytope [Zie06, Chapter 3], whose vertices are the vertices (0-dimensional faces) of the polytopes, and



(i) Simplex traversal in worst-case

(ii) Graph and diameter of a hypercube

Figure 3.4: Simplex traversal, Klee-Minty style traversal, and Graph and diameter of a polytope

whose edges are the edges (1-dimensional faces) of the polytope (a 3d cube) as shown in Figure 3.4.(ii).

3.3.2 Combinatorial view

The combinatorial view has to do with recent advances in geometry with the diameter of the graph and the Hirsch conjecture [Zie06] of 1957:

Hirsch conjecture. Let $m > d \geq 2$ and let P be a d-dimensional polytope with m-facets (inequalities), and let $G(P)$ be its graph. Then $\text{diam}(G(P)) \leq m - d$.

Having been unresolved for more than half a century, the Hirsch conjecture has been proved to be incorrect by Santos [San10] in 2010, who gave a counterexample for it. Santos's proof consists of constructing a 43-dimensional polytope (a prismatoid) with 86 facets having a diameter greater than 43. After Santos construction, smaller dimensional constructions have been given.

Remark about Hirsch conjecture and observation. 3.3.1 In the estimate in Observation 3.3.1, we are just assuming that for *normal cases* in polyhedral compilation, we are observing linear number of pivots, namely $\mathcal{O}(m+n)$. This could mean $m+n$ pivots, $2(m+n)$ pivots, $2m+3m$ pivots etc., as observed by Todd [Tod02]. The recent Hirsch conjecture disproval on the other hand is about the problem complexity, not algorithmic complexity, and is about absence of any linear time path for *any* polytope; this does not mean a $\mathcal{O}(m+n)$ number of pivots, but exactly $m-d$. Hence, it being disproved by the above mentioned work of Santos does *not* have practical implications, and our estimate is *independent* of the recent disproval of Hirsch conjecture.

A variation of Hirsch conjecture may be true for polyhedra: namely there may be two constants a and b such that diameter is always $am + bd$. This new conjecture is still open and is called *Linear Hirsch Conjecture*.

Linear Hirsch conjecture. Is there a linear function $f(m,d)$ such that for any polytope P of dimension d and with m facets, $\text{diam}(G(P)) \leq f(m,d)$?

Problem	Convex Polyhedra	TVPI	m-TVPI	UTVPI	m-UTVPI (DBM)
LPFEAS	WPT	$\mathcal{O}(mn^2 \log m)$		$\mathcal{O}(mn)$	
LPOPT	WPT	$\mathcal{O}(m^3n^2 \log m \log B)$		X	
ILPFEAS	SNPC	SNPC	WNPC	SNPC	X
ILPOPT	SNPC	SNPC	WNPC	SNPC	X
Fourier-Motzkin	$\mathcal{O}(m^{2^n})$	$\mathcal{O}(mn^2 \log m)$		$\mathcal{O}(mn^2 \log m)$	
GEN ($\mathcal{V} \leftrightarrow \mathcal{H}$)	EXP(n), EXP(d)	EXP(n),EXP(d)		EXP(n), EXP(d)	

Table 3.2: Convex Polyhedra, (U)TVPI Sub-Polyhedra and monotone (U)TVPI Sub-Polyhedra: Comparison of (worst-case) complexities. The problem complexities are denoted in red, while the worst-case complexities of the best known algorithms are denoted in blue. Also, $X = \mathcal{O}((m \log n)(m + n \log n))$.

The above linear Hirsch conjecture, as well as similar polynomial variations of it, are still open at the time of this writing. If any of them are proved to be correct, then the implications of solving LP through Simplex could be huge [KS10] mainly because of the bridge they would make between the gap between the intuitive understanding of the algorithm and its theoretical complexity.

3.4 Complexities of Convex Polyhedra and Sub-polyhedra

In Table 3.2, we summarize the relevant complexities[6] of general convex polyhedra with TVPI and UTVPI Sub-Polyhedra along with their monotone versions. Each problem instance is defined to be in n variables, m inequalities and d dimensions. For complexities that are not exactly known, we will be using their corresponding complexity class. More specifically, WPT stands for Weakly Polynomial Time, and SNPC and WNPC stand for Strongly NP-Complete and Weakly NP-Complete respectively[7].

Linear Programming (rational) feasibility (LPFEAS). The LPFEAS problem has weakly polynomial time complexity for convex polyhedra. On TVPI polyhedra, it has strongly polynomial time complexity because of results of Hochbaum-Naor [HN94] and Cohen-Megiddo [CM94]. Since the problem is defined over rational polyhedra on which the monotonization transformation does not change the precision, the non-monotone variants of the problem on both TVPI and UTVPI polyhedra have the same complexities as their monotone variants. The complexity of feasibility of UTVPI polyhedra and m-UTVPI polyhedra is the complexity of Bellman-Ford algorithm [Bel58, FF62].

Linear Programming (rational) Optimization (LPOPT). The LPOPT problem on convex polyhedra is weakly polynomial time. On TVPI polyhedra, the complexity is $\mathcal{O}(m^3n^2 \log m \log B)$ due to the result of Wayne [Way99]. Though better than convex polyhedra, it is still weakly polynomial time due

[6] In this table, we take a little liberty of mixing problem complexities with algorithm complexities.
[7] We thank Dorit Hochbaum for suggesting corrections to this table.

to presence of B in the complexity formula. For UTVPI polyhedra, as the constraint matrix has only $\{0, \pm 1\}$ elements, the problem has strongly polynomial time complexity. Using the duality result of linear programming, the *flow problems*–of which min-cost-flow and Bellman-Ford problems could be considered as special cases, for TVPI and UTVPI polyhedra respectively–can be shown to have the same complexities as the LPOPT problem.

Integer Linear Programming Feasibility/Optimization (ILPFEAS/ILPOPT). The ILPFEAS problem is well known to be strongly NP-Complete [GJ79, prob. MP1, p.245] for convex polyhedra. On TVPI polyhedra, the problem has been proved to be strongly NP-Complete by Lagarias [Lag85]. For general UTVPI polyhedra as well, the same complexity holds as the vertex cover problem can be reduced to it. But, the corresponding problem over monotone TVPI systems is weakly NP-Complete as proved by Hochbaum-Naor [HN94] with complexity $\mathcal{O}(mnB^2 \log(Bn^2/m))$. For monotone UTVPI polyhedra, the corresponding complexity is $\mathcal{O}(mn \log(n^2/m))$ which is clearly strongly polynomial time. All the entries in the ILPOPT row have the same complexities as the corresponding complexities as ILPFEAS. (Though strictly speaking, only decision problems are tackled by NP-Completeness.)

Fourier-Motzkin. The problem of Fourier-Motzkin is well known to be doubly exponential in the number of variables with the number of inequalities rising to an exhorbitant $\lfloor \frac{m}{2} \rfloor^{2^n}$. Hochbaum-Naor's result on the other hand shows the polynomiality for TVPI polyhedra. The same result applies to for UTVPI polyhedra as well.

Generator representation (GEN ($\mathcal{V} \leftrightarrow \mathcal{H}$)). Even for Interval polyhedra, the problem of converting from constraint to generator representation is exponential as illustrated by the n-dimensional hypercube. Hence this problem is exponential for any of the polyhedra and sub-polyhedral classes (or lattices) considered.

3.5 Polyhedral Approximations in Static Analysis

The static analysis community has been considering the tradeoffs between different Sub-Polyhedra as a means of trading expressiveness for computational complexity in solving abstract interpretation problems [CC77]. The pioneering work of Cousot and Cousot [CC76] introduced Interval (Interval) abstract domain, and was followed by the introduction of Polyhedron (Poly) abstract domain by Cousot and Halbwachs [CH78]. In recent times, this was followed by the breakthrough work by Miné [Min01a] on Octagon abstract domain (Octagons) in 2001. This work led to the use of many Sub-Polyhedra which have been found to be both expressive enough as well as cost effective for many static analysis and other run-time program verification problems like value range analysis, bounds checking of indices of array variables, buffer overflows, validation of array accesses etc. It is interesting that almost the entire range of Sub-Polyhedral abstract domains being used in recent times fit within the range defined by the works of Cousot, Cousot and Halbwachs in the late 70s: interval abstract domain by Cousot and Cousot [CC76], and polyhedron abstract domain by Cousot and Halbwachs [CH78].

In formulating these applications, the static analysis community uses descriptions of programs called *abstract domains*. An abstract domain is presented as a lattice $\langle D, \sqsubseteq, \sqcup, \sqcap \rangle$, with $\sqsubseteq$ being an ordering predicate, and $\sqcap$ and $\sqcup$ being the associated meet and join operators respectively. Many of the static analysis problems can be formulated as general polyhedral operations in the Poly abstract domain, in which D are convex sets that can be described by finite number of general linear inequalities. But the static analysis community has found Poly to be too expensive for the applications of interest, for several reasons.

Firstly, the need for join ($\sqcup$) operators on polyhedra. The most precise join operation calculates the closure of the convex hull of the two input polyhedra, that is, a set of inequalities that represent the smallest space that includes the two input polyhedra. One straightforward way to do the above is to use the generator representation of the two polyhedra and compute the convex hull. This method needs the generator representation of both of them, and a conversion from constraint representation to generator representation and back. Hence it can take exponential time as convex-hull of even two d-dimensional hypercubes requires just $2d$ inequalities for each hypercube, while needing 2^d vertices. In spite of many practical techniques to reduce the complexity, the asymptotic cost cannot be reduced by a large amount [HMG06].

Secondly, it is the recursive nature of (flow) equations which are solved in an iterative way until a fixpoint is reached. In solving these equations, the meet ($\sqcap$) operator usually adds a few additional inequalities to an existing system and then the system is used for other operations like entailment checks (checking if $P_1 \subseteq P_2$ holds), or subjected again to join or even the projection operations. These requirements lead to the need for a so called *closed system*, in which all information between any pairs of variables can be inferred by using the set of constraints that involve only those pairs of variables. An example illustrating for closure operation on a TVPI system is annotating the system $\{ax + by \leq c;\ dx + ez \leq f;\}$ by the additional implied constraint $aez - bdy \leq af - cd$ if $a > 0 \wedge d < 0$, so that the relationship between variables y and z can be discovered in constant time. Geometrically, closure can also be considered as the process of calculating all possible projections. Needless to say, closure is very costly for Poly.

Finally, another reason is the scale of problems that are being dealt with. For the more demanding applications coming from run-time verification problems, the more powerful Poly are simply unaffordable. Miné [Min06, page 4] says "*... polyhedron domain has a memory and time cost unbounded in theory and exponential in practice*", while Laviron and Logozzo [LL09, page 2] say "*... Poly easily propagates the constraints. However, ... the price to pay for using Poly is too high: the analysis not scale beyond dozens of variables, ... while mostly one wants to solve hundreds of constraints and variables.*". These results are further substantiated in the earlier study by Halbwachs et al. [HMG06]. It should also be noted that the abstract interpretation problems also have to deal with memory scalability problems because of large size of input instances that they aim to solve. The combined effects of the high cost of time as well as space complexities of Poly cause the above mentioned scalability issues.

Hence, Sub-Polyhedral abstract domains started being used, by either restricting the shape of the polyhedron (the weakly relational domain approach) or by a priori limiting the number of linear-inequalities that can be solved (the bounded domains approach). The most obvious in the first category are (multidimensional) intervals, or "boxes", which have been used by Cousot and Cousot [CC76]. Beyond these

Sub-Polyhedra	Approximation (nature of constraint) $(a, b, c \in \mathbb{Q};)$
Intervals [CC76]	$a \leq x_i \leq b$
DBM [Bag97, Min01a, SKS00]	$x_i - x_j \leq c; x_i, x_j \geq 0$
Octagons (UTVPI) [Min01a, Min01b, Min06]	$\pm x_i \pm x_j \leq c$
TVPI [SKH02]	$ax_i + bx_j \leq c$
SubPolyhedra (SubPoly) [LL09]	LinEq $\otimes$ Interval
Pentagons [LF10]	$a \leq x_i \leq b \wedge x_i < x_j$
Logahedra [HK09]	$ax_i + bx_j \leq c; a, b \in 2^{\mathbb{Z}}$
Octahedra [CC04]	$\sum a_i x_i \leq c;\ a_i \in \{-1, 0, 1\}$
Convex Polyhedra [CH78]	$\sum a_i x_i \leq c$

Table 3.3: Some Sub-polyhedra used for Abstract Interpretation

are the Difference Bound Matrices (DBMs or MUTVPIs) by Bagnara et al, Miné and Shaham et al. [Bag97, Min01a, SKS00], Octagons, a.k.a. Unit Two Variables Per Inequality (Unit-TVPI or UTVPI) by Miné [Min01a, Min01b, Min06], Two Variables Per Inequality (TVPI) by Simon, King and Howe [SKH02], SubPolyhedra (not to be confused with the generic term Sub-Polyhedra using in this thesis) by Laviron and Logozzo [LL09], Octahedra by Clariso and Cortadella [CC04] etc. Other interesting ones include Logahedra by Howe and King [HK09] and Pentagons by Logozzo and Fähndrich [LF10] and template polyhedra by Sankaranarayanan et al. [SSM04, SSM05]. (U)TVPI is a special case of linear, (unit-)two-variable, template polyhedra. More powerful than all of these, of course are the Polyhedron (Poly) abstract domain by Cousot and Halbwachs [CH78]. Some of these varieties of Sub-Polyhedra are summarized in Table 3.3.

Of these, we discuss in detail the (U)TVPI domains primarily because of their worst-case polynomial time complexities, because of having graph theoretic as well as polyhedral interpretations, as well as because of the maturity of the existing tools and applications.

3.6 A Summary of Related Work

In this section quickly (re-)summarize the related work. Though this chapter had summaries of various works at different places, it is intended that this kind of summarization of related work at a single place would help in understanding the complexity context of sub-polyhedra, as well as place in focus our search for proper abstractions with the correct asymptotic complexities.

Algorithms for (U)TVPI Sub-Polyhedra. Sub-polyhedra, in particular (U)TVPI polyhedra had been of interest to the theoretical and algorithmic community because of the search for strongly polynomial time algorithms. UTVPI was pioneered by the work of Pratt [Pra77], while TVPI was by Megiddo [Meg83]. The state of the art in feasibility testing for TVPI systems is strongly polynomial time algorithms by Hochbaum and Naor [HN94], and Cohen and Megiddo [CM94] and for optimization of TVPI systems is weakly polynomial time algorithms by Wayne [Way99]. We are not aware of existing implementations of the above algorithms.

The strongly polynomial time algorithms for TVPI system for feasibility [HN94, CM94] should be contrasted with the weakly polynomial time algorithm by Wayne [Way99]. This discrepancy is an example of a *complexity theoretic duality gap* between the two problems. At the time of this writing, no strongly polynomial time algorithm is known for general optimization, namely objective function of arbitrary length, on TVPI polyhedra. Special cases, like maximization of a single variable and finding lexicographic maximum can be solved using the strongly polynomial algorithms which solve feasibility.

The state of the art for feasibility and optimization of UTVPI systems is by the well understood Bellman-Ford algorithm, which involves constructing a nearly $2m \times 2n$ size incidence matrix (constraint graph) encoding the difference constraints of the monotonized system and testing it for presence of negative weight cycles using the shortest-path problem. The latter has been extensively studied, beginning from [Bel58, FF62] to most recently in [CGG$^+$10]. Efficient, though closure based methods for the same [Min06, BHZ09] are available in Apron [JM09] and PPL [BHZ08, PPL].

Monotonizing transformation. The monotonizing transformation can be done either on TVPI or UTVPI constraints to obtain a TU matrix. It has a rich history from the study of pre-Leontief matrices by Dantzig: a general convex system where each constraint has at most one positive element is called a *pre-Leontief* system. The reduction transformation of UTVPI to MUTVPI for algorithmic purposes seems to have been re-invented many times, though Hochbaum [Hoc04, HN94] attributes the formalization of its application to Edelsbrunner et al. [ERW89]. In the many works of Hochbaum like [HMNT93, Hoc98, HN94], it has been used for approximation of NP-Complete problems like vertex-cover, minimum-SAT, scheduling with precedence constraints, generalized 2SAT etc. Even the dual version of monotonizing the columns has been studied by Hochbaum [Hoc04] so that it can work as approximation algorithm for the flow problems whose duals are TVPI programs, like maximum-cut problem (an NP-Complete problem) and generalized flow problem. The abstract interpretation community has also been using it beginning with the work of Miné [Min06], though the attributions in some references [LM05, SS10] are incorrect.

Our presentation on monotonization closely follows Hochbaum's [Hoc04] and Miné's [Min06, Min04]

presentations. Miné [Min06] uses the above transformation for transforming the UTVPIs to MUTVPIs (DBMs). With such transformation, he uses an adjacency matrix representation for the DBM could be used for storing the original UTVPI constraints. Though we have covered the monotonization in its most general form, namely for TVPI constraints, for implementational reasons however, we would be converting only UTVPI systems into MUTVPI systems. Also, for most of this dissertation, we deal with rational polyhedra and hence, the loss of precision by monotonization does not affect us. An exception to this is in Chapter 8 on page 101, where we extend the already developed rational approximation framework to obtain integer approximatons of NP-Complete problems. Even for many abstract interpretation purposes like the use of Octagons by Miné, the conversion of UTVPI's to DBMs is not a concern as they too are *mostly* concerned with the emptiness of the rational polyhedron.

Simplex complexity and Hirsch conjecture. In the complexity aspect, Sub-polyhedra fare better than general convex polyhedra for which efficient algorithms are usually based on the widely used Simplex algorithm [Dan63]. Schrijver [Sch86] is a good reference for linear programming, while Ziegler [Zie06] is for the geometric aspects. Santos [San10, Zie06] disproved the Hirsch conjecture in 2010. Our presentation on Hirsch conjecture follows Kim and Santos's expository paper [KS10] and Kim's thesis [Kim10].

Sub-polyhedra in static analysis. The complexities of operations on general convex polyhedra are known [JMSY94, HMG06, Min06] to be considerably more time consuming than (U)TVPI polyhedra for abstract domain operations. Hence, Sub-polyhedra have been widely employed in abstract interpretation problems by the static analysis community as a means of trading precision for scalability. An excellent presentation of Sub-Polyhedra and its applications to abstract interpretation can be found in Miné's thesis [Min04]. A comparison of the use of these, as well as more varieties of Sub-Polyhedra, along with their applicability to polyhedral compilation can be found in our work [UC11]. Of the many classes of Sub-Polyhedra, (U)TVPI polyhedra have extensively been used [Min06, BHZ09, SK10], with impressive results such as in Astrée [BCC$^+$03, CCF$^+$09]. Surprisingly enough, UTVPI polyhedra have first been used by Balasundaram-Kennedy [BK89] for loop parallelization applications. We will summarize their approach in the next chapter.

3.7 Conclusions, Contributions and Perspectives

In this section, we review the conclusions and contributions of this chapter, as well as discuss some perspectives.

Conclusions. In this chapter, we study the complexities of algorithms on (U)TVPI Sub-polyhedra. We showed that they have excellent theoretical properties that can be put to use in polyhedral compilation, and in particular affine scheduling. We also study the use of polyhedral sub-classes (Sub-polyhedra) as abstract domains by the static analysis community. We discovered that the static analysis community has been exploiting the richness of sub-polyhedra as a means of trading precision for execution time, and the same theme could help in obviating the scalability problem in the polyhedral compilation as well.

Contributions. Our contribution is in understanding which Sub-polyhedra, among the ones which have been of interest to theoreticians or to the static analysis community, are possibly the best suited to reduce the complexity of affine scheduling. Part of this chapter has been published in [UC11].

Perspectives. One initial goal of this work was to see if any existing Sub-polyhedral libraries like Octagons by Apron [JM09] and PPL [PPL] or TVPI by TVPI-lib [SKH02] can easily be adapted to be used in the affine scheduling framework like the one developed in GRAPHITE [TCE+10]. We discovered that the reliance of the existing static analysis libraries on closure, and generator representation would be limitations for scheduling purposes. Addition of simplex solver in these existing libraries would not have helped the utility purpose of the libraries themselves because these varieties of sub-polyhedra are particularly suited for *graph theory based* methods of solution, not a *tableau based one* like simplex. Also, use of such an existing library would *still* need an approximation algorithm, one that transforms a general convex polyhedron into the chosen kind of Sub-polyhedra. Such algorithms have been not studied previously. This subject is developed in a later chapter (Chapter 6).

In the next chapter we will first study various methods in which approximations can be used in polyhedral compilation. This will be followed by proposal of a new scheme that uses (U)TVPI under-approximations of the polyhedra, by which low complexity algorithms of (U)TVPI polyhedra can be exploited.

Chapter 4

Polyhedral Scheduling and Approximations

4.1 Introduction

In Chapter 2 we showed that the current scheduling methods which are based on general convex polyhedra are unscalable. In Chapter 3 we studied various varieties of sub-polyhedra, along with a special emphasis on (U)TVPI sub-polyhedra. We learnt that because of the low complexity operations on these latter varieties, they could be used in affine scheduling. In this chapter, we study and propose methods to reduce the scalability bottleneck using (U)TVPI sub-polyhedra.

Overview of this section. Firstly, in the current section, we describe a rough method in which affine scheduling are made scalable. Next, in Section 4.2 we briefly discuss some previous techniques which have similar motivation as ours. This is followed by Section 4.3 where we study some existing algorithms that approximate a general convex polyhedron by a (U)TVPI polyhedron.

The above is followed by Section 4.4 where two new scheduling heuristic approaches which use sub-polyhedra are introduced, with an example to illustrate their working in Section 4.5. This is followed in Section 4.6 by a re-summarization of related work. Finally in Section 4.7, we present some conclusions, contributions, and perspectives.

4.1.1 Approaches for making affine scheduling scalable

The excellent worst-case complexities of operations and algorithms on sub-polyhedra provide a motivation for using them in polyhedral scheduling. Whichever be the flavor of sub-polyhedra—whether they are TVPI or UTVPI or some other—some choices have to be made about its use. Furthermore, there could be general convex polyhedra based approaches, that attempt to improve the scalability problem by employing proper heuristics. Here is one classification of such methods[8].

Dependence approximation. One may assume that the dependence polyhedra satisfy some criterion. The input to the optimization or parallelization problem is chosen to belong to a pre-determined flavor of

[8]The level of detail in this chapter is at an expert level. Though a quick introduction has been provided in Chapter 2, a gentler introduction is in Darte et al. [DRV00, Chapter 5] and Bastoul [Bas12, Chapter 2].

sub-polyhedra. The dependence sets which do not satisfy those criterion must be *Over-approximated* by an algorithm or heuristic. Some schedules may be lost, but only correct transformations will be modeled. This necessary approximation is in the same spirit as classical abstractions and algorithms presented in [YAI95, DRV00]. We briefly discuss these classical dependence approximations in Section 4.2.1.1, and then by a similar approach by Balasundaram-Kennedy [BK89] which uses UTVPI sub-polyhedra in Section 4.2.1.2.

Constraint (Farkas polyhedra) approximation. Another design choice is to keep the dependence relations as general polyhedra, but approximate the constraints on the affine transformation itself. After the transformation to the dual space through the Farkas lemma, the constraint polyhedron–the (μ, λ)-polyhedron of Farkas multipliers–must be *Under-Approximated* (UA). Again, some legal affine transformations may be lost. We will discuss constraint approximation with (U)TVPI sub-polyhedra in Section 4.4.

Simplex-based methods. Beyond the above sub-polyhedral methods are simplex-based heuristic methods. The approach of Feautrier in his modular scheduling [Fea06] involves reducing the complexity by calling the simplex algorithm on smaller systems or *modules* as well as making the simplex method aware of the dependence graph structure. We will study his approach in Section 4.2.3.1. In Section 4.2.3.2, we propose a new heuristic by which the simplex algorithm itself could be tuned to have fixed number of pivots so as to reduce its complexity.

Additional approximations. There are at least two more ways to use sub-polyhedra in polyhedral scheduling. These are *domain approximation* and *objective function approximation*. In most cases, iteration domains defined by the loop bounds form multi-dimensional intervals, and most non-interval cases belong to the UTVPI flavor or triangular loops: As observed by Feautrier [Fea92a], "*In nearly all practical cases, each iteration domain is defined by two inequalities per loop*". Approximations are possible, but require specific handling in the code generator to generate precise iteration domain traversals; we believe the problem can be reduced to code generation for irregular, data-dependent control flow [BPCB10]. Regarding the approximation of objective functions, the strongly polynomial time bounds for (U)TVPI polyhedra discussed in Chapter 3 only holds for a specific class of feasibility and optimization problems: when the objective function has bounded number of elements, or when they have the same sign, or when it is a lexicographic maximum. When the objective function is arbitrary, linear programming seems to be required. This is discussed in detail in the literature [HN94, CM94, Way99]. We believe solutions can be found to enable both kinds of approximations, but leave these as open problems.

4.1.2 Conditions for approximation

A general polyhedron has to be converted into a Sub-polyhedra for it use the algorithms or libraries that specialize in it. An approximation can be an over or an under-approximation. In both cases, the approximation algorithm must consider some issues:

- **Legality:** The approximated polyhedron should not yield illegal optimizations. Since the main emphasis is on correctness of the transformation, the approximation should satisfy legality conditions.

Examples for legal transformations are over-approximations in the dependence polyhedron space and under-approximations in the Farkas-polyhedron space.

- **Cost:** Low complexity running time: The conversion process should be asymptotically small (polynomial-time) as well as appreciably simpler. Further, it is preferable if its complexity is strongly polynomial.
- **Tightness:** The approximated polyhedron should be as close to the original polyhedron as possible. In case of rational approximation, the volume of the approximated sub-polyhedron should be as close to the original one as possible. In case of integer approximation, it should leave-out/add as few number of integer points as possible.
- **Expressiveness:** The approximation should give non-trivial results—feasible, non-degenerate polyhedra—for a sizable number of interesting inputs. Also, if the approximation algorithm does not take into account the objective functions, the feasible points in the result may not yield very interesting affine transformations in terms of the optimization problem they are attempting to address. Both feasibility and objective function affect the expressiveness of the overall approach. This expressiveness can be measured in terms of the quality of the overall optimization on approximate sub-polyhedra.

There are some more constraints that the algorithm must satisfy. We point them in Section 4.2.1.1 and 4.4.

4.2 Some Existing Approaches for Scalable Loop Transformations

In this section, we cover some existing approaches which can be adapted for scalability purposes, and ones which had already attempted to do so. First we quickly summarize the classic dependence over-approximations [YAI95] in the parallelization literature. Then, we summarize another early approach proposed by Balasundaram-Kennedy [BK89] which uses UTVPI polyhedra for task level parallelism purposes. This is followed by studying Creusillet's approach [Cre96] who proposed scalable approaches for array region analysis using both under- and over-approximations. Next, we summarize Feautrier's modular scheduling approach [Fea06] which is the first in the affine scheduling literature to declare the simplex algorithm itself as the main culprit of scalability. We also introduce a new method that involves tuning the simplex algorithm to run with fixed number of pivots, and follow with some miscellaneous alternatives, like the vertex method.

4.2.1 Dependence Over-Approximations

4.2.1.1 Classic dependence over-approximations

Over-approximations of dependence polyhedra, including sub-polyhedra and beyond-polyhedral abstractions, is a fairly well-known technique and have already been used in parallelization. This kind of dependence approximation leads to a loss of precision in the input itself. Different dependence abstractions have been proposed for dealing with loop nest optimization and parallelization. They have been excellently summarized by Yang et al. [YAI95]. Our view of them is summarized in the following hierarchy, where

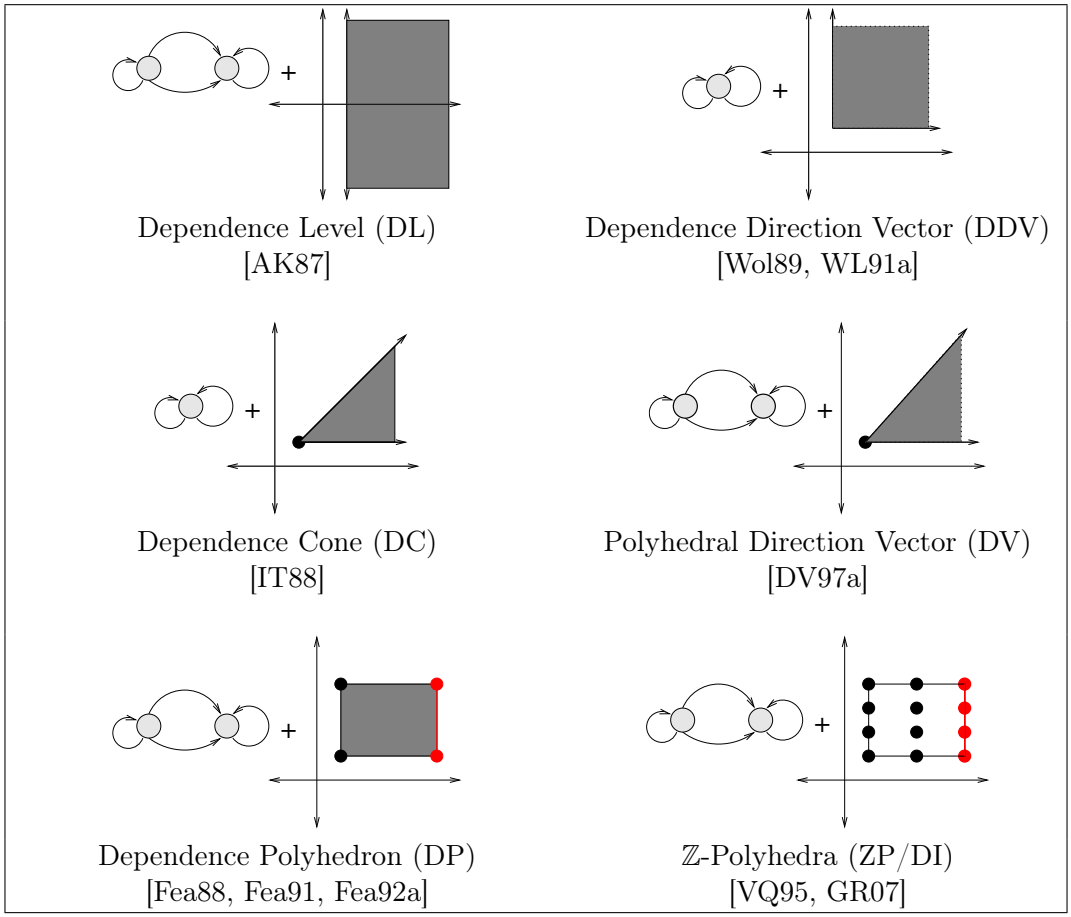


Figure 4.1: The "zoo" of some dependence abstractions (over-approximations) used in the parallelization literature. The first row represents non-polyhedral (sub-polyhedral) domains, while the second and third rows polyhedra based domains. The last row represents domains which can encode the domains including the parameters.

DL stands for dependence levels, DDV for dependence direction vectors, DC for dependence cones, DV for polyhedral direction vector, DP for dependence polyhedron and DI for dependence $\mathbb{Z}$-polyhedron:

$$\underbrace{\text{DL} \subset \text{DDV}}_{\text{sub-polyhedral}} \subset \underbrace{\text{DC} \subset \text{DV} \subset \text{DP}}_{\text{polyhedral}} \subset \underbrace{\text{DI} \subset \cdots}_{\text{beyond polyhedal}} \tag{4.1}$$

A pictorial representation of the abstractions is shown in Figure 4.1. Here is a short summary of each of the above.

Dependence Levels (DL). Allen-Kennedy [AK87] introduced Dependence Levels for loop distribution and vectorization of loop programs. It is the weakest dependence model in the above hierarchy, which neither takes into account complex dependences in the input programs, or the the domain information. But surprisingly, the abstraction captures all the information necessary for the particular kind of transformations that their parallelization algorithm aims for.

Dependence Direction Vectors (DDV). It was introduced by the work of Wolfe [Wol89]. Wolf-Lam [WL91a] successfully used it for unimodular transformations, namely: skewing, permutation and reversal, with their program model having a single block enclosing all the statements and hence with a single vertex in the Reduced Dependence Graph. This abstraction also is too weak to capture complex affine dependences as well as domain information. However, for unimodular transformations over a single basic block, it is just enough.

Dependence Cone (DC). Irigoin-Triolet [IT88] introduced it for parallelization as well as tiling. Their program model supports non-uniform dependences, which are summarized by rays and vertices that look like uniform dependences. Their parallelization algorithm cannot handle complexities in the Dependence Graph, because it is quite similar to Lamport's algorithm [Lam74]. However, their work along with Feautrier's, which introduced dependence polyhedron DP, are the first to use polyhedra in compilation of loop programs.

Polyhedral Direction Vectors (DV). Darte-Vivien [DV97a] re-introduced the polyhedral variations of direction vectors of Wolf-Lam [WL91a] in the context of inner-loop parallelization. Their parallelization algorithm cannot handle affine dependences as it assumes an a priori uniformization step, nor domain information including parameters. However, the use of polyhedral concepts, as well as being able to handle complex RDGs along with the link to KMW-decomposition makes their work unique.

Parametrized Dependence Polyhedra (DP). This was introduced by Feautrier [Fea88, Fea91] in the context of analyzing exact dataflow analysis of the input SCoPs. Using this encoding, affine dependences can be encoded precisely using parametrized polyhedra. This is the most powerful framework within polyhedral scheduling because of the generality of the affine scheduling algorithm of Feautrier [Fea92a]. The latter is both a particular scheduling algorithm, as well as a *family of algorithms* because of the crucial step into dual space that is taken by means of application of affine form of Farkas lemma. By this step, the

twin problems of encoding symbolic parameters and needing to handle dependence edges between different vertices in the dependence graph can be handled in a unified and simple way.

Parametrized $\mathbb{Z}$-polyhedra and Dependence Information (ZP/DI). $\mathbb{Z}$-polyhedra was introduced by Le Verge [VQ95] to be the intersection of a rational polyhedron with integer lattice. They are also called Linearly Bound Lattices (LBL) and can encode index sets with non-unit strides. Gupta-Rajopadhye [GR07] extend them for loop tiling.

All the above varieties of abstractions are different approximations of DP, and of DI which can only be described using Presburger arithmetic [Pug92, Pug91], and of the exact, potentially non-affine relations. Considering dependence relations, an approximation must be an over-approximation; it will only yield correct transformations, although it may induce a loss of potentially interesting schedules.

At this time of writing (in 2012/2013), the beyond polyhedral dependences using $\mathbb{Z}$-polyhedra are well considered to be unscalable because of the integer programming problems they have to solve. In Chapter 2, we had also established that polyhedral scheduling is as well unscalable, though it involves solving only a simplex algorithm. So we can safely consider that sub-polyhedral dependence abstractions could be scalable. But, the overall motivation for the sub-polyhedral abstractions in the hierarchy in Equation 4.1 on the preceding page as well as the original papers that proposed them, has been to solve a particular parallelization algorithm or transformation. For example, DL has been proposed for data-parallelism detection and loop reversal, and DDV for all unimodular transformations mainly targeted at inner-loop parallelism. Modern architectures need transformations more powerful than just inner loop parallelism that these can offer. Arguably, the primary transformations that modern architecture with multi-cores needs are outer-loop parallelism along with inner-loop parallelism, tiling and loop fusion. Also, the goal of the comparison by Yang et al. [YAI95] has not been to find whether the particular dependence abstraction *scales* well, but whether the particular abstraction *fits* well its application purpose.

4.2.1.2 Balasundaram-Kennedy's simple sections

UTVPI polyhedra have previously been used by Balasundaram and Kennedy [BK89] in loop parallelization applications, namely for data dependence analysis and domain simplification. The authors call them *simple sections* and are perhaps the first to use sub-polyhedra in compilation. Since their work is also related to parallelization, same as our objective, we summarize their work.

The problem that Balasundaram-Kennedy want to solve is *task level parallelism*, namely, detection and exploitation of parallelism between different loop nests, or between different subroutines. For this, they propose *Data Access Descriptors (DADs)*, as an abstraction for summarization of data dependences between different tasks.

Similar abstractions have been proposed earlier by Callahan [Cal87], who suggested that dependences between tasks be summarized using *Regular Section Descriptors (RSDs)*, which are but affine functions of the form $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$, where A is restricted to only upper-triangular matrices.[9] Though this abstraction

[9] This is only one of the many definitions for RSDs. In the literature, there exist other definitions. According to their initial definition [CK88], they can represent elements, rows, columns, diagonals, and their higher dimensional analogs. Another definition is that they can encode array regions in a triplet notation (`lb : ub : step`) on each array dimension.

stems from the valid observation that normal loop nests mostly have domains and data accesses of the above variety and hence are quite useful for summarization, operations on RSDs indeed are as costly as general convex polyhedra. Hence Callahan introduces *restricted-RSD* (RRSD), a relational domain which can encode only accesses of the type: single rows, single columns, diagonals, or access of entire arrays.

Beginning from the above two kinds of descriptors, Balasundaram-Kennedy introduce DADs as an intermediate descriptor using simple sections—which are but UTVPI polyhedra—as access patterns, with a goal of achieving a precision vs. cost tradeoff. The following is their definition of DADs:

Definition 4.2.1.2 [Data Access Descriptor (DAD)]. Given an n-dimensional array A that is accessed within a region $\mathcal{R}$ of a program, the Data Access Descriptor (DAD) for A in $\mathcal{R}$, denoted by $\delta_{\mathcal{R}}(A)$ is a 3-tuple $\langle \theta, S, \tau \rangle$, where θ is the reference template, S is the simple section approximating the portion of A that is accessed within $\mathcal{R}$, and τ is the traversal order of S.

Example 4.2.1.2 [DAD of Balasundaram-Kennedy]. The following is a typical descriptor of Balasundaram-Kennedy:

$$\mathrm{Dep}_{R_1}(A) = \left\langle x_1, x_2 \;\middle|\; \begin{array}{c} 1 \le x_1, x_2 \le 100 \\ 2 \le x_1 + x_2 \le 200 \\ -99 \le x_1 - x_2 \le 99 \end{array} \;\middle|\; x_1 \succ x_2 \right\rangle$$

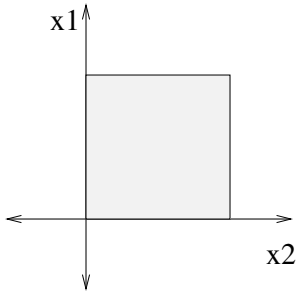
It can be seen that the data access patterns (the S fields) are UTVPI polyhedra. In Figure 4.2, we have shown some examples taken from their paper.

To obtain a collection of DADs from input loop program(s) with array accesses, Balasundaram-Kennedy suggest some over-approximation conditions including support of union and intersection operators on UTVPI polyhedra. They even suggest modifying Banerjee's conditions for dependence analysis. Balasundaram-Kennedy cite task parallelism and task pipelining as applications which could use their DAD abstraction.

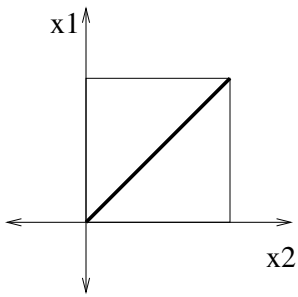
Clearly, Balasundaram-Kennedy's approach is a step in the right direction to solve the scalability problem of parallelization. They note that general convex polyhedra [Tri84, IT88], if used for encoding dependences, is quite general but costly. Simple sections could be considered to be a parametric integer set abstraction, a concept which was introduced later by Yang et al. [YAI95], and hence could be considered to form a basis of the later work. UTVPI as dependence abstractions are not directly comparable with other abstractions such as DC, but is larger than DDV and smaller than DV.

Balasundaram-Kennedy's observation to create an intermediate abstraction supported by low complexity operators of intersection and union preceeds the much later Octagon abstract domain work by Miné [Min06]. For static analysis purposes as well, their work could be considered as predating the latter work of using sub-polyhedra, but for the fact they do not introduce any normal form for use as an abstract domain. Miné's work [Min06] on introducing closure algorithms with tightening operators, along with his data structural support in Apron [JM09], makes it more general than their work.

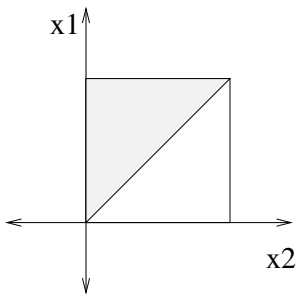
Even Balasundaram and Kennedy's framework also is not based on affine scheduling, which is much powerful, as it encodes many more kinds of useful transformations. Their parallelization scheme can only




```
for(i = 1; i <= 100; i++)
{
  for(j = 1; j <= 100; j++)
  {
    A[j][i] = ...
  }
}
```




```
for(i = 1; i <= 100; i++)
{
  for(j = 1; j <= 100; j++)
  {
    A[i][i] = ...
  }
}
```




```
for(i = 1; i <= 100; i++)
{
  for(j = 1; j <= i; j++)
  {
    A[i][j] = ...
  }
}
```

Figure 4.2: Some programs and the representation of their data access descriptors using Balasundaram-Kennedy's Simple Sections [BK89]

cover some specific kinds of transformations because of coarseness of the dependence graph. It needs to be extended for statement level parallelism and for loop-nests greater than depth 2. Extension of their work would also mean precise handling of parameters used in loop bounds because their input loop definitions can only have fixed bounds and not parameters. Though they suggest that their model can be extended for symbolic variables, it can easily be seen that the TVPI-ness is lost in such cases: $\{i + j \leq 10\}$ is a UTVPI constraint, while $\{i + j \leq N\}$ is not. The latter has to be handled either by making some assumptions on N, or by an over-approximation or even by solving parametric UTVPI programs. Such special handling of parametric variables is not needed in affine scheduling because it involves a solving a *non-parametric polyhedron* obtained by application of Farkas lemma.

4.2.1.3 Our view of dependence over-approximations

Feautrier's polyhedral approximation of dependences (as well as the more powerful DI abstraction) are really different from the ones lower in the hierarchy because they can handle the parameters symbolically. A parameter can be considered a constant that is not known at compile time, which is not equivalent to considering it as an additional index-variable in several key problems, including linear programming and piece-wise feasibility with respect to the parametric context. But the surprising fact is that Farkas-based affine transformations do not need to solve a parametric linear programming formulation at all. Hence it is clear that though solving parametric linear programs cannot be avoided for solving the dependence analysis problem, it can however be avoided for affine transformation algorithms using the Farkas lemma or Vertex method. Such algorithms can use existing sub-polyhedra without the need for a parametric extension.

The view in this dissertation is that polyhedral abstractions like DP which include a provison for handling parameters, should be used in loop transformations as they are extremely powerful to capture domains, dependences and transformations in a single framework as well as being extremely rich to handle complex affine transformations like the ones mentioned above. The scalability problem should be tackled by proper algorithmic under-approximations of such a rich space rather than pre-determining either the input dependence information or the type of transformations.

4.2.2 Creusillet's approximations in array region analysis

Creusillet in her dissertation [Cre96][10] suggests multiple ways in which array region analysis can be done, while building theoretical frameworks—exact and approximate semantics—which are independent of the particular representations and analyses. Specifically, she defines several types of array regions: READ and WRITE regions to summarize the effects of instructions and procedures; and IN and OUT regions that respectively contain the array elements imported and exported by the particular region.

Creusillet's work is not in the context of program fragments in a restricted context like SCoPs [Fea91], but in more general and unstructured languages like FORTRAN, with even an interprocedural context. She notes that analyses that begin with dependence analysis in a restrictive language like SCoPs may be quite costly; also, in spite of the preciseness that is possible, the limitatons may be *too* restrictive.

[10]We are thankful to Prof. François Irigoin who pointed us to the works in this section.

Analysis in more general languages however, while having a much wider area of application, conversely may be cheaper. The goal of her work is to perform analyses like interprocedural dependence analysis using READ and WRITE regions, and locality optimizations like array privatization using IN and OUT regions.

In this latter context, the traditional approaches have been to define may (resply. must) sets as over-(resply. under-)approximations of the exact solutions. Creusillet notes that under-approximations have been tried and found to yield trivial or uninteresting approximations. The main reason for this is that when the chosen domain is not closed under set-union—including domains like convex polyhedra [Tri84], and weaker ones like Regular Section Descriptors (RSDs) [Cal87, CK88]—the under-approximations are not uniquely defined. This directly means that, given an exact array region, no best under-approximation can be defined. Over-approximation however can be uniquely defined. Moving to more general domains—like Presburger formulae—for under-approximations may lead to unscalable solutions. Wonnacott [Won95] reports the results of such an approach using Omega [Pug92, Pug91], but his analyses are limited to intra-procedural level; in an inter-procedural level, it is not known how such general domains would scale.

Creusillet [Cre96, Chapter 4] therefore proposes a method which uses both under- and over-approximate analysis at the same time. Primarily, she proposes theoretical framework using Complete Partial Orders (lattice theoretic frameworks [CC77]) to define under- and over-approximations. This is followed by proposal of a computable exactness condition for checking whether a particular over-approximation is equal to the exact solution. In this latter case, the under-approximation need not be computed as the same solution works as both over- and under-approximations. The advantage of the above work is that exact computation of the costly under-approximation is avoided, while at the same time, achieving the same through the exactness conditions.

The framework is part of the PIPS project [PIP], thanks to her extensive implementation. Experiments on large-size real-life applications lead to several loops containing procedural calls were successfully parallelized, thanks to the algorithms suggested in her dissertation. Notable among the similar approaches is the work of Polaris [Tu95], where a similar approach has been applied using lists of RSDs. Further, Havlak's work [HK91] is also with a similar theme.

The work in this dissertation is not directly comparable to Cresillet's work; the former is about programs in a restricted context, while the latter is in a more broader context. Also, the former is primarily about affine transformations for tiling and code-generation, while the latter is about the dependence analysis and optimizations like array privatization. Further, in the above work, there is no *asymptotic* reduction in the cost of analysis; this is mainly because convex polyhedra have been used as elements of the semantic framework in the implementation.

In spite of the above differences, her work has a positive inflence on this thesis. Her algorithms and implementation have been tested extensively and offer interesting application perspectives for our work. Also, a sub-polyhedral domain like (U)TVPI polyhedra, along with supporting algorithms which find over- as well as under-approximations algorithms can augment the work; it could provide an improvement in the asymptotic compexity. This suggestion is because her work predates the more recent works of Miné [Min06] on UTVPI-polyhedra, and Simon et al. [SKH10]'s on TVPI-polyhedra. Further work is needed in making

the two communities of static analysis and loop-transformation build from the other's work.

