UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

Ph.D. Thesis

Modélisation, simulation de différents types d'architectures de noeuds de calcul basés sur l'architecture ARM et optimisés pour le calcul haute-performance

Design Space Exploration of 64-bit ARM Compute Nodes for Highly Energy Efficient Exascale

Joël WANZA WELOLI

LEAT/Bull

A dissertation submitted to the Ecole Doctorale Sciences et Technologies de L'Information et de la Communication (EDSTIC) of the Université Côte d'Azur for the degree of Doctor of Electronic

Directed by: Cécile Belleudy and Sébastien Bilavarn

Thesis defense: June 24th 2019

Members of the Thesis jury:

- Mr. Bertrand Granado, Professor, Sorbonne Université, Examiner
- Mr. Frédéric Petrot, Professor, Université Grenoble Alpes, Examiner
- Mr. Dominique Lavenier, CNRS Research Director, IRISA
- Mr. Elyes Zekri , Dr. Eng., Bull/Atos
- Mr. François Verdier, Professor, Université Côte d'Azur
- Mr. Gabor Dozsa, Dr. Eng., Arm
- Mrs. Cécile Belleudy, Associate Professor, Université Côte d'Azur, Thesis director
- Mr. Sébastien Bilavarn, Associate Professor, Université Côte d'Azur, Thesis co-supervisor

MONT BLANC

anrt association nationale recherche technologie

Bull atos technologies

Design Space Exploration of 64-bit ARM Compute Nodes for Highly Energy Efficient Exascale



Joël Wanza Weloli
Department of Electronic
University of Côte d'Azur

This dissertation is submitted for the degree of
Doctor of Electronic and Computer Architecture

Laboratoire d'Electronique,
Antennes et Télécommunications

2018-2019

I would like to dedicate this thesis to my loving parents, especially to my mother Vicky, the first teacher I ever had.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has less than 150 figures.

Joël Wanza Weloli
2018-2019

Acknowledgements

Foremost, I would like to express my acknowledgments to my Ph.D advisors Cecile Belleudy and Sébastien Bilavarn for their support and help during my Ph.D study and research. Thank you Cecile for being the first one to believe in my Ph.D project. My sincere gratitude to Sébastien for his patience and helpful guidance in all the time for research and writing of this thesis.

My sincere thanks also go to Sylvie Lesmanne who hired me at Bull for my Ph.D student position in the hardware architecture team. Also, to Said Derradji for being a surefire support within Bull and a continuous mentoring on diverse exciting European projects. I'm very grateful to the different organizations who funded the work of this thesis: Bull, ANRT, European commission via Mont-Blanc 2 and 3 projects, but also the LEAT laboratory for hosting this collaboration.

Last but not the least, I would like to thank my family, friends and close colleagues who supported me throughout during these years. Thank you all for always being there. You all have been a real human inspiration to me. A special dedication to Benoît Welterlen (if you thought I was about to forget you :-) Thank you for having given to me my chance as an intern at Bull.

And the very finally, Thanks a lot to all the jury members.

Abstract

English version:

Title: *Design Space Exploration of 64-bit ARM Compute Nodes for Highly Energy Efficient Exascale.*

Abstract: This work is part of a family of European projects called Mont-Blanc whose objective is to develop the next generation of Exascale systems. It addresses specifically the issue of energy efficiency, at micro-architectural level first by considering the use of 64-bit Armv8-A based compute nodes and an associated relevant SoC topology, and examine also the runtime aspects with notably the study of power management strategies that can be better suited to the constraints of HPC highly parallel processing. A design space exploration methodology capable of supporting the simulation of large manycore computing clusters is developped and lead to propose, design and evaluate multi-SoC and their associated SoC Coherent Interconnect models (SCI). This approach is then used to define a pre-exascale achitecture allowing to globally reduce the complexity and cost of chip developments without sacrifying performances. The resulting partitioning scheme introduces interesting perspectives at technology level such as the integration of more compute nodes directly on an interposer based System-in-Package (SiP), possibly based on 3D Through Silicon Vias (TSVs) using High Memory Bandwidth (HBM). Energy efficiency is addressed more directly in second instance by studying current power management policies and proposing two strategies to help reducing power while preserving performances. The first one exploits finer application execution knowledge to adjust the frequency of extensive parallel threads and better balance their execution time. The second strategy reduces core frequencies at synchronisation points of jobs to avoid running the cores at full speed while it is not necessary. Experiment results with these strategies, both in simulation and real hardware, show the possibilities offered par this approach to address the strong requirements of Exascale platforms.

Version française:

Titre: *Modélisation, simulation de différents types d'architectures de nœuds de calcul basés sur l'architecture ARM et optimisés pour le calcul haute-performance.*

Résumé: Ce travail s'inscrit dans le cadre de la famille de projets Européens Mont-Blanc dont l'objectif est de développer la prochaine génération de systèmes Exascale. Il s'intéresse particulièrement à la question de l'efficacité énergétique, d'abord au niveau micro-architectural en considérant l'utilisation de nœuds de calcul basés sur l'Armv8-A 64-bit associée à une topologie SoC pertinente, puis en examinant les aspects exécutifs notamment par une étude de stratégies de gestion énergétique (power management) mieux adaptées à des contraintes de traitement massivement parallèle. Une méthodologie d'exploration architecturale capable de supporter la simulation de larges clusters de calcul parallèle est définie et exploitée pour proposer, développer et évaluer des modèles multi-SoC et de réseaux de communication associés (SoC Coherent Interconnect, SCI). Cette démarche est ensuite poursuivie pour définir une architecture Exacale permettant de réduire globalement la complexité et les coûts de développement en dégradant le moins possible les performances. Le partitionnement de la puce permet ainsi des possibilités intéressantes au niveau technologique telles que l'intégration de nœuds supplémentaires basée sur des technologies System-in-Package (interposer), ou 3D Through Silicon Vias (TSVs) et High Memory Bandwidth (HBM). En second lieu, les aspects énergétiques sont abordés plus directement par l'étude de politiques de gestion énergétique existantes et en proposant deux stratégies pour permettre réduire la consommation en préservant les performance. La première exploite une perception applicative plus fine pour ajuster la fréquence de nombreuses tâches parallèles et mieux équilibrer leurs temps d'exécution. La seconde stratégie réduit la fréquence des cœurs aux points de synchronisation des tâches pour limiter les fonctionnements inutiles à pleine puissance. Les résultats d'expérimentation obtenus avec ces stratégies, à la fois en simulation et sur plateforme réelle, montrent les possibilités offertes par cette approche pour répondre aux fortes contraintes des plateformes pre-exascale sur le plan énergétique.

Contents

Contents vi

List of Figures x

List of Tables xiii

List of terms xv

I BACKGROUND 1

1 Introduction 2

1.1 Objectives and Contexts . . . 3

1.2 Thesis contributions . . . 3

1.2.1 Main contributions . . . 3

1.2.2 Published papers . . . 4

1.3 Thesis outlines . . . 5

2 State-of-the-Art 6

2.1 HPC background . . . 6

2.1.1 History . . . 6

2.1.2 Exascale Challenges . . . 7

2.1.3 Applications . . . 12

2.2 HPC Compute node architectures . . . 15

2.2.1 Introduction . . . 15

2.2.2 Intel Xeon E series architecture: Skylake (Haswell) . . . 17

2.2.3 Intel Xeon Phi Coprocessor architecture: Knights Landing . . . 20

2.2.4 Emerging compute node architectures: SunWay . . . 22

2.2.5 Emerging compute node architectures: Opteron . . . 24

2.2.6 Arm vs Intel Compute node architecture . . . 26
2.2.7 The first Arm-based HPC Cluster: Tibidabo . . . 27
2.3 Aarch64 architecture and ARMv8 processors . . . 29
2.3.1 ARMv8 or AArch64 ? . . . 29
2.3.2 ARMv8-A processors : Cortex-A75 . . . 30
2.3.3 SVE: Scalable Vector Extension . . . 32
2.4 Research projects . . . 33
2.4.1 Design space exploration methodologies . . . 33
2.4.2 TSAR Architecture example . . . 34
2.4.3 Global projects overview . . . 36
2.4.4 Asian projects . . . 36
2.4.5 American project : Exascale Computing Project (ECP) . . . 38
2.4.6 European Projects . . . 39
2.5 Conclusion . . . 40

II EXPERIMENTATION AND ANALYSIS 42

3 Modeling and exploration methodology 43
3.1 Introduction . . . 43
3.2 Virtual prototyping and system components . . . 44
3.2.1 Simulation tools . . . 44
3.2.1.1 Vista . . . 44
3.2.1.2 SoC Designer . . . 45
3.2.1.3 GEM5 . . . 48
3.2.1.4 System Generator Canvas . . . 49
3.2.1.5 Platform Architect . . . 50
3.2.2 Hardware platforms . . . 54
3.2.3 programming support for HPC . . . 57
3.2.3.1 Libraries . . . 57
3.2.3.2 Programming models . . . 57
3.3 Exploration methodology . . . 59
3.3.1 Definition and metrics . . . 59
3.3.2 Extended VPU platform . . . 60
3.3.3 Correlation study . . . 62
3.3.3.1 Evaluation of virtual platforms . . . 62
3.3.3.2 Processing efficiency . . . 62

3.3.3.3 Memory and cache consistency . . . 64
3.3.3.4 Scalability . . . 64
3.4 Conclusion . . . 67

4 Architectural exploration 68
4.1 Memory coherency and SoC partitioning . . . 68
4.2 Cache coherence protocols . . . 69
4.2.1 Overview . . . 70
4.2.2 Snoop Transaction types . . . 70
4.2.3 Directory based filtering . . . 71
4.2.4 ARM Coherence Protocols . . . 72
4.3 SoC Coherent Interconnect . . . 73
4.3.1 Description . . . 73
4.3.2 SCI Architecture . . . 73
4.3.3 Cache model . . . 75
4.3.3.1 Cache controller . . . 75
4.3.3.2 Cache Snoop controller . . . 76
4.3.4 SCI Snoop Filter model . . . 76
4.3.5 SoC Partitionning . . . 78
4.3.5.1 Partitioning topology overview . . . 78
4.3.5.2 Multi-SoC scenarios . . . 79
4.3.5.3 Coherent Proxy extensions . . . 80
4.4 Simulations . . . 83
4.4.1 Directory-based snoop filtering benefits . . . 83
4.4.2 Partitioning analysis . . . 85
4.4.3 Parallel programming efficiency . . . 86
4.5 Conclusion . . . 90

5 Power management 91
5.1 HPC energy efficiency constraints . . . 91
5.1.1 Existing power strategies for HPC . . . 91
5.1.2 OS based strategies . . . 93
5.1.3 Energy efficiency improvement . . . 94
5.2 Evaluation of OS power strategies . . . 95
5.2.1 Simulation framework . . . 95
5.2.1.1 GEM5 . . . 95
5.2.1.2 Benchmarks . . . 95

5.2.1.3 Energy efficiency evaluation 96
5.2.2 Simulation results 96
5.3 Power strategies for HPC 100
5.3.1 Soft big.LITTLE strategy simulations with GEM5 100
5.3.2 Blocking point strategy simulations with GEM5 103
5.4 Measurements with Cavium ThunderX2 107
5.4.1 Platform description 107
5.4.2 Soft big.LITTLE strategy execution on ThunderX2 108
5.4.3 Blocking point strategy execution on ThunderX2 111
5.5 Conclusion 113

III CONCLUSIONS 114

6 Conclusions and Perspectives 115

Bibliography 117

A Gem5 Related work 126

B Gem5 full system stack effort 127

List of Figures

2.1 HPC systems performance projections . . . 7
2.2 Petersen Graph (Moore Graph illustration) . . . 9
2.3 Interconnect Familly in the Top500 . . . 10
2.4 Non-exhaustive HPC applications . . . 12
2.5 Imbrication of annotations in a compute node . . . 15
2.6 x86 architectures hegemony in the HPC Top500 [June 2017] . . . 17
2.7 Intel processors hegemony history in the HPC Top500 . . . 17
2.8 Haswell EP Architecture block diagram . . . 18
2.9 Skylake Architecture block diagram . . . 19
2.10 Haswell based quad-socket platform example . . . 20
2.11 Knights Landing Architecture block diagram overview . . . 21
2.12 Knights Landing (KNL) based Chips . . . 22
2.13 Purley Platform: bi-socket Storm Lake Integrated with Skylake . . . 22
2.14 Sunway SW26010 compute node architecture diagram . . . 24
2.15 AMD Opteron . . . 25
2.16 Internal view . . . 26
2.17 Arm vs INTEL Design flexibility . . . 27
2.18 Tibidabo Arm-based Cluster . . . 28
2.19 Arm architecture generations . . . 29
2.20 The Arm Cortex-A75 processor improvements . . . 30
2.21 Arm DynamIQ concept overview . . . 31
2.22 TSAR Architecture . . . 35
2.23 Post-K Computer Hardware . . . 37
2.24 The US Exascale Computing Project Roadmap . . . 38
2.25 ECP Timeline . . . 39

3.1 Modeling levels . . . 44
3.2 VISTA modelled platform and output view . . . 46

3.3 Reference platform for SoC Designer evaluation . . . 47
3.4 Gem5 Medium scale platform . . . 49
3.5 48 AFM Cores block diagram . . . 51
3.6 Platform example based on Cortex-A57 Fast Model . . . 52
3.7 Equivalent VPU based platform . . . 52
3.8 Dhrystone task graph . . . 53
3.10 Task graphs results With vs Without L3 cache . . . 54
3.9 Simulation outputs: Fast models vs. VPU (Dhrystone benchmark) . . . 54
3.11 Juno board diagram . . . 55
3.12 APM first ARMv8 processor : Xgene1 Storm . . . 56
3.13 AMD Seattle's architecture : Floorplan and block diagram . . . 56
3.14 VPU vs Traffic generator . . . 60
3.15 Global Modeling Methodology . . . 61
3.16 Extensible VPU platform (Dhrystone benchmark) . . . 61
3.17 Principle of correlation study . . . 62
3.18 SGEMM GFlops Scalability . . . 63
3.19 SGEMM Performance efficiency correlation . . . 63
3.20 Arm Juno vs. AFM memory level analysis . . . 64
3.21 From 1 to 48 threads scalability . . . 65

4.1 SoC partitioning scenarios (Mont blanc project) . . . 68
4.2 on-Chip Coherent Interconnect . . . 74
4.3 2 SoC : simple Chip-toChip topology . . . 78
4.4 Four small SoC : The on-Chip Interconnect partitioning topology alternatives . . . 78
4.5 Large scale ARM based VPU : partitioning scenarios . . . 80
4.6 2 ×64 and 4 ×32 SoC partitioning with coherent proxy ports . . . 81
4.7 Coherent proxy entensions . . . 82
4.8 Number of transactions for different SoC and directory configurations . . . 84
4.9 DGEMM cache statistics . . . 84
4.10 Benchmark performance for different SoC and directory configurations . . . 85
4.11 Impact of programming models on throughput (blackscholes) . . . 87
4.12 Impact of programming models on performance (blackscholes, 1×128 cores) 88
4.13 Impact of programming model on L2 cache misses (blackscholes, 1×128 cores) . . . 88
4.14 Impact of programming models on the number of snooping transactions and overhead (blackscholes, 1×128 cores) . . . 89

4.15 Snooping traffic statistics (blackscholes, 1 ×128 cores) 89

5.1 Frequency and Power on Cavium's ThunderX2 processor 97
5.2 Efficicency evaluation of Linux governors (config: Nparallel_256x256_matrix_products) 99
5.3 2parallel_256x256_matrix_products and 2parallel_128x128_matrix_products 102
5.4 Soft big.LITTLE strategy scalability 103
5.5 Before and After blocking point strategy: 4 clock domains 105
5.6 Scaling up Before and After blocking point strategy 107
5.7 Power profiles of the Nparallel_MxM_matrix_products on Cavium ThunderX2 109
5.8 Nparallel_MxM_matrix_products global power summary 110
5.9 Power profiles of the distributed_matrix_product_10240x10240 for 1 node 111
5.10 Power profiles of the distributed_matrix_product_10240x10240 for 2 nodes 112

List of Tables

2.1 Continents HPC shared systems 36

4.1 Cache characteristics in the 128 cores VPU platform 75

5.1 Execution time and power breakdown for Blocking point strategy 113

List of terms

Roman Symbols

AICS RIKEN Advanced Institute for Computational Science

API Application Programming interface

APL ARM Performance Libraries

ASIP Application-Specific Instruction Set Processors

ATLAS Automatically Tuned Linear Algebra Software

CERE Codelet Extractor and REplayer

CISC Complex Instruction Set Computing

CMT Cluster Management Tools

CPE Computer Processing Element

CPI Cycle Per Instruction

CSC Cache Snoop Controller

FFT Fast Fourier Transform

HAS High level Architecture Specifications

HBM High Bandwidth Memory

HPC High Performance Computing

HPCG High Performance Conjugate Gradients

IP Intellectual Property, generaly used in this document to designate under license features

ISA Instruction Set Architecture

ISA Instruction Set Architectures

LAPACK Linear Algebra PACKage

LISA Language Instruction Set Architecture of ARM fast models

MPE Management Processing Element

MPI Message Passing Interface

MPSoC Multi-Processor Systems-on-Chip

NoC Network-on-chip

OpenMP Open Multi-Processing

QPI QuickPath Interconnect

RISC Reduced Instruction Set Computing

RTL Register-Transfer Level

SIMD Single Instruction, Multiple Data

STT Structural Simulation Toolkit

SVE Scalable Vector Extension

UPI UltraPath Interconnect

VPU Virtual Processing Unit

Greek Symbols

HPC High performance Computing

Part I

BACKGROUND

Chapter 1

Introduction

The performance of supercomputers has traditionally grown continuously with the advances of Moore's law and parallel processing, while energy efficiency could be considered as a secondary problem. But it quickly became clear that power consumption was the dominant term in the scaling challenges to reach the next level. It is roughly considered that 20 times energy efficiency improvement is required for exascale computing (1018 FLOPS) to cope with the tremendous electrical power and cost incurred by such computational capacity. The idea of using concepts borrowed from embedded technologies has naturally emerged to address this.

First prototypes based on large numbers of low power manycore microprocessors (possibly millions of cores) instead of fast complex cores started to be investigated, putting forward a number of proposals for improvement at node level architecture to meet HPC demands. These works covered a variety of 32-bit RISC cores ranging from Arm Cortex-A8 and Cortex-A9 to more recently Cortex-A15 and Cortex-A7 cores and addressed for example dual and quad core systems based on Arm Cortex-A9 cores.

The different results indicated various processing limitations to meet HPC performance requirements, in terms of double precision floating point arithmetic, 32-bit memory controllers (limiting the address space), ECC memory (e.g. for scientific and financial computing), and fast interconnect (communication intensive applications) and additionally confirmed that the variability in performance and energy could largely be attributed to floating point and SIMD computations, and interactions with the memory subsystem.

Other works which addressed explicit comparison against x86 based systems also pointed out the need for higher levels of performance to meet HPC demands. [4] concludes that

the cost advantage of Arm clusters diminishes progressively for computation-intensive applications (i.e. dynamic Web server application, video transcoding), and other works like [8] conducted on Arm Cortex-A8, Cortex-A9, Intel Sandybridge, and an Intel Atom confirmed that Arm and x86 could achieve similar energy efficiency, depending on the suitability of a workload to the micro architectural features at core level. Of the works addressing the feasibility of Arm SoCs based HPC systems, efforts focused widely on single-node performance using micro benchmarks. Fewer studies considered large-scale systems exceeding a few cores even though multinode cluster performance is an essential aspect of future Exascale systems.

Considering further that new generations of cores such as the ARMv8-A ISA support features to improve specifically on HPC workloads (64-bit address space, 64-bit arithmetic, high speed interconnects, fast memory hierarchies), this work is one of the first to describe outcomes of research opened up with these perspectives.

1.1 Objectives and Contexts

The different contributions of this work take place in the context of European Exascale reseacrh efforts funded by the European Commission under the Horizon 2020 programme. Especially, they are part of the long term family of Mont Blanc 1/2/3 projects investigating Arm based HPC clusters and and their software ecosystem for the realization of densely integrated HPC compute nodes under critical system power constraints. The general philosophy of these approaches is to build on the long experience gained in embedded system technologies to bring processing and power efficiency to the next level. Aspects of this research therefore address centrally the recent 64-bit ARMv8 architecture for power efficiency, but also the unavoidable architectural (parallelism and memory hierarchy), methodological (design space exploration, hardware and software co-development), and runtime (power and resource management) related aspects.

1.2 Thesis contributions

1.2.1 Main contributions

A first contribution therefore is to describe an evaluation of available tools, models and platforms able to set the foundations of a methodical system level exploration approach for HPC applications scaling up to 128 Arm 64-bit cores and how it was used to examine

the relevance of SoC partitioning to limit complexity, cost and power consumption. A second achievement is, based on previous methodology, to deeply explore and evaluate the relevance of chip level partitioning based on the SoC Coherent Interconnect (SCI) developped by Atos Bull for its Exascale interconnect (BXI) technology. Finally, a last part of the study addresses the runtime aspect and investigates specific HPC improvements at power management level to also account for the important amount of power that can be saved additionally at runtime. In addition to this global perspective, the following more specific points are other contributing elements to Exascale research efforts: i) The evaluation of different tools, models and methodologies allowing the design and analysis of large HPC sytems possibly made of several compute nodes. ii) The definition of an approach allowing mutli level hardware and software analysis. iii) The design choices description and evaluation of a SoC Coherent Interconnect design. iv) The analysis and evaluation of a directory filtering based cache coherence management protocol in a large multi SoC design context. v) An efficiency analysis of parallel programming models. vi) An investigation of specific power strategies for HPC and their implementation analysis on virtual and real platforms. vii) The potential of specific HPC power strategies to further improve power savings.

1.2.2 Published papers

- WELOLI, J. Wanza, BILAVARN, Sébastien, DE VRIES, Maarten, et al. Efficiency modeling and exploration of 64-bit Arm compute nodes for exascale. Microprocessors and Microsystems, 2017, vol. 53, p. 68-80.
- WELOLI, Joël Wanza, BILAVARN, Sébastien, DERRADJI, Said, et al. Efficiency Modeling and Analysis of 64-bit Arm Clusters for HPC. In : 2016 Euromicro Conference on Digital System Design (DSD). IEEE, 2016. p. 342-347.
- WELOLI, Joël Wanza, BILAVARN, Sébastien, DERRADJI, Said, et al. Modélisation et analyse de l'Efficacité de supercalculateurs HPC basés sur l'ISA ARMv8 64-bit. GDR SOC SIP 2016 (Poster).
- Joël Wanza Weloli, Maarten De Vries, Said Derradji, Sébastien Bilavarn, Cecille Belleudy. Platform Architect for HPC ARM-based SoC Design. Synopsys User Group France. June, 2017.

1.3 Thesis outlines

The outline of the thesis is to present firstly a detailled discussion of state of the art efforts related to Exascale High Performance Computing (HPC) systems covering the most relevant academic and industry architecture and research projects worlwide. The core of the matter is made in chapters 3, 4 and 5 adressing respectively methodology definition and model evaluations, architectural exploration and analysis, and the investigation of specific HPC power management strategies. Each aspect comes with its own experimentation, result analysis and conclusions that are drawn keeping as close as possible to realistic operation (real platforms, operating system, runtime software, model relevance, benchmarks, etc.) to assess processing and power efficiency improvements as reliably as possible. This will lead finally to a global conclusion in chapter 6 to indicate the key achievements that can be formulated from the different results and several perspectives arising from them.

Chapter 2

State-of-the-Art

2.1 HPC background

High Performance Computing appeared with the continuous and increasing needs for computation power to perform large amounts of complex and scientific workloads. The main ambition in this context has always been to exploit the most advanced technologies to deploy such large scale systems also referred to as "Supercomputers". A Supercomputer is composed of a massively parallel array of thousands of processors in a way to realize very greedy computational requirements.

HPC aplications are typically based on processing real numbers. This is why the unit of measurement of HPC cluster performances is not the number of executed Instruction Per Second (IPS) but rather the number of Floating Point Operations (64-bit double precision) Per Second (FLOPS). Appropriate floating point units must be therefore at least 64-bit wide to enable the encoding of large numbers. This is why only 64-bit Instruction Set Architectures (ISA) are used in current HPC systems.

2.1.1 History

As always, the idea precedes the innovation and the history of HPC is no exception to the rule. Indeed, the term "super computing" was first introduced in the New York World newspaper in 1929 to designate a set of IBM tabulators at Columbia University. In the 1960's, Seymour Roger Cray, an American electrical engineer, became one of the pioneering architects of supercomputer systems. He designed the CDC 6600 series, a family of computers for the Control Data Corporation which was released in 1964 and is usually considered to be the first supercomputer in history[1, 2].

Half a century later, the most powerful supercomputer in the world is currently the Sunway TaihuLight in China with 93 petaflops (200 billion times faster than CDC6600) [June 2017 http//top500.org]. Most HPC projects in the world today aims at reaching the exaflops level of performance at the 2020+ horizon. However, performance is no longer the unique preoccupation of modern HPC designers. With the tremendous electrical power and cost incurred by such computational capacity, power consumption quickly became a major player in the overwhelming system complexity introduced by this new supercomputing milestone.

2.1.2 Exascale Challenges

An exascale supercomputer is a HPC system capable to reach its peak performance at one exaflops. The exascale challenge can be more specifically stated as the requirement to build a system with a power footprint similar to that of petaflops machines. Therefore, the objective is to improve performances by a factor of 1000 under a power budget of 20MW, so that the performance per watt ratio is 50 GFlops/watt. First exaflops supercomputers are expected around 2020 by the HPC community (figure 2.1).

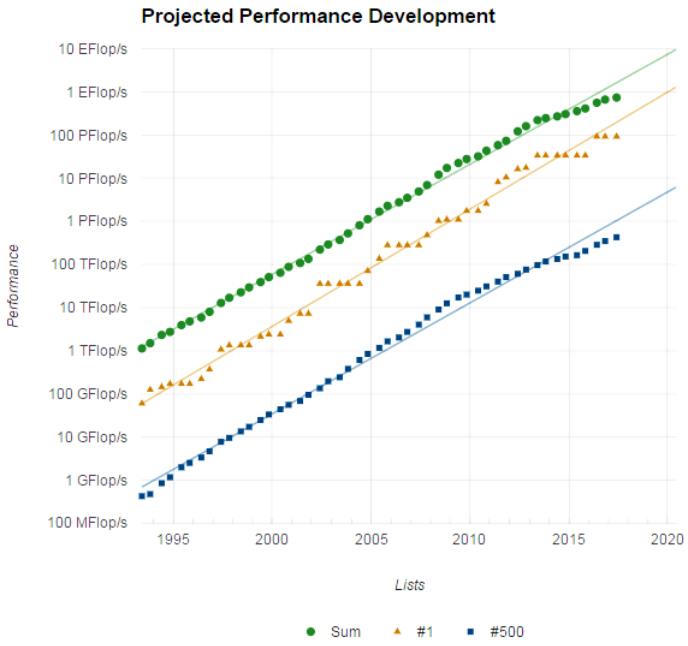


Figure 2.1: HPC systems performance projections
https://www.top500.org/statistics/perfdevel/

The first question which comes to mind is whether we can effectively exploit such a machine or not. Are HPC applications ready to get the benefits of the considerable engineering and building efforts associated? Actually, after reaching the petaflops milestone in 2008, it appeared that only few applications were able to fully exploit the capabilities of the system a that time[3]. So one of the major problems is to address scalability with applications whereas there is no actual system yet. Namely, scalability of a workload is the capability to maintain the same performance efficiency while increasing the number of parallel processors in a cluster. Considering that improving applications is a continuous effort over time, some approaches call for further efforts in formal modeling, static analysis and optimization, runtime analysis and optimization, and autonomic computing by successive and stepwise improvements [3]. Application scalability concerns rely on an incremental process adjusted on an ongoing basis when new information and asumptions about a given exascale system become available.

With a large cluster of compute nodes, the interconnection network represents a relevant portion for any HPC system. In an exascale architecture, the costs both in terms of economic and power consumption can't be overlooked and is very much dependent on the type of network fabrics used and the deployed interconnexion topology. It has been observed that many proposals addressed large-radix routers to build scalable low-distance topologies with the aim of minimizing these costs [4]. The main criticism with this approach is that it fails to consider a potential unbalance of the network utilization which may lead to suboptimal designs. Therefore concepts in advanced geometry can help to achieve optimized non-intuitive solutions. This is the case for example in [4] where authors propose a set of networks based on incidence graphs of projective. These graphs form plans based on generalized and very symmetrical Moore's graphs. In graph theory, a Moore graph is a regular graph with a maximum number of vertices for a given degree and diameter (see figure 2.2) [5]. Their simulations show that projective networks provide very good scalability and well balanced network utilization. This may offer a competitive alternative for exascale-level interconnection network design [3, 5].

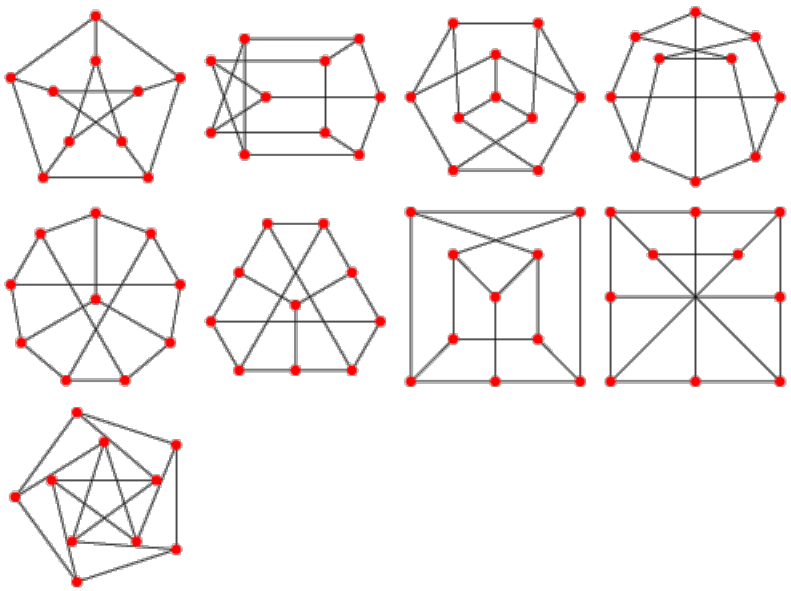


Figure 2.2: Petersen Graph (Moore Graph illustration)
http://mathworld.wolfram.com/PetersenGraph.html

Exascale systems will obviously require a large and complex network topology supporting hundreds of thousands of endpoints. In the top 500 Supercomputer Rankings, top-ranked HPC systems commonly use a specific high bandwidth interconnect for the compute network such as InfiniBand or others (figure 2.3). Nevertheless, in [6] for example, a cost-effective interconnect solution based on Ethernet technology is proposed to scale Ethernet fabrics up to the level of exascale computing considering the expected topology, routing, forwarding table management and address assignment.

Another critical aspect in upcoming systems relates to reliability and resiliency. This is a real issue for designers, producers and users of today's large scale computing systems. Moreover, due to the large size of modern HPC installations, no hardware vendor can carry out full-scale product testing before delivery to a supercomputing center. The problem becomes even more critical with the increase at system-scale level. Authors in [7], from their experience with accelerators at the U.S. Department of Energy laboratories, present in experimental and analytical data to characterize or quantify errors impacts today and future large-scale systems. They have tested the raw sensitivity of GPU memory structures based on Fermi and Kepler architectures. They observed no significant differences except that the Kepler based GPU was always less prone to bit

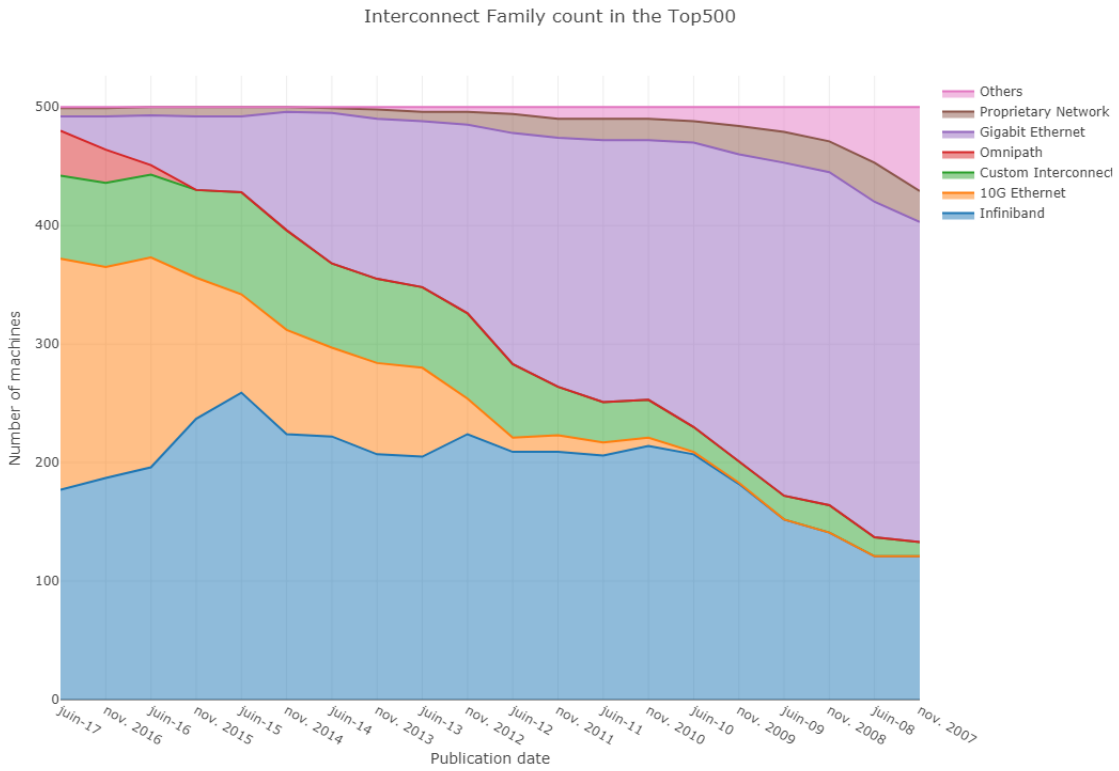


Figure 2.3: Interconnect Familly in the Top500

corruptions. Authors attributed this improvement to the better cell design of Kepler architecture. This leads them to start studying the benefits and tradeoffs associated with fault injection and AVF (Architectural Vulnerability Factor) analysis using Multi2Sim simulation infrastructure[8]. The goal was to show how applications can be designed more robustly to reduce vulnerability.

However, there are always unexpected variations of performance in contemporary large-scale supercomputers independent from the inherent system realiability and resiliency. Such as bandwidth issues, inconsistent error logging, detectable uncorrectable errors or silent data corruption[9]. In [10], the authors explore a learning based mechanism that allows a system to generate holistic models of performance variations based on runtime observation of workloads. These criticality models help applications to detect execution variations when they occur in order to take relevant actions to reduce their effect. Simulations based on logistic regressions on small scale clusters and [10] show an accurate modeling way of criticality factors.

Nevertheless, as long as computing power in large scale HPC clusters increases exponentially over time, failure rates will necessary follow the same path. Learnings from the petaflops experience reflect that some types of failures happen every few days (such as storage failures at checkpoint/restart or node and cluster node oustages)[11]. Current hardware based resilience detection techniques of the next generation of HPC exascale supercomputer are projected to reach several failures per hour [8–13]. Different approaches are possible to address this. An alternative partial memory protection scheme based on region-based memory management is proposed for example in [13] to provide application agnostic fault protection based on a concept of regions called 'havens'.

Many other innovative solutions and ideas are developped to work also on low cost and power efficient high end computational resources. In [14] for example, a collaborative solution is proposed in order to assign huge amounts of computational resources for complex university projects workloads. This approach relies on peering idle computational resources of users connected to the network. Of course this solution can not compete with exascale because the relative computing power limitation would require billions of user devices to approach the exascale level of performance. However this method can help better exploit external unused resources to reduce power consumption instead.

From this overview of considerations involved in exascale research, it comes out that the main challenges are to address parallelism concurrency, data locality management and error resilience. Therefore the exascale challenges cover many expertise areas: energy and power, memory and storage, concurrency and consistency, locality and resiliency and a huge software migration complexity on the burden of HPC software developers. Regarding this, HPC developers have therefore to take another step towards software migration. Large scale system such as an exascale system implies a new complexity, as shown in [15] in the scenario demonstrating challenges that may results from the migration of OS-bypass network example. The next sub-section (2.1.3) introduces which are the areas covered by HPC applications.

2.1.3 Applications

Applications for High Performance Computing are specifically designed to take advantage of the theoretical supercomputing power inherent to the parallel nature of HPC systems. Just like supercomputer designers who are seeking to exploit all possible technology advances, the challenge for software developers is to keep the computational efficiency of applications as close as possible to the system peak performance. Several open source standards and industrial programming models, compilers and profiling tools with domain-specific software stack libraries dedicated to HPC are facilities that can be used in multiple types of applications.

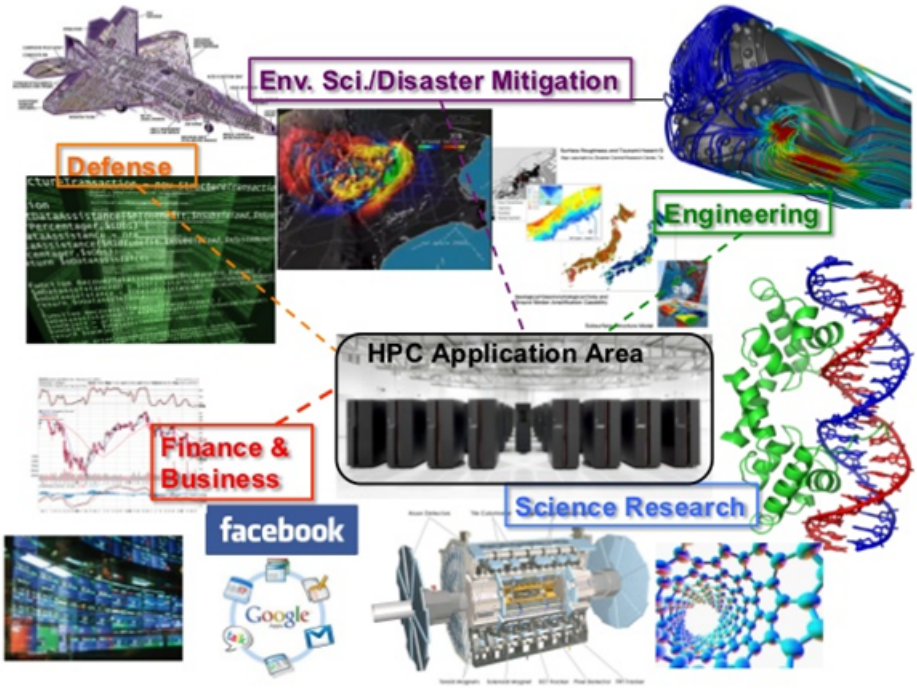


Figure 2.4: Non-exhaustive HPC applications

HPC applications cover a very large variety of domains (figure 2.4). A non-exhaustive list of areas might include astrophysics, bioinformatics and biophysics, biology, chemistry and biochemistry, climate and weather modeling, computational fluid dynamics and heat transfer, databases and data mining, deep learning, finance and bank, geophysics and Earth imaging, material sciences, mathematics, modeling, signal and image processing, molecular dynamics, nanoscience, physics, quantum chemistry, structural mechanics and so on. These applications have in common a strong requirement in terms of execution

time to perform an important amount o complex computations.

