Caractérisation de la performance temporelle et de la consommation électrique de systèmes embarqués basés sur des plates-formes multiprocesseurs/coeurs et mettant en oeuvre du logiciel temps réel : FORECAST : perFORmance and Energy Consumption AnalysiS Tool

Joffrey Kriegel

Université de Nice Sophia-Antipolis
Embedded Systems Department - Laboratoire d'Electronique, Antennes et Communications

Advanced Architecture Laboratory - Thales Communications and Security

Une thèse soumise pour le titre de Docteur
à l'université de Nice Sophia-Antipolis

Caractérisation de la performance temporelle et de la consommation électrique de systèmes embarqués basés sur des plates-formes multiprocesseurs/cœurs et mettant en œuvre du logiciel temps réel

FORECAST : perFORmance and Energy Consumption AnalysiS Tool

Présentée et soutenue publiquement par
Joffrey KRIEGEL

29 Janvier 2013

Membres du jury :

Daniel CHILLET	Rapporteur
Frédéric PETROT	Rapporteur
François VERDIER	Président du jury
Michel AUGUIN	Directeur de thèse
Alain PEGATOQUET	Co-directeur de thèse
Florian BROEKAERT	Tuteur industriel
Agnès FRITSCH	Invitée

Plus vous avez d'idées, moins vous arrivez à les structurer.

Malédiction du Créatif

Résumé

La multiplication des plates-formes embarquées disponibles sur le marché rend de plus en plus complexe le choix d'une plate-forme pour un produit. L'arrivée des architectures multi-processeurs augmente encore plus ce phénomène. Dans le contexte industriel actuel, il est nécessaire de disposer d'une méthodologie et des outils associés permettant d'évaluer rapidement ces plates-formes et celles qui apparaîtront dans le futur sur le marché afin de faire des premiers choix tôt dans le cycle de conception des produits.
Précédemment, il était nécessaire d'attendre l'arrivée sur le marché des plates-formes de test afin d'exécuter sur ces plates-formes des benchmarks et des applications afin d'évaluer leur performance et leur consommation.

Nous proposons ici une méthodologie et les outils associés permettant de modéliser un système (logiciel et matériel) puis d'estimer ses performances et sa consommation d'énergie. Notre méthodologie s'appuie sur des modèles simples à mettre en œuvre utilisant uniquement des informations présentes dans les documents techniques des constructeurs.
Autre avantage de notre approche, la simulation réalisée s'appuie sur du code exécutable généré automatiquement afin de s'exécuter en natif sur un ordinateur. Cela permet une exécution rapide des scénarios de test et offre la possibilité de faire de l'exploration d'architectures.

Nous avons procédé à diverses validations en utilisant des applications variées (décodeur H.264, application radio, benchmarks classiques, ...) et en comparant les performances et la consommation estimée avec l'équivalent sur des plates-formes réelles (OMAP3/4, i.MX6, QorIQ, ...). Cela a permis d'évaluer l'erreur d'estimation de FORECAST (l'outil développé lors de cette thèse) et ainsi de s'assurer que le taux d'erreur reste dans des bornes admissibles c'est-à-dire inferieures à 20%.
Nous avons d'un autre côté comparé notre approche avec celles développées dans deux autres projets OpenPeople (ANR) et COMCAS (Catrene) afin de s'assurer que le rapport effort/précision de notre approche est intéressant.

Abstract

The number of available commercial platforms is constantly increasing. The choice of an architecture that fit as much as possible the requirements is therefore more and more complex. This is even more real with the availability of recent multiprocessors architectures. As a consequence, methodologies with their associated tools are required in order to quickly evaluate future platforms, so that choices can be made early in the design flow. So far, evaluating either the performance or the power consumption of a dedicated platform was performed through executing benchmarks and applications on this platform.

In this thesis, a new methodology with its associated tools, called FORECAST, is proposed to model both the hardware and software of a system, and then to estimate its performance and its power consumption. Our methodology is based on efficient models, easy to characterize using only information provided by constructor datasheets. Moreover, our approach is able to automatically generate an executable code of the system that can be simulated on the host machine. This simulation allows a rapid execution of multiple test cases. Our approach is therefore well adapted for performing architecture exploration.