4.2.3 Simplex Based Methods

4.2.3.1 Feautrier's scalable and modular scheduling

Feautrier in his scalable and structured scheduling [Fea06] begins with a complexity estimate of affine scheduling, and concludes that affine scheduling methods which use cubic time simplex algorithm to solve the linear programs for scheduling easily lead to algorithms with 6^{th} power complexity in the number of variables in the input program[11], making the parallelizer highly unscalable.

Feautrier hence proposes a modular specification language *Communicating Regular Processors (CRP)*, an extension of C programming language, in which smaller systems or *modules*, comparable to functions of C could be defined. The modules communicate with each other through `inport`s and `outport`s, and communication channels between two modules are multi-dimensional arrays. The modules in CRP system could be scheduled independently from each other, leading to smaller size simplex problems and hence better compilation times, almost linear in the number of modules in an input system. Feautrier also proposes search methods in which the original program could be decomposed into smaller systems. Clearly, the modules in CRP can be compared to *systems* of Alpha programming language [MQRS90, QR02] and to *processes* in Kahn Process Networks [Kah74]. Using the above CRP system, Feautrier reports the execution times of usual schedulers, starting from tens of minutes and ending much longer. These longer times concur with the ones we showed in Chapter 2.3.2 on page 11 using the PLuTo system.

To reduce the compilation times, Feautrier suggests several heuristic techniques. Like the earlier work of Feautrier [Fea92a], the modular scheduling algorithm begins the simplification process of Farkas multipliers with Gaussian elimination. Though the above step does not remove all the λ-multipliers which appear in equations, it is helpful to some extant. For elimination of all of the λ-multipliers, Feautrier suggests three approaches: Fourier-Motzkin, parametric linear programming and Minkowski decomposition. Feautrier suggests using Minkowski decomposition since an individual module is likely to be small, and this method does not generate redundant constraints like the other two methods. This step has the advantage of a redundancy elimination at an individual modular level, thereby contributing to a smaller number of constraints in the overall system.

The above steps leave a polyhedron with only μ-multipliers as variables. For solving the linear system involving only these Farkas multipliers, Feautrier suggests *stepwise scheduling,* a heuristic modification of the simplex algorithm, linked with an elimination process which exploits the structure of the dependence graph and hence the sparsity of the simplex tableau. He also suggests methods by which statements could be grouped together so that smaller simplex problems could be solved. All these steps lead to improvements in the execution times of the scheduler.

[11] The discepancy between this 6^{th} power and the 5^{th} power estimate we made in Section 2.3.5 on page 16 is simply because of different heuristic estimates of Simplex algorithm. Both concur that affine scheduling involves solving an LP problem of size $d|E| \times d|V|$. We assume optimistically that the latter is $\mathcal{O}(|E|^2|V|) \approx \mathcal{O}(|V|^5)$, while Feautrier assumes pessimistically that it is $\mathcal{O}(|E|^3) \approx \mathcal{O}(|V|^6)$. Though we presented *some* empirical evidence in Chapter 2 on page 7 followed by some more in the later Chapter 7 on page 89, further experimental validation of either of these complexity estimates, or even obtaining new one, by doing experiments on simplex across wide range of benchmarks like PolyBench [PB$^+$] needs to be done.

These methods are complimentary to using sub-polyhedra based methods: they can be complimentary to each other. Also, each of the individual steps in Feautrier's method, namely elimination of λ-multipliers locally, redundancy reduction, using the structure of dependence graph and grouping of statements certainly are steps in the positive direction. However, the main limitations of the approach is that it does not guarantee any *asymptotic improvement* over existing methods. Also, it still needs to construct the simplex tableau whose size $\mathcal{O}(mn)$ is *quadratic* in the size of the input system, and hence the input dependence graph. This can be a serious limitation in JIT compilers with memory constraints. Further, there is a chance that the optimizer loses its portability because it is tied to a particular simplex algorithm. Ideally, the parallelizer algorithm would like to use an out of the box implementation of simplex, like PIP [Fea88], and modifying the latter is an involved process.

4.2.3.2 Tuning the simplex algorithm

Another way of continuing to using Farkas based affine scheduling algorithms and still obtain a reduction in the time complexity, is to tune the simplex algorithm that is used for solving linear programs. Broadly, in this method, the simplex algorithm is made to run for a fixed predetermined number of pivots.[12]

As indicated earlier in Section 3.3.1 on page 28, we can assume that simplex *usually* takes $\mathcal{O}((m+n)mn)$ observed time, with the number of pivots being always linear ($\mathcal{O}(m+n)$) in number. These number of pivots can be reduced to a constant number, meaning that the simplex algorithm is tuned so that it automatically stops after K number of pivots, where K is an pre-determined constant, independent of either m or n. With this method, the observed complexity of simplex would be $\mathcal{O}(Kmn)$. If K is sufficiently small and fixed, like ten or so, then the overall complexity would be $\mathcal{O}(mn)$ which is not only small, but asymptotically the same as Bellman-Ford as guaranteed by UTVPI polyhedra. Also, this method has the additional advantage that it needs no further implementation than using a standard LP solver like PIP [Fea88] if the number of pivots can be bounded.

More empirical evidence is needed, but the following are the a priori objections to the above method, which is a heuristic for the simplex, rather than methods based on Sub-polyhedra.

This method cannot be made to give a `lexmin`, and the solution picked up by the basic feasible solution (initial solution) and improved a little by the constant number of pivots may be quite bad for a schedule.

For use in a parallelizer which needs a *theoretical* worst-case complexity guarantee, complexity of the above method is closely linked with a simplex implementation which can provide such a support for constant number of pivots. Though simplex implementation itself is fairly standard, a good implementation which can exploit sparsity, as well as provide good pivoting is quite an involved one. Compilers tend to use an out of box implementation provided by PIP [Fea88] or CPLEX [CPL] which do not have such a provison and such kind of investment in the LP infrastructure may be underutilized by such heuristic fixing of the number of pivots.

Also, the method *still* takes a $m \times n = \mathcal{O}(mn)$ space to construct and store the simplex tableau, and quadratic space can be quite heavy on the system. In spite of simplex algorithms taking various

[12] We thank Armin Größlinger for suggesting this approach at IMPACT-12 workshop as an alternative to our (U)TVPI based methods.

precautions about the table size increase by exploiting sparsity, our experience with PIP in our unscalability experiments (in Chapters 2 on page 7, and Chapter 7 on page 89) is that the above memory requirement is quite heavy for large SCoPs, especially for ones with more than 1000 dependences, and using any tableau based construction in JIT compilation could be unacceptable.

Further, we see below that using the (U)TVPI approximations can not only be applied to polynomial time affine scheduling methods for better worst-case performance, but also have the potential to give polynomial time approximate algorithms for loop scheduling problems that are NP-Complete. Also, if during code generation, integer vertices are needed, the UTVPI approximation can be tuned for that. These two problems cannot be solved with simplex based method(s) without paying the price of solving ILP programs. See Chapter 8 on page 101 for more details.

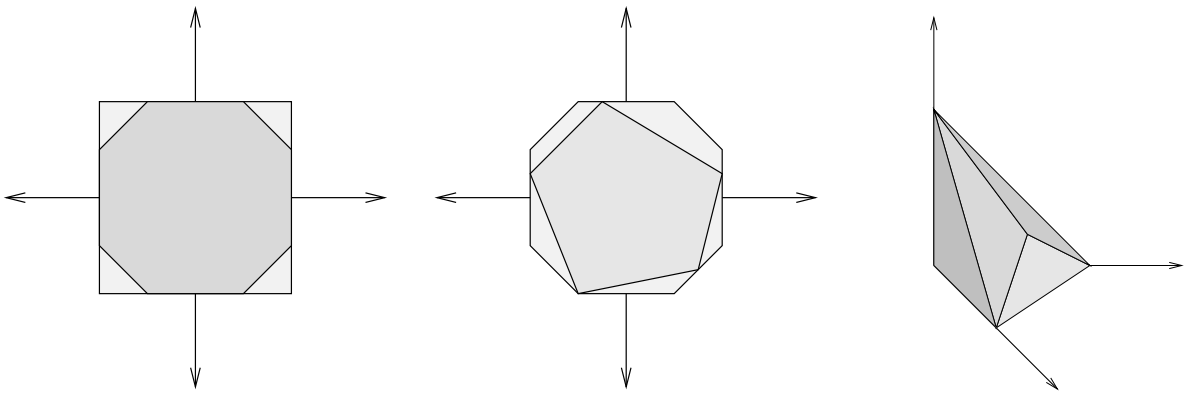
4.2.4 Other alternatives

To address this scalability challenge, we also consider some other existing alternatives for scalability problem. First we cover two non Farkas-based methods, the vertex method [RPF86, QD89] and the Darte-Vivien's method [DV97a] and then summarize Pouchet's heuristics [Pou].

The Vertex method. The vertex method was the most general "pre-Feautrier" affine scheduling algorithm and was proposed by Rajopadhye et al. [RPF86] and Quinton and van Dongen [QD89]. In this method, non-linear constraints on schedule coefficients (as variables) are avoided by converting dependence constraints from the hyperplane to the dual vertex/ray representation (Minkowski decomposition). This process leads to a constraint system with too many variables and constraints: practical dependence polyhedra are likely to have a very small hyperplane representation while having an exponential number of vertices. The method has more scalability limitations than Farkas-based scheduling, as shown by Feautrier [Fea92a] and confirmed experimentally by Balev [BQRR98].

Darte-Vivien's method. Darte and Vivien's algorithm [DV97a], which relies on the DV abstraction which has been briefly summarized in the dependence abstractions section, is a non-Farkas method. It involves one LP call per depth of the decomposition: $\mathcal{O}(Z) \approx \mathcal{O}(m^2 n) \approx \mathcal{O}(|E|^2|V|) \approx \mathcal{O}(|V|^5)$, if we assume the d, the depth of the decomposition tree to be bounded. Though it is likely that the gaussian eliminations of simplex algorithm terminate earlier for uniform dependences as evidenced in the slower rate of growth for `ITR-SEIDEL` example when compared to the `ITR-MATMUL` example in Section 2.3.2 on page 11, and there is even some empirical evidence [Mei96] to show that the running time of this algorithm is practically smaller than Feautrier's method, this complexity is *asymptotically* the same as Feautrier's algorithm,

Pouchet's FM library and heuristics. The other notable approach which reduces the cost of solving the linear systems that arise in polyhedral compilation is implemented in Pouchet's redundancy aware Fourier-Motzkin (FM) library [Pou]. It has been known that Fourier-Motzkin has doubly exponential complexity, though it empirically scales well [Pug92]. Though a detailed comparison is needed, we note



Interval Over-Approximation of an Octagon

Miné's UTVPI Over-Approximation of a Polyhedron [Min06]

Simon et al.'s TVPI Over-Approximation of a Polyhedron [SKH10]

Figure 4.3: Some Existing Sub-Polyhedral Approximations

that Pouchet's method does not *asymptotically* reduce the complexity in a predictable way. Nevertheless, it is clear that the overall search space contains many valid yet useless transformations, and many redundant ones in terms of generated code or behavior on the target platform. This is why we consider our work to be complementary to the FM approach and the heuristics therein.

4.3 Some Existing Algorithms to Approximate into Sub-polyhedra

From the sub-polyhedral literature, we are not aware of any systematic study of under-approximation algorithms. We however are aware of two algorithms on over-approximation. Here is a summary of them. Some more heuristic methods are listed in conclusions section.

4.3.1 Interval over-approximation

Interval over-approximation of an arbitrary polyhedron is a trivial approximation and has been assumed and solved many times in polyhedral literature.

For a system with n variables and m constraints, one can run $2n$ linear programs, twice per each variable x_i, with the objective function being $\max(x_i)$ and $\min(x_i)$, each returning the upper-bound and lower-bound respectively of variable x_i.

The above method is simple, and has cost $2nZ$, which on average case takes $\mathcal{O}(m^2n^2)$ time, and could be quite affordable if n and m are small. Further, the method can use an existing simplex implementation.

But, it is likely that the above approximations are relatively uninteresting for a majority of programs.

4.3.2 UTVPI over-approximation by Miné

Miné [Min06, Section 4.3] proposes an algorithm to over-approximate a general polyhedron into a UTVPI-over-approximation. The algorithm however uses the generator representation and has the cost $\mathcal{O}(d^2(|R|+$

$|V|)$), where d is the dimension and $|R|$ and $|V|$ are the number of rays and vertices respectively. It returns the tightest over-approximation-UTVPI of the given polyhedron.

Briefly, the method uses a greedy approach of incrementally building an Octagon, exploiting another already developed algorithm to do convex union of two octagons and octagonalize the result. The over-approximation algorithm begins with an arbitrary vertex of the input polyhedron, which is trivially an Octagon and iteratively adds vertices to it, each time obtaining an incremental octagonal convex-hull for that particular set of vertices which have been over-approximated. A similar procedure is repeated for each ray of the polyhedron as well. The result of this algorithm is the tightest over-approximation for the input polyhedron.

If the above algorithm of Miné is considered to be equivalent of Prim's algorithm for finding minimum spanning tree for a directed graph in the sense of maintaining a single tightest UTVPI-OA polyhedron of the generators already approximated, it can even be adapted to run similar to Kruskal's algorithm by maintaining a forest of UTVPI-OAs [CSRL01] which can be incrementally coalesced to form the overall UTVPI-OA.

Needless to say, the method given as well as its variation suggested above use the generator representation of the polyhedron which could be costly. We are aware of no efficient algorithm to directly compute the UTVPI over/under-approximations using only the constraint representation.

4.3.3 TVPI over-approximation by Simon-King-Howe

Simon, King and Howe [SKH10, Section 3.2.6] propose an algorithm to over-approximate a general polyhedron into its TVPI-over-approximation.

Briefly, the approximates a constraint $a_1x_1 + \cdots + a_nx_n \leq c$ of a Polyhedron P using the set of constraints $a_jx_j + a_kx_k \leq c - c_{j,k}$ for every $1 \leq j < k \leq n$. They propose that each $c_{j,k}$ can be calculated as the result of a linear programming problem: $c_{j,k} = minExp(\sum_{i\in[1,n]\setminus\{j,k\}} a_ix_i, \mathrm{OA}(P_i))$ and $\mathrm{OA}(P_i)$ is the incremental over-approximation polyhedron obtained at the i'th step of computation. The above step[13] needs to be iterated for each non-TVPI constraint of P.

For converting a general polyhedron with n variables and m constraints into a TVPI over-approximation, it can be seen that the above algorithm has cost n^2m^2Z, which in average case takes $\mathcal{O}(m^4n^3) \approx \mathcal{O}(n^7)$ time, assuming $m = \mathcal{O}(n)$. This could be quite expensive, unless n is very small.

Further, the above method is hampered by unbounded variables in the original set of constraints. For example the polyhedron $\{x, y, z|x + y + z \leq 1\}$ cannot be over-approximated by the above method as $c_{j,k}$ variables are returned as unbounded variables each time in the calls to $minExp$.

Other than the above result, as far as we know, this process of efficiently over-approximation of a general linear program using TVPI constraints is an open problem.

[13] One way of looking at the above step is that it fills in $c_{j,k}$ entries into a lower-diagonal matrix of size $n \times n$.

4.4 A New Schema using Schedule Space Under-Approximation

In this thesis, we propose that the Farkas system P be *Under-Approximated (UA)* to improve the scalability of the scheduling algorithm. This means that instead of searching for an optimal feasible point in P, we search in $P_a = \text{UA}(P)$. The above approximation is legal and only leads to a conservative approximation of losing schedules, though it is well proven [PBB$^+$11] that P is highly redundant with respect to schedule points. The overall process has to ensure that the approximation algorithm, as well as the solution finding time be scalable algorithms. We can restrict these requirements further and say that both of these algorithms should have *worst-case strongly polynomial time* running times matching the complexities of (U)TVPI sub-polyhedra introduced in Chapter 3 and summarized in Table 3.2 on page 30. The polynomiality is not a general restriction for the approximation algorithm, as one can separate the overall process of finding schedules into an offline and online process.

The above approximation can also be done on a per-dependence-edge basis. In this method, the per-dependence-edge Farkas polyhedra P_e are under-approximated, and the solution is found from the overall polyhedron obtained by putting together all the per-dependence UAs. Namely, by doing

$$P_a = \text{UA}(P) = \cap_{e \in E} \text{UA}(P_e)$$

If the above approximation leads to a non-empty polyhedron, then we can find schedules using $\text{UA}(P)$ instead of P.

In the above, we are exploiting the property that each of the individual P_e's of polyhedral compilation are guaranteed to be non-empty [Fea92a] directly as a result of dependence analysis. In polyhedral compilation, this property is true, as otherwise the statements corresponding to that particular dependence-edge could as well be removed from the program as a form of dead-code elimination. It also helps that each of the per-dependence-edge Farkas polyhedra P_e are comparably much smaller than the overall polyhedron. This leads to the following direct approximation method:

A direct approximation method using UA into (U)TVPI sub-polyhedra.

1. Approximate each P_e into a (U)TVPI-polyhedron $\text{UA}(P_e)$.
2. Build the complete constraint system by intersecting all the individual approximations: $\text{UA}(P) = \cap_{e \in E} \text{UA}(P_e)$.
3. Solve the system $\text{UA}(P)$ using appropriate solution technique: Hochbaum-Naor's technique [HN94] for TVPI polyhedra and Bellman-Ford for UTVPI polyhedra.

In Step 1 of the above method, as the individual constraint polyhedra ($\mathcal{P}_e$'s) could be quite small, we could even use high complexity techniques. Step 2 of the above method is exactly as suggested by Feautrier and involves collating the under-approximated sub-polyhedral systems. Step 3 of the above method uses the (U)TVPI solution methods, like Fourier-Motzkin pruning of Hochbaum-Naor to determine the feasibility of a TVPI linear systems and Bellman-Ford algorithm for UTVPI systems. Both these algorithms strongly polynomial time algorithms (with $\mathcal{O}(mn^2 \log m)$ and $\mathcal{O}(mn)$ worst-case complexity respectively).

We claim that a valid schedule can be determined using the above method in strongly polynomial time. It follows directly from the polynomial-time claims of under-approximation as well as the bounds of the solution techniques.

Here is an alternative method in which a (U)TVPI-UA can be obtained.

Indirect under-approximation using duality. To obtain an sub-polyhedral under-approximation of a rational polyhedron P, one can transform the polyhedron into the dual-space obtaining P^*, and then obtain the over-approximation of the dual-polyhedron $\text{OA}(P^*)$ and convert the over-approximated dual polyhedron into primal space ($\text{OA}(P^*) \rightarrow \text{UA}(P)$) to get an under-approximation $\text{UA}(\mathcal{P})$. In effect, we will be doing $P \rightarrow P^* \rightsquigarrow \text{OA}(P^*) \rightarrow \text{UA}(P)$. The proof of these methods[14] is trivial using a simple convexity argument on the polyhedra involved. This over-approximation[15] can run on a per-dependence edge basis as well.

For the above methods to work, each dual polyhedron P_e^* need be over-approximated by a min-cost-flow polyhedron, or even a generalized min-cost-flow polyhedron, as defined by Wayne [Way99]. This is because TVPI-polyhedra and min-cost-flow polyhedra are LP-duals of each other. The resultant over-approximation's dual polyhedron will be a TVPI polyhedron.

The above procedures make the under-approximation and over-approximation as asymptotically comparable methods, but for the transformation into dual-space. In case when the constraints/vertices of the input polyhedron are too many, then the cheaper direct method could be used.

4.5 Scheduling using Under-Approximations: An Example

We illustrate the direct under-approximation approach on Feautrier's one-dimensional scheduling [Fea92a, Example 1][16] which induces the following constraint system:

$$\begin{array}{rcrcrcrcr}
\mu_{2,1} & - & \mu_{2,2} & - & \mu_{1,1} & + & \mu_{1,2} & = & \lambda_{1,1} - \lambda_{1,2} \\
 & & & & \mu_{2,3} & - & \mu_{2,4} & = & \lambda_{1,3} - \lambda_{1,4} \\
 & & \mu_{2,2} & + & \mu_{2,4} & - & \mu_{1,2} & = & \lambda_{1,2} \\
 & & \mu_{2,0} & - & \mu_{1,0} & - & 1 & = & \lambda_{1,0}
\end{array}$$

After simplification by Gaussian elimination, followed by removal of λ-variables, the above becomes:

$$\begin{array}{lrcrcrcl}
C_1: & & & \mu_{2,0} & \geq & \mu_{1,0} & + & 1 \\
C_2: & \mu_{2,1} & + & \mu_{2,4} & \geq & \mu_{1,1} & & \\
C_3: & \mu_{2,2} & + & \mu_{2,4} & \geq & \mu_{1,2} & & \\
C_4: & & & \mu_{2,3} & \geq & \mu_{2,4} & + & 1
\end{array}$$

[14] These methods work only when $0 \in \mathcal{P}$. Otherwise, the polyhedra should be shifted so as to satisfy this condition. We will see in the next chapter how this complication can be avoided by using conical polarity.

[15] This over-approximation is *not* the same as the dependence over-approximation discussed in Section 4.2.1.1. Here, it is the dual of the Farkas constraint system that is being over-approximated, while there, it is the dependence polyhedron that is being over-approximated.

[16] The same example is also in Darte et al. [DRV00, Example 20, p. 220–222].

Let us denote the sub-systems as $\mathcal{P}_1 = \{C_1, C_2, C_3\}$ from dependence $\mathcal{D}_1$ and $\mathcal{P}_2 = \{C_4\}$ from dependence $\mathcal{D}_2$. It can be seen that the sub-systems $\mathcal{P}_1$ and $\mathcal{P}_2$ are not independent, sharing variable $\mu_{2,4}$ with each other. Feautrier suggests two schedules to the above system.

- Schedule 1: Arbitrarily choose $\mu_{1,0} = \mu_{1,1} = \mu_{1,2} = \mu_{2,1} = \mu_{2,2} = \mu_{2,4} = 0$ and $\mu_{2,0} = \mu_{2,3} = 1$ which leads to the schedules $\{\theta(S_1, i, N) = 0; \theta(S_2, i, j, N) = j + 1\}$.
- Schedule 2: This time, one can choose $\mu_{1,0} = \mu_{1,2} = \mu_{2,2} = \mu_{2,4} = 0$ and which leads to $\mu_{2,1} = \mu_{1,1} = 1$. The schedules are $\{\theta(S_1, i, N) = i; \theta(S_2, i, j, N) = i + j + 1\}$.

Let us now study three sub-polyhedra and the impact on the affine schedules, in an increasing order of expressiveness. All three systems are amenable to strongly polynomial algorithms to compute schedules. Though in this thesis, we do not use interval approximations, we begin with an interval under-approximation for the sake of exposition.

Interval under-approximation. Neither of the sub-systems $\mathcal{P}_1$ and $\mathcal{P}_2$ are Intervals. One can however under-approximate each of them so that the resultant sub-systems are interval sub-polyhedra. Here is one possibility among many others.

For $\mathcal{P}_2$, choose the under-approximation $\text{Interval}(\mathcal{P}_2) = \{\mu_{2,3} \geq 1, \mu_{2,4} = 0\}$ which is an interval. This leads to $\mathcal{P}_1$ having the system $\{\mu_{2,0} \geq \mu_{1,0} + 1, \mu_{2,1} \geq \mu_{1,1}, \mu_{2,2} \geq \mu_{1,2}\}$, which is still not an interval. One can then choose $\{\mu_{1,0} = 0, \mu_{1,1} = 0, \mu_{1,2} = 0\}$ leading to

$$\text{Interval}(\mathcal{P}_1) = \{\mu_{2,0} \geq 1, \mu_{2,1} \geq 0, \mu_{2,2} \geq 0\}$$

The overall system is $\{\mu_{2,3} \geq 1, \mu_{2,4} = 0, \mu_{2,0} \geq 1, \mu_{2,1} \geq 0, \mu_{2,2} \geq 0\}$. One solution of the system is $\{\mu_{2,3} = 1, \mu_{2,4} = 0, \mu_{2,0} = 1, \mu_{2,1} = 0, \mu_{2,2} = 0\}$, which results in schedules $\{\theta(S_1, i, N) = 0; \theta(S_2, i, j, N) = j + 1\}$, which is same as Schedule 1 by Feautrier[Fea92a].

UTVPI under-approximation. It can be observed that $\mathcal{P}_2$ is already a UTVPI. On the other hand, $\mathcal{P}_1$ is "mostly" a UTVPI with constraint C_1 already being a UTVPI needing no approximation, while C_2 and C_3 have to be under-approximated into UTVPI. To obtain the UTVPI under-approximation intuition, one can observe that $\mu_{2,4}$ is a shared variable between the non-UTVPI constraints C_2, C_3 and UTVPI constraint C_4. Since there is no other shared variable, we are free to choose any non-negative values to them, without effecting the others. The variable $\mu_{2,4}$ along with each of the pairs of variables $\{\mu_{2,1}, \mu_{1,1}\}$ and $\{\mu_{2,2}, \mu_{1,2}\}$, creates halfspaces passing though origin dividing their corresponding $3d$-space accordingly. One can simply set the value $\mu_{2,4} = 1$ leading to $\text{UA}(C_2) = \{\mu_{2,1} + 1 \geq \mu_{1,1}\}$ and $\text{UA}(C_3) = \{\mu_{2,2} + 1 \geq \mu_{1,2}\}$. Solving this system and choosing the other variables as $\mu_{1,0} = \mu_{1,1} = \mu_{1,2} = 1$ and $\mu_{2,0} = \mu_{2,1} = \mu_{2,2} = 0$ and leads to the schedules $\{\theta(S_1, i, N) = 1 + N; \theta(S_2, i, j, N) = j + N\}$.

TVPI under-approximation. Since $\mathcal{P}_2$ is already a UTVPI it is a TVPI as well, while $\mathcal{P}_1$ needs to be under-approximated into one. The above UTVPI under-approximations are valid as TVPI under-approximations well. This is true because each coefficients of constraints belong to $\{0, \pm 1\}$.

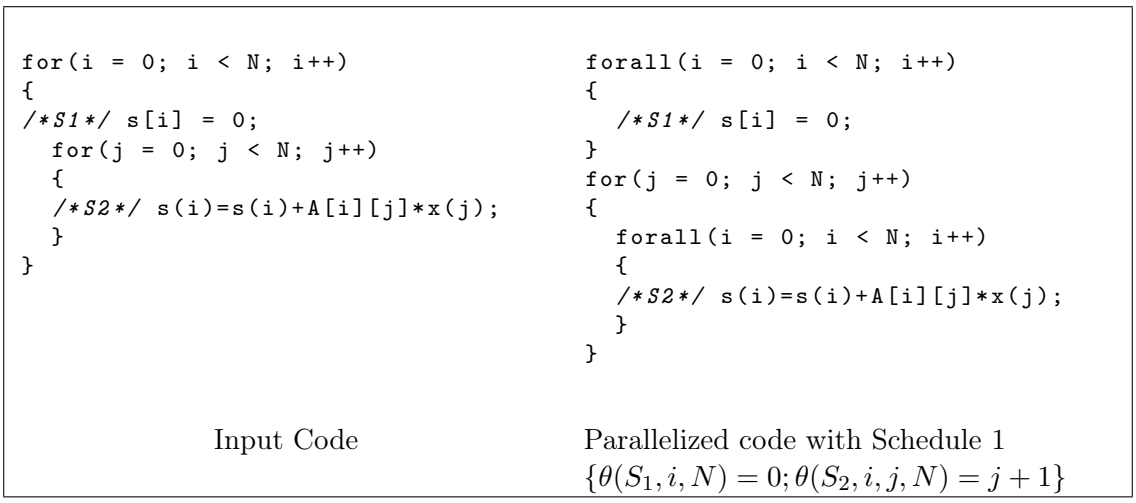


Figure 4.4: Under-Approximations and their code

The method presented in this section is to find the UA in a heuristic manner for illustration purposes. In the next chapters, we show how the above schedules can be found by an algorithm that runs in strongly polynomial time time.

4.6 A Summary of Related Work

In this section, we re-summarize the related work.

Dependence approximations. Yang et al. [YAI95], Darte et al. [DRV00, Chapter 5], as well as Darte-Vivien [DV97b] are excellent references on dependence abstractions. Our summarization in Section 4.2.1.1 is based on their papers. Similar to these dependence over-approximations is the approach of Balasundaram-Kennedy [BK89], summarized in Section 4.2.1.2, who used UTVPI polyhedra for data dependence analysis and domain simplification in the context of loop parallelization, namely task-level parallelism and task-level pipelining. The "zoo" of polyhedral dependence abstractions has been inspired by the corresponding one for abstract domains by Miné in his dissertation [Min04, Figure 2.9].

Creusillet in her dissertation on array region analysis [Cre96] suggested the limitations of under-approximations in a more general framework.

Polyhedral statistics. An implementation of Shostak [Sho81]'s *loop residue algorithm* has been done by Triolet as part of her dissertation [Tri84] in the PIPS4U project [PIP]. Also, Pugh [Pug92] as part of Omega test refers to the work of Maydan et al. [MHL91] who performed statistical analysis of dependence analysis constraints in Perfect Club Benchmarks as part of the SUIF project. Among the statistics is the reference to *loop residue algorithm* of Shostak [Sho81] for (U)TVPI constraints. The references indicate that a small percentage (86% of 9% of the total) can be handled using the algorithms of Shostak with a fixed cubic time asymptotic cost. Needless to say, the percentages are very small to be significant and in Chapter 7, we present statistics based on the PolyBench benchmarks that supersede these. Also, such

statistics are still on the dependence polyhedra, without any over-approximation algorithms for constraints or polyhedra that are not (U)TVPI.

Unscalability of affine scheduling. In the vast framework of affine scheduling literature, Feautrier's approach in his scalable and modular scheduling [Fea06] is the closest to our approach in Section 4.4 and is a proper predecessor. Feautrier himself builds from his earlier work on scheduling [Fea92a] by properly defining and addessing the scalability problem and suggests some methods by which it can be alleviated. The questions raised there as well as its positive influence on our work cannot be under-estimated. The new scheme that we introduced in Section 4.4 builds from his work, and we consider both contributions as complementary to each other. But, it should be noted that any future work that attempts to couple both these techniques should be aware of the (mutual) limitation that simplex based methods do not preserve the (U)TVPI-ness of the constraint matrix because of the way gaussian eliminations—the main engine of simplex algorithm—operate: a gaussian elimination algorithm on a TVPI matrix normally destroys its TVPI-ness.

Approximations of polyhedra. We are not aware of any previous algorithm for finding low-complexity approximations of general polyhedra into sub-polyhedra other than finding the interval ("box") polyhedral OAs. The literature is scarce about polyhedral approximations and more so for UAs. The naïve method of projecting out some dimensions using a method like Fourier-Motzkin gives an over-appoximation, and at a very high time complexity.

Miné [Min06, Section 4.3] adapts the vertex method for finding the UTVPI-OA of a general polyhedron. Simon et al. [SK10, Section 3.2.6] propose an iterative algorithm for the TVPI-OA of a general polyhedron using Linear Programming. We have summarized these algorithms briefly in Section 4.3. Vivien and Wicker [VW04] propose an algorithm to find the parallelepiped-OA of a 3d-polyhedron in vertex representation. Howe and King [HK09, Section 4.4] propose using a heuristic "projection based method" to approximate a finite set of Poly constraints into Logahedra constraints using TVPI as intermediate step. More details are not given.

Kanade et al. [KAI$^+$09] propose a heuristic method of finding the bounded vertex representation of a polyhedron by converting its representation to generator form, and throwing away some vertices, thereby obtaining an under-approximation. Colón and Sankaranarayanan [CS11, Section 5] as part of extending their earlier work on template polyhedra [SSM04, SSM05] also suggest using under-approximations, just like the above mentioned work of Kanade et al. [KAI$^+$09]. Both the above works primarily need over-approximations like many other works of static analysis, but algorithmically solve the under-approximation problem by convering into the polar space.

Kelly et al.'s work on transitive closure [KPRS96] also could be considered to be a polyhedral under-approximation of the transitive closure problem which is defined on general presburger arithmetic. This latter problem, though being undecidable in general, has a wide range of applications and many of them in parallelism, like determining the redundancy of inter-processor synchronizations.

4.7 Conclusions, Contributions and Perspectives

We review the conclusions and contributions of this chapter, followed by discussing some perspectives.

Conclusions. In this chapter, we started with the conclusion of previous chapers that a scalability problem exists in affine scheduling methods whose primary engine is solving linear programs which use simplex algorithm and searched for alternatives. We studied some past work on the well treaded path of dependence approximation in the spirit of in Yang et al. [YAI95], as well as by Darte et. al [DRV00]. We studied also some existing (U)TVPI (over-)approximation algorithms, ones that convert a general polyhedron into its over-approximations, and showed their limitations.

Contributions. A new schema of polyhedral scheduling, *"Sub-polyhedral scheduling using (U)TVPI under-approximations of Farkas polyhedra"* has been introduced in Section 4.4. It is a promising approach with better potential as it can build using the power of affine scheduling technique and the better worst-case complexities of the (U)TVPI polyhedra. Two alternative strategies have been proposed: under-approximating on a per-dependence basis; or running the under-approximation algorithm on the overall Farkas polyhedron. The first scheme gives rise to separation of a preprocessing step of under-approximation followed by the actual scheduling step. The preprocessing can be run in an initial phase and even on a per-dependence edge basis. The scheduling phase which is the main bottleneck because of the large-size of the LP problems whose solving causes scalability issues could instead use the Under-Approximated polyhedra, which are but intersection of the per-dependence edge approximations. To our best knowledge our work is the first to propose such a scheme. The following table refers to the complexities of the methods proposed in this chapter:

Polyhedra and Algorithm	Complexity to solve overall scheduling problem	
	Time	Memory
UTVPI with Bellman-Ford (Section 4.4)	$\mathcal{O}(mn) \approx \mathcal{O}(\|E\|\|V\|) \approx \mathcal{O}(\|V\|^3)$	$\mathcal{O}(m+n)$ Incidence matrix
TVPI with Hochbaum-Naor (Section 4.4)	$\mathcal{O}(mn^2 \log m) \approx \mathcal{O}(\|E\|\|V\|^2 \log \|E\|)$ $\approx \mathcal{O}(\|V\|^4 \log \|V\|)$	$\mathcal{O}(m+n)$ Incidence matrix
Simplex with fixed # of pivots (Section 4.2.3.2)	$\mathcal{O}(Kmn) \approx \mathcal{O}(mn) \approx \mathcal{O}(\|E\|\|V\|)$ $\approx \mathcal{O}(\|V\|^3)$	$\mathcal{O}(mn)$ Tableau
Convex Polyhedra with Simplex	$\mathcal{O}((m+n)mn) \approx \mathcal{O}(\|E\|^2\|V\|) \approx$ $\mathcal{O}(\|V\|^5)$	$\mathcal{O}(mn)$ Tableau

Table 4.1: Complexities of Affine Scheduling with Convex and Sub-polyhedra

Part of this chapter has been published in [UC11].

Perspectives. One initial goal of this work was to see if a new parallelization algorithm can be designed using (U)TVPI sub-polyhedra similar to existing dependence abstractions like DC, DDV etc, but for

multi-core architectures. The motivation of the above is the wide success of the FCO algorithm of Griebl et al. [GFG02] as implemented in PLuTo [BHRS08] with coarse-grain parallelization objectives. It still faces the same scalability problems as Feautrier's algorithm [Fea92a]. Such algorithm are similar to Wolf-Lam's algorithms [WL91a, WL91b], though using Balasundaram-Kennedy's simple sections abstraction [BK89], each of which were shown to be limited in scope.

Certainly, as pointed out by Yang et al. [YAI95], the previous dependence abstractions were tuned either for a particular kind of input programs, or for a particular kind of transformations. But, none of these were designed with scalability objectives as part of design considerations. Balasundaram-Kennedy's simple sections [BK89] comes under the same category, and though the authors hint at scalability as an objective it misses out the opportunities offered by affine scheduling. Hence, the study showed that re-adaptation of existing dependence methods by restricting the types of input is limited in scope[17].

Hence, a new scheme using (U)TVPI-UAs of Farkas polyhedra has been proposed in Section 4.4. Such a scheme is more powerful than the existing methods because of our ground in affine scheduling framework, and the use UA of Farkas polyhedra. But, we must also design scalable algorithms for constructing the (U)TVPI-UAs of general polyhedra. This is explored in the next two chapters, by first developing an under-approximation framework in Chapter 5 followed by proposing new under-approximation algorithms in Chapter 6.

[17] A possible open problem here is to design a parallelization algorithm similar to Lim and Lam's synchronization free techniques which uses affine transformations [LL97], though using (U)TVPI sub-polyhedra with a goal of acheiving scalability.

Chapter 5

A Framework for (U)TVPI Under-Approximation

5.1 Introduction

In this chapter, we define a simple and sound mathematical background to build (U)TVPI under-approximations of polyhedra, to linearize the problem of finding approximations, and to prove that the algorithms we construct in later chapters return valid approximations. To give these sufficient conditions, we take two approaches: an intuitive and geometric explanation using duality/polarity in Section 5.4, followed by a more direct way using the equivalent Farkas lemma in Section 5.5.

Starting with a polyhedron given in constraint form as $P = \{\mathbf{x} | A\mathbf{x} + \mathbf{b} \geq \mathbf{0}\}$, we show that simple under-approximations of P can be obtained by reasoning about over-approximations of the dual (polar), associated with the transpose matrix $[A \,|\, \mathbf{b}]^T$. To accomplish the above, in Section 5.2 and Section 5.3, we introduce some basic lemmas about convexity like homogenization and polarity. Next, in Section 5.4, we construct a simple framework for reasoning about under-approximations and over-approximations in a unified manner. Finally, in Section 5.5, we show how the above construction could be used to define a per-constraint approximation scheme which helps to obtain TVPI approximations of the non-TVPI constraints in the input polyhedron. The reader is referred to standard books for a complete coverage [Sch86, Zie06].

5.2 Some Background in Convexity

The overall goal of this section is to introduce some convexity concepts and relate them to TVPI under-approximations. Much of the math here is presented from standard books [Sch86, Zie06].

5.2.1 Homogenization

Homogenization can be done on Polyhedra in $\mathcal{H}$-form (constraint form) or in $\mathcal{V}$-form (vertex form or generator form). The following definition is when P is in $\mathcal{H}$-form.

Definition 5.2.1 [Homogenization]. Let $P = P(A, \mathbf{b})$ ($P = \{\mathbf{x}|A\mathbf{x} + \mathbf{b} \geq \mathbf{0}\}$) be a $\mathcal{H}$-polyhedron, then its homogenization is a cone and also a $\mathcal{H}$-polyhedron:

$$\text{homog}(P) = \left(\begin{pmatrix} A & \mathbf{b} \\ \mathbf{0}^T & 1 \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix} \right) = C(P) \tag{5.1}$$

A similar reverse operation dehomog can be defined for the reverse operation of de-homogenization.

It can be noted that if A is a $m \times n$-system (m constraints and n variables), and $\mathbf{b}$ is a $m \times 1$-vector, then the homogenized constraint system C (or homog(P)) is of size $(m+1) \times (n+1)$. Note that the constants dimension has become an additional dimension in the $(n+1)$-dimensional space. We would be referring to this dimension as *homogenizing* dimension and the other dimensions as *non-homogenizing* dimensions. Also note that homogenizing is a process that can be reversed by special marking of the dimensions. It should also be noted that homogenization is a rather trivial process, which can be constructed in linear time, and routinely done so in libraries like PIP/PolyLib. It however needs to be mentioned because this work deals with (U)TVPI constraints, which are defined to be having at most two non-zero coefficients in the non-homogenizing dimensions.

5.2.2 Polarity and conical polarity

The following is the definition of polar of a polyhedron.

Definition 5.2.2a [Polarity]. For $P \subseteq \mathbb{R}^d$, the polar set is defined by:

$$P^* = \left\{ \mathbf{c} \in (\mathbb{R}^d)^* : \mathbf{c}\mathbf{x} \leq 1 \text{ for all } \mathbf{x} \in P \right\} \subseteq (\mathbb{R}^d)^*$$

In standard books like Schrijver [Sch86], some of the theorems of polarity assume that the origin is an internal point of a polyhedron, namely that $\mathbf{0} \in \text{int}(P)$. For ease of notation, we use subscript o to denote that kind of pre-supposition on the polyhedra: P_o is a polyhedron such that $\mathbf{0} \in \text{int}(P)$. The polyhedron which satisfies this restriction can be expressed in $\mathcal{H}$-form as $P_o = \{\mathbf{x}|A\mathbf{x} + \mathbf{b} \geq \mathbf{0}; \mathbf{b} \geq \mathbf{0}\}$. Note the additional non-negativity restriction on $\mathbf{b}$.

A polyhedral cone C, however, because $\mathbf{0} \in C$, by definition.

Definition 5.2.2b [Polarity Properties: Polyhedra]. For $P_o^* \in \mathbb{R}^d$, whose vertices are columns of the matrix V and whose rays are the columns of the matrix Y, the following are equivalent:

$$P_o^* = \text{conv}(\mathbf{0}, V) + \text{cone}(Y) \iff P_o = \left\{ \mathbf{x} \in \mathbb{R}^n | V^T\mathbf{x} \leq \mathbf{1}; Y^T\mathbf{x} \leq \mathbf{0} \right\}$$

The above correspondences give rise to some well known polarity properties namely that: the vertices (respectively, edges, and facets) of the primal correspond to the facets (respectively, ridges, and vertices) of the polar. (A facet is a $(n-1)$-dimensional face.) Furthermore, the constraints of a polyhedron correspond to vertices/rays of the polar. We use these correspondences in the proofs of this dissertation.

From the above, we can easily derive the following definition of the polar of a cone C:

Definition 5.2.2c [Conical Polarity]. For $C \in \mathbb{R}^n$, the polar set is defined by

$$C^* = \{\mathbf{c} \in (\mathbb{R}^n)^* : \mathbf{cx} \leq 0 \text{ for all } \mathbf{x} \in C\} \subseteq (\mathbb{R}^n)^*$$

5.2.3 Polarity, (U)TVPI and (U)TCPV

A polyhedron can be represented in either of $\mathcal{H}$-form or $\mathcal{V}$-form. But, polyhedra that occur in polyhedral compilation are almost always described in $\mathcal{H}$-form. This means that the polar can be described, using a straightforward linear time construction, in $\mathcal{V}$-form. Converting the primal to $\mathcal{V}$-form or the polar to $\mathcal{H}$-form is however costly as it involves a call to Chernikova algorithm which has exponential time complexity. In the following lemma, we implicitly use the above *dual* interpretation, *without* making an actual call to Chernikova. This dual interpretation is different from the dual representation in libraries like PolyLib, where a polyhedron (in either primal or dual space) is described using both $\mathcal{H}$-form and $\mathcal{V}$-form.

Note that in the above lemma, as we are using $\mathcal{H}$-form (U)TVPI in primal, it implies that by a straightforward construction, the polar is in $\mathcal{V}$-form.

Lemma 5.2.3 [(U)TVPI and (U)TCPV]. For a TVPI-polyhedron, the polar has vertices and rays which have not more than two non-zero components in the non-homogenizing dimensions. For a UTVPI-polyhedron, the polar has vertices and rays which have no more than two non-zero components each in the non-homogenizing dimensions. The components belong to $\{-1, +1\}$. We define the polar of a (U)TVPI constraint as *(Unit-)Two-Components-Per-Vector* or *(U)TCPV* vector. Further, we define the polar of a (U)TVPI polyhedron as a *(U)TCPV polyhedron*.

5.3 Polarity and Approximations

5.3.1 Polar of polar

The proofs of the following can be found in Schrijver [Sch86] and hence are omitted.

Lemma 5.3.1a [Polar of Polar and Equality]. (i) $(P_o^*)^* = P_o$, (ii) $(P^*)^* = P \cup \{\mathbf{0}\}$, and (iii) $(C^*)^* = C$.

The proof of the following is straightforward (almost obvious), but we do it for completeness.

Lemma 5.3.1b [Polarity and Inclusivity]. For any two polytopes P_o and Q_o, $P_o \subseteq Q_o \Leftrightarrow P_o^* \supseteq Q_o^*$.

Proof:

- To prove this, assume that P_o is expressed in $\mathcal{V}$-form and Q_o in $\mathcal{H}$-form. Let $P_o = \text{conv}(\mathbf{0}, V_P)$ and let $\mathbf{c} \in V_P$ be an arbitrary vertex of P_o. Further, let $Q_o = \{\mathbf{x} \in \mathbb{R}^n | V_{Q_o^*}^T \mathbf{x} \leq \mathbf{1}\}$ (this definition of Q_o in $\mathcal{H}$-form is possible because of only-if part of Definition 5.2.2b). Because of the inclusivity in the if-direction $P_o \subseteq Q_o$, we have that $V_{Q_o^*}^T \mathbf{c} \leq \mathbf{1}$ is true. Comparing this latter with the definition of Polar of P_o, $P_o^* = \{\mathbf{c} \in (\mathbb{R}^d)^* : \mathbf{cx} \leq 1 \text{ for all } \mathbf{x} \in P_o\} \subseteq (\mathbb{R}^d)^*$, by Definition 5.2.2a), we infer that

all the vertices of Q_o^* are enclosed by an arbitrary inequality of P_o^*. Since the same method can be iterated for any of the vertices of P_o, we have that $P_o^* \supseteq Q_o^*$.

- This proof part is similar to the "if part", but assuming that P_o^* is expressed in $\mathcal{V}$-form and Q_o^* in $\mathcal{H}$-form.

The above lemmas can obviously be extended for any general polyhedra and for cones as well.

Lemma 5.3.1c [Polarity and Inclusivity]. For any two cones K_1 and K_2, $K_1 \subseteq K_2 \Leftrightarrow K_1^* \supseteq K_2^*$.

5.3.2 Under-Approximation and Over-Approximation

The following corollary can be derived from the above lemmas.

Corollary 5.3.2 [Toggling between OA and UA]. (i) $(\text{OA}(P_o^*))^* \subseteq P_o$, and (ii) $(\text{OA}(C^*))^* \subseteq C$

The above corollary means that by taking the polar of a polyhedron and taking the resultant's Over-Approximation (OA) one obtains a polyhedron whose polar is an UA of the original polyhedron. Hence, the first half above corollary can be used to find the UA of a polyhedron by converting into polar space and taking the OA of its polar polyhedron.