These application domains are generally considered to have one of these characteristics: sensitivity to memory latency, bandwidth, communication, and sometimes all of them. A study conducted on the Stampede supercomputer at the Texas Advanced Computing Center (TACC), one of the most performant open science HPC system, have shown for example that only ten of their workloads approached 50% usage of the execution resources. This has led the authors in [16] to emphasize the impact of hardware sensitivity in predicting the performance of important workloads (running on specific processors) to allow HPC centers to design better performing system configurations.

To deliver both processing and memory performance, the next generation of HPC systems will have to associate many-core compute nodes with heterogeneous memory technologies such as to make the best bandwidth and latency trade offs. For example, the Intel Xeon Phi Knights Landing (KNL) technology goes in this direction by combining a high-performance RAM interface for 3D-stacked DRAM (High Bandwidth Memory) in addition to a high capacity but low bandwidth DDR4 technology. This processor has been experimented in a few application studies [17, 18]. Authors in [17] address the problem with a memory-heterogeneity aware run-time system to guide data prefetch and eviction processes. Their results show a performance improvement by a factor of two for the matrix multiplication.

However, while HBM should increase performances up to four times compared to conventional DDR4 SDRAM, the conclusions drawn from [18] are basically that many factors impact the real effectiveness on a set of representative scientific and data-analytics workloads. Results indeed show that there is a real benefit from using HBM for applications with regular memory accesses with up to three times better performance. On the flip side, applications with random memory access patterns are latency-bound and therefore suffer from performance degradations when using only HBM. Activating hyper-threading (thus doubling the number of processes or threads) may reduce this effect.

Another important aspect for the emerging class of HPC applications is the importance of runtime optimizations. Authors in [19] and [20] have investigated the impact of compiler optimizations and hardware predictability. In fact, hardware predictability allows to minimize the allocation of hardware resources at runtime [20]. The reason behind this is the claim that the strict power efficiency constraints required to reach

exascale will dramatically increase the number of detected and undetected transient errors in future systems [19]. Their results show that highly-optimized codes are generally more vulnerable than unoptimized codes for several mission-critical applications. Sustainable trade-offs must be found between compiling optimizations and application vulnerability, because certain types of optimizations may provide only marginal benefits and considerably increase application vulnerability. These considerations show the intrinsic link between applications and the hardware in the search for performance. In the following, we focus on the hardware capabilities with an overview of HPC compute node architectures.

2.2 HPC Compute node architectures

2.2.1 Introduction

To avoid any confusion, we first clarify the terminology used in this paragraph. In the abstract example of figure 2.5, the system is considered a bi-socket compute node because it contains two CPUs. A compute node may well be mono or multi-socket but it must have a single operating system. The socket here is a co-processor which contains other processors generally connected through a Network on Chip (NoC) interconnect. In this example, each processor is homogeneous and has a multi-core architecture. Each core implements a Simultaneous Multi Threading (SMT) technology and has two Hyper-threads. SMT is an option that can be turned on or off, used to improve parallelization of computations in a signle core.

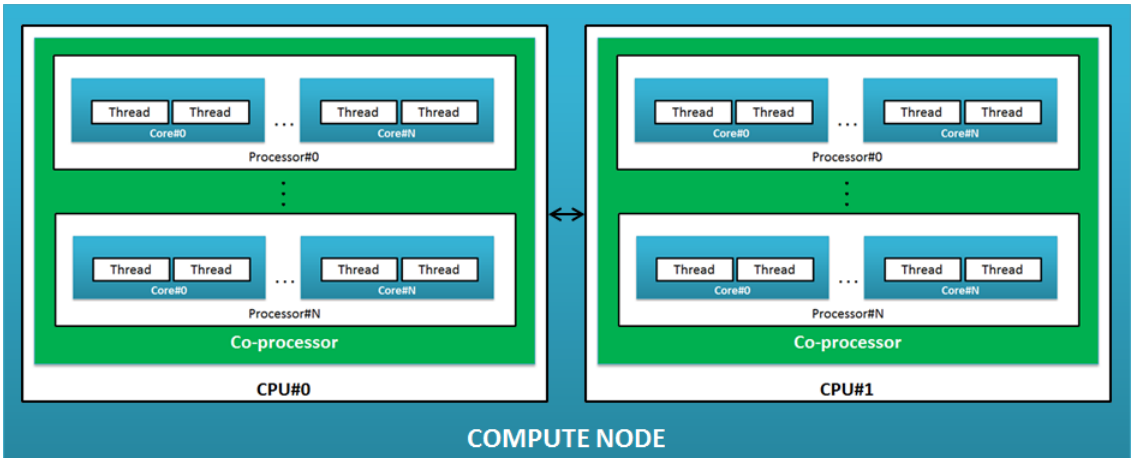


Figure 2.5: Imbrication of annotations in a compute node

The HPC compute node is the basic building block of the entire supercomputing infrastructure. In practice, HPC is about aggregating the computing power produced by several of these compute nodes. So a compute node is basically a server designed with the most advanced technologies to deliver the highest performance. It includes processors, memory system, disk and interconnect. The architecture has direct impacts on both the mechanical aspect of the server packaging and the thermal management of energy dissipation produced by the boards. In a HPC cluster, we distinguish two types of nodes: compute nodes as introduced previouslly and login nodes. Compute nodes are used for computation only while login or master nodes are responsible of the Cluster Management Tools (CMT) which handle functions such as job scheduling, cluster moni-

toring, cluster reporting, user account management, and power management strategies.

The main focus of High Performance Computing is always about the execution time deployed to compute a workload. In a HPC cluster, the workload to be processed is composed of a very large amount of parallel threads, each one being bound to a processor core. In an ideal parallelism situation, global performance is constrained by the execution time of the slowest thread but in pratice, workload division involves synchronizations between consumers and producers of data. An harmfull consequence of desynchronization is when the execution of cores does not benefit from stable loads (unbalanced threads, heterogenous cores, complex data dependencies) which may dramatically lead to very poor processing efficiency.

This is one of the reasons why at the hardware level, all basic compute node architectures are usually supposed to be composed of either homogeneous CPUs or GPUs. The software effort to reduce desynchronization issues is also addressed by trying to avoid unnecessary barrier synchronizations and limit the operating system "noise" which is responsible of unbalanced schedule of running processes between cores. Isolation of computing cores is also employed by dedicating one core to host the OS. However the cost of this solution can be very expensive for multicore architectures with high performance cores while it may be acceptable in a many-core system based on low power and smaller cores.

Figure 2.6 and 2.7 shows how x86-64 processors dominate the panorama of HPC. Intel mainly proposes two families of processors to supply the high-end and high performance server market: Intel Xeon E series and Intel Xeon Phi series.

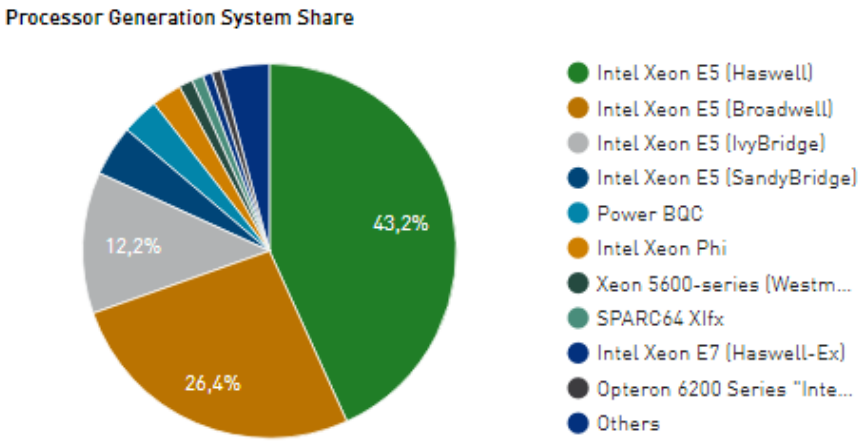


Figure 2.6: x86 architectures hegemony in the HPC Top500 [June 2017]

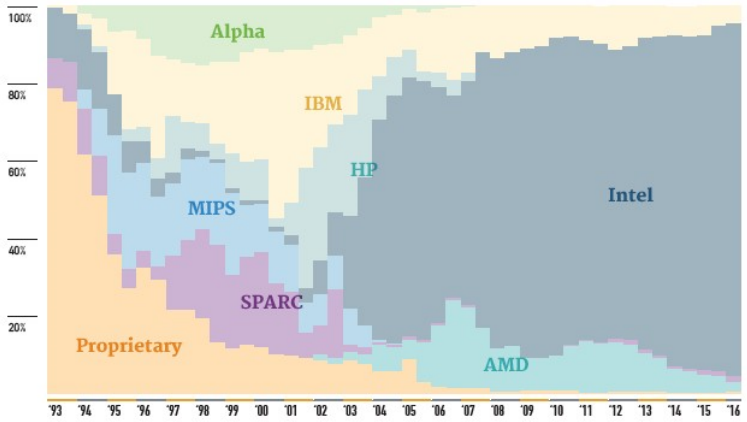


Figure 2.7: Intel processors hegemony history in the HPC Top500
https://www.nextplatform.com/2016/06/20/china-topples-united-states-top-supercomputer-user/

2.2.2 Intel Xeon E series architecture: Skylake (Haswell)

Intel®'s 6th generation Core™ microarchitecture is usually designated under the development code name "Skylake". Skylake results from a continuous effort since Nehalem, SandyBridge, IvyBridge, Haswell and previously Broadwell Intel x86 based microarchitecture. The Xeon Skylake-EP series are processors targeting high-end requirements in terms of power and performance that are well suited for HPC processing. For example the Intel Skylake-EP Xeon E5-2699 V5 features a 32 core platform produced in 10nm process with 48 Mo L3 cache, 6 memory channels and high bandwidth DDR4 memory

[12].

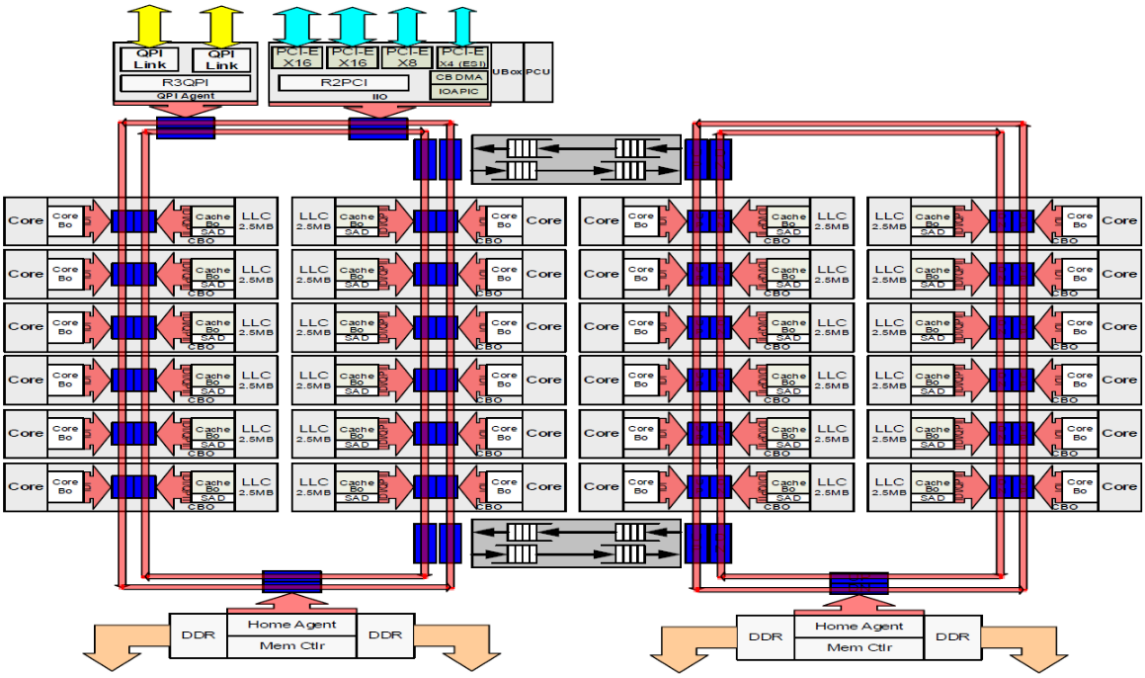


Figure 2.8: Haswell EP Architecture block diagram

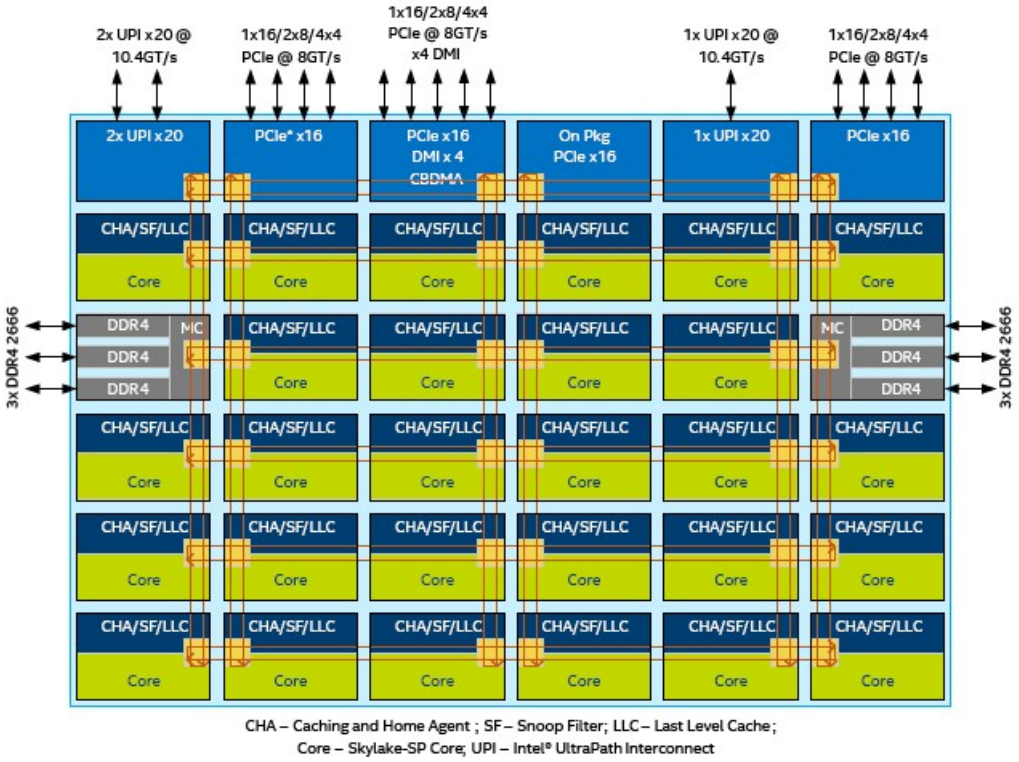


Figure 2.9: Skylake Architecture block diagram

Figures 2.8 and 2.9 respectively represent the architecture block diagrams of Haswell and Skylake processors. There is a inherent difference based on the manufacturing technology 14nm process for Broadwell and 10nm for the latest Skylake processors, which leads obviously to a reduction of power consumption. However, one of the substantial evolution is the on-Chip coherent interconnect: Broadwell deploys a double ring bus (two set of rings connected through on-die bridges) interconnect between cores whereas Skylake is based on a 2D Mesh topology.

The mesh architecture improves scalability (more cores on the same die) with a higher bandwidth and reduced latencies. This topology also improves data sharing and unbalanced memory accesses between all cores/threads. Another technique to scale up the number of cores and the memory of Xeon-E based platforms is to use sophisticated point-to-point processor interconnect such as Intel UPI/QPI (UltraPath Interconnect/QuickPath Interconnect) proxy modules (figure 2.10). This type of platform is not well-suited for HPC requirements because of the effort needed to maintain coherency between CPUs which increases latencies without a significant gain of bandwidth.

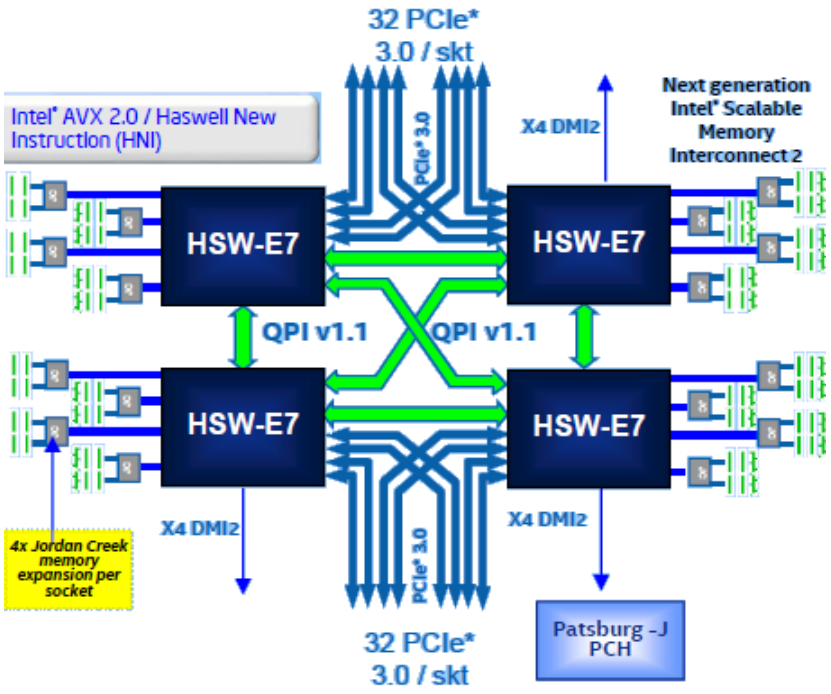


Figure 2.10: Haswell based quad-socket platform example

Like most CPUs designed for performance, Skylake includes vector processing units in addition to classic arithmetic units. Single Instruction Multiple Data (SIMD) is the common technique shared by all vector units. Skylake has support for Advanced Vector Extensions (AVX-512) which define 512-bit wide operations. Depending on the size of data types, this means for example that 16 32-bit operations (single precision) can be processed simultaneously at the same CPU clock rate. So for a core running at 1GHz, it has a theoretical performance of 16 Gflops (single precision) or 8 Gflops (double precision).

2.2.3 Intel Xeon Phi Coprocessor architecture: Knights Landing

Intel Xeon phi coprocessors are more recent compared with the Xeon-E family. This generation of coprocessors is based on the Many Integrated Core (MIC) Architecture. As opposed to classic multi-core architectures like Xeon-E processors, the system combines smaller and lower-power performance multi-core processors into a single chip. Their performance is equivalent to those of classical high speed CPUs but they also provide

massive parallelism and vectorization capabilities that are more suited to massively parallel software. This type of architecture is considered to be more relevant for most HPC workloads, extending as well the benefits in terms of of power efficiency.

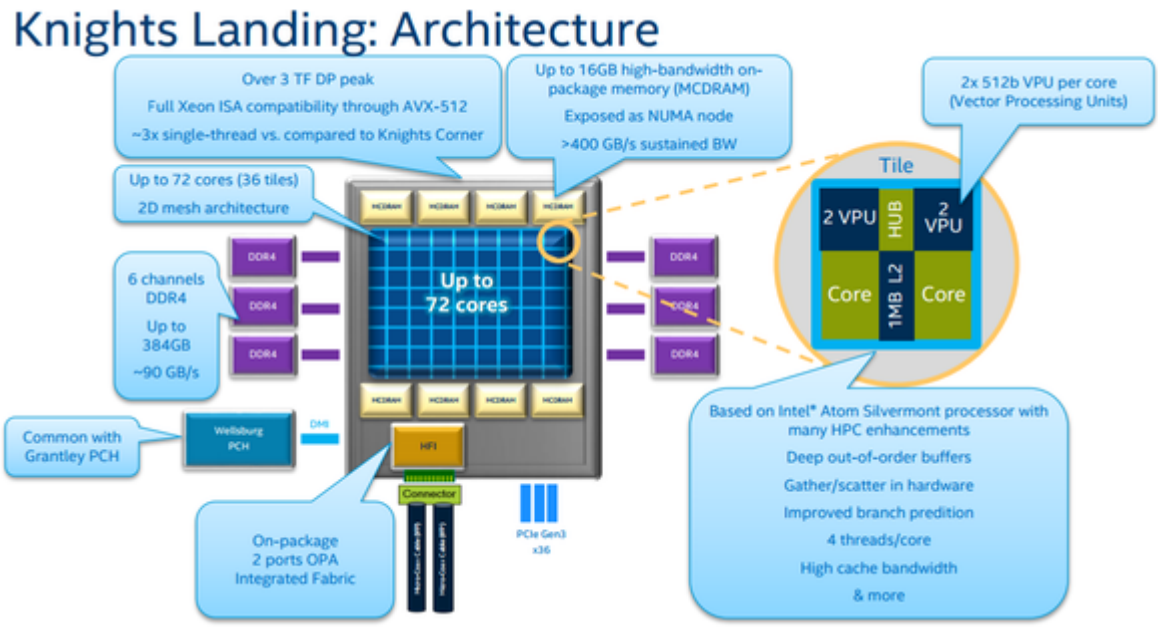


Figure 2.11: Knights Landing Architecture block diagram overview

The Knights Corner product family is the first MIC architecture produced by Intel. They have been designed in 22nm process technology and combine 50 coprocessors. Knights Landing is the codename of the second generation of Intel Xeon Phi architectures manufactured in 14nm process. They integrate 3D-stacked on-Package Multi-Channel DRAM (MCDRAM), a version of High Bandwidth Memory (HBM), and up to 72 Silvermont cores (Atom processor with four threads per core). In addition, each core has two 512-bits vector processing units with support for compatibility with AVX-512 SIMD instructions. Knights Hill will be the third generation, following Knights Landing but produced in 10nm process technology [21].

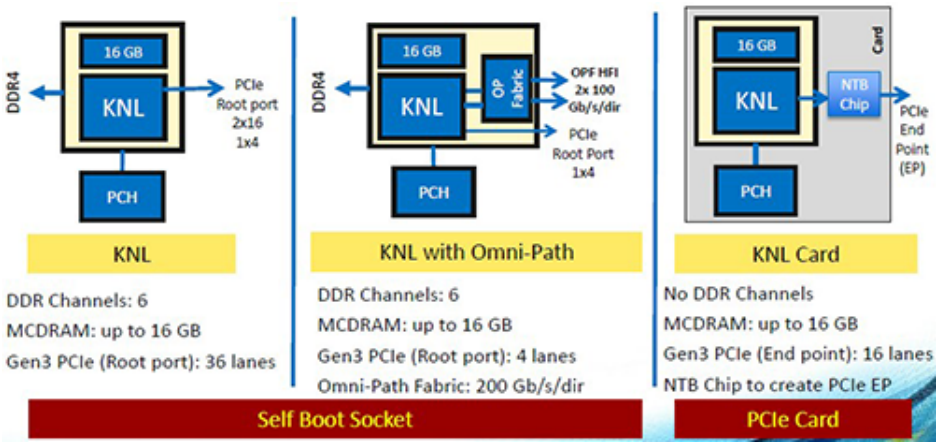


Figure 2.12: Knights Landing (KNL) based Chips

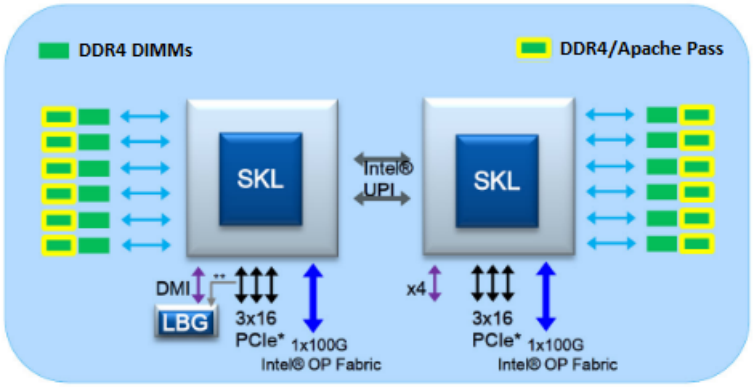


Figure 2.13: Purley Platform: bi-socket Storm Lake Integrated with Skylake

Figure 2.12 shows examples of chip configurations involving Knights Landing coprocessors. OmniPath (OP) is an Intel proprietary fabric that can be deployed as a HPC cluster interconnect. However, KNL chips provide a PCIe root port that can host a PCI device from others compatible fabrics, like Infiniband. Figure 2.13 shows an example of a Skylake based bi-socket motherboard. This platform commonly serves as a compute node in most of Intel Xeon processors based Supercomputers.

2.2.4 Emerging compute node architectures: SunWay

Sunway architecture is a China homegrown 64-bit Reduced Instruction Set Computing (RISC) based machine ISA. The custom-designed Sunway SW26010 processor chip is the 4th generation of this family. These processors are used in the Sunway TaihuLight

supercomputer which currently ranks first in the last HPC Top500 (august 2017). The SW26010 processor is a many-core system like Intel Knight Landing but it integrates much more lightweight cores and a cache-free architecture (no L2, L3 caches). Actually, a Sunway SW26010 compute node is made of four CPUs (SW26010 processor). Each CPU contains one Management Processing Element core (MPE), an 8x8 mesh of cores called Computer Processing Element (CPE) and a memory controller supporting 8GB DDR3 [22].

The cores in MPE and CPE are based on a 64-bit RISC architecture, SIMD instructions, and out of-order execution running at 1.45GHz. They can all participate in the workload computations but only MPE cores can be responsible of performance management and Input/Output communications [22]. In addition, each MPE core has a 256KB L2 cache and dual pipeline (8 flops per cycle per pipeline) while each CPE core is cache-free and has only one signle floating point pipeline that performs 8 flops per cycle per core (64-bit floating point arithmetic). Therefore the SW26010 compute node has a theoretical performance peak of 4(CPU clusters)*64(CPE cores)*8 flops/cycle*1.45GHz + 4(CPU clusters)*1(MPE core)*16 flops/cycle*1.45GHz or 3.0624 Teraflops/second [22].

The Sunway TaihuLight supercomputer has a computation level called *supernode* which is a set of 256 compute nodes. In addition, the entire system has 40 cabinet systems, where each cabinet system has 4 supernodes. So, there are 160 supernodes or 40960 compute nodes or 10649600 64-bit RISC cores in this supercomputer. The Sunway TaihuLight supercomputer theoretical performance is about 125.436 Petaflops but the practical performance reached on a Linpack benchmark is 93 Petaflops with 15.371MW power budget. This leads to a computational efficiency (the ratio of the obtained performance per the theoretical one) of about 74.14 %. The Sunway TaihuLight system has been designed to deliver super-efficient floating point performances. Indeed, this supercomputer strikes an impressive 6.074 gigaflops/watt when most of the ten in the Top500 hit more or less 2 gigaflops/watt. But this result is still far from the 50 gigaflops/watt required to cope with exascale requirements [23].

However, Sunway TaihuLight also has limitations coming from inherent memory weaknesses. Actually when using HPCG (High Performance Conjugate Gradients), a set of benchmarks to collect better data movement metrics, it comes out that Sunway TaihuLight is lagging far behind all other systems in the ten of Top500 with six times less peak performance efficiency. This shows how moving data through the computing block

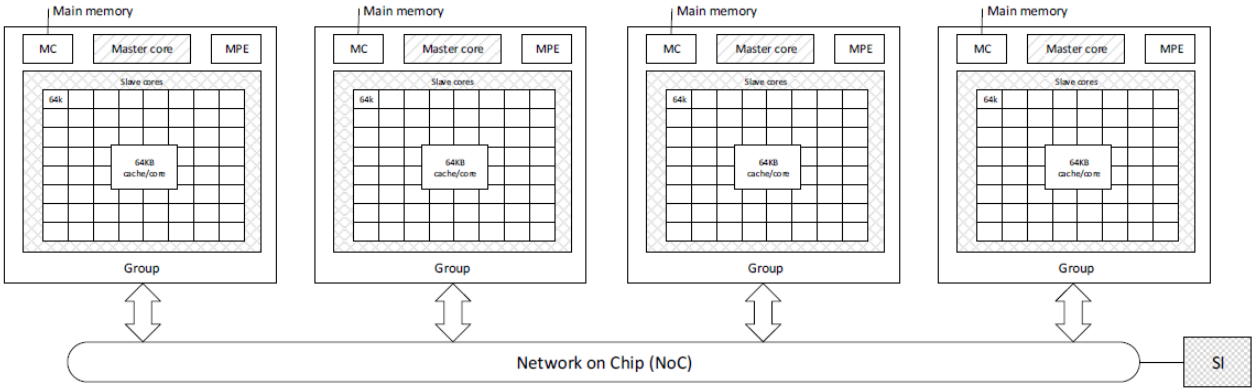


Figure 2.14: Sunway SW26010 compute node architecture diagram

hierarchy can be very expensive and strongly limit the performances of real workloads [23].

2.2.5 Emerging compute node architectures: Opteron

Opteron is an Advanced Micro Devices (AMD) x86 ISA processor designed for both desktop and server markets with support of AMD64 (x86-64). This compute node is composed of the 16-core AMD Opteron 6274 processors 16 cores(L2 1MB per core), x86, from 2.2GHz up to 3.1 GHz 32nm manufacturing process. The Titan supercomputer funded by the U.S. Department Of Energy (DOE) for the Oak Ridge National Laboratory (ORNL) is based on Opteron processors as compute nodes, the Cray Gemini Interconnect and NVIDIA K20x GPUs. This supercomputer designed by Cray and includes 299008 AMD Opteron cores and 18688 NVIDIA Tesla K20 GPU accelerators. Titan performs 17.6 PFlops and ranks as the 4th world fastest super-computer [Top500: June, 2017].



Figure 2.15: AMD Opteron

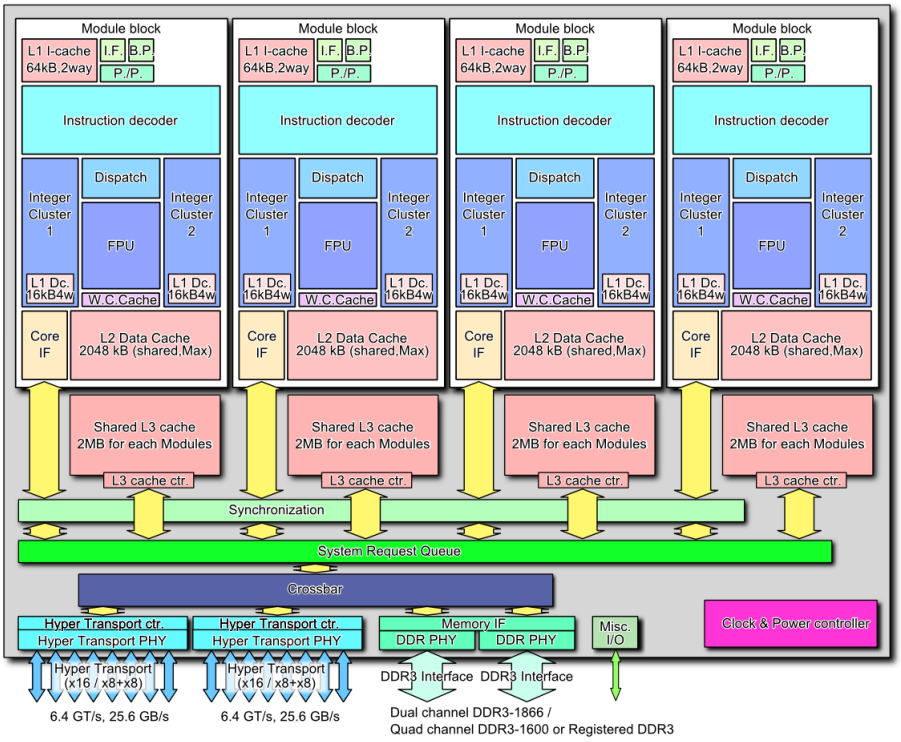


Figure 2.16: Internal view

2.2.6 Arm vs Intel Compute node architecture

The opposition between Arm and Intel architectures is no longer a question of "RISC versus CISC ISA" as traditionally discussed in the 1980s and 1990s when chip area and processor design complexity were the main constraints for hardware architects [24]. Nowadays, power and energy are the primary design constraints for the exascale computing milestone. The RISC vs CISC debate has turned secondary as Arm, deeply rooted in the field of low-power ISA for mobile and embeded systems, entered the high-peformance server market.

On this point, surveys like [24] conclude from a large measurement analysis of mobile, desktop, and server computing workloads that there is nothing fundamentally more energy efficient in one ISA class or another. However, in a comparison between Arm and Xeon processors approaching the question in terms of time-to-solution (time spent to perform a specific workload), peak power, and energy-to-solution (energy required for a given time-to-solution) [25], results lead to the conclusion that even if Arm processors (at that time) provides a lower peak power, Xeon still gives a better trade-off from "the user's

point-of-view" [25]. Even if user expectations in HPC are often schizophrenic about the trade-offs between performance and energy consumption. In another study comparing Arm Cortex-A8 and Intel Atom processors [25], concrete measurements show that Atom deliver better raw performance while Cortex-A8 has significantly better power efficiency. These conclusions are globally confirmed in [26] in a different context addressing the low-power processor Arm big.LITTLE (32-bit ISA heterogeneous Cortex-A15/A7) and a high performance Intel Sandy Bridge-EP (64-bit ISA homogeneous) processor.

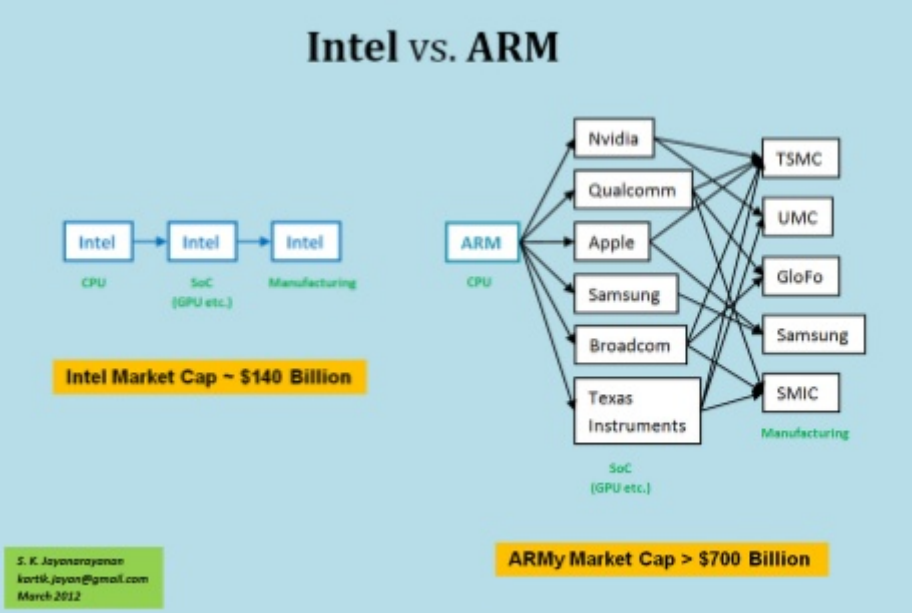


Figure 2.17: Arm vs INTEL Design flexibility

In summary, it emerges from this survey that many studies lead to divergent, even contradictory conclusions on this debate. Each side of the debate makes valid arguments but it is too early to clearly define which architecture is the most suitable for HPC applications or workloads profiles. Arm remains an unkown player in processing HPC heavyweigth workloads. To understand the trends of power consumption and efficiency, we need to consider the energy spent to power an entire HPC system[25–27]. This is precisely what we will discuss in the next section: the first Arm-based supercomputer.

2.2.7 The first Arm-based HPC Cluster: Tibidabo

Tibidabo is a mountain overlooking Barcelona, but also a code name used for the first large-scale Arm-based HPC cluster. It was designed for the Barcelona Supercomput-

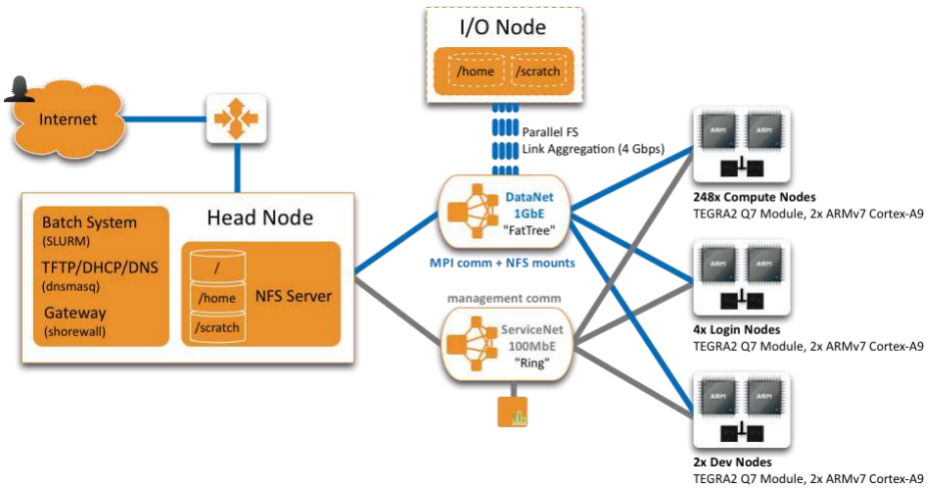


Figure 2.18: Tibidabo Arm-based Cluster
https://www.hpc2n.umu.se/sites/default/files/PSS2013%20-%20Presentation.pdf

ing Center (BSC) and exploited in their MareNostrum supercomputer[28]. The compute node in Tibidabo is a NVIDIA Tegra2 SoC including a low-power dual-core Arm Cortex-A9 manufactured in a 40nm TSMC process and running at 1GHz. Tibidabo has 248 computes nodes and achieves 120 MFlops/Watt on HPL benchmarks. Designers projects that for a theoretical cluster of Arm Cortex-A15 chips (16 cores), the energy efficiency would increase by a factor of 8.7 reaching a value of 1046 MFlops per Watt [28].

However, Cortex-A9 and Cortex-A15 are both 32-bit processors. They suffer from important limitations in this regard, such as 4 GB address space per application which is not enough for real HPC workloads. The ARMv8 64-bit ISA, announced in october 2011[29], was therefore a much awaited technology to address future server class low power applications [28]. We examine this opportunity and detail this technology in the following section.

2.3 Aarch64 architecture and ARMv8 processors

2.3.1 ARMv8 or AArch64 ?

Arm initially for Advanced RISC Machine describes a processor architecture based on the RISC ISA. To develop an Arm-based SoC chips, many others components are proposed by companies as Intellectual Property (IP). Most of these IPs are developed by the british company Arm Holdings [29]. Under an Arm architectural license, constructors such as Apple, Samsung, Cavium and many others design their own processors. This business model gives flexibility to manufacturers architects for innovative added values and so researchers in processor architecture.

ARMv8-A is the code name of a new family of low power high performance Arm 64-bit processors. It introduces 64-bit capabilities alongside the existing 32-bit mode and can therefore support the two execution modes [30, 31]:

- AArch64 featuring enhancements for 64-bit registers , memory accesses and 64-bit instructions.
- AArch32 which is optional in the ARMv8 architecture specification. Its role is to mainain backwards compatibility with the Armv7-A 32-bit ISA (Figure 2.19).

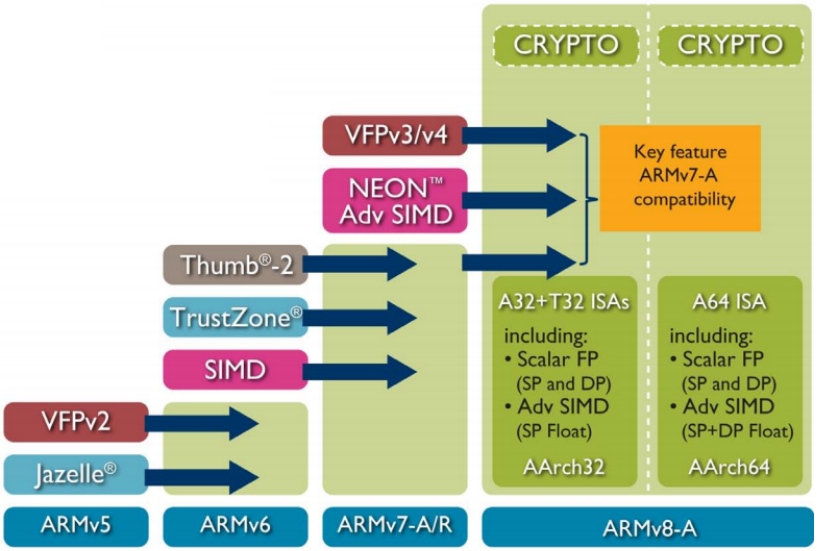


Figure 2.19: Arm architecture generations

First issues in the development of the ARMv8 ISA were the ability to access a large

virtual address space (up to 48 bits from a translation table base register) and higher native performances. ARMv8 also features an advanced SIMD engine supporting the full IEEE 754 standard and additional floating-point instructions for IEEE754-2008 [32, 33]. These extensions have been designed specifically to cope with HPC requirements to allow the expansion of Arm processors from embeded systems to mobiles (smartphone/tablet), desktops and servers usages.

2.3.2 ARMv8-A processors : Cortex-A75

The Cortex-A75 is the latest application processor of the Cortex-A series and the most powerful to date (October, 2017). The Cortex-A series is a familly of processors designed to give the highest performance. They come in addition the Cortex-R series for "Exceptional performance" intending real-time applications and the Cortex-M serries designed for cost effective deterministic microcontroller applications. The Cortex-A75 is the successor of the Cortex-A73 and both have been designed at Sophia Antipolis/France unlike previous Cortex-A72 and Cortex-A57 developped in Austin Texas. This fact explains some of the similarities or differences between both micro-architectures.

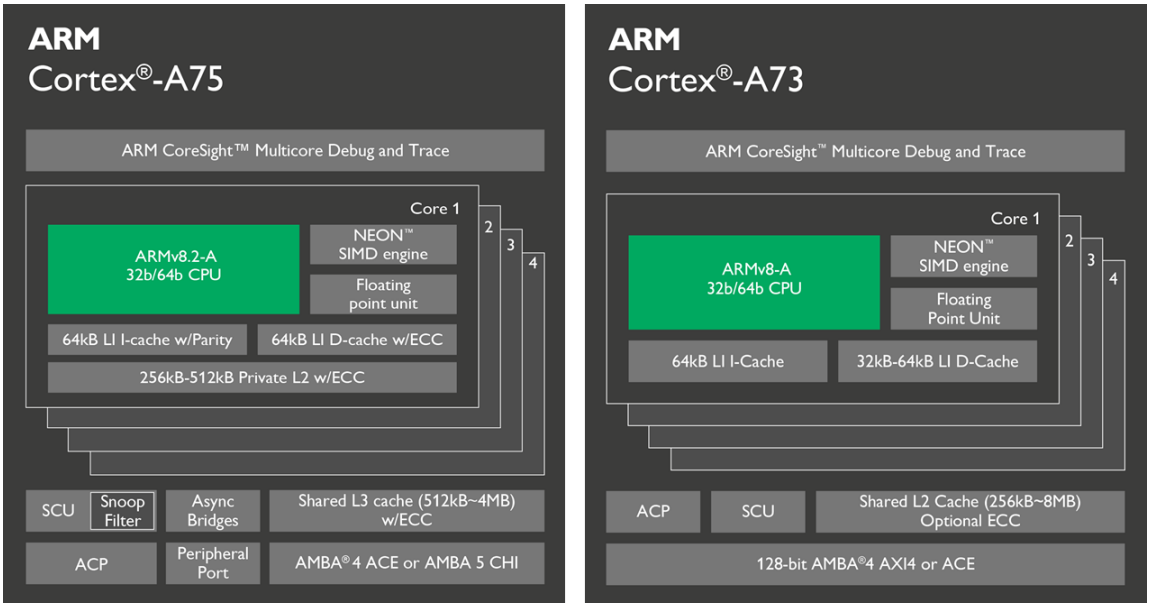


Figure 2.20: The Arm Cortex-A75 processor improvements

The Cortex-A75 is an out-of-order pipeline micro-architecture. Its intruction fetch unit handles four stages while several decode blocks provide the ability to decode up to 3 instructions per cycle or 6 NEON/FPU micro-operations with the two floating point

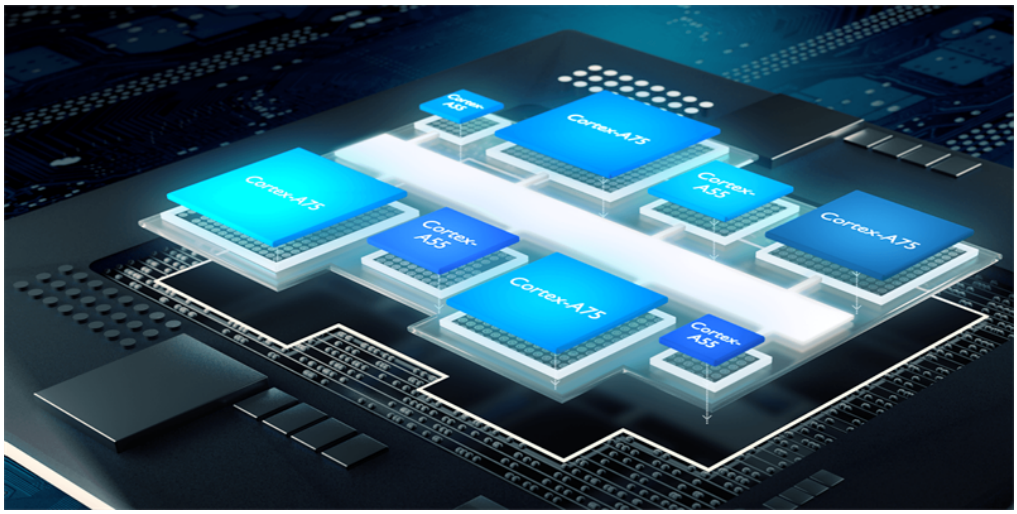


Figure 2.21: Arm DynamIQ concept overview
https://community.arm.com/processors/b/blog/posts/arm-dynamiq-expanding-the-possibilities-for-artificial-intelligence

units available. The high level view of the Cortex-A75 in the figure 2.20 show that its architecture can handle an optional integrated L2 cache operating at core speed. Reducing de facto latencies by more than 50% compare to the A73. In addition, the Cortex-A75 core comes with a new Arm concept for processing efficiency called "DynamIQ" technology. DynamIQ is similar to the well known big.LITTLE architecture but is much more flexible and allows up to 8 cores combinaisons of big and LITTLE coress in a CPU cluster (figure 2.21). These cores could be entirely different from different generations of Arm Cortex-A families [34]. For example, DynamIQ enables to include a Cortex-A9/15, Cortex-A53/57, CortexA73/A75 in same processor. A recent technique exploring efficient ways to find the best big.LITTLE configuration improving energy-efficiency is discussed in [35]. They used Armv7 (32-bit) processors Cortex-A15 and Cortex-A7 in serveral big.LITTLE architectures scenarios. Results confirm that globally, big.LITTLE architectures allow to reach better energy/performance trade-offs compared to homegeous Arm CPU clusters. DynamIQ is still under active development but the technology combined with the ARMv8 ISA intends to radically change the power efficiency paradigm for HPC.

In addition, several improvements have been done to increase the floating point processing units. They can fully perform and respect the IEEE double-precision floating operations sdantard. Their vector size grew from 16×128bit to 32×128b [36]. Nevertheless, the number of floating point operations executed per cycle remains poorer compare

to the Intel's AVX of Xeon processors seen previously here 2.2.2. This is exactly why Arm and Fujitsu have annouced a common effort to design an extension to the ARMv8 ISA called "SVE" (Scalable Vector Extension) at the Hot Chips 28 Symposium (2016).

2.3.3 SVE: Scalable Vector Extension

In theory, SVE allows extening vector length from 128 to 2048 bits [37] resulting in 2 to 32 Flops per cycle per vector. In addition SVE doesn't replace or an extension of Advanced SIMD?? the usual ARMv8 architecture but a separate architectural extension with a new set of A64 instruction encodings corriger cette phrase. SVE focuses on HPC scientific workloads and supports a vector-length-agnostic HPC programming model. This means that it scales automatically amoung all vectors length without a need of recompilation. Actually, Arm has submitted patches to GCC and LLVM to support the auto-vectorization for VSE.

2.4 Research projects

2.4.1 Design space exploration methodologies

This section discusses system level approaches associating architecture expertise and modeling methodologies in a way to analyze and study conventiently all kind of implementations choices. Indeed, nowadays, System-on-Chips are made of billions of transistors, interconnect and other components. However, several hurdles are hindering the productivity of SoC researchers and architects. The architectural complexity is continously increasing as designers refine the solution space covering a range of different axes such as processing elements, on-chip coherent interconnect, memory hierarchy or the compute node interconnect fabrics. Because of this complexity, the challenge is always to find trade-offs between the need for accuracy in a wide spectrum of configurations and simulation times.

Architectural development and analysis are generally based on system level modeling in order to abstract specific implementation details contrary to RTL (Register-Transfer Level) sources. For example, authors in [38] propose an integrated open framework for the design space exploration of heterogeneous Multi-Processor Systems-on-Chips (MPSoC). They present a pre-existing standalone CAD methodology which integrates a state-of-the-art ASIP (Application-Specific Instruction Set Processors) toolchain within a full-featured virtual platform coded in LISA for processing blocks and SystemC for interconnects and memory models. Using LISATek for the design flow, they have shown that the mix of these two environments enables SoC designers to cover all dimensions of the configuration space by getting immediate feedbacks. The main concern of the design space exploration problem is therefore a perpetual research for the most efficient way to address simulation speed and accuracy with the most advanced current technology.

Besides simulation complexity, there are also many other requirements to address in the design of modern high performance processing models of chips, such as modeling power consumption and SoC floorplanning [39]. A more recent study proposed a framework for power, area, and thermal simulation [40]. This methodology was applied for the design of a Network-on-Chip (NoC) architecture. The principle is to define an approach combining large-scale HPC simulation (SST , Structural Simulation Toolkit [41]), a set of power libraries (McPAT [42], IntSim [43, 44] and ORION[45]) and a thermal library (HotSpot [46]) within the same environment. Results showed that there was no significant difference between results produced by the two power libraries McPat and ORION.

However, today only McPat has the most up-to-date version. SST doesn't integrate ARMv8 models. Nevertheless, one of these tools can always be used for other purposes. For example authors in [47] demonstrated the gem5 tool accuracy for manycore architecture exploration. They used Armv7 ISA and a dual-core processor model of Cortex-A9. Results showed that their processing accuracy varies from 1.39% to about 18%. Nowadays, Gem5 integrates ARMv8 processors and most recent Arm features. In addition, the same authors have also demonstrated in [35] that coupling Gem5 to the power estimator McPat efficiently increases the accuracy of both processing and power modeling.

Another key idea in the Exascale process is co-design. The general principle is based on the process of code developement in closer relation with hardawre. In HPC, two main approach are used to achieve this: library based code optimization and compute kernels optimization. Authors in [48] with full awareness of implications, argue that the exascale design space exploration is prohibitively expensive, at least partially due to the size and complexity of workloads. Indeed, Application code may contain millions of lines and often relies on many independant libraries. They rather suggest to use mini-applications to capture key performance indicators which can be used in the hardware design space exploration. This methodology is expecting a potential reduction of the order of the exploration by a factor of a thousand [48].

In a similar reasoning, authors in [49] have presented a framework called "CERE" for Codelet Extractor and REplayer. CERE is an LLVM based Codelet Extractor and Replay framework. This framework finds and extracts the hotspots of an application as codelets to be replayed, modified or compiled. Their results on the SPEC 2006 FP benchmarks, shows that CERE codelets cover 90.9% and accurately replay 66.3% of the execution time. With the codelet mini-apps, CERE is a good candidate to keep the realism of HPC applications in time-consuming simulations. This method allows to reduce feedback loops time between software developers and hardware designer when co-designing a system.

2.4.2 TSAR Architecture example

Serveral studies have managed to scale up simulation challenges of modeling clusters of compute nodes. For example, the TSAR project developped an architecture supporting an agnostic processor ISA (working independently of the instruction set RISC and CISC) and maintaining coherency up to 4086 building blocks (compute nodes)[50]. The TSAR

architecture implements the Distributed Hybrid Cache Coherence Protocol (DHCCP) and a directory based coherence protocol with support of Non Uniform Memory Access (NUMA) see figure 2.22. The platform is specified in such a way that hardware components can be reused. There are two levels of simulation accuracy: Cycle-Accurate / Bit-Accurate and Transaction Level Model with Distributed Time (TLM-DT). These simulations help authors to produce an advanced network on chip solution supporting both packet switched virtual channels and a broadcast service[50].

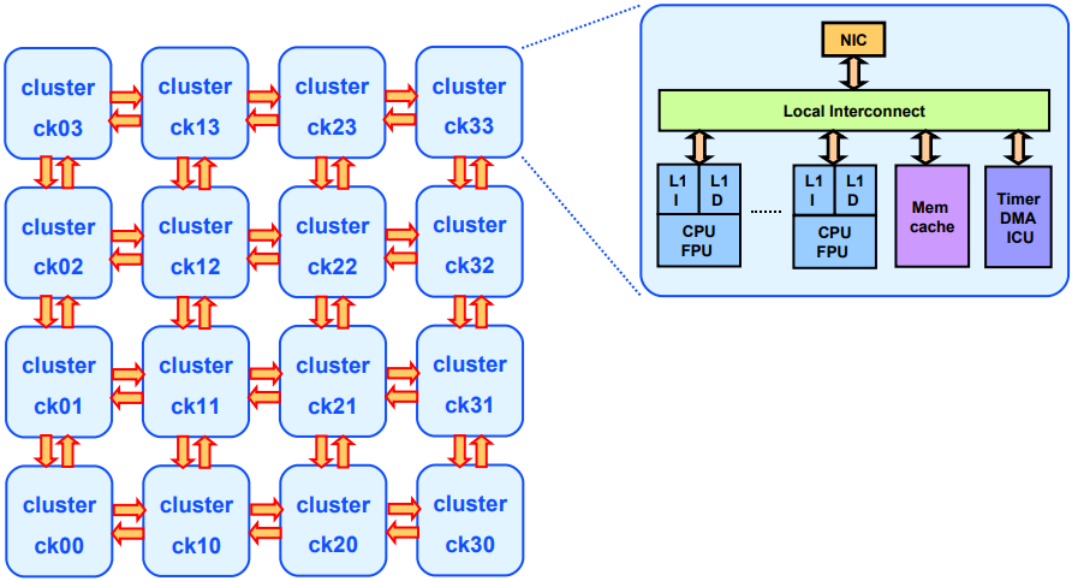


Figure 2.22: TSAR Architecture

It turns out from this overview that many codesign approaches have been investigated, with performance and efficiency requirements in mind. Each approach often tackles a separate aspect of the code optimization problem (libraries, kernel, communication, etc). A great support in the codesign process is therefore to rely on an environment where each of these problems can be addressed. In the following we review main Exascale projects around the world that have led to significant advances regarding the different issues raised.

2.4.3 Global projects overview

Asia, America and Europe accounts for 98.8% of the use of HPC systems in the world (table2.1).

Continents	Number of systems	System share (%)	Rmax (PFlops)	Rpeak (PFlops)	Number of cores
Asia	212	42.4	319.9	529.91	29 013 340
Americas	176	35.2	257.4	377.78	12 496 998
Europe	106	21.2	165.5	217.34	6 306 364
Oceania	5	1	4.5	5.8	230 232
Africa	1	0.2	1.02	1.36	32 855

Table 2.1: Continents HPC shared systems
https://www.top500.org/statistics/list/

We address more specifically in the following projects of countries who invest important efforts in exascale research.

2.4.4 Asian projects

Over four consecutive years now, China has hosted the two world's fastest supercomputers as measured by the Linpack benchmark: the 93 petaflops Sunway TaihuLight since june 2016 and the 33 petaflops Tianhe-2 (MilkyWay-2) since june 2013. China is obviously leading Asian research towards exascale computing. The country is funding at least three exascale prototypes. The first one to be delivered is Sugon developped by Dawning Information Industry and owned by the Chinese Academy of Sciences. The second exascale project is the Tianhe familly supercomputers designed by the National University of Defense Technology (NUDT), and finally the Sunway TaihuLight built by Sunway[51]. It has already been announced that the fisrt exacale prototype could be deployed sooner than expected, as soon as 2019 on the coast of Shandong province to support oceanographic research in the South China Sea and around world. Unfortunately without details on the architectural features.

In 2011, Japan held the title the world fastest supercomputer for the first time with the 'K computer'[52] running the Linpack benchmark at 8.16 Pflops (10.5 Pflops with recent updates). The K computer was designed by Fujitsu for the japanese RIKEN Advanced Institute for Computational Science (AICS) located in Kobe. Its architec-

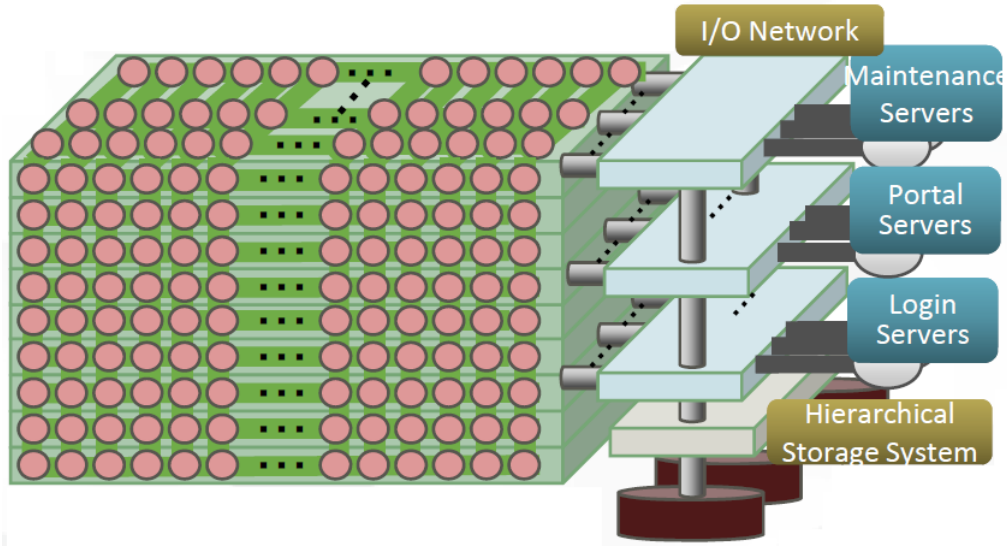


Figure 2.23: Post-K Computer Hardware

ture is based on SPARC64 processors and consists of more than 80 000 compute nodes (CPU) connected through a fujitsu interconnect called Tofu [53]. Each CPU is made of 8 cores running at 2GHz sharing 6MB of L2 cache. The CPU delivers peak performance and performance per Watt of respectively 128 GFlops and 2.2 GFlops/W. To cope with scientific and other HPC workloads, each core has 256-bit large double precision floating point data registers. There are four floating-point multiply-and-add execution units among which two can operate in parallel with SIMD instructions [52].

In april 2014, the japanese government launched the FLAGPSHIP 2020 project in order to develop a successor to the K computer and properly named the "Post K Computer" [54]. RIKEN AICS is in charge of its development and Fujitsu has been selected as the vendor partner. Both companies annouced that Post-K Computer would be based on the Arm 64-bit architecture to reach exascale performance by 2020 (mostly 2022 after recent CPU design headaches [54]). This is why, as seen in 2.3.2, Fujitsu is the lead partner of the Arm HPC extension effort SVE. The RIKEN Post-K supercomputer (figure 2.23) is based on a manycore architecture and HPC-optimized CPU (TSMC 10nm FinFET) coming from the ARMv8 ISA with HPC Extension SVE. It also benefits from a 6D mesh/torus Interconnect topology and a three level hierarchical storage system (silicon disk, magnetic disk and storage for archive). Post-K will also implement a job-dedicated local file system to exploit I/O locality for acceleration and scalability[55, 56].

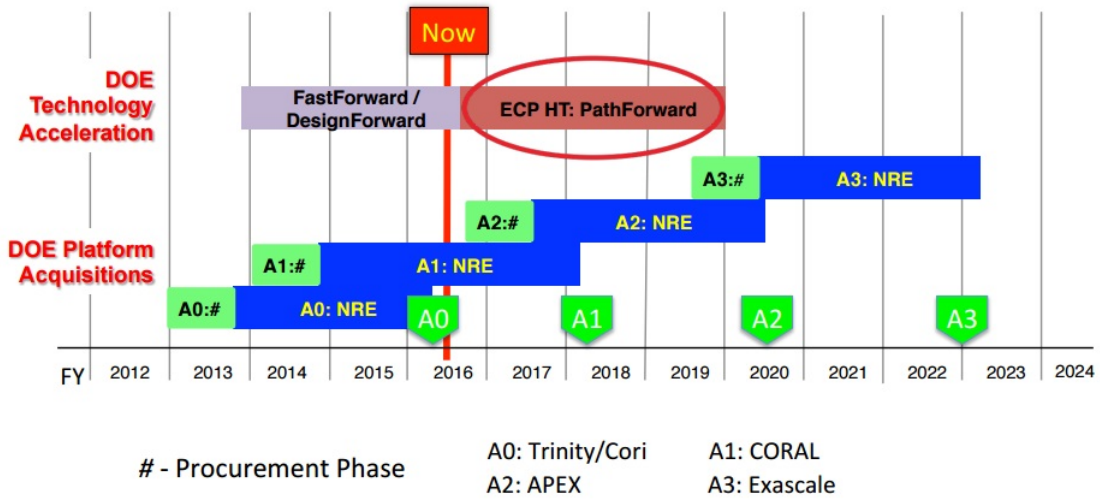


Figure 2.24: The US Exascale Computing Project Roadmap

2.4.5 American project : Exascale Computing Project (ECP)

The Information Processing Techniques Office (IPTO), which is part of the US Defense Advanced Research Projects Agency (DARPA), was the first to undertake exascale research as soon as 2007 [57]. One of the main objectives at that time was to identify the overall challenges and problems to provide a sufficient basis for the development and deployment of Exascale-class systems by 2015. The reality of these challenges and constraints have led public authorities to adjust the inital ambitions and announced that such achievement would not take place before 2023-2025 (figure 2.24) [58].

The US Exascale Computing Project (ECP), started in 2016, is the current american initiative for research, design and delivery of at leat two exascale supercomputers by 2023, which includes both software and hardware technologies. This research is funded by the Department of Energy (DoE), National Nuclear Security Administration (NNSA) and the Office of Science (SC) as a common effort to reach the objective with two projects: FastForward and DesignForward [59]. To address energy efficiency, reliability and overall performance problems of exascale computing, the following U.S. companies have been involved : Advanced Micro Devices (AMD), Cray Inc. (CRAY), Hewlett Packard Enterprise (HPE), International Business Machines (IBM), Intel Corp. (Intel) and NVIDIA Corp. (NVIDIA). The ECP has three phases (see figure 2.25). Applications, Software and Hardware developments are run in parallel with pre exascale tests between 2020 and 2023.

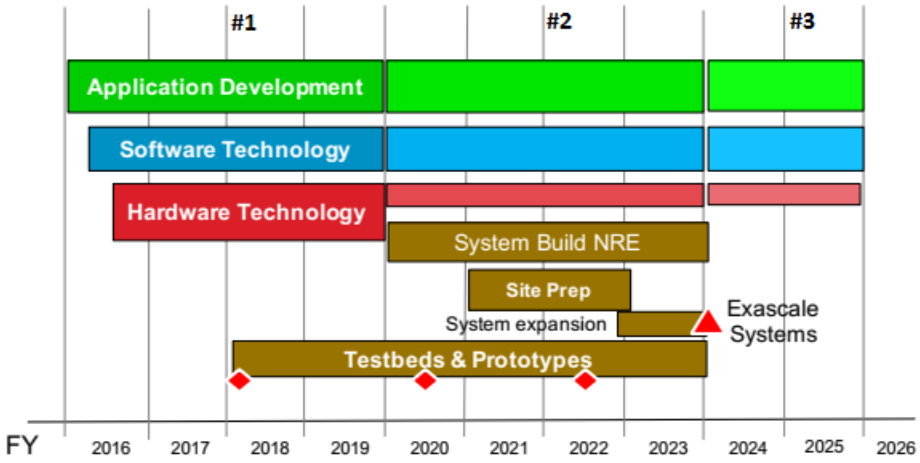


Figure 2.25: ECP Timeline

2.4.6 European Projects

European HPC programme towards exascale computing began with efforts funded by the European Commission (EC). Since 2011, the European Extrem Data & Computing Initiative (exDCI) has supported key research in areas such hardware design, programming models, algorithms, tools and applications within seven parallel projects funded by the EU's Seventh Framework Programme (FP7/2007-2013)[60]: CRESTA (Collaborative Research into Exascale Systemware, Tools and Applications), DEEP-ER (Dynamical Exascale Entry Platform – and its Extended Reach), EPiGRAM (Exascale Programming Models), NUMEXAS (Numerical Methods and Tools for Key EXAScale Computing Challenges in Engineering and Applied Sciences), EXA2CT (Exascale Algorithms and Advanced Computational Techniques) and MONT BLANC (European Approach Towards Energy Efficient High Performance).

As seen in the Introduction, the work leading this thesis was done in the framework of Mont-Blanc phase 2 project. Compared to American and Asian approaches, the European projects seem one level below in terms of funds allocated, scattered, repetitive or even contradictory. It is a direct consequence of the divergent interests between the member states, each wanting to lead its own project. Take the example of the MaX project: MAterials design at the eXascale (2015-2018) aiming to create "an ecosystem of capabilities, ambitious applications, data workflows and analysis, and user-oriented services. At the same time, MaX enables the exascale transition in the materials domain, by developing advanced programming models, novel algorithms, domain-specific libraries, in-memory data management, software/hardware co-design and technology-

transfer actions". One wonders how this project is complementary to the previous 7 projects.

2.5 Conclusion

This chapter introduces the field of High Performance Computing: the main actors, applications and technologies deployed, but also current and upcoming challenges. The field of applications covers serveral domains and disciplines, from fundamental sciences, indutrial modeling and simulation or bank analyses and financial predictions. All of these applications has in common a huge amount of data to process and relatively short time results. Since few years, has startted the race to the exascale. An exascale supercomputer will be capable to perform a "Quintillion" of instructions per second. At the global level, research and development are still far from getting there and the road to the exascale is peppered with difficulties in hardware and software. Researchers and engineers must change paradigms and explore inovative concepts.

In most scientific publications about exascale computing, "co-design" stands out as the most suitable methodology to build efficient and optimized machines. In short, co-design is a method where there are constant exchanges between two parallel developments in the software stack and the hardware architecture. The goal is to ensure a better computational efficiency, but it is clear that the problem related to energy consumption is the first factor of all the requirements. Thus, HPC could become HPCPE with the predominance of power efficiency. Manufacturers offer various multi-core or many-core based architectures, such as, the low power and cache-free manycore architecture Sunway SW26010 running RISC ISA. There is a new upcoming of HPC processor architecture: Arm. Arm has demonstrated the energy efficiency of its low power processors in the field of embedded systems. The collaboration with the japanese RIKEN institute and the HPC vendor Fujitsu to build the first Arm-based "Post K Computer" to reach exascale.

Also, we have addressed different design exploration methodologies to analyse and develop system architecture opportunities.. This chapter concluded with an overview of the reports and major exascale projects known at the global level. There are competing industrial and research projects in Asia, Europe and the United States, with different deployement deadlines, but the same purpose towards exascale computing. Exascale computing aims to boost economic competitiveness, as weel as advanced fundamental research disciplines. Next chapter introduces the modeling methodology applied based

on a combination of tools and models at different levels of abstraction: Arm fast models, the open source Gem5 framework and the Synopsys Platform Architect tool. This chapter also presents results about correlation and scalability of workloads between the modelled virtual platforms and existing Arm-based platforms, such the Arm Juno board, the Arm Seattle or the Applied Micro's Xgene1.

Part II

EXPERIMENTATION AND ANALYSIS

Chapter 3

Modeling and exploration methodology

3.1 Introduction

This chapter presents modeling methodologies, existing tools and the study of their combination in an efficient trade-off between simulation time, complexity and result accuracy. We examine first relevant state of the art tools, models and platforms to define an exploration approach matching all these needs. We then characterize a set of relevant HPC benchmarks on different platform configurations to verify that we meet all conditions for exploration effectiveness given a set of architectural requirements to consider (performance, memory architecture, interconnect, scalability).

Figure 3.1 decribes the main levels of abstraction when designing a system on chip. Indeed, Algorithmic refers to the highest level where components can be seen as functions and their interactions in the overall system. Modeling at this level usually consists of behavioural blocks described with a high level language (SystemC, UML VHDL, etc.), sometimes using schematics or graphs. Architectural studies aims to provide the chip's High level Architecture Specification (HAS). Therefore, simulating at this level requires both to remain fairly realistic (close to hardware components internal behaviours) while keeping simulation times acceptable in order to make a large campaign of configurations and exploration of architectural choices. RTL, Gate and Transistor abstraction levels allow to address logical design, physical design and the manufacturing technology process. Languages such Verilog or VHDL are commonly used for these purposes.

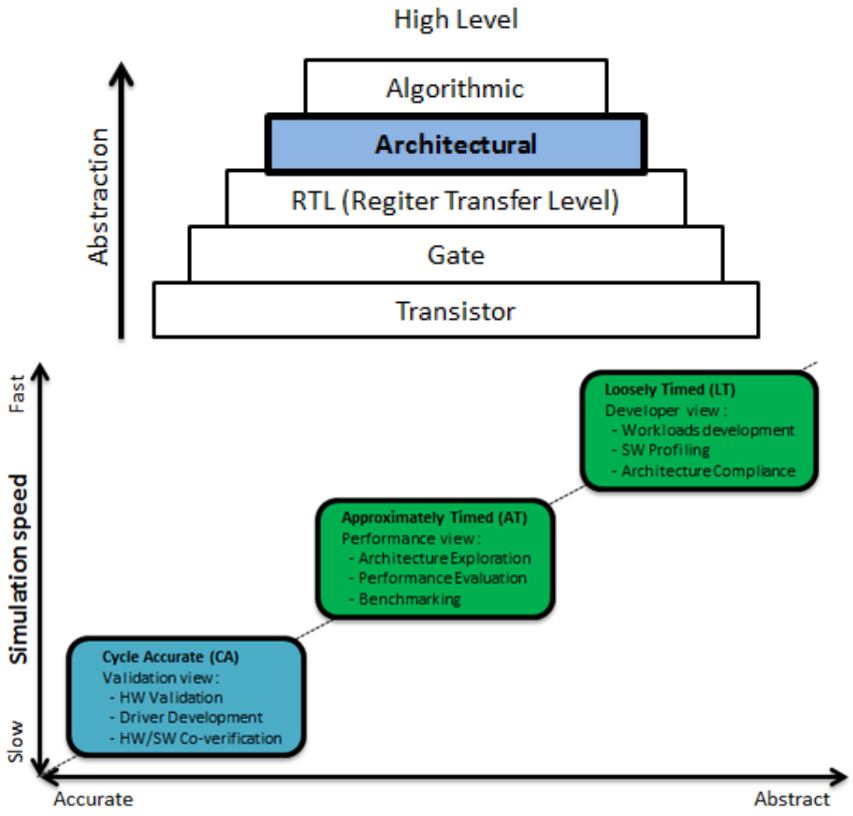


Figure 3.1: Modeling levels

3.2 Virtual prototyping and system components

3.2.1 Simulation tools

Our approach is to choose complementary tools and platforms supporting Arm 64-bit features in a way to define a global methodology for configuration and architecture exploration. The list of tools we have evaluated includes Vista (Mentor Graphics), SoC Designer (Carbon), GEM5 (academic open source), System Generator Canvas (Arm) and Platform Architect (Synopsys). In addition to performance, ergonomy and correctness, the main concerns that have been raised for selection are the ability to address multiscale analysis (possibly up to several tens of 64-bit Arm cores), support for coherency in a multicore and multi-socket system and different types of memories and interconnect.

3.2.1.1 Vista

Vista is a TLM 2.0 simulation framework from Mentor Graphics for architectural design exploration and virtual prototyping. It provides early abstract functional models of

complex hardware of full system architectures with complex bus architectures and multi-core communications models. The virtual platforms can be used beneficially to develop and debug embedded software (e.g. driver and firmware) possibly under a complete OS control, before real hardware is available. An example is illustrated in figure 3.2 on a realistic dual core system including memory and communication system but also common peripherals (sdcard, usb, ethernet) under the control of a fully functional Linux kernel. Such level of modeling and execution detail here is justified by the need to evaluate real code intended for embedded platforms. For example, simulation of the architectural model of figure 3.2 takes a few minutes to boot linux. Hower, the limitation is the fact that system architeture is Armv7 32-bit.

3.2.1.2 SoC Designer

SoC Designer, initially developped by Arm and owned now by Carbon Design Systems, is a tool for the development of SystemC simulation platforms allowing architecture profiling and software validation in parallel with hardware development. It is based on a set of cycle accurate and wide range of fast models of Arm IPs that can be easily configured and assembled to model a fully functional system architecture and perform a detailed analysis of related hardware and software. An interesting aspect of this framework is that the models are generated directly from Arm register transfer level (RTL) code ensuring high model relevance. The figure 3.3 shows the reference platform package used for evaluation. It is composed of two Cortex-A57 clusters coherently connected through the Cache Coherent Interconnect CCI-400 and supports the full Arm ACE coherence protocol. The system includes a ROM model, implementing the firmware stack to boot the platform, and the Generic Interrupt Controller GIC-400. Like Vista previously, the resulting cycle-accurate virtual prototypes provide a way to develop and validate software before committing to physical hardware implementations. Both Vista and SoC Designer based platforms will be therefore less effective outside the scope of embedded systems, in particular to address significantly bigger designs such as those expected in HPC architectures reaching the order of several tens of cores. It can therefore be used to investigate locally what happens at cluster or compute node level, including power consumption, but for the purpose of our study we will need also to address simulation frameworks enabling modeling capabilities at a higher level of abstraction than TLM/RTL.