5.4 A Construction for Under-Approximation

Using the theorems of the previous section, one can reason about UA by talking about OA in the dual/polar space. But, using OA to find valid UA using duality-properties, means that one has to deal with the many cases of the polyhedron being bounded or not, whether it is homogenized or not (like in Lemmas 5.3.1), and whether origin is an internal point or not (like in Corollary 5.3.2.i). Further, the Farkas constraint polyhedra (even the per-dependence ones) that are obtained in polyhedral analysis do not necessarily have the origin as an internal point. Also, they are never bounded. So, we need a single framework to deal with all these cases.

In this section, we create a simple construction, using which the problem of finding a UA of a polyhedron in $\mathcal{H}$-form is reduced to the problem of finding a OA of its polar cone in $\mathcal{V}$-form. The construction uses homogenization and conical polarity and handles all the above cases in a unified and simple manner. The construction is the mathematical framework that gives us the basis to prove that the UAs we later construct in Chapter 6 are valid approximations.

The super-scripts $\mathcal{H}$ and $\mathcal{V}$ show whether the particular polyhedron is in constraint or generator form respectively.

$$P^{\mathcal{H}} \xrightarrow{\text{homog}} C^{\mathcal{H}} \xrightarrow{\text{polar}} K^{\mathcal{V}} \xrightarrow{\text{OA}} K_a^{\mathcal{V}} \xrightarrow{\text{depolar}} C_a^{\mathcal{H}} \xrightarrow{\text{dehomog}} P_a^{\mathcal{H}}$$

The following is some notes on the above construction:

- [Step 0] $P^{\mathcal{H}}$: Input P, the Polyhedron to be under-approximated in $\mathcal{H}$-form.

- [Step 1] $P^{\mathcal{H}} \xrightarrow{\text{homog}} C^{\mathcal{H}}$: Compute Cone C by homogenizing P. Namely, $C = \text{Homog}(P)$. C is a cone in $\mathcal{H}$-form.
- [Step 2] $C^{\mathcal{H}} \xrightarrow{\text{polar}} K^{\mathcal{V}}$: Compute $K = C^*$, the Polar cone of the cone C. K is a cone in $\mathcal{V}$-form.
- [Step 3] $K^{\mathcal{V}} \xrightarrow{\text{OA}} K_a^{\mathcal{V}}$: Compute K_a, the Over-Approximation of K such that $K \subseteq K_a$. K_a is a cone in $\mathcal{V}$-form.
- [Step 4] $K_a^{\mathcal{V}} \xrightarrow{\text{depolar}} C_a^{\mathcal{H}}$: Take the polar cone (DePolar) of K_a to result in C_a, which will be a cone in $\mathcal{H}$-form.
- [Step 5] $C_a^{\mathcal{H}} \xrightarrow{\text{dehomog}} P_a^{\mathcal{H}}$: Dehomogenize[18] C_a to result in P_a, which will be a Polyhedron in $\mathcal{H}$-form.

In Step 0, P could be a general polyhedron in $\mathcal{H}$-form, possibly in non-minimal form (having redundant constraints). Step 1 follows from Definition 5.2.1. Here, the dimensions have to be marked as *homogenizing* and *non-homogenizing* dimensions so that it can help in approximation (Step 3), as well as in the dehomogenization (Step 5) later. No new homogenization is needed for performing this step other than what is performed in standard libraries. As mentioned above, this is needed in particular because the subject of this dissertation is (U)TVPI systems, which by definition cannot directly operate on cones without this special marking. Step 2 follows from Definition 5.2.2.

In the approximation process in Step 3, the Over-Approximation K_a has to satisfy (U)TCPV properties, so as to respectively obtain the (U)TVPI approximation of P. More particularly, for K_a to be a TCPV over-approximation of K, it should have not more than two non-zero components in the non-homogenizing dimensions. Further, for K_a to be a UTCPV over-approximation of K, K_a should be a TCPV vector and the non-zero components should be from $\{-1, +1\}$ (Lemma 5.2.3). In the approximation, one can neither throw away the offending dimensions, nor can throw away the offending vectors themselves (ones that are not (U)TCPV), as the remaining vectors may result in an under-approximation of P. If K is being approximated on a per-constraint basis, the OA process could be such that K_a has multiple vectors for each vector of K. The result of Step 3 is $K_a = \text{OA}(K)$. Steps 4 and 5 are reverse of Steps 2 and 1 respectively.

In the above, P_a (respectively C_a and K_a) are approximations of P (respectively C and K). More particularly, we have the following theorem:

Theorem 5.4 [Approximations and the UA-OA Construction]. Let P be a polyhedron defined in $\mathcal{H}$-form. Let $C = \text{Homog}(P)$, $K = C^*$, $K_a = \text{OA}(K)$, $C_a = K_a^*$, and $P_a = \text{DeHomog}(C_a)$. Then, we have: (i) $K_a \supseteq K$, (ii) $C_a \subseteq C$ and (iii) $P_a \subseteq P$.

Proof: In (i) $K_a = \text{OA}(K)$ by definition. In (ii), $C_a = \text{UA}(C)$, because of Corollary 5.3.2.ii. Finally, in (iii) $P_a = \text{UA}(P)$ because of Corollary 5.3.2.i.

[18] We refer dehomogenizing as the reverse of Definition 5.2.1 (Equation 5.1 on page 60).

The above construction algorithm can be done in linear time and can be considered to be a reinterpretation of the input polyhedron in $\mathcal{H}-$form so that the validity of the UA can be proved, and simple scalable algorithms could be obtained for (U)TVPI approximation. In the above algorithm, using homogenization requires the origin to be an internal point. And, using polarity reduces the UA problem to the OA problem. But, one cannot swap the two steps of homogenization and polarity (swapping Steps 1 and 2 and following it by swapping Steps 4 and 5), for example, by taking the polar of P and then taking the resultant's homogenization. Doing so means that one has to "chop away" the origin from the polar of the resultant's OA as in Lemma 5.3.1a.ii. Such a process involves the generator representation and is costly.

Observe that K in the above construction is in $\mathcal{V}$-form. Its generator vectors are columns of the following matrix:

$$K = \text{cone} \begin{pmatrix} A^T & \mathbf{0} \\ \mathbf{b}^T & 1 \end{pmatrix} \tag{5.2}$$

In matricial form, K is of size $(n+1) \times (m+1)$. K could thus be seen to be a transpose of Equation 5.1 on page 60. K can also be written as the following:

$$K = \text{cone} \left\{ \begin{pmatrix} \mathbf{a}_1^T \\ b_1 \end{pmatrix}, \ldots, \begin{pmatrix} \mathbf{a}_j^T \\ b_j \end{pmatrix}, \ldots, \begin{pmatrix} \mathbf{a}_m^T \\ b_m \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix} \right\} \tag{5.3}$$

The objective of Step 3 remains to find the TCPV-OA of Polyhedron K given in $\mathcal{V}$-form. This problem can be seen as trying to find a cone K_a such that:

$$K \subseteq \text{cone}\,(K_a) \tag{5.4}$$

By this Equation 5.4, if each column of K can be written as a conical sum of the vectors in K_a, then the approximation remains a valid approximation. If each vector in K_a is a TCPV vector, then the corresponding P_a would be a TVPI approximation of P.

The alternate to the above scheme is to use conical polarity.

A simplification using conical polarity. From the definitions given in previous section, K, the polar cone corresponding to P can directly be constructed from homog(P) in Equation 5.1 on page 60, and its generator vectors are columns of the following matrix:

$$K = \text{cone} \begin{pmatrix} A^T & \mathbf{0} \\ \mathbf{b}^T & 1 \end{pmatrix}$$

We would still need the $\begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix}$ augmentation because of preservation of lineality space of the primal in the polar. From the above, if we let P_a (respectively K_a) be the approximation of P (respectively K), we have the following equivalence:

$$K_a \supseteq K \Leftrightarrow P_a \subseteq P \tag{5.5}$$

From which follows Equation 5.4 and the explanation therein.

5.5 Approximation Scheme for TVPI-UA using TCPV-OA

In this section, we formulate the sufficient conditions to prove that the UAs we later construct in Chapter 6 are valid approximations. In fact, with the Farkas lemma in Section 2.2 on page 9 and the equivalent homogenization-polarity construction in Section 5.4, we do have all the necessary ammunition to construct an approximation scheme which allows us to built TVPI-UAs of general polyhedra. UTVPI-UAs are simple extensions of the material in this section.

In the rest of this section, for ease of exposition, we assume that the polyhedron P is in 3-dimensions ($n = 3$)[19]. The latter polyhedra, also called as 3VPI polyhedra, are general enough to represent any arbitrary polyhedron. For polyhedra that are not 3VPI, a reduction exists that transforms any arbitrary polyhedron into its equivalent 3VPI polyhedron.

According to Shostak [Sho81], the transformation from general convex polyhedra to 3VPI polyhedra is said to have been suggested by R.Tarjan and is very similar to the reduction of an arbitrary n-variable boolean satisfiability ("SAT") problem to a 3SAT problem. This transformation is not an approximation, which is the subject of this chapter. Here is an example to illustrate the reduction.

Example 5.5 [Reduction from convex polyhedra to 3VPI]. The constraint $w + x + y + z \leq 1$ is obviously not 3VPI. It can be seen to be equivalent to the system $\{w + x \leq v; w + x \geq v; v + y + z \leq 1\}$ which is 3VPI. □

Note that the UA algorithms we present do not need to this reduction. We use it only to show equivalence.

Let the j-th column of K, with $1 \leq j \leq m$, be $K^j = \begin{pmatrix} a_{j,1} & a_{j,2} & a_{j,3} & b_j \end{pmatrix}^T$. Then we have

$$K^j = \begin{pmatrix} a_{j,1} \\ a_{j,2} \\ a_{j,3} \\ b_j \end{pmatrix} \in \text{cone} \begin{pmatrix} t_1^{j,1} & t_2^{j,1} & 0 \\ t_1^{j,2} & 0 & t_3^{j,2} \\ 0 & t_2^{j,3} & t_3^{j,3} \\ p_1^j & p_2^j & p_3^j \end{pmatrix} = \text{cone}(T^j) \tag{5.6}$$

It can be noticed that each column of the matrix T^j is a TCPV vector and hence when it is subjected to the reverse transformation of Steps 4 and 5 as discussed in Section 5.4, yields a set of TVPI constraints. The above is what we call as an *approximation scheme*. In this scheme,

Definition 5.5 [Approximation scheme]. A non-TVPI constraint $a_{j,1}x_1 + a_{j,2}x_2 + a_{j,3}x_3 + b_j \geq 0$ in the original system is replaced by the set of constraints $\text{UA}(x_1, x_2, x_3) = \{t_1^{j,1}x_1 + t_1^{j,2}x_2 + p_1^j \geq 0; t_2^{j,1}x_1 + t_2^{j,3}x_3 + p_2^j \geq 0; t_3^{j,2}x_2 + t_3^{j,3}x_3 + p_3^j \geq 0\}$.

[19]It should be clarified that the n here refers neither to the number of elements in the constraint, nor to the number of elements in the polar vector. It just refers to the dimension of the primal. In such a case, the number of significant elements in the constraint and dual-vectors is $n + 1$.

In the above approximation scheme, every column vector of K which has more than two non-zero components is replaced by a set of TCPV vectors, such that the original vector remains in the conical sum of the replacements. The conical combination of the replacement vectors is an OA of the original vector K^j. The above scheme remains valid as long as the (t, p) variables and the (a, b) constants satisfy the convexity requirement. One way for ensuring the same is by making the (t, p)-variables satisfy the following additional constraints, which we would be referring to as *context constraints* for reasons that will be exposed later:

$$\begin{pmatrix} t_1^{j,1} & +t_2^{j,1} & \\ t_1^{j,2} & & +t_3^{j,2} \\ & t_2^{j,3} & +t_3^{j,3} \\ p_1^j & +p_2^j & +p_3^j \end{pmatrix} = \begin{pmatrix} a_{j,1} \\ a_{j,2} \\ a_{j,3} \\ b_j \end{pmatrix} \tag{5.7}$$

When such a set of T_j matrices is defined for each vector K_j of K, then, we have $K_a = \text{cone}\left\{T^1, T^2, \ldots, T^m\right\}$ and the resultant TCPV approximation would be $K \in \text{cone}\left\{T^1, T^2, \ldots, T^m\right\}$. With the above, we give the following theorem, whose proof builds on Theorem 5.4 and the arguments given in this section.

Theorem 5.5 [TVPI and UA]. Let P_a be the polar of the K_a vectors obtained in Equation 5.7. Then, P_a is a valid TVPI-UA of the original Polyhedron P.

Proof: We employ the affine form of Farkas lemma, with the premise of non-emptiness of P guaranteed because of the per-constraint method. For the UA to be proper, the affine form corresponding to the original constraint $\mathbf{a}_j\mathbf{x} + b_j$ should be expressible as a positive sum of the replacement TVPI vectors

$$\begin{aligned} &a_{j,1}x_1 + a_{j,2}x_2 + a_{j,3}x_3 + b_j \\ &\qquad \equiv \lambda_0^j + \lambda_1^j(t_1^{j,1}x_1 + t_1^{j,2}x_2 + p_1^j) + \lambda_2^j(t_2^{j,1}x_1 + t_2^{j,3}x_3 + p_2^j) + \lambda_3^j(t_3^{j,2}x_2 + t_3^{j,3}x_3 + p_3^j) \end{aligned}$$

where the λ-multipliers satisfy $\lambda_0^j, \lambda_1^j, \lambda_2^j, \lambda_3^j \geq 0$. Pairwise matching for each of the x-variables yields:

$$\begin{pmatrix} a_{j,1} \\ a_{j,2} \\ a_{j,3} \\ b_j \end{pmatrix} = \begin{pmatrix} & \lambda_1^j t_1^{j,1} & +\lambda_2^j t_2^{j,1} & \\ & \lambda_1^j t_1^{j,2} & & +\lambda_3^j t_3^{j,2} \\ & & \lambda_2^j t_2^{j,3} & +\lambda_3^j t_3^{j,3} \\ \lambda_0^j & +\lambda_1 p_1^j & +\lambda_2^j p_2^j & +\lambda_3^j p_3^j \end{pmatrix} \tag{5.8}$$

Setting $\lambda_1^j = \lambda_2^j = \lambda_3^j = 1$ yields the context constraints defined in Equation 5.7 proving the validity of the approximation, except for the additional λ_0^j in the homogenizing-dimension-matching. Setting $\lambda_0^j = 0$ is safe, and we will see later that it actually contributes to the quality of the approximation. ■

The problem remains to find such a set of (t, p) variables satisfying the above context constraints Equation 5.7, so that the approximation remains valid. It can be seen that searching for the (t, p) variables directly could lead to a non-linear (quadratic) formulation. Even searching for t-variables that satisfy the above constraints turns out to be non-linear. But, as they are existentially quantified, the equations be

solved using advanced quantifier elimination techniques like [Tar51], which are well known to be unscalable beyond small inputs and certainly not polynomial.

In the next chapter however, we propose some linearizations of the above mentioned approximation scheme, so that the approximation algorithms remain scalable. This is done first as a heuristic where both (t, p)-variables are arbitrarily fixed, and then as a more formal method where only the t-variables are fixed, while the p-variables are found by an LP formulation.

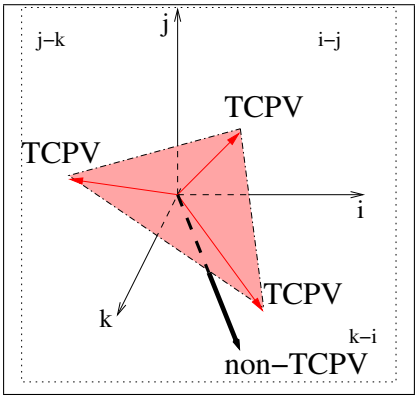


Figure 5.1: An illustration of TCPV approximation of a non-TCPV general vector

5.6 Conclusions

In this chapter, we use a simple framework based on polarity and showed that it can be used for designing the Under-Approximation algorithms. We also studied a direct application of Farkas lemma as an alternate to the framework, which achieved the same very concisely[20]. A more detailed presentation of this material can be found in [UC11, UC12]. An illustration of the duality construction is shown in Figure 5.1.

Three equivalent alternatives: This chapter showed three alternate ways of obtaining under-approximations through the use of duality: by the use of a homogenization-polarity construction, use of conical polarity and Farkas lemma. Each of these are equally powerful to the other and show that there are many alternate ways of looking at our scheme and proving its validity.

This chapter is closely tied with the algorithms presented in the next chapter. So, more detailed conclusions and perspectives are offered after covering the algorithms.

[20]We thank Paul Feautrier and Armin Größlinger for suggesting this proof simplification.

Chapter 6

Polynomial Time (U)TVPI Under-Approximation Algorithms

6.1 Introduction

In this chapter, we use the framework developed in Chapter 5 and develop worst-case polynomial time algorithms for obtaining (U)TVPI under-approximations of polyhedra. Firstly, in Section 6.2, we give a strongly polynomial time per-constraint algorithm for TVPI-UA. Then, in Section 6.3, we give an LP based per-constraint weakly polynomial time algorithm for TVPI-UA. In Section 6.4 we propose various ways in which the above LP based algorithm can be applied when multiple non-TVPI constraints are present. In Section 6.5, we give a strongly polynomial algorithm that gives a UTVPI-UA of a TVPI polyhedron. In Section 6.6, we briefly sketch a parametrized UTVPI-UA approximation method. In Section 6.7, we discuss the metrics and complexity of the algorithms. Finally, in Section 6.8, we conclude with a summary of the contributions, and discuss some perspectives.

6.2 The Median method for TVPI-UA

In this section, we introduce a simple per-constraint heuristic using the framework developed in Sections 5.4 and 5.5. The main idea of this approximation is Equation 5.4 on page 64, which states that the original vector can be approximated by any set of the replacement TCPV vectors, as long as the former remains in the cone of the latter.

Definition 6.2 [Median method: 3-d case]. The inequality $ax+by+cz+1 \geq 0$ can be approximated by the set of inequalities $\{ax + by + \frac{2}{3} \geq 0; ax + cz + \frac{2}{3} \geq 0; by + cz + \frac{2}{3} \geq 0;\}$.

Geometric intuition for the above derives from the observation of the polar space, where the above

approximation is the following:

$$\begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix} \in \text{cone} \begin{pmatrix} \frac{1}{2}a & \frac{1}{2}a & 0 \\ \frac{1}{2}b & 0 & \frac{1}{2}b \\ 0 & \frac{1}{2}c & \frac{1}{2}c \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

The intuition for the above approximation comes from Equation 5.6 on page 65. In that equation, the values of t-variables and p-variables satisfy Equation 5.7 on page 66; such assignments have to be found out to obtain an approximation. As explained in Chapter 5, the t-variables have to be determined by a choice so that linearity is satisfied. The method in this section fixes the p variables also in a heuristic manner by dividing the available "budget" in the homogenizing dimensions equally between the values in the homogenizing dimensions of the replacement TCPV vectors. This is not the only way for obtaining an approximation. It is one of the many ways, which is intuitive and builds the platform for more powerful under-approximations that we will see in later sections. We call this the median method because the original vector is the median of the replacement vectors in the polar space.

4-d case. For illustration purposes, the approximation is the following for 4d case

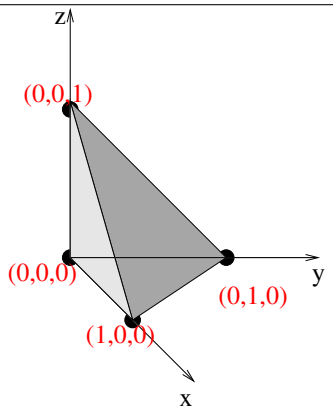
$$\begin{bmatrix} a \\ b \\ c \\ d \\ 1 \end{bmatrix} \in \text{cone} \begin{pmatrix} \frac{1}{3}a & \frac{1}{3}a & \frac{1}{3}a & 0 & 0 & 0 \\ \frac{1}{3}b & 0 & 0 & \frac{1}{3}b & \frac{1}{3}b & 0 \\ 0 & \frac{1}{3}c & 0 & \frac{1}{3}c & 0 & \frac{1}{3}c \\ 0 & 0 & \frac{1}{3}d & 0 & \frac{1}{3}d & \frac{1}{3}d \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{pmatrix}$$

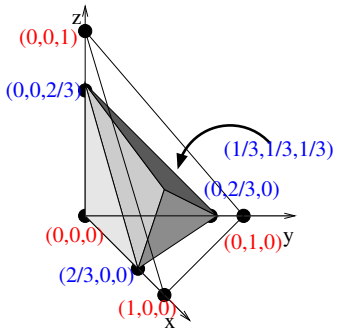
General n-d case. The above can be easily be generalized to n-d polyhedra. Let s be the sparsity of a polyhedron, i.e., the number of non-zero variables (outside the homogenizing dimension) for a specific constraint, with $1 \leq s \leq n$. Let $q = \binom{s}{2} = s(s-1)/2$ and let $r = s - 1$. The resultant set of TCPV vectors corresponding to the particular constraint are $n+1$ dimensional, with cardinality q. The coefficients of the non-homogenizing dimensions are divided by r, while the homogenizing dimension is uniformly divided by q.

In each of the cases, it can be seen that the approximation being proposed is a TCPV approximation, making its polar a TVPI approximation. It can also be verified using the construction given in the earlier sections and Equations 5.4 and 5.6 that the UA proposed in each case is a valid approximation.

Example 6.2 [Median method on pyramid]. Let the input system be the triangular pyramid (or 3d-simplex or Trirectangular Tetrahedron)

$$\{x \geq 0;\ y \geq 0;\ z \geq 0;\ x + y + z \leq 1\}$$





$$\begin{pmatrix} -x & & \\ & -y & \\ & & -z \\ x & +y & +z \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} -x & & \\ & -y & \\ & & -z \\ x & +y & \\ & y & +z \\ x & & +z \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ \frac{2}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{pmatrix}$$

3d-simplex (triangular pyramid)

An under-approximation by median method

Figure 6.1: Median Method TVPI-UA for a 3d-simplex

Only the inequality $x + y + z \leq 1$ is not TVPI and is approximated by the set of inequalities

$$\{x + y \leq \frac{2}{3}; x + z \leq \frac{2}{3}; y + z \leq \frac{2}{3}\}$$

It can be seen that the approximation is a non-empty TVPI system (it is a UTVPI system), with vertices

$$\{(0,0,0), (\frac{2}{3},0,0), (0,\frac{2}{3},0), (0,0,\frac{2}{3}), (\frac{1}{3},\frac{1}{3},\frac{1}{3})\}$$

each of which are inside the vertices of the original system $\{(0,0,0), (1,0,0), (0,1,0), (0,0,1)\}$. The above approximation is shown in Figure 6.1. The above example is a modified form of a "classic" Farkas system [DRV00, Fea92a] induced in the compilation of the matrix-vector-product: $\mathtt{s[i]} += \mathtt{A} * \mathtt{x[i]}$, and as shown in the worked out example in Section 4.5 on page 53 in this thesis. □

With reference to the choice of Farkas multiplier λ_0 in Theorem 5.5, it can be seen that setting $\lambda_0 = 0$ in the above example has the advantage that the three replacement TVPI hyperplanes intersect exactly on the original non-TVPI hyperplane. Any other strictly positive choice for λ_0 would mean that the point of intersection would lie strictly inside the positive half-space of the original and thus giving rise to an UA which is in a way less effective.

Example 6.2 [Median method on skewed pyramid]. Let the input system be the skewed triangular pyramid

$$\{x, y, z \geq 0; 1000x + 100y + 10z \leq 1\}$$

Only the inequality $1000x + 100y + 10z \leq 1$ is not TVPI and is approximated by the set of inequalities

$$\left\{1000x + 100y \leq \frac{2}{3}; 1000x + 10z \leq \frac{2}{3}; 100y + 10z \leq \frac{2}{3}\right\}$$

It can be seen that the resultant approximation is a non-empty TVPI system with vertices:

$$\left\{(0,0,0), \left(\frac{1}{1500}, 0, 0\right), \left(0, \frac{1}{150}, 0\right), \left(0, 0, \frac{1}{15}\right), \left(\frac{1}{3000}, \frac{10}{3000}, \frac{100}{3000}\right)\right\}$$

each of which are inside the vertices of the original system, which are

$$\left\{(0,0,0), \left(\frac{1}{1000}, 0, 0\right), \left(0, 0, \frac{1}{10}\right), \left(0, \frac{1}{100}, 0\right)\right\}$$

□

It can be seen that the median method is simple and easy to implement, but does not have any guarantee of ensuring that the resultant approximation is non-empty. In the next section, we will generalize this method to formulate a parametrized approximation and formulate an LP problem to find an approximation, which satisfies the properties given in Equation 5.7 on page 66.

6.3 LP-based Parametrized TVPI approximation

To ease the exposition, we primarily deal with 3-dimensional polyhedra again; higher dimensional extensions are straightforward.

In Section 6.2, we saw a per-constraint approximation scheme in which the system $\mathcal{S}(\mathbf{x}) = \{\mathbf{x}|\mathbf{a}_1\mathbf{x} \geq b_1; \ldots; \mathbf{a}_j\mathbf{x} \geq b_j; \ldots; \mathbf{a}_m\mathbf{x} \geq b_m;\}$, with $\mathbf{a}_j\mathbf{x} \geq b_j$, a non-TVPI constraint is approximated by the system $\mathcal{S}_1(\mathbf{x}) = \{\mathbf{x}|\mathbf{a}_1\mathbf{x} \geq b_1; \ldots; \mathrm{UA}_1(\mathbf{a}_j\mathbf{x} \geq b_j); \ldots; \mathbf{a}_m\mathbf{x} \geq b_m;\}$, with $\mathrm{UA}_1(\mathbf{a}_j\mathbf{x} \geq b_j)$ defined as the following: $\mathrm{UA}_1(\mathbf{a}_j\mathbf{x} \geq b_j) = \{\mathbf{x}|a_{j,1}x_1 + a_{j,2}x_2 \geq \frac{2}{3}b_j; a_{j,1}x_1 + a_{j,3}x_3 \geq \frac{2}{3}b_j; a_{j,2}x_2 + a_{j,3}x_3 \geq \frac{2}{3}b_j;\}$. As seen earlier, the above median method is simple, but does not have any guarantee of ensuring that $\mathcal{S}_1$ is non-empty.

The median method can easily be extended by defining the approximation as a *parametrization* on the values in the homogenizing dimension entries: as $\mathrm{UA}_2(\mathbf{a}_j\mathbf{x} \geq b_j) = \{(\mathbf{x}, \mathbf{p}^j)|a_{j,1}x_1 + a_{j,2}x_2 \geq 2p_1^j; a_{j,1}x_1 + a_{j,3}x_3 \geq 2p_2^j; a_{j,2}x_2 + a_{j,3}x_3 \geq 2p_3^j; \sum \mathbf{p}^j = b_j;\}$ where the values of the 3-dimensional $\mathbf{p}^j$-vector are unknown and have to be found out.

In the above approximation, it can be observed that the coefficients in the non-homogenizing dimensions (t-variable values) are fixed, as in the median method. But, the coefficients in the homogenized dimension (p-variable values) are unknown and have to be decided. The context constraint $\sum \mathbf{p}^j = p_1^j + p_2^j + p_3^j = b_j$ is not arbitrary. It is determined by the choice of the multipliers for the t-variables so that the K^j vector is in the convex-sum of T^j, as given in Equation 5.6.

The resultant system is $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j) = \{(\mathbf{x}, \mathbf{p}^j) | \mathbf{a}_1\mathbf{x} \geq b_1; \ldots; \mathbf{a}_{j-1}\mathbf{x} \geq b_{j-1}; \mathrm{UA}_2(\mathbf{a}_j\mathbf{x} \geq b_j); \mathbf{a}_{j+1}\mathbf{x} \geq b_{j+1}; \ldots; \mathbf{a}_m\mathbf{x} \geq b_m;\}$. In the discussion below, we show that the Higher Dimensional (HD) system $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ can be interpreted in two ways, a geometric and an algorithmic ways, each with its own merits.

6.3.1 A parametrized approximation

$\mathcal{S}_{HD}$ can geometrically be considered as a parametrized approximation, with $\mathbf{p}^j$ being the parametric vector and the context constraint $\sum \mathbf{p}^j = b_j$ considered as the parametric context. When the values of the vector $\mathbf{p}^j$ are known, then the system:

$$\mathcal{S}_2(\mathbf{x}) = \mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j) | \mathbf{p}^j \tag{6.1}$$

is a non-parametrized LP problem, which can be tested for feasibility in the usual $\mathbf{x}$-variables.

The meaning of the above equation is the following: Let us assume that the rational polyhedron represented by the system $\mathcal{S}(\mathbf{x})$ of size $m \times n$ is non-empty, and j is an arbitrary non-TVPI constraint of $\mathcal{S}(\mathbf{x})$ with $1 \leq j \leq m$; there exist a pair $\left\langle \mathbf{p}^j , \mathcal{S}^j_{HD}(\mathbf{x}, \mathbf{p}^j) \right\rangle$, where $\mathbf{p}^j$ is a rational vector and $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ is a higher dimensional system such that the polyhedron represented by the system $\mathcal{S}_2(\mathbf{x}) = \left\{ \exists \mathbf{p}^j \mid \mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j) \wedge \sum \mathbf{p}^j = b_j \right\}$ is a valid under-approximation of the polyhedron of $\mathcal{S}(\mathbf{x})$.

Since the context constraint $\sum \mathbf{p}^j = b_j$ is respected, it follows from the proofs of earlier sections that $\mathcal{S}_2(\mathbf{x})$ is a proper approximation of $\mathcal{S}(\mathbf{x})$. If the value of the vector $\mathbf{p}^j$ is not known, $\mathcal{S}_2(\mathbf{x})$ can be considered as a parametrized approximation of $\mathcal{S}(\mathbf{x})$. Note that the context constraint is only on the p-variables, since the t-variables have been assigned a fixed value.

6.3.2 An LP formulation

Algorithmically we can consider $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ to be a non-parametric LP system with unknown variables $(\mathbf{x}, \mathbf{p}^j)$. Such an interpretation is possible because there exist no non-linear terms in the definition of $\mathcal{S}_2(\mathbf{x}, \mathbf{p}^j)$ and it is only the feasibility of the approximation that is interesting. Supposing the system $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ is solved for feasibility, with the unknown variables vector as $(\mathbf{x}, \mathbf{p}^j)$, and a valid assignment for the values of both $\mathbf{x}$-variables as well as the $\mathbf{p}^j$-variables is found, then the system $\mathcal{S}_2(\mathbf{x})$ as given by Equation 6.1 can be considered an approximation of the system $\mathcal{S}(\mathbf{x})$, as long as the $\mathbf{p}^j$-variables satisfy the constraint $\sum \mathbf{p}^j = b_j$. On the other hand, $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^j)$ is a higher dimensional system than $\mathcal{S}(\mathbf{x})$ and hence cannot be considered as an approximation of the latter. It is an intermediate form useful for algorithmic purposes.

Example 6.3.2 [TVPI-UA: LP method]. Here is a reduced example from Banerjee's book [Ban92]. Let the original system be

$$\mathcal{S}(x,y,z) = \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \middle| \begin{pmatrix} & & -z \\ x & & -z \\ & -y & +z \\ -x & +y & +z \end{pmatrix} \geq \begin{pmatrix} -3 \\ 0 \\ 1 \\ -1 \end{pmatrix} \right\}$$

It is clearly not a TVPI system as the fourth constraint is non-TVPI. The median method applied to this constraint yields the system:

$$\mathcal{S}_1(x,y,z) = \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \middle| \begin{pmatrix} & & -z \\ x & & -z \\ & -y & +z \\ -x & +y & \\ -x & & +z \\ & y & +z \end{pmatrix} \geq \begin{pmatrix} -3 \\ 0 \\ 1 \\ -\frac{2}{3} \\ -\frac{2}{3} \\ -\frac{2}{3} \end{pmatrix} \right\}$$

The above system turns out to be an empty system because of the contradiction in the constraints. Such a trivial under-approximation is uninteresting. On the other hand, the method in this section leads to the following higher dimensional system:

$$\mathcal{S}_{HD}(x,y,z,p_1,p_2,p_3) = \left\{ \begin{pmatrix} x \\ y \\ z \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} \middle| \begin{pmatrix} & & -z & & & \\ x & & -z & & & \\ & -y & +z & & & \\ -x & +y & & +2p_1 & & \\ -x & & +z & & +2p_2 & \\ & y & +z & & & +2p_3 \\ & & & p_1 & p_2 & p_3 \\ & & & -p_1 & -p_2 & -p_3 \end{pmatrix} \geq \begin{pmatrix} -3 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix} \right\}$$

where the variables p_1, p_2, p_3 are additional context variables and the last pair of constraints are for the context constraint $p_1 + p_2 + p_3 = 1$. When system $\mathcal{S}_{HD}$ is solved for a rational feasible point with normal `lexmin` as cost function using PIP [Fea88], and the set of p-variables that are obtained as solution $(p_1, p_2, p_3) = (\frac{1}{2}, 0, \frac{1}{2})$ are substituted, we obtain:

$$\mathcal{S}_2(x,y,z) = \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \middle| \begin{pmatrix} & & -z \\ x & & -z \\ & -y & +z \\ -x & +y & \\ -x & & +z \\ & y & +z \end{pmatrix} \geq \begin{pmatrix} -3 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \right\}$$

The reader can verify the non-satisfiability of approximation $\mathcal{S}_1$ and satisfiability of the approximation $\mathcal{S}_2$. □

The above method involves only one call to a standard LP solver. The disadvantage is that the system $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p})$ has $n + \binom{\|\mathbf{a}_j\|}{2} = n + \|\mathbf{a}_j\|(\|\mathbf{a}_j\| - 1)/2 \approx \mathcal{O}(n + \|\mathbf{a}_j\|^2)$ dimensions, where $\|\mathbf{a}_j\|$ is the number of non-zero elements in the vector $\mathbf{a}_j$. As the method involves a call to an LP solver, its theoretical cost is not strongly polynomial time.

6.4 Multiple-constraint LP formulations

When there exist multiple non-TVPI constraints in $\mathcal{S}(\mathbf{x})$, each one of them has to be approximated to find a TVPI approximation of the polyhedron. Let m_k be the number of non-TVPI constraints in $\mathcal{S}(\mathbf{x})$, with $m_k \leq m$. Without loss of generality, we can assume that the constraints have been ordered such that the non-TVPI constraints come first, followed by the TVPI constraints. This means that the constraints of $\mathcal{S}(\mathbf{x})$ are $\{1, \ldots, m_k, \ldots, m\}$, with the constraints $\{1, \ldots, m_k\}$ being non-TVPI constraints and the constraints $\{m_k + 1, \ldots, m\}$ being TVPI constraints.

6.4.1 One-shot method

A straightforward way in which one can find a TVPI approximation of the above non-TVPI system $\mathcal{S}(\mathbf{x})$ is to construct a system $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^1, \ldots, \mathbf{p}^{m_k})$ in which all the non-TVPI constraints in $\mathcal{S}(\mathbf{x})$ are approximated using the scheme described in Section 6.3.2. This system can be solved using an LP formulation in variables as $\{\mathbf{x}, \mathbf{p}^1, \ldots, , \mathbf{p}^{m_k}\}$. An approximation of $\mathcal{S}(\mathbf{x})$ could be found as $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^1, \ldots, \mathbf{p}^{m_k})|\mathbf{p}^1, \ldots, \mathbf{p}^{m_k}$. It can easily be shown that the latter system is a proper approximation as long as the context constraints $\sum \mathbf{p}^1 = b_1, \ldots, \sum \mathbf{p}^{m_k} = b_{m_k}$ are respected.

But, finding the approximations of all the non-TVPI constraints simultaneously in the above fashion would lead to an LP system, with too an high increase in the number of dimensions. $\mathcal{S}_{HD}(\mathbf{x}, \mathbf{p}^1, \ldots, \mathbf{p}^{m_k})$ could have up to

$$n + \sum_{l=1}^{m_k} \binom{\|\mathbf{a}_l\|}{2} = n + \sum_{l=1}^{m_k} \|\mathbf{a}_l\|(\|\mathbf{a}_l\| - 1)/2$$

dimensions which could be as large[21] as $\mathcal{O}(n + \widehat{\|\mathcal{S}_{m_k}\|}^3)$, with $\widehat{\|\mathcal{S}_{m_k}\|}$ being the average number of non-zero coefficients/elements in the non-TVPI constraints in $\mathcal{S}(\mathbf{x})$.

[21] Though $\widehat{\|\mathcal{S}_{m_k}\|}$ itself is a small number, there are cases even in PolyBench 2.0—like `adi` and `fdtd-2d`—for which, the additional dimensional-increase by $\widehat{\|\mathcal{S}_{m_k}\|}^3$ could be considerable. More importantly, the system is very *skewed*: with respect to the ratio of constraints to variables, with respect to the presence of zero and non-zero elements in the constraint elements, as well as with respect to the number of equalities (leading to a non-trivial lineality space with high dimensions). For such skewed systems, simplex based solvers like PIP take longer times, than the near cubic-time (of Observation 3.3.1 on page 28) that is generally observed. We are grateful to Paul Feautrier for pointing this complexity aspect of Simplex and PIP in a personal communication.

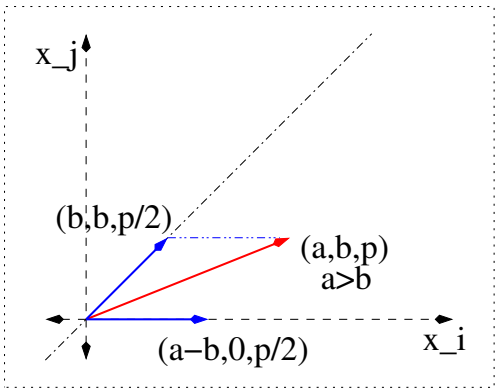


Figure 6.2: TCPV to UTCPV Approximation

6.4.2 Iterative methods

We could however iterate the above process described in Section 6.3.2 on page 73 for each of the m_k non-TVPI constraints in $\mathcal{S}(\mathbf{x})$ on an iterative basis. Clearly, there could be choice in the method in whether the original system is being updated with the approximation constraints of each non-TVPI constraint or not. When the original system is immediately updated, we call the method *incremental*. When the approximations of all non-TVPI constraints are found by constructing LP formulations on the same LP system each time, we call the method *independent*.

It could be noticed that each of the above methods involves multiple LP calls – one for each non-TVPI constraint $\mathcal{S}(\mathbf{x})$. This means making up to $\mathcal{O}(m)$ LP calls in total for building the approximation system. But, each of the LP systems is of $\mathcal{O}(n + \widehat{\|\mathcal{S}_{m_k}\|^2})$ -dimension and is not that high-dimensional when compared to the above one-shot formulation.

6.5 Per-constraint UTVPI-UA of TVPI

Let us sketch a simple per-constraint algorithm that takes one TVPI constraint and returns its UTVPI under-approximation system.

From Lemma 5.2.3, a vector in the polarity construction needs to be TCPV and to have equal magnitude components in the non-homogenizing dimensions for the original to be UTVPI. So, the intuition for this algorithm is similar to the TCPV-OA, namely that reasoning about the TCPV original vectors in the polar space and computing a set of UTCPV vectors which over-approximate the original TCPV vector would result in an over-approximation.

Suppose the TCPV vector has components as (a, b, p) in the (x_i, x_j, x_0) dimension (with x_0 being the homogenizing dimension), then we can replace it with two UTCPV vectors. The replacement has to just take care of the fact that the new UTCPV vectors are such that the original vector is in the conical sum of the replacements. As the case when $a = b$ means that the vector is already a UTCPV vector, the other

cases can be handled in the following way:

$$\begin{pmatrix} a \\ b \\ p \end{pmatrix} \in \begin{cases} a > b: & \text{cone} \begin{pmatrix} b & a-b \\ b & 0 \\ \frac{p}{2} & \frac{p}{2} \end{pmatrix} \\ a < b: & \text{cone} \begin{pmatrix} a & 0 \\ a & b-a \\ \frac{p}{2} & \frac{p}{2} \end{pmatrix} \end{cases}$$

In either of the above cases, it can be noticed that the first vector is a UTCPV vector and the second is an interval vector. The first case is illustrated in Figure 6.2.

Lemma 6.5 [Validity of the above UTVPI approximation]. Given a TVPI constraint, the above method returns a valid UTVPI-UA of the constraint.

Proof: The geometric proof derives from the observation that the sum of the corresponding UTCPV replacement vectors is the original vector. Hence every vector in the polar space that is reachable by the original vector is reachable by these replacement vectors, meaning that they give an OA in the polar space. In the primal space, the replacements necessarily give an UA (Cor. 5.3.1b). ■

Example 6.5 [TVPI to UTVPI-UA]. Let $P^{\mathcal{V}} = \text{conv}\left\{\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}\right\}$, with the $\mathcal{H}$-form of P being

$$P^{\mathcal{H}} = \{1 \leq x \leq 4; 1 \leq y \leq 2; x + 3y \leq 7\}$$

It can be seen that P is trivially a TVPI system, but not a UTVPI system, as the third constraint $x + 3y \leq 7$ is a non-UTVPI constraint. The UTVPI approximation by the above method is $\text{UA}(P^{\mathcal{V}}) = \{1 \leq x \leq 4; 1 \leq y \leq 2; x + y \leq \frac{7}{2}; 2y \leq \frac{7}{2}\}$ which can be seen to be non-empty with vertices

$$\text{UA}(P^{\mathcal{V}}) = \text{conv}\left\{\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ \frac{7}{4} \end{pmatrix}, \begin{pmatrix} \frac{5}{2} \\ 1 \end{pmatrix}, \begin{pmatrix} \frac{7}{4} \\ \frac{7}{4} \end{pmatrix}\right\}$$

The under-approximation is shown in Figure 6.3. □

The cost of finding the UA is linear and the method is simple to implement, just like the median method mentioned in Section 6.2. Just like that method, this method is not guaranteed to generate a non-empty under-approximated system.

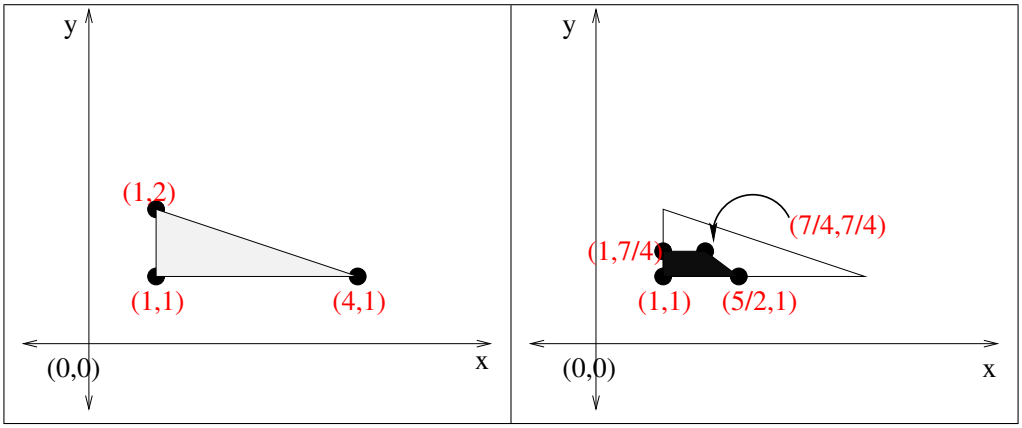


Figure 6.3: Example for TVPI to UTVPI Approximation

6.6 LP-based Parametrized UTVPI Approximation

This approximation is similar to the parametrized TVPI approximation covered in Section 6.3, in the sense of searching for p variables instead of t's.

But, as the search is for UTCPV vectors instead of TCPV vectors, the p-values that are equal in each column can as well be scaled down to $\{-1, +1\}$ depending on the sign of the particular value in the original vector. This latter preservation of signs is needed because of the convexity restriction of Equation 5.4 on page 64. It needs a positive sum and not an arbitrary sum. If we define the sign function on a non-zero rational element c to be as $sg(c) = \begin{cases} c > 0: & +1 \\ c < 0: & -1 \end{cases}$, and the vector $\mathbf{v}^T = \begin{pmatrix} sg(a_1) & sg(a_3) & sg(a_3) & b \end{pmatrix}^T$ then, the approximation is $\begin{pmatrix} a_1 & a_3 & a_3 & b \end{pmatrix}^T \in \text{cone}\,(T_j|U_j)$ where

$$\left(T^j|U^j\right) = \left(\begin{array}{ccc|ccc} v_1 & v_1 & 0 & v_1 & 0 & 0 \\ v_2 & 0 & v_2 & 0 & v_2 & 0 \\ 0 & v_3 & v_3 & 0 & 0 & v_3 \\ p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{array}\right)$$

With appropriate context constraints involving variables $p_1, \ldots, p_6$, and the overall linear program solved with an LP solver, the above can result in a non-empty UTVPI-UA. Ths similarity of this matrix with DDV-vectors is to be noted.

6.7 Metrics and Discussion

Let us now study the size of the new system, the complexity of the conversion, the overall complexity of finding a solution, and discuss some fundamental and methodological limitations of our approach.

6.7.1 Sizes

Let the original matrix be of size $m \times n$: m constraints and n variables with m_k and m_t being the number of non-TVPI and TVPI constraints respectively. Also let the overall sparsity factor of the system be $\hat{s}$, which means that for a constraint in the input system, on average $\hat{s}$ variable elements are non-zero. It will be seen in later sections that for practical purposes, $\hat{s}$ is a small constant (little more than 4) and relatively independent of n.

TVPI-UA. For a system described as above, each of the TVPI-UA methods (given in Sections 6.2 and 6.3) replaces a non-TVPI constraint $\mathbf{a}_j$ with the same number of $\binom{\|\mathbf{a}_j\|}{2} = \|\mathbf{a}_j\|(\|\mathbf{a}_j\| - 1)/2$ TVPI constraints. Doing the above process for each of the m_k non-TVPI constraints creates an approximated TVPI system of size $m_a \times n$, where $m_a = m_t + \widehat{\|\mathcal{S}_{m_k}\|}(\widehat{\|\mathcal{S}_{m_k}\|} - 1)m_k/2$. It can be assumed that the new system is approximately of the size $\hat{s}^2 m \times n$. In the rare case that $\hat{s} = n$ (which means that the constraint matrix does not have any zero entries) then the size of the resultant TVPI constraint system is $\frac{n(n-1)m}{2} \times n$ which is of the order of $n^2 m \times n$.[22]

UTVPI-UA. For the UTVPI approximation given in Section 6.5, the number of additional constraints is twice what it is for TVPI-UA case; this means that they have the same order of complexity increase as the previous ones.