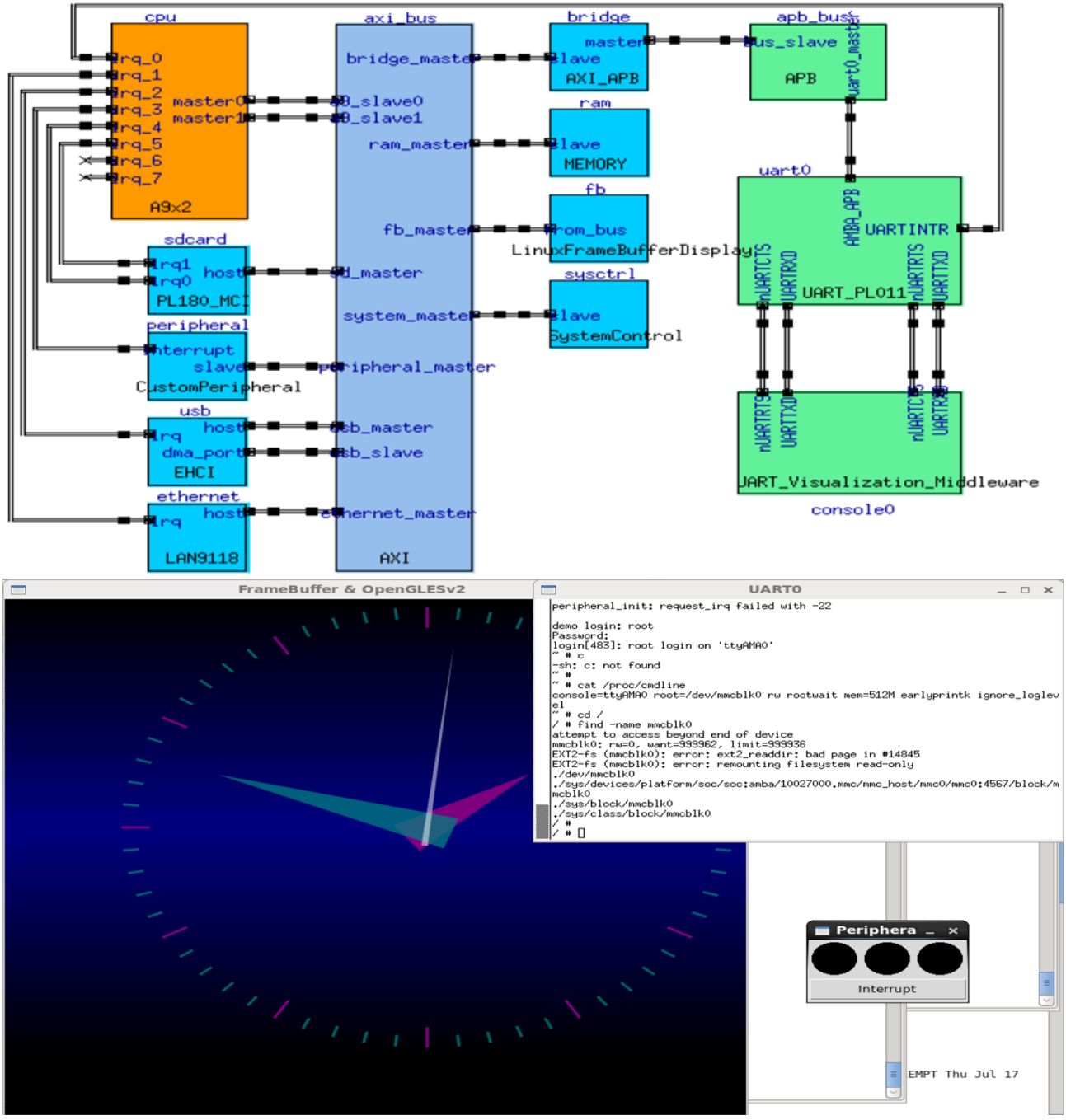


Figure 3.2: VISTA modelled platform and output view

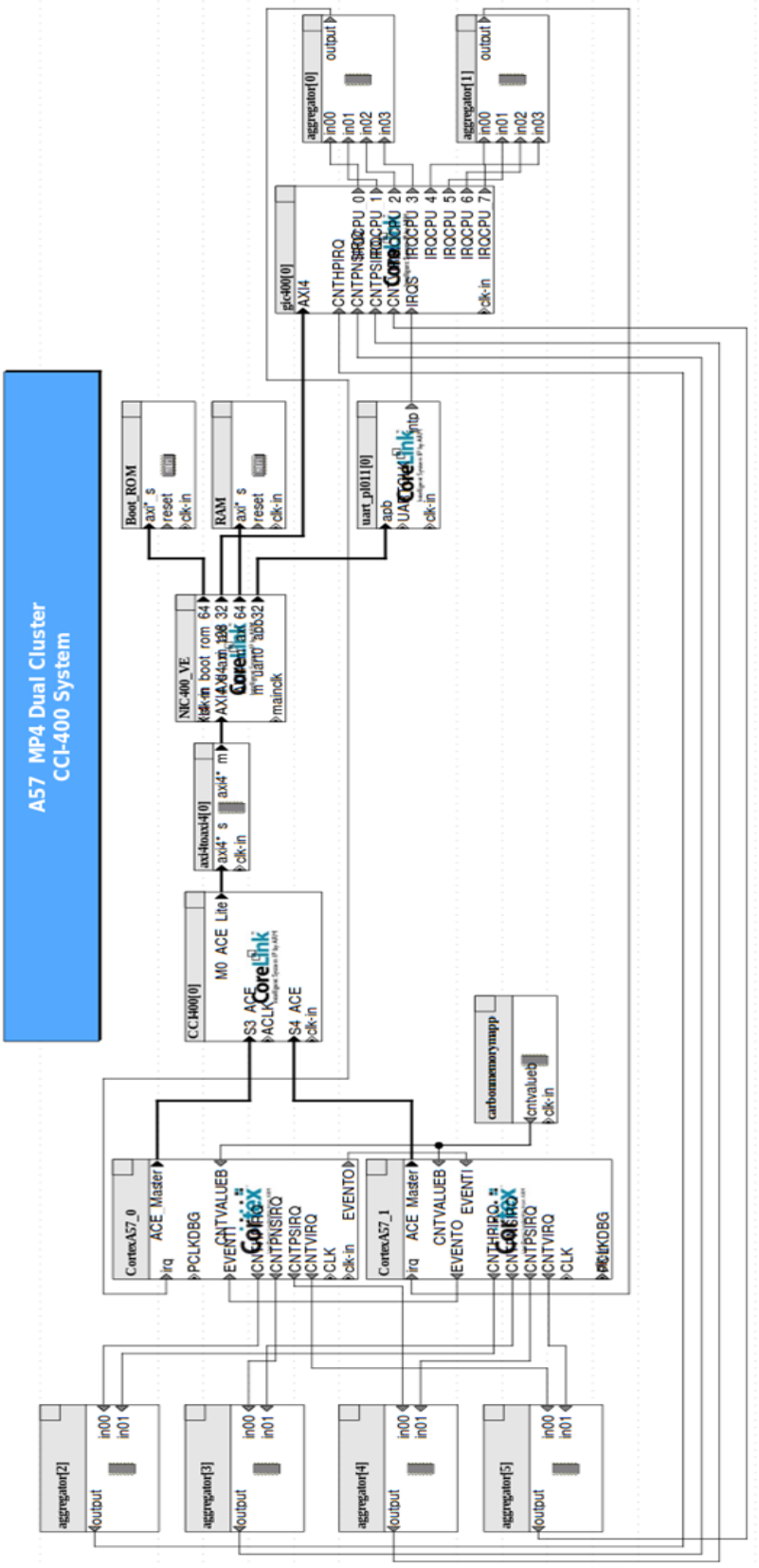


Figure 3.3: Reference platform for SoC Designer evaluation

3.2.1.3 GEM5

Gem5is an academic open source system modeling framework written in C++ and python. The default components designed with this environment are delivered under a BSD license (permissive free software license). It is possible to add new capabilities according to the users needs within the limits of interoperability between existing components in the tool. Actually, the Gem5 simulator is a "modular discrete event driven computer system simulator", which means that components can be rearranged, parameterized, extended or replaced easily to suit our needs, and time is considered as a serie of discrete events.

Gem5 is intended for the simulation of one or more computer systems in various ways. It provides interchangeable CPU models with different levels of detail (one-CPI CPU, in-order CPU and out-of-order CPU) but it is also possible to add a custom CPU model. All CPU models use the same high-level Instruction Set Architecture (ISA). CPUs can be further abstracted by the use of traffic generators, either based on statistical behaviours or traces. It also supports multi-system components (CPU, crossbars, caches, etc.) which make easier the creation of complete Arm based SoC platforms. SystemC/TLM co-simulation can be profitably used by including Gem5 as an event kernel SystemC thread. For all of these reasons, Gem5 a good match for architectural analysis at different levels. Multi-scale simulations are carried out with the following modes:

- Full system mode: a complete system including devices, operating system (linaro or Ubuntu) and file system can be configured to simulate the execution of applications (benchmarks). In theory, Gem5 supports up to 8 64-bit Arm generic cores in this mode, but only four actually boot in practice. This limitation comes from the Generic Interrupt Controller (GIC) modeland should be fixed in the coming months.

- Syscall emulation mode: binaries are directly executed on the platform created. No Operating System is needed, the simulator directly provides a small set of operating-system-like services (Syscalls). Simulation is faster but only for simple programs. Full HPC applications can not be processed in this mode because library dependencies require system calls that are unsupported in this mode.

The medium scale configuration of figure 3.4 is considered to address a relevant HPC topology. At the time of this modeling evaluation, it was the largest full system platform we could create because of the GIC model limitation. Software stack from the gem5

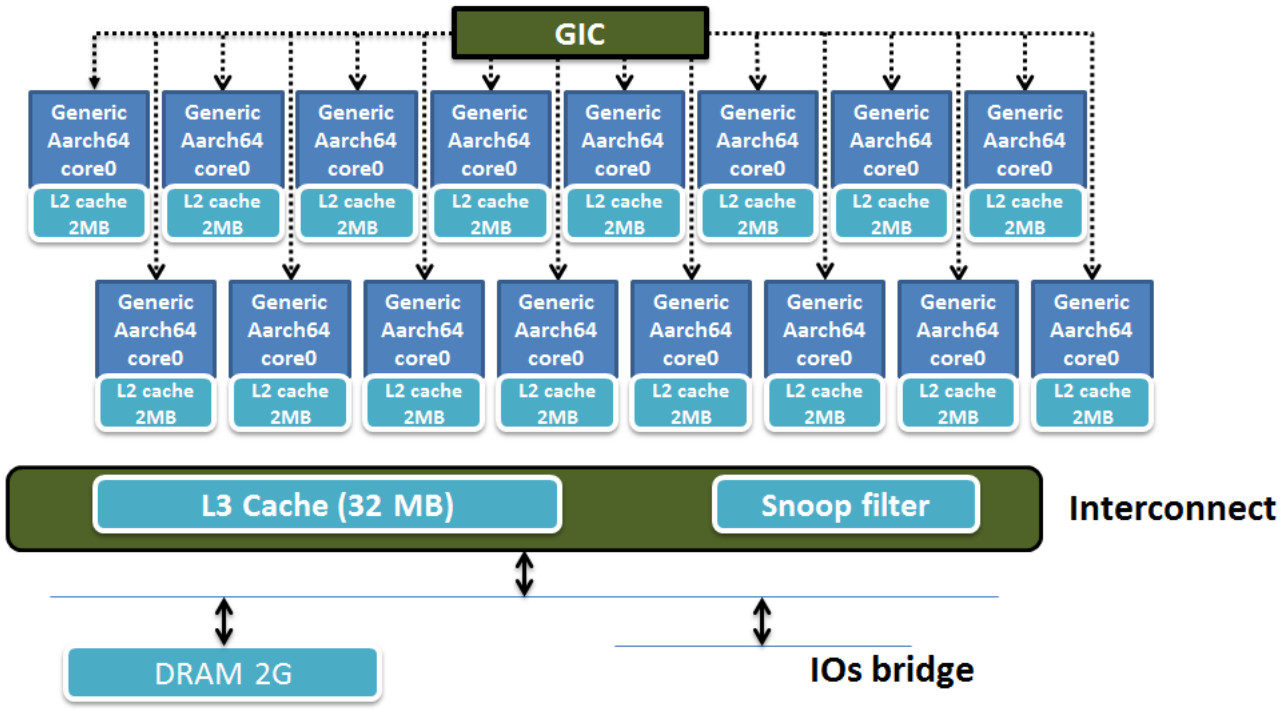


Figure 3.4: Gem5 Medium scale platform

community can then be used to build a linaro kernel supporting the resulting system architecture.

3.2.1.4 System Generator Canvas

This part of the evaluation focuses on Arm models, profiling and debugging tools. System Generator Canvas is used to model virtual platforms with Arm Fast Model IPs (AFM). AFM are described using LISA (Language Instruction Set Architecture) and can be used to define accurate prototypes supporting simulation at instruction level, also making use of joint hardware and software co-development to reduce the final integration time. AFM are loosely timed models which leads to very fast simulation times but the results cannot be used for performance prediction without additional timing annotations.

In an effort to continuously improve the accuracy of these models, Arm roadmaps plan to add further timing annotations to the components (CPU, caches, interconnect, memory) by setting latencies or CPI (Cycle Per Instruction) derived from real workload execution on an existing board. This allows to improve greatly performance predictions for

system architectural studies. There are two main possibilities when using these models: the first one is to generate an Arm Fixed virtual Platform (AFP) which can only be used within Arm toolchains, and the second is to export the platform as a black-block subsystem compliant with SystemC/TLM-2.0. The resulting methodology allows creating mixed platforms combining LISA/AMBA, SystemC/TLM models and compliant interfaces, making it easier to add custom SystemC/TLM components. To fully evaluate this framework, it has been decided to address all Arm tools relying on Fast Models, namely DS-5 for debugging and Arm Compiler 6 supporting optimized code generation. Considering this, the characterisation objectives are to build an AFM version of an Arm Juno-r1 board (in the subsection 3.2.2), study the correlation between the real platform and models, build the largest system configuration based on Arm IPs (Cortex-A57, Cortex-A72, CCN508, CCN512, GICv3), address software development and simulation, debug and profile HPC benchmarks with DS-5 and compare Arm Compiler 6 and GCC in terms code optimization ability. The topology set with this platform is illustrated in figure 3.5.

3.2.1.5 Platform Architect

Platform Architect with Muticore Optimization (MCO) is a SystemC TLM methodology from Synopsys. It is essentially a graphical environment which allows to easily capture, configure, simulate and analyze system-level performance and power consumption of multicore Arm based SoC platforms by means of virtual prototypes. The development of generic virtual platforms benefits from various libraries of SystemC TLM models to be conveniently used for architectural explorations, including traffic generators, interconnect, memory subsystem and processor models. Task modeling is based on application task-mapping and task-driven traffic generation using a Generic File Reader Bus Master (GFRBM) and Virtual Processing Unit (VPU). The VPU minics the real processor behavior as far as performing memory transactions and consuming concerned processing cycles. The memory traffic is bidirectional, each VPU can issue cacheable requests and respond to snooping transactions coming from other VPUs. The task graph consists of a number of synchronized threads, each thread gets mapped on one VPU and the coherency protocol maintains the consistency of the shared data. This methodology helps to focus on e the global system interconnect and memory architecture without executing the real firmware/software stack. The MCO methodology takes advantage of previous modeling and traffic generation features to further abstract and speedup SoC architecture simulations. The analysis takes place in two steps: task-graph generation and optimization using trace-driven traffic generation.

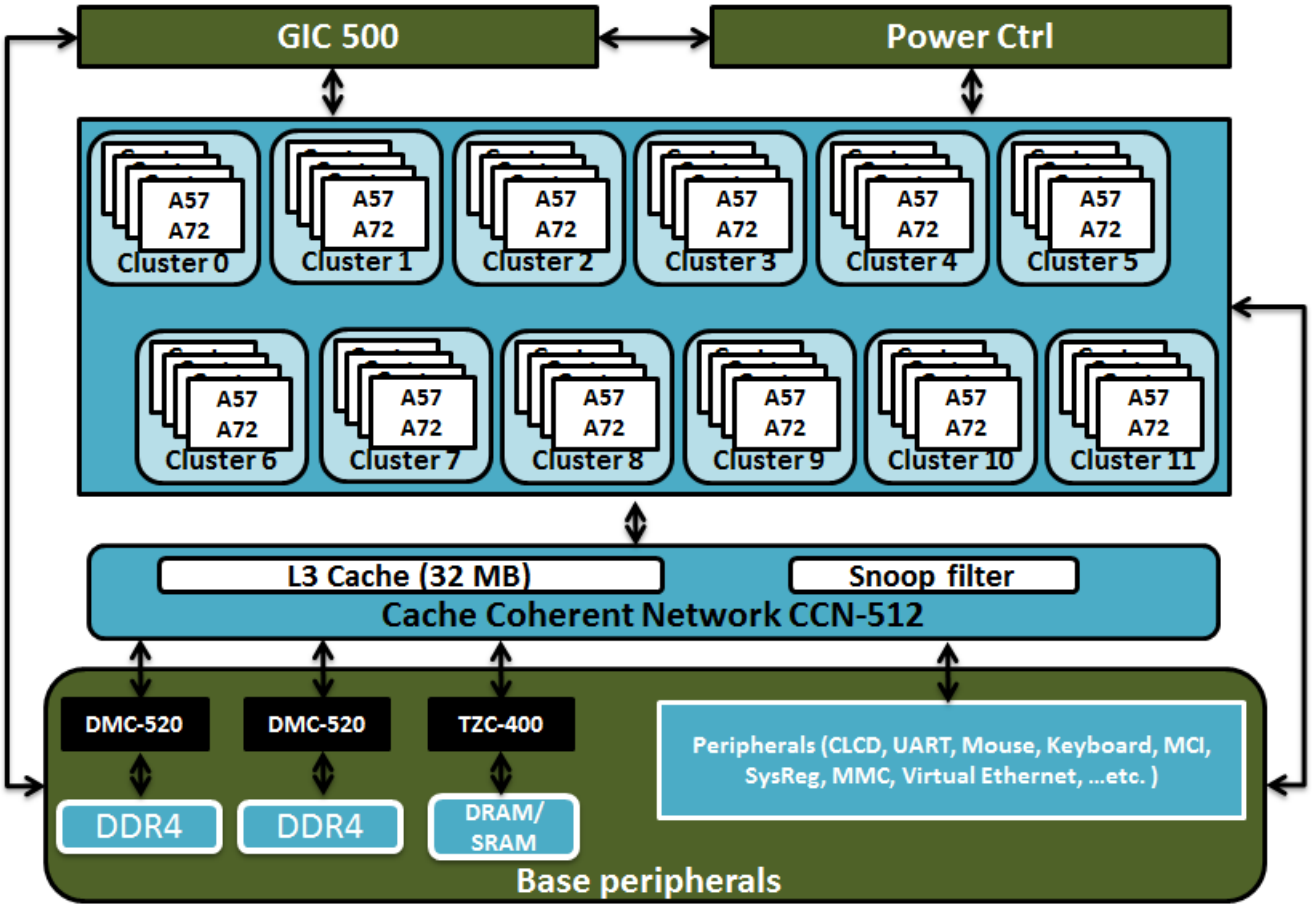


Figure 3.5: 48 AFM Cores block diagram

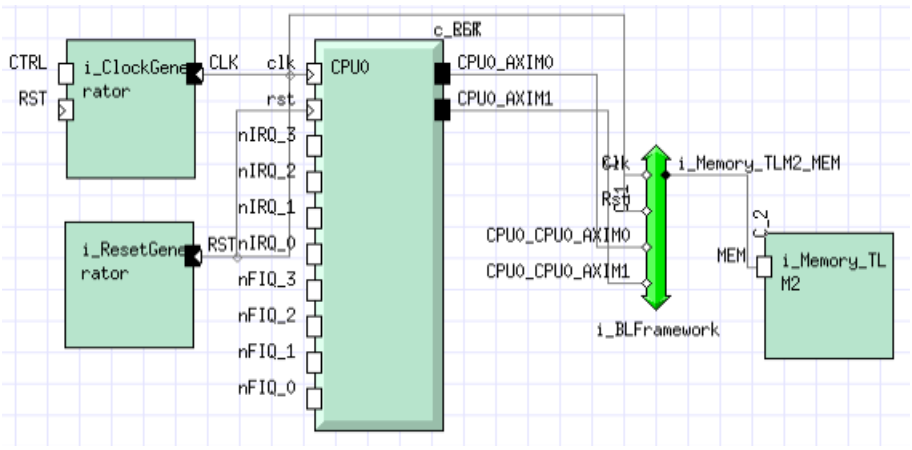


Figure 3.6: Platform example based on Cortex-A57 Fast Model

The example of figure 3.6 illustrates the configuration of a virtual platform based on Cortex-A57 Fast Models which will be used to create a SystemC performance model of the application (task-graph) and bridges to translate AMBAPV transactions from Cortex-A57 in TLM2 to interact with the rest of the system. We also have a simple memory model, a TLM bus, and clock and reset generators. The objective here is to generate a trace database from Dhrystone execution on the CPU. The simulation is done in frontdoor mode, meaning that the CPU really uses the bus transactions for requests to the memory and all traces are saved. This is heavy for disk usage.

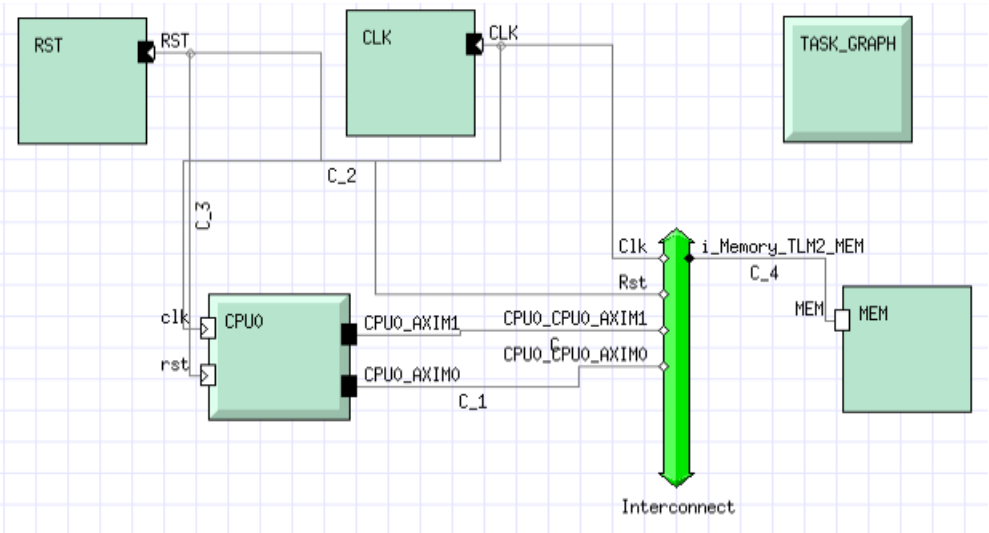


Figure 3.7: Equivalent VPU based platform

Figure 3.7 is the corresponding VPU platform used for interconnect and memory subsystem performance optimization based on previous trace-driven traffic generation. The main difference is that this platform is now based on a VPU to mimic the Cortex-A57 fast model behavior. This is useful when it comes to address large scale exploration involving complex high speed interconnects that might not be compatible with fast models. A VPU is different from a traffic generator, but behaves like a real CPU without the need of a full software layer (BIOS, bootloaders, OS, libraries). However it is able to react to the traffic on the interconnect and respond to snoop requests, making it possible this way to analyse data coherence issues properly. The following describes the global approach on the Dhrystone benchmark. Figure 3.8 represents the corresponding task graph wich is derived from simulation results of an AFM platform.

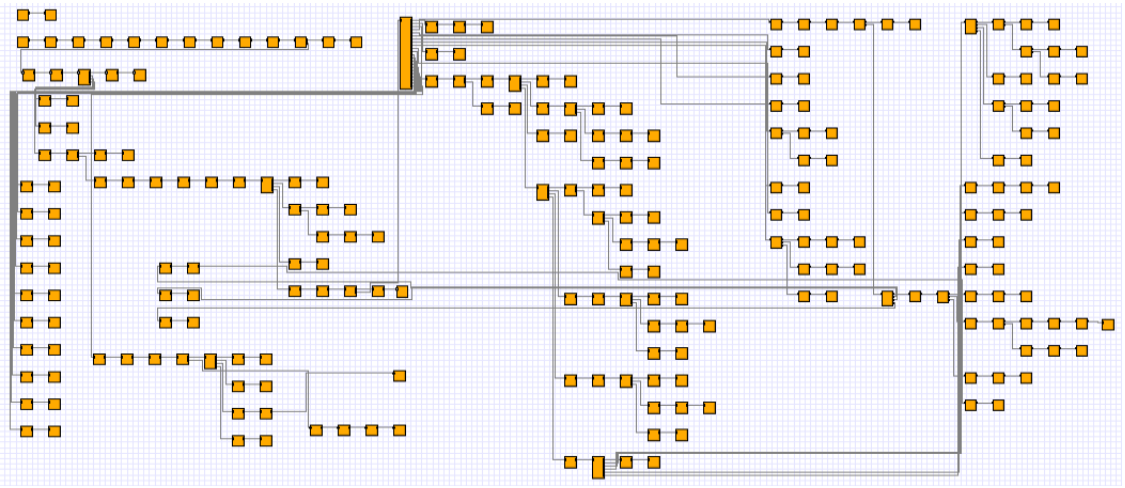


Figure 3.8: Dhrystone task graph

A set of parameters can be used to filter the task graph in terms of minimum number of instructions, access types and function context in the workload. This allows a better view of the execution profile and dependencies among functions, which will be useful later to derive a task graph and analyse application parallelism and scalability. Figure 3.9 shows the distribution of dhrystone benchmark amoung available VPU cores, cache hit and miss, and the number of snoop trancations received and responses emmited.

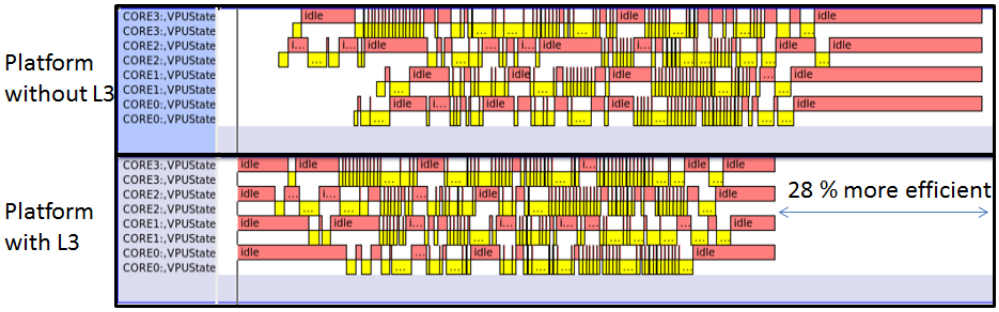


Figure 3.10: Task graphs results With vs Without L3 cache

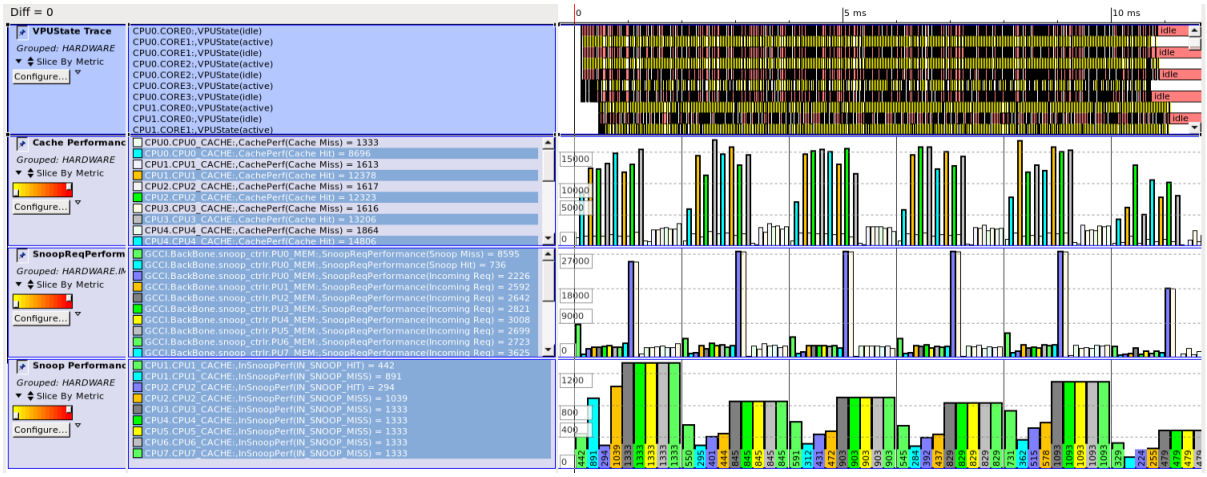


Figure 3.9: Simulation outputs: Fast models vs. VPU (Dhrystone benchmark)

For instance figure 3.10 shows that adding the level 3 cache within the on-Chip coherent interconnect, leads to over 28% of execution time reduction.

3.2.2 Hardware platforms

In this section, we describe a set of real ARMv8 based development boards available at the beginning of this work. These platforms were used, as a preliminary step of this work, to verify the relevance of the different virtual prototyping and simulation tools addressed in subsection 3.2.1. The Juno Arm Development Platform is one of the first available platforms featuring the ARMv8-A performance and low power ISA with clusters of two Cortex-A57 and four Cortex-A53 cores. This board is particularly useful to deal with the performance cluster (dual Cortex-A57 of the MPCore big.LITTLE processor), Cache Coherent Interconnect (CCI-400) and DDR3-1600 dual channel memory controller, and check the accuracy of previous models (figure 3.11).

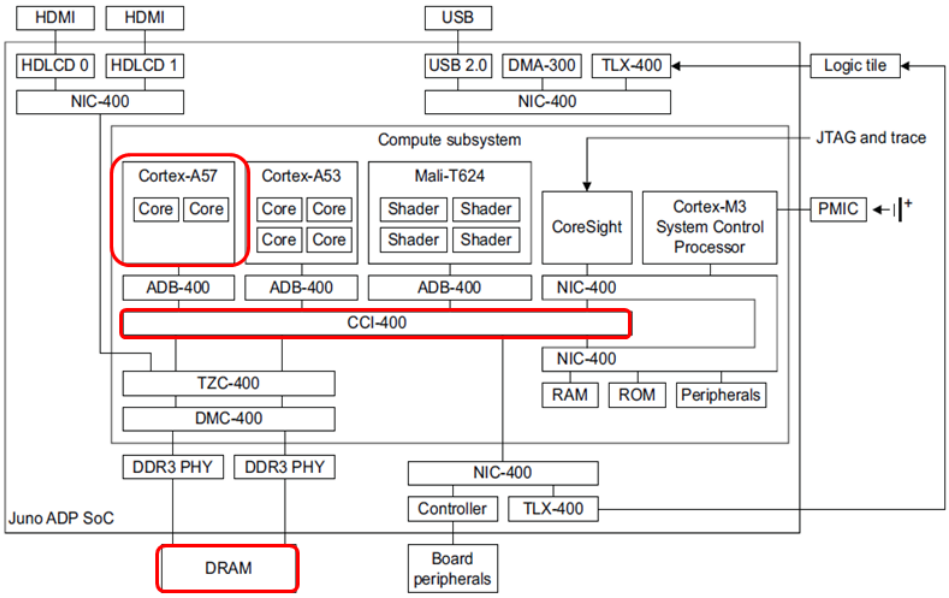


Figure 3.11: Juno board diagram

Applied-Micro (APM) is also among the first to propose a new generation of platforms based on ARMv8 processors. It features eight cores runnning at 2.4GHz, 8MB L3 cache, and 4 DDR memory channels manufactured in 40nm TSMC process. APM X-Gene1 is made of a SoC including four clusters of two Arm 64-bit cores running at 2.4 GHz, the APM coherent network Interconnect and a DDR3 controller supporting 16GB (figure 3.12). In our characterisation effort of real hardware, we also addressed the use of an AMD Seattle board based on four clusters of two Cortex-A57 cores with AMD Coherent Interconnect at 2 GHz and two DDR4-3733 memory controllers (figure 3.13). This chip is designed at the 28nm process level.

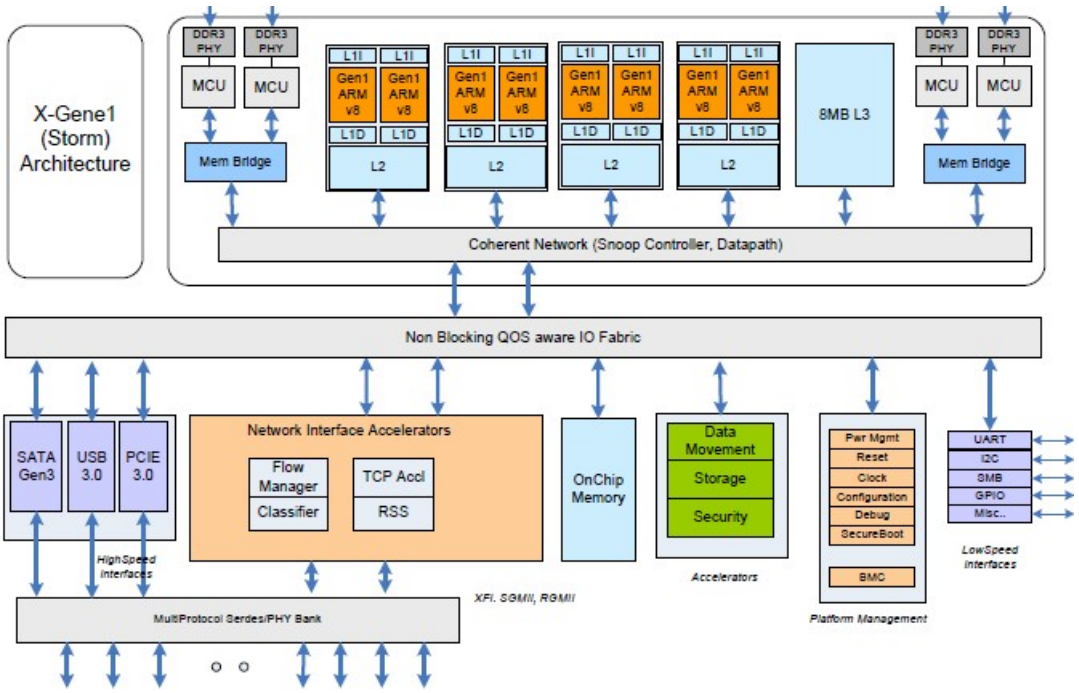


Figure 3.12: APM first ARMv8 processor : Xgene1 Storm

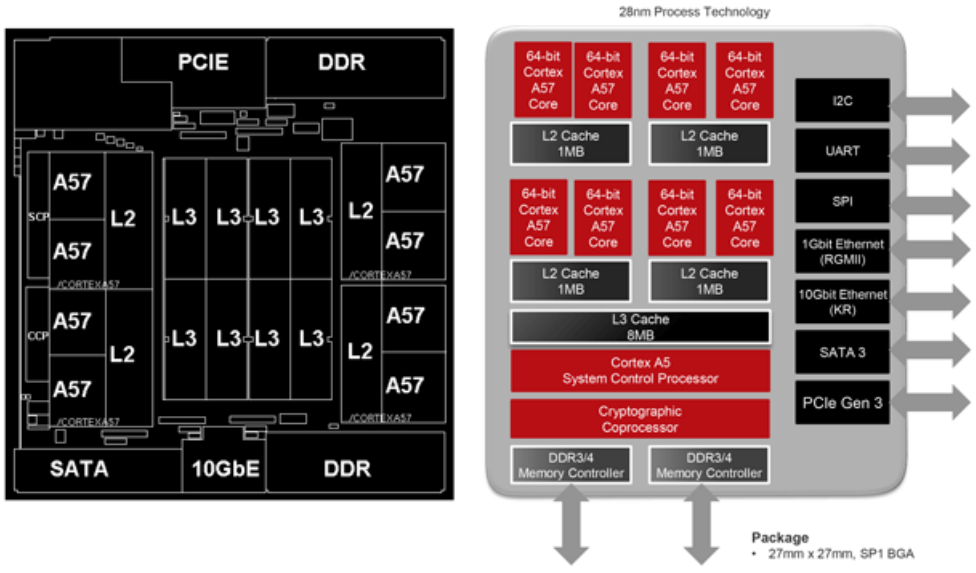


Figure 3.13: AMD Seattle's architecture : Floorplan and block diagram

3.2.3 programming support for HPC

To analyse performances carefully, hardware is not the only consideration. HPC applications rely heavily on specific libraries to further build on specific processing features. This subsection introduces the set of features that are used essentially to optimize software performances.

3.2.3.1 Libraries

Automatically Tuned Linear Algebra Software(ATLAS) is an open-source project that aims to provide a set of commonlly used HPC mathematical calculations. These libraries can be automatically tuned for a specific hardware platform to ensure higher performances. HPC workloads generally process millions of callbacks of mathematical functions. Accumulating small gains, but on a very large number of function calls, can potentially lead to very important execution speedups. Arm Performance Libraries (APL) are optimized ATLAS libraries tuned by Arm software developpers for ARMv8 high-performance processors. APL are based on BLAS, LAPACK (Linear Algebra PACKage) and FFT (Fast Fourier Transform) routines with OpenMP parallelization directives [61]. The DGEMM benchmark exposes typical floating point processing that can be exploited with this library. First evaluations on this benchmark report for example 12% performance improvement compared to the original open-source version of OpenBLAS. APL is used systematically to optimize floating point processing in the following.

3.2.3.2 Programming models

OpenMP (Open Multi-Processing) is an API (Application Programming Interface) for parallel programing at compute node level. It provides a set of compiler directives, library routines and environment variables to improve parallelism in different programming languages such as Fortran or C/C++. It was used in different simulations to derive multithread implementations while taking care of accesses to shared memory data. OpenMPI is an open-source project of the standard Message Passing Interface (MPI) implementation. The definition of this message passing based library results from a consensus made by the MPI standardization forum. OpenMPI is developed and maintained by a consortium of academic, research, and industry partners. In fact, contrary to OpenMP, MPI works at cluster level and allows to manage communications between compute nodes. OmpSs (OpenMP StarSs) is an effort developped by the Barcelona Supercomputing Center (BSC) to integrate features from the StarSs programming model

they developped into a standard programming model (OpenMP). Specifically, it aims at extending OpenMP with support for asynchronous parallelism and heterogeneity. As a result, OmpSs is tailored to address GPU acceleration and should not have a particular benefit for CPU based compute nodes. Its directives extend OpenMP for acceleration based APIs like CUDA or OpenCL. POSIX (Portable Operating System Interface) thread (Pthread) is the native parallel execution model that can be found in Linux distributions [62]. POSIX threads is a lower level API for working with threads offering fine-grained threading-specific code to permit control over threading operations. Unlike OpenMP, the use of Pthreads requires explicit parallelism expression in the source code (e.g. hard-coded number of threads).

3.3 Exploration methodology

3.3.1 Definition and metrics

An important element in the exploration methodology is to evaluate different possible design alternatives on processing efficiency. We consider two metrics to evaluate the processing efficiency. The first one is based on floating point operations per second (GFLOPS) which is reflective of the processing power for HPC workloads, and the second is the FLOPS efficiency expressing the ratio of actual versus theoretical FLOPS supported by the system, and can be calculated as follows:
$PerformanceEfficiency(\%) = \frac{ObtainedPerformance}{TheoreaticalPerformance} = \frac{MeasuredPerformance}{NumberOfCores*Frequency*NumberOps}$

Another key point is to extend previous approach to allow the robust analysis of memory hierarchy, performance (execution time and throughput) and scalability (considering the possible impacts of cache). Given previous outcome, these metrics and more especially the cache statistics only relates to the cluster level (L1 and L2 cache) and more importantly with the L1 cache which has the most performance impact typically have a hit rate above 95% in real world applications. The ability of virtual platforms to produce reliable memory / interconnect configurations and related transaction information represents a fundamental aspect of HPC architecture design analysis.

Finally, since the target platform is designed to take advantage of large ARMv8-A clusters, communication topology and memory system are key issues to address, a final important aspect is to extend the analysis at a larger scale. We aim to support simulations up to 128 cores four our specific exploration needs. This can greatly benefit here from the use of VPUs introduced in subsection 3.2.1.5. Figure 3.14 illustrates the main difference between the VPU and the commonly used traffic generator. In fact, a VPU acts like a real processor in terms of outgoing and incoming traffic. Traffic generators are known te be one way issuing, the traffic is always from top to botom. In that case, it is not possible to implement any coherence method between virtual masters as they can not communicate between each other. By opposition, VPUs rely on a task graph featuring the load/store and read/miss ratio of each task. The traffic inside the on-chip interconnect is characterized in a realistic way.