Constraint graph (MUTVPI/DBM system). The Bellman-Ford algorithm constructs a constraint graph, or incidence matrix, of nearly twice the size of the input UTVPI system before solving it: a $m \times n$ input system results in $(2n + 1)$ nodes and $(2m + 2n)$ edges, due to simple transformations of addition of positive and negative forms (x_i^+, x_i^-) of each variable [Min06, Section 2.2], and addition of a pseudo-source-vertex for the shortest-paths problem [AMO93]. The former is a relaxation of UTVPI into difference constraints and is exact for rational points [Min06], though some (odd) integer points in the original are lost.

6.7.2 Complexity of conversion

Both the median method of TVPI approximation covered in Section 6.2 and the UTVPI approximation covered in Section 6.5 are strongly polynomial time algorithms as they do not use any LP call when constructing the approximation. The parametric approximation covered in Section 6.3.2 is a weakly polynomial time algorithm for it needs to solve an LP problem for finding the homogenizing dimension values.

For multiple constraints case, complexity of conversion is one LP call for the one-shot algorithm of Section 6.4.1, and one LP call per non-TVPI constraint for both incremental and independent algorithms. Each of the above numbers are weakly polynomial time in complexity, but worst-case times nonetheless.

[22] Reminder: We use the $\times$ to denote the sizes of matrices or systems.

6.7.3 Complexity of finding a feasible solution

For the TVPI approximation, as the worst-case complexity of Hochbaum-Naor is $\mathcal{O}(mn^2 \log m)$ [HN94], applying it to the resultant approximated system would lead to $\mathcal{O}(\hat{s}^2 mn^2 \log \hat{s}m)$ time with constraint sparsity $\hat{s}$, and $\mathcal{O}(n^4 m \log nm)$ in the unlikely case when $\hat{s} = O(n)$.

For the UTVPI approximation, as the theoretical worst-case complexity of solving UTVPI systems is $\mathcal{O}(mn)$ (if we use the traditional Bellman-Ford algorithm on the difference constraints), the corresponding complexities would be $\mathcal{O}(\hat{s}^2 mn)$, and $\mathcal{O}(n^3 m)$ respectively. In Chapter 7, we see empirical evidence suggesting that in practise, a TVPI constraint *always* turns out to be a UTVPI constraint, making this method very attractive.

6.7.4 Preprocessing and Limitations

It can be noticed that our algorithms do not ask for removal of redundant inequalities, which is as hard as determining if the input system is feasible.

Polyhedra in compilation have lot of duplicate constraints, which gets reflected in the approximations as well. Compilers like PLuTo remove these by methods like textual matching which leads to a large decrease in the number of constraints. A more advanced scheme, finely-tuned to UTVPI constraints, would use a hashing or radix-sort technique [AN98] packing the UTVPI constraint in a few integers: exploiting the fact that the non-homogenizing dimension values are from $\{0, \pm 1\}$, while the homogenizing dimension values are *always* from a very small set $[-5, 5]$. It would bring the simplification complexity to linear time. We thus believe that the problem of duplicate elimination is asymptotically insignificant or can be divided per-dependence-edge as amortized-cost in addition to other fixed costs like dependence feasibility tests, Farkas multiplier simplification and Gaussian eliminations.

The following example shows an inherent limitation of the process of TVPI-UA itself.

Example 6.7.4 [Polyhedra with non-satisfactory approximations]. The polyhedron described using the constraints $\{x + y + z \geq -1;\ x + y + z \leq 1\}$ is not TVPI. Though it is unbounded in both directions, the only TVPI approximations that are possible of the above polyhedron are bounded TVPI polyhedra, which can be considered a failure of the UA approach. Similarly, the TVPI polyhedron (which is non-UTVPI) described by the constraints $\{x + 2y \geq -1;\ x + 2y \leq 1\}$ can only be under-approximated by UTVPI polyhedra that are finite. The second (planar) example is shown in Figure 6.4. □

The above kinds of polyhedra can be considered as degenerate cases and identified by a pre-processing step that removes the lineality space from the input polyhedron. Furthermore, the Farkas polyhedra that arise in polyhedral scheduling are always pointed polyhedra and can never have non-trivial lineality space.

6.7.5 Integer scaling and TU polyhedra

One limitation of the parametric approximations of Sections 6.3 and 6.6 is that they may result in *rational* polyhedra, whose constraints are of the form $ax_i + bx_j \leq c;\ a, b, c \in \mathbb{Q}$ for TVPI and $\pm x_i \pm x_j \leq c;\ c \in \mathbb{Q}$, for UTVPI. Having such rational polyhedron as approximations could be a limitation in cases when an integer

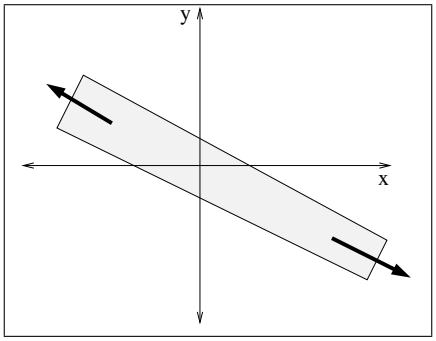


Figure 6.4: A planar polyhedron which admits only trivial UTVPI Under-Approximations. Because the above polyhedron has non-trivial lineality space, it admits only trivial UTVPI under-approximations which are bounded.

solution is needed (for scheduling purposes), or an overall integer polyhedra is needed (for code generation purposes). But, by a small change in the design of these methods, per-constraint—or per-dependence (in the case of using one-shot algorithm)—methods, it can be ensured that the resulting difference constraints under-approximations have integer constraints of the type $x_i - x_j \leq c$, where $c \in \mathbb{Z}$.

The integer method. This new method would be essentially be the same as the previously mentioned methods. It would however involve scaling up the pseudo-parametric variables, changing the context constraints accordingly, and *solving an ILP problem*—instead of the usual LP problem—though on a per-P_e basis.

Total Unimodularity. Such a local transformation will not only induce integer schedules because incidence matrices are a sub-class of network matrices, but also has the additional property that all vertices of the resultant overall polyhedron are integral; this is because such matrices are Totally Unimodular [Sch86, Chapter 19].

It will be seen later that this method would help in polynomial time approximations of NP-Complete problems and code-generation. In this aspect, UTVPI systems are really special. TVPI systems do not have the property because of the result of Lagarias [Lag85], who proved that solving integer-TVPI systems is also NP-Complete.

We continue with the example 6.3.2 to show that the approximation can result in a TU matrix.

Example 6.3.2 (continued.) [TU approximation]. In the earlier example, there are two alternatives in which the resultant system can be obtained:

$$\mathcal{S}_2^{\text{Rat}}(x,y,z) = \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \middle| \begin{pmatrix} & & -z \\ x & & -z \\ & -y & +z \\ -x & +y & \\ & y & +z \\ -x & & +z \end{pmatrix} \geq \begin{pmatrix} -3 \\ 0 \\ 1 \\ -\frac{1}{2} \\ -\frac{1}{2} \\ 0 \end{pmatrix} \right\}$$

$$\mathcal{S}_2^{\text{Int}}(x,y,z) = \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \middle| \begin{pmatrix} & & -z \\ x & & -z \\ & -y & +z \\ -x & +y & \\ & y & +z \\ -x & & +z \end{pmatrix} \geq \begin{pmatrix} -3 \\ 0 \\ 1 \\ -1 \\ -1 \\ 0 \end{pmatrix} \right\}$$

It can be noticed that the homogenizing dimension values for $\mathcal{S}_2^{\text{Rat}}(x,y,z)$ are rational numbers, while that for $\mathcal{S}_2^{\text{Int}}(x,y,z)$ are integers. Monotonizing both the systems to yield a MUTVPI (DBM) system whose homogenizing dimensions have integer values would yield a Totally-Unimodular approximation for the integer system. (Though it would need further scaling to take care of the homogenizing dimension halving in the UTVPI $\xrightarrow{\text{TRANS}}$ MUTVPI conversion.)

In such a resultant MUTVPI (DBM) systems where each constraint is exactly of the form: $x_i - x_j \leq c$; $c \in \mathbb{Z}$, solving a simple rational programming problem would automatically yield a solution all of whose values are integers. Obtaining the same for a system like $\mathcal{S}_2^{\text{Rat}}(x,y,z)$ would mean solving an integer programming problem, which could be much costlier because of the associated cost with Gomory cuts [Sch86]. □

6.8 Conclusions and Perspectives

In this section, we first discuss some conclusions and contributions, and then present some perspectives.

6.8.1 Conclusions and contributions

In this chapter, we began with the framework given in the previous chapter and gave under-approximation algorithms that take a general convex polyhedron and give some (U)TVPI under-approximations. The algorithms take polynomial time. The Table 6.1 summarizes the complexities of the methods introduced in this chapter:

Conversion of Polyhedra (UA/TRANS)	Metric		
	Sizes	Complexity of Conversion (LP based methods)	Improvement in Complexity of Solution
Poly $\xrightarrow{\text{UA}}$ TVPI	$m \times n \xrightarrow{\text{UA}} \hat{s}^2 m \times n$	$Z(\hat{s}^2 m, n)$	$Z(m,n)$ to $\mathcal{O}(\hat{s}^2 mn^2 \log \hat{s}m)$
TVPI $\xrightarrow{\text{UA}}$ UTVPI	$m \times n \xrightarrow{\text{UA}} 2m \times n$	$Z(m,n)$	$\mathcal{O}(mn^2 \log m)$ to $\mathcal{O}(mn)$
UTVPI $\xrightarrow{\text{TRANS}}$ MUTVPI	$m \times n \xrightarrow{\text{TRANS}} (2m + 2n) \times (2n + 1)$ incidence matrix	–	$\mathcal{O}(mn)$ to $\mathcal{O}(mn)$ (No loss of precision in rational case.)
Poly $\xrightarrow{\text{UA}}$ MUTVPI	$m \times n \xrightarrow{\text{UA}} (\mathcal{O}(\hat{s}^2 m + 2n)) \times (2n + 1)$ Tableau $\xrightarrow{\text{UA}}$ Incidence matrix	$Z(\hat{s}^2 m, n)$	$Z(m,n)$ to $\mathcal{O}(\hat{s}^2 mn)$

Table 6.1: Complexities of Under-Approximation Algorithms

Contributions. The problem of finding an (U)TVPI-UA of a convex polyhedron is both easy and difficult because there are many solutions to the problem; for a given convex polyhedron, there *always* exists a non-empty (U)TVPI under-approximation. We have defined algorithms that are simple, run in polynomial time employing linear programming, and are easy to illustrate in an algorithmic and geometric sense. We believe that these algorithms can be used as they are, or can be used as a framework to design other (U)TVPI-UA algorithms.

In Section 6.7.5 we introduced an algorithm which involves solving an ILP problem on a per-dependence basis, with the result that the overall polyhedron has integral (not just rational) properties. We also believe that this is a unique contribution to polyhedral compilation, because it helps the overall intersection of per-dependence polyhedra to very cheaply have integer vertices. The system obtained can be solved using

Bellman-Ford with a very low polynomial time cost. This is because the per-dependence edge polyhedra are mostly trivially small and solving ILP problem on them is usually never more costly than solving a normal LP problem. As far as we know, this method has not been previously explored in polyhedral compilation up to now. In Chapter 8 on page 101 we study some potential applications of this method. In this aspect, it should be noted that UTVPI polyhedra are special, even when compared to TVPI polyhedra. because of the result of Lagarias [Lag85] who proved the NP-Completeness of solving integral TVPI-systems, such a Totally Unimodular approximation is not possible even with TVPI polyhedra.

6.8.2 Perspectives and discussion

Different methods for using the UA algorithms. For converting an arbitrary convex polyhedron into a TVPI under-approximating polyhedron so that Hochbaum-Naor's algorithm [HN94] can be used, the method is the following: Poly $\xrightarrow{\text{UA}}$ TVPI. On the other hand, for converting into a UTVPI polyhedron so that Bellman-Ford algorithm can be used to solve the resultant, the method we suggest is the following:

$$\text{Poly} \xrightarrow{\text{UA}} \text{TVPI} \xrightarrow{\text{UA}} \text{UTVPI} \xrightarrow{\text{TRANS}} \text{MUTVPI} \tag{6.2}$$

For this latter process, one can alternatively choose the method Poly $\xrightarrow{\text{UA}}$ UTVPI $\xrightarrow{\text{TRANS}}$ MUTVPI, where the intermediate step of involving TVPI polyhedra is skipped, but there is seldom any loss of precision when doing the approximation TVPI $\xrightarrow{\text{UA}}$ UTVPI. This is due to the nature of constraints in polyhedral compilation, which already have TVPI-ness and constraints that are already TVPI are highly likely to be UTVPI as well. Direct UTVPI approximation from Poly to UTVPI would mean insertion of 6 vectors: the normal 3 UTCPV vectors and an additional 3 interval vectors corresponding to interval constraints: vectors of the form $\begin{pmatrix} \pm 1 \\ 0 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ \pm 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 0 \\ \pm 1 \end{pmatrix}$, where the sign of the ± 1 vector is determined by the sign of the particular component of the vector that it is approximating.

Feasibility control. One important characteristic of under-approximation methods is the preservation or not of the feasibility of the UA. In this respect, the median method though having other advantages– like for example, ease of implementation–comes with the serious weakness that it offers no guarantee of preservation of feasibility when the input polyhedron is non-empty. The LP-independent method and LP-incremental methods are a little better in this respect, and only the one-shot method comes with an absolute guarantee that the UA is always non-empty when the input is. This aspect will be covered more in Chapters 7 and 8, where more advanced clustering techniques are discussed and implemented so that the output, in the presence of intersection of many polyhedra can come with such a guarantee. The overall intersection of under-approximations is ensured to be non-empty.

Necessity of polarity framework. The previous chapter discussed two alternate ways of reasoning about the approximations: a homogenization-polarity framework along with the equivalent application of affine form of Farkas lemma. One question this raises is whether the former is really needed. One

traditional answer is that the homogenization-polarity framework is completely subsumed by the Farkas lemma application and hence the former is redundant. But, it is our opinion that the use of polarity (in particular conical polarity) is a unique contribution to this aspect. It is indeed true that Farkas lemma makes the former redundant, but, it is also true that with Farkas lemma being such a fundamental aspect of analysis of convex polyhedra, *everything* else in it comes literally for free once it is assumed.

Also, the duality concept and Farkas lemma are *exactly* equivalent to each other, both having their ground in Farkas-Minkowski-Weyl theorem about finite generation of convex cones [Sch86, Cor. 7.1a]. So, the criticism that the polarity framework is redundant because it uses duality stands on weak ground. The reconciliation could be that the polarity framework uses the geometric aspect of duality, while the equivalent Farkas lemma application uses the equational aspect of duality.

The polarity framework offers a couple of adantages as well. Primarily, the construction using polarity offers a unique geometric perspective finding approximations. This latter aspect is obviously not provided by the Farkas lemma application. Also, the polarity construction can obviously be extended for the dual problem of OA as well. Furthermore, when applied to the problem of finding (U)TVPI-UA of a general convex polyhedron, it offers many choices, mainly because there are many ways of choosing a valid UA. Other users of the approximation framework, like for solving verification problems, may find that the study of the framework while designing their UAs be useful to provide hints at the kind of approximation that would suit their purposes.

More powerful alternatives

Other than the LP based methods that we proposed to find (U)TVPI-UA of a polyhedron, we study two alternative methods to do the same and show their limitations: one using generator representation and another using non-linear solvers or quantifier elimination methods. Obviously, neither of them scale on the overall P and the only scalable means of running them would be on a per-P_e basis.

Generator representation. The straightforward alternative of finding UA is to use the generator representation of the (U)TVPI polyhedra. This algorithm could be similar to the OA algorithm designed by Miné for UTVPI polyhedra and studied in Section 4.3.2 on page 50. Such a method could have a guarantee of non-emptiness of the approximation, and as well as an additional guarantee with respect to the volume ratio of with respect to the original. The LP based method provides the former in its one-shot variation, but not the latter. Obviously, this method comes with the prohibitive exponential cost associated with Chernikova algorithm. Except for some benchmarks—like `pca` and `tce`—in PolyBench 3.0, such a method may be acceptable on a per-dependence basis. But, the advantages it comes with are only on a single polyhedron case. Since the problem in polyhedral compilation involves intersections of many per-dependence polyhedra, such prohibitive asymptotic cost may not tolerable, except in some offline and online compilation methods.[23]

[23]The biggest open problem here is of course to study if Under/Over-Approximation of convex polyhedra using UTVPI polyhedra may help in obtaining a strongly polynomial time bounds for the optimization problem for convex problem, as shown in Figure 6.5.

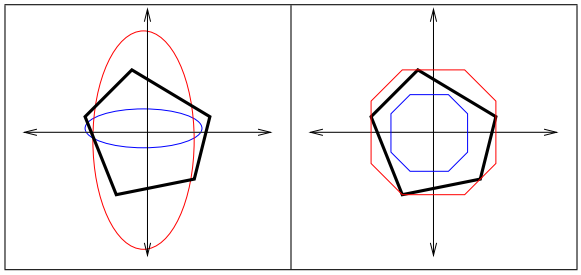


Figure 6.5: Ellipsoid style UTVPI Approximation of Convex Polyhedra

Non-linear methods. The other powerful methods would be to design quadratic or non-linear methods using the Equation 5.6 on page 65 and Equation 5.8 on page 66. As hinted earlier, the key observation here is to notice that those equations are potentially quadratic, but existentially qualified. Hence, they can be solved by quantifier elimination. These latter methods have come a long way since their original discovery by Tarski [Tar51]. The solvers of the so called Cylindrical Algebraic Decomposition (CAD) methods have been improved in recent times [ACM84]. They have even been applied to dependence analysis, parametrized scheduling of tiled parameters and even by code generation by Größlinger [Grö09]. They however have extremely high computational complexity and come with no guarantee. Another alternative is to use geometric programming and posynomials [BKVH07]. These latter methods have also been used to extend the polyhedral compilation framework: by determining the tile-sizes ("parametric-tiling") by framing a non-linear optimization problem by Renganarayana et al. [RR08]. As far as we are aware of, these methods also have no guarantee of worst-case polynomial complexity, and hence have low scalability.

Limitations of the UA algorithms

Clearly, there are issues with the methods proposed in this chapter: dimensional increase of the intermediate system to find the homogenizing dimension values in the parametric approximations, and the increase of the size of the resultant systems with respect to the constraint form and the vertex form.

Dimensional increase of the intermediate system. The LP-independent method involves solving an LP problem on the High Dimensional system whose number of variables increases quadratically with respect to the sparsity in the original system. However, we have found that it scales well for *all* the systems in the benchmarks in PolyBench 2.0, either on a per-dependence basis, or when applied to the overall Farkas system. The latter could be used for example in an offline and online compilation mode. Also, the nearly cubic size increase of the intermediate HD system for one-shot method is indeed too high to run on the overall Farkas system, unless additional precautions are taken to eliminate duplicates as suggested below. When this method of approximation is run on the overall Farkas polyhedron, clearly, the one-shot system cannot be used primarily because of the dimensional increase limitation. This was empirically observed when running the one-shot system on the PolyBench 3.0 benchmarks. The reader is also invited to read Footnote 21 on page 75 in this regard.

The method we suggest is to run the LP-independent or LP-incremental methods on the overall Farkas systems for an offline online compilation basis. On a per-dependence basis, the one-shot will scale well for

most of the benchmarks in polyhedral compilation. For the rest of the cases it may be necessary to run the LP-independent or LP-incremental method on the per-dependence edge systems, but in those very rare cases, we suggest using the LP-independent method because of the simplicity of the approach, and the ease of implementation.

We see in Chapter 7 that the LP-Independent method is quite scalable. We have also found from experiments that the one-shot algorithm on the per-dependence edge basis is scalable as well, because the number of non-TVPI constraints on a per-P_e basis is quite low.

So as to limit the dimensional increase, additional precautions, like temporary removal of the zero dimensions in the HD system, one in which all the variables do not have any active components, could be done. Though we have not implemented this method, it is likely to help in the scaling of any of the methods and especially on a per-dependence edge basis, where many variables are likely to be inactive.

Increase in number of constraints (and generators). A quadratic increase in the number of resultant constraints proportional to sparsity ($\hat{s}^2$) for each of the methods suggested in this chapter may seem excessive. But, it seems unavoidable because of the topological nature (limitedness) of TVPI constraints when compared to general convex polyhedra. It is also a matter of fact that the resultant systems have an increase in the number of vertices. If the resultant of these UA algorithms are converted to a generator representation, for instance for verification purposes, then, a heuristic bounding of these could be applied. See for example Kanade et al. [KAI+09].

Duplicate constraints. Related to the above is the increase in the duplicates in the resultant over-approximated systems. As suggested in Section 6.7.4, they can removed by a post-processing step exploiting the simple nature of the output (U)TVPI-UA systems. But, the duplicates can also be limited before the LP call itself, by taking into account the repetition among the pairs of coefficients. For example, when constructing the TVPI-UA of the system $\{2x + 3y + 4z \leq 1; 2x + 3y - 4z \leq 1\}$ by using one-shot method, the normal method is to construct the HD system independent of the fact that the coefficients of x and y variable pair in the two constraints are exactly the same. Instead, the HD system could be formulated taking this fact into account: having fewer variables and constraints in the HD system. Though this leads to the formulation where the pseudo-parametric constraints have to be formulated carefully, such an improvement also leads to having fewer constraints in the resultant UA system.

More details that are discussed at the end of next chapter, where the results of implementation of these algorithms in the PLuTo source-to-source compiler will be presented.

Chapter 7

Experimental Evaluation of Under-Approximations

In this chapter, we study the experimental results of the UA-algorithms designed in the previous chapter.

7.1 Introduction

We carried out systematic experiments, tiling and parallelizing the 16 PolyBench (2.0) [PB$^+$] benchmarks shipped along with the source-to-source polyhedral optimizer PLuTo (PLuTo-0.6). After our experiments, a new version PLuTo (PLuTo-0.8) was released and which optionally uses ISL instead of PIP. Farkas systems are extracted from PLuTo. We compare the default linear programming calls to PIP [Fea88] with the TVPI-UA and then the UTVPI-UA algorithms. The UTVPI-UA is further relaxed into a constraint graph (or incidence matrix) which is fed to our implementation of the Bellman-Ford algorithm, along with the supporting routines. This process is:

$$\text{Poly} \xrightarrow{\text{UA}} \text{TVPI} \xrightarrow{\text{UA}} \text{UTVPI} \xrightarrow{\text{TRANS}} \text{MUTVPI} \xrightarrow{\text{BellmanFord}}$$

Our experimental framework used the PiPLib [Fea88] and PolyLib [Wil93] libraries, with an option to use FMLib [Pou] as well. Note that we still use PIP for the much smaller linear programming problems arising from the feasibility enhancing heuristics. Also, PLuTo makes use of PIP in parts unrelated with the scalability of affine scheduling problems: the PIP calls from PLuTo originate from the functions: `dep_satisfaction_test` (DS), `get_dep_direction` (DIR) and `find_permutable_hyperplanes` (FPH).

We focus on FPH calls which correspond to affine scheduling problems. There are many more calls of the DS and DIR varieties, more than two orders of magnitude, when compared to the FPH calls, but the optimization of the former pair of LP calls is entirely a different problem from the FPH variety in many ways. Primarily, the former are dependence polyhedra and need to be *over-approximated* [DRV00, UC11] (Section 4.1.1 on page 37) as otherwise, incorrect transformations would result. The latter of course are Farkas polyhedra, the main topic of this experimentation following our new scheme in Chapter 4 (Section 4.4 on page 52), and need to be *under-approximated* leading to a conservative approximation and

perhaps a loss of useful schedules.

7.2 Features of the polyhedra

Here we are referring to Table 7.1. The initial three columns refer to the size of the loop nest: L is the number of loops, S the number of statements, and D the number of dependences. The next sets of columns indicate the polyhedral characteristics for the different varieties of calls in PLuTo. As the DS and DIR variety are similar types of calls, they have been summarized together.

The number of polyhedra of different types are indicated in the P columns (P_{DS}, P_{DIR} and P_{FPH}). As written earlier, there are lot more polyhedral calls of DS/DIR than FPH. But, assuming that the loop-nest depth d is a small bounded constant, the former are smaller systems and are linearly dependent on the size of the loop nest: $\mathcal{O}(\max(|E|, |V|)$. For a particular benchmark and variety of polyhedra (DS/DIR or FPH), n and $\hat{m}$ columns indicate the number of variables (unknowns), and the average number of constraints in the particular LP formulation respectively. The $\hat{m}_t$ column indicates the average number of TVPI constraints for that benchmark and variety of call.

DS/DIR. As it can be expected, most of the constraints in the DS/DIR polyhedra are TVPI constraints. In these polyhedra, there is *never* more than 1 constraint per polyhedron which is non-TVPI. In all the cases, whenever a constraint is TVPI, it is *always* turns out to be a UTVPI constraint, having the same absolute magnitude for the two coefficients, when it has two entries. An interval constraint like $x_i \leq c$ or $x_i \geq c$, with only one non-zero coefficient, is of course UTVPI as well.

FPH. In the FPH variety, it can be noticed that the sizes of some of the FPH polyhedra are small and comparable to the DS/DIR polyhedra. This is either the result of a small problem size or because PLuTo uses Gaussian and Fourier-Motzkin elimination, along with a syntactic heuristic to reduce the duplicate constraints. As it can be expected, $\hat{m}_t$, number of TVPI constraints in the original system, is highly benchmark dependent. But, just like in DS/DIR polyhedra, a TVPI constraint always happens to be a UTVPI constraint.

$\widehat{\|\mathcal{S}_{m_k}\|}$ is the average number of non-zero coefficients in the non-TVPI constraints for that benchmark. It can be seen that this number, though again being benchmark dependent, is a small constant when compared to the dimension size n. $\hat{m}_a$ is the number of constraints in the new (approximated TVPI) system. As seen earlier, $\hat{m}_a = \widehat{\|\mathcal{S}_{m_k}\|}(\widehat{\|\mathcal{S}_{m_k}\|} - 1)(m - m_t)/2 + m_t$.

The relative growth of the approximated system with respect to the original one is defined as the ratio between the sum of entries in the $\hat{m}_a$ and $\hat{m}$ columns. We found the average value of this to be 8, meaning that the overall sparsity factor $\hat{s}$ is a little more than 4. Sometimes the growth of the approximated system is significant, but it has to be remembered that $\hat{m}_a$ is the number of constraints without any simplification, while m is the obtained after systematic simplification and elimination of duplicates in PLuTo.

For comparison purposes with $\hat{m}_a$, we have added the $\hat{m}_u$ column, which is the average unsimplified system size when the simplification techniques used in PLuTo are turned off. It can be observed that $\hat{m}_u$ and $\hat{m}_a$ are comparable. Across all benchmarks, our experience shows that the approximated system

	Loop nest			DS(1)/DIR(2)					FPH						
Bench	#L	#S	#D	$\#P_1$	$\#P_2$	n	$\hat{m}$	$\hat{m}_t$	#P	n	$\hat{m}$	$\hat{m}_t$	$\widehat{\|\mathcal{S}_{m_k}\|}$	$\hat{m}_a$	$\hat{m}_u$
adi	12	4	54	90	564	9	20	19	3	20	200	65	5	1844	614
corcol	12	6	14	38	194	9	17	16	3	22	22	13	5	130	77
covcol	13	7	26	41	228	15	25	24	3	24	18	14	4	29	55
dsyr2k	3	1	3	9	18	8	18	17	3	7	8	6	3	11	18
dsyrk	3	1	3	9	18	8	18	17	3	7	8	7	3	11	18
fdtd-2d	11	4	48	39	168	10	22	21	3	20	96	35	6	1010	367
gemver	7	4	13	29	161	6	13	12	2	14	21	15	4	48	47
jacobi-1d	4	2	10	16	88	7	14	13	2	10	32	14	5	232	104
jacobi-2d	6	2	14	21	84	9	19	18	3	12	65	15	7	1135	212
lu	5	2	10	12	60	11	23	22	3	10	35	17	5	232	106
matmul	3	1	3	9	18	10	21	21	3	9	9	7	4	20	24
mvt	4	2	11	31	68	6	13	12	2	9	20	12	3	46	52
seidel	3	1	27	37	162	12	24	23	3	8	33	15	5	168	179
ssymm	8	3	15	33	126	8	19	19	3	14	15	10	3	22	36
strmm	3	1	4	8	24	8	17	16	3	7	12	8	3	20	30
tmm	3	1	3	9	18	10	21	21	3	9	11	8	4	23	30

Table 7.1: Problem size, Polyhedral and TVPI-ness characteristics

undergoes simplification and duplication removal techniques, it is reduced to a much smaller system, comparable to the one in m columns. Also, asymptotically, the size increase due to the sparsity does not matter much.

7.3 UA feasibility

Here we are referring to Table 7.2. These columns refer to the median method discussed in Section 6.2 on page 69 and to the LP-based independent method discussed in Section 6.4.2 on page 76 respectively. For the latter LP-based independent method, we took the overall Farkas system constructed by PLuTo and we have implemented the under-approximation Poly $\xrightarrow{\text{UA}}$ TVPI on only the overall Farkas system, one that is obtained after putting together all the per-dependence systems and after simplification of PLuTo; this method is not on a per-dependence basis.[24]

The PLuTo calls of the FPH variety are for LexMin. In the current table, we discuss the results of feasibility only. The columns YY (Yes-Yes), YN (Yes-No), denote the feasibility (Y) or infeasibility (N) of the original and approximated systems respectively. They have been highlighted accordingly. Since the FPH system is used to find an optimization point in the overall system, a YN entry would mean loss of parallelization.

It can be seen that the LP based independent method performs much better than the median method. The latter performs poorly (19 YY and 26 YN cases or 6 out of 16 PolyBench problems), but surprisingly well considering the simplicity of the approach. We expect the incremental method to have much better

[24]The explanation of the per-dependence variations and their feasibility results are explained in detail in Section 8.4.2 on page 110.

	Median		LP-Indep	
Bench	YY	YN	YY	YN
adi	0	3	0	3
corcol	1	2	0	3
covcol	3	0	3	0
dsyr2k	3	0	3	0
dsyrk	3	0	3	0
fdtd-2d	0	3	1	2
gemver	0	2	2	0
jacobi-1d	0	2	0	2
jacobi-2d	0	3	0	3
lu	0	3	0	3
matmul	3	0	3	0
mvt	0	2	2	0
seidel	0	3	3	0
ssymm	3	0	3	0
strmm	0	3	3	0
tmm	3	0	3	0
Total	19	26	29	16

Table 7.2: UA effectiveness: Median and LP-Independent methods (PolyBench 2.0)

performance than the current independent method (29 YY and 16 YN cases or 10 out of 16 PolyBench problems).

7.4 Scalability comparison: Simplex vs. Bellman-Ford

Figures 7.1 and 7.2 show the execution times of the Simplex on the original system vs. Bellman-Ford (BF) on a UTVPI approximated system[25].

- The former uses the quite well used dual-simplex implementation of PIP [Fea88] (similar [Tod02] to many LP solvers like CPLEX [Bix02]) called on a (non-parametric) *rational polyhedron for feasibility*.
- The latter is our implementation of a standard Bellman-Ford algorithm on the constraint-graph of almost double size obtained from the MUTVPI-polyhedron (the monotonized-UTVPI polyhedron) called for testing the presence of *negative weight cycles*.

The input programs `matmul` and `seidel` are the same as in Figures 2.2 on page 12, and each call of `auto_trans` of PLuTo to find schedule hyperplanes has tens of calls of the above variety, which explains the difference in scales between current figures (Figure 7.1 and 7.2) and in Figure 2.3 on page 13. Prior to solving either of these, the duplicates were eliminated using a syntactic matching as in PLuTo. Even on the systems with duplicates, the improvements were similar, though more prominent.

[25]The UTVPI system used in these scalability experiments is obtained as a result of a *clustering* algorithm. This particular algorithm, called PD4, is a part of our methods to improve feasibility of the under-approximation. These are explained in detail in Section 8.4.2 on page 110

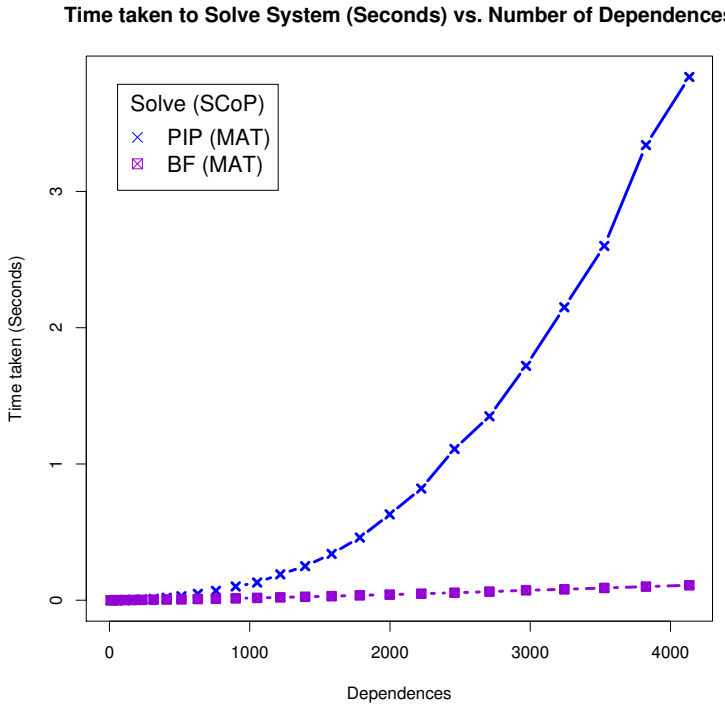


Figure 7.1: Simplex (PIP) vs. Bellman-Ford (BF) for unrolled examples of `matmul` (`ITR-MATMUL`)

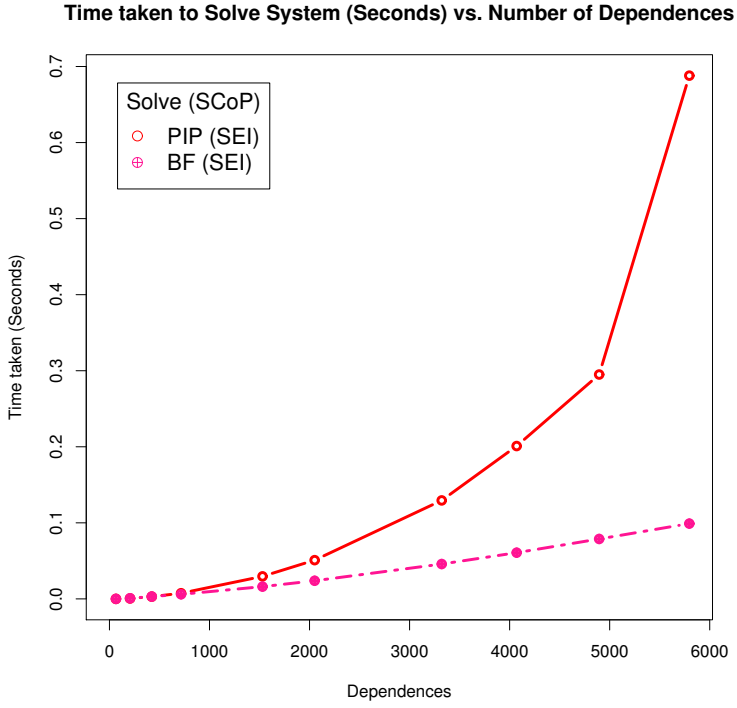


Figure 7.2: Simplex (PIP) vs. Bellman-Ford (BF) for unrolled example of `seidel` (`ITR-SEIDEL`)

		Perf. Comparison (Seconds)			
Benchmark	Orig	Par cur	Par new	Til cur	Til new
gemver	0.31	0.15	0.15	0.15	0.15
mvt	1.40	0.27	0.28	0.42	0.43
seidel	11.8	3.6	3.6	11.5	11.5

Table 7.3: UA Code Performance. The *Orig* column indicates the execution times of the particular unoptimized benchmark. Columns *Par cur* and *Til cur* respectively indicate the execution times of the parallelized and tiled programs obtained by running PLuTo on the benchmarks. Columns *Par cur* and *Til cur* respectively indicate the execution times of the parallelized and the tiled programs when the overall Farkas system is replaced by the TVPI under-approximated system obtained as a result of running the LP-independent method.

It can be seen from the graphs that for their respective inputs, BF (worst-case $\mathcal{O}(\hat{s}^2mn) \approx \mathcal{O}(|E||V|) \approx \mathcal{O}(|V|^3)$) is asymptotically as well as empirically faster than PIP's Simplex (observed $\mathcal{O}((m+n)mn) \approx \mathcal{O}(|V|^5)$) (not counting that it is weakly polynomial time), meeting the scalability challenge. The regression coefficients (linear R^2 using the R statistical package [R]) for the above formulae are around 99.9%. The curves for BF appear linear because the x-axis counts the number of dependences $|E|$, which is practically $\mathcal{O}(|V|^2)$. Though the memory improvements are not shown in the above graphs, the linear memory (worst-case $\mathcal{O}(m+n)$) of BF is considerably lighter[26] than the quadratic memory (worst-case $\mathcal{O}(mn)$) of Simplex.

The relatively lesser magnitudes and hence lesser improvements of `seidel` when compared to `matmul` could be attributed to the fact that while the former has just uniform dependences which are easily amenable to the Gaussian eliminations of PIP, the latter has more affine dependences and also induces many more Farkas multiplier variables. From the magnitude of each of these curves and the relative improvements for these two example programs, it is safe to say that sub-polyhedral techniques would provide more significant improvements for programs with predominant affine dependences than for ones with predominant uniform dependences.

7.5 UA generated code performance

We are referring here to Table 7.3, limiting ourselves to a subset of the YY cases in the previous table that our current implementation could handle. In each case, we replaced the original system(s) by the approximated TVPI ones obtained by the independent LP method. The cost function was unchanged and the solution was found using PIP. It can be seen that performance gains closely match the default polyhedral method in PLuTo, despite the approximation taking place. The impact of YN approximations on PLuTo's effectiveness is yet to be studied, but we express some hope on PLuTo's loop distribution heuristic to break infeasible systems.

Though the experiments in Table 7.3 are limited in number, obtaining these itself involved a considerable engineering effort in a research prototype tool like PLuTo. This was partly because of our explicit

[26]Initial evaluation on a Quad-Core Dell desktop with 16GB memory using the `top` program of linux showed up to 40% memory occupied by PIP program for an ITR-MATMUL program of around 4000 dependences. The memory occupied by BF was always less than 10%. Though more exhaustive experiments are needed, we believe that these differences are significant.

decision to not invest much further on the experimental front. We also chose to defer further evaluation until PLuTo is adapted to facilitate the integration of per-dependence approximations, and some engineering issues with the interpretation of PIP results could be circumvented. Examples of these issues include methods to interpret the schedules obtained by the Bellman-Ford algorithm to obtain permutable hyperplanes, and ways to integrate them with the rest of the scheduler, as well as need to reengineer PLuTo's backend—for example in function like `pluto_detect_transformation_properties`—so that the per-dependence systems can be replaced by their under-approximated systems—perhaps obtained by the one-shot algorithm—in a streamlined fashion, along with calling the necessary routines like duplicate elimination and feasibility testers at proper check-point locations. More details on some of these are discussed in the perspectives section at the end of this chapter.

7.6 UA verification

We used PIP and PolyLib matrices which were compatible even with other libraries like FMLib. This opened up some possibilities of using available routines from any of these libraries for checking feasibility of the systems. PolyLib has a routine, `PolyhedronIncludes`, that takes two Polyhedra P and Q and returns whether $P \subseteq Q$. It needs P and Q in generator representation and checks for inclusivity of each of the vertices of P in Q. Obviously, such a method does not work the large schedule matrices that PLuTo handles.

Another option is to verify the under-approximation using PIP while both P and Q are still in constraint form. This would involve framing an LP call to verify that none of negations of each of the constraints of P have a point in Q; this can be done by ensuring that each of the m systems $(P_i)^c \cap Q$ is empty, where $(P_i)^c$ is the negation of the ith constraint $(1 \leq i \leq m)$ of P. Such a process can iteratively be applied to each of the constraints of P and hence can be done in polynomial time, though it involves m LP calls.

Implementation wise, the simplest option is to test for emptiness of the under-approximation by testing for a feasible point in it using PIPLib, with the proof of the UA algorithms taking care that the UA constructed is a proper UA. We have done this.

7.7 Conclusions and Perspectives

Conclusions and contributions. In this chapter we studied the implementations of the algorithms studied in the previous chapter. The methods show asymptotic improvement, scale well and show a positive answer to the scalability challenge in polyhedral scheduling. We have also shown an evidence of the execution of the optimized program.

Perspectives and discussion

Polyhedral characteristics. The polyhedral characteristics in Table 7.1 are unique in many ways because they expose the relative simplicity of the polyhedra that arise in polyhedral compilation. For example, the sizes of the benchmarks (L, S and D) with respect to the sizes and numbers of the polyhedra, the separation of dependence polyhedra from the Farkas polyhedra, the study of sparsity coefficients etc., validate some folklore hunches: such as dependence constraints are mostly (U)TVPI polyhedra, Farkas polyhedra have low sparsity, etc. Our work supersedes a previous such study by Pugh [Pug92] and Maydan et al. [MHL91]. It is hoped that our study will motivate more characteristics from more recent versions of benchmarks like PolyBench 3.0, including the crucial and possibly hard to characterize exact average case empirical complexity of simplex algorithm on these varieties of polyhedra bringing to closure the Footnote 11 on page 47.

Feasibility of UA. The result implementation of the algorithms that are introduced in Chapter 6, namely, the median and LP-independent methods, have been shown in Section 7.3. These deal only with a single polyhedron, and a limited per-dependence implementation case. More powerful techniques, like clustering methods, along with a discussion of the feasibility preservation properties, suited for the interesection of per-dependence polyhedra are discussed in Chapter 8.

Scalability comparison. One possible limitation of the numbers in Tables 7.2 and 7.3 is the lack of timing improvements for the benchmarks in PolyBench 2.0. The simple reason for this is that the number of dependences in these benchmarks are quite small to show *significant* relative improvements. To notice this, we can compare the number of dependences in PolyBench 2.0 as shown in Table 7.1 on page 91 (the #D columns) with the number of dependences in Figure 2.3 on page 13 (also reflected in Figure 7.1 on page 93). The former are mostly in tens, while the latter are in thousands. It is safe to think that when the number of dependences crosses 500, the improvements of our methods would be more pronounced. With the reasons explained in Chapter 2 on page 7, this should not be considered as a limitation; some benchmarks in the later released PolyBench 3.0—like `pca` and `tce`—are already large enough and have several hundred dependences. Also, as indicated in Chapter 2, LLVM/Polly has a more severe scalability problem than PLuTo. For PolyBench 2.0 examples, the compilation times are in *tens of seconds* which are unacceptable even for a conventional compiler. Furthermore, for scalability comparisons about asymptotic improvements like ours, we believe that curves like Figure 7.1 on page 93 and Figure 7.2 on page 93 make the strongest case.

Generated code performance. We have just shown a proof of concept of the executed code performance in Table 7.3 on page 94. The generated code performs well when compared to the original code that PLuTo generates when compared with PIP. The above table is only partial and should be augmented with more analysis: code performance for per-dependence methods (and their interaction with YY and YN cases), and the number of final timings of the executed code in Table 7.3 on page 94. We believe there are valid reasons for both of these in both theoretical and practical sense.

More advanced techniques for interpreting schedules. One theoretical reason is the way PLuTo currently makes a call to PIP for obtaining the schedule. PLuTo constructs an overall Farkas polyhedron and makes a `lexmin` call to PIP on it, while the cost function of a UTVPI polyhedron solved for a feasible point using Bellman-Ford algorithm is `max`. The dual LP program of Bellman-Ford consequently has `min` in the objective function, but the constraints are column-wise UTVPI, not row-wise. This problem could be overcome by using Johnson's algorithm [CSRL01], after running a Bellman-Ford on the constraint graph.

Briefly, Johnson's algorithm for finding All Pair Shortest Paths begins with an application of Bellman-Ford algorithm on an augmented graph which has an additional source vertex connected to all the other vertices by zero-weight edges. It also involves an application of Dijkstra's algorithm on each of the vertices. But, as this latter algorithm can only be applied when the edge weights are non-negative, it constructs a new graph—obtained as a result of a *reweighing* step—using the original graph and the solution given by Bellman-Ford. The overall algorithm is not much costlier than Bellman-Ford; it takes an overall time of $\mathcal{O}(|V|^2 \log |V| + |V||E|)$, which is comparable to $\mathcal{O}(|V||E|)$ of the latter.

The reweighing step of Johnson's algorithm is equivalent to interpreting the shortest paths solution returned by Bellman-Ford and hence which can be used to *apply* the schedules in PLuTo to find permutable hyperplanes. It can even be thought of as skewing or rotating the axes accordingly.

Schedule interpretation of permutable hyperplanes. Closely related to the above is the interpretation of permutable hyperplanes as well as fusion heuristics implemented in PLuTo for application of schedules. The PLuTo algorithm of Bondhugula et al. [BHRS08] needs as many independent hyperplanes as there are dimensions for each statement in the input program. Hence it accumulates the best solution available in an iterative fashion all the time ensuring that linear independence [Pen55] is satisfied till all the dependences are satisfied. The above method we suggested—of obtaining a feasible point using Bellman-Ford and applying it using Johnson's algorithm—should be tuned to this process of PLuTo, and we leave this as an open problem for now.

Some implementation issues

Advanced algorithms to remove duplicates. Our current method to eliminate duplicate constraints is a simple syntactic one employing lexical comparison of the constraints. We improved from the one implemented in PLuTo. For constructing a non-duplicate system from an input system of size $m \times n$, it takes $\mathcal{O}(mn \log m)$ time, with sorting of the constraints (using `quicksort` routine) dominating the complexity. For general polyhedra, this cost can obviously not be improved any further. But as indicated in the earlier

chapter, this cost can be reduced to *linear* time for UTVPI system using radix sort based schemes [AN98], and such an improvement could make a significant difference. We think that it is imperative that a UTVPI approximation employing Bellman-Ford algorithm should employ the latter because of the comparable $\mathcal{O}(mn)$ complexity of Bellman-Ford.