Our modeling methodology is based on the use of Arm Fast Models and Synopsys Platform Architect. Arm Fast Model based platforms are used to run real software and then generate application profile traces for function execution and memory accesses. From these, we generate a performance workload model (also referred to as task graphs)

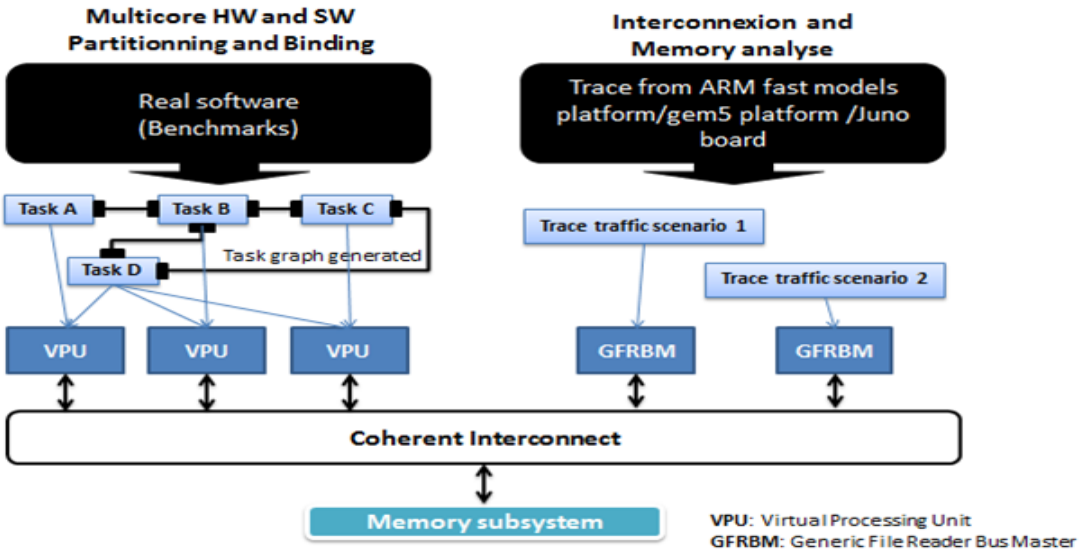
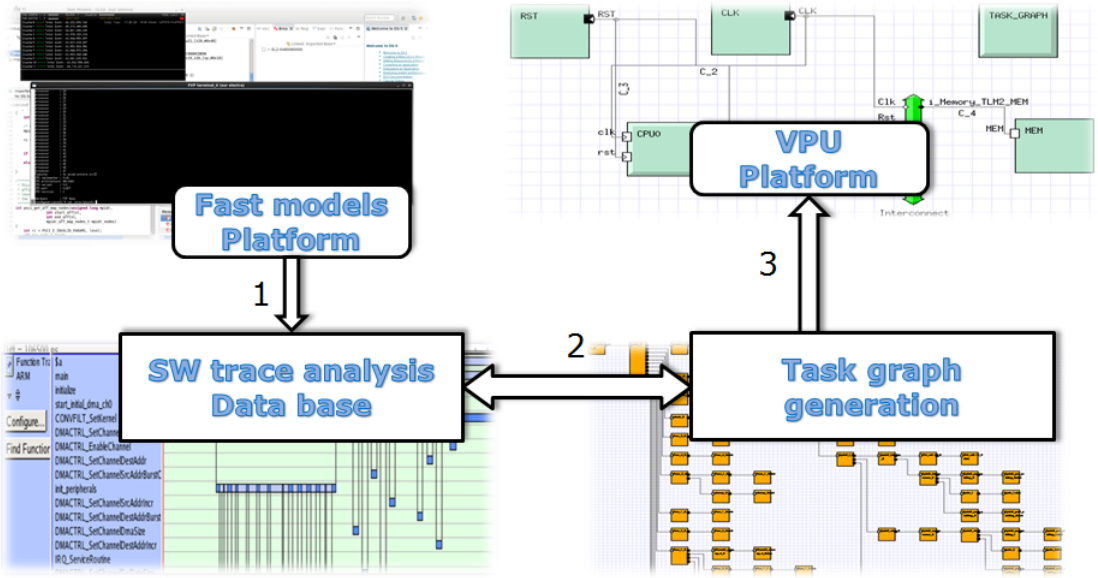


Figure 3.14: VPU vs Traffic generator

which gets mapped on processing resources (also knowns as Virtual Processing Units or VPUs) inside an architectural model including also contains a SoC cache Coherent Interconnect (SCI) and memory subsystem. Figure 3.15 summarises our performance analysis flow. First of all, we run real applications on a running a Linux kernel on an Arm Fast Models based Virtual Platform. Then, from the executed software functions and memory access traces, task graphs are generated using Synopsys graph generator tool. The task graph obtained is used to perform realistic simulation traces in our large scale Virtual Processing Unit (VPU) platform in Platform Architect (SystemC/TLM models). Figure 3.15 summarizes the overview of our exploration methodology and modelling flow. These considerations lay the foundation for proper investigation of architecture capabilities which are explored in the following in terms of processing efficiency, memory hierarchy, interconnect, topology and scalability.

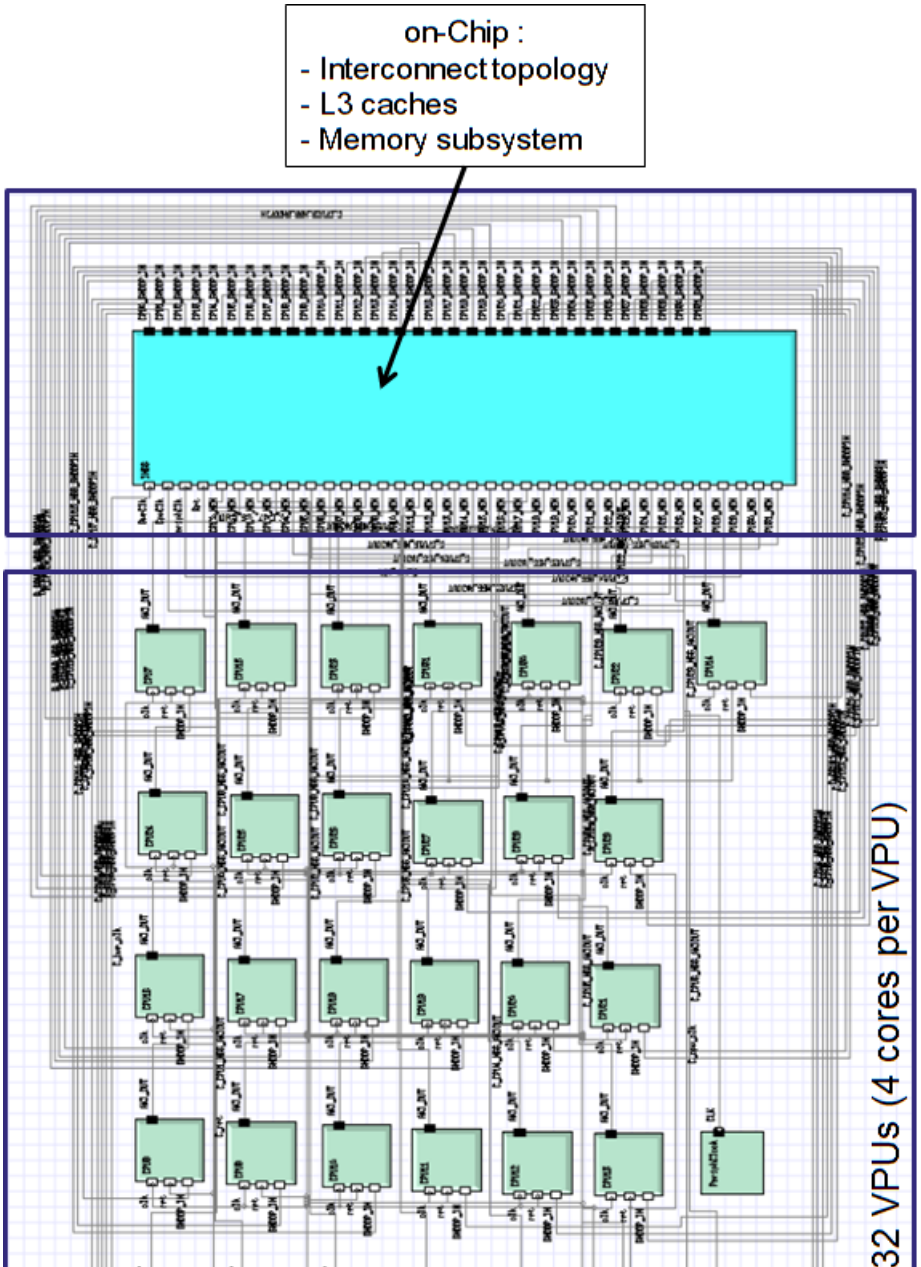
3.3.2 Extended VPU platform

Thus released from software constraints, our goal now is to build a platform (figure 3.16) with a coherent interconnect, supporting tens of processors (up to 128 cores or more), efficient memory controllers, and analyze the scalability of the system. Figure 3.16 provides a high level view of our 128 cores VPU platform. We have different types of VPU (equivalent of a CPU), PCIE, SATA, etc. The IMSS block contains the coherent interconnect topology and the memory subsystem including memory controllers.



T

Figure 3.15: Global Modeling Methodology



3.3.3 Correlation study

3.3.3.1 Evaluation of virtual platforms

Correlation in the following refers to the analysis of the simulation or execution results produced by at least two types of systems: one real board and its equivalent as a virtual platform. Actually, both systems come with identical configurations in terms of architecture, frequency, memory bandwidth, type of memory controller, interconnect (figure 3.17). With the help of the different tools and platforms exposed previously, we start by configuring virtual platforms in similar configurations to Arm Juno and AMD Seattle boards in a way to compare against fast models for the set of architectural parameters previously exposed. Task graphs are then derived from fast models execution traces, which are further used to simulate equivalent VPU platforms. Perf, a lightweight profiling tool for Linux, is used to get the execution profile and average CPI of the same test programs executed on the real boards and finally compare the results.

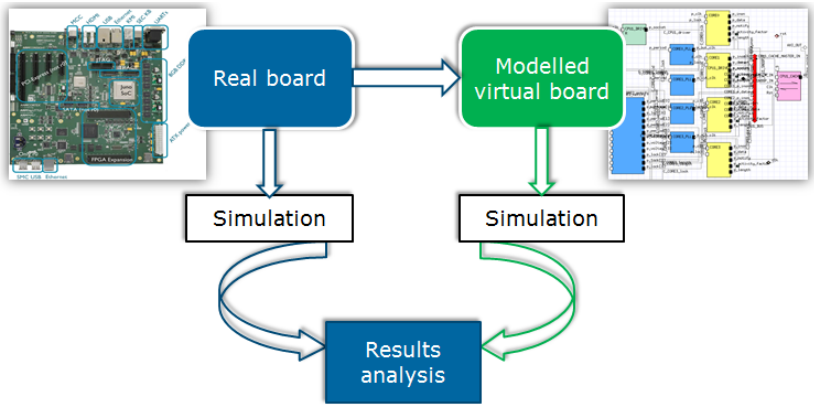
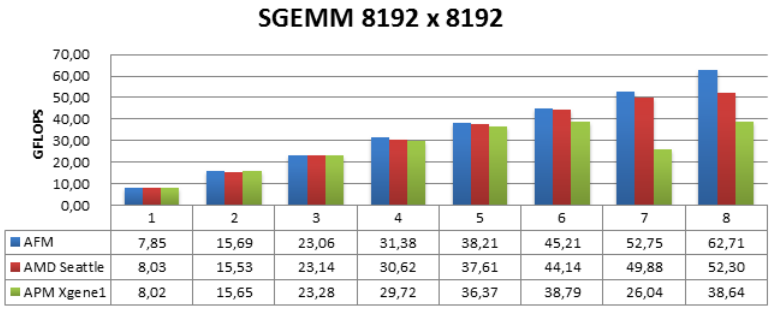


Figure 3.17: Principle of correlation study

3.3.3.2 Processing efficiency

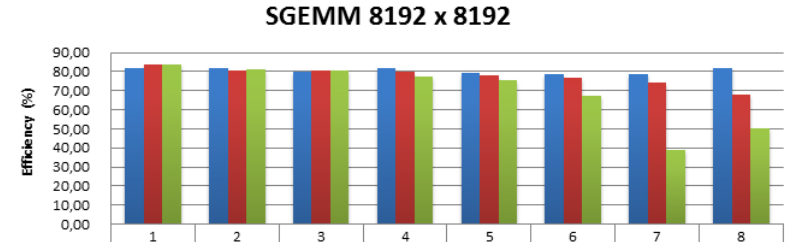
Figures 3.18 and 3.19 show examples of correlation study for three platforms:

- AFM (Arm Fast Models based virtual platform) : 2 clusters of 4 Cortex-A57 cores, with CCN512 Interconnect
- AMD Seattle board : 4 clusters of 2 Cortex-A57 cores, with AMD coherent Interconnect
- APM Xgene1 board : 4 Clusters of 2 ARMv8 cores, AppliedMicro (APM) coherent Interconnect



	1	2	3	4	5	6	7	8
AFM	7,85	15,69	23,06	31,38	38,21	45,21	52,75	62,71
AMD Seattle	8,03	15,53	23,14	30,62	37,61	44,14	49,88	52,30
APM Xgene1	8,02	15,65	23,28	29,72	36,37	38,79	26,04	38,64

Figure 3.18: SGEMM GFlops Scalability



	1	2	3	4	5	6	7	8
AFM	81,77	81,72	80,07	81,72	79,60	78,49	78,50	81,65
AMD Seattle	83,65	80,89	80,35	79,74	78,35	76,63	74,23	68,10
APM Xgene1	83,54	81,51	80,83	77,40	75,77	67,34	38,75	50,31

Figure 3.19: SGEMM Performance efficiency correlation

We consider for this analysis the same SGEMM (Single precision General Matrix Multiply) benchmark, optimized with the latest Arm Performance libraries for optimized basic linear algebra subprograms (BLAS) and using POSIX threads (Pthreads) for multithreading. All of these platforms operate at 2400 MHz with an average CPI of 1.35 obtained on Juno and then applied to AFM platform.

The SGEMM 8192 x8192 benchmark allocates 1500 MB matrix values in memory. We observe that the scalability of this benchmark evolves with the number of used codes (from 1 to 8). Each thread of the workload is assigned to a specific core and they exchange transactions to each others through the local distributed memory. The AMD platform is the one with hardware characteristics are closest to the AFM platform and one can see that their results are similar with a small margin error.

The traffic on the interconnect increases with the number of running threads. This affects the performance of the workload because the coherent protocol implementation can cause additional latencies. Here we have three systems with different interconnect, the CCN512 of AFM platform seems to be more robust to distributed memory require-

ments of this application. Its efficiency remains high (81,65%) with 8 threads, while that of AMD is 68,10% and that of APM is of 50,31% (figure 3.19).

3.3.3.3 Memory and cache consistency

Memory hierarchy strongly affects performances in large computer architecture design. A careful analysis is essential at this level to make a reliable architectural decisions. Figure 3.20 therefore reports the ability of virtual platforms (Arm Fast Models) to capture cache level statistics compared to the real hardware. The metrics considered here are cache hit/miss statistics (here L1, but also possibly L2) and data rate performance (throughput) on the STREAM benchmark.

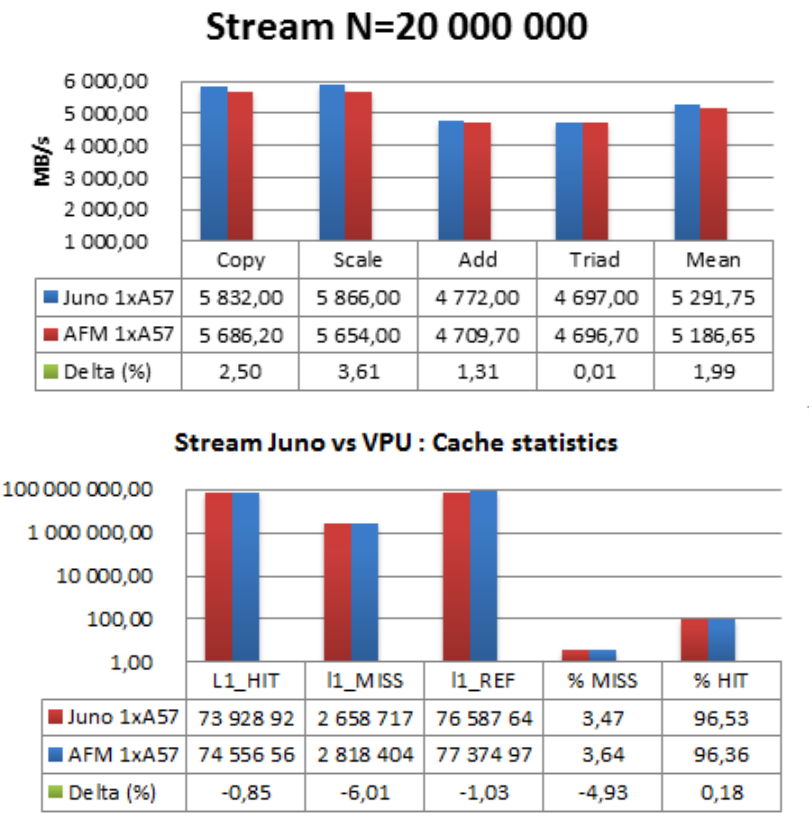


Figure 3.20: Arm Juno vs. AFM memory level analysis

3.3.3.4 Scalability

As we address large systems that can possibly consist of hundreds of cores, scalability i.e. the ability to process efficiently an increasing amount of threads, is a very important concern. This question is first discussed on the SGEMM benchmark example configured

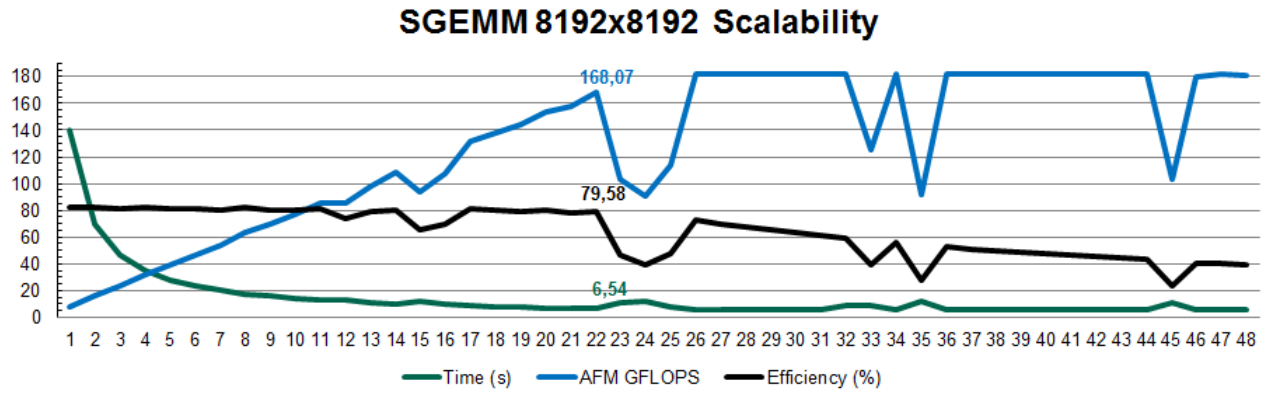


Figure 3.21: From 1 to 48 threads scalability

with up to 8 threads. The platforms used for this correlation study are based on Arm Fast Models (two clusters of four Cortex-A57 cores with CCN512 Interconnect), an AMD Seattle board (four clusters of two Cortex-A57 cores with AMD coherent interconnect) and an APM Xgene1 board (four clusters of two ARMv8 cores with APM coherent Interconnect). We can examine in figure 3.21 the ability of the AFM platform to ensure a continuous increase of performances. We additionnaly inspect the efficiency and traffic generated on the interconnect which are important matters to verify the ability of the system to scale up with larger workload memory space utilization.

Due to CCN-512 limitations, we target configurations up to 48 cores in the following simulations. Figure 6 reports performance measurements in terms of execution time, number of floating point operations per second and efficiency while increasing the number of threads. The efficiency is defined as the ratio between the measured and the theoretical maximum performance. X-axis represents the number of threads and values on the y-axis relies on a common scale for performance (seconds and GFLOPS) and efficiency (%). Inspecting the time and GFLOPS traces, system performance increases until the 22nd thread and then drops, indicating a peak for a 8192*8192 configuration (involving 1.5 GB of RAM). This means that beyond this peak value, increasing the number of cores is useless for this benchmark configuration. As the parallelism grows, the distribution of workload reaches a point above which there is an heterogeneity of computations caused by desynchronizations between threads due to an under-utilization of some cores. This is also the reason for multiple non deterministic variations we can observe after this point. A larger SGEMM matrix size would be required to reach the peak performance at the 48th thread, but we start to exceed here the limits of AFM abstraction level leading to prohibitive simulation times (more than 2 weeks). The re-

duction of the execution time is proportionally divided by the number of threads as we process the same workload with increasing number of cores. As previously noticed, there is a point where thread heterogeneity limits the efficiency of parallelization leading therefore to a performance threshold level.

3.4 Conclusion

In this chapter, we have examined in detail how a combined use of relevant models, tools, platforms and benchmarks could be used to define a robust design space exploration approach adapted to the tight processing efficiency constraints of upcoming HPC SoCs, especially in the new perspectives offered by 64-bit ARMv8-A cores. Proper architectural exploration is decomposed in two steps that allow i) reliable modeling and simulation at node/cluster level and ii) scalability analysis of a larger number of nodes using ARMv8-A core models. Reported experiments and results have shown the ability of the approach to reliably study central design parameters, namely in terms of FLOPS performance and efficiency, cache and memory hierarchy, and scalability support up to 128 nodes. Correlation results demonstrated the functional accuracy of Arm fast models. Therefore, the combination of tools towards our methodology and modelling goals are described in the following: i) Arm fast models platform: to maintain a SoC composed of the last Arm IPs versions available and provided as fast models, to enable software/hardware cosign, firmware development and HPC workloads optimizations and ii) Platform Architect will be mainly used for interconnect and the memory sub-system exploration, independently of software constraints, by using interactive traffics called task graphs, generated from a real HPC software execution on the fast models based platforms and iii) Gem5 will be used for cycle accurate ARMv8 generic cores and clustering simulations.

Since the target platform is designed to take advantage of large ARMv8-A clusters, communication topology and memory system are key issues to address. In that respect, SoC partitioning becomes an attractive option to consider due to high development and production costs of large monolitic chips in the latest silicon technologies. Next architectural study extends therefore previous exploration approach (Platorm Architect, 64-bit Arm cores and a specific interconect) with necessary hardware requirements, especially regarding inter chip cache coherence support between compute nodes, in a way to study the impacts and efficiency conditions in different partitioning scenarios. In the next chapter, we will explore opportunities for multi-SoC partitioning based on a directory-based coherent interconnect (SCI) defined specifically for this purpose. Using the methodolgy described we will analyse the impact of SoC partitioning in different scenario and topologies, and compare their coherence traffic overhead.

Chapter 4

Architectural exploration

4.1 Memory coherency and SoC partitioning

This chapter discusses in detail different aspects of micro-architectural design space exploration based on the previously defined methodology. This exploration study addresses more specifically the validity of SoC partitioning as an alternative to using large SoC designs. Indeed, instead of having one large SoC which is very expensive to manufacture, there may be two or four small SoCs ensuring coherency between the 128 cores and providing acceptable performance and power efficiency. On the basis of a monolithic SoC design integrating 128 ARMv8-A cores, high bandwidth memory (HBM/HMC) and a Bull Exascale computing network interface controller, the idea is to evaluate coherent multi-SoC models (i.e. two SoCs of 64 cores, four SoCs of 32 cores) communicating through chip-to-chip ports and coherent proxies (figure 4.5).

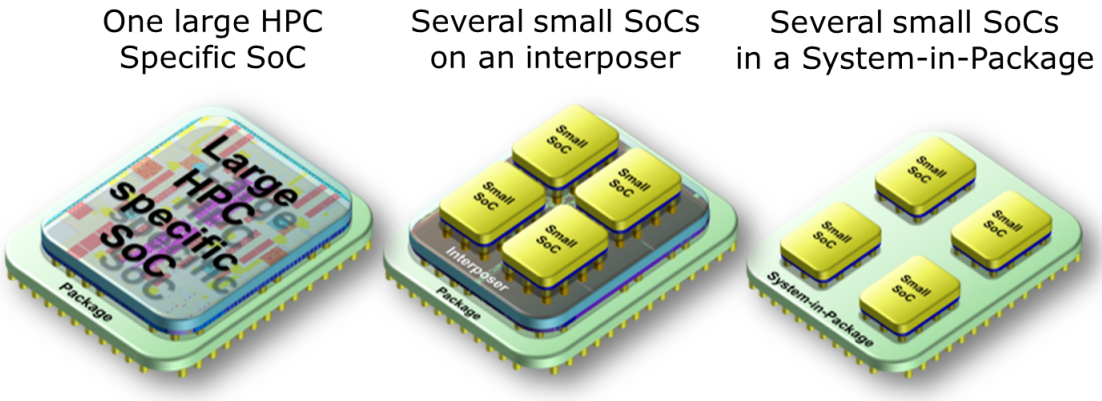


Figure 4.1: SoC partitioning scenarios (Mont blanc project)

Considering the global architecture model (section 4.3.5.3), a bottleneck lies in the cache coherence management because existing snoop based protocols don't scale with the large number of caches that are commonly found in HPC processing. Therefore we introduce a coherence extension of the SoC interconnect (SCI) required for this partitioning. This aims at reducing the complexity of the coherence protocol and additionally can significantly reduce the energy consumption from the interconnect as well as the tag lookups in the remote caches.

4.2 Cache coherence protocols

To address the discussions about different methods leading to ensure data consistency for a given memory hierarchy in multiprocessor architectures, let us first start in defining what coherence stands for. A suitable definition of cache coherence is given in [63] as single-writer-multiple-reader (SWMR) invariant. It means that for any memory location or space, at a given cycle time, there might be only one single writer or a number of cores that may read it. Consequently, implementing a cache coherence mechanism requires avoiding the situation where two separated caches contain two different values for the same memory address at the same moment. In a multicore system, there are generally several caches where data coherence must be achieved. Thereby, a cache coherence protocol is used in multiprocessor architectures to provide processors with a consistent view of main memory. Especially, the goal is to reflect the writes made by each processor to the others, by modifying or invalidating the common cache lines. Then, the cache coherence protocol provides states to each piece of the main memory stored in the caches. Considering the MSI (Modified, Shared, Invalid) cache coherence protocol, a cache line in Modified state is the only valid copy. The cache instance is qualified as the owner but the copy in memory can be different. To ensure coherence, the cache line must be copied to memory in case of a cache line replacement. The Shared state means that the line is not owned by the local cache (even if it has been modified by it) and is at least shared with the owner cache. The local cache is not responsible for global data consistency. Finally, the Invalid cache line state causes a cache miss as the stored data is untrustworthy. There are several extensions of the basic MSI (a.k.a. Illinois) cache coherence protocol, such as MESI (Modified, Exclusive, Shared), MOSI (Modified, Owned, Shared, Invalid) and MOESI (Modified, Owned, Exclusive, Shared, Invalid). These methods address the way of maintaining consistency between the multi-level cache hierarchy and the main memory. In the past, the memory consistency was handled at the software level, but the performance requirements became critical as the systems got bigger[64]. This is typically

inadequate for typical HPC systems. Hardware-based cache coherence protocols thus appeared to be more efficient and are commonly used now in many computing systems such as servers. A cache coherence protocol specification must define several key elements of the protocol background such as hardware components, coherence granularity, caches sizes, transactions types, coherence interconnect channels or the impact of transactions on cache line states. Therefore for the same coherence protocol, there may be several implementations depending on the type of architecture, memory hierarchy, number of cache elements and topology of the on-Chip coherent interconnect.

4.2.1 Overview

There are many comparison studies between the two main hardware based coherency approaches: snooping cache coherence protocols (snoopy protocol) and directory-based cache coherence protocols [15,17,18] . It always emerges that the first one is logically simple, ideal for ring interconnect topologies but does not scale with large numbers of caches. This comes from the number of snooping broadcast transactions which increases quadratically with $N * (N - 1)$, where the N is the number of coherent agents [19] . The second one scales very well but the drawback is that transactions take more time because of the additional complexity of directory filtering. The traffic scales in theory with $N * ((N - 1) + 1) = N^2$ where +1 is due to the request for directory lookup [65]. In reality, this overhead only happens when all caches in a system have a copy of the requested data which is very rare in a large scale system (tens of caches). As it also depends on the workload parallelism, snooping traffic could thus be reduced between caches sharing a copy of the same data. The main counterpart of directory based cache coherency techniques is their implementation cost based upon on-chip SRAM storage which size depends on the number of cache lines to manage in the system. Therefore, the potential for reducing cache coherence complexity attracts a lot of interest as it impacts the processing efficiency of large scale compute nodes. The goal in the following is to develop a simulation based analysis for HPC benchmarks in different workload configurations in order to evaluate the impacts of coherence protocol overhead and SoC partitioning. To achieve this we use the ARM AMBA 4 ACE protocol (described in the subsection 4.2.4) supporting features for either snoop or directory based approaches .

4.2.2 Snoop Transaction types

There are two types of snoop transactions: snoop coherence (Write updates) and snoop data request (Read requests) transactions. Snoop coherence transactions are required to

maintain coherence of the data shared between caches. Snoop data request transactions are used to save cluster request times in case of read snoop hits, meaning that the data requested is available at least in one nearby cache. These transactions have to be considered in the cost evaluation of SoC partitioning. It is unclear whether there is a benefit in maintaining inter-SoC read snoop transactions, since the latencies increase de facto (additional interposer delay for inter-SoC communications), and it could even be more efficient to seek directly the requested data in the local L3 cache or local memory when the address belongs to the local (intra-SoC) partition. In this regard the multi-SoC architecture has NUMA properties, a central foundation of the Intel's MESIF coherence protocol where the "Forward" state determines at least one cache element per socket to be responsible of the coherency. Its behaviour is similare to the "Shared" state and in addition, it indicates the designated cache responder for any requests for a given line. This improves the overall performances by saving the bandwidth required to satisfy memory conflicting requests [66]. For coherence transactions, one potential way to reduce write snoop transaction overheads is to implement a mechanism of Dynamic Self-Invalidation (DSI) as proposed in [3], in which each cache should care of removing its local copy of a cache block before a conflicting access occurs, then the directory cache line is updated to avoid further potential snoops. Results show that DSI can reduce up to 41% of the execution time of a sequentially consistent full-map coherence protocol and up to 26% of both invalidation and acknowledgment messages by exploiting tear-off blocks. Even though DSI does not impact the consistency of program execution because it has exactly the same semantics as a cache replacement policy, it can however lead to unnecessary cache misses [66]. This option was not addressed in our exploration study because we only focus on the coherence protocol and influence of the directory, but it could be envisaged for physical design specification of the final system.

4.2.3 Directory based filtering

A directory-based coherence protocol is more appropriate to reduce the number of transactions to those that are only necessary. Limiting the number of transactions will also reduce the energy consumption due to activity in the interconnect as a direct consequence (reduces traffic and average latencies). This solution requires the use of a "snoop filter" as presented by Moshovos et al [67]. They propose two means of filtering: i) on the requesting side (source filtering) by using a "source snoop filter" which sollicitates the common directory and sends transactions only to the caches which can have the data of the specified cache line address, ii) at the receiver side (destination filtering) to decide whether or not to send a transaction to the corresponding cache controller depending on

the availability or status of the requested data (each cache interconnect port having its own private directory). In the first approach, there is only one global directory for all requesting caches, which means that the directory covers all cache lines, then filters the requests before they reach the on-chip interconnect. The resulting decrease of traffic can help to significantly reduce activity (thus power) in the interconnect and search of labels in the remote caches [66]. The second approach has the advantage of decentralizing the snoop filter into small logical directories. However, it has no impact at all on the internal traffic of the interconnect. With the source filtering approach, there is only one logical centralized filter (directory). As the directory must have a complete view of all system cache lines, the on-chip interconnect topology must be well sized to avoid bottlenecks, but this is the commonly used approach.

4.2.4 ARM Coherence Protocols

Regarding the target ARMv8 based compute, SoC coherence must be compatible with ARM standards. There are several existing protocols to choose from, depending on the needs of the coherence approach, system scalability and the expected traffic in the memory subsystem. The AMBA protocol (Advanced Microcontroller Bus Architecture) is a popular open-standard used for communications of several types of functional blocks in a SoC design. Master (requester block) and slave (receiver block) concepts are common to all other communication protocols. The on-chip interconnect connects all components and peripherals with different types of ports (interfaces) and deploying different protocols. Introduced in 1996, the AMBA1 specification addresses two types of buses found in embedded SoC designs: the Advanced system Bus (ASB) and the Advanced Peripheral Bus (APB). The single clock-edge version named AMBA2 High-performance Bus (AHB) was introduced two years later to scale with more complex designs. This procotol, like previous ones, is based upon low complexity shared buses in an attempt to minimize the area overhead. Then ARM introduced in 2003 the Advanced Extensible Interface (AXI) protocol to reach even higher performance and bigger non-coherent SoCs. AMBA3 is widely deployed in most of embeded systems. However, the AMBA4 ACE (AXI with Coherent Extension) has been introduced since 2011 in order to meet specific requirements at server level (multi-processor). Since the AMBA4 protocol suffer critics about its incompatibility to non-ring based on-chip topologies, the current AMBA 5 CHI (Coherent Hub Interface) comes with improvements on this topic.Architecture exploration in the following contribution is based on ACE.

4.3 SoC Coherent Interconnect

4.3.1 Description

The main role of the SoC Coherent Interconnect (SCI) is to manage the communications between processors (ARM clusters) and to ensure data consistency of the whole memory hierarchy. Unlike traditional approaches which tries to extend the coherence of several chips with an additional off-chip coherent interface (e.g. UPI, QPI see 2.2.2), the proposed SCI is an on-chip solution designed on top of the ARM AMBA 4 ACE protocol. Even though there is a more recent ARM coherence protocol called AMBA 5 CHI, the conclusions drawn on the SoC interconnect architecture and coherent protocol overhead in the exloration study are not less relevant. The SCI must:

1. perform parallel communications and determine the order of transactions when multiple transactions are received at the same time.

2. issue efficiently an incoming snoop request transaction from a cluster to other coherent clusters with a copy of the data according to the snoop transaction type.

3. centralize and generate snoop responses for the initiating clusters and apply the required updates or request operations to the next level of the memory hierarchy.

4. perform writeback / writethrough operations and snoop write transactions to ensure coherence of the data.

4.3.2 SCI Architecture

All clusters are in the same shareability domain (see figure 4.2). This defines a set of initiating components and determines which other components are invoved when issuing coherence transactions. The overall system is composed of 32 VPUs (see 4.3.3), each VPU can possibly exchange snoop transactions with the 31 other VPUs. This may lead to very large snoop controllers and important snoop traffic to ensure global data coherence. In such a type of multi-socket system, a snoop based coherence scheme is not very efficient because the traffic complexity grows with $N * (N - 1) * T$, where N is the number of L2 caches (here 32) and T is the number of incoming transactions, corresponding to the sum of L2 cache misses and non-cacheable operations. In the following analysis, we focus on read cacheable transactions because they represent more than 99% of the traffic generated on the interconnect.

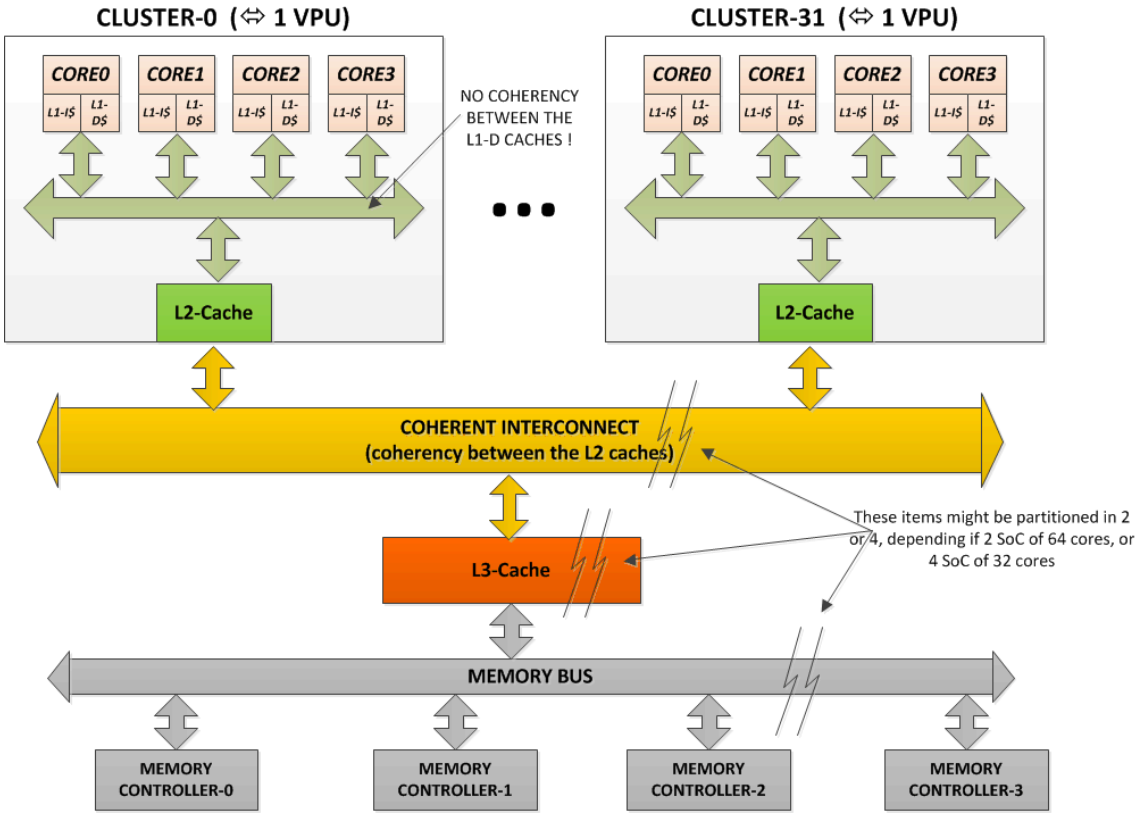


Figure 4.2: on-Chip Coherent Interconnect

Cache	L1 Icache	L1 Dcache	L2 cache	L3 cache
Size (in #Bytes)	32 KB	32 KB	2MB x 32 clusters	64 MB
Latency (cycle)	3	4	5	15
Associativity (#Ways)	4	4	16	16
Replacement algorithm	Round robin	Round robin	Round robin	Round robin
WB/WT (for D caches)	-	WB	WB	WB
Cache Line size (B)	64	64	64	64
Data width (bit)	64	64	128	128
Address width (bit)	64	64	64	46
Outstanding buffer	32	32	64	256
Protocol	AXI	AXI	ACE	ACE

Table 4.1: Cache characteristics in the 128 cores VPU platform

WB : Write Back

WT : Write Through

4.3.3 Cache model

The design analysis is based on Platform Architect methodology. The platforms configured are shown in Figure 4.6. A Virtual Processing Unit (VPU) is generated to mimic the behavior of a real ARM processor exchanging transactions through the SCI. Table 4.1 reports the characteristics of the cache subsystem in the platform model.

4.3.3.1 Cache controller

The L2 cache controller must manage the last level of cache shared between cores of a same cluster. Its behaviour is described in the VPU model featuring three bidirectional main ports:

- AXI_OUT initiating InOut port, responsible for sending data requests to the attached snoop controller (for this VPU) and for receiving responses.
- SNOOP_IN slave InOut port, used for incoming snoop transactions from other snoop controllers.
- MASTER_IN slave InOut port, receiving cacheable/Non-cacheable operations from the four cores of the CPU cluster.

These ports support a TLM2 FTAXI (Fast Transaction model of the AXI protocol) interface performing the following ACE transactions:

- Non-snooping transactions (Write)
- Coherent transactions (Read and Write)
- Memory update transactions (cache line state update, writeback, etc.)

The cache replacement policy is based on Round-Robin Least Recently Used (LRU) algorithms. LRU being the L2 cache of the CPU cluster. Transaction latencies considered for analysis are:

- TAG lookup delay for any incoming cacheable request through the MASTER_IN port. This is the time needed by the lookup cache hardware to identify the cache line index of a corresponding address.
- Data access delay in case of a hit, which represents the latency of a cache for transfering data to the requesting master (core or cluster).

4.3.3.2 Cache Snoop controller

The Cache Snoop Controller (CSC) is a component in the SCI. The CSC collects input requests from its corresponding clusters and then:

1. Sends each snoop request address to the Central Snoop Filter (CSF) which queries the presence bit of the cache line containing that address to the directory.
2. Sends snoop transactions only to the VPU that may have a copy of the data, or to the Coherent Proxy extension if the data is stored on an external SoC partition.
3. Issues the response to the requester through the response channel (in case of a snoop hit), or transmits the request to the level 3 cache (in case of snoop miss).

In case of write coherence transactions, the Cache Snoop Controller is responsible for updating all the caches containing a copy of the cache line in the overall memory subsystem. Further details are available in figure 4.7 (section 4.3.5.3).