Integer vs. rational solutions. The traditional method is to solve a rational LP problem on a per-dependence basis; this results in rational coefficients in the homogenizing dimension as discussed in Section 6.3.1 on page 73. This method can be improved by solving an ILP formulation on a per-dependence basis; this results in integer coefficients in the homogenizing dimensions as discussed in Section 6.7.5 on page 80. This improvement also comes at a price with respect to scaling of the coefficients. Namely, choosing to have integer homogenizing dimension values raises two issues: increase of homogenizing dimension coefficients in the higher-dimensional intermediate system $\mathcal{S}_{HD}$, and the corresponding increase of the homogenizing dimension values in the resultant approximated system.

The solution to the integer program may not be the same as the one obtained by running a rational program and simply scaling it up. Also, the choice of integer solution may lead to insufficiency of normal (32-bit) precision offered by PIP [Fea88]. With all the examples in PolyBench 2.0, we did not need to use more precision than 32-bits for integers. However, this may not always be the case. A wider precision like 64-bits, or even the use of multi-precision libraries like GMP, may be needed to solve the intermediate systems and store the resultant homogenizing dimension values.

It has also to be noted that for both TVPI as well as UTVPI sub-polyhedra, the algorithms to solve the overall system P—Hochbaum-Naor for TVPI polyhedra and Bellman-Ford for UTVPI polyhedra—are *strongly polynomial* algorithms. So, this increase in precision will not affect the overall complexity asymptotically. It will however affect it, though only by a constant factor. Such is not the case with ILP formulation on the overall polyhedron, which potentially takes exponential time and depends on the bit size as well, because of its strong NP-Completeness.

Bounding box for pseudo-parametric dimensions. The pseudo-parametric variables, as defined in our LP formulations, necessarily have a non-trivial lineality space: the variables are unbounded in either direction because their constraints have equalities. When asked to provide a `lexmin` on such unbounded polyhedra (especially ones with non-trivial lineality space), PIP returns unbounded values by encoding them as rational numbers with 0s in the solution's denominators.[27] The corresponding numerators also do not respect the pseudo-parametric constraints, as they were never expected to do so.

This issue could be overcome by a *minor* trick; bounding the HD system in a fixed size box, by adding additional constraints to the pseudo-parametric variables: $-K \leq p_1, \ldots, p_m \leq K$, where $2K$ is the width of the bounding box. In this way, the lineality space issue would not be an issue in PIP and the solution values would respect the constraints. As the homogenizing dimension values could be either positive or negative, the p-variables need to be bound in both directions with the bounding box needing to span either directions of the origin.

[27] We thank Cédric Bastoul for helping to decode this apparent "error" of PIP.

The size of the bounding box however, is not straightfoward to obtain: it needs some empirical testing and could be even be benchmark dependent. Too small bounding box values, like $K = 1$ may lead to overflows when solving the resultant system because their non-homogenizing dimension coefficients have large values[28]. On the other hand, too big bounding box values like $K = 100$, have the detrimental effect of the overflow of the intermediate system. By experimentation, we have found that any value of K less than 10, for example like $K = 8$, is a good value for benchmarks in PolyBench 2.0.

In this chapter, we studied the implementation of the algorithms studied in earlier chapters. In the next chapter, we will study a program theoretic application of the under-approximation.

[28]This is because PIP does not take in rational constraints: a constraint like $2x + 3y \leq \frac{1}{10}$, where the $\frac{1}{10}$ is obtained as homogenizing dimension value from solving the higher dimensional system should be re-encoded as $20x + 30y \leq 1$.

Chapter 8

Applications to Loop Transformation Problems

In the previous chapters, we have mainly focused on Farkas polyhedra arising in Feautrier's scheduler and its application in FCO scheduling for tiling. One important strength of our under-approximation framework is its ability to trade precision for scalability using (U)TVPI approximations of polyhedra. Scalability arises from the excellent worst-case complexities of sub-polyhedral operations, and the ability to largely implement the approximation as an independent optimization problem for each dependence-edge. Because of the strength of these properties in our approach, it happens to be general enough to be applied to other compilation problems, related and unrelated to scheduling. In this chapter, we will see a variety of compilation and loop-transformation problems which could benefit from our framework.

8.1 Introduction

The following is a brief overview of this applications chapter.

LP formulations for affine loop transformations. In Section 8.2 we apply the approximation to shifting 1-d loops for pipelining as formulated by Calland et al. [CDR98] and for compaction as formulated by Darte and Huard [DH00a]. This formulation is particularly interesting because of the use of a variation of min-cost flow algorithm by the authors, whose dual program has TVPI constraints. This is followed by DAG scheduling problems in circuit synthesis [CZ06, JZPC08, CLZ09] which use formulations with UTVPI constraints and Total Unimodularity results.

In Section 8.3, this is followed by a brief summarization of Darte and Vivien's scheduling algorithm [DV97a] and a suggestion to apply an approximation. This is followed in Section 8.4 by more general affine scheduling techniques that are based on application of Farkas lemma; namely Feautrier's framework [Fea92a, Fea92b] in Section 8.4.1, and Forward Communications Only (FCO) framework [GFG02, BHRS08] in Section 8.4.2. For approximating the latter, we also discuss some practical feasibility enhancing techniques: namely, clustering of the constraints. Since the PLuTo framework already provides a way

in which these could be evaluated, we show the experimental results of implementing the above methods in Section 8.4.3. These experimental results are extensions of the ones shown in Chapter 7.

ILP and mixed $0-1$ ILP formulations of loop transformations. Within the loop transformations themselves, some problems have been shown to be NP-Complete, with their algorithms consisting of ILP or mixed-0-1-ILP formulations. We discuss a sample of these problems and show how our approximating frameworks can be used for obtaining heuristics. These techniques use the approximation technique introduced in Section 6.7.5 on page 80.

For doing the above, first, in Section 8.5, we cover the ILP formulation of multi-dimensional shifting as formulated by Darte and Huard [DH02]. Then in Section 8.6, we explain how some formulations in Darte and Huard's array contraction results for shifting or fusion [DH05] can utilize the approximation framework for obtaining approximate solutions. Since these latter formulations also subsumes some previously other formulations like [MS97], we summarize them as well.

A framework for approximation of loop transformations. In Section 8.7, we formulate a general framework for approximation, which summarizes the above formulations and our methods to approximate the techniques.

Implication of UTVPI approximation on code generation. The above will be followed by briefly discussing in Section 8.8 how the above Totally-Unimodular results hinted in Section 6.7.5 on page 80 can possibly be used to generate simpler code.

In Section 8.9, we discuss some conclusions and perspectives.

8.2 Shifting $1d$-loops for Pipelining and Compaction

Since the general resource constrained optimal cyclic scheduling problem is known to be NP-Hard [HM94], the problem has traditionally been decomposed into a two phases: first solving a cyclic scheduling problem without resources constraints where the shifting factor is found using retiming techniques, followed by solving an acyclic graph scheduling problem taking into account the resource constraints using traditional list scheduling techniques.

To solve the cyclic scheduling (decomposed software pipelining) problem, Calland et al. [CDR98] give an LP formulation whose constraint matrix is built from incidence matrices of the original and a retimed dependence graph, and prove it to be a Totally Unimodular (TU) matrix. The following is their LP formulation:

$$\begin{cases} \min \sum_{e \in E} \delta(e) & \\ 0 \leq \delta(e) \leq 1 & \text{for all e} \in \text{E} \\ q(v) - q(u) + d(e) + \delta(e) \geq 1 & \text{for all u, v} \in \text{V; } e = (u, v) \in E \\ q(v) - q(u) + d'(u, v) \geq 1 & \text{for all u, v} \in \text{V} \\ & \text{such that } \Delta(\text{u, v}) > \Phi_{\text{opt}} \end{cases} \tag{8.1}$$

It should be noted that in the above LP formulation, $\mathbf{q}$ and δ are the variables. The former $q(v)$ standing for the rational retiming applied on vertex v, and the latter $\delta(e)$, which is a binary decision variable, for whether the edge e is selected or not as a result of the retiming. Also, the functions $\mathbf{d'}$ and Δ are defined over pairs of vertices using the Lieserson-Saxe retiming algorithm [LS91], and are found out prior to the above LP formulation by solving an All Pairs Shortest Paths problem on G.

The above LP formulation can be written in matrix-constraint form as the following:

$$\min \left\{ (0,1) \begin{pmatrix} q \\ \delta \end{pmatrix} \middle| \begin{pmatrix} \mathbf{0} & -I \\ \mathbf{0} & I \\ C & I \\ C' & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{q} \\ \delta \end{pmatrix} \leq \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{d-1} \\ \mathbf{d'-1} \end{pmatrix} \right\} \tag{8.2}$$

where C is the transpose of the $|V| \times |E|$ size incidence matrix of G, C' is the transpose of the incidence matrix of G' whose edges are the pairs (u, v) such that $\Delta(u, v) > \Phi_{opt}$, and I is the $|E| \times |E|$ identity matrix. Also, the vector $\mathbf{d-1}$ (resply. $\mathbf{d'-1}$) is of size $|E| \times 1$ ($|E'| \times 1$, where $|E'|$ is the number of edges in G') and constructed with the edge weights of the original graph G (resply the retimed graph G').

It may seem that solving the above set of inequalities in (8.2) may need an ILP solver because of the binary nature $\{0, 1\}$ of the δ-variables, but Calland et al. give the following result, which proves the polynomial nature of above problem:

Theorem 8.2.1 [TU of Calland et. al's LP formulation]. The constraint matrix of the above formulation (Equation 8.2) is a Totally-Unimodular (TU) matrix.

For proving the above we need the following characterizations of TU matrices:

Lemma 8.2.2 [Characterization of TU matrices]. Let A be a matrix with entries from $\{0, \pm 1\}$, then the following are equivalent:

1. A is totally unimodular, i.e., each square submatrix of A has a determinant $\{0, \pm 1\}$;
2. [Ghouila-Houri characterization of TU matrices [GH62]] Each collection of rows of A can be split into two parts so that the sum of rows in one part minus the sum of the rows in the other part is a vector with entries from $\{0, \pm 1\}$.
3. [Raghavachari's constructive characterization of TU matrices [Rag76]] If a row (or column) of A has at most one non-zero element, then A is TU iff the matrix obtained by removing the particular row (or column) is TU.

Proof of Theorem 8.2.1: The sets of rows $\begin{pmatrix} 0 & -I \end{pmatrix}$ and $\begin{pmatrix} 0 & I \end{pmatrix}$ are obviously TU. The sub-matrices C and C' are TU because they are incidence matrices of directed graphs, and the latter are TU because they are a sub-class of network matrices. So, the sub-matrix $\begin{pmatrix} C' & 0 \end{pmatrix}$ is TU as well. For seeing that the remaining rows $\begin{pmatrix} C & I \end{pmatrix}$ are TU, notice that the I matrix adds exactly one additional 1 to each of these rows of the matrix. Any square sub-matrix which includes any row from these sets of rows will have determinant from $\{0, \pm 1\}$ because of Ghouila-Houri theorem. When applying the above theorem, it note that Schrijver [Sch86] uses the transposes of the matrices referred to here.[29] ■

UTVPI-UA of the above formulation. It can be noticed that as given, the formulation in (8.2) has at most 3 non-zero elements per row in the $\begin{pmatrix} C & I \end{pmatrix}$ rows and is clearly not a (U)TVPI polyhedron. But, it can benefit from our framework as the constraints are constructed on a per-dependence-edge basis, and the constraint matrix elements belong to the set $\{0, \pm 1\}$. To do the above, three strategies can be used to obtain the (U)TVPI approximation:

$$P_e^0 = \begin{pmatrix} C(e) & I(e) \end{pmatrix} \begin{pmatrix} \mathbf{q} \\ \delta \end{pmatrix} \leq (d(e) - 1)$$

$$P_e^1 = \begin{pmatrix} 0 & -I \\ 0 & I \\ C(e) & I(e) \end{pmatrix} \begin{pmatrix} \mathbf{q} \\ \delta \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ d(e) - 1 \end{pmatrix}$$

$$P_e^2 = \begin{pmatrix} 0 & -I \\ 0 & I \\ C(e) & I(e) \\ C' & 0 \end{pmatrix} \begin{pmatrix} \mathbf{q} \\ \delta \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ d(e) - 1 \\ \mathbf{d'} - \mathbf{1} \end{pmatrix}$$

In the above, the P_e's are the per-dependence-edge polyhedra that are being sent to the Under-Approximation engine for approximation. The $\left[\begin{pmatrix} C(e) & I(e) \end{pmatrix}, d(e) - 1 \right]$ row corresponding to P_e^0-polyhedron

[29] After writing the above proof, a much simpler and alternative proof for the above was noticed by the author. It uses the above mentioned Raghavachari's constructive characterization of TU matrices [Rag76], listed as Lemma 8.2.2.3.

is the actual per-edge constraint, which can be annotated with the additional rows as a way of augmentation.

In P_e^0 strategy, the approximation is done only for P_e^0, while in the P_e^1 strategy, the bounding box constraints are added to the P_e^0 constraints, and finally for P_e^2 strategy, the $\left[\begin{pmatrix} C' & 0 \end{pmatrix}, \mathbf{d'} - \mathbf{1}\right]$ constraints are added.

Feasibility preservation of UA systems. It is important to notice that the following matrix is always feasible because it encodes the identity schedule:

$$P_{base} = \begin{pmatrix} 0 \\ 0 \\ C(e) \\ C' \end{pmatrix} (\mathbf{q}) \leq \begin{pmatrix} 0 \\ 1 \\ d(e) - 1 \\ \mathbf{d'} - \mathbf{1} \end{pmatrix} \equiv \begin{pmatrix} C(e) \\ C' \end{pmatrix} (\mathbf{q}) \leq \begin{pmatrix} d(e) - 1 \\ \mathbf{d'} - \mathbf{1} \end{pmatrix}$$

The presence of δ-variables is forces us to answer the feasibility question. We also have to note that in this particular formulation of Calland et al.'s, the above formulation P_{base} is already UTVPI. But, this discussion is relevant because we see this Calland et al.'s formulation as a sub-case of more powerful LP/ILP formulations, where the UTVPI-ness of the matrices in the absence of variables equivalent to δ-variables of Calland et al. is not necessarily guaranteed.

P_e^0 can be approximated by the median method and so can be P_e^1 and P_e^2. Each of these can also be approximated using the one-shot method. The latter can also be done on even the entire even P_e as well resulting in an offline online strategy.

This original schedule can be added into the rest of the constraints as a means of seeding: namely, by adding some solutions of P_{base} to each of P_e^0, P_e^1 or P_e^2. Then there will be some points which are common to each of the per-dependence methods. Once this seeding is done, the Under-Approximation can happen on a per-dependence basis. The following lemma directly follows.

Lemma 8.2.3 [Seeding Lemma] The Under-Approximation will always preserve the canonical schedule.

The trick for the above proof is to use the one-shot approximation technique, which always guarantees the feasibility of the UA. The resulting UA will have the canonical schedule and be UTVPI as well.

In the rest of this section, we cover two additional applications that use a very similar framework.

Darte-Huard's Loop Shifting for compaction. Darte-Huard's loop shifting for compaction [DH00a] uses a very similar framework to the above paper, where an LP formulation which is a variation of a min-cost flow problem is used with the goal of finding a valid retiming such that the number zero-weight edges in the retimed graph is mimimized. Just as in the formulation of Calland et al., the zero-weight edges correspond to the number of active edges in the ith iteration, and hence minimizing them corresponds to a per-iteration constraint graph which is more amenable to the list scheduling process that follows. Darte-Huard propose an purely graph theoretic algorithm using out-of-kilter techniques for solving min-cost flow problem, resulting in an overall algorithm with $\mathcal{O}(|E||V|^2)$ worst-case complexity. Though the

above technique is by itself strongly polynomial, we propose that the above algorithm could as well benefit from a UTVPI approximation resulting in a lower complexity strongly polynomial time algorithm.

DAG scheduling in circuit synthesis. In a series of papers, Cong et al. [CZ06, JZPC08, CLZ09] propose scheduling using difference constraints and Totally Unimodular concepts. More specifically, Cong and Zhang [CZ06] propose scheduling of DAGs using difference constraints so that behavioural synthesis of hardware (#States, #Cycles etc.) can be optimized. Jiang et al. [JZPC08] extend the above work and propose DAG scheduling for low-power optimization in behavioural synthesis. They prove that the scheduling constraints are rational difference constraints $x_i - x_j \leq c; c \in \mathbb{Q}$ and hence TU, resulting in polynomial time scheduling problem. Finally, Cong et al. [CLZ09] extend the above case for integer difference constraints of the form $x_i - x_j \leq c; c \in \mathbb{Z}$ with addition of what they term as soft-constraints (ones that can be broken), again with the overall goal of low-power behavioural synthesis. The overall system is again a difference constraint matrix and hence satisfies TU properties.

Cyclic-scheduling techniques being more general than DAG scheduling. It can be seen that the above circuit synthesis formulations [CZ06, JZPC08, CLZ09] are a sub-problem of Calland et al's [CDR98] and Darte-Huard's [DH00a] cyclic scheduling formulations. This is because an algorithm for cyclic scheduling can very easily be used for solving a DAG scheduling problem: for example, by creation of a pseudo-loop around the body. Even the use of TU matrices in either of these formulations is very similar. Also, the LP formulations from the above circuit synthesis are very similar to the list scheduling techniques of Hanen-Munier [HM01, HM94] (Chapter 2 of [DRV00]). These latter results have greedy algorithms using which can be seen to subsume the above mentioned results from circuit synthesis.

Relationship of UA framework to the above sets of problems. Since we have shown in the above section that Calland et al's formulation can be approximated using our UA technique, the above circuit synthesis problems can also be solved with the same cubic-time complexity of Bellman-Ford.

Similarity of problems from different areas. The above similarity of our concepts to problems from another area (namely circuit synthesis) is very encouraging, but we should also note that the problem of scheduling is very vast with many applications from fields quite unrelated to each other. In fact, the concepts of scheduling graphs with delays either on vertices or on edges, with or without cycles, and with the vertex/edge weights being uni/multi-dimensions is extremely powerful to encode not just from loop-optimization or circuit synthesis, but even from other areas such as job-shop or task scheduling. Due to the above reason, it is likely that the concepts of Totally Unimodularity get reused many times. The formulation by Durr and Hurand [DH06] for list scheduling can be considered as another instance of a scheduling application using TU concept.

8.3 Darte-Vivien's PRDG Scheduling

Darte-Vivien's scheduling algorithm [DV97a] builds on their previous work which revisits the decomposition of Karp-Miller-Winograd (KMW) on Systems of Uniform Recurrence Equations (SUREs) [DV95]. Their works bring out the connection between the KMW decomposition algorithm to determine the computability of SUREs and polyhedral scheduling algorithms.

Darte-Vivien show that for dependence graphs annotated with direction vectors (multi-dimensional dependence vectors whose components are from $\mathbb{Z} \cup \{+, 0+, -, 0-, *\}$), an *apriori uniformization step* can transform these dependence graphs into an equivalent Polyhedral Reduced Dependence Graph (PRDG) which encodes the polyhedral representation of these dependences. Also, the *computability test of KMW* can directly be used to find schedules of this PRDG with uniform dependences. Further, there exists a simple linear programming *duality relationship* between the computability and the PRDG scheduling algorithm.

Feautrier's algorithm [Fea92a, Fea92b] is generally accepted to be the most powerful framework and algorithm to detect and extract parallelism. Also, Darte-Vivien's scheduling algorithm accepts uniform dependences which are more restricted than the affine dependences accepted by Feautrier's algorithm. However, it is rather surprising to note that if optimality is defined such that it involves detecting the amount of parallelism in the particular encoding accepted by the parallelization algorithm, Darte-Vivien's algorithm is optimal [Viv03]. It should also be noted that as the scheduling algorithm of Darte and Vivien takes in only uniform dependences as a result of the uniformization step, it does not need to use the affine form of Farkas lemma as used by Feautrier. But, the algorithm still needs duality techniques when constructing the schedule vectors obtained as the result of KMW-decomposition.

In this section we cover neither the computability result of KMW nor the uniformization step of Darte-Vivien, instead referring to the excellent coverage in [DRV00, Ch. 4 and 5]. The following is the main result of Darte-Vivien:

Theorem 8.3.1 [Darte-Vivien's Computability and Scheduling]. Let $G = (V, E, \mathbf{w})$ be a Polyhedral Reduced Dependence Graph (PRDG) with $\mathbf{w} : E \to \mathbb{Z}^d$ be the multi-dimensional edge weights (or dependences). The computability test of the above PRDGs can be adapted to find linear schedules using LP duality techniques. More precisely, the following are equivalent:

$$\min \left\{ \sum_e v_e \,|\, \mathbf{q} \geq 0;\ \mathbf{v} \geq 0;\ \mathbf{q} + \mathbf{v} \geq \mathbf{1};\ C\mathbf{q} = \mathbf{0}; W\mathbf{q} = \mathbf{0}; \right\} \begin{array}{c} \text{Computability} \\ \text{Formulation} \end{array} \left[P^{DV}_{COMP}\right] \tag{8.3}$$

$$\iff \min \left\{ \sum_e v_e \,|\, \mathbf{q} \geq 0;\ \mathbf{v} \geq 0;\ \mathbf{x} \geq 0;\ \mathbf{q} + \mathbf{v} = \mathbf{1} + \mathbf{x};\ C\mathbf{q} = \mathbf{0};\ W\mathbf{q} = \mathbf{0}; \right\} \begin{array}{c} \text{Computability} \\ \text{Formulation} \end{array} \left[P^{DV}_{COMP}\right] \tag{8.4}$$

$$\equiv \max \left\{ \sum_e z(e) \,|\, \mathbf{0} \leq \mathbf{z} \leq 1;\ X\mathbf{w}(e) + \rho_v - \rho_u \geq \mathbf{z} \right\} \begin{array}{c} \text{Scheduling} \\ \text{Formulation} \end{array} \left[P^{DV}_{SCH}\right] \tag{8.5}$$

where C is the $|V| \times |E|$ size incidence matrix of G, $W = \left[\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{|E|}\right]$ is the $n \times |E|$ size dependence matrix whose columns are the weights of edges of G. The formulations in equations (8.3) or (8.4) are essentially the same except for the additional (slack) variables $\mathbf{x}$. Equations (8.4) and (8.5) are of course related by duality.

Darte-Vivien suggest a KMW style decomposition algorithm to determine the schedule of the input PRDG. Instead of determining the computability of the input SURE, Darte-Vivien decomposition algorithm returns the schedule for PRDG G, with $(X_S^1, \rho_S^1), (X_S^1, \rho_S^2), \ldots, (X_S^{d-1}, \rho_S^{d-1})$ returning the d-dimensional schedule for each vertex of PRDG, with the some number of dependences satisfied at each level of the decomposition, inducing smaller sub-graph $G' \subset G$, where G' is generated by all the edges of G that belong to at least one multicycle with zero-weight.

They show also show the following property of the graph G':

Lemma 8.3.2 [Darte-Vivien's schedules and Strong and Weak Separability]. Let G' be the subgraph of G induced by zero-weight multicycles. The edges of G' can be characterized by the following property

$$\begin{cases} \text{Weakly Separating Hyperplane} \;\; e \in G' : & Xw(e) + \rho_v - \rho_u = 0 \\ \text{Strongly Separating Hyperplane} \; e \notin G' : & Xw(e) + \rho_v - \rho_u \geq 1 \end{cases} \tag{8.6}$$

Approximation of Darte-Vivien's formulations. Before designing (U)TVPI-UA strategies specific to Darte-Vivien's scheduling algorithm, we need a few observations. Firstly, as pointed above, the input of Darte-Vivien is (polyhedral) direction vectors and hence by itself can be considered as an approximation of the exact (affine) polyhedral dependences of Feautrier [Fea91]. Hence, (U)TVPI-approximation applied to Darte-Vivien's algorithm for purposes of scalability needs to have a good guarantee of feasibility. Secondly, there already exists *some* practical evidence [Mei96] to show that for dependences which are amenable to both Darte-Vivien's algorithm as well as Feautrier's, the former is more scalable than the latter. Thirdly, the approximation can proceed using either of the formulations in (8.3) or (8.5). Obviously, in the former, it needs to be a OA, while in the latter it needs to be a UA of the overall polyhedra. Finally, the matrix C, which is used in the computability formulation of Darte-Vivien is already a UTCPV matrix because it is the incidence matrix of a directed graph with each column having exactly one $+1$ element, and exactly one -1 element. Also, the W matrix has tendency to have low *column-wise sparsity* with elements most likely from $\{0, \pm 1, \pm 2\}$. These can be further transformed into a *unit-PRDG matrix* W, whose elements are from $\{0, \pm 1\}$ with this transformation involving addition of pseudo-nodes much similar to Darte-Vivien's uniformization step.

Approximation strategies for Darte-Vivien. The following are suggestions of some approximation strategies for Darte-Vivien.

1. **[DV-UA1] Enumeration by fixing the z-variables:** The $\mathbf{z}$-variable in (8.5) can be redefined to be a binary decison variable, i.e., $\mathbf{z} \in \{0, 1\}$, with the binary values defining whether the particular edge z_e is selected at that particular level of decomposition (1) or not (0). Each of the

resultant polyhedra $P^{DV}_{FIXZ}(X,\rho) = P^{DV}_{SCH}(X,\rho)|\mathbf{z}$ can be approximated using one-shot strategy UA($P^{DV}_{FIXZ}(X,\rho)$). Note that this latter polyhedron is always a feasible polyhedron and results in a valid under-approximation. For example, for $|E| = 3$, this yields three polyhedra $P^{DV}_{SCH}(X,\rho)|z_1$, $P^{DV}_{SCH}(X,\rho)|z_2$ and $P^{DV}_{SCH}(X,\rho)|z_3$. Once this approximation is done, so as to maximize the number of edges satisfied at the lowest depth, the edges that are left can be maximized. This can proceed in a decreasing fashion for $|E|, |E|-1, \ldots 2, 1$ edge.

2. **[DV-UA2] Per-Dependence UA strategy:** The per-dependence polyhedron can be constructed as
   $P_e = \{\mathbf{0} \leq \mathbf{z} \leq 1;\ X\mathbf{w}(e) + \rho_v - \rho_u \geq \mathbf{z}\}$ which can be approximated using the one-shot approach. The overall collection of Under-Approximations UA(P_e) can be solved. The above can be run on a cluster (bigger collection of dependences) of dependences, like per-SCC component as well.

3. **[DV-UA3] OA of the Computability constraints:** For this, we need to notice that between the two LP formulations for computability, the main constraints are $\{C\mathbf{q} = \mathbf{0}; W\mathbf{q} = \mathbf{0};\}$. These could be approximated into UTCPV constraints as well.

Each of the above could happen on a strongly connected component of the PRDG. The theoretical feasibility results of the above strategies are left for future work. Also, evaluating the efficacy of the above approximation strategies is difficult because polyhedral optimizers have no implementation of Darte-Vivien's algorithm.

8.4 Affine Scheduling Frameworks and Clustering

The previous sections discuss algorithms for the under approximation of one single polyhedron, so that the UA polyhedron can be solved as a means of trading expressiveness (e.g., the ability to find "good" affine transformations) for scalability. In affine scheduling however, the overall system P is an intersection of many polyhedra, each of them arising from a dependence-edge as in the case of Feautrier's scheduler [Fea92a, Fea92b], or the above along with some additional constraints as in the case of Bondhugula's scheduler [GFG02, BHRS08]. More specifically:

- Feautrier's scheduler to minimize latency: $P_L = \cap_{e \in E} P_e$, where P_e is the per-dependence-edge Farkas polyhedron. A given dependence edge induces a (weak satisfaction) constraint in the system until it is strongly satisfied by outer dimensions of the multidimensional affine schedule (i.e., the sink of a dependence is scheduled at a strictly higher time step than the source).

- Bondhugula's FCO scheduler to expose loop tiling opportunties: $P_{FCO} = \cap_{e \in E} P_e \cap_{v \in V} H_v \cap_{v \in V} N_v$, where P_e is similar to Feautrier's scheduler's per-dependence polyhedron, enforcing weak dependence satisfaction only, but *at all dimensions* of the multidimensional affine schedule; while H_v collects the per-statement linear independence constraints, and N_v collects the per-statement non-negativity or trivial solution avoidance constraints.

These two cases are different because, thanks to the conditions in [Fea92b], the polyhedron built by Feautrier's multi-dimensional scheduler is known to be *always* feasible. This condition however is not necessarily true for FCO-based systems: though the number of problematic, additional constraints $|H_V| + |N_V|$ is small (around $2|V|$ or less than 10%) when compared to Farkas multiplier constraints $|P_E|$, the overall polyhedron could be empty, complicating the UA procedure.

The feasibility preservation problem is to find a non-empty UA(P) in each of the above cases, *without* making any queries—like an LP call—on the overall system P.[30] In this section we discuss practical methods to increase the control on feasibility for the overall approximated system.

8.4.1 Feautrier's latency minimization

Let us first discuss a technique for Feautrier's scheduler. Let us start by approximating each of the P_e by calling the one-shot algorithm discussed in Section 6.4.1, and construct the overall approximation by putting all the individual approximations together: $\mathrm{UA}(P) = \cap_{e \in E} \mathrm{UA}(P_e)$. The overall approximation is the result of $|E|$ calls to the one-shot procedure. This will be affordable because each of the P_e-s is quite small.

Feautrier's algorithm always finds a (rational) multi-dimensional affine schedule, which guarantees that P is a feasible polyhedron. The main question is whether feasibility is preserved when intersecting the per-dependence approximations. Surprisingly, the answer is yes. Notice that the original schedule of the source program is always feasible. The key idea is to *seed* the approximation of the per-dependence P_e polyhedra, forcing this original schedule into the UA. It simply amounts to substituting the values of the coefficients of the original schedule in the Farkas constraints, and adding the resulting (in)equalities to the systems before applying the one-shot UA method.

This result is important because of the large number of affine scheduling heuristics deriving from Feautrier's algorithm. Feautrier's fine-grain parallelization method is already useful for loop vectorization [Fea92b]. Our seeding approach is directly applicable to coarse-grain parallelization heuristics of Lim and Lam [LL97] as well.

We could have stopped here and declared success. But more advanced methods combining tiling for locality enhancement and parallelism extraction require additional effort to preserve feasibility. The next section studies the most successful of these heuristics, pushing us to enhance the under-approximation method to tackle feasibility preservation in presence of more complex affine constraints.

8.4.2 Griebl et al.'s *Forward Communications Only* tiling in PLuTo

In this section, we discuss the approximation of P in the context of Bondhugula's FCO scheduler, as implemented in PLuTo. Here, seeding with the original schedule is not possible since the loops of the source program may not be directly permutable/tilable. We study four Per-Dependence (PD) constraint clustering techniques: PD1–PD4.

[30] If such a query could be made, then it could as well be used to solve for the objective function resulting in the schedule...

PD1: Fully-separate. Each P_e is approximated alone, and all the additional constraints are approximated together as in:

$$\mathrm{UA}(P) = \bigcap_{e \in E} \mathrm{UA}(P_e) \bigcap \mathrm{UA}\left(\bigcap_{v \in V} H_v \bigcap_{v \in V} N_v \right) \tag{8.7}$$

The overall approximation is the result of $|E| + 1$ applications of the one-shot method.

PD2: Total augmentation. Each P_e is augmented with *all* the additional constraints, and this augmented polyhedron is approximated, as in:

$$\mathrm{UA}(P) = \bigcap_{e \in E} \mathrm{UA}\left(P_e \bigcap_{v \in V} H_v \bigcap_{v \in V} N_v \right) \tag{8.8}$$

The overall approximation is the intersection of of $|E|$ one-shot approxiations, and each element in H_V and N_V could be approximated multiple-times.

PD3: Selected augmentation. Each P_e is augmented with *its section* of the additional constraints H_u and N_u (for statements u on either side of dependence edge) and the augmented per-dependence polyhedron is approximated. If $e : v_1 \to v_2$,

$$\mathrm{UA}(P) = \bigcap_{e \in E} \mathrm{UA}\left(P_e \bigcap H_{v_1} \bigcap H_{v_2} \bigcap N_{v_1} \bigcap N_{v_2} \right) \tag{8.9}$$

The overall approximation is the intersection of $|E|$ one-shot approximations.

PD4: Just Farkas. Each P_e is approximated independently, just like in Feautrier's scheduler, while the additional constraints are thrown away. The overall approximation is the intersection of $|E|$ one-shot approximations.

This method clearly does not give an UA. It is meant to better characterize the source of the infeasibility in the previous three methods. Indeed, considering the intersection of UAs of per-dependence, weak-satisfaction constraints we know that the P_e's are homogeneous cones. This tells us that the origin can be used as a seed to build a non-empty approximation. But since the additional constraints are of a very specific and well understood form, and are in a way artificial, can be subjected to graph theoretic routines in the final Bellman-Ford routine. Our view is that separation of the Farkas-polyhedra on a per-dependence basis is the main problem, as they constitute the majority of constraints.

Clustering of dependences. The method here is to run the approximation on a larger partition or a cluster of the dependence graph instead of a single dependence edge. In this hierarchy, a cover or a selection of edges of the graph is selected, and the UA algorithm runs on the Farkas-polyhedron induced by the edges in that cover. There are many choices for such a cover, depending on the methodology used, whether program theoretic (loop based), algorithmic (graph based), or operational (linear prgramming based).

		Median		LP (Ind)		PD1		PD2		PD4	
Benchmark	P_{FPH}	YY	YN	YY	YN	YY	YN	YY	YN	YY	YN
adi	3	0	3	0	3	0	3	0	3	3	0
corcol	3	1	2	0	3	1	2	3	0	3	0
covcol	3	3	0	3	0	3	0	3	0	3	0
doitgen	4	4	0	-	-	4	0	4	0	4	0
dsyr2k	3	3	0	3	0	3	0	3	0	3	0
dsyrk	3	3	0	3	0	3	0	3	0	3	0
fdtd-2d	3	0	3	1	2	0	3	0	3	3	0
gemver	2	0	2	2	0	0	2	1	1	2	0
jacobi-1d	2	0	2	0	2	0	2	0	2	2	0
jacobi-2d	3	0	3	0	3	0	3	0	3	3	0
lu	3	0	3	0	3	0	3	0	3	3	0
matmul	3	3	0	3	0	3	0	3	0	3	0
matmul-init	3	3	0	–	–	3	0	3	0	3	0
mvt	2	0	2	2	0	2	0	2	0	2	0
seidel	3	0	3	3	0	0	3	0	3	3	0
ssymm	3	3	0	3	0	3	0	3	0	3	0
strmm	3	0	3	3	0	0	3	0	3	3	0
tmm	3	3	0	3	0	3	0	3	0	3	0
SUM	52	26	26	29	16	28	24	31	21	52	0

Table 8.1: UA effectiveness with constraint and dependence clustering techniques (18 out of 36 benchmarks from PolyBench 3.0)

8.4.3 Experiments in feasibility of clustering techniques

In Table 8.1, the calls denote the number of polyhedra of FPH variety for that benchmark, same as their numbers in Table 7.1 on page 91. These numbers are for TVPI-UA. The results for UTVPI-UA are exactly similar to these: the feasibility characteristics are invariant with respect to TVPI and UTVPI methods because rational polyhedra are used in the approximations.

These columns refer to the median method discussed in Section 6.2 on page 69, to the LP-based independent method discussed in Section 6.4.2 on page 76 and the per-dependence methods discussed in Section 8.4.2 respectively. All the columns except for the LP-independent method refer to the per-dependence methods. The LP-independent method however, was again implemented on the overall Farkas system: that is obtained after putting together all the individual systems and after simplification by PLuTo.

The PLuTo calls of the FPH variety are for `lexmin`. In the current table, we discuss the results of feasibility only. The columns YY (Yes-Yes), YN (Yes-No), denote the feasibility (Y) or infeasibility (N) of the original and approximated systems respectively. They have been highlighted accordingly. Since the FPH system is used to find an optimization point in the overall system, a YN entry would mean loss of parallelization.

It can again be seen that the LP-indep method (29 YY and 16 YN cases or 10 out of 16 PolyBench problems) performs much better than the median method (19 YY and 26 YN cases or 6 out of 16 PolyBench

problems). The latter performs not as poorly as the simplicity of the approach would hint to. We expect the incremental method to have much better performance than the current independent method. The results of PD1 fare only marginally better than the median method. The results of PD2 are close to LP-indep and are better than the median method (24 YY and 21 YN cases or 8 out of 16 PolyBench loop nests). Advanced clustering strategy PD3 has not been implemented. PD4 (Just Farkas) gives 100% feasibility preservation results (45 YY and 0 YN), making it quite attractive. The next sections study the impact on compilation time and on the performance of the generated code. Interpreting these UA systems so as to get permutable loops is left for future work.

8.5 Darte-Huard's Multi-dimensional Shifting for Parallelization

Darte and Huard in their multi-dimensional loop alignment [DH02] show that the external shifting problem is NP-Complete. Their result is surprising because this particular transformation can be framed as part of general affine transformations. Indeed, the framework of Feautrier [Fea92a, Fea92b] uses a formulation which uses rational LP program and hence can be solved in (weakly) polynomial time. And, shifting could be a result of this formulation as it is encoded in the search space as well [DSV97].

Darte-Huard however point out the difference between *general affine transformations* and what could be called as *niche transformations*. The former, with their (haystack) formulations encode in themselves multitudes of useful transformations, and even sequences of transformations in the search space. The latter (needle) formulations encode specific varieties of transformations, like shifting, fusion, distribution, etc., which are particularly useful for many reasons like smaller complexity of the parallelizer, more control of the solution, simplicity of the code generated etc. As an example of these latter kind of transformations, Darte-Huard prove that the complexity of multi-dimensional-loop shifting for parallelization, namely finding a constant shift to the outer loops so that inner loops can be parallelized—a simple enough transformation, and a multi-dimensional variation of the $1d$-shifting, which is circuit retiming [LS91] studied in Section 8.2—is NP-Hard.

8.5.1 Introduction

A sequence of loops with unit steps and uniform dependences, and each statement is surrounded by exactly n loops, such a program can be modelled as a directed graph $G = (V, E, \mathbf{w})$, where V is the set of statements, E is the set of dependences and $\mathbf{w} : E \to \mathbb{Z}^n$ is a n-dimensional weight vector annotating each dependence edge E.

Notation: The i-th component of the weight vector $\mathbf{w}(e)$ is denoted by $w(e)_i$. Applying a retiming $r : V \to \mathbb{Z}$ to a 1-dimensional dependence graph $G = (V, E, w)$ changes the edge weight of an arc $e = (u, v)$ from $w(e)$ to $w'(e) = w(e) + r(v) - r(u)$. Analogously, applying a multi-dimensional retiming $\mathbf{r} : V \to \mathbb{Z}^n$ to a graph of dimension n, can be considered to be a composition of n scalar retimings, one for each of 1-dimensional graph. We can denote by $G' = (V, E, \mathbf{w}')$ the graph obtained from G after a retiming $\mathbf{r}$. Applying a retiming $\mathbf{r} : V \to \mathbb{Z}^n$ changes the set of weights of an arc $e = (u, v)$ from $\mathbf{w}(e)$ to

$\mathbf{w}'(e) = \mathbf{w}(e) + \mathbf{r}(v) - \mathbf{r}(u)$.

The conditions given by Leiserson-Saxe [LS91] for the retiming of a 1-dimensional graph state that a retiming is legal if and only if all the resulting edges have non-negative weights. An n-dimensional extension of the conditions can be given which states that a n-dimensional retiming is legal iff each of the resulting arc weights in G' are lexico-nonnegative.

Following the above notations, Darte-Huard define internal and external shifting.

8.5.2 External shifting complexity

Circuits and cycles. In defining the different varieties of shifting, Darte-Huard make the crucial and unique distinction between circuits and cycles. In the general graph theoretical notations [CSRL01, Die06], a circuit is loosely used as equivalent to a cycle. Also, the weight of a path is defined to be the sequence of edges while *respecting the edge directions*. For purposes of multi-dimensional retiming, Darte-Huard use the same meaning with respect to circuits, but they point out that for cycles, the direction does *not* matter.

Using the above distinction in extending the conditions of Leiserson-Saxe [LS91] means that there is a correspondence between the weight of (undirected) cycles and the presence of valid retimings: namely, there is a multi-dimensional retiming which induces zero-weight edges in the retimed graph iff the original graph has zero-weight cycles. (This is rather counter-intuitive, but has been explained by Darte and Huard [DH00b].) Darte and Huard also make the distinction between internal and external shifting (which is used in multi-dimensional shifting).

Internal and external shifting. Internal shifting is but self-retiming, where parallel loops are searched for on a per-dimension basis. External shifting is trying to shift the outer dimensions so that the inner loops can be parallelized by internal shifting.

Darte-Huard prove that the internal shifting can be solved using a $\mathcal{O}(|E| + |V|)$ time algorithm, which is a variation of Depth-First-Search. External shifting, on the other hand is a much harder problem. Darte-Huard prove the NP-Completeness of external shifting.

The following is Theorem 2 of Darte-Huard [DH02].

Lemma 8.5.1 [NP-Completeness of External Retiming] Let $G = (V, E, \mathbf{w})$ be a graph of 2-dimensions and the problem of deciding if there exists a legal retiming, in the first dimension, such that G_r has only zero-weight cycles is strongly NP-Complete.

Proof: The proof involves a reduction from 3SAT to a variation of shifting problem. For details, refer Darte and Huard [DH00b]. ■

Since the problem is NP-Complete in general, Darte-Huard give an exact ILP formulation.

Lemma 8.5.2 [ILP formulation of Darte-Huard]. Let $G = (V, E, \mathbf{w})$ be a graph of 2-dimensions and define the quantity $\chi = \max(\sum_{e \in E} |w(e)_2|, 1)$. There exists a legal retiming such that the innermost loop is

parallel if and only if there exists a retiming r such that, for each arc $e = (u, v)$, we have $\chi w'(e)_1 + w'(e)_2 \geq 0$ and $\chi w'(e)_1 - w'(e)_2 \geq 0$.

The above lemma mentions about two retimings, the first being the internal retiming (or self-retiming of the inner-loop), and the second one being the external retiming. Or, in other words, the external retiming prepares the dependences of the inner loop, and then, internal retiming exposes the inner parallel loop.

To understand the above ILP formulation, we need to remember that for a vector $\mathbf{w}(e)$, the ith component is denoted by $w(e)_i$. Hence the weight vectors in two dimensions of an edge e are $\begin{pmatrix} w(e)_1 \\ w(e)_2 \end{pmatrix}$. Also intuitively, the constant χ can be considered to be a multiplier larger than the weight vectors in the 2nd dimension, so that they can be carried by the outer dimension. The $\geq$ condition is nothing but the multi-dimensional extension of the Leiserson-Saxe [LS91] retiming conditions. For a PRDG in 2-dimensions, the above ILP formulation is equivalent to the following:

$$\forall e = (u, v) \in E,\ \chi\,(w(e)_1 + r(v)_1 - r(u)_1) \pm (w(e)_1 + r(v)_1 - r(u)_1) \geq 0; \tag{8.10}$$

Since $\chi > 0$, assuming that there exists such a retiming $\mathbf{r}$, the above constraints can be used to derive (using Farkas lemma) the additional first dimension retiming constraint $w(e)_1 + r(v)_1 - r(u)_1 \geq 0$, which can also be obtained because of retiming conditions.

As an illustration, the following is an example from Darte-Huard.

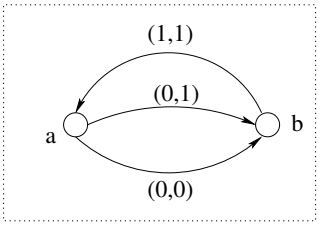


Figure 8.1: Darte-Huard External Retiming Example

Example 8.5.3 [ILP formulation example]. We begin with the example shown in Figure 8.1. (This example is same as Figure 12 in [DH02] or Figure 13 in [DH00b]). Here, $\chi = \max((1 + 1 + 0), 1) = 2$. The constraints are the following:

$$\begin{cases} e_1 = (b \to a): & 2w'(e_1)_1 \pm w'(e_1)_2 \geq 0; \\ e_2 = (a \to b): & 2w'(e_2)_1 \pm w'(e_2)_2 \geq 0; \\ e_3 = (a \to b): & 2w'(e_3)_1 \pm w'(e_3)_2 \geq 0; \end{cases}$$

which leads to the following equivalent system, assuming $\mathbf{r}_a = \begin{pmatrix} r_1^a \\ r_2^a \end{pmatrix}$ and $\mathbf{r}_b = \begin{pmatrix} r_1^b \\ r_2^b \end{pmatrix}$

$$\begin{cases} e_1 = (b \to a): & 2\left(1 + r_1^a - r_1^b\right) \pm \left(1 + r_2^a - r_2^b\right) \geq 0; \\ e_2 = (a \to b): & 2\left(0 - r_1^a + r_1^b\right) \pm \left(1 - r_2^a + r_2^b\right) \geq 0; \\ e_3 = (a \to b): & 2\left(0 - r_1^a + r_1^b\right) \pm \left(0 - r_2^a + r_2^b\right) \geq 0; \end{cases}$$

The change in signs in the above system for the r-variables is because of change in directions of the arcs: for e_1 it is $b \to a$, while for e_2 and e_3 it is $a \to b$. The above system is equivalent to the following:

$$\begin{cases} e_1: & +2r_1^a - 2r_1^b + r_2^a - r_2^b + 3 \geq 0; \; +2r_1^a - 2r_1^b - r_2^a + r_2^b + 1 \geq 0; \\ e_2: & -2r_1^a + 2r_1^b - r_2^a + r_2^b + 1 \geq 0; \; -2r_1^a + 2r_1^b + r_2^a - r_2^b - 1 \geq 0; \\ e_3: & -2r_1^a + 2r_1^b - r_2^a + r_2^b \quad \geq 0; \; -2r_1^a + 2r_1^b + r_2^a - r_2^b \quad \geq 0; \end{cases} \tag{8.11}$$

Darte-Huard suggest that the above ILP formulation can be solved using standard ILP solvers. Also, they suggest the following retimings: $\mathbf{r}_a = \begin{pmatrix} r_1^a \\ r_2^a \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\mathbf{r}_b = \begin{pmatrix} r_1^b \\ r_2^b \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ as a possible solution. It can be seen by substitution that these values satisfy the system in Equation 8.11. □

8.5.3 Heuristics and approximations

For a retiming problem of d-dimensions, the above problem formulation by Darte and Huard has $d|V|$ variables and $2d|E|$ constraints. But, it can be noticed that the above ILP formulation is constructed on a per-dependence-edge basis and has 4 non-zero variables per constraint. Also, the constraints are general (not just $\{0, \pm 1\}$) because of the χ-coefficients. The above system leads to a variety of heuristics and approximations.

First we begin with some heuristics suggested by Darte and Huard:

Heuristic 8.5.4 [Heuristics of Darte-Huard]. (i) Beginning from the innermost loop till the outermost loop, use internal shifting on a per-loop basis, exposing as many parallel loops at each individual level. (ii) Use Bellman-Ford on the outer loops to expose zero-weight edges on each innerloops.

Of the above methods, (i) can be done in linear time because of the strategy suggested by Darte-Huard where the directions of the edges are obliterated by using cycles instead of circuits, and using a depth-first-scan based algorithm to expose parallelism. Also, (ii) can be done in a very affordable $\mathcal{O}(|E||V|)$ worst-case time. More details are in [DH00b]. The above methods are inexpensive, but are not as powerful as ones based on external retiming. The cases when they work may be limited.

We can also have the following approximation using a LP rounding scheme.

Approximation 8.5.5 [LP relaxation]. A simple approximation would be to relax the ILP program of Darte-Huard in Equation (8.10) into an LP program. The resulting rational LP constraints could be solved in polynomial time using the simplex algorithm. Indeed, rational relaxation of an ILP formulation

and LP-rounding of the rational solution to obtain an integer point for the original problem has been a standard way in which approximation algorithms of NP-Complete problems has been designed [Vaz01].

But, the above approach has the limitation of needing a proper rounding scheme which preserves the legality of the resultant rational retimed graph. A legal rational retiming can directly be applied to G followed by an expansion of G, but, it solves the retiming of the expanded graph, not of the original graph. Also, such an expansion has the undesired side-effect of increasing the code size. Furthermore, the LP-relaxation may be quite expensive to solve as well, with no guarantee of worst-case running time. This is particularly true because of the large size ($2d|E| \times d|V|$) of the LP formulation, which takes up to $\mathcal{O}(d^3|E|^2|V|) \approx \mathcal{O}(|V|^5)$ time assuming the observed time of simplex algorithm for bounded d.

The above limitations of a straightforward LP-relaxation leads to the following approximation using UTVPI-UAs.

Approximation 8.5.6 [Rational (U)TVPI-UA as heuristic]. A heuristic for Darte-Huard's formulation that builds on our approximation techniques is one that UTVPI-UAs each per-edge constraints and solves the overall system using Bellman-Ford algorithm. This is valid because rational relaxation and subsequent UA will result in fewer integer points, but no new ones. The overall UA, if non-empty, can be solved in polynomial time. Since the sparsity of the input system is small and constant (= 4), the (U)TVPI systems have about $d|V|$ variables and $16d|E|$ constraints. The UTVPI systems, or equivalently their difference constraint MUTVPI approximations, can be solved in $\mathcal{O}(d^2|E||V|)$ worst-case time, which is around $\mathcal{O}(|E||V|)$ worst-case time for bounded d.

The above approximation comes with a guarantee of bounded cubic worst-case running time, but does not come with a guarantee of non-emptiness of the resultant polyhedron. Also, the question of rational retiming still remains unresolved, which leads to the following approximation.

Approximation 8.5.7 [Integer UTVPI-UA as heuristic]. This approximation is the same as Approximation 2, but uses the integer scaling discussed earlier so as to directly result in an integer multi-dimensional shift. In this method, an *ILP problem* is solved on a per-P_e-basis, while solving a *normal Bellman-Ford problem* on the overall UA polyhedron resulting in an integer multi-dimensional shift.

Such a polynomial time shifting algorithm is not possible even with a TVPI-UA, as solving integer-TVPI polyhedra is known to be NP-Complete by the result of Lagarias [Lag85].

Validity of UTVPI-UA heuristic It should however be noted that the claim in the above UA-based heuristics (Approximations 8.5.6–7) is *quite different* from the claim made in the algorithm given in [SLP96]. In this latter paper, the authors give a polynomial time algorithm for shifting for fusion by formulating an LP problem with difference constraints, and propose to solve the resulting system using Bellman-Ford algorithm which has worst-case cubic running time. Furthermore, the authors claim (Theorems 3,12 of [SLP96]) that their algorithm always finds such a retiming function. As the external shifting problem has been proved by Darte-Huard to be NP-Complete, unless $P = NP$, the claimed polynomiality of the above result is invalidated by Darte-Huard's result.

Our above claim on the other hand is much weaker because it is just a heuristic. Also, our approximations can be considered to be similar to, as well as an extension of heuristics given by Darte-Huard [DH02, DH00b], where linear or Bellman-Ford based heuristics are given. Furthermore, note that the above approximation could result in UAs that are empty when run on a per-dependence edge basis.

Feasibility preservation. We propose some methods by which the problem of feasibility of the TVPI-UA could be solved:

1. **One-shot UA-method:** Running a one-shot algorithm on all the constraints obviously guarantees that the per-dependence UA is feasible, but still does not provide a guarantee that the overall intersection of per-dependence UAs is non-empty.

2. **Fixing the sign of the retiming vectors:** One limitation of the TVPI approximations is that they are sign unaware and the approximation spans too many orthants. To alleviate this problem, the sign of the retiming vectors could be fixed a priori: namely by redefining each retiming vector $\mathbf{r} = (\mathbf{r}^+ - \mathbf{r}^-)$, with $\mathbf{r}^+, \mathbf{r}^- \geq \mathbf{0}$. By this assignment, followed by fixing one of the $\mathbf{r}^+$ or $\mathbf{r}^-$ to be $\mathbf{0}$ so that the resultant retiming constraints are of the form $\sum a_i r_i + c \geq 0$ where a_i is a non-negative constant, then it could be ensured that the overall UA space spans only the positive orthants and hence is non-empty (this method can be considered to be a rotation of the orthants so that the polar vectors are in one single positive space).

3. **Seeding method for UTVPI-Approximations:** For feasibility guarantee of the constraint system, we need to observe that the variables in the constraint system are the components of the retiming vectors. The origin is always a feasible retiming; and a retiming of $\mathbf{0}$ at each of the vertices of the PRDG is always a valid retiming with the retimed graph corresponding to the original program. If this seed value is preserved in each of the per-dependence UAs, the overall intersection of the per-dependence UAs will have this point as well as (hopefully) other interesting points, which can be searched for by the Bellman-Ford algorithm.

Empirical evidence is needed to determine the efficacy of the above methods in practical problems and the corresponding improvement on running times. As in Darte-Vivien, we are not aware of any implementation of the above algorithm.

8.6 Darte-Huard's Array Contraction and other ILP problems

Darte and Huard [DH05] show the NP-Completeness of two problems *Loop Fusion for array contraction* and *Loop Shifting for array contraction*. In this section, we show that both the formulations by Darte-Huard are amenable to using our framework in multiple ways: under-approximation or enumeration with the basic engine being a strongly polynomial time UTVPI solver.

8.6.1 Loop fusion for array contraction

In loop fusion for array contraction, Darte-Huard give the following exact ILP formulation. Let $G = (V, E)$ be the dependence graph annotated by uniform dependence distances—with the non-uniform dependences handled in an appropriate way, like uniformization—and let $\mathcal{A}$ be the set of array variables. Also, they define simple conditions which classify each arc e as contractable or not. An arc $e = (u, v) \in E$ is called fusion-preventing if it is annotated by a negative dependence distance; this means that source u and destination v should be placed in different loops, and the source in a loop textually before. An arc $e = (u, v) \in E$ with a non-negative distance is not fusion preventing; the source should be placed in a loop before the destination, or it can be placed in the same loop. The overall resultant of this variety of fusion partition is a parition of vertices V into disjoint subsets, as long as they sasisfy legality conditions. Darte and Huard suggest the following ILP formulation for the same:

$$\begin{cases} \text{min} & \sum_{a \in \mathcal{A}} \delta_a \\ \\ \text{subject to} & \begin{cases} \text{if e} = (\text{u}, \text{v}) \in \text{E is not fusion preventing} & \rho_v \geq \rho_u \\ \text{if e} = (\text{u}, \text{v}) \in \text{E is fusion preventing} & \rho_v \geq \rho_u + 1 \\ \text{if e} = (\text{u}, \text{v}) \in \text{E is fusion preventing} & N\delta_a \geq \rho_v - \rho_u \end{cases} \end{cases} \tag{8.12}$$

It can quickly be noticed that in the above ILP formulation: (i) δ_a is a binary variable representing whether the array variable $a \in \mathcal{A}$ can be contracted or not; (ii) N is a constant and is heuristically set to number of edges; (iii) the cost function minimizes the number of contractable arrays.

UTVPI Under-Approximations using enumeration. For designing an approximation or enumeration strategy for the above formulation, we have to observe that the δ-variables are binary variables and define the cost function. If these variables are set, the rest of the polyhedron induces a UTVPI polyhedron, which can be solved using Bellman-Ford. The δ-variables could be iterated upon with their range simply being $0 \ldots |\mathcal{A}|$. The above strategy gives rise to a very straightforward enumeration method using UTVPI polyhedra. Since the objective is to minimize the number of accesses, the enumeration can proceed by increasing the array variables that are to be contracted, in a sequential manner.

8.6.2 Loop shifting for array contraction

In their second ILP formulation, Darte-Huard give an ILP formulation for loop shifting for array contraction, that extends the above formulation for array contraction and their earlier formulation for multi-dimensional shifting [DH02], which was studied in Section 8.5. Hence, the approximation-strategies listed in Section 8.5, as well as and enumeration strategy listed in Section 8.6.1 can be applied to design heuristics or enumeration methods. The quality of result is equivalent by construction.

8.6.3 Other 0-1 ILP problems: weighed loop fusion

Darte and Huard's above mentioned fusion models [DH05] are more powerful than the ones by Song et al. [SXWL01]. Also, Darte and Huard's models for contraction rectify, and make precise the NP-Completeness result for weighed loop fusion by Kennedy and Mckinley [KM94].

Darte and Huard [DH05] also extend the models of Megiddo and Sarkar [MS97] for the weighed loop fusion problem. This latter paper by Megiddo and Sarkar also has ILP formulation, with a corresponding LP relaxation. Furthermore, they show that a simplification of this LP relaxation has a Totally-Unimodular constraint matrix and hence admits strongly polynomial time solutions. We study these formulations because of their relevance to our work.

Megiddo-Sarkar's input model. The input to Megiddo-Sarkar's algorithm is a directed (but acyclic) graph, called Loop Dependence Graph LDG, whose vertices are perfectly nested loops and whose edges are dependences (data dependences, like array variables) from the source loop vertex to target loop vertex. Though the input model of Megiddo-Sarkar is quite weak when compared to the other formulations in this chapter–like Darte-Huard's formulation [DH05]–but it will be elaborated upon because of their use of LP relaxations of ILP formulations and TU approximations of the same.

Megiddo and Sarkar denote by $C \subseteq E$, as the set of ordered pairs of vertices ("contractable arcs") and B the set of unordered pairs of nodes (pairs of nodes that have non-zero weight but are not connected by a summary arc). They formulate an ILP formulation (Problem 3.5 of Megiddo and Sarkar [MS97]) and then give approximations and relaxations. Also, using a series of transformations, they convert the above formulation into the following (Problem 6.3 of Megiddo and Sarkar [MS97]). Let $\epsilon_{ij} \in \{-1, 1\}$ for every $(i, j) \in B$. A constant ϵ_{ij} is chosen a priori for edge (i, j), and π-variables are to be found out.

Lemma 8.6.1 [LP relaxation of [MS97]]. The following LP formulation gives a solution to the π-variables:

$$\begin{cases} \min_{\pi} & \sum_{(i,j)\in B\cup C} \epsilon_{ij} w_{ij}(\pi_j - \pi_i) + \sum_{(i,j)\in C} w_{ij}(\pi_j - \pi_i) \\ \\ \text{subject to} & \begin{cases} \pi_i - \pi_j \geq 1 & ((i,j) \in NC) \\ \pi_i - \pi_j \geq 0 & ((i,j) \in C) \\ (\pi_j - \pi_i)\epsilon_{ij} \geq 0 & ((i,j) \in B) \end{cases} \end{cases} \tag{8.13}$$

Megiddo-Sarkar's formulations and UTVPI approximations. Megiddo and Sarkar claim that the above LP relaxation in Equation (8.13) is TU, which means that it can directly result in an optimal solution in strongly polynomial time. Instead, the LP-relaxation and its UTVPI approximation will be a direct consequence of using our approaches in this dissertation, and more particularly in this particular chapter; the suggestion above is directly advocating for our UTVPI, and consequently MUTVPI, approximation approaches.

8.7 A General UA-Framework of Loop Transformation Polyhedra

UA framework. The formulations seen above in this chapter, there is a common pattern of formulating mixed $0-1$-ILP problems, where some variables ($\mathbf{x}$) are restricted to rationals ($\mathbb{Q}$), some ($\mathbf{y}$) to integers ($\mathbb{Z}$), while the rest (δ) to binary ($\{0,1\}$) decision values. We use this insight to develop a general framework for MILP approximation that is particularly suited for loop transformation polyhedra: polyhedra whose solutions are valid affine loop transformations.

For this let us consider the general set of scheduling constraints:

$$\begin{cases} \min & \sum \mathbf{c}(\mathbf{x},\mathbf{y},\delta) \\ \text{subject to} & A\mathbf{x}+B\mathbf{y}+C\delta \leq \mathbf{d} \\ & \mathbf{x}\in\mathbb{Q}^{n_x};\ \mathbf{y}\in\mathbb{Z}^{n_y};\ \delta\in\{0,1\}^{n_\delta} \end{cases} \tag{8.14}$$

where A, B, C are constraint matrices and $\mathbf{x}, \mathbf{y}, \delta$ are the variables of the scheduling problem. $\mathbf{c}$ is a linear cost function specific to the problem at hand.

We can use this above generic MILP formulation to design under-approximation algorithms for loop scheduling or affine transformation problems. We use Table 8.2, to characterize the different problems and their complexities. We use the notation of Section 3.4 on page 30: SPT means strongly polynomial Time, WPT means Weakly Polynomial Time because the algorithm has been reduced to LP, and NPC means that the problem is NP-Complete.

Using the above notation, Calland et al. [CDR98] and Darte and Huard [DH00a] use TU properties of A to formulate a (strongly) polynomial time algorithm. This is possible because A is derived using G.

Similarly, Darte and Vivien [DV97a] formulate an LP formulation, which, though using binary decision variables, takes weakly polynomial time. Feautrier [Fea92a, Fea92b] and FCO of Griebl et al. [GFG02, BHRS08] formulate weakly polynomial time LP based algorithm, where the $\mathbf{x}$-variables are constructed using Farkas multipliers. Feautrier's multi-dimensional scheduling and the FCO scheduling use appropriate binary decision variables, though the overall algorithm taking weakly polynomial time. In a similar way, Lim-Lam [LL97] and Thies et al. [TVA07], use a similar framework, though for purposes of synchronization free parallelism and memory optimization respectively.

Darte and Huard [DH02] formulate NP-Complete problems where $\mathbf{y}$-variables are constructed using the retiming vectors. Darte and Huard [DH05] build from the above formulation and the structure of the graph to formulate a $0-1$ ILP formulation for the NP-Complete problems of array contraction.

UA strategies. Broadly, using the above framework, we can design UA strategies so that each of the above problems can be solved with a low worst-case polynomial complexity: This can be done in three different ways: (i) relaxing all $0-1$ variables and integer ($\mathbb{Z}$) variables to rationals ($\mathbb{Q}$), and finding a UA in a relaxed framework; (ii) replacing the $\mathbb{Q}$-polyhedron by its UA, followed by enumeration of the $0-1$ variables; (iii) separating the overall $\mathbb{Q}$-polyhedron to smaller scalable sub-problems that can be under-approximated using graph theoretical properties.

Algorithm	Goal	$\mathcal{O}(?)$	Characterization of variables in the formulation		
			A and $\mathbf{x}$ ($\mathbf{x} \in \mathbb{Q}^{n_x}$)	B and $\mathbf{y}$ ($\mathbf{y} \in \mathbb{Z}^{n_y}$)	C and δ ($\delta \in \{0,1\}^{n_\delta}$)
Calland et al.. [CDR98] and Darte-Huard [DH00a]	Shifting for Parallelization and Compaction	SPT	A is function of incidence matrix of G	–	δ_i represents whether edge i is chosen or not as a result of retiming.
Darte-Vivien [DV97a]	Multi-dimensional scheduling for inner loop parallelization	WPT	A is function of incidence matrix of G and (uniform) dependence weight vectors	–	δ_i represents whether edge i is chosen or not at that level of KMW decomposition.
Feautrier [Fea92a, Fea92b]	Latency mimimization	WPT	Per-dependence edge Farkas polyhedron	–	
Griebl et al. [GFG02, BHRS08]	Expose permutability for tiling	WPT		–	–
Darte and Huard [DH02]	Multi-dimensional shifting for compaction	NPC	–	$\mathbf{y}$ represents the multi-dimensional retiming vector	–
Darte and Huard [DH05]	Loop fusion for array contraction	NPC	Function of dependence graph	–	δ_i represents whether array variable i is contracted or not.
	Loop shifting for array contraction	NPC		$\mathbf{y}$ represents the multi-dimensional retiming vector similar to [DH02]	

Table 8.2: Framework for Unifying Loop Transformation Problems (G is the dependence graph of the input program)

A combination of the above methods could also be used. Even when the UA methods of is not effective, it could give rise to an effective enumeration strategy that could be used. Briefly, relaxation has already been used in Calland et al. [CDR98] and Darte and Huard [DH00a]. Relaxing $\mathbf{y}$ to rational variables followed by (U)TVPI approximation has been suggested for Darte and Huard's multi-dimensional scheduling [DH02]. Enumeration on δ-variables using the UA of the relaxation could be used for any NP-Complete problems with ILP/MILP formulations. Furthermore, the unit-schedule (canonical schedule) preserved in the UA could be used for a feasibility preservation.

8.8 Schedules with no Modulos

We state in Section 6.7.5, that thanks to the Totally Unimodular(TU) property, a restriction of the homogenizing dimensions to integers can ensure that *all the vertices* of the resultant UTVPI-UA are integers. In this method, the under-approximation is again performed on a per-dependence basis, though with the additional restriction that the homogenizing dimension of the under-approximation is integer. This latter restriction is possible by scaling the homogenizing dimension choices along with appropriate pseudo-parametric values, and making an ILP call, rather than the usual LP call per-dependence. The overall result is a UTVPI approximation.

Because of the TU property, this latter approximation is a polyhedron with integer vertices. When using these UAs in scheduling, the primary advantage is that the schedule can be found in strongly polynomial time with very low $\mathcal{O}(mn)$ complexity; it has the additional advantage of simpler code being generated, even with affine schedules obtained as a result of (rational) linear programming. The FCO scheduler in PLuTo uses integer linear programming by default for the same purpose, and such a use could be expensive. Rational schedule coefficients may indeed induce modulos in loop bounds and conditional expressions of the generated code. This is eliminated by such an a priori restriction of the schedules.

8.9 Conclusions and Perspectives

We review the conclusions and contributions of this chapter and summarize some related work.

Conclusions

In this chapter, we have provided a preliminary idea on how the under-approximation strategies developed in earlier chapters could be applied to various scheduling problems in loop optimization. We began with the simplest case of problems [CDR98, DH00a] that have already been reduced to strongly polynomial time. This was followed by more general transformations [DV97a, Fea92a, Fea92b, GFG02, BHRS08] that are linear programming based and hence have a weakly polynomial time complexity. This was further followed by special case transformations [DH02, DH05], which are computationally much harder problems because of their NP-Completeness reductions. We suggested heuristics, approximation and enumeration strategies on a case by case basis. The methods should be seen as initial approaches on how UA methods could be applied, with more study being needed.

Contributions

Under approximation of scheduling problems. We think that this is perhaps the first systematic and algorithmic approximation scheme to be applied to formulations in loop scheduling. Firstly, we studied a gradation of problems: according to the type of loop transformations they target ($1d$-transformations, more general nd-transformations, or specific nd-transformation), and according to their algorithmic complexity—strongly polynomial time, weakly polynomial time, or NP-Complete). Each of these transformations could benefit from the under-approximation methodology developed in the previous chapters.

Though we have suggested for some NP-Complete problems, it could even be used with more recent and more powerful techniques as well. This means the fusion in the presence of hardware and other constraints by Bondhugula et al. [BGDR10], and the complete convex characterizations of loop transformations by Pouchet et al. [PBB+11]. Each of these could benefit from our framework, where the rational part is simplified (UA'd) to a (U)TVPI polyhedron in a preprocessing step, before the actual solution step.

Parts of the experiments (clustering techniques) and some of the applications listed in this chapter have been presented in [UC13].

Scheduling amenable to code generation. In Section 8.8, we had suggested a very preliminary method in which the schedule module of the polyhedral code generation is made aware of the latter code generation module. This can be considered to be in the same spirit of Darte and Vivien [DV97a] and Darte and Huard [DH02] who hint that more complicated schedules may not practically yield sufficient advantage because the code generated for those schedules may be very complicated. But, as far as we are aware of, this aspect of exploiting Totally-Unimodularity has not been previously approached in loop optimization. The approach in this section should however be seen as different from the approach in the

following chapter (Chapter 9), where the complexity of the code generaton is independently reduced to polynomial time using the complexity guarantee of (U)TVPI polyhedra.

Limitations

The under-approximation methods are limited in a couple of ways: feasibility and practical effectiveness. For many of the applications in this chapter, more study of the preservation of feasibility via the preservation of the unit schedule is needed. Furthermore, most of the applications in this chapter need more empirical study regarding their effectiveness.

Feasibility and effectiveness. In particular, the approximations for the $1d$-transformations [CDR98, DH00a] and even more general transformations like Darte-Vivien [DV97a] may not be practically significant because of the restricted practicability of the original formulations. However, the insights given by these original formulations and the approximation or enumeration techniques are significant to be applicable to problems of greater use and complexity. Feautrier's formulation [Fea92a, Fea92b], FCO formulation [GFG02, BHRS08], and the NP-Complete formulations of Darte and Huard [DH02, DH05] are such problems.

Even though the effectiveness of these latter three applications may also be limited, and the approximation or enumeration techniques suggested may be too simple to begin with, it is hoped that the ideas presented here will lead to a new method to derive approximation algorithms for loop transformation problems. In this sense, the effectivenss of the approximations is related to finding approximations to NP-Complete problems by Hochbaum [HMNT93, Hoc04].

Improvement of methods. Lack of the feasibility guarantee of the approximations is a limitation. It could be overcome using the idea of preservation of canonical schedules suggested many times in the chapter. Furthermore, the constraints or the dependences could be grouped using clustering techniques similar to those suggested in Section 8.4.2.. Finally, some of the enumeration strategies given here have the severe limitation that they do not have a polynomial time guarantee. They can however be improved by studying the particular problem carefully, as indicated in Section 8.7.

From Chapter 4 till the current chapter, we applied the (U)TVPI sub-polyhedra based under-approximation techniques to affine scheduling problems. In this chapter, the application was extended to a variety of affine transformation problems. In the next chapter, we change tracks by applying the use of (U)TVPI polyhedra to the polyhedral code-generation problem.

Chapter 9

An Approach for Code Generation using (U)TVPI Sub-Polyhedra

In this chapter, we explore how we can use TVPI polyhedra for sequential code generation, with a goal of polynomial time complexity guarantee. Though our goal is to solve the code generation problem, the problem we actually solve is a sub-problem: that of *polyhedra scanning using TVPI sub-polyhedra*. The difference between the two is that the latter scanning problem does not need to take care of how the imperfectly nested loops are encoded, how the code for the statements in the loop (loop body) is encoded etc.

9.1 Introduction

Beginning from a schedule—possibly a multi-dimensional schedule—for a set of input statements, the problem of code generation is loosely defined to finding a *scanning order* by touching each integer point in the domains exactly once, while respecting the dependences between the statements. For serial code, the scanning order means to generate `for` loops for the input domains. In this chapter, we will propose TVPI based methods for doing the same in asymptotically polynomial time using TVPI properties.

Structure of this introduction. For doing the above, we would first quickly summarize the current code generation methods in Section 9.1.1. This will be followed by Section 9.1.2, where we will identify the problem in code generation to be the exponentialities in the algorithm by Quilleré et al. [QRW00]. This will be followed by listing of the goals and some hypothesis in Section 9.1.3, followed by showing the potential of TVPI polyhedra in Section 9.1.4.

9.1.1 A summary of current code generation methods

Very broadly, the three main methods of polyhedral code generation are based on the geometric algorithms of Fourier-Motzkin (FM), Parametric Integer Programming (PIP) and separation of polyhedra. Three primary metrics are used to evaluate these methods: the size of the generated code, the execution time of the code generator, and the execution time of the generated code.

FM based method. The FM based methods are based on the seminal work of Ancourt and Irigoin [AI91], who suggested using Fourier-Motzkin algorithm for generating the loop bounds for the `for` loops. The main intuition of this method is to use the projection method of FM to each of the individual dimensions. The resultant bounds of each loop index are affine terms of the outer dimensions and the parameters and can be used to generate the bounds of their corresponding loops. This method has been extended by many researchers for the cases of multiple polyhedra and statements cases.

A primary limitation of FM is the algorithm itself, that is known to take worst-case doubly exponential time because of the large number of redundant constraints that are generated. Inspite of good implementations in libraries like FMLib [Pou] and Omega [Pug91], the algorithm has a doubly exponential time complexity. Also, the main limitation of the FM based methods is not only its time complexity, but also the inactive or redundant areas in the generated code. This is a directl result of the redundant constraints generated by the algorithm.

An implementation of this method improved with a redundancy eliminating mechanism has been suggested by Kelly, Pugh and Rosser [KPR95] and exists in the Omega library. This code generator, CodeGen, is one of most widely used and has recently been improved by the work of Chen [Che12] in Omega 2.0 to CodeGen+. This latter improvement is discussed in the related work section at the end of this chapter.

PIP based method. The PIP based method are the work of Boulet and Feautrier [BF98]. This latter algorithm is based on the intuition that the successive integer points that have to be scanned can be found as a result of integer programming `lexmin`s, and these calls can further be parametrized. Similar to the FM based methods, the PIP methods still generates redundant control and even needs to duplicate code. The method is based on the PIP [Fea88], but, there is no widely used implementation of this method for the case of multiple polyhedra. So, it is not known how it practically fares on real benchmarks. Furthermore, it does not aim to generate high-level loop statements like the other tools.

Separation based method (QRW algorithm). The separation based methods started with the algorithm by Quilleré, Rajopadhye and Wilde [QRW00], called the QRW method. The QRW method is the only method of the above three methods which can generate exact code with non-redundant control. This algorithm has been extended and implemented in CLooG [Bas04a, Bas04b] and is the most widely used among the above three algorithms. The main intuition of this method is based on separation of polyhedra and their projections. The algorithm is given in a simplified form [Bas11, Bas12] in Figure 9.1.

Input: A polyhedron list $\mathcal{D}_i$, a context C and the current dimension d.
Output: The AST of the code scanning the input polyhedra.

1. **Intersect** each polyhedron with the context C.
2. **Project** the polyhedra onto the outermost d-dimensions.
3. **Separate** these d-dimensional projections into disjoint polyhedra.
   (This step generates loops for dimension d and lists of polyhedra for dimension $d+1$.)
4. **Sort** the loops to respect the lexicographic order.
5. Recursively generate loop nests that scan each list with dimension $d+1$, under the context of the dimension d.
6. Return the AST for dimension d.

Figure 9.1: Algorithm QRW for Polyhedral Code Generation

The QRW algorithm is a high-level one, involving only abstract domain operations—such as intersection, projection, separation (involving difference), image—on polyhedral objects. The polyhedral objects are separated into disjoint unions of polyhedral objects. This contributes to the high cost of the algorithm. Furthermore, many of the above operations cannot be performed with the $\mathcal{H}$-form representation only. They involve a $\mathcal{V}$-form representation[31] as well, and a corresponding $\mathcal{V} \leftrightarrow \mathcal{H}$ conversion of the underlying polyhedral objects, which again contributes to the time complexity of the algorithm.

As suggested in the QRW paper, a version of CLooG, which we refer to as CLooG-PolyLib, employs PolyLib [Wil93] for these polyhedral operations. This library uses the dual representation of polyhedra, where both the constraints and generators of polyhedral objects are maintained. The Chernikova algorithm is used for the dual conversion.

9.1.2 The complexity problem: exponential complexities in the QRW method

Approaching QRW method. We believe that polynomial time properties of (U)TVPI sub-polyhedra can be used to reduce the time complexities of either of the FM-based methods or the QRW algorithm. However, due to the above listed advantages of QRW algorithm as well as the widespread use of CLooG, we target it for generating code in polynomial time. So as to understand what *Polynomial time code generation* means, we define and understand where the exponential complexities in Quilleré et al. [QRW00, Bas04a], as implemented in CLooG-PolyLib [Bas04a, Bas04b], are coming from.

The two exponential steps in QRW. In the QRW algorithm implemented in CLooG-PolyLib, there are two sources of exponential complexity, the first from the abstract set theoretic separation of polyhedral

[31] Reminder: As in the rest of this thesis, we use the notation that $\mathcal{H}$-form is for constraint representation, while $\mathcal{V}$-form is for generator representation.

objects, and the second from the representations of the polyhedral objects and the underlying algorithms of the corresponding polyhedral operations.

The first exponential complexity involves taking n abstract polyhedral objects and performing set theoretic domain operations on them. The key step is the decomposition into a disjoint union of polyhedra. As illustrated in Figure 9.2.a, it is easy to see that separation of n objects returns 3^n objects. If $|V|$ is the number of statements to be scanned, and d is the number of non-parameter dimensions, the complexity of separation is $3^{|V|d}$ operations [Bas12, Section 5.2].

The second complexity is from the polyhedral operations that the QRW algorithm uses and builds from. The polyhedral operations that are needed for the separation based algorithm are projection, intersection, union, image, separation, difference etc.,[32] As noted earlier, the QRW algorithm itself is independent of any particular implemetation, and hence uses any library that can support these abstract domain operations. These operations are typically supported by polyhedral libraries like PolyLib [Wil93], or Presburger arithmetic library like Omega [Pug92], or integer set libraries like ISL [Ver10]. With any of these libraries, the operations are expensive because of the non-convexity of the operations, or because of the internal representation of the polyhedral operations, or simply because they have high and typically non-polynomial-time asymptotic computational complexity. Though the internal representation by the library is only incidental to the running of the QRW algorithm, it is a fact that it increases the overall complexity further.

If the polyhedral objects are represented using a library like PolyLib [Wil93], it uses a canonical dual representation, where both $\mathcal{H}$ and $\mathcal{V}$ forms are internaly stored for each polyhedral object and the $\mathcal{V} \leftrightarrow \mathcal{H}$ conversion is done when necessary. Given two convex polyhedra P_1 and P_2 and affine function $\mathcal{T}$, the operations projection $\texttt{Proj}(P_1, x_i)$, intersection $P_1 \cap P_2$, $\texttt{Image}(P, \mathcal{T})$, and $\texttt{PreImage}(P, \mathcal{T})$ are closed operations on polyhedra, while operations like difference $P_1 - P_2$, and consequently separation $\texttt{Separate}(P_1, P_2)$ are not even closed for convex polyhedra, typically returning a union of polyhedra[33]. Also, some of these operations like $\texttt{conv}(P_1, P_2)$ are best done in $\mathcal{V}$-form. Furthermore, for many of these operations, PolyLib uses $\mathcal{V} \leftrightarrow \mathcal{H}$ conversion for simplification purposes [Wil93]. This dual representation is algorithmically supported by Chernikova algorithm [Ver92] and is well known to take worst-case exponential time. Hence, these operations add a layer of exponentiality.

Alternatively, if the abstract objects are represented using $\mathbb{Z}$-polyhedral or integer-set libraries like Omega [Pug92] or ISL [Ver10] a similar argument shows that these operations involve solving problems like minimization on ILP polyhedra [Sch86], Generalized Basis Reduction [Coh93] etc., each of which are NP-Complete [GJ79].

Illustrations of the above exponentiality is shown in Figure 9.2.b.

The overall cost. The overall cost of code generation is the product of the above two terms [Bas04a, Bas04b]. Though the product of the two terms is not apparent in the way the algorithm is written in Figure 9.1, it is indeed true because the QRW algorithm is a recursive one with the domain lists, as well

[32] CLooG-PolyLib also uses some additional operations like PreImage in its implementation.

[33] Computing the difference of two convex polyhedra is not a straightforward implementation needing the constraints in DNF followed by a conv.hull operation for over-approximation. One polyhedral implementation is in PolyLib [Wil93] thanks to Wilde, while another is in the PIPS [PIP] compiler, thanks to Leservot [Les96].

$$\begin{cases} \text{Separate}\,(S_1, S_2) & = \{S_1 - S_2, S_1 \cap S_2, S_2 - S_1\} \\ \text{Separate}\,(S_1, S_2, S_3) & = \left\{ \begin{array}{c} S_1 - S_2 - S_3, \ldots, \\ S_3 - (S_1 - S_2), \ldots, \\ S_3 - (S_1 \cap S_2), \ldots, \\ (S_1 \cap S_2) - S_3, \ldots, \\ S_1 \cap S_2 \cap S_3 \end{array} \right\} \end{cases}$$

(a) Separation of n abstract objects into their disjoint union resulting in an exponential size output

$$\begin{cases} P_1 \cap P_2 & \equiv P_1^{\mathcal{H}} \cap P_2^{\mathcal{H}} \\ P_1 \cup P_2 & \equiv P_1^{\mathcal{V}} \cup P_2^{\mathcal{V}} \end{cases}$$

(b) Operations on polyhedra need expensive algorithms like Chernikova, resulting in exponential time.

Figure 9.2: Illustration of the two exponentialities associated with QRW algorithm, when it uses PolyLib for representation of its polyhedral objects.

as the context passed as parameters from dimension d to $d+1$. A separation at dimension d gives more polyhedral objects to be processed in the higher dimensions. Furthermore, the list of polyhedra processed at dimension d needs to be processed by a $\mathcal{V} \leftrightarrow \mathcal{H}$ conversion because of the operations in the next dimension $d+1$: separation at step d which is best done with $\mathcal{V}$-form is followed by an intersection at step $d+1$ which is best done with $\mathcal{H}$-form.

Our reduction of exponential cost of code generation to polynomial time is based on CLooG-PolyLib implementation, which uses the PolyLib library for representation of its polyhedral objects. In this chapter, we would be referring to only CLooG-PolyLib. Hence, we would informally be referring to these complexities as *the two exponential complexities* in polyhedral code-generation.

9.1.3 Our goal, the inputs and some hypotheses

In this section, we define the goal of this chapter, followed by understanding the restricted variety of domains that the sub-polyhedral code-generation algorithms take as inputs. Since there are bound to be domains which are not of this specific form, we suggest some methods of preprocessing the inputs which are not of this restricted variety.

TVPI input: The primary goal of this chapter is to analyze and suggest improved QRW-algorithms in the case when each domain $\mathcal{D}_i$ is a (U)TVPI domain. Though we will specifically mention TVPI polyhedra in this chapter, many statements are valid for UTVPI polyhedra as well.

Legal input: Each domain $\mathcal{D}_i$ can be a parametrized domain, but in a limited way. For example, if we consider indices i, j and parameters M and N, $i + j \leq 10$ is allowed and so are $i \leq N + 10$ and $M \leq N$; but $i + j \leq N$ or $i \leq N + 2M$ are not. Though this restriction seems limited when compared to general convex constraints, constraints like $i + 2j \leq 10$ as well as $i + 10 \leq 2N$, which cannot be encoded by interval

"box" constraints are encoded by TVPI. Since the meaning of variables is known, we refer to constraints without distinguishing the indices from the parameters.

There are two ways of managing domains which are not already TVPI:

TVPI over-approximation of domains: Domains which are already not TVPI can be over-approximated first and de-approximated later. This approximation and de-approximation process can happen on a per-statement basis. These over-approximations act like a padding of the domains of the individual statements so that additional integer points, that do not belong to the actual specified domain of the statement can be scanned. The padding is simplified in a post-processing phase, after the QRW code generation. Since the per-statement domains are likely to be very small constraint systems, the over-approximation algorithm based on the generator representation could be used. The over-approximation algorithms in Chapter 4.3 on page 50, with examples illustrated in Figure 4.3 on page 50 can be used.

Instantiation of parameters. Parameters can be instantiated into constants using extraneous domain information. For example, if it is known statically that $N < M$, then N and M can be assigned constants sufficiently large or sufficiently characteristic of the particular input so that constraints involving N and M can be analyzed as TVPI constraints. For example, constraint $i + j \leq N$ becomes $i + j \leq 100$ and constraint $i - j \leq M$ becomes $i - j \leq 200$; also, constraint $i + j \leq N + 2M$ results in the constraint $i + j \leq 500$ with $N = 100 \wedge M = 200$ as instantiation.

9.1.4 Potential of TVPI sub-polyhedra

Suppose we do begin with some of the above mentioned hypotheses: for example, the over-approximation of each domain by sub-polyhedra, followed by possible instantiation of parameters so as to reduce the number of active variables per constraint. The questions we want to answer in this chapter are the following:

- Can we give a guaranteed time complexity for code generation assuming the underlying method is CLooG-PolyLib implementation of QRW algorithm? For the above, do (U)TVPI sub-polyhedra help?
- Is the guaranteed time-complexity polynomial as well (like POLY(Stmts, Constr, Loops) etc.)? Which of the two exponential steps get reduced to polynomial time complexity and how?
- Since the gain in complexity requires a trade-off in the other metrics of code size and/or generated code execution time, what will the (U)TVPI-approximation lose when compared to general convex polyhedra?

Briefly, the answers to the above questions are: TVPI sub-polyhedra indeed do help in polynomial time code generation by reducing *both* of exponential steps to polynomial time. The generated code will be intermediate to, and in between what is generated by a naïve method using interval polyhedra, and the most sophisticated method of QRW as available in CLooG. The loss of precision—measured in terms of scanning the inactive code—would be due to two reasons: the approximation of non-TVPI domains into TVPI polyhedra and because of using TVPI sub-polyhedra.

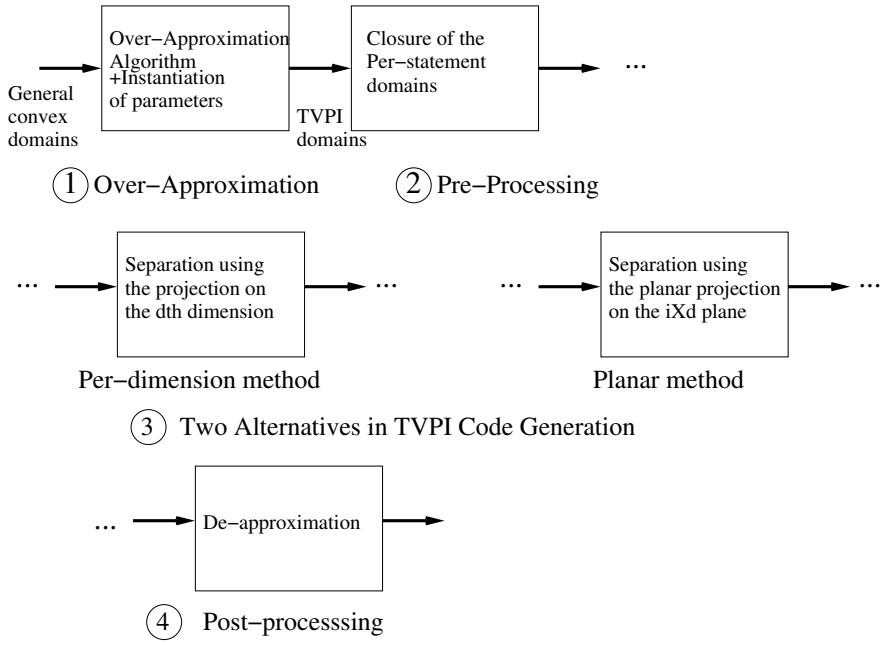


Figure 9.3: An Overview of Code Generation using TVPI polyhedra (QRW-TVPI)

Our exposition would alternate between the closure properties, graphical representation and geometric properties of TVPI polyhedra. Here is a brief summary of our approach.

A summary of our approach. We primarily rely on polynomial time geometric and closure operations offered by (U)TVPI polyhedra. As discussed in Chapter 3, the polynomial time closure operations let us extend the geometric algorithms developed for planar polyhedra to general TVPI polyhedra. This implies that the projection and separation of polyhedral objects, which are the two key polyhedral operations used by the QRW algorithm, can be performed in guaranteed polynomial time for TVPI sub-polyhedra.

Specifically, we exploit the fact that, while the enumeration of generators of general convex polyhedra takes exponential time because of the inherent nature of generators, the enumeration of the vertices of the $n(n-1)/2$ planar projections of TVPI polyhedra can be done in polynomial time. If we begin with TVPI sub-polyhedra as domains, we can design algorithms that allow for guaranteed polynomial time projection and separation operations.

These ideas result in the design of a sub-polyhedral code-generation schema: QRW-TVPI and two consequent specializations of the QRW algorithm using TVPI sub-polyhedra: QRW-TVPI1, a per-dimensional algorithm with lesser precision but using linear time separation complexity, and QRW-TVPI2, a planar algorithm with polynomial time complexity which uses a closure-based separation method. This second one comes with two specializations: a heuristic-based one with lower precision and a closure-based algorithm which provides better control overhead of the generated code.

Our methods are illustrated in Figure 9.3.

9.1.5 An overview of this chapter

We begin with a quick reminder of the properties of TVPI sub-polyhedra in Section 9.2. This section is a rewrite of material in Chapter 3, with some additional key lemmas and remarks.

In Section 9.3, we introduce QRW-TVPI: a TVPI code generation schema based on the polynomial properties of TVPI polyhedra. This can be used to create two variations of the QRW algorithm each of which provide a guarantee in the complexity. In Section 9.4, we present QRW-TVPI1, which is a simple per-dimension variation of the method. This is followed in Section 9.5, by the introduction of QRW-TVPI2, a variation of QRW algorithm for TVPI polyhedra. Both these algorithms are small variations of the QRW algorithm and reduce the two exponential steps in the QRW algorithm to polynomial time.

In Section 9.6, we quickly re-summarize some additional related work. Finally, this chapter is concluded in Section 9.7 with a list of contributions, perspectives and comments about related work.

9.2 TVPI Polyhedra Properties: A Quick Summary

The following result is by Graham [Gra72, dBKS00] ("Graham's scan" algorithm) shows the polynomiality of finding the vertices of planar polyhedra (polygons). It has been improved by Simon and King [SK04] to the problem of finding the generators (both vertices and rays) of planar polyhedra taking many special geometric cases of unbounded polyhedra, lineality space etc., into consideration:

Lemma 9.2.1 [Convex hull of planar $\mathcal{H}$-Polyhedra]. Given a planar polyhedron with m constraints, its generator representation (convex-hull) can be found in worst-case polynomial time $\mathcal{O}(m \log m)$.

The above very low complexity polynomial time algorithm is to be contrasted with the corresponding exponential time complexity for 3-dimensional and higher dimensional (general convex) polyhedra of the usual Chernikova algorithm [Ver92]. This latter exponential time is not a limitation of the Chernikova algorithm, but characteristic of the output itself which could be up to $n^{\lfloor \frac{m}{2} \rfloor}$ size on non-planar polyhedra. Even the classic interval "box" polyhedra (a hypercube) in n-dimensions which is the simplest case of sub-polyhedra has an exponential number of vertices. The above complexity result holds true for TVPI sub-polyhedra as well: the problem of enumerating *all their vertices* takes exponential time.

Polyhedral vs. graphical methods for TVPI polyhedra. It is well known that for general convex polyhedra, obtaining the exact projections on any 2-dimensional plane[34] takes *exponential* time using either of the Chernikova or Fourier-Motzkin methods. But, following the classic works of Shostak [Sho81] and Aspvall and Shiloach [AS80], the breakthrough in TVPI algorithms by Hochbaum and Naor [HN94] is in noticing that the 2-dimensional projections of a TVPI polyhedron can be computed *using graph-theory methods*, rather than polyhedral ones. On a 2-dimensional projection, using polyhedral methods, however, does not imply exponential time because of the polynomial complexity of the Graham's scan algorithm (see above).

[34] This corresponds to what the Omega paper [Pug92] calls as *2d-dark-shadow*.

Given a general TVPI polyhedron, let the *breakpoints* be the set of vertices associated with the i-th dimension. In other words, they define the envelopes of each of the $i - j$ planes for $1 \leq j \leq n; j \neq i$. The following result is from Hochbaum and Naor [HN94]:

Lemma 9.2.2 [Number of breakpoints]. Given a general TVPI polyhedron with m constraints in n dimensions, the number of polyhedral vertices[35] associated with any dimension i is linear in the total number of constraints: $\mathcal{O}(m)$. (The actual number is $m + 4n$.)

The following result is also by Hochbaum and Naor [HN94].

Lemma 9.2.3 [Fourier-Motzkin on TVPI polyhedra]. The Fourier-Motzkin elimination to determine the feasibility of general TVPI polyhedra—as opposed to planar polyhedra—can be performed in strongly polynomial time with complexity $\mathcal{O}(mn^2 \log m)$.

As noted in Chapter 3, the above result is surprising, because FM is known to take doubly exponential time on general convex polyhedra, taking up to $\lfloor \frac{m}{2} \rfloor^{2^n}$ time. The FM method on TVPI polyhedra is related to the following corollary of TVPI polyhedra, again due to Hochbaum and Naor [HN94]:

Corollary 9.2.4 [Projection and TVPI polyhedra]. Given a TVPI polyhedron in n dimensions, projecting out a dimension (variable) results in a TVPI polyhedron in $n - 1$ dimensions. The complexity of this projection is $\mathcal{O}(mn \log m)$.

This means that TVPI polyhedra are closed under projection. The intuition for this can be obtained in a geometric way or a graph theoretical way. The graph theoretic intuition for the same is shown in Figure 9.4. All the TVPI algorithms involve a simple graph-based representation of the TVPI constraints and a polynomial time closure. This operation is cubic: $\max(n^3, m^3)$.

Closure properties. The closure operation on TVPI polyhedra adds additional constraints to its $\mathcal{H}$-form; these additional constraints can be considered to be all possible projections of the TVPI system. Using this so-called closed system, the projection of the system on the plane $x_i \times x_j$ can be inferred using only the constraints in which x_i and x_j have non-zero coefficients. Furthermore, obtaining the closure simplifies many operations involving multiple TVPI systems. For example, convex union of two TVPI-systems P_1 and P_2 can be done by first obtaining the closure of both P_1 and P_2, followed by performing a planar convex-hull on each of the $\frac{n(n-1)}{2}$ planar systems, with the overall method having very low complexity. The conventional method on the other hand, would involve finding the $\mathcal{V}$-form of each of the systems and hence has high computational complexity.