4.3.4 SCI Snoop Filter model

The snoop filter model implemented in the virtual platform has input signals from each initiator (VPU clusters) and output signals to each slave (L3 cache slice partitions and memory controllers). It supports the directory protocol for all L2 caches in the system. It provides an access arbitration on the snoop address and snoop data channels. Supporting

multiple snoop transactions for the same cache line is a requirement specified in the ACE protocol. In addition, the filter permits writing dirty cache lines to the main memory when the flag IsDirty is set. The snoop filter has 32 snoop controllers, 1 centralized snoop controller, several ring buses and Mux/DeMux components (figure 4.7). A rigorous model of this coherence extension scheme is defined in the VPU model which is then used for architecture exploration in the following. From the task graphs, VPUs generate traffic on the interconnect and the coherence is ensured by a proxy component.

4.3.5 SoC Partitionning

4.3.5.1 Partitioning topology overview

Connecting multiple chips on an interposer migth look simple. In the scenario where there are two SoCs, the solution is straightforward in terms of chip-to-chip topology. The on-chip interconnect therefore allow for bi-directional coherence extension as illustrated in figure 4.3.

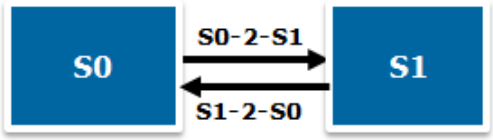


Figure 4.3: 2 SoC : simple Chip-toChip topology

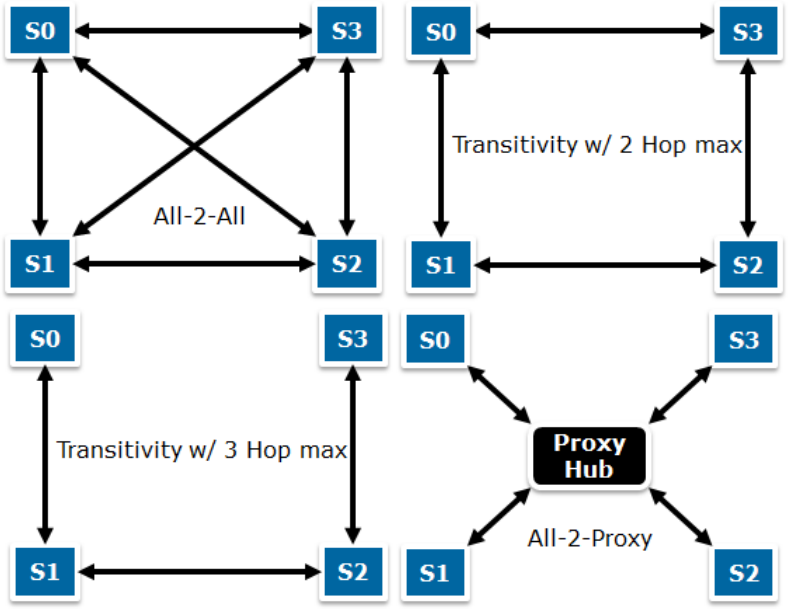


Figure 4.4: Four small SoC : The on-Chip Interconnect partitioning topology alternatives

However there are different ways to connect four SoCs: through an additional routing module (Hub or Switch) in the proxy or with indirect (forwarding) connections. In this study, we have considered the following topologies (figure 4.4):

1. All-2-All : this topology requires three additional coherent ports on the SCI for proxy extension.

2. Transitivity with a maximum of two hops. Thereby, two additional coherent ports are necessary on the SCI for proxy extension. Each socket has two direct connections to the first two neighbors and must use one of them to forward the traffic to the destination (maximum of two jumps for each SoC partition).

3. Transitivity with a maximum of two hops. This topology has the greatest routing complexity and may be unbalanced. Partitions S1 and S2 are the preferred ones (two hops maximum) when S0 and S3 have to undergo three jumps to send/receive a request/response.

4. All-2-Proxy : a coherent proxy hub is responsible in switching communications. Therefore, only one additional port is needed on the SCI to manage coherence via the immediate neighbour connected. The proxy could be directly implemented on the multi-SoC Interposer. Nevertheless, sizing this proxy is a key point to avoid traffic congestions. This topology is the one selected in the final architecture to validate the multi-SoC coherence requirement.

4.3.5.2 Multi-SoC scenarios

Figure 4.5 provides a high level view of a large SoC configuration including 128 ARM cores, high bandwidth memory (HBM/HMC) and a Bull Exascale computing network interface controller [12]. As stated previously, we explore a coherent SoC partitioning scheme to help reducing the very large complexity and cost of the resulting design. This partitioning is based on the three following scenarios :

1. One large SoC (left side of figure 4.5)

2. Two smaller SoCs: this is the scenario involving two small SoCs called sockets. Each socket includes 16 VPU clusters (64 cores per cluster). The two sockets are connected with chip-to-chip coherent proxy ports. In this context, a latency of 20 cycles is required for external transactions (the interconnect being clocked at 2 GHz).

3. Four smaller SoCs: the principle is similar. We used the All-to-Proxy topology to connect the four SoCs for two reasons: i) it has less routing complexity in comparison to the All-to-All topology. ii) because this is the worst case when implementing a single hop configuration, as the Proxy hub is sensitive to bottlenecks.

Partitioning leads to Non Uniform Memory Accesses (NUMA) as each SoC will have

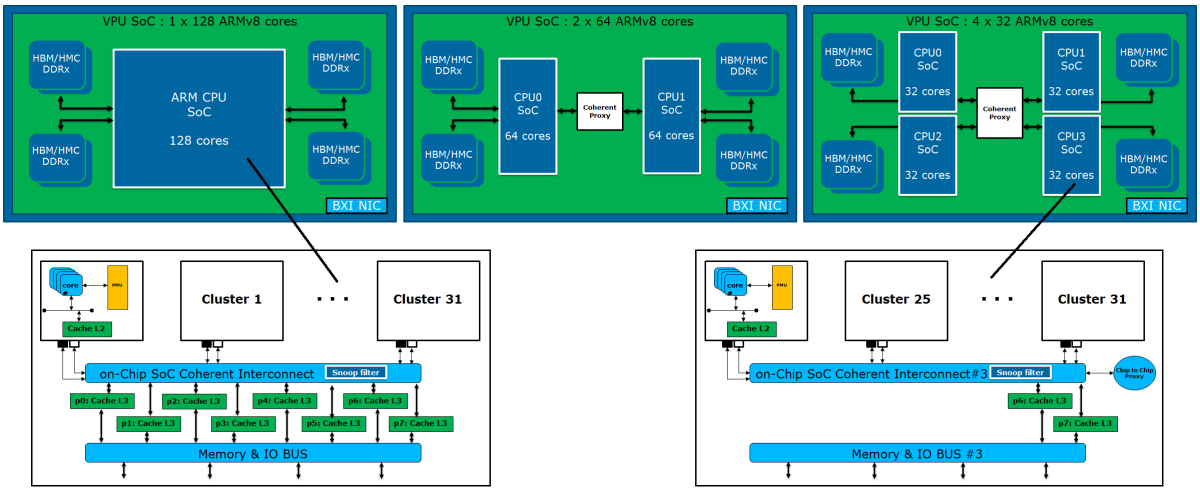


Figure 4.5: Large scale ARM based VPU : partitioning scenarios

access to its own local memory area and external memory areas as well. In this type of architecture, the programmer is in charge with explicit load balancing (threads and data) and the exploration analysis takes this into account. The SoC Coherent Interconnect (SCI) has a crossbar topology, meaning that each VPU has direct links to other VPUs. This asumption corresponds to the ideal topology of a Network on Chip. In a classical single SoC, employing such a full crossbar topology is irrelevant because of the too large number of interacting devices.

4.3.5.3 Coherent Proxy extensions

We consider two partitioning scenarios of the SCI resulting from splitting a single-SoC 128 cores topology in two and four partitions. Figure 4.6 shows the two block diagrams for the corresponding cluster configurations of 2×64 and 4×32 cores. All the processors in a partitioned scenario must be able to communicate coherently as if they were connected to the same on chip coherent interconnect. All SoCs are thus connected through chip-to-chip coherent proxy ports. The main role of these coherent proxy ports is precisely to enable both coherent transactions with the neighbouring sockets and accesses to external memory areas. The L3 cache is considered to be LLC (Last Level Cache) near each memory controller to save latencies for memory requests.

Coherence extension must remain transparent from a cluster cache agent perspective. Indeed, 128 cores are distributed between 32 clusters of 4 cores. They all are in the same shareability domain and must be able to communicate with coherence whether they are

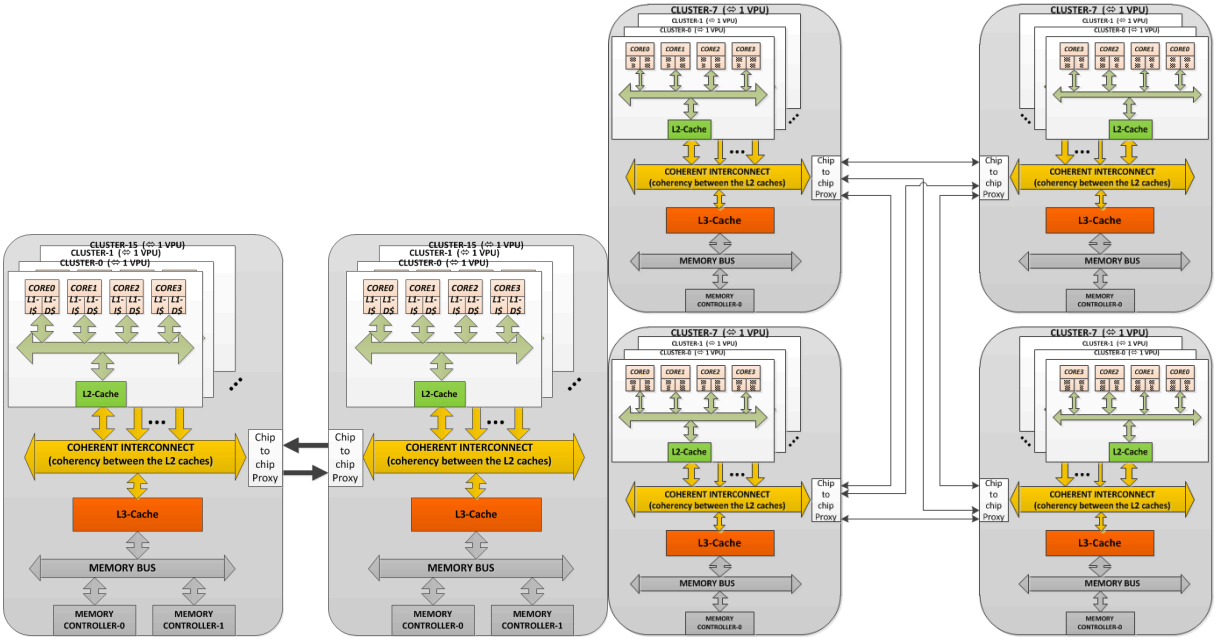


Figure 4.6: 2 ×64 and 4 ×32 SoC partitioning with coherent proxy ports

in the same SoC or not. Figure 4.3.5.3 depicts exactly the SystemC / TLM2 directory based filtering model as it would be implemented in the SCI. For example, in the scenario where there is an incoming snoop request from the L2 cache, the snoop controller sends a request to the directory to locate all copies of the data in the nearby peer caches (in its shareability domain). Then it queries directly the caches identified in the local socket or through the proxy extension if an external transaction is involved. In case of a snoop miss, the request is forwarded to the next cache level (L3). The proxy component architecture is quite similar to an empty L2 cache. It repeats incoming snooping requests from a SoC to the other(s). This leads to additional delays and asymmetric waiting response times to the snoop controller requests.

The snoop controller is a module attached to each coherent interface (port) in the SCI. It is responsible for processing coherent transactions from a cluster (VPU) to the directory and the (N −1) other peer interfaces. When the SCI manages 32 coherent ports (32 clusters of four cores), each snoop controller must be able to communicate with the other 31 controllers. In the scenario of partitioning in two SoCs, it will communicate only with 16 snoop controllers (15 coherent VPU ports and the proxy one) instead of 31 snoop controllers in a large SCI (Figure 4.7). This reduces de facto the logic design complexity and the corresponding physical area. The idea remain the same when partitioning in four SoCs, reducing connections from 31 links to 8 (7 peer ports + 1 hub proxy port) in a one-to-all topology or from 31 to 10 (7 peers ports + 3 direct proxy port) for an all-to-all

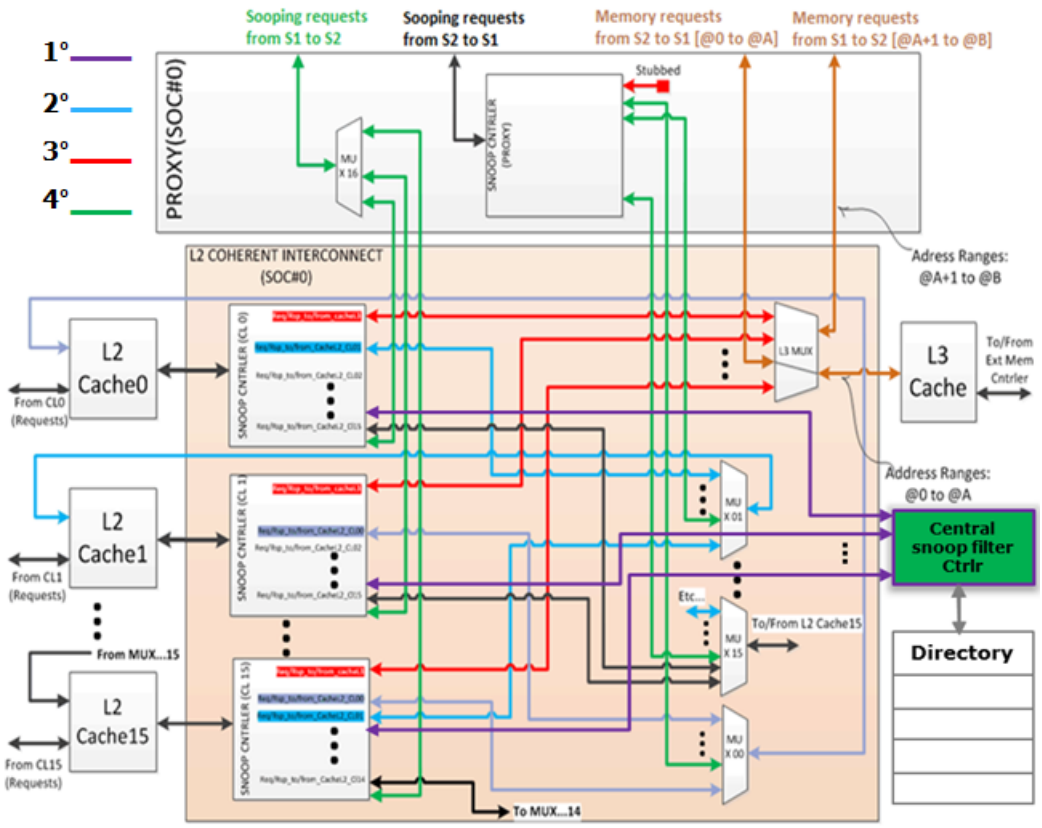


Figure 4.7: Coherent proxy entensions

topology. The remainder of the exploration study will therefore focus on evaluating the relevance of these partitioning asumptions and the associated coherence extension.

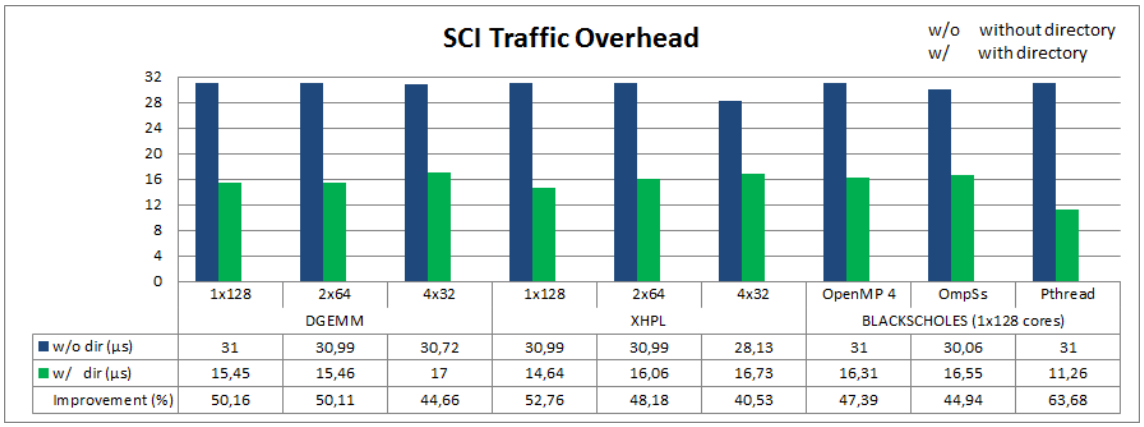
4.4 Simulations

In this assessment study, we address first the ability of using previous directory based snoop filtering model in the SCI to reduce the complexity of cache coherence management and the impact on performances considering different types of benchmarks. We then analyse the relevance of two SoC partitioning configurations with two SoCs / 64 cores (2×64) and four SoCs / 32 cores (4×32) against one SoC / 128 cores (1×128), with and without the proposed snoop filtering scheme. Finally we consider more specifically the effect of different parallel programming paradigms on the internal traffic of the coherent interconnect and the corresponding performances.

4.4.1 Directory-based snoop filtering benefits

Figure 4.8 reports the analysis of snoop transactions generated in each of the three configurations of 32 masters (128 cores). Without directory, 31 transactions need to be generated for each cache miss (or invalidation request) to update every copy of the data. Including a directory reduces the traffic generated in the SCI between 40% and 63%, which is in line with the reduction of size of the snoop controller by a factor of two, from 31 snooping output ports to 16. In turn, transaction benefits improve benchmark execution times by reducing L2 cache miss penalties. Therefore, figure 4.10 compares execution times of the same benchmarks, with and without directory, to examine these gains. The impact of using a directory in our coherent multi-SoC architecture model is thus an average execution time improvement of 14% (4%–26%). Despite transaction savings of about two, net performance gains are limited by the relative low number of cache misses in practice in real applications (around 2.7% reported for DGEMM in Figure 4.9).

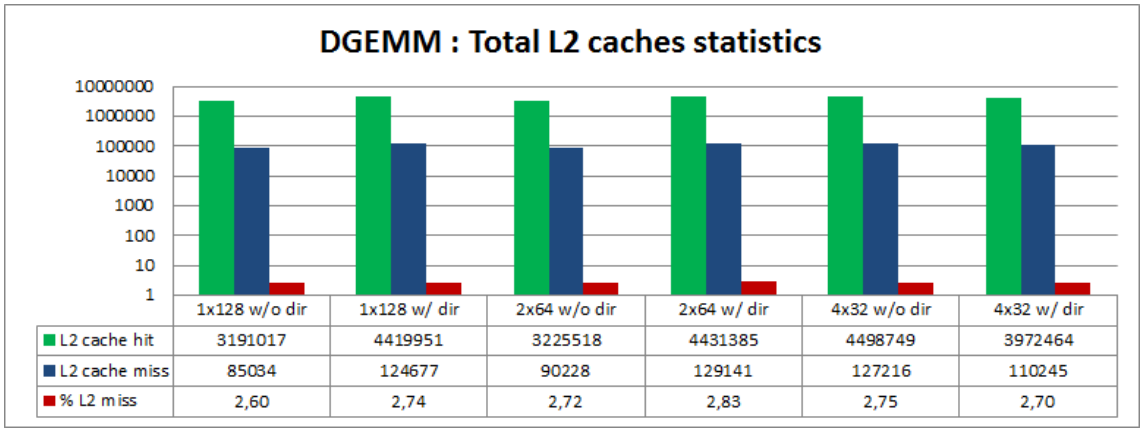
However, the consistency of data shared between processors is very complex in large scale computing systems. The use of snoop filtering in the three considered SoC models ensures both data coherency and processing efficiency by reducing transactions to those that are strictly necessary. The overhead for each incoming transaction is thus reduced by more than 40% which is an important matter in consumption and scalability at the memory subsystem level, while performance may be slightly improved at the same time. Memory consistency among clusters is thus ensured with significantly less coherence traffic in light of these results. Despite little benefits in terms of net system performance, this should however bring enough improvement to promote multi-SoC partitioning at low chip-to-chip communication costs, which is the question discussed next.



	1x128	2x64	4x32	1x128	2x64	4x32	OpenMP 4	OmpSs	Pthread
	DGEMM			XHPL			BLACKSCHOLES (1x128 cores)		
w/o dir (µs)	31	30,99	30,72	30,99	30,99	28,13	31	30,06	31
w/ dir (µs)	15,45	15,46	17	14,64	16,06	16,73	16,31	16,55	11,26
Improvement (%)	50,16	50,11	44,66	52,76	48,18	40,53	47,39	44,94	63,68

Figure 4.8: Number of transactions for different SoC and directory configurations

Overhead = number of transactions generated for one incoming request



	1x128 w/o dir	1x128 w/ dir	2x64 w/o dir	2x64 w/ dir	4x32 w/o dir	4x32 w/ dir
L2 cache hit	3191017	4419951	3225518	4431385	4498749	3972464
L2 cache miss	85034	124677	90228	129141	127216	110245
% L2 miss	2,60	2,74	2,72	2,83	2,75	2,70

Figure 4.9: DGEMM cache statistics

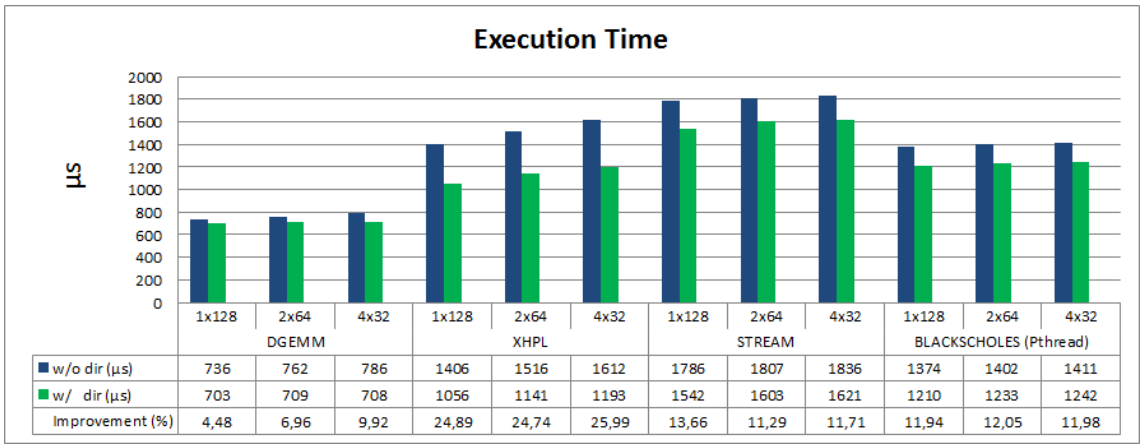


Figure 4.10: Benchmark performance for different SoC and directory configurations

4.4.2 Partitioning analysis

If we focus more specifically on multi-SoC configurations (2×64, 4×32 w/o dir) against single-SoC (1×128 w/o dir) in the results of figure 4.10, we can observe a legitimate deterioration of performances when no directory is used, increasing gradually with SoC partitioning due to the additional latencies coming from chip- to-chip transactions. Considering SoC partitioning in a large scale manycore system does not seem at first to be an effective approach as shown with up to 15% drop in performance for XHPL in configuration 4×32. However, the presence of a directory improves both single-SoC and multi-SoC performances by respectively 13.7% and 14.33% (in average) since transactions are reduced to what is strictly required for cache coherence. The additional complexity introduced by the directory is widely compensated by the removal of internal/external waiting delays. This gain rise up from 4% up to 25% depending on the workload profile. The XHPL benchmark, which employs a large memory space and is very sensitive to latencies, is therefore the ideal example to show the impact of a directory in our on-chip interconnect with 25% performance improvement.

In addition, performances remain stable in the presence of a directory with an average variation of 1.9% across all SoC configurations and benchmarks. If we compare the partitioned (2×64, 4×32 w/ dir) versus single-SoC (1×128 w/ dir) topologies, it can be verified that the impact of partitioning on execution times has been efficiently limited through the use of the directory (4.3% for 2 ×64 and 5.4% for 4 ×32). The performance level is more stable in both partitioned SoCs because there is always a copy of shared data in nearby caches resulting in no snoop misses. These results let us therefore expect an average performance penalty of 5.2% resulting from SoC partitioning (in two and four SoCs for a 128 nodes example). But employing the defined directory-based filtering

scheme is efficient enough at reducing chip-to-chip cache coherency transactions and to get rid of this overhead with even a mean improvement of 10.1% (over single- SoC without directory). The coherence extension scheme then promotes interesting opportunities such as the integration of more compute nodes directly on an interposer based System-in-Package (SiP), possibly based on 3D Through Silicon Vias (TSVs) using High Memory Bandwidth (HBM), to approach the processing power and efficiency of Exascale requirements.

4.4.3 Parallel programming efficiency

Another factor which may affect the value of SoC partitioning relates to software and parallelism. The issue here is how to best minimize outgoing transactions between the partitioned SoCs at the programming level. We have thus considered three parallel programming models (OpenMP, OmpSs and POSIX Threads) on a blackscholes application (part of the PARSEC benchmark suite [20]) to investigate their influence on the efficiency of the directory. OpenMP, OmpSs and POSIX Threads are application programming interfaces (APIs) for multi-platform shared-memory parallel programming. OpenMP provides high level threading options using code and dataflow annotations that are then used by the run-time system for execution and synchronization. POSIX threads is a lower level API for working with threads offering fine-grained threading-specific code to permit control over threading operations. Unlike OpenMP, the use of Pthreads requires explicit parallelism expression in the source code (e.g. hard-coded number of threads). OmpSs is a mix of OpenMP and StarSs, a programming model developed by the Barcelona Supercomputing Center. It provides a set of OpenMP extensions to enable asynchronous tasks, heterogeneity (accelerators) and exploit more performance out of parallel homogeneous and heterogenous architectures.

With previous results showing little influence of partitioning when using a directory, the following analysis is restricted to a single-SoC configuration. Fig. 15 reports execution times of the blackscholes benchmark for each programming model. There are comparatively few differences between OpenMP and OmpSs results because of their similarity. The application receives little benefits (4.6% performance improvement) from OmpSs specific features to better exploit the architecture model. However, figures 4.11 and 4.12 show two clear inflection points with 82.6% less performance and 40.9% fewer application throughput using Pthreads compared to OpenMP and OmpSs. Investigating further shows that Pthreads has 28% less throughput than OpenMP (in configuration 1×128 with directory) for 79% more cache misses (see figure 4.13). In turn, cache misses are responsible for generating 23.5% more traffic in the SCI compared to OpenMP (see

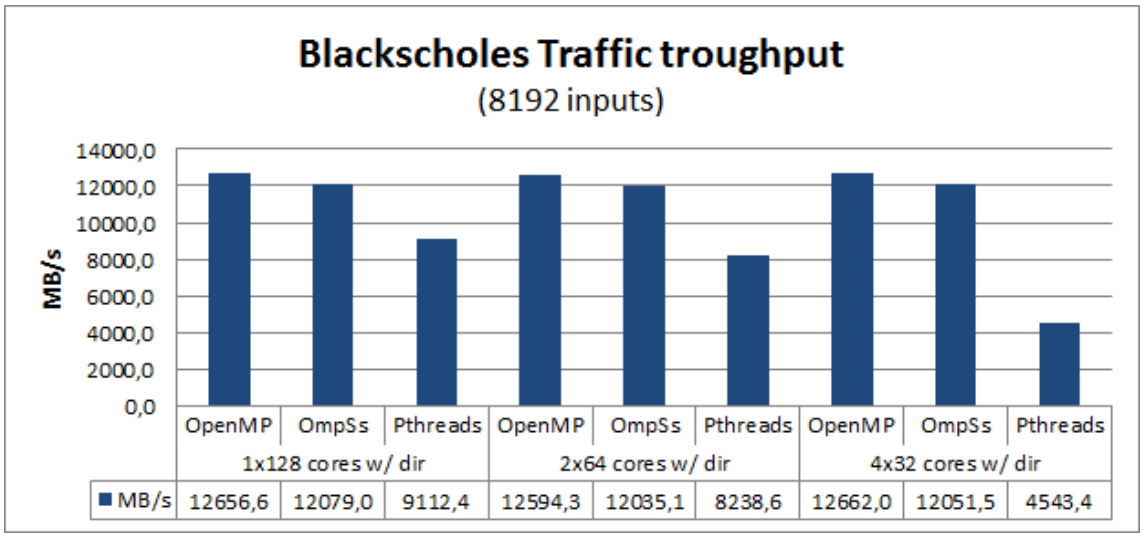
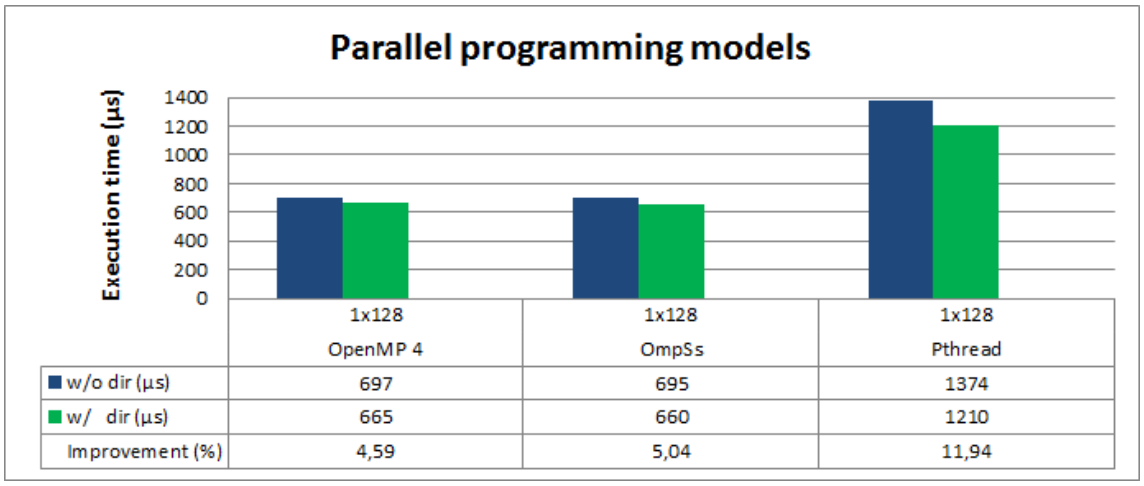


Figure 4.11: Impact of programming models on throughput (blackscholes)

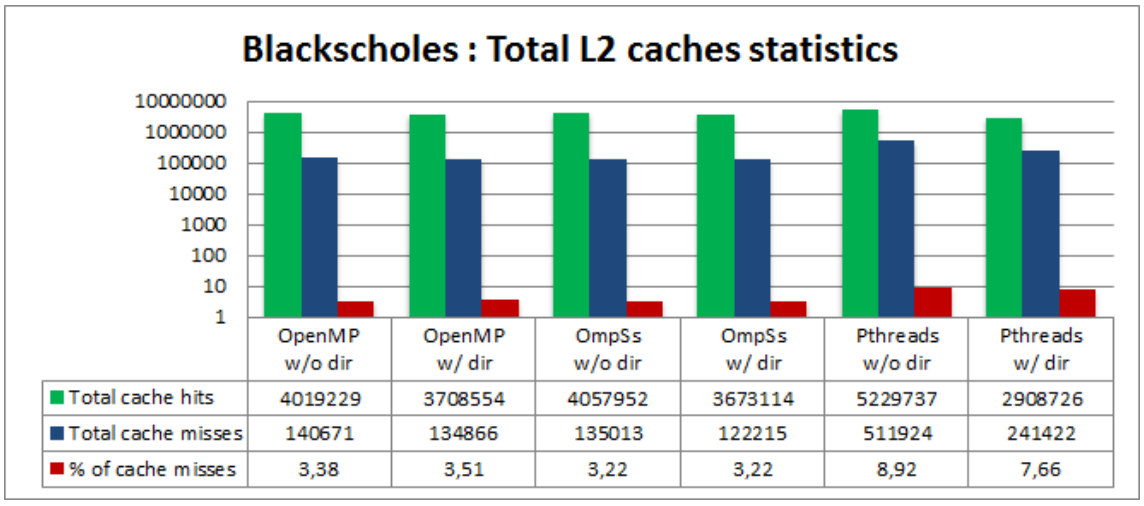
figure 4.14).

Figure 4.15 confirms that Pthreads has not led to an efficient use of the directory. With 77.37% snoop misses, the on-chip interconnect had to carry out the transport of five times more transactions than OpenMP. The reasons of these weaknesses lie mainly in the capability of the software model to address efficiently high degrees of parallelism. In the task-graph based parallel versions of blackscholes used for OpenMP and OmpSs, the work is better divided into units of a predefined block size which allows having much more task instances and better load balance than Pthreads. The effects from less reliable parallelism exploitation can therefore increase significantly (up to a factor of two) when scaling up to 128 cores. Besides the fact that limited conclusions can be drawn on the effectiveness of a programming model which depends on how well the software was partitioned and coded, this study shows however the potential for deep analysis of appropriate parallelism exploitation by the application. These results are beneficial to help tuning the architecture and design of algorithms and software, to identify and correct programming shortcomings and further improve parallel processing efficiency.



	1x128 OpenMP 4	1x128 OmpSs	1x128 Pthread
w/o dir (µs)	697	695	1374
w/ dir (µs)	665	660	1210
Improvement (%)	4,59	5,04	11,94

Figure 4.12: Impact of programming models on performance (blackscholes, 1×128 cores)



	OpenMP w/o dir	OpenMP w/ dir	OmpSs w/o dir	OmpSs w/ dir	Pthreads w/o dir	Pthreads w/ dir
Total cache hits	4019229	3708554	4057952	3673114	5229737	2908726
Total cache misses	140671	134866	135013	122215	511924	241422
% of cache misses	3,38	3,51	3,22	3,22	8,92	7,66

Figure 4.13: Impact of programming model on L2 cache misses (blackscholes, 1×128 cores)

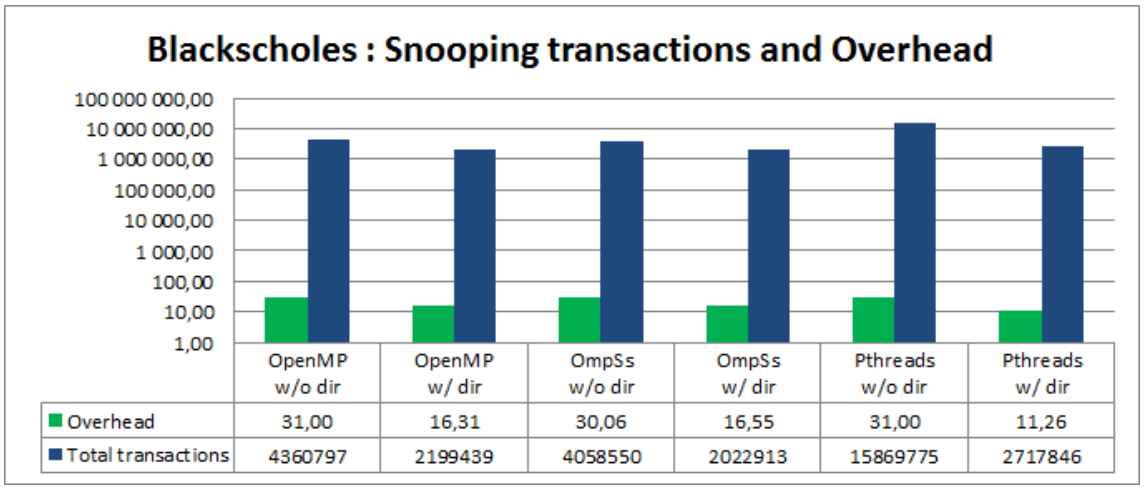


Figure 4.14: Impact of programming models on the number of snooping transactions and overhead (blackscholes, 1×128 cores)

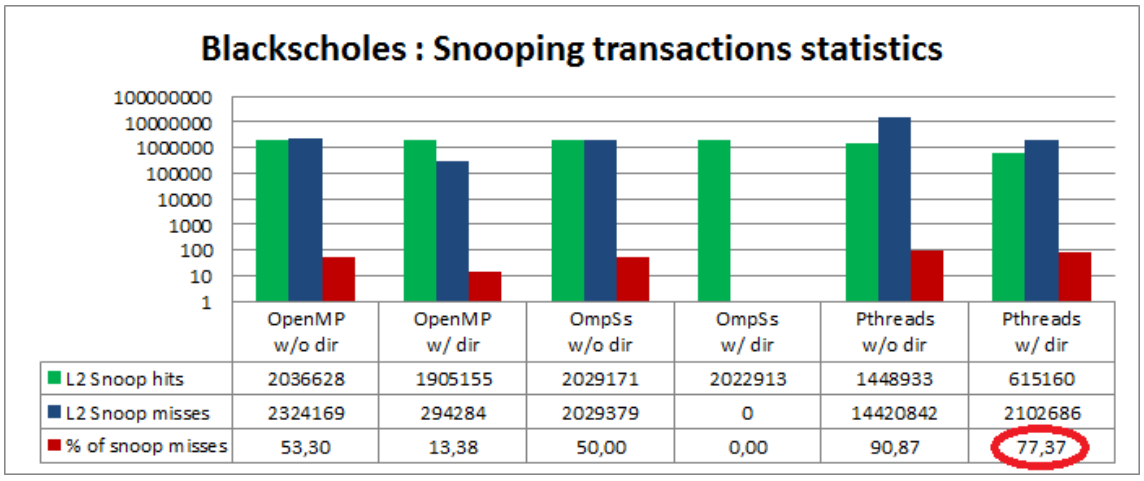


Figure 4.15: Snooping traffic statistics (blackscholes, 1 ×128 cores)

4.5 Conclusion

In this chaper, we have examined in detail a set of architectural exploration opportunities for multi-SoC partitioning based on a directory-based coherent interconnect (SCI) defined specifically for this purpose. Exploration of partitioned (2×64, 4×32) versus single-SoC (1×128) topologies with this coherent interconnect have shown to decrease significantly the associated internal traffic (55.3%) and to limit enough the existing partitioning overhead (4.3% for 2×64 and 5.4% for 4×32) such as to permit an average 10.1% execution time saving compared to the situation where no partitioning / directory is used. Additionally, the analysis of parallel programming efficiency on a concrete example confirmed the validity of directory filtering with the ability to identify and correct software weaknesses for better parallel processing efficiency.

The exploitation of interesting opportunities available from these results are in the context of the Mont-Blanc project for an efficient use of a large number of 64-bit ARMv8-A cores. The ability to partition the system into different SoCs at no loss of performance ensures the feasibility of the solution, and introduces a few possibilities. Indeed cluster configuration can be fine-tuned to better match application requirements. The choice of the number of cores sharing the same L2 cache in a cluster can be based for example on the sensitivity of the workload to cache latencies: one core per cluster for workloads with high cache miss penalties (e.g. with rather iterative processing, limited data set, low amount of shared data), four cores per cluster to address applications with higher parallelism and lower cache miss penalties. Also the size of the L2 caches can be tailored to the SCI directory complexity. As the full directory covers all lines of the L2 caches of a coherent cluster peer, their needs and size depend on the SCI topology (i.e. Mesh, Ring). Finally, this partitioning scheme introduces technology level perspectives for chip integration, such as the integration of more compute nodes directly on an interposer based System-in-Package (SiP), possibly based on 3D Through Silicon Vias (TSVs) using High Memory Bandwidth (HBM).