Graph theoretically, as TVPI polyhedra can be represented using directed graphs, closure can be said to be adding additional directed edges to the graph with specific weight so that the sub-graph induced by vertices x_i and x_j can be used to infer the properties between variables x_i and x_j. This is shown in Figure 9.4.

[35] Hochbaum and Naor [HN94] refer to these as breakpoints.

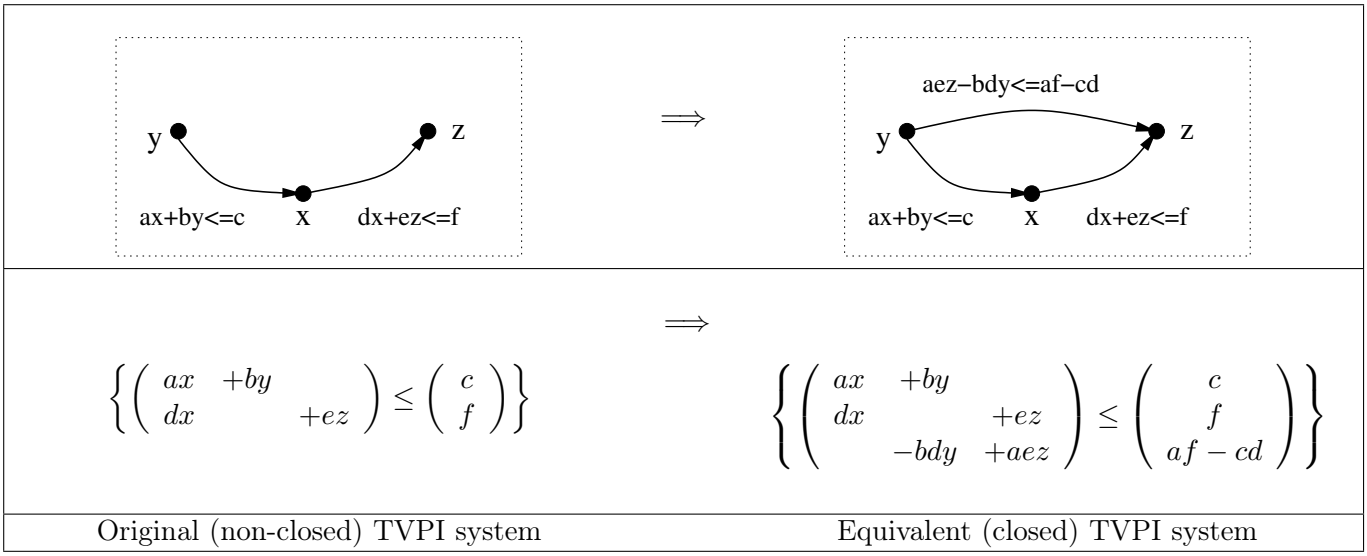


Figure 9.4: Closure of TVPI constraints. If $a > 0 \wedge d < 0$ (and $b < 0 \wedge e > 0$), an additional transitivity edge can be added between variables y and z resulting in a closed system.

Lemma 9.2.5 [Differences of two planar polyhedra]. Given two planar polyhedra P_1 and P_2, with a total of m constraints, their differences $P_1 - P_2$ and $P_2 - P_1$, are computed in time polynomial in m: POLY(m).

Though we do not provide the details of the above crucial step, the main proof of its polynomiality is in Lemma 9.2.1, which guarantees the polynomiality of the convex hull operation for planar polyhedra. Even less efficient algorithms like the one by Avis and Fukuda [AF92] which takes quadratic time, or an out of box implementation of the Chernikova algorithm [Ver92]—which is readily available from a library like PolyLib—which takes a cubic time, could be used, while still guaranteeing a polynomial time. Since the intersection of two polyhedra $P_1 \cap P_2$ can also be perfomed in polynomial time, we can say that separation of two planar polyhedra can be done in time polynomial time.

9.3 Our Schema for TVPI-based Code Generation Algorithms

The general schema for sub-polyedral code generation methods in this chapter:

Algorithm QRW-TVPI: CG using TVPI sub-polyhedra.

1. Over-approximate the domains to result in TVPI polyhedra.
2. Preprocess the individual TVPI polyhedral domains.
3. Run a variation of QRW algorithm.
4. De-approximate each of the over-approximations.

The above algorithm schema builds from the ideas introduced in Section 9.1. We explain each of its steps in the following sub-sections below.

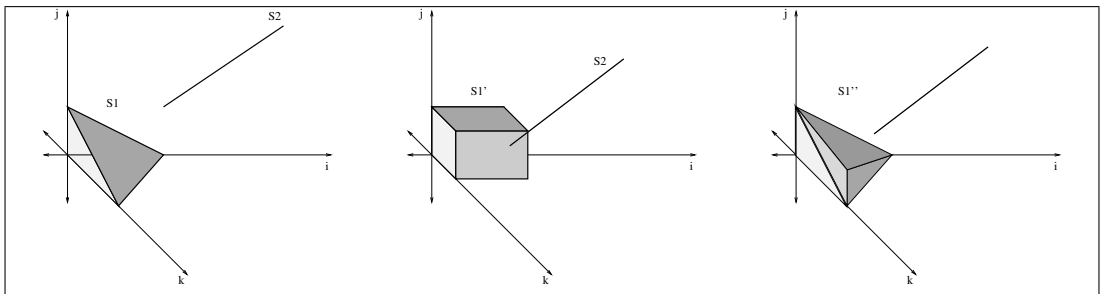
9.3.1 Over-approximation and de-approximation

In Step 1 of QRW-TVPI, we do the over-approximation of the domains, followed by a possible instantiation of the parameters. The overall goal of the step is to ensure that all the constraints of all domains given to the QRW-TVPI algorithm are in the (U)TVPI form. As this step is performed on a per-statement basis, it can as well use a costly algorithm that uses the $\mathcal{V}$-form. The effectiveness of the TVPI based code generation methods depends on the OA method, as statements whose projections do not originally intersect may intersect. Hence a costly algorithm, which uses the generator representation, could be used.

Miné [Min06, Section 4.3] adapts the vertex method for finding the UTVPI-OA of a general polyhedron. This could be used as the per-statement systems are extremely small. Simon et al. [SK10, Section 3.2.6] propose an iterative algorithm for the TVPI-OA of a general polyhedron using Linear Programming. These algorithms are briefly summarized in Section 4.3 on page 50.

This over-approximation however should be carefully designed, as otherwise, it may lead to inseparability of the over-approximations when the original statements are separable. In Figure 9.5, we show a 3-dimensional extension of the figure from Quilleré et al. [QRW00, Figure 1, page 472]. The projection of Figure 9.5.b on the (i, j)-plane results in the QRW example.

Example 9.3.1 [OAs and precision]. Figure 9.5.a shows two statements S_1 and S_2 in three dimensions. The statement S_1 has a domain that is non-TVPI ($D_{S_1} = \{0 \leq i, j, k; i + j + k \leq 1\}$) while the statement S_2 has a TVPI domain ($D_{S_2} = \{k = 1; i = j; 1 \leq i, j \leq 2\}$). The original domains do not intersect and can be separated using the current scheme in CLooG-PolyLib. S'_1 and S''_1 are two possible TVPI over-approximations of domain of S_1. As shown in Figure 9.5.b, using the TVPI over-approximation S'_1 for S_1 leads to intersection of the domains of S'_1 and S_2 and more complicated separation cases. In contrast, as shown in Figure 9.5.c, the over-approximation S''_1 for S_1 does not intersect with S_2 leading to separability of the two statements S''_1 and S_2. The former will lead to guards for the cases and hence code with more control overhead, while the latter would lead to more simpler code with less control-overhead. □



(a) Original domains: the domain of S_1 is not TVPI, while the domain of S_2 is. The QRW method can do exact separation.

(b) A naïve TVPI approximation: S_1 has been over-approximated into an TVPI (interval) domain S'_1, which intersects with S_2.

(c) A better TVPI approximation: S_1 has been over-approximated into a TVPI domain S''_1 which does not intersect with S_2.

Figure 9.5: Per-statement over-approximation as pre-processing of code generation and consequent loss of precison

Loss of precision by OA. As illustrated in Figure 9.5.b, the OA leads to possible control overhead in the TVPI code generation scheme. Even when the original domains are disjoint, the over-approximations and their projections can intersect with each other: This is because $S_1 \cap S_2 = \phi \not\Rightarrow \mathrm{OA}(S_1) \cap \mathrm{OA}(S_2) = \phi$.

De-approximation. Step 4 of QRW-TVPI is the de-approximation step and is the conjugation of Step 1. It is similar to the post-processing of Omega, or to the backtracking and pattern-matching phase in CLooG.

9.3.2 Preprocessing and variations of QRW algorithm

In Step 2 of QRW-TVPI, we preprocess the domains. Then Step 3 with a small variation of the QRW algorithm, sketched in Section 9.1.1, is applied. These variations are different for the two algorithms that we introduce in this section.

Preprocessing. Each of the algorithms in the QRW-TVPI schema, namely QRW-TVPI1 and QRW-TVPI2, need some preprocessing similar to Hochbaum-Naor's, which is finding the generators of all the planar projections. Alternatively, each of the per-statement TVPI domains could be subjected to closure in their graph theoretical representation. These steps reduce the overall complexity since the algorithms we introduce are based on utmost pair-wise planar projections.

The variations of QRW algorithm are based on the observation that in steps 1–3 of the QRW algorithm, the d-dimensional projections are polyhedral objects in the separation algorithms. These operations can be done with guaranteed polynomial time on TVPI polyhedra. At dth step of recursion of the original QRW algorithm, the separation is using the d-dimensions projections of each of the domains. So, the specializations depend on the following key observation:

Lemma 9.3.2 [Sub-dimensional projections for scanning]. Assume that the scanning order of the dimensions in the QRW algorithm is $x_1, x_2, \ldots, x_d$, with x_1 being the first dimension that has been scanned, and x_d being the current dimension being scanned ($d \geq 2$). Any separation scheme which looks at the projection of the current d-dimensional domain on only some of the dimensions from $\{x_1, x_2, \ldots, x_d\}$ is valid as long as the (current) dth-dimension (x_d) is also being considered.

The above lemma means that a separation of the domains which looks at only the dth dimensional 1-dimensional objects (which are interval ranges) is valid. So is the separation of domains which looks at the 2-dimensional projections, like the ones on $x_{d-1} \times x_d$ or $x_1 \times x_d$ planes. Sub-dimensional scheme for the dth dimension that looks at only the projection on $x_{d-2} \times x_{d-1}$ plane however, is not valid.

This observation leads to two variations of the QRW-TVPI scheme: QRW-TVPI1 which uses a per-dimensional projection and separation method, and QRW-TVPI2 which uses a planar projection and separation scheme[36].

[36]Since both methods rely heavily on the Fourier-Motzkin of Hochbaum-Naor, they can alternatively be called FM-TVPI1 and FM-TVPI2, but for the fact that they do a separation and sorting after projection, similar to the QRW algorithm.

QRW-TVPI1: a per-dimension separation. In this variation, the separation of the polyhedra in Step 3 of QRW is on a per-dimensional basis, using only the dth dimension. Consequently, in Step 2 of this method, the projection is found on the dth dimension only. The main intuition for this method is that for each domain of each statement, all the dimensions other than dimension d can be eliminated (projected away) in guaranteed polynomial time using Hochbaum and Naor's Fourier-Motzkin elimination method. The correctness of this method is clear from the above Lemma 9.3.2. The polynomiality of this method is from the polynomiality of Hochbaum-Naor's algorithm in Lemma 9.2.3. We discuss this method and its limitations in Section 9.4

QRW-TVPI2: closure based planar separations. In this variation, the separation of the polyhedra in Step 3 of QRW is on a planar basis. Since choosing the dth dimension x_d is mandatory, we can use the planar projection on the $x_i \times x_d$ plane (for $d \geq 2$ and for any i such that $1 \leq i \leq d-1$). The key design issue is which i is to be chosen, among the $d-1$ possible choices. A possible heuristic is to fix $i = d-1$, but another possible alternative is to generate code for all the $d-1$ planes $\{x_i \times x_d \,|\, 1 \leq i \leq d-1\}$, and to choose the best of the projections among different i's according to a metric. Again, the correctness of this variation is clear from Lemma 9.3.2. The polynomiality of this method is from Lemmas 9.2.3–5. We discuss this method and its limitations in Section 9.5.

9.4 QRW-TVPI1: a Per-dimension Separation Method

In this section, we discuss the QRW-TVPI1 algorithm, its strengths and weaknesses. In Section 9.4.1, we study the QRW specialization for TVPI polyedra, with an example. We study its complexity and other metrics and some limitations in Section 9.4.2.

Preprocessing. As algorithm QRW-TVPI1 just involves finding the dth dimensional projection of TVPI polyhedra, the preprocessing involves finding the TVPI approximations of each per-statement domains, followed by possible instantiations. The algorithm needs only the 1-dimensional ranges of each statement on each dimension.

9.4.1 QRW-TVPI1: Algorithm, details and example

The algorithm QRW-TVPI1 is given in Figure 9.6. It is nothing but a simple per-dimension specialization of QRW algorithm. Due to the similarity with the latter, we highlight the differences with the QRW algorithm sketched in Section 9.1.1.

Details. Steps 1, 4–6 are identical to the QRW algorithm. In Step 2, the domain TVPI polyhedra is projected on dimension x_d, for which, the Fourier-Motzkin of Hochbaum-Naor's algorithm can be used. In Step 3, these projections are separated into disjoint polyhedra. Since these projections are linear interval ranges, a simple sort based scheme can be used, which also simplifies Step 4.

Input: A polyhedron list, a context C and the current dimension d.
Output: The AST of the code scanning the input polyhedra.

1. **Intersect** each polyhedron with the context C.
2. **Project** the polyhedra onto the x_d-dimension.
3. **Separate** these 1-dimensional projections (intervals) into disjoint polyhedra. (This step generates loops for dimension d.)
4. **Sort** the loops to respect the lexicographic order.
5. Recursively generate loop nests that scan each list with dimension $d+1$, under the context of the dimension d.
6. Return the AST for dimension d.

Figure 9.6: Algorithm QRW-TVPI1: Per-Dimension TVPI Specialization of QRW

Example 9.4.1 [QRW-TVPI1]. In Figure 9.7, we illustrate the steps of the algorithm which does the per-dimensional projection and scanning. The first dimension (i) does the exact separation, just like QRW and then generates the code for scanning. However, for scanning on the second dimension (j), the algorithm again obtains the per-dimensional projections using Fourier-Motzkin and uses them for separation of the projections. These separations are used for generating code for scanning this dimension. On the other hand, the QRW algorithm does exact separation of the domains, obtaining minimal control-overhead. The figure also shows the extra code which is scanned by QRW-TVPI1. □

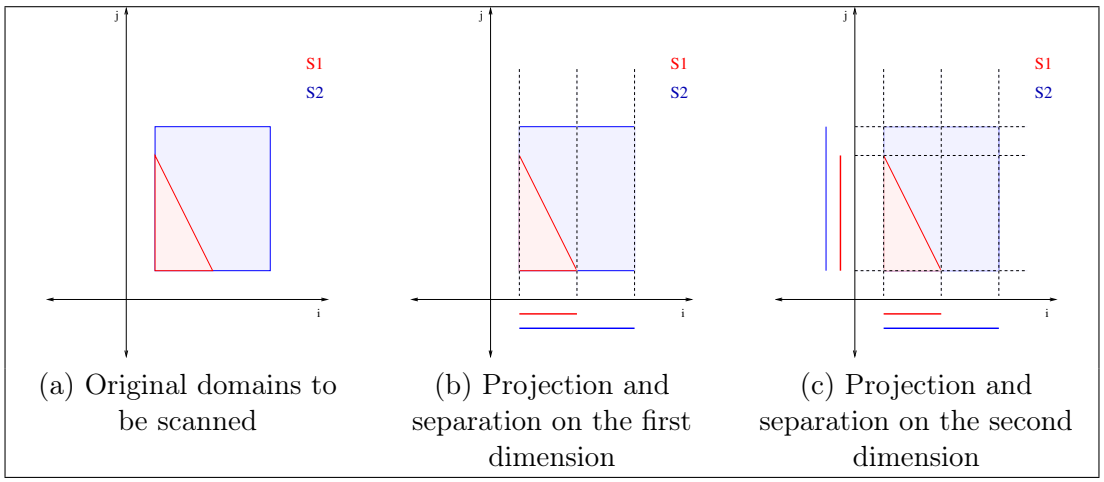
9.4.2 Complexity, metrics and discussion

We have the following two results about the complexity:

Projection and separation. Step 2 can be done in strongly polynomial time because of Lemma 9.2.3 for the Fourier-Motzkin of Hochbaum-Naor.

In Step 3, there is a separation of the linear interval ranges as in the QRW algorithm, though specialized for linear interval ranges. This is followed by sorting of the interval ranges. Both of these can be done in linear time thanks to the following lemmas.

Convexity of statement spans. The domain of each statement is a convex polyhedron. So, its rational projection is a convex region as well. The statements can be considered to have lifetimes. A statement has a beginning and ending of its lifetime. A statement spans contiguous set of breakpoints. Once the statement is finished for the last time, it never "restarts". For showing the polynomiality, we need the following lemma from from the theory of chordal perfect graphs [Gol04].



(a) Original domains to be scanned

(b) Projection and separation on the first dimension

(c) Projection and separation on the second dimension

Figure 9.7: QRW-TVPI1: per-dimension separation and code generation for two statements on a planar polyhedral domain. Also shown are projections of the two statements on the two axes which enable the code generation.

Lemma 9.4.2 [Statement Elimination Order of Statement]. The statement endings form a linear Perfect Elimination Order (PEO).

The above is a straightforward result from the convexity of the original domains. The above result means that a Lexicographic Breadth First Search (LexBFS) on the statement endings can be done as as suggested by Rose, Leuker and Tarjan [RTL76]. This means that the sorting can be done in linear time, where the ranges are iterated upon by maintaing an annotated list of statement ranges. A statement enters this list at the beginning of its lifetime and gets removed at its last occurrance.

Lemma 9.4.3 is a consequence from the above results.

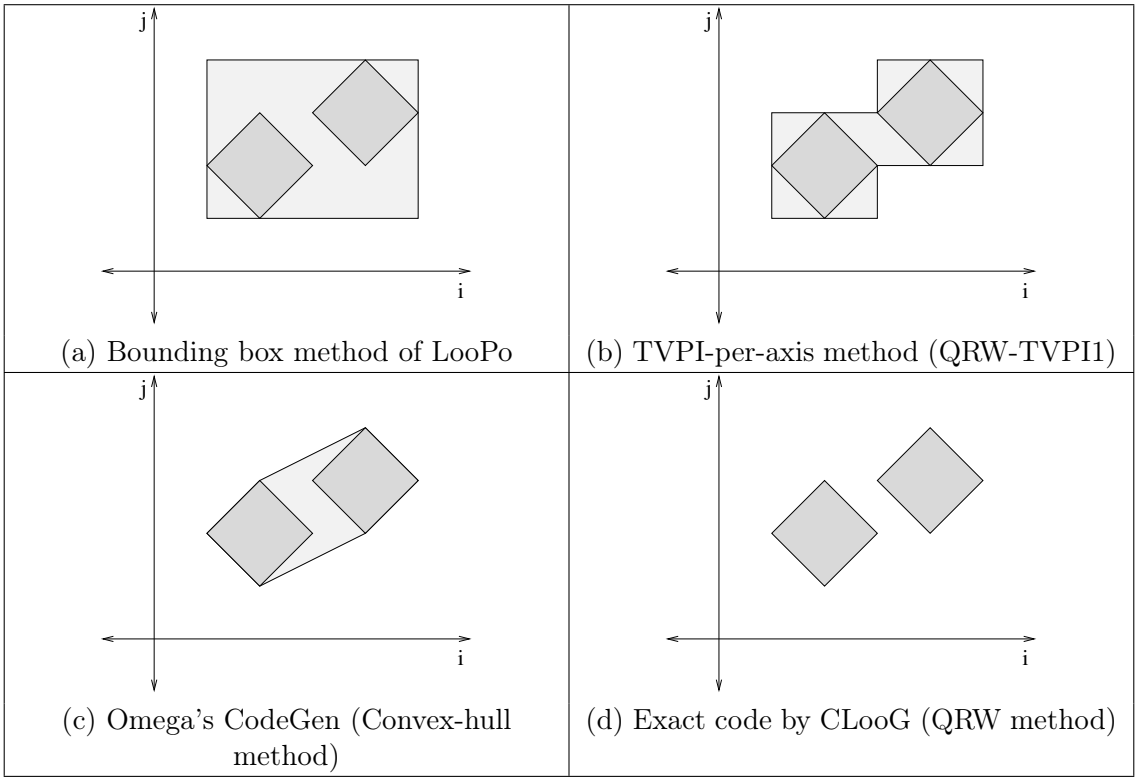
Lemma 9.4.3 Projection, separation and sorting in QRW-TVPI1 can be done in polynomial time. Hence QRW-TVPI1 does code generation in polynomial time.

We show a limitation of the code generation method in the following example:

Example 9.4.4 [A hierarchy of precisions]. The example in Figure 9.8 shows that the QRW-TVPI1 can help in generating code which is more powerful than one generated by a simple bounding box, but less powerful than Omega's CodeGen, with all of them being less powerful than the QRW's scheme in CLooG. □

9.5 QRW-TVPI2: Closure Based Planar Separations

In this section, we extend the per-dimension variation studied in the previous section to introduce planar variations. In Section 9.5.1, we study the QRW specialization for planar polyedra, and illustrate it with an example. Then, we study its complexity, other metrics and its limitations, and list open questions in Section 9.5.2.



(a) Bounding box method of LooPo

(b) TVPI-per-axis method (QRW-TVPI1)

(c) Omega's CodeGen (Convex-hull method)

(d) Exact code by CLooG (QRW method)

Figure 9.8: Code generation with increasing levels of precision for two sets of `for` loops with alternative methods. The *non active areas* scanned by the methods decreases from (a) to (d). They are maximum for the (a) bounding box method and zero for (d) QRW method in CLooG. The polynomial time guarantee of TVPI polyhedra is traded (for a lot of) control overhead by having to scan non-active code.

9.5.1 QRW-TVPI2: Algorithm, details and example

The algorithm QRW-TVPI2 is given in Figure 9.9. Algorithm QRW-TVPI2 is a planar specialization of the QRW algorithm. Again, due to the similarity with the latter, we highlight the differences with the QRW algorithm sketched in Section 9.1.1.

Details Just like in QRW-TVPI1, Steps 1, 4–6 of this algorithm are identical to the QRW algorithm. In Step 2, the TVPI over-approximated domain are projected on the plane $x_i \times x_d$. This is by the Fourier-Motzkin of Hochbaum-Naor's algorithm. Alternatively, it could be done by closure-based graph theoretical methods—where the per-statement TVPI domains are subjected to closure—so that the overall effort is amortized. In Step 3, these projections are separated into disjoint polyhedra as in QRW algorithm.

Example 9.5.1 [QRW-TVP2]. In Figure 9.10 we illustrate the steps of the algorithm QRW-TVP2 which performs planar projection and scanning. The scanning order of the dimensions is $i \rightarrow k \rightarrow j$, and as shown in Figure 9.10.a, the domains are TVPI polyhedra to begin with. In this particular example, the dimension k is degenerate as the `for` loops for that particular dimension are but simple `if` cases. This was simply for purpose of geometric illustration of the two cases.

The first dimension to be scanned is i. The algorithm does exact separation just like QRW, using linear

Input: A polyhedron list, a context C and the current dimension d.
Output: The AST of the code scanning the input polyhedra.

1. **Intersect** each polyhedron with the context C.
2. **Project** the polyhedra onto the $x_i \times x_d$-plane.
   ($\{i \mid 1 \leq i \leq d-1\}$ is found by a search mechanism, or fixed by a heuristic.)
3. **Separate** these 2-dimensional projections into disjoint polyhedra.
   (This step generates loops for dimension d and lists of polyhedra for dimension $d+1$.)
4. **Sort** the loops to respect the lexicographic order.
5. Recursively generate loop nests that scan each list with dimension $d+1$, under the context of the dimension d.
6. Return the AST for dimension d.

Figure 9.9: Algorithm QRW-TVPI2: Planar Specialization of QRW algorithm

interval polyhedra. This is shown in Figure 9.10.b. For generating code for the next dimension k, as the context is two dimensional (i, k), the algorithm does a Fourier-Motzkin on the plane (i, k) and does exact separation, again just like QRW. This is shown in Figure 9.10.c.

For generating code for the third dimension (j), as the context is 3-dimensional (i, k, j), the algorithm can use either of the (i, j) or (k, j) planar projections for separating and consequently for generating code. Assuming that the algorithm has to use the plane (i, j), it does exact separation, just like QRW algorithm. This is shown in Figure 9.10.d. □

9.5.2 Complexity, metrics and discussion

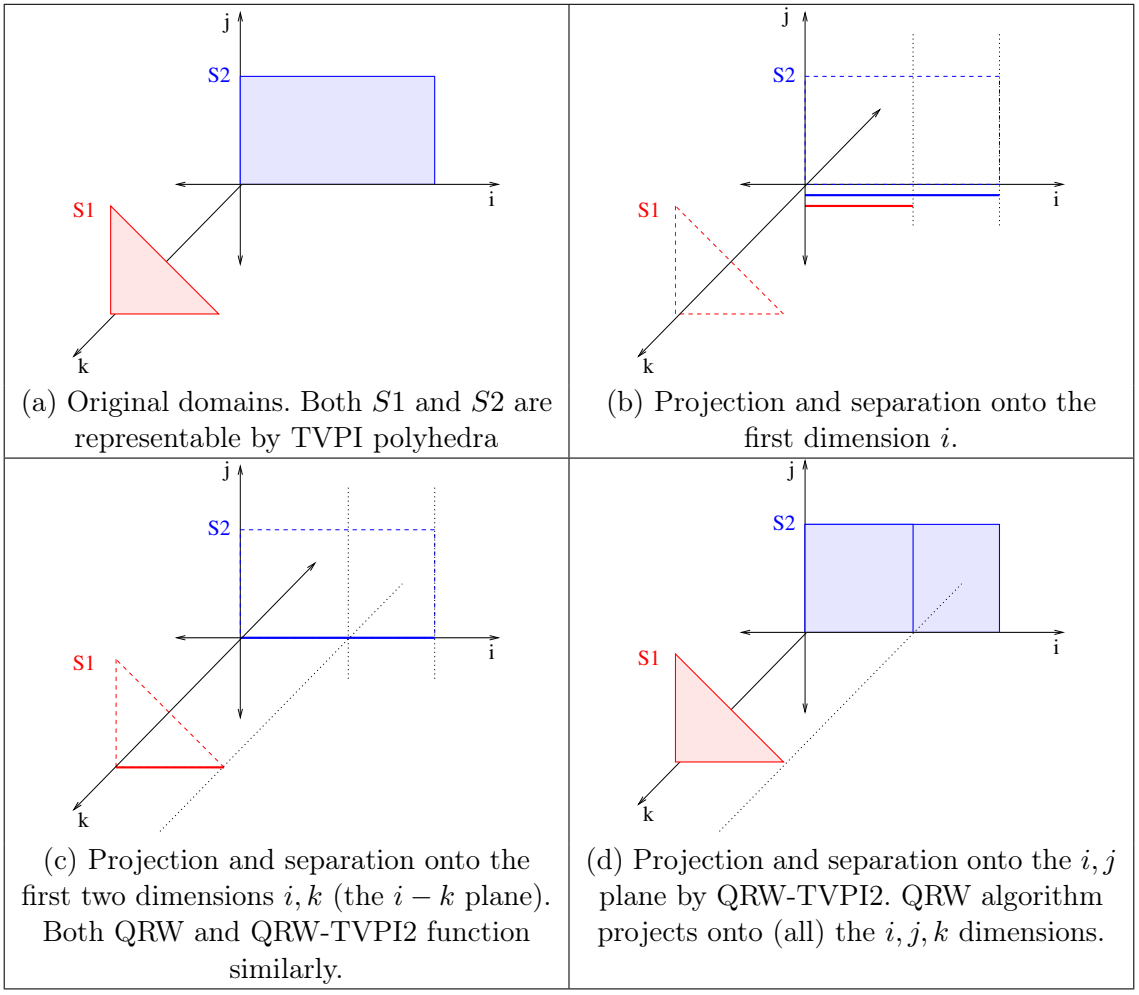
In this section we discuss the complexity and limitations of this algorithm.

Complexity. The projection of Step 2 can be performed by Hochbaum-Naor's Fourier-Motzkin in strongly polynomial time. Alternatively, as the per-statement domains can be represented using graph representation, closure based method can be used for ensuing a polynomial time. Step 3 can be done in polynomial time because of Lemma 9.2.5.

Since the key operations of intersect, project and separate operations can be done in polynomial time, we have the following lemma:

Lemma 9.5.2 [Complexity of QRW-TVPI2]. Code generation algorithm for TVPI polyhedra using the planar method QRW-TVPI2 has polynomial time complexity.

Clearly, QRW-TVPI2 is more powerful than QRW-TVPI1. The domains in Example 9.4.1 as well as in Example 9.4.4 can be scanned exactly by this algorithm without loss of precision because the domains



(a) Original domains. Both $S1$ and $S2$ are representable by TVPI polyhedra

(b) Projection and separation onto the first dimension i.

(c) Projection and separation onto the first two dimensions i, k (the $i - k$ plane). Both QRW and QRW-TVPI2 function similarly.

(d) Projection and separation onto the i, j plane by QRW-TVPI2. QRW algorithm projects onto (all) the i, j, k dimensions.

Figure 9.10: QRW-TVPI2: planar separation and code generation for two statements on 3-dimensional domain. Also shown are projections of the two statements on the two axes which enable the code generation. The scanning order is $i \rightarrow k \rightarrow j$.

there are planar and hence representable by TVPI polyhedra. For $2d$-loops, loops with at most two indices, the algorithm works exactly as QRW algorithm. But, for higher dimensional loops, it offers a choice.

Mechanism for selecting x_i: In the above QRW-TVPI2 algorithm, for each $d \geq 2$, a dimension $\{i \,|\, 1 \leq i \leq d-1\}$ has to be chosen. Among these $d-1$ choices, we suggest the following alternatives for selecting x_i.

- **Search:** Since there are $d-1$ possible choices, a linear search can be done. The key issue here is designing a cost function which can be used as an evaluation metric: namely, which $\{i|\, 1 \leq i \leq d-1\}$ ensures that code generated using the $x_i \times x_d$ plane is the best?
- **Heuristic:** Alternatively choose i to be $d-1$ (or even to any predetermined $\{i|\, 1 \leq i \leq d-1\}$), such that code is always generated using the projection on the $x_i \times x_d$ plane. This method has the advantage of very simple implementation, though there could be loss of precision when better alternatives exist. For example, there may exist a $\{i' \,|\, 1 \leq i' \leq d-2\}$ such that code generated by the plane $x_{i'} \times x_d$ is much superior to the one generated by the choice of $x_{d-1} \times x_d$.

To show the limitations of the heuristic and the power of the search mechanism, we use Example 9.5.1 again.

Example 9.5.1 (contd.) [search vs. heuristic]. In this example, when generating code for the third dimension j, there exists a choice as to which planar projection to choose for separation: $i \times j$ or $k \times j$. If the algorithm QRW-TVPI2 uses the plane $k \times j$, then, the code generated for the dimension j is trivial: it is equivalent to what the less powerful per-dimensional scheme QRW-TVPI1 generates.

Instead, as illustrated in Figure 9.10.d, choosing the planar projection $i \times j$ for the same purposes would mean that the code generated is *exact*, and equivalent to the one generated by QRW algorithm. □

The above example shows that the heuristic selection of i is limited and can result in a poor choice for code generation. It also shows that there exists a choice while selecting a planar projection for separating the domains. The QRW algorithm does not need to make this choice as it can use the d-dimensional polyhedra for separation. By making this choice however, QRW-TVPI2 runs in guaranteed polynomial time, while being more powerful than QRW-TVPI1.

We leave this sub-section with the following open questions:

Open questions:

- In the search mechanism suggested, what are possible metrics to evaluate the best of the $\{x_i \times x_d \,|\, 1 \leq i \leq d-1\}$ planes?
- Among these $d-1$ choices, can the metric provide a control in the generated code, similar to CLooG's `-l` and `-f` options?
- Also, for real world examples, how effective are the above alternatives in terms of reduction of scanning of inactive code?

- Furthermore, can the separation problem in Step 3 of QRW-TVPI2 be reduced to a graph theoretical problem by making a correspondence between the above choices and the code generated?

9.6 A Summary of Additional Related Work

We summarize additional related work.

Code generation methods and QRW algorithm. An excellent summary of code generation methods is in the encyclopediac reference by Bastoul [Bas11]. The above reference helped in preparation of Section 9.1.1 on page 128 of this chapter. Quilleré et al's [QRW00, Bas04a] algorithm (QRW algorithm) whose extension is in Bastoul [Bas04a] and Bastoul [Bas04b, Chapters 6,7]. The algorithm is implemented in the widely used code-generator CLooG with many improvements. CLooG is designed so that the basic implementation within is independent of the library which supports the necessary polyhedral operations. The focus of this chapter is however on CLooG-PolyLib, which uses the PolyLib library for polyhedra operations.

Implementation of closure operation. An implementation of the algorithms suggested in this chapter could use the closure based implementations like TVPI by TVPI-lib [SKH02] or Octagons by Apron [JM09] or PPL [PPL].

Chen's recent PLDI work. Recently, Chen [Che12] improved CodeGen to CodeGen+. The latter is based on Omega 2.0 and it appears from the algorithm that it is quite similar to the one by Kelly, Pugh and Rosser [KPR95], with the improvements being limited to changes in the post-processing methods along with changes in the way control structures in the AST—in particular the if conditionals—are removed by post processing.

CodeGen, and hence CodeGen+ operate with the philosophy that the input should come with a lexicographic ordering of the statements. Such a method is quite different from how CLooG operates, which expects separate inputs for the domains and the scattering functions. This kind of separation gives CLooG more freedom in generating many varieties of code with its many flavors—like its two options `-l <depth>` and `-f <depth>`—which allow tuning the amount of control in the generating code, which Omega based code generators cannot do. This latter reason makes the paper under consideration [Che12] mistate that CLooG could generate incorrect code, while actually CLooG does not need to know about the correctness of the generated code: its responsibility solely being limited to respecting the input scattering functions which it does all along.

The implementation of CodeGen+ has also been compared to a version of CLooG. Among the metrics of code compactness (generated code size), code generation time, GCC compile time (of the generated code) and code performance, Chen's implementation reports dramatic improvements over CLooG on the code size (and consequently, the GCC compile time) and the code generation time metrics. It however appears from the numbers in the paper that the improvements of the executed code of CodeGen+ on CLooG are not significant. This is not surprising because the algorithm of QRW which is the main engine

of CLooG, generates exact code with no redundancy and hence improving on the executed code is not possible. On the other hand, improving on the complexity of the QRW algorithm is possible because of the cost of the algorithms used in it, and it is likely that the significant gains in code generation time of Chen's implementation are due to the core engine and the above improvements.

In spite of the improvements of Chen's algorithm over CLooG over running time, it is unlikely that offers asymptotic improvements over CLooG. This is because the basic engine of CodeGen+ is potentially as costly—if not more costly—as the basic engine of CLooG. It is likely that the Omega 2.0 does efficient pattern matching for common cases as suggested much earlier by Pugh [Pug92]. But, as we pointed out in Chapter 4, these heuristics of Omega need to be augmented with an appropriate over- or under-approximation methods so as to obtain asymptotic improvements.

9.7 Conclusions, Perspectives and Related Work

In this section, we review the conclusions and contributions of this chapter, as well as discuss some perspectives.

9.7.1 Conclusions and contributions

Conclusions. In this chapter, we showed how the polynomial complexities of TVPI polyhedral operators can be exploited to obtain polynomial time complexities for code generation. First, we identified the two sources of exponentialities in QRW algorithm: the separation of abstract polyhedral objects, and the complexities associated with the algorithms of the underlying polyhedral operations. Then, we proposed a TVPI code generation scheme which exploits the polynomial properties of TVPI polyhedra. Using this scheme, we presented two algorithms–a per-dimensional one, and a planar one–each of which are specializations of the QRW algorithm and which do the crucial polyhedral operations of projections and separation in asymptotically polynomial time. The insights for these algorithms are very simple: for TVPI polyhedra, finding the $2d$-projections for each pairs of dimensions, Fourier-Motzkin and hence, the crucial operations of polyhedral projection and separation can be performed in a guaranteed polynomial time.

Loss of precision. Though guaranteeing a polynomial time complexity, the algorithms presented exhibit two causes of precision loss. The first source of precision loss is the over-approximation method. This can lead to non-empty intersection of the over-approximated domains $\mathrm{OA}(S_1) \cap \mathrm{OA}(S_2) = \phi$, even when the original domains are disjoint, $S_1 \cap S_2 = \phi$. This leads to an increase of code-size, and hence an increase of compilation time of the generated code, and an increase of the control overhead and hence execution time the generated code.

The second source of precision loss is the projection and separation methods. In the per-axis (per-dimension) separation method QRW-TVPI1, the loss of precision can be significant. The planar schemes in QRW-TVPI2 are better. In these latter, the heuristic which uses the $x_{d-1} \times x_d$ plane for separation is less precise than the closure-based search scheme which generates the code for each of the $x_i \times x_d$ planes with $\{i \,|\, 1 \leq i \leq d-1\}$, and chooses the one with the *best* code that is generated, with best defined according to an appropriately pre-defined metric, like controlling the control-overhead.

Hierarchies of precision. Let us use the symbol $\leq$ to denote, rather loosely, a gradation of precision of two sub-polyhedral code-generation algorithms. Informally, it captures the quality of generated code in terms of control overhead, or equivalently the volume of non-active code scanned: $\mathcal{A}_1 \leq \mathcal{A}_2$ for two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ means that $\mathcal{A}_1$ has more control overhead, and hence scans more inactive code than $\mathcal{A}_2$. With the above notation, the algorithms introduced fit in the following hierarchy of code generation:

$$\underbrace{\text{QRW-TVPI1} \;\leq\; \text{QRW-TVPI2}_{\text{Heuristic}} \;\leq\; \text{QRW-TVPI2}_{\text{Search}}}_{\text{Polynomial Time}} \;\leq\; \underbrace{\text{QRW}}_{\text{Exponential Time}}$$

In addition to the algorithms listed above, we have the corresponding additional UTVPI specialization of the QRW algorithm: QRW-UTVPI1 and QRW-UTVPI2, which use similar ideas and all of which introduce the following hierarchy:

$$\text{QRW-UTVPI1} \leq \text{QRW-TVPI1} \leq \text{QRW-UTVPI2} \leq \text{QRW-TVPI2}$$

Each algorithm in the above hierarchy is polynomial because of the reduced complexities of projection and separation.

Contributions. The result in this chapter could be seen either as a new code generation method, or a more precise complexity analysis for CLooG when the domains are (U)TVPI polyhedra. We introduced some open problem which could be used to make a closer connection between the geometric methods used in polyhedral code generation, in particular the QRW algorithm and graph theory.

The material in this chapter has not been published anywhere before.

Memory improvements. As for sub-polyhedral scheduling, the memory savings of (U)TVPI polyhedra could be considerable, making the overall system very light on the system. We believe that, as suggested in Halbwachs et al. [HMG06], many techniques could be used to improve the memory requirement, but beyond an certain level, asymptotic improvement is not possible and sub-polyhedral methods like ours have to be used.

9.7.2 Perspectives and discussion

Advantages in using (U)TVPI polyhedra

Improvements over general polyhedra. When compared to general convex polyhedra, the improvements in complexities suggested by the TVPI specializations of QRW and its approximations can be gauged from the following: for normal polyhedra, obtaining the d-dimensional projections may take exponential time. Given a TVPI-OA of each domain, we can run QRW algorithm, but the d-dimensional separation algorithm takes exponential time. So, using (U)TVPI polyhedral over-approximations, along with the improvements to the QRW algorithm is one simple way in which the exponential parts can be reduced to polynomial time.

Practical utility and limitations. Alternatively, it also true that the exponentialities mentioned in Section 9.1.2 on page 129 are rarely observed and the running time of CLooG is quite fast. In fact, this has been reported many times in the polyhedral literature—for example by Wilde [Wil93] in PolyLib, Ancourt [Cor91] for PIPS, Pugh [Pug92] for Omega—that the exponentialies due to the polyhedral operations ("second exponentiality" in Section 9.1.2) in are seldom observed in practical inputs. Even the QRW paper suggests similar reasons for using PolyLib with its first implementation: that it scales well because the number of dimensions is small. Furthermore, CLooG-PolyLib has been a successful code-generator. But, this chapter is about improvement of theoretical complexity using better guarantees offered by (U)TVPI

polyhedra. Beyond the above limitations, the new methods proposed also do need a backtrack or un-approximation phase in which the guards have to be removed. This process may be costly and could be difficult to implement as well.

Statistical verification. One of our assumptions is that a good portion of constraints from real-world benchmarks are TVPI to begin with. A thorough experimental verification from CLooG benchmarks, as well as, from Omega benchmarks could be done; such a validation could help formalize many folklore hunches, like our ealier statistical study in Section 7.2. Also helpful in this aspect are the statistics referring to (U)TVPI domains in Maydan et al. [MHL91] and the subsequent comments by Pugh [Pug92]. Furthermore, looking at how Omega handles UTVPI domains may be helpful.

Though in this chapter we focussed mainly on TVPI polyhedra, UTVPI specialization would have some additional advantages:

QRW's modulo operations and UTVPI polyhedra. It is known [Bas04b, Bas12] that the QRW algorithm is limited by the costly modular operations it generates. In the previous chapter, in Section 8.8 on page 123, we suggest that obtaining a UTVPI-UA of the per-dependence Farkas polyhedron which has integer homogenizing dimension values would result in the overall Farkas polyhedon to have just integer vertices because of Total-Unimodularity. The same reason applies to removing the modularities associated with QRW algorithm. These can be removed by obtaining UTVPI over-approximations and using the Totally Unimodularity and integer vertices associated with UTVPI sub-polyhedral over-approximations. A similar method has originally been hinted by Bastoul and Griebl [Bas04b, Section 6.2.2, page 134] who suggest restricting the transformation coefficients to $\{0, \pm 1\}$ so that modulos are not generated. Using UTVPI schedules has the advantage of integer-vertices along with the additional advantage of improved guarantee of polynomial time complexity.

Dealing with non-TVPI constraints

The non-TVPI constraints have to be over-approximated on a per-statement basis initially. After the QRW-TVPI code generation, they have to be de-approximated.

Over-approximations. For general convex polyhedra, the methods suggested in this chapter need an over-approximation algorithm. The effectiveness of the TVPI based code generation methods depends on the OA method as domains with non-intersecting original projections may intersect when over-approximated. Hence a costly algorithm which uses the generator representation could be used. Miné [Min06, Section 4.3] adapts the vertex method for finding the UTVPI-OA of a general polyhedron. This could be used because the per-statement domain systems are extremely small. Simon et al. [SK10, Section 3.2.6] propose an iterative algorithm for the TVPI-OA of a general polyhedron using Linear Programming. We have summarized these algorithms briefly in Section 4.3 on page 50. But, this over-approximation should be carefully designed as it could decrease the quality of the generated code by introducing additional guards, i.e., control overhead.

Limitation for un-approximable domains. Similar to the polyhedra in Section 6.7.4 on page 80 for which there exist only trivial under-approximations, there may be real world domains—for examples, ones with non-trivial lineality space—that cannot be over-approximated by a TVPI domain. For these kinds of degenerated cases, the usual QRW method can be followed. Or, techniques which involve splitting of these domains along with an under-approximation method could be used.

De-approximation. Though we have not discussed the implications of the over-approximation process in this chapter, in particular the de-approximation (un-approximation) algorithm, we consider its design and practical implications quite crucial for the success of the schemes suggested here. This phase could be similar to the methods in CLooG or CodeGen of Omega. We however leave its discussion for future work.

Code generation and (U)TVPI as abstract domain

The understanding that operators on (U)TVPI sub-polyhedral domains have excellent complexity measures led to their use in static analysis with wide ranging success. The very same reason led to the proposal of (U)TVPI sub-polyhedral scheduling and (U)TVPI sub-polyhedral code-generation as a means to reduce the complextiy of the most prominent loop parallelization applications of scheduling and code generation. But, sub-polyhedral scheduling, which is the main contribution of this thesis and covered in the earlier chapters, does not use the abstract domain properties of sub-polyhedra at all, instead directly relying on the complexity of under-approximation algorithms, and the complexity of solution of (U)TVPI sub-polyhedral linear programs. In contrast, sub-polyhedral code generation proposed in this chapter, by its reliance of image, projection, intersection, and union operations, uses (U)TVPI sub-polyhedra more as an abstract domain.

Chapter 10

Conclusions and Perspectives

It is well true that automatic loop parallelization is the most powerful optimization designed as part of compiler optimizations. Though it is surprising that it took a couple of decades for main stream compiler community to accept the polyhedral compilation framework, it has now become the acceptable norm for compilation of loop programs. Examples are the implementations of variations of the Forward Communications Only framework of PLuTo implemented in the various polyhedral compilers like IBM-XL, Reservoir Labs R-Stream, GCC-GRAPHITE and LLVM/Polly. A direct consequence of the above success, is that it is likely that the size of the input programs to the polyhedral compiler would increase, and hence it becomes more pertinent for the polyhedral community to solve the scalability challenge.

Summary. In this thesis we contributed to some of the challenges of scalability in polyhedral compilation by introducing algorithmic analysis and complexity based approximation based methods to use (Unit-)Two-Variables-Per-Inequality sub-polyhedra. More particularly, we proposed *sub-polyhedral compilation using (Unit-)Two-Variables-Per-Inequality* as a means of obviating the scalability challenges in polyhedral compilation. The methods in this thesis should only be seen only as the *initial steps* in solving the hard to solve scalability problem.

Contributions

The following are the contributions of this thesis.