The considerations of two previous chapters address architectural and design related concerns. Another essential aspect of energy efficiency, especially for large scale mutliprocessor systems, is related to the relevant runtime management of resources. Power management is therefore an important part of the efficiency question which is the subject of the following and last chapter.

Chapter 5

Power management

5.1 HPC energy efficiency constraints

As it is so well described in [68], there are two major constraints related to the energy efficiency in a high-performance computing system: the operational cost and the system reliability, in addition to environment respect. A strong and strategic power management can lead to reduce energy consumption and avoid excess heat to the cooling system which improves system reliability and operational costs (and somehow the environmental impact). We may distinguish a static power management (SPM) and a dynamic power management (DPM). SPM mostly consists of utilizing low-power technologies to reduce power. On the other hand, DPM tries to improve the energy efficiency using both power-scalable components such as hardware-enforced processor power bounds, and software capabilities. The proposal in this chapter describes two workload related strategies to show some opportunities and potential pitfalls using Dynamic Voltage Frequency Scaling (DVFS) in a HPC environment.

5.1.1 Existing power strategies for HPC

In the HPC world, there are several strategies that tend to mitigate both the waste of computing ressources and power consumed. The observation is always the same: except for the benchmarks specially developed to reach the peak performance of a system, real applications barely use 50% of ressources [69]. Actually, we can distinguish two profiles of workloads: memory bandwith bound and compute bound applications. To overcome the tradeoff between power and performance in HPC applications, co-scheduling two applications with different profiles shows that the runtime can be decreased by 28% and the energy consumption by 12%, respectively, compared to the best case possible execution

[70]. This solution sounds pragmatical but it decreases the workload performance and so might not suit some HPC use-cases. Also, besides the well known DVFS technique, Power capping, instead of managing the processor's frequency directly, offers the user to simply specify a time window with a power bound, and the hardware guarantees that the average power will not exceed the specified bound over each window [71]. However, defining a power budget for a workload and letting the system fend for itself to meet this budget could be a good concept but it seems not to be mature enough yet. Actually, power clamping can lead to subtle and specific problems with HPC applications. Indeed, the processors manufacturing process inevitably introduces small variations in terms of power consumption [72]. Then, placing a power budget on the processors moves the variation from power (each processor must now operates at a specified number of watts) to performance (if at least one less-efficient processor is being very slower at the specified power bound, the entire massivelly parallelized application can be fatally desynchronized and therefore deliver a poor performance) [71].

Furthermore, in [71] the authors make a harsh criticism of research in power-aware supercomputing focusing on trading a loss of performance for energy savings. This is a fundamental discussion when debating about power in the HPC domain. Actually, the main argument being that this approach doesn't match well with the goal of supercomputer stakeholders, which is to make an existing machine run as fast as possible (in other words, never accept to loose performance, to gain power !). Authors even suggest to stop measuring the utilization as a percentage of node-hours, but rather as a percentage of maximum watts used. Indeed, when an application is well optimized to match a specific supercomputer, the processors are efficiently used and have therefore a power consumption close to the maximum. For them, Intel's Running Average Power Limit (RAPL) technology should allow moving beyond power savings and into power scheduling in HPC, promoting DVFS replacement and treating power as a schedulable resource. In the same perspective, authors in [69] criticize the process of making individual nodes more efficient and measuring the results in terms of flops per watt. Indeed, traditional power design approaches use worst-case power provisioning: the total power allocated to the system is determined by the maximum power draw possible per node. Authors argue that overprovisioning hardware with respect to power combined with intelligent, hardware-enforced power bounds consistently leads to greater performance. Also, as the best configuration depends on application scalability and memory contention, leveraging system overprovisioning requires that applications use effective configurations which leads to an average speedup of more than 50% over worst-case provisioning.

At platform or HPC cluster level, the power management strategy is mostly integrated in the software that manages job requests for compute resources (compute nodes), like in SLURM (Simple Linux Utility for Resource Management). One of the major and most recent tools is introduced in [73] with a run-time system called 'Conductor'. Conductor utilizes non uniform power distribution, RAPL, DVFS, and DCT (Dynamic Concurrency Throttling) for optimizing HPC application performance under a power constraint. It is supposed to choose and adapt configurations dynamically based on application characteristics, resulting in better execution on a number of cases. The configuration analyzer dynamically selects the optimal thread concurrency level and DVFS state subject to a hardware-enforced power bound. Its adaptive power balancing scheme efficiently determines where critical paths are likely to occur so that more power is distributed to those paths. Encouraging results show a best-case performance improvement of up to 30%, and average improvement of 19.1%.

5.1.2 OS based strategies

Dynamic voltage frequency scaling (DVFS) as introduced in [74], became an ubiquitous approach for processor power management and accepted as a standard technique to reduce power and energy consumption. There is an effective relationship between frequency and power because the CPU dynamic power is proportional to the clock frequency and to the square of the supply voltage. CPU voltage being itself related with the level of the intended clock frequency. The implementation of DVFS at operating system level is known as Linux CPUfreq governors: userspace, performance, powersave, ondemand, conservative, and recently schedutils. Those drivers are mutually exclusive, they depend on the type of DVFS technology deployed at hardware or processor level and each of them has its own unique behavior, purpose, and suitability in terms of workload. Mostly, these OS based strategies are designed for general purpose usage. They are expected to be widely used and generally applicable but the counterpart is that they basically make coarse ajustment of processing resources to the global workload and are often inefficient too fine tune the system to actual application variations and particularities. The typical example is that of video coding where an onDemand governor will end up switching to the maximum processor frequency for the full duration of the sequence, while there exist execution variations at application level (e.g. intra-coded versus predicted frames) that can be exploited to adapt the best processor frequency to the actual processing requirements. These capabilities have been investigated in the field of embedded systems to develop more efficient power strategies (e.g. deadline scheduling, low

power scheduling, energy aware scheduling). The objective therefore is to investigate the extent to which it can be used to improve specifically on HPC processing constraints [75, 76].

5.1.3 Energy efficiency improvement

In the embedded world, the goal is generally to minimize the absolute energy required to process an application, more or less regardless of performances, in a way to maximise battery life. The benefits can be represented in measures of energy efficiency such as the energy-delay product. The problem is different in HPC because the absolute requirement is to run the application at the maximum possible performance. This implies that the aim here is to save power as long as it does not affect performances, or at least accepting a tolerable degradation of performance. A first intuitive application of this is to exploit the variations of thread execution times. An essential condition for any energy efficient processing system is load balancing. Several threads running on multiple computing resources will inevitably terminate at different times. Therefore the global performance is determined by the termination of the slowest thread, and DVFS can be used to slow down the execution of faster threads. The reductions of processor frequencies decrease the total power consumption of the application, while keeping the same performance level, therefore the global performance per Watt of the system. This first HPC strategy will be referred to as Soft big.LITTLE strategy in the following. Actually, in opposition to the official ARM big.LITTLE technology, where the size of heterogeneous cores (big and LITTLE) are hardly defined, the concept here introduces more flexibility at software level among homogeneous multicore processors.

A second interesting application of HPC dedicated strategies is related to communications. Because of the inherent large number of parallel threads, the situation where a global final processing gathering the final result is likely to occur. This scenario often involves communications and synchronisation mechanisms which does not require to run the processors at full clock rate. This second strategy will be referred to as Blocking point strategy in the following. Since communication and synchronisation primitives are made available in libraries, the achievement of a Blocking point strategy can be integrated in libraries, it is transparent to the application and eases greatly the use of the strategy. These considerations introduce two approaches suited with the absolute performance constraints of HPC operating i) at application level and ii) at library level to better process massively parallel workloads. The soft big.LITTLE strategy is suitable in the case where the size of each thread in terms of number of operations or cycle can

be predicted. The second one would be appropriate in the case where DVFS latencies are much smaller than the time of a point-to-point communication between two nodes. Both strategies could be combined to provide a fine grain adaptation for a specific workload in massively parallel compute nodes. In the following, we investigate the efficiency of both proposed strategies by simulation using GEM5 and real experimentation with a representative platform (Cavium ThunderX2 96 cores). We introduce GEM5 and the hardware configuration setup for these simulations, in a way to assess first the relevance of existing strategies (Linux governors) and then compared to the two proposed HPC strategies.

5.2 Evaluation of OS power strategies

5.2.1 Simulation framework

The section below describes our simulation framework, benchmarks and investigates the relationship between the processor frequency and its power consumption.

5.2.1.1 GEM5

Gem5 is an open source software for the simulation of hardware architectures with different specific components, such as the unique machine-type, caches, memories and buses. The simulations addressed here are made in full system mode, which means that the platform boots a Linux Operating System. It requires compiling a specific linux kernel with cpufreq drivers, platform related device tree source and bootloaders to match with our design. When starting those experiments, by default DVFS was not supported for ARMv8 in Gem5. To enable DVFS for each core, a DVFS-Handler module have been developped and so a new machine type that supports several clock and voltage domains per cluster (see appendix1 **??**).

5.2.1.2 Benchmarks

The applications used to illustrate the improvement potential of the two proposed strategies are based on floating point matrix multiplications (128x128, 256x256, 512x512, 1024x1024). This choice of a simple workload avoids sub-optimization of the source code and allows an easy parallelization over the cluster. Two configurations are used for parallel processing. In the first one, each thread processes the full multiplication of two matrixes. Therefore N threads process the multiplication (MxM) of 2*N matrixes in parallel using N cores. We can play with M (size the matrix) for different purposes. The

first configuration will be referred to as "Nparallel_MxM_matrix_products". The second one called "distributed_matrix_product_MxM", is a single matrix multiplication MxM distributed among N threads. Each thread is responsible to process (M*M)/N elements for the resulting matrix. An element at (a,b) position being the product of the a^{th} line of the first matrix and b^{th}colunm of the second matrix. At the beginning, the master thread initializes two matrixes and each thread knows which lines to process according to its rank. For each itteration, all the other threads send their processed elements to the master (using MPI_Send). Then, the master thread receives one element after another (using MPI_Recv) and aggregates those partial results in their respective position in the resulting matrix, and so on until the last one. Therefore there are (N-1) senders and only 1 receiver. MPI_Send being a blocking operation, the sender waits for an acknowledgement from the receiver to continue. This delay depends on the position of the couple sender/receiver in the cluster. When they are in the same compute node, this is faster as it is similar to a simple memory sharing. When they are in different compute nodes, it depends then on the network and the size of the data sent.

5.2.1.3 Energy efficiency evaluation

As stated in section 5.1.3, in HPC the energy efficiency improvements are generally reported in terms of performance per Watt. In our simulations, we will address power by monitoring clock frequencies, as gem5 only provides a generic ARMv8 core model from which power extrapolation could be sensless. However the consummed power is strongly related to frequency of the cores:

$P_{dynamic} = A.C.v^2.f$

Where, A is the percentage of active gates; C is the total capacitance load; v is the supply voltage; f is the processor frequency. Then, the accumulated energy is:

$E = \sum P_{dynamic} * dt$

The instructive figure 5.1 shows in a practibable way this relation. These results are otbained when running a stress program on the Cavium's ThunderX2 ARMv8 processor. The measured frequency and core power evolve together in a very linear fashion.

5.2.2 Simulation results

The results of figure 5.2 adresses the evaluation of existing OS based power strategies (Linux governors) using previous simulation framework. They report the variations of frequency over time for the benchmark "Nparallel_256x256_matrix_products" where N, the number of cores (clock domains) varies from 2, 4, 8 and 16. These four platform

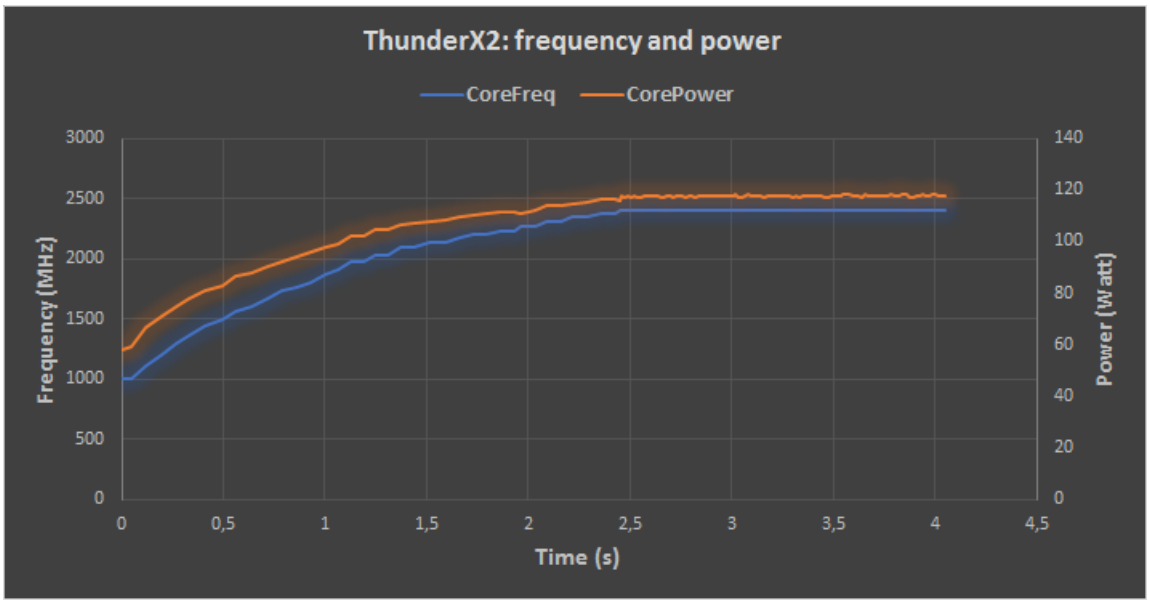


Figure 5.1: Frequency and Power on Cavium's ThunderX2 processor

configurations are used to assess the efficacy of typical standard power strategies in the situation of large parallel workloads and to see how they scale. The blue trace reports the average frequency for the N clock domains, for the sake of clarity. In addition, the mean value of this frequency trace is computed (in orange) as a measure of the expected reduction of the average power consumption (as shown in previous section) and therefore, of the performance per watt ratio. These variations are measured when running the benchmark under five Linux standard governors: Performance, Ondemand, Schedutil, Conservative and Powersave.

All the results are similar to the expected behaviors. The strategies of Performance and Powersave governors are very straightforward by keeping frequency at their maximum and minimum value (respectively) over the full duration of the benchmark. The execution time is very stable whatever the number of cores because each thread runs an independent matrix product of the same processing complexity. Obviously these two strategies are basic and certainly not the effective at all for High Performance Computing (Performance consumes maximum power, and Powersave is not fast at all).

Conservative and ondemand governors are more adaptive. The "conservative" governor, much like the "ondemand" governor, sets the CPU frequency depending on the current system load. Actually, the conservative governor is conservative in terms of allocated compute ressource. Compared with the ondemand governor, CPU speed increases and

decreases gradually rather than jumping to the maximum speed the moment there is any load on the CPU. Both are tweaked in the same manner and we can observe practically in the results how they differ in this way of setting the CPU frequency. The first one is progressive while the second one can switch more quickly between minimun and maximum frequency values. Adaptation of the Ondemand governor is a bit more reactive with the actual workload. This is visible in particular at both thread synchronisation barriers, before and after processing the parallel matrix multiplication threads: domains are mostly set to minimal frequency at the begining, when the program mostly waits for all threads to be created, and at the end, when the program reaches the termination of the slowest thread. However, in the strict processing phase of the matrix multiplication, a clock domain is set at maximum frequency for both governors from the actual start of the multiplication to the termination of the slowest thread. As the number of clock domains grows, we therefore observe an increase of both the average frequency and execution time, resulting from synchronisations with the last terminating thread.

Similar to ondemand and conservative, the schedutil governor operates from a workload estimation with the difference that the information comes directly from the kernel scheduler to try to better adapt CPU frequency in a timely manner. It aims at better coupling power policies with the Linux kernel scheduler, as load estimation is achieved through the scheduler's Per-Entity Load Tracking (PELT) mechanism which also provides information about the recent load [77–79]. As a result, the variations of frequencies are a bit more relevant, which translates into more regularity (less variation) on the frequency profiles. Nevertheless, the behavior of the strategy within the actual processing phase sets the frequencies at their maximum over the full duration. Consequently, there is no power saving like Ondemand and Conservative governors in the actual thread execution phase (outside both synchronisation barriers introduced only for simulation purposes).

These simulations are very useful to verify the correctness of the simulation procedure since the traces of frequencies are just as expected. In all cases (except powersave which is not relevant for performance purpose), frequency is always set at the maximum value for the duration of a job. Also, while this is appropriate in terms of performance, it leads to maximum power consumption regardless of actual runtime variations that occurs in practice that can be used to further adapt processing power with clock scaling. These opportunities are explored on two practical case studies in the following subsection.

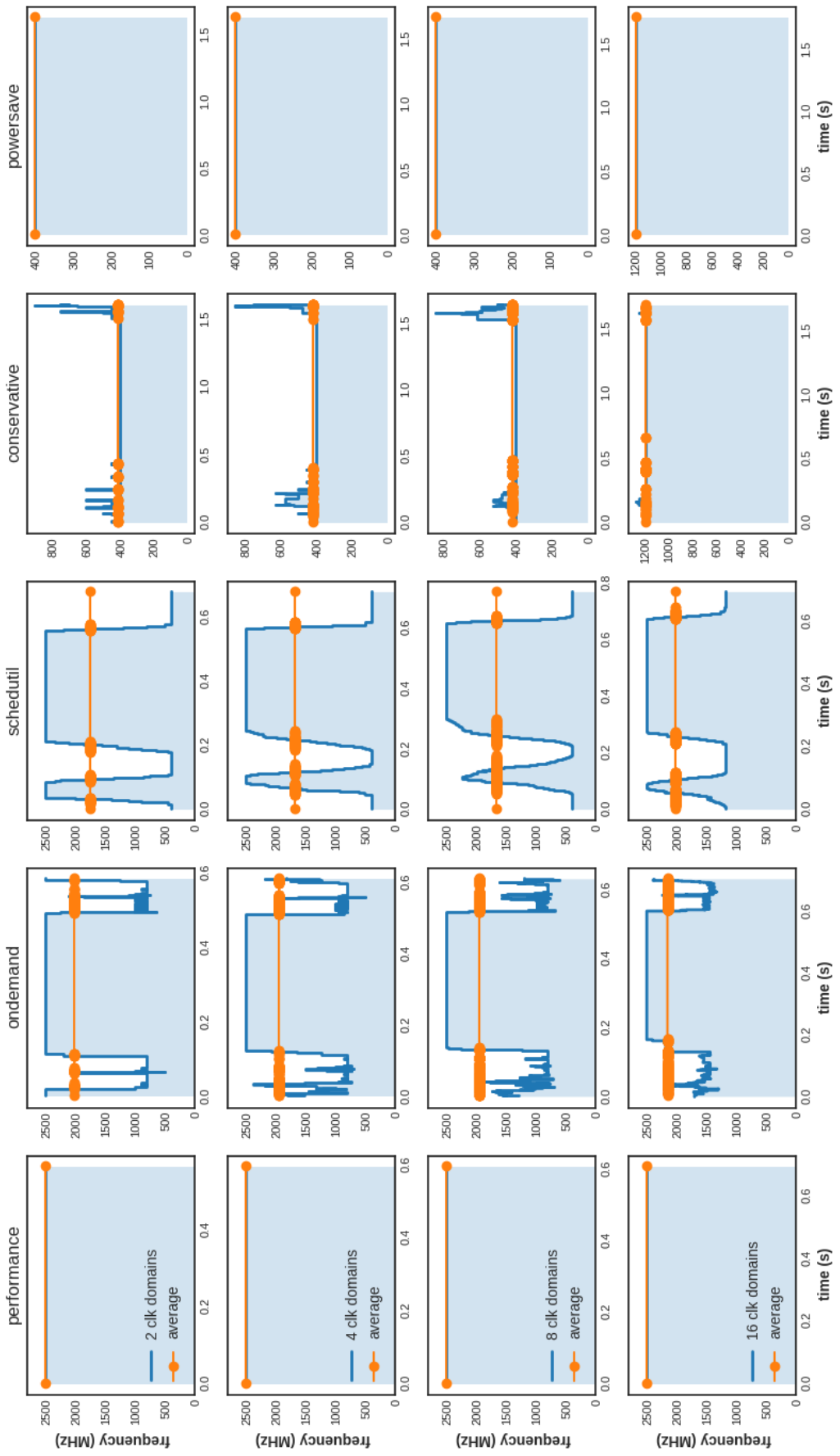


Figure 5.2: Efficicency evaluation of Linux governors (config: Nparallel_256x256_matrix_products)

5.3 Power strategies for HPC

DVFS is decribed in [80] as an ubiquitous technique for CPU power management in modern computing systems. Hower, reducing processor frequency/voltage leads to a decrease of CPU power consumption and an increase in the execution time. The goal in this section, is to try to make the second affirmation being false. In other words, try to save power and keeping the same performance. The idea is to take advantage of some variations inherent to the execution, algorithm, or parallelization of an application to save power consumption of the processors using DVFS. However, to obtain a successful energy-performance trade-off for very large scale parallel applications, it is necessary to take into account both application and platform characteristics, such as the application sensitivity to frequency scaling or the processor latency when switching between two frequencies.

5.3.1 Soft big.LITTLE strategy simulations with GEM5

In previous gem5 simulations of standard existing governors, the system could greatly benefit from a finer adjustment of frequency. An interesting possibility lies for example in slowing down fast threads instead of being run at full speed, in a way to match with the termination of the slowest (odd) thread. This would allow to reduce power of the corresponding domains, without any impact on performance since the execution time of the slowest thread would remain the same. This strategy is referred to as "soft big.LITTLE" in the following 5.1.3. This strategy can be applied when you can predict the latency of each thread. It consists in attributing to each core a frequency weighted by the latency of the corresponding thread or process. Slow threads will run on the cores with higher frequencies and it's the other way around for fast threads. Considering two parallel threads, T_1 and T_2, where T_1 will execute about I_1 instructions and I_2 for T_2 and assuming $I_2 \succ I_1$,which means that T_2would be slower if they are executed on two cores at the same frequency, T_2 defines the global execution time. The Soft big.LITTLE strategy provides an adjustment by making T_1 as slow as possible, but not slower than T_2, in a way to save power with the same performance. Therefore the frequencies must be computed in respect with $\frac{F_2}{F_1} \leq \frac{I_2}{I_1}$. For example if T_2 executes twice more instructions than T_1, the ratio between the chosen frequencies must be at the most of a factor 2 to save power without infringing the workload performance.

Figure 5.3 shows the frequencies of four clock domains when running two parallel 256x256 and 128x128 matrix products under the Soft big.LITTLE strategy. To produce controlled

variations of execution time, the multiplication process is split into even and odd threads where even threads are running a matrix multiplication twice bigger than odd threads. The number of operations (additions and multiplications) required in a product of two square matrices NxN is about $n^2(n + (n - 1))$, when $n = N$. Therefore, we must set the frequency of odd threads (N=128) in such a way that the ratio with even threads (N=256) is less or equal 8. The objective is then to reduce the differences in execution time thanks to DVFS. In gem5 simulations, we will deal with the couple of frequencies 2500 MHz for even threads and 400 MHz for odd threads. The corresponding ratio of frequencies is $6.25 \leq 8$.

The three traces on the left reflect the execution for standard OS power strategies (performance, ondemand and schedutil). The average frequency is about 2200 MHz for schedutil and ondemand, while it stands at 2500MHz for performance as expected. the execution times are similar in the three cases which means that schedutil and ondmand are a bit more energy efficient than performance. Comparing with the Soft big.LITTLE strategy on the right, we observe a clear drop of frequencies which accurately corresponds to an improvement of 20% with the performance governor itself and 36% with the ondemand governor (compared to 2500 MHz for performance previously). For some reason the efficiency of schedutil governor decreases, but we can see on the right side of figure 5.3 that the average frequency is globally reduced, which means that we have saved power according to the approach adopted in 5.2.1.3.

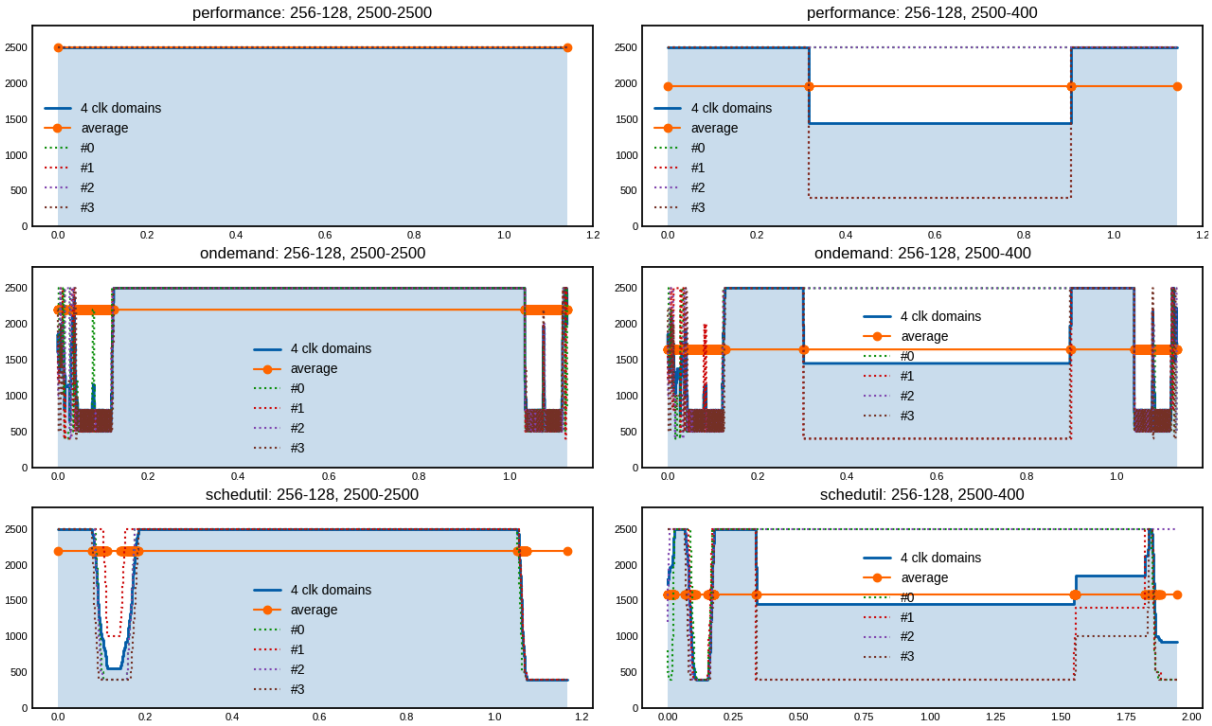


Figure 5.3: 2parallel_256x256_matrix_products and 2parallel_128x128_matrix_products

As it is equally important to keep an identical level of performance, left of figure 5.4 reports comparative execution times of OS power (blue plot) and Soft big.LITTLE strategies (green plot) for an increasing level of parallelism (a clock domain is composed of one core in these simulations). On the right side, profiles of frequencies (related to dynamic power) and the corresponding improvements are reported using the same parallel 256 and 128 matrix products. The plots on the left show similar execution times (when green and blue dots overlap), when on the left the red line shows the amount of frequency (power) reduction up to 40%. Therefore, we observe similar execution times with less power consumption where the strategy is applied. This is precisely the goal when improving the compute efficiency of a workload.

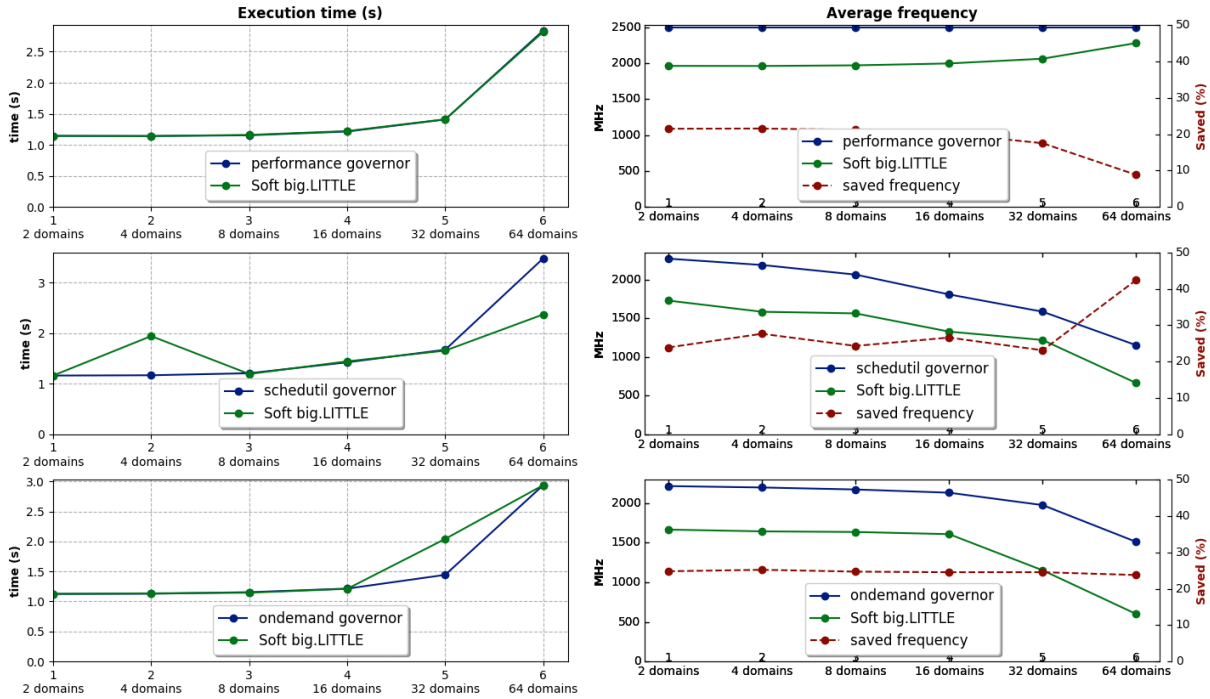


Figure 5.4: Soft big.LITTLE strategy scalability

5.3.2 Blocking point strategy simulations with GEM5

This second strategy is based on saving power at some blocking points of a workload (barriers or synchronization points) instead of running uselessly the system at full speed. Actually, when developing code intended for HPC, we could also take advantage of inherent parallelism management issues. Indeed, processing large parallel applications implies thread communications and synchronisations that does not need to operate the cores at the highest frequency. For example a core waiting for a result can be set at minimum frequency without any impact on the execution time, and this could be implemented in the library providing thread management facilities for instance. To evaluate the potential power savings (without losing in terms of execution time), we address an example based on MPI. The MPI_Send mechanism is a blocking transaction that allows to send a message to one or several threads, and wait for acknowledgement before proceeding with the rest of the execution. The power saving evaluation approach is to setup a couple of frequencies (Max, Min) in the folowing way: (1) set the Min frequency before MPI_Send and the Max after MPI_Send. Measurements are carried out considering seven pairs of (Max, Min) frequencies (2500-2500, 2500-2300, 2500-2000, 2500-1800, 2500-1200, 2500-800, 2500-400 MHz), using a 1024x1024 distributed matrix

product parallelized with four threads on four clock domains. We can observe in figure 5.5 that the average frequency decreases from 2500MHz (in configuration 2500-2500) to 1600MHz (configuration 2500-400), which corresponds to 36% gain in terms of average power is compared to the reference execution without strategy (upper left of the figure). The execution times tend to increase as we switch to lower frequencies (13.2 seconds compared to 6.4 seconds without strategy). This is due to the limited number of threads (4) used in this experiment. The four threads are mapped on four cores in the same compute node. Communication times between threads are relatively fast in this case, and especially these latencies are smaller than that of switching frequencies. The impact on execution times comes from the fact that we are not in an ideal configuration for the Blocking Point strategy which requires larger levels of parallelism (involving inter compute nodes communications) to be efficient. In the following, we increase the number of threads (and cores) to further investigate this.

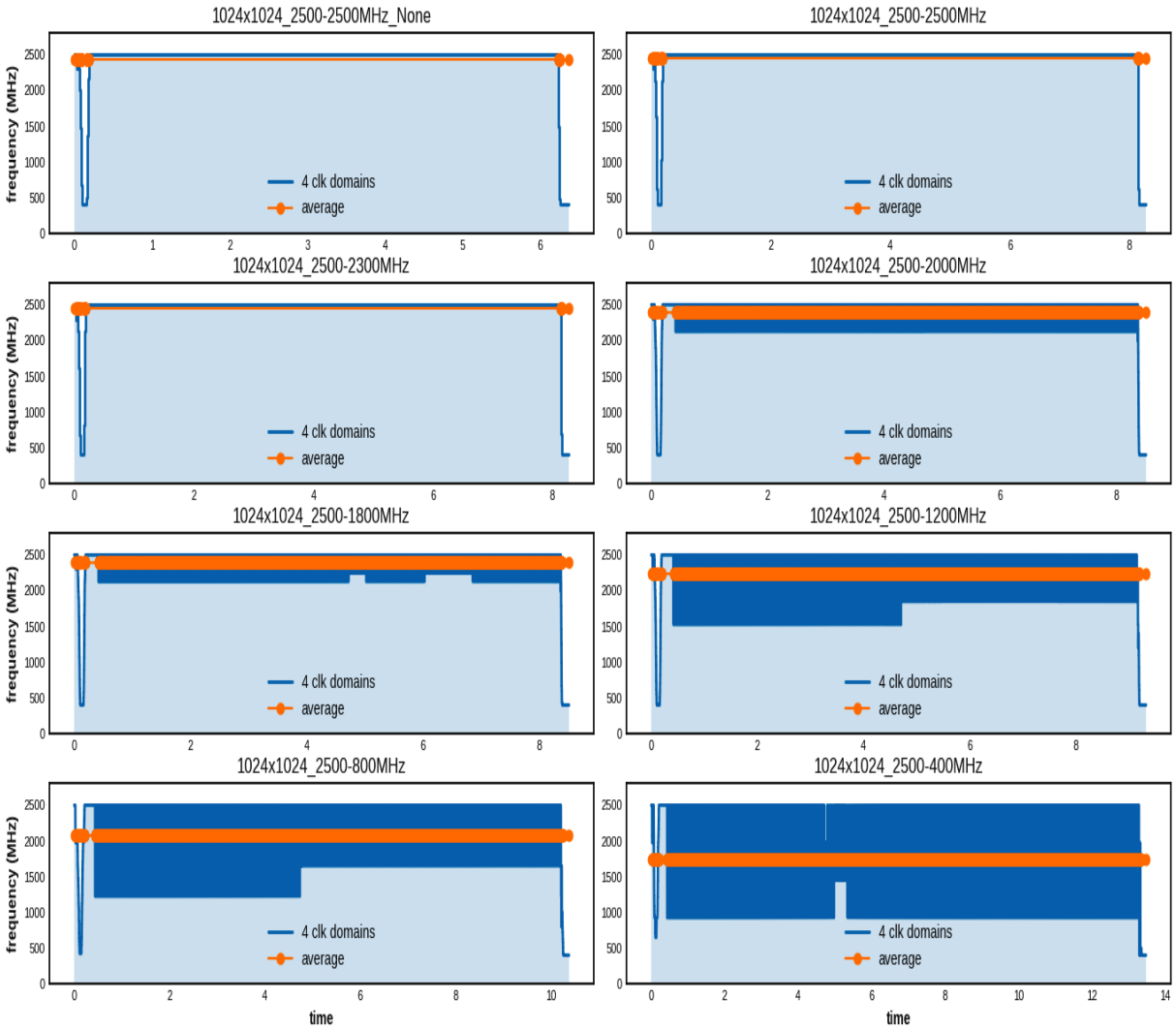


Figure 5.5: Before and After blocking point strategy: 4 clock domains

Within a compute node, MPI works mostly like simple memory sharing, thus with relatively fast communications. In a HPC cluster, compute nodes communicate through a compute network. The Network Interconnect Controller (NIC) of each compute node is connected to a level 1 switch, which can also be connected to a level 2 switch, and so on depending on the topology deployed for the cluster. As a result, communications between threads when different compute nodes are involved are significantly slower. As there was no ARM based cluster available at the time of this study, we further analysed the impact of such inter compute node communications on the Blocking Point strategy with gem5 simulations. The results of figure 5.6 show the effects of scaling up the number of cores (clock domains) from 2 to 64 within a single compute node. Running three benchmark configurations of 256x256, 512x512 and 1024x1024 matrix

products and considering seven pairs of frequencies as previously, we can now observe that performance is scaling better with the workload. Execution time is supposed to be divided by two when the number of computing ressources (cores) has doubled. On the left are represented the absolute values of the execution times and on the right, the normalized values to better figure out the slow down effects. The objective being to save power without increasing the execution time, we can note that the execution time tends to remain stable for important number of threads (32, 64). This remains true whatever the different pairs of switching frequencies and especially for highly parallel configurations (1024x1024). The results here show that we can realistically expect the blocking point strategy to be efficient for workloads deployed in a large HPC cluster, with very limited impact on performance. We verify the validity of these claims on a real system in the following.

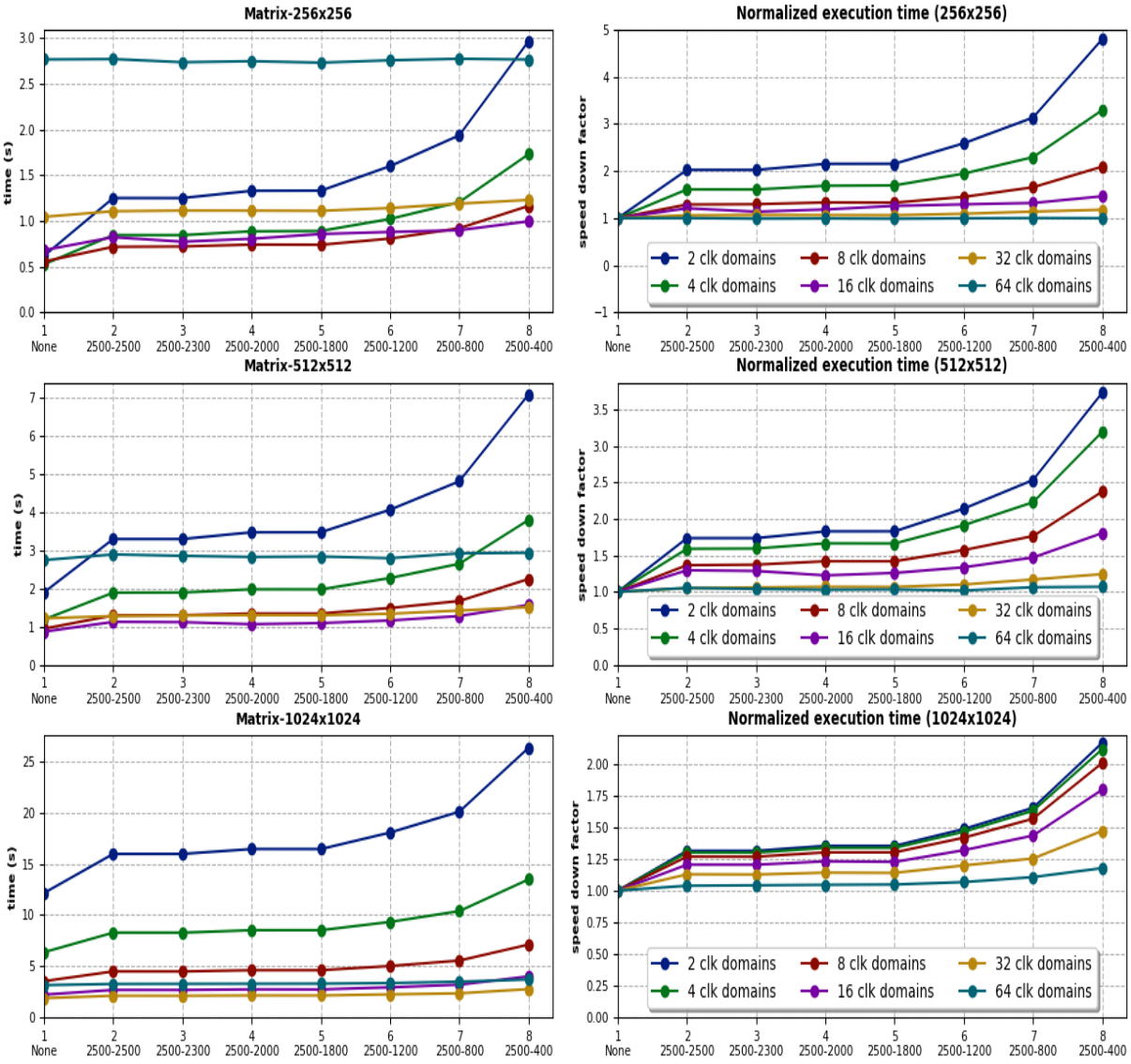


Figure 5.6: Scaling up Before and After blocking point strategy

5.4 Measurements with Cavium ThunderX2

5.4.1 Platform description

Cavium ThunderX2 is the lastest ARM based server processor available since the thesis has started. ThunderX2 is a family of 64-bit multi-core ARM server microprocessors introduced in early 2018. The SoC model is based on 64-bit ARM v8 based processors featuring a total of 96 cores organised in two sockets of 48 cores (CPU-A and CPU-B). The memory system supports up to 2 TiB of quad/hexa/octa-channel DDR4 2666 MT/s

memory which is made fully cache coherent across dual sockets using Cavium Coherent Processor Interconnect (CCPI2). This platform is a good fit for the evaluation and analysis of the actual benefits of the two proposed power management strategies as it also provides chip power measurement features allowing per socket analysis. The following experiments were made possible thanks to an early access to the Dibona platform produced for the European Mont-Blanc3 project.

5.4.2 Soft big.LITTLE strategy execution on ThunderX2

Figure 5.7 shows the evolution of the Cavium ThunderX2 chip power over time using the Nparallel_MxM_matrix_products benchmark used previously for Gem5 simulations. Global and per socket (CPU-A, CPU-B) power consumption are addressed for six pairs of frequencies supported by the platform (2000-2000, 2000-1800, 2000-1600, 2000-1400, 2000-1200, 2000-1000 MHz). We have set the complexity of even threads (2560x2560 matrix product) and that of odd threads (2048x2048 matrix product) such that the execution time of even threads is exactly twice that of odd threads. This allows to ease the control of thread execution time, adjustment of frequencies, and analysis of the corresponding power savings.

Without strategy (corresponding to the 2000-2000 configuration), power traces show clearly the two phases of a multiplication: i) even + odd threads up to the half of the execution and ii) only odd threads until the end of execution at 280s. The growing effect of the strategy is visible in the following configurations (2000-1800, 2000-1600, 2000-1400, 2000-1200, 2000-1000 MHz) as we decrease the speed of fast even threads to progressively match 280s. In this way, the global application performance remain strictly unaltered (280s) despite slowing down even threads. Figure 5.8 provides a zoom on the global power consumption profiles which allows to better measure the relative improvements of the soft big.LITTLE strategy, especially when we compare the optimal use of the strategy (blue trace) with the absence of strategy (pink trace). We can observe that better is the couple of frequencies implementing the Soft big.LITTLE strategy, better is the spread of power consumption while keeping similar execution time. This is what we expected as intruduced in the subsection 5.3.1 and demonstrated by simulations results using Gem5 (see figure 5.3). Comparing the [2000,2000]MHz and [2000,960]MHz configurations, the average power is reduced from 287W to 262W (>8% of power reduction).

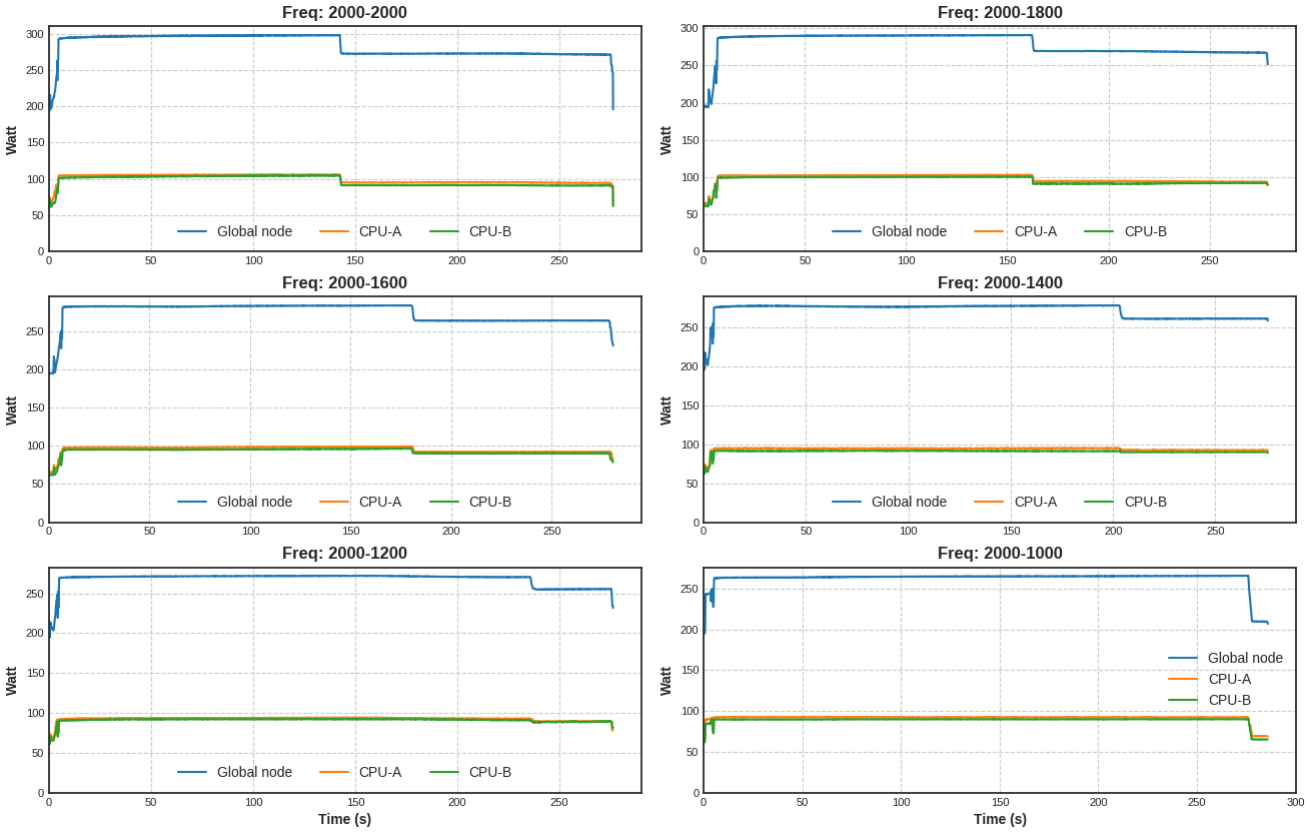


Figure 5.7: Power profiles of the Nparallel_MxM_matrix_products on Cavium ThunderX2

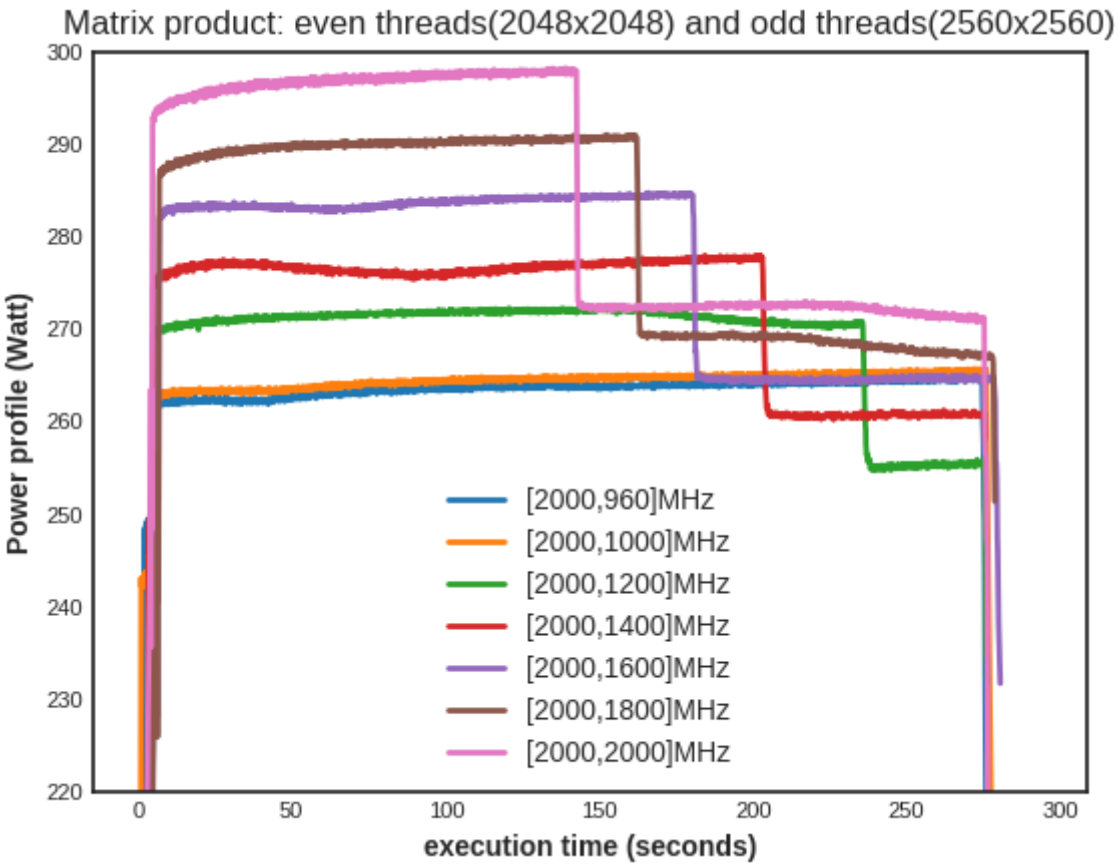


Figure 5.8: Nparallel_MxM_matrix_products global power summary

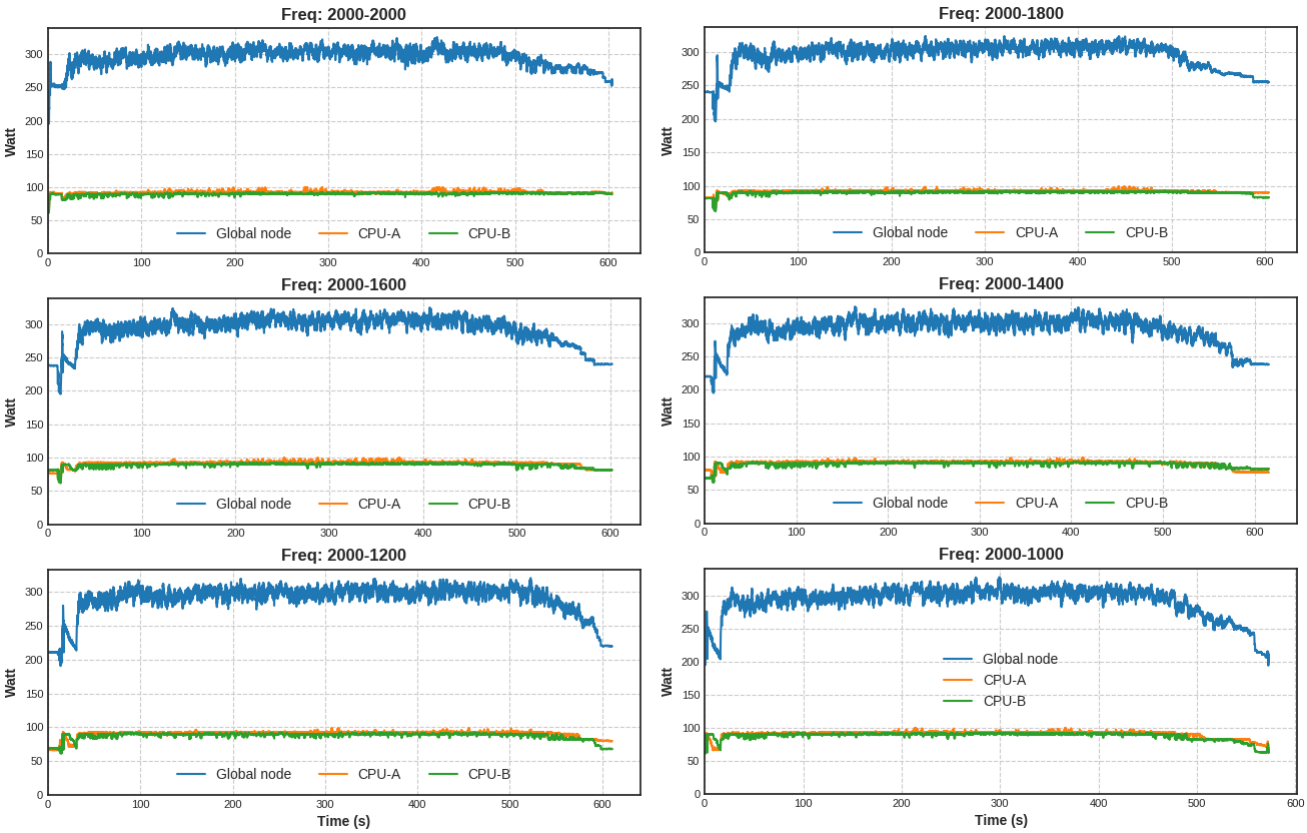


Figure 5.9: Power profiles of the distributed_matrix_product_10240x10240 for 1 node

5.4.3 Blocking point strategy execution on ThunderX2

The results of figure 5.9 come from the benchmark distributed_matrix_product_MxM described previously for Gem5 simulations. The principle of the strategy being to decrease frequency at the end of a thread for communication and synchronisation purposes, the gains depend on the cost of the associated latencies. The experiment is set at a single node level, where communications are performed through shared memory.

In this case, communication latencies are lower than that of changing frequency and there is no benefit the Blocking point strategy. This is what can be seen in the results where power traces keep similar global profiles around 300W whatever the frequency drop.

However, for two nodes (figure 5.10), slower inter node communications are involved. Active waiting in idle state at thread synchonization points can therefore benefit from decreasing core frequencies. This is visible in the results where the synchronisation phase (from 0 to 620s) allow to decrease power from 245W (at 2000 MHz) to 190W (at 1000 MHz), saving more than 20% of power consumption on the synchronization phase. Pro-

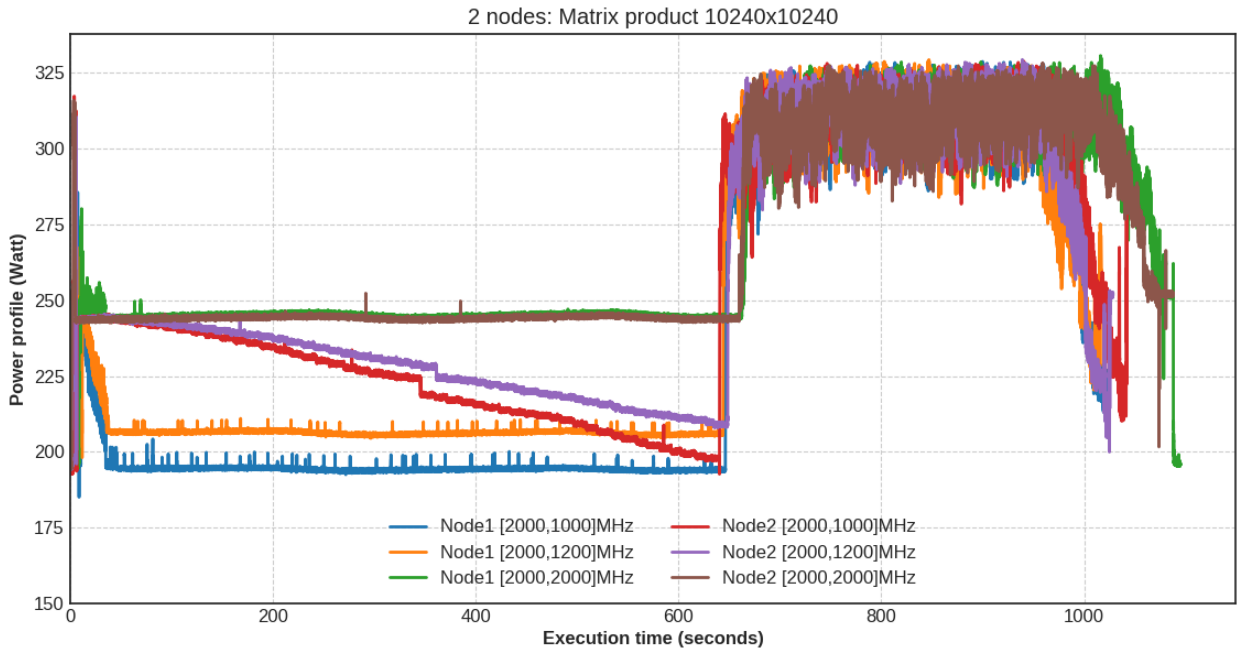


Figure 5.10: Power profiles of the distributed_matrix_product_10240x10240 for 2 nodes

cessing phase on the other hand occurs from 620 to around 1300s. The corresponding average power reduction of the distributed_matrix_product_10240x10240 benchmark on the full duration (synchronisation + processing) is 10%.

The question which needs to be addressed is to analyse how much these savings impact performances. Table 5.1 shows that even in the frequency configuration worst affecting performances ([2000, 1000]), the variation of execution time is less than 2%. Therefore these results show a promising potential of the Blocking point strategy to reduce power significantly, at negligible performance loss, in systems having high communication costs (i.e. most HPC highly parallel applications). Considering [2000,2000] and [2000,1000] configurations for the CPU-A power for instance, we can observe that at core level, this strategy leads to power saving from 90 Watt to 77.19 or more than 14% of power reduction.

Frequencies	[2000,1000]	[2000,1000]	[2000,1200]	[2000,1200]	[2000,2000]	[2000,2000]
Node ids	#1	#2	#1	#2	#1	#2
Execution time	1255.68	1256.99	1269.09	1260.17	1294.85	1279.82
Global power	241.21	256.44	247.66	259.47	270.72	270.09
CPU-A power	77.19	79.83	78.78	81.91	91.30	90.00
CPU-B power	78.24	83.57	79.83	84.23	92.89	87.50
Memory power	52.96	51.90	52.54	51.79	52.76	51.46
Infiniband power	12.13	12.93	12.13	12.93	12.12	12.94

Table 5.1: Execution time and power breakdown for Blocking point strategy

5.5 Conclusion

The intent of this chapter was to show the potential of more advanced power strategies to improve HPC energy efficiency. We debated about the goal of supercomputer stakeholders, which is to make an existing machine to run as fast as possible. Should we add "even if workloads are not efficient or cannot exceed algorithmic limitations" ? In light of existing power management, we proposed two specific strategies designed to better match the constraints of HPC processing, especially in the face of widespread workload based approaches in that matter. Investigating their potential power savings on realistic use cases let us conclude there is a global power saving potential of more than 14% at CPU level depending on application and platform parameters. It must be noted that the implementation aspects of these strategies are not fully defined as our focus was rather to investigate the potential of dedicated HPC power strategies, which is an original contribution of this thesis. Future works could investigate further implementation issues (like prediction for the soft big.LITTLE strategy), or other specific energy aware scheduling schemes for HPC (e.g. , dynamic reconfiguration) which is certainly a promising direction of research for energy efficient HPC, in the light of power efficiency improvement results of both strategies. These results, which have to be regarded as preliminary estimates, show nevertheless that there is a true potential in developing HPC dedicated strategies. These strategies show it is possible to build on specific HPC processing characteristics (DVFS load balancing, library synchronisation) and realize potentially a better performance per Watt ratio improvement.

Part III

CONCLUSIONS

Chapter 6

Conclusions and Perspectives

This work contributed to investigate the use 64-bit ARMv8-A cores to find significantly more processing and energy efficiency in future HPC exascale supercomputers. Three of the fundamental sides of the efficiency question have been adressed through the methodological, architectural and executive aspects. A first global outcome from this approach is the profit at the architectural level. The SoC partitioning scheme explored let the ability to exploit a large number of 64-bit ARMv8-A cores and reduce the complexity of cache coherence management at no perfomance cost, which represents a first direct contribution to complexity (thus power) reduction. In addition, this solution introduces other opportunites to capitalize on recent integration technologies (3D chip stacking, TSV, HBM) at less power cost.

On a methodological development point of view, the exploration approach defined and used illustrate some benefits of tightly-coupled Hardware/Software Co-design, a concept born in the embedded world, to better address specific Exascale constraints resulting from the expansion of the number of processing cores instead of increasing processor frequency. Software influences architecture at different levels that are not limited to user applications and algorithms, but also programming models and system software. These problematics led to a wider reflection and to consider runtime software and more precisely power management which is an important issue when it comes to power efficiency. The potential of more responsive power strategies tuned to the specific performance constraints of HPC was demonstrated and this original thesis contribution opens other perspectives. Overall, this study extends quite logically to pursue the investigations of methodical holistic approaches able to explore and implement a better match between software and architecture at development and execution levels, allowing both algortihm well tuning to the complex processing capabilities and the best fit with the applications

at runtime.

In the last lines of this thesis manuscript, one thing is clear: that the hardware and software ecosystem around the Arm technologies are now mature enough in comparison to the leading ones for HPC. The skepticism of the community at the beginning is replaced by a little bit of enthusiasm, with several encouraging preview conclusions of the new systems using the latest ARMv8-based processor (ThunderX2). At the European level, the EU has rectified the supercomputing strategy with the Europe Processor Initiative (EPI) consortium to deliver an exascale machine based on EU processor by 2023. However, generally speaking HPC is still about silicon technology. The next generation of CPU will be manufactured at 5nm process and today no one seriously argues about the fact that Moore's Law is getting close to fundamental physical limits as processor features are approaching the size of atoms. The post Moore computing research may offer several perspectives to post-exascale systems. Recent progress in quantum computing hardware with fully functional Quantum Processing Units (QPU) and associated ISA; will offer new directions for High Performance Computing. The challenge will be to integrate or combine QPUs in the silicon computational workflows [81–84].

Bibliography

[1] Sao-Jie Chen, Guang-Huei Lin, Pao-Ann Hsiung, and Yu-Hen Hu. *Hardware software co-design of a multimedia SOC platform*. Springer Science & Business Media, 2009.

[2] John Impagliazzo and John AN Lee. *History of Computing in Education: IFIP 18th World Computer Congress, TC3/TC9 1st Conference on the History of Computing in Education, 22-27 August 2004, Toulouse, France*. Springer, 2004.

[3] Erika Abraham, Costas Bekas, Ivona Brandic, Samir Genaim, Einar Broch Johnsen, Ivan Kondov, Sabri Pllana, and Achim Streit. Preparing hpc applications for exascale: challenges and recommendations. In *2015 18th International Conference on Network-Based Information Systems*, pages 401–406. IEEE, 2015.

[4] Cristóbal Camarero, Carmen Martínez, Enrique Vallejo, and Ramón Beivide. Projective networks: Topologies for large parallel computer systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):2003–2016, 2017.

[5] Edy Tri Baskoro, Mirka Miller, Ján Plesník, and Štefan Znám. Digraphs of degree 3 and order close to the moore bound. *Journal of Graph Theory*, 20(3):339–349, 1995.

[6] Mariano Benito, Enrique Vallejo, and Ramón Beivide. On the use of commodity ethernet technology in exascale hpc systems. In *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, pages 254–263. IEEE, 2015.

[7] Nathan DeBardeleben, Sean Blanchard, David Kaeli, and Paolo Rech. Field, experimental, and analytical data on large-scale hpc systems and evaluation of the implications for exascale system design. In *2015 IEEE 33rd VLSI Test Symposium (VTS)*, pages 1–2. IEEE, 2015.

[8] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. Multi2sim: a simulation framework for cpu-gpu computing. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 335–344. IEEE, 2012.

[9] Nathan DeBardeleben, Sean Blanchard, Laura Monroe, Phil Romero, Daryl Grunau, Craig Idler, and Cornell Wright. Gpu behavior on a large hpc cluster. In *European Conference on Parallel Processing*, pages 680–689. Springer, 2013.

[10] Brian Kocoloski, Leonardo Piga, Wei Huang, Indrani Paul, and John Lange. A case for criticality models in exascale systems. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 213–216. IEEE, 2016.

[11] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. In *Journal of Physics: Conference Series*, volume 78, page 012022. IOP Publishing, 2007.

[12] Daniel Dauwe, Sudeep Pasricha, Anthony A Maciejewski, and Howard Jay Siegel. A performance and energy comparison of fault tolerance techniques for exascale computing systems. In *2016 IEEE International Conference on Computer and Information Technology (CIT)*, pages 436–443. IEEE, 2016.

[13] Saurabh Hukerikar and Christian Engelmann. Havens: explicit reliable memory regions for hpc applications. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2016.

[14] Nunziato Cassavia, Sergio Flesca, Michele Ianni, Elio Masciari, Giuseppe Papuzzo, and Chiara Pulice. A peer to peer approach to efficient high performance computing. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 539–542. IEEE, 2017.

[15] Simon Pickartz, Stefan Lankes, Antonello Monti, Carsten Clauss, and Jens Breitbart. Application migration in hpcâa driver of the exascale era? In *2016 International Conference on High Performance Computing & Simulation (HPCS)*, pages 318–325. IEEE, 2016.

[16] Carlos Rosales, Antonio Gómez-Iglesias, Si Liu, Feng Chen, Lei Huang, Hang Liu, Antia Lamas-Linares, and John Cazes. Performance prediction of hpc applications on intel processors. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1325–1332. IEEE, 2017.

[17] Kavitha Chandrasekar, Xiang Ni, and Laxmikant V Kale. A memory heterogeneity-aware runtime system for bandwidth-sensitive hpc applications. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1293–1300. IEEE, 2017.

[18] Ivy Bo Peng, Roberto Gioiosa, Gokcen Kestor, Pietro Cicotti, Erwin Laure, and Stefano Markidis. Exploring the performance benefit of hybrid memory system on hpc environments. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 683–692. IEEE, 2017.

[19] Rizwan A Ashraf, Roberto Gioiosa, Gokcen Kestor, and Ronald F DeMara. Exploring the effect of compiler optimizations on the reliability of hpc applications. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1274–1283. IEEE, 2017.

[20] Antoni Portero, Jiri Sevcik, Martin Golasowski, Radim Vavrík, Simone Libutti, Giuseppe Massari, Francky Catthoor, William Fornaciari, and Vít Vondrák. Using an adaptive and time predictable runtime system for power-aware hpc-oriented applications. In *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*, pages 1–6. IEEE, 2016.

[21] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. Knights landing: Second-generation intel xeon phi product. *Ieee micro*, 36(2):34–46, 2016.

[22] Tiffany Trader. China debuts 93-petaflops âsunwayâ with homegrown processors, 2016.

[23] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, et al. The sunway taihulight supercomputer: system and applications. *Science China Information Sciences*, 59(7):072001, 2016.

[24] Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12. IEEE, 2013.

[25] Katie Roberts-Hoffman and Pawankumar Hegde. Arm cortex-a8 vs. intel atom: Architectural and benchmark comparisons. *Dallas: University of Texas at Dallas*, 5, 2009.

[26] Edson L Padoin, Daniel AG de Oliveira, Pedro Velho, and Philippe OA Navaux. Time-to-solution and energy-to-solution: a comparison between arm and xeon. In *2012 Third Workshop on Applications for Multi-Core Architecture*, pages 48–53. IEEE, 2012.

[27] Edson Luiz Padoin, Laércio Lima Pilla, Márcio Castro, Francieli Z Boito, Philippe Olivier Alexandre Navaux, and Jean-François Méhaut. Performance/energy trade-off in scientific computing: the case of arm big. little and intel sandy bridge. *IET Computers & Digital Techniques*, 9(1):27–35, 2014.

[28] Nikola Rajovic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones, and Alex Ramirez. Tibidabo: Making the case for an arm-based hpc system. *Future Generation Computer Systems*, 36:322–334, 2014.

[29] Grisenthwaite Richard. Armv8-a technology preview, 2011.

[30] Tim Northover. Aarch64: Armâs 64-bit architecture, 2012.

[31] Catalin Marinas. Linux on aarch64 arm 64-bit architecture, 2012.

[32] Arm Ltd. Arm architecture reference manual, 2005.

[33] Arm Ltd. Armv8 instruction set overview, 2011.

[34] Matt Humrick. Exploring dynamiq and armâs new cpus: Cortex-a75, cortex-a55, 2017.

[35] Anastasiia Butko, Abdoulaye Gamatié, Gilles Sassatelli, Lionel Torres, and Michel Robert. Design exploration for next generation high-performance manycore on-chip systems: Application to big. little architectures. In *2015 IEEE Computer Society Annual Symposium on VLSI*, pages 551–556. IEEE, 2015.

[36] Nigel Stephens. Armv8-a next-generation vector architecture for hpc. In *2016 IEEE Hot Chips 28 Symposium (HCS)*, pages 1–31. IEEE, 2016.

[37] Nigel Stephens, Stuart Biles, Matthias Boettcher, Jacob Eapen, Mbou Eyole, Giacomo Gabrielli, Matt Horsnell, Grigorios Magklis, Alejandro Martinez, Nathanael Premillieu, et al. The arm scalable vector extension. *IEEE Micro*, 37(2):26–39, 2017.

[38] Federico Angiolini, Jianjiang Ceng, Rainer Leupers, Federico Ferrari, Cesare Ferri, and Luca Benini. An integrated open framework for heterogeneous mpsoc design space exploration. In *Proceedings of the conference on design, automation and test in Europe: proceedings*, pages 1145–1150. European Design and Automation Association, 2006.

[39] Wai Kai Chen. *The electrical engineering handbook*. Elsevier, 2004.

[40] Ming-yu Hsieh, Arun Rodrigues, Rolf Riesen, Kevin Thompson, and William Song. A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):63–68, 2011.

[41] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, Rolf Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. The structural simulation toolkit. *SIGMETRICS Performance Evaluation Review*, 38(4):37–42, 2011.

[42] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. ACM, 2009.

[43] Deepak C Sekar, Azad Naeemi, Reza Sarvari, Jeffrey A Davis, and James D Meindl. Intsim: A cad tool for optimization of multilevel interconnect networks. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 560–567. IEEE, 2007.

[44] Deepak Chandra Sekar. *Optimal signal, power, clock and thermal interconnect networks for high-performance 2d and 3d integrated circuits*. PhD thesis, Georgia Institute of Technology, 2008.

[45] Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne Hambrusch, and Rahul Shah. Orion 2.0: native support for uncertain data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1239–1242. ACM, 2008.

[46] Kevin Skadron, Mircea R Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling

and implementation. *ACM Transactions on Architecture and Code Optimization (TACO)*, 1(1):94–125, 2004.

[47] Anastasiia Butko, Rafael Garibotti, Luciano Ost, Vianney Lapotre, Abdoulaye Gamatie, Gilles Sassatelli, and Chris Adeniyi-Jones. A trace-driven approach for fast and accurate simulation of manycore architectures. In *The 20th Asia and South Pacific Design Automation Conference*, pages 707–712. IEEE, 2015.

[48] Sudip S Dosanjh, Richard F Barrett, DW Doerfler, Simon D Hammond, Karl S Hemmert, Michael A Heroux, Paul T Lin, Kevin T Pedretti, Arun F Rodrigues, TG Trucano, et al. Exascale design space exploration and co-design. *Future Generation Computer Systems*, 30:46–58, 2014.

[49] Pablo De Oliveira Castro, Chadi Akel, Eric Petit, Mihail Popov, and William Jalby. Cere: Llvm-based codelet extractor and replayer for piecewise benchmarking and optimization. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(1):6, 2015.

[50] Alain Greiner. Tsar: a scalable, shared memory, many-cores architecture with global cache coherence. In *9th International Forum on Embedded MPSoC and Multicore (MPSoCâ09)*, volume 15, 2009.

[51] Tiffany Trader. China plans 2019 exascale machine to grow sea power, 2017.

[52] Mitsuo Yokokawa, Fumiyoshi Shoji, Atsuya Uno, Motoyoshi Kurokawa, and Tadashi Watanabe. The k computer: Japanese next-generation supercomputer development project. In *IEEE/ACM international symposium on low power electronics and design*, pages 371–372. IEEE, 2011.

[53] Yuuichirou Ajima, Tomohiro Inoue, Shinya Hiramoto, and Toshiyuki Shimizu. Tofu: Interconnect for the k computer. *Fujitsu Sci. Tech. J*, 48(3):280–285, 2012.

[54] RIKEN. Riken aics annual report, 2017.

[55] Bob Sorensen. Japan's flagship 2020 âpost-kâ system. *Computing in Science & Engineering*, 21(1):48–49, 2019.

[56] RIKEN. Post-k supercomputer with fujitsuâs original cpu, a64fx powered by arm isa, 2018.

[57] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15, 2008.

[58] Paul Messina. The exascale computing project. *Computing in Science & Engineering*, 19(3):63–67, 2017.

[59] Nicole Hemsoth. A look inside china's chart-topping new supercomputer, 2016.

[60] Bernier-Bruna Pascale, Eisenreich Sabrina, Gimenez Binder Renata, Inglis Catherine, Markidis Stefano, Rossi Ricardo, Soriano Cecilia, Smith Lorna, and Vander Aa Tom. Europe towards exascale : A lookback on 5 years of european exascale research collaboration, 2016.

[61] Joshua Wyatt Smith, Graeme A Stewart, Arnulf Quadt, and Rolf Seuster. Atlas software stack on arm64. In *J. Phys. Conf. Ser.*, volume 898, page 072001, 2017.

[62] David R Butenhof. *Programming with POSIX threads*. Addison-Wesley Professional, 1997.

[63] Daniel J Sorin, Mark D Hill, and David A Wood. A primer on memory consistency and cache coherence. *Synthesis Lectures on Computer Architecture*, 6(3):1–212, 2011.

[64] Ross Weber. Modeling and verifying cache-coherent protocols, vip, and designs. *Jasper Design Automation, June 2011*, 2011.

[65] Ashley Stevens. Introduction to amba® 4 aceâ¢ and big. littleâ¢ processing technology. *ARM White Paper, CoreLink Intelligent System IP by ARM*, 2011.

[66] Rasmus Ulfsnes. Design of a snoop filter for snoop based cache coherency protocols. Master's thesis, Institutt for elektronikk og telekommunikasjon, 2013.

[67] Andreas Moshovos, Gokhan Memik, Babak Falsafi, and Alok Choudhary. Jetty: Filtering snoops for reduced energy consumption in smp servers. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 85–96. IEEE, 2001.

[68] Giorgio Luigi Valentini, Walter Lassonde, Samee Ullah Khan, Nasro Min-Allah, Sajjad A Madani, Juan Li, Limin Zhang, Lizhe Wang, Nasir Ghani, Joanna Kolodziej, et al. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1):3–15, 2013.

[69] Tapasya Patki, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R De Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 173–182. ACM, 2013.

[70] Jens Breitbart, Josef Weidendorfer, and Carsten Trinitis. Case study on co-scheduling for hpc applications. In *2015 44th International Conference on Parallel Processing Workshops*, pages 277–285. IEEE, 2015.

[71] Barry Rountree, Dong H Ahn, Bronis R De Supinski, David K Lowenthal, and Martin Schulz. Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 947–953. IEEE, 2012.

[72] John D Davis, Suzanne Rivoire, Moises Goldszmidt, and Ehsan K Ardestani. Accounting for variability in large-scale cluster power models. Exascale Evaluation and Research Techniques Workshop (EXERT), 2011.

[73] Aniruddha Marathe, Peter E Bailey, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R de Supinski. A run-time system for power-constrained hpc applications. In *International conference on high performance computing*, pages 394–408. Springer, 2015.

[74] Chung-Hsing Hsu and Ulrich Kremer. *Compiler-directed dynamic voltage and frequency scaling for cpu power and energy reduction*. PhD thesis, Rutgers University, 2003.

[75] Sébastien Bilavarn, Jabran Khan, Cécile Belleudy, and Muhammad Khurram Bhatti. Effectiveness of power strategies for video applications: a practical study. *Journal of Real-Time Image Processing*, 12(1):123–132, 2016.

[76] Robin Bonamy, Sébastien Bilavarn, Fabrice Muller, François Duhem, Simon Heywood, Philippe Millet, and Fabrice Lemonnier. Energy efficient mapping on many-core with dynamic and partial reconfiguration: Application to a smart camera. *International Journal of Circuit Theory and Applications*, 46(9):1648–1662, 2018.

[77] D Suleiman, M Ibrahim, and I Hamarash. Dynamic voltage frequency scaling (dvfs) for microprocessors power and energy reduction. In *4th International Conference on Electrical and Electronics Engineering*, 2005.

[78] Dominik Brodowski and Nico Golde. Linux cpufreq governors. *Linux Kernel. https://www. kernel. org/doc/Documentation/cpu-freq/governors. txt*, 2013.

[79] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230, 2006.

[80] Maja Etinski, Julita Corbalán, Jesús Labarta, and Mateo Valero. Understanding the future of energy-performance trade-off via dvfs in hpc environments. *Journal of Parallel and Distributed Computing*, 72(4):579–590, 2012.

[81] Keith A Britt and Travis S Humble. High-performance computing with quantum processing units. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):39, 2017.

[82] Keith A Britt and Travis S Humble. Instruction set architectures for quantum processing units. In *International Conference on High Performance Computing*, pages 98–105. Springer, 2017.

[83] Alexander J McCaskey, Eugene F Dumitrescu, Dmitry Liakh, Mengsu Chen, Wuchun Feng, and Travis S Humble. Extreme-scale programming model for quantum acceleration within high performance computing. *arXiv preprint arXiv:1710.01794*, 2017.

[84] Keith A Britt, Fahd A Mohiyaddin, and Travis S Humble. Quantum accelerators for high-performance computing systems. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–7. IEEE, 2017.

Appendix A

Gem5 Related work

Refer to the github web page for more details about gem5 with aarch64 per core DVFS capabilities: https://github.com/jwanza/gem5-aarch64-dvfs/tree/master

Appendix B

Gem5 full system stack effort

Refer to the github web page for more details about the binaries and images developed to use gem5 with DVFS features at core level: https://github.com/jwanza/gem5-FullSystem-stack

Also, you will find plenty of scripts that enable a quick analysis of the simultations results.