Scalability challenges and sub-polyhedral compilation

The first contribution of this thesis is the complexity driven analysis of the different modules of the polyhedral compiler. In particular with the success of various polyhedral compilers one is directly lead to ask the obvious questions: "Will they scale for large inputs? If not, what are the techniques to make them scale?". For answering the above questions, using theoretical and empirical analysis of the various modules of polyhedral compilation, we showed that current methods that use general convex polyhedra as polyhedral abstractions (for dependences, domains and transformations) do not scale. Trying to integrate the polyhedral compiler as a phase in any compiler would mean solving the *scalability challenge*.

The suggestion of *sub-polyhedral compilation* is but a natural progression from the above answers. We suggested that sub-polyhedra could be used in affine scheduling and polyhedral code generation so that practically scalable and asymptotially better algorithms could be designed. The key concepts in our approaches are *algorithmic approaches* and *complexity driven approximations.*

Affine scheduling using (U)TVPI sub-polyhedra

In understanding the various options available options for use in scalability in sub-polyhedra, *(Unit-)Two-variables-Per-Inequality Sub-Polyhedra* or *(U)TVPI sub-polyhedra* come at the forefront. We studied in Chapter 3 that these latter flavors of sub-polyhedra have been used both by the theoretical (algorithmic) community as well as by the static analysis community.

The advantages with these varieties of sub-polyhedra are that they admit *graph theoretic algorithms for solving linear optimization problems* when the input problem formulations are of restricted (U)TVPI form. More particularly, TVPI constraints can be solved using (generalized) min-cost flow algorithms, while UTVPI constraints can be solved using Bellman-Ford and Johnson's algorithms. For the same reasons, the static analysis community has been successfully using (U)TVPI sub-polyhedra as abstract domains for abstract interpretation problems.

In Chapter 4, we introduced the concept of *sub-polyhedral scheduling using Unit-Two-Variables-Per-Inequality polyhedra.* This method solves the scalability problem from within the affine scheduling framework by using the concept of *sub-polyhedral Under-Approximations (UA)* and can be applied in two main ways: UAs of the per-dependence Farkas polyhedra to result in an intersection of these individual approximations, and UA of the overall Farkas polyhedron which can be used on an offline/online basis.

Polynomial time algorithms for (U)TVPI approximation

The above sub-polyhedral affine scheduling methods also do need appproximation algorithms with low complexity. We proposed worst-case polynomial time algorithms to compute *(U)TVPI under-approximations* beginning from a general convex polyhedron.

For doing the above, in Chapter 5 we developed a simple *polarity based framework* that can be used to design approximation algorithms. In Chapter 6, we used the above framework to design simple and *asymptotically worst-case polynomial time algorithms* that take a general convex polyhedron and under-approximate it into (U)TVPI polyhedra. One of the algorithms finds the UA in a heuristic fashion, while the other by framing a higher dimensional linear program. To stay within polynomial time, care was taken to linearize the under-approximation model, avoiding the exponential vertex and ray construction associated with the Chernikova algorithm. We explored different heuristics relying on bounded-size linear programs to trade complexity for feasibility.

In Section 6.7.5, we also suggested a small variation of the above framework to obtain *integer polyhedral approximations of rational polyhedra,* by using ILP formulations of small size. This method brings the concept of *Total-Unimodularity* from the mathematical programming community to be used in polyhedral compilation.

Experimental evaluation of our techniques

In Chapter 7, we showed initial results of the above approximations when integrated into the PLuTo polyhedral compiler. We began with a study of the *polyhedral characteristics* of the benchmarks in PolyBench 2.0.

This was followed by studying the *experimental results of preservation of feasibility* of the under-approximations using various algorithms that we proposed. We showed that the feasibility is preserved for 10 out of 16 number of benchmarks from PolyBench 2.0. We also studied the *scalability experiments*, where we compared the running times of solving the original system using traditional simplex vs. solving the under-approximated system using Bellman-Ford. We showed an *asymptotic improvement in scalability*, answering positively the scalability challenge. We also showed that a preliminary integration of the above UA polyhedra into PLuTo yields code for a few PolyBench 2.0 benchmark cases that does not suffer significant increase in execution time.

Applications to various loop transformation problems

In Chapter 8, we showed an illustration of the power of our approximation scheme by *applying it for important loop transformation* problems. We introduced a *hierarchy of loop transformation problems* regarding the type of transformations they can affect: $1d$ transformations, more general multi-dimensional (affine) transformations which encode multitudes of useful transformations, and specific tailor-made multi-dimensional transformations. We also introduced a gradation of these problems regarding their computational complexity: strongly-polynomial time, weakly polynomial time and hence admitting LP formulations, or NP-Complete and hence admitting ILP or MILP formulations. Each of the above problems can admit *polynomial time under-approximation algorithms*, which can be used for both understanding the problem complexity, as well as for obtaining algorithmic solutions.

Unique to the above under-approximation based solutions is the use of Total Unimodularity result in approximations in scheduling which was introduced in Section 6.7.5. By this method, an integer polyhedral under-approximation is obtained by solving an ILP formulation on a per-dependence basis. When the per-dependence under-approximations are UTVPI approximations, the resultant overall under-approximation—which is the intersection of all these individual under-approximations—will result in a system for which LP formulation will automatically result in an *integer optimal solution* as well, in cases when the under-approximation preserves the feasibility.

Sub-polyhedral code generation

In Chapter 9, we proposed a *sub-polyhedral code generation using (U)TVPI polyhedra*. For doing the above, we began with a complexity oriented theoretical study of the of Quilleré, Rajopadhye and Wilde [QRW00] (QRW) algorithm, which is considered to be the most successesful of the polyhedral code generation schemes because of their implementation in the widely used code generator CLooG [Bas04a]. We identified that the two causes of exponentiality that can cause scalability challenges in the code generation can be reduced to polynomial time using (U)TVPI polyhedra.

Using simple concepts like (U)TVPI sub-polyhedral over-approximations of per-statement domains and use of sub-dimensional projections for scanning, we introduced a scheme to create *specializations of the QRW algorithm.* Using the above scheme, we proposed two new algorithms: a per-dimensional one which does the separation of the domains on a per-dimensional basis; and a planar one which does the separation by using projections on planar polyhedra.

Both of these algorithms can solve the polyhedral code generation in guaranteed worst-case polynomial time.

Perspectives

Though we have raised many questions in the course of this thesis and provided only a few answers, we hope that the methods suggested in this thesis are interesting enough to be starting points for further research. The following are some broad directions in which the methods suggested in this thesis could be extended.

Program theoretic approach

One primary focus of this thesis is on the algorithmic (complexity theoretic) advantages that (U)TVPI sub-polyhedra offers, and designing the ways in which these can exploited in polyhedral compilation. Such a kind of approach could be further strengthened by complementing it with a study of program theoretic (and transformational) approach as well. In this, a study of the kind of programs that are best suited for (U)TVPI sub-polyhedral compilation could be done. Or, a classification of transformations could be made which are easily possible by solving the inexpensive graph theoretic algorithms of min-cost flow or Bellman-Ford algorithms in the dual space. However, this understanding could be quite different from the approach of Balasundaram and Kennedy [BK89], in the sense that the work in the new direction is still on the Farkas polyhedron; and our suggested method still does an under-approximation of this latter, while preserving specific interesting program transformations in the primal.

Examples where such under-approximations could help are the the multi-dimensional shifting by Darte and Huard [DH00a] and the array contraction results by Darte and Huard [DH05]. Though both of these transformations are interesting, the NP-Completeness of the associated problems could hinder their practical application. However, approximations which preserve interesting transformations from these formulations would be of use to the community. Equally important is the case of the fusion heuristics by Pouchet et al. [PBB$^+$11] which could also benefit from (U)TVPI approximations.

Feasibility preservation

Related to the above direction is the problem of preservation of feasibility of intersections of under-approximations. Indeed, the difficult problem in the methods given in Chapter 6 is the lack of a scalable way to reliably preserve the non-emptiness of the under-approximation when the original polyhedron is non-empty. The one-shot linear programming method offers this guarantee, but is clearly not in strongly polynomial time and there could be limitations in applying it over multiple dependences. We know that

this problem of finding a feasible UA by paying a small cost is difficult, because a strongly polynomial method with this guarantee could directly lead to a solution of finding strongly polynomial feasibility test for general polyhedra.

In Chapter 8, we have provided some hints for preserving the feasibility in the presence of multiple-dependences arising from different loop transformation applications; namely by preserving the unit-schedule (canonical schedule) for each dependence and collecting the overall under-approximations, clustering the constraints and dependences so that one-shot method can be applied to the indvidual clusters, and even clustering of input sections of programs. Still, this problem largely remains unsolved and there needs to be further theoretical study as well as practical work in each of these directions.

Such a study is bound to be complicated for an affine formulation like Forward Communications Only. In this latter transformation as formulated by a scheduler like PLuTo, the LP problems are on a level-by-level basis searching for permutable hyperplanes, and with the inherent limitation that there is no guarantee of preservation of feasibility in the original (un-approximated) Farkas constraint system. For such input systems, the study of preservation of feasibility of the under-approximated systems (as in the YY or YN cases), along with the consequent impact on the amount of parallelism lost is indeed bound to be challenging.

Experiments and impact on the transformed code

Another limitation of the methods suggested in this thesis is the fact that we have been successful in generating code only for a few cases from PolyBench as part of experiments. Though we have discussed this issue at length in the conclusions section of Chapter 7, this needs more work in the sense of a complete evaluation of the performance impact of the approximation on all PolyBench benchmarks using an approximated objective function compatible with the algorithm which is used for solving the approximated systems. The latter aspect which is measured in terms of the running times of the output generated code, is the true measure of the loss of performance and hence another answer to the precision vs. cost tradeoff.

Future experiments should also include fast duplicate elimination schemes and their impact on the running times, and a complementary study of the memory complexity of the original as well as the under-approximated systems, known to be overwhelmingly in our favor.

Memory complexity reduction

This latter aspect of memory reduction of the polyhedral compiler is, at least sometimes arguably more important than the time-complexity reduction, which we have primarily focused in this thesis. Though it is well understood that both polyhedral scheduling and polyhedral code-generation are memory intensive operations, little can be done with the current algorithms themselves, because of their inherent limitations.

The polyhedral scheduling inherently reduces to constructing a large (quadratic) size simplex tableau, which can be quite costly in practice. Also, polyhedral code-generation of the QRW algorithm is well known to be highly memory intensive, primarily because of the need for dual representations of polyhedra and the way existing libraries like PolyLib [Wil93] implement these. (CLooG-PolyLib [Bas04a, Bas04b] is known to face these memory scalability problems.) While both these operations can be prohibitively

expensive even for a conventional compiler, they will be more so for a compiler like LLVM/Polly, and steps should be taken to reduce them. In contrast, either of the (U)TVPI sub-polyhedra can be encoded in linear memory and hence are extremely light on the memory requirements—which has been practically observed in the course of experiments in this thesis—and this will be quite useful for any variety of compilation system.

Sub-polyhedral code generation

The particular aspect of raising more questions than answering them is true more so for our work on polyhedral code generation in Chapter 9, where we have relied on the complexities of the (U)TVPI sub-polyhedra to design polynomial time code generation schemes. This result could be improved in several ways.

One important subroutine in the design of sub-polyhedral code-generation schemes is the method to deal with non-(U)TVPI constraints and domains. The method we have focused in the chapter is to over-approximate the per-statement domains by using existing algorithms, followed by a de-approximation phase. Both of these phases need to be designed carefully because of the possible impact they have on the running time, as well as the quality of code-generated. We have suggested some options for both of these phases in the above chapter. Either of these operations could be based on the generator representation because of the small cost of the per-statement domains, and hence designing them should be done with an emphasis more on improving precision in the form of generated code, and not with reducing the running time of the algorithms.

Equally important is the study of the impact of the precision vs. cost tradeoff on practical inputs. Use of (U)TVPI polyhedra is bound to result in loss of precision whch reflects in the form of more control-overhead in the generated code, but it remains to be seen how it effects practically. Trying to reduce the control-overhead using heuristics methods like Omega does for CodeGen [KPR95] is an option, but it is likely that such method would be engineering intensive, and more importantly lose the simplicity of the QRW algorithm.

There is also a need for designing a graph theoretic algorithm to choose the proper plane with an appropriate cost function in case of our planar algorithm QRW-TVPI2. Since the algorithm in effect has to search for a plane (among the possible $d-1$ alternatives) to project the domains such that a proper cost function is minimized, its design could also involve a tunable parameter by which one can control the precison of the generated code.

Further, there is need for representative examples, whether real or artificial, which clearly illustrate the exponentialities of the QRW algorithm as implemented in CLooG-PolyLib [Bas04a, Bas04b]. Beginning with such examples would clearly illustrate the improvements in the running time as well as showing the loss of precision by use of our specializations.

We believe each of these are challenging aspects to be solved in sub-polyhedral code-generaton to achieve asymptotically better code-generation schemes which are practically useful as well.

*****

A Chronological Order

The following is a chronological order of the contributions in this thesis:

- ... 2009-2010: Wide success of PLuTo algorithm and its implementation. Scalability challenges in IBM compiler and Reservoir Labs R-Stream compiler. Integration of a polyhedral compiler in the GCC's GRAPHITE.
- Early and mid 2010: Study of use of (U)TVPI polyhedra by theoretical community and sub-polyhedra by the static analysis community.
- Late 2010: Proposal of sub-polyhedral affine scheduling using under-approximations.
- IMPACT-11 [UC11] acceptance and feedback.
- Early 2011: Scalability complexity study with unrolled examples.
- Mid 2011: Design of (U)TVPI Under-approximation algorithms.
- Late 2011: Implementation in PLuTo polyhedral compiler.
- IMPACT-12 [UC12] acceptance and feedback.
- Early and middle 2012: Implementation, asymptotic improvement by using UTVPI polyhedra. Various applications, Total-Unimodularity result.
- POPL-13 [UC13] acceptance and feedback.
- Late 2012: Proposal of sub-polyhedral code generation.

A Personal Bibliography

International Conferences/Workshops

- (with Albert Cohen): Sub-Polyhedral Scheduling Using (Unit-)Two-Variable-Per-Inequality Polyhedra, published in *40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (ACM-POPL-2013)*, Rome, Italy, January 2013.
- (with Albert Cohen): A Case for Strongly Polynomial Time Sub-Polyhedral Scheduling Using Two-Variable-Per-Inequality Polyhedra, published in *2nd International Workshop on Polyhedral Compilation Techniques (IMPACT-2012), in conjunction with HiPEAC'12,* Paris, France, April 2012.
- (with Albert Cohen): Potential and Challenges of Two-Variable-Per-Inequality Sub-Polyhedral Compilation, published in *1st International Workshop on Polyhedral Compilation Techniques (IMPACT-2011), in conjunction with CGO'11*, Chamonix, France, April 2011.
- (with GRAPHITE team): GRAPHITE Two Years After: First Lessons Learned From Real-World Polyhedral Compilation, published in *2nd International Workshop on GCC Research Opportunities (GROW'10),* Pisa, Italy, January 2010.
- (with L. Renganarayana and S. Rajopadhye): Combined ILP and Register Tiling: Analytical Model and Optimization Framework, published in *The 18th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2005).* Conference held in IBM Research Hawthorne.

Textbook chapters

- (with D. Das and V. Sreedhar): Advanced SSA Construction Algorithms, in Fabrice Rastello (ed.), *SSA-based Compiler Design*, Springer, 2013 (expected).

International Journals

- (with Dibyendu Das, Benoît Dupont de Dinechin): Efficient liveness computation using merge sets and DJ-graphs, published in *ACM Transactions on Architecture and Code Optimization (ACM-TACO-2012), presented in HiPEAC'12,* Paris, France, January 2012.
- (with D. Das): A Practical and Fast Iterative Algorithm for ϕ-function Computation using DJ-graphs, published in *ACM-Transactions on Programming Languages and Systems (ACM-TOPLAS),* 2005.
- (with M. Madhavan, P. Shankar and S. Rai): Extending Graham Glanville Techniques for Optimal Code Generation, published in *ACM-Transactions on Programming Languages and Systems (ACM-TOPLAS),* 2000.

Bibliography

[ACM84] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical algebraic decomposition i: The basic algorithm. *SIAM J. Comput.*, 13(4):865–877, 1984. 86

[AF92] David Avis and Komei Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8(3):295–313, October 1992. 136

[AI91] Corinne Ancourt and François Irigoin. Scanning polyhedra with do loops. In *Proceedings of the third ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '91, pages 39–50, New York, NY, USA, 1991. ACM. 128

[AK87] Randy Allen and Ken Kennedy. Automatic translation of fortran programs to vector form. *ACM Trans. Program. Lang. Syst.*, 9(4):491–542, October 1987. 40, 41

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., NJ, USA, 1993. 25, 79

[AN98] Arne Andersson and Stefan Nilsson. Implementing radixsort. *J. Exp. Algorithmics*, 3, September 1998. 80, 98

[AS80] Bengt Aspvall and Yossi Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM J. Comput.*, 9(4):827–845, 1980. 23, 134

[Bag97] R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy, March 1997. Printed as Report TD-1/97. 25, 33

[Ban92] U. Banerjee. *Loop Transformations for Restructuring Compilers: The Foundations*. Kluwer Academic Publishers, Boston, 1992. 8, 74

[Bas04a] Cédric Bastoul. Code generation in the polyhedral model is easier than you think. In *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, pages 7–16, Juan-les-Pins, France, September 2004. 3, 128, 129, 130, 146, 155, 157, 158

[Bas04b] Cédric Bastoul. *Improving Data Locality in Static Control Programs*. PhD thesis, University Paris 6, Pierre et Marie Curie, France, December 2004. 128, 129, 130, 146, 150, 157, 158

[Bas11] Cédric Bastoul. Code generation. In David A. Padua, editor, *Encyclopedia of Parallel Computing*, pages 310–318. Springer, 2011. 128, 146

[Bas12] Cédric Bastoul. *Contributions to High-Level Program Optimization.* Habilitation Thesis. Paris-Sud University, France, December 2012. 37, 128, 130, 150

[BCC00] Denis Barthou, Albert Cohen, and Jean-Francois Collard. Maximal static expansion. *International Journal of Parallel Programming*, 28(3):213–243, 2000. 15

[BCC+03] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A static analyzer for large safety-critical software. In *PLDI*, pages 196–207. ACM, 2003. 25, 35

[Bel58] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958. 30, 34

[BF98] Pierre Boulet and Paul Feautrier. Scanning polyhedra without do-loops. In *IEEE PACT*, pages 4–11. IEEE Computer Society, 1998. 128

[BGDR10] Uday Bondhugula, Oktay Gunluk, Sanjeeb Dash, and Lakshminarayanan Renganarayanan. A model for fusion and code motion in an automatic parallelizing compiler. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 343–352, New York, NY, USA, 2010. ACM. 124

[BHRS08] Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *PLDI*, pages 101–113, 2008. 3, 7, 9, 58, 97, 101, 109, 121, 122, 124, 125

[BHZ08] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008. 25, 34

[BHZ09] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. Weakly-relational shapes for numeric abstractions: improved algorithms and proofs of correctness. *Formal Methods in System Design*, 35(3):279–323, 2009. 25, 34, 35

[Bix02] Robert E. Bixby. Solving real-world linear programs: A decade and more of progress. *Oper. Res.*, 50(1):3–15, January 2002. 28, 92

[BK89] Vasanth Balasundaram and Ken Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In *PLDI*, pages 41–53, 1989. 35, 38, 39, 42, 44, 55, 58, 156

[BKVH07] S. Boyd, S.J. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. *Optimization and Engineering*, 8(1):67–127, 2007. 86

[BPCB10] Mohamed-Walid Benabderrahmane, Louis-Noël Pouchet, Albert Cohen, and Cédric Bastoul. The polyhedral model is more widely applicable than you think. In *ETAPS CC'10*, LNCS, Cyprus, March 2010. Springer-Verlag. 15, 18, 38

[BQRR98] Stephan Balev, Patrice Quinton, Sanjay V. Rajopadhye, and Tanguy Risset. Linear programming models for scheduling systems of affine recurrence equations - a comparative study. In *SPAA*, pages 250–258, 1998. 49

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Berichte über verteilte messysteme. Cambridge University Press, 2004. 27

[Cal87] C. D. Callahan, II. *A global approach to detection of parallelism*. PhD thesis, Houston, TX, USA, 1987. UMI Order No. GAX87-18697. 42, 46

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976. 31, 32, 33

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY. 19, 31, 46

[CC04] Robert Clarisó and Jordi Cortadella. The octahedron abstract domain. In Giacobazzi [Gia04], pages 312–327. 33

[CCF+09] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Why does Astrée scale up? *FMSD*, 35(3):229–264, Dec 2009. 25, 35

[CDR98] Pierre-Yves Calland, Alain Darte, and Yves Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Trans. Parallel Distrib. Syst.*, 9(1):24–35, January 1998. 101, 103, 106, 121, 122, 123, 124, 125

[CGG+10] Boris V. Cherkassky, Loukas Georgiadis, Andrew V. Goldberg, Robert E. Tarjan, and Renato F. Werneck. Shortest-path feasibility algorithms: An experimental evaluation. *J. Exp. Algorithmics*, 14:7:2.7–7:2.37, January 2010. 34

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY. 31, 33

[Che12] Chun Chen. Polyhedra scanning revisited. In *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*, PLDI '12, pages 499–508, New York, NY, USA, 2012. ACM. 128, 146

[CK88] David Callahan and Ken Kennedy. Analysis of interprocedural side effects in a parallel programming environment. *J. Parallel Distrib. Comput.*, 5(5):517–550, 1988. 42, 46

[CLZ09] Jason Cong, Bin Liu, and Zhiru Zhang. Scheduling with soft constraints. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD '09, pages 47–54, New York, NY, USA, 2009. ACM. 101, 106

[CM94] Edith Cohen and Nimrod Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.*, 23:1313–1350, December 1994. 24, 25, 30, 34, 38

[Coh93] H. Cohen. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer, 1993. 130

[Cor91] Ancourt Corrinne. *Génération de code pour multiprocessors â mémories locales*. PhD thesis, Université Paris VI, France, March 1991. 149

[CPL] IBM ILOG CPLEX: High-performance software for mathematical programming and optimization. See `www.ibm.com/software/integration/optimization/cplex-optimizer/`. 48

[Cre96] Béatrice Creusillet. *Array Region Analyses and Applications*. PhD thesis, École des Mines de Paris (Mines ParisTech), France, December 1996. 39, 45, 46, 55

[CS11] Michael Colón and Sriram Sankaranarayanan. Generalizing the template polyhedral domain. In Gilles Barthe, editor, *ESOP*, volume 6602 of *Lecture Notes in Computer Science*, pages 176–195. Springer, 2011. 56

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. 25, 51, 97, 114

[CZ06] Jason Cong and Zhiru Zhang. An efficient and versatile scheduling algorithm based on sdc formulation. In *Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pages 433–438, New York, NY, USA, 2006. ACM. 101, 106

[Dan63] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963. 27, 35

[dBKS00] M. de Berg, van Krefeld, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000. 24, 134

[DH00a] Alain Darte and Guillaume Huard. Loop shifting for loop compaction. *Int. J. Parallel Program.*, 28(5):499–534, October 2000. 101, 105, 106, 121, 122, 123, 124, 125, 156

[DH00b] Alain Darte and Guillaume Huard. Loop shifting for loop parallelization. Technical Report RR2000-22, LIP, ENS-Lyon, France, May 2000. 114, 115, 116, 118

[DH02] Alain Darte and Guillaume Huard. Complexity of multi-dimensional loop alignment. In *STACS '02*, pages 179–191, London, 2002. Springer-Verlag. 102, 113, 114, 115, 118, 119, 121, 122, 123, 124, 125

[DH05] Alain Darte and Guillaume Huard. New complexity results on array contraction and related problems. *J. VLSI Signal Process. Syst.*, 40(1):35–55, May 2005. 102, 118, 120, 121, 122, 124, 125, 156

[DH06] Christoph Dürr and Mathilde Hurand. Finding total unimodularity in optimization problems solved by linear programs. *CoRR*, abs/cs/0602016, 2006. 106

[Die06] R. Diestel. *Graph Theory*. Graduate Textx In Mathematics. Springer London, Limited, 2006. 114

[DRV00] Alain Darte, Yves Robert, and Frédéric Vivien. *Scheduling and Automatic Parallelization*. Birkhaüser, 2000. 2, 3, 7, 8, 37, 38, 53, 55, 57, 71, 89, 106, 107

[DSV97] Alain Darte, Georges-André Silber, and Frédéric Vivien. Combining retiming and scheduling techniques for loop parallelization and loop tiling. *Parallel Processing Letters*, 7(4):379–392, 1997. 113

[DV95] Alain Darte and Frédéric Vivien. Revisiting the decomposition of karp, miller and winograd. In *ASAP*, pages 13–25. IEEE Computer Society, 1995. 107

[DV97a] Alain Darte and Frédéric Vivien. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. *Int. J. Parallel Program.*, 25:447–496, December 1997. 40, 41, 49, 101, 107, 121, 122, 124, 125

[DV97b] Alain Darte and Frédéric Vivien. Parallelizing nested loops with approximations of distance vectors: A survey. *Parallel Processing Letters*, 7(2):133–144, 1997. 55

[ERW89] Herbert Edelsbrunner, Günter Rote, and Emo Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theor. Comput. Sci.*, 66(2):157–180, 1989. 34

[Fea88] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22(3):243–268, 1988. `http://www.piplib.org/`. 8, 13, 28, 40, 41, 48, 74, 89, 92, 98, 128

[Fea91] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, 1991. 2, 8, 40, 41, 45, 108

[Fea92a] Paul Feautrier. Some efficient solutions to the affine scheduling problem: I. one-dimensional time. *IJPP*, 21:313–348, October 1992. 3, 7, 9, 16, 18, 38, 40, 41, 47, 49, 52, 53, 54, 56, 58, 71, 101, 107, 109, 113, 121, 122, 124, 125

[Fea92b] Paul Feautrier. Some efficient solutions to the affine scheduling problem: Part ii: Multidimensional time. *IJPP*, 21:389–420, December 1992. 3, 9, 18, 101, 107, 109, 110, 113, 121, 122, 124, 125

[Fea06] Paul Feautrier. Scalable and structured scheduling. *International Journal of Parallel Programming*, 34(5):459–487, 2006. 38, 39, 47, 56

[FF62] L R Ford, Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962. 30, 34

[GFG02] Martin Griebl, Paul Feautrier, and Armin Größlinger. Forward communication only placements and their use for parallel program construction. In William Pugh and Chau-Wen Tseng, editors, *LCPC*, volume 2481 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2002. 3, 7, 9, 58, 101, 109, 121, 122, 124, 125

[GFL04] Martin Griebl, Peter Faber, and Christian Lengauer. Space-time mapping and tiling: a helpful combination. *Concurrency and Computation: Practice and Experience*, 16(2-3):221–246, 2004. 3, 7, 9

[GGL12] Tobias Grosser, Armin Größlinger, and Christian Lengauer. Polly - performing polyhedral optimizations on a low-level intermediate representation. *Parallel Processing Letters*, 22(4), 2012. 15

[GH62] Alain Ghouila-Houri. Caractérisation des matrices totalement unimodulaires. *C. R. Acad. Sci, Paris*, 254:1192–1194, 1962. 104

[Gia04] Roberto Giacobazzi, editor. *Static Analysis, 11th International Symposium, SAS 2004, Verona, Italy, August 26-28, 2004, Proceedings*, volume 3148 of *Lecture Notes in Computer Science*. Springer, 2004. 163, 172

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. 27, 31, 130

[GJ00] Ewgenij Gawrilow and Michael Joswig. polymake: a framework for analyzing convex polytopes. In Gil Kalai and Günter M. Ziegler, editors, *Polytopes — Combinatorics and Computation*, pages 43–74. Birkhäuser, 2000. 23, 27

[GL94] Martin Griebl and Christian Lengauer. On the space-time mapping of while-loops. *Parallel Processing Letters*, 4:221–232, 1994. 15

[GLS93] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Algorithms and combinatorics. Springer-Verlag, 1993. 27

[Gol04] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics. Elsevier, 2004. 140

[GR07] Gautam Gupta and Sanjay Rajopadhye. The z-polyhedral model. In *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '07, pages 237–248, New York, NY, USA, 2007. ACM. 40, 42

[Gra72] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972. 24, 134

[Gri96] Martin Griebl. *The Mechanical Parallelization of Loop Nests Containing* while *Loops*. PhD thesis, University of Passau, 1996. also available as technical report MIP-9701. 15

[Grö09] Armin Größlinger. *The Challenges of Non-linear Parameters and Variables in Automatic Loop Parallelisation*. PhD thesis, Universität Passau, Innstrasse 29, 94032 Passau, 2009. 86

[Hač79] L. G. Hačijan. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244(5):1093–1096, 1979. 27

[HK91] P. Havlak and K. Kennedy. An implementation of interprocedural bounded regular section analysis. *Parallel and Distributed Systems, IEEE Transactions on*, 2(3):350–360, 1991. 46

[HK09] Jacob M. Howe and Andy King. Logahedra: A new weakly relational domain. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis*, ATVA '09, pages 306–320, Berlin, Heidelberg, 2009. Springer-Verlag. 33, 56

[HM94] Claire Hanen and Alix Munier. Cyclic scheduling on parallel processors: An overview. In Philippe Chrétienne, Edward G. Coffman, Jan Karel Lenstra, and Zhen Liu, editors, *Scheduling theory and its applications*. J. Wiley and sons, 1994. 103, 106

[HM01] C Hanen and A Munier. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. *Discrete Applied Mathematics*, 108(3):239–257, 2001. 106

[HMG06] N. Halbwachs, D. Merchat, and L. Gonnord. Some ways to reduce the space dimension in polyhedra. *FMSD*, pages 79–95, Jul 06. 32, 35

[HMG06] N. Halbwachs, D. Merchat, and L. Gonnord. Some ways to reduce the space dimension in polyhedra computations. *Form. Methods Syst. Des.*, 29:79–95, July 2006. 32, 149

[HMNT93] Dorit S. Hochbaum, Nimrod Megiddo, Joseph Naor, and Arie Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Math. Program.*, 62:69–83, 1993. 23, 34, 125

[HN94] Dorit S. Hochbaum and Joseph Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, 1994. 23, 24, 25, 30, 31, 34, 38, 52, 80, 84, 134, 135

[Hoc98] Dorit S. Hochbaum. Instant recognition of half integrality and 2-approximations. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '98, pages 99–110, London, UK, UK, 1998. Springer-Verlag. 34

[Hoc04] Dorit S. Hochbaum. Monotonizing linear programs with up to two nonzeroes per column. *Oper. Res. Lett.*, 32(1):49–58, 2004. 25, 34, 125

[IT88] F. Irigoin and R. Triolet. Supernode partitioning. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '88, pages 319–329, New York, NY, USA, 1988. ACM. 40, 41, 43

[JM09] Bertrand Jeannet and Antoine Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV*, pages 661–667, 2009. 25, 34, 36, 43, 146

[JMSY94] Joxan Jaffar, Michael J. Maher, Peter J. Stuckey, and Roland H. C. Yap. Beyond finite domains. In Alan Borning, editor, *PPCP*, volume 874 of *Lecture Notes in Computer Science*, pages 86–94. Springer, 1994. 35

[JZPC08] Wei Jiang, Zhiru Zhang, Miodrag Potkonjak, and Jason Cong. Scheduling with integer time budgeting for low-power optimization. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, ASP-DAC '08, pages 22–27, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press. 101, 106

[Kah74] Gilles Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974. 47

[KAI+09] Aditya Kanade, Rajeev Alur, Franjo Ivancic, S. Ramesh, Sriram Sankaranarayanan, and K. C. Shashidhar. Generating and analyzing symbolic traces of simulink/stateflow models. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 430–445. Springer, 2009. 56, 87

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984. 27

[Kim10] Edward D. Kim. *Geometric Combinatorics of Transportation Polytopes and the Behavior of the Simplex Method*. PhD thesis, Davis, CA, USA, 2010. `http://arxiv.org/abs/1006.2416`. 35

[KM72] V. Klee and G. J. Minty. How Good is the Simplex Algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press Inc., New York, 1972. 28

[KM94] Ken Kennedy and Kathryn S. McKinley. Maximizing loop parallelism and improving data locality via loop fusion and distribution. In *Proceedings of the 6th International Workshop on Languages and Compilers for Parallel Computing*, pages 301–320, London, UK, UK, 1994. Springer-Verlag. 120

[KMW67] Richard M. Karp, Raymond E. Miller, and Shmuel Winograd. The organization of computations for uniform recurrence equations. *J. ACM*, 14(3):563–590, July 1967. 2

[KPR95] Wayne Kelly, William Pugh, and Evan Rosser. Code generation for multiple mappings. In *Frontiers '95: The 5th Symposium on the Frontiers of Massively Parallel Computation*, pages 332–341, McLean, VA, USA, February 1995. 128, 146, 158

[KPRS96] Wayne Kelly, William Pugh, Evan Rosser, and Tatiana Shpeisman. Transitive closure of infinite graphs and its applications. *International Journal of Parallel Programming*, 24(6):579–598, 1996. 56

[KS10] Edward Kim and Francisco Santos. An update on the hirsch conjecture. *Jahresbericht der deutschen Mathematiker-Vereinigung*, 112:73–98, 2010. 10.1365/s13291-010-0001-8. 30, 35

[Lag85] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM J. Comput.*, 14:196, 1985. 31, 81, 84, 117

[Lam74] Leslie Lamport. The parallel execution of do loops. *Commun. ACM*, 17(2):83–93, February 1974. 41

[Les96] Arnauld Leservot. *Analyses interprocédurales du flot des données*. PhD thesis, Université Paris VI, France, March 1996. 130

[LF10] Francesco Logozzo and Manuel Fähndrich. Pentagons: A weakly relational abstract domain for the efficient validation of array accesses. *Sci. Comput. Program.*, 75(9):796–807, 2010. 33

[LL97] Amy W. Lim and Monica S. Lam. Maximizing parallelism and minimizing synchronization with affine transforms. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '97, pages 201–214, New York, NY, USA, 1997. ACM. 58, 110, 121

[LL09] Vincent Laviron and Francesco Logozzo. Subpolyhedra: A (more) scalable approach to infer linear inequalities. In Neil D. Jones and Markus Müller-Olm, editors, *VMCAI*, volume 5403 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2009. 32, 33

[LM05] Shuvendu K. Lahiri and Madanlal Musuvathi. An efficient decision procedure for UTVPI constraints. In Bernhard Gramlich, editor, *FroCos*, volume 3717 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2005. 34

[LS91] Charles Leiserson and James Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991. 10.1007/BF01759032. 103, 113, 114, 115

[Meg83] Nimrod Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12(2):347–353, 1983. 23, 34

[Meg84] Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, January 1984. 27

[Mei96] Wolfgang Meisl. Practical methods for scheduling and allocation in the polytope model. Master's thesis, Department of Informatics and Mathematics, University of Passau, Germany, September 1996. http://www.infosun.fim.uni-passau.de/cl/loopo/doc/. 49, 108

[MHL91] Dror E. Maydan, John L. Hennessy, and Monica S. Lam. Efficient and exact data dependence analysis. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, PLDI '91, pages 1–14, New York, NY, USA, 1991. ACM. 55, 96, 150

[Min01a] Antoine Miné. A new numerical abstract domain based on difference-bound matrices. In Olivier Danvy and Andrzej Filinski, editors, *PADO*, volume 2053 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2001. 25, 31, 33

[Min01b] Antoine Miné. The octagon abstract domain. In *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, WCRE '01, pages 310–, Washington, DC, USA, 2001. IEEE Computer Society. 33

[Min04] A. Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Polytechnique, Palaiseau, France, December 2004. http://www.di.ens.fr/~mine/these/these-color.pdf. 25, 34, 35, 55

[Min06] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006. 25, 32, 33, 34, 35, 43, 46, 50, 56, 79, 137, 150

[MQRS90] C. Mauras, P. Quinton, S. Rajopadhye, and Y. Saouter. Scheduling affine parameterized recurrences by means of variable dependent timing functions. In *Proceedings of the International Conference on Application Specific Array Processors, 1990*, pages 100 –110, sep 1990. 47

[MS97] Nimrod Megiddo and Vivek Sarkar. Optimal weighted loop fusion for parallel programs. In *SPAA*, pages 282–291, 1997. 102, 120

[PB+] Louis-Noël Pouchet, Uday Bondhugula, et al. The polybench benchmarks. http://sourceforge.net/projects/polybench/. 11, 14, 47, 89

[PBB+11] Louis-Noël Pouchet, Uday Bondhugula, Cédric Bastoul, Albert Cohen, J. Ramanujam, P. Sadayappan, and Nicolas Vasilache. Loop transformations: convexity, pruning and optimization. In *POPL*, pages 549–562, 2011. 7, 52, 124, 156

[Pen55] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(03):406–413, 1955. 97

[PIP] PIPS4U. Pips: Automatic parallelizer and code transformation framework. http://www.pips4u.org. 14, 46, 55, 130

[Pou] Louis-Noël Pouchet. The Fourier-Motzkin Library. http://sourceforge.net/projects/fmlib/. 49, 89, 128

[PPL] PPL. The parma polyhedral library. http://bugseng.com/products/ppl/. 25, 34, 36, 146

[Pra77] V. R. Pratt. Two easy theories whose combination is hard. Technical report, Massachusetts Institute of Technology, Cambridge, Mass, 1977. http://boole.stanford.edu/pub/sefnp.pdf. 25, 34

[Pug91] William Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, Supercomputing '91, pages 4–13, New York, NY, USA, 1991. ACM. 8, 42, 46, 128

[Pug92] William Pugh. A practical algorithm for exact array dependence analysis. *Commun. ACM*, 35(8):102–114, August 1992. 42, 46, 49, 55, 96, 130, 134, 147, 149, 150

[QD89] Patrice Quinton and Vincent Van Dongen. The mapping of linear recurrence equations on regular arrays. *VLSI Signal Processing*, 1(2):95–113, 1989. 49

[QR02] Patrice Quinton and Tanguy Risset. Structured scheduling of recurrence equations: Theory and practice. In Ed F. Deprettere, Jürgen Teich, and Stamatis Vassiliadis, editors, *Embedded Processor Design Challenges*, volume 2268 of *Lecture Notes in Computer Science*, pages 112–134. Springer, 2002. 47

[QRW00] Fabien Quilleré, Sanjay Rajopadhye, and Doran Wilde. Generation of efficient nested loops from polyhedra. *International Journal of Parallel Programming*, 28:469–498, 2000. 10.1023/A:1007554627716. 3, 127, 128, 129, 137, 146, 155

[R] R. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. 13, 15, 94

[Rag76] M. Raghavachari. A constructive method to recognize the total unimodularity of a matrix. *Mathematical Methods of Operations Research*, 20:59–61, 1976. 104

[RPF86] Sanjay V. Rajopadhye, S. Purushothaman, and Richard Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In Kesav V. Nori, editor, *FSTTCS*, volume 241 of *Lecture Notes in Computer Science*, pages 488–503. Springer, 1986. 49

[RR08] Lakshminarayanan Renganarayana and Sanjay Rajopadhye. Positivity, posynomials and tile size selection. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 55:1–55:12, Piscataway, NJ, USA, 2008. IEEE Press. 86

[RTL76] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. 141

[San10] Francisco Santos. A counterexample to the hirsch conjecture. *CoRR*, abs/1006.2814, 2010. http://arxiv.org/abs/1006.2814. 29, 35

[Sch86] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986. 8, 9, 27, 35, 59, 60, 61, 81, 82, 85, 104, 130

[Sho81] Robert Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28:769–779, October 1981. 22, 55, 65, 134

[SK04] Axel Simon and Andy King. Convex hull of planar h-polyhedra. *Int. J. Comput. Math.*, 81(3):259–271, 2004. 24, 134

[SK10] Axel Simon and Andy King. The two variable per inequality abstract domain. *Higher Order Symbol. Comput.*, 23(1):87–143, March 2010. 35, 56, 137, 150

[SKH02] Axel Simon, Andy King, and Jacob M. Howe. Two variables per linear inequality as an abstract domain. In Michael Leuschel, editor, *LOPSTR*, volume 2664 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2002. 33, 36, 146

[SKH10] A. Simon, A. King, and J. Howe. The Two Variable Per Inequality Abstract Domain. *Higher Order and Symbolic Computation*, 23(1):87–143, 2010. 24, 46, 50, 51

[SKS00] Ran Shaham, Elliot K. Kolodner, and Shmuel Sagiv. Automatic removal of array memory leaks in java. In David A. Watt, editor, *CC*, volume 1781 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2000. 25, 33

[SLP96] Edwin Hsing-Mean Sha, Chenhua Lang, and Nelson L. Passos. Polynomial-time nested loop fusion with full parallelism. In *ICPP, Vol. 3*, pages 9–16, 1996. 117

[Sma98] Steve Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20:7–15, 1998. 10.1007/BF03025291. 27

[SQ93] Yannick Saouter and Patrice Quinton. Computability of recurrence equations. *Theoretical Computer Science*, 116(2):317 – 337, 1993. 8

[SS10] Andreas Schutt and Peter J. Stuckey. Incremental satisfiability and implication for utvpi constraints. *INFORMS J. on Computing*, 22(4):514–527, October 2010. 34

[SSM04] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constraint-based linear-relations analysis. In Giacobazzi [Gia04], pages 53–68. 33, 56

[SSM05] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In Radhia Cousot, editor, *VMCAI*, volume 3385 of *Lecture Notes in Computer Science*, pages 25–41. Springer, 2005. 33, 56

[ST04] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51:385–463, May 2004. 28

[SXWL01] Yonghong Song, Rong Xu, Cheng Wang, and Zhiyuan Li. Data locality enhancement by memory reduction. In *Proceedings of the 15th international conference on Supercomputing*, ICS '01, pages 50–64, New York, NY, USA, 2001. ACM. 120

[Tar51] A. Tarski. *A decision method for elementary algebra and geometry.* Univ. of California Press, Berkeley, 2nd edition, 1951. 67, 86

[TCE+10] K. Trifunovic, A. Cohen, D. Edelsohn, F. Li, T. Grosser, H. Jagasia, R. Ladelsky, S. Pop, J. Sjödin, and R. Upadrasta. Graphite two years after: First lessons learned from real-world polyhedral compilation. In *GCC Research Opportunities Workshop (GROW'10)*, Pisa, Italy, January 2010. 15, 18, 36

[Tod02] Michael J. Todd. The many facets of linear programming. *Mathematical Programming*, 91:417–436, 2002. 28, 29, 92

[Tri84] Rémi Triolet. *Contribution á la parallélisation automatique de programmes Fortran comportant des apples de procedure.* PhD thesis, University Paris 6, Pierre et Marie Curie, France, 1984. 43, 46, 55

[Tu95] Peng Tu. *Automatic Array Privatization and Demand-Driven Symbolic Analysis.* PhD thesis, Univ. of Illinois at Urbana-Champaign, Center for Supercomputing Res. and Dev., May 1995. 46

[TVA07] William Thies, Frédéric Vivien, and Saman Amarasinghe. A step towards unifying schedule and storage optimization. *ACM Trans. Program. Lang. Syst.*, 29, October 2007. 121

[UC11] Ramakrishna Upadrasta and Albert Cohen. Potential and Challenges of Two-Variable-Per-Inequality Sub-Polyhedral Compilation. In *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11), in conjunction with CGO'11*, Chamonix, France, April 2011. 6, 19, 35, 36, 57, 68, 89, 159

[UC12] Ramakrishna Upadrasta and Albert Cohen. A Case for Strongly Polynomial Time Sub-Polyhedral Scheduling Using Two-Variable-Per-Inequality Polyhedra. In *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12), in conjunction with HiPEAC'12*, Paris, France, January 2012. 6, 19, 68, 159

[UC13] Ramakrishna Upadrasta and Albert Cohen. Sub-Polyhedral Scheduling Using (Unit-)Two-Variable-Per-Inequality Polyhedra. In *40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2013) (Accepted for publication)*, Rome, Italy, January 2013. 6, 19, 124, 159

[Vas07] Nicolas Vasilache. *Scalable Program Optimization Techniques In The Polyhedral Model.* PhD thesis, Paris-Sud 11 University, September 2007. 7, 9

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001. 117

[Ver92] H. Le Verge. A Note on Chernikova's Algorithm. Technical Report 635, IRISA, Rennes, France, 1992. 8, 130, 134, 136

[Ver10] Sven Verdoolaege. *isl*: An integer set library for the polyhedral model. In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *ICMS*, volume 6327 of *Lecture Notes in Computer Science*, pages 299–302. Springer, 2010. 130

[Viv03] Frédéric Vivien. On the optimality of feautrier's scheduling algorithm. *Concurrency and Computation*, 15(11-12):1047–1068, 2003. 107

[VQ95] H. Le Verge and P. Quinton. Recurrences on lattice polyhedra and their applications to the synthesis of systolic arrays. Technical report, IRISA, 1995. 40, 42

[VW04] Frédéric Vivien and Nicolas Wicker. Minimal enclosing parallelepiped in 3d. *Comput. Geom. Theory Appl.*, 29:177–190, November 2004. 56

[Way99] Kevin D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. In *STOC '99*, pages 11–18, NY, USA, 1999. ACM. 23, 30, 34, 38, 53

[Wil93] D. Wilde. A library for doing polyhedral operations. Technical Report 785, IRISA, Rennes, France, 1993. 8, 89, 129, 130, 149, 157

[WL91a] M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. Parallel Distrib. Syst.*, 2(4):452–471, October 1991. 40, 41, 58

[WL91b] Michael E. Wolf and Monica S. Lam. A data locality optimizing algorithm. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, PLDI '91, pages 30–44, New York, NY, USA, 1991. ACM. 58

[Wol89] M.J. Wolfe. *Optimizing supercompilers for supercomputers*. Research monographs in parallel and distributed computing. Pitman, 1989. 2, 40, 41

[Won95] David Wonnacott. *Constraint-Based Array Dependence Analysis*. PhD thesis, University of Maryland, College Park, 1995. 46

[YAI95] Yi-Qing Yang, Corinne Ancourt, and François Irigoin. Minimal data dependence abstractions for loop transformations: Extended version. *International Journal of Parallel Programming*, 23(4):359–388, 1995. 38, 39, 42, 43, 55, 57, 58

[ZC90] H. Zima and B. Chapman. *Supercompilers for parallel and vector computers*. ACM Press frontier series. ACM Press, 1990. 8

[Zie06] G.M. Ziegler. *Lectures on polytopes*. Graduate texts in mathematics. Springer Science, 2006. 8, 27, 28, 29, 35, 59

List of Figures

List of Tables

Index