A lot of experimentations have been performed using our tool FORECAST for different applications (H.264 video decoder, radio application, benchmarks...) and different hardware platforms. Results obtained both in performance and in power consumption have then been compared with existing platforms (OMAP3, OMAP4, i.MX6, QorIQ...), but also with two collaborative projects, OpenPeple (ANR) and COMCAS (Catrene), dealing also with performance and power estimations. The comparison demonstrates the accuracy of our approach as the estimation is always below a 20% error margin. These experimentations have also shown that our methodology provides a very efficient ratio between the modeling effort and the accuracy of the estimations.

Keywords : *Modélisation, Performance, Consommation d'énergie, Multi-coeurs*

Remerciements

Je souhaiterais sincèrement remercier Agnès Fritsch, Responsable du laboratoire AAL et Michel Auguin, Directeur de Recherches au CNRS, pour leurs conseils avisés, leur disponibilité et leurs encouragements tout au long de mon doctorat.

Je voudrais également remercier :

- Florian Broekaert, Ingénieur à Thales Communications and Security et Alain Pegatoquet, Maître de conférences à l'université de Nice Sophia-Antipolis, pour avoir suivi mes travaux et pour leur aide.

- Daniel Chillet, Maître de conférences à l'université de Rennes 1 et Frédéric Pétrot, Professeur au Grenoble Institute of Technology, qui ont accepté de juger mes travaux.

- François Verdier, Professeur à l'université de Nice pour avoir accepté de présider mon jury.

- L'équipe du laboratoire AAL de Thales Communications and Security pour leur aide.

- Les différents stagiaires qui ont permis de faire avancer l'ensemble des travaux.

Un grand merci aussi à ma compagne et ma famille pour leur soutien et leur patience.

Table des matières

Chapitre 1

Introduction

En quelques années, le nombre de systèmes embarqués présents dans notre quotidien a considérablement augmenté. Aujourd'hui, ce domaine est encore en pleine expansion et on observe que de nouveaux produits (tablettes, Smartphones, GPS...) sont mis sur le marché à une cadence toujours plus élevée. Afin de rester compétitifs dans ce secteur ultra concurrentiel, les industriels sont contraints de proposer en un minimum de temps des produits toujours plus innovants et fournissant aux utilisateurs toujours plus de fonctionnalités. En effet, le "time to market" est très souvent un point clé de la réussite des entreprises de ce secteur. Du fait de l'intégration continue de nouvelles fonctionnalités dans ces équipements électroniques, la conception de systèmes embarqués est devenus de plus en plus complexe. La puissance de calcul nécessaire s'est également accrue (augmentation de la fréquence processeur et augmentation du nombre de cœurs). Mais depuis, les problématiques liées à la consommation d'énergie, telles que l'extension de l'autonomie, la réduction de la dissipation thermique ou de la consommation électrique, sont également devenus des enjeux majeurs.

En effet, la consommation d'énergie des équipements embarqués a tendance à augmenter (Figure 1.1), en partie, à cause du logiciel de plus en plus complexe. Il est donc nécessaire de trouver une plate-forme disposant à la fois des performances de calcul nécessaires, mais aussi minimisant la consommation d'énergie.

Par ailleurs, cet accroissement de la consommation induit une dissipation thermique plus élevée exigeant des systèmes de refroidissement de plus en plus performants et gourmands eux aussi en énergie. Réduire la consommation énergétique est donc devenu une nécessité lorsque le produit se situe dans un appareil embarqué ou dans un espace confiné comme par exemple à l'intérieur d'un avion, d'un navire ou d'un sous-marin.

Ainsi, les décisions de conception matériel et logiciel effectuées au début des projets sont très importants. L'objectif ultime étant de proposer l'architecture la mieux adaptée aux besoins clients, la moins cher, la moins risquée et possédant des capacités d'évolutions.

De nombreuses études ont montré que le processeur constitue une source importante de consommation. L'augmentation de la fréquence de fonctionnement entre les différentes générations de processeurs a aggravé ce problème. De plus, le frein induit par la consommation d'énergie sur l'accroissement de la fréquence de fonctionnement ne permet plus de répondre aux exigences applicatives, en terme de traitements de données, dans les nouveaux équipements. La solution adoptée, compte tenu aussi des spécificités des nouvelles applications, consiste à d'introduire du parallélisme au sein des architectures. D'après la nouvelle loi de Moore, le nombre d'unités de traitement double tous les deux ans dans un système sur puce (2007 : 4 coeurs, 2009 : 8 coeurs, 2011 : 16 coeurs).

Pour toutes ces raisons (choix rapide d'architecture, contraintes de consommation et de performances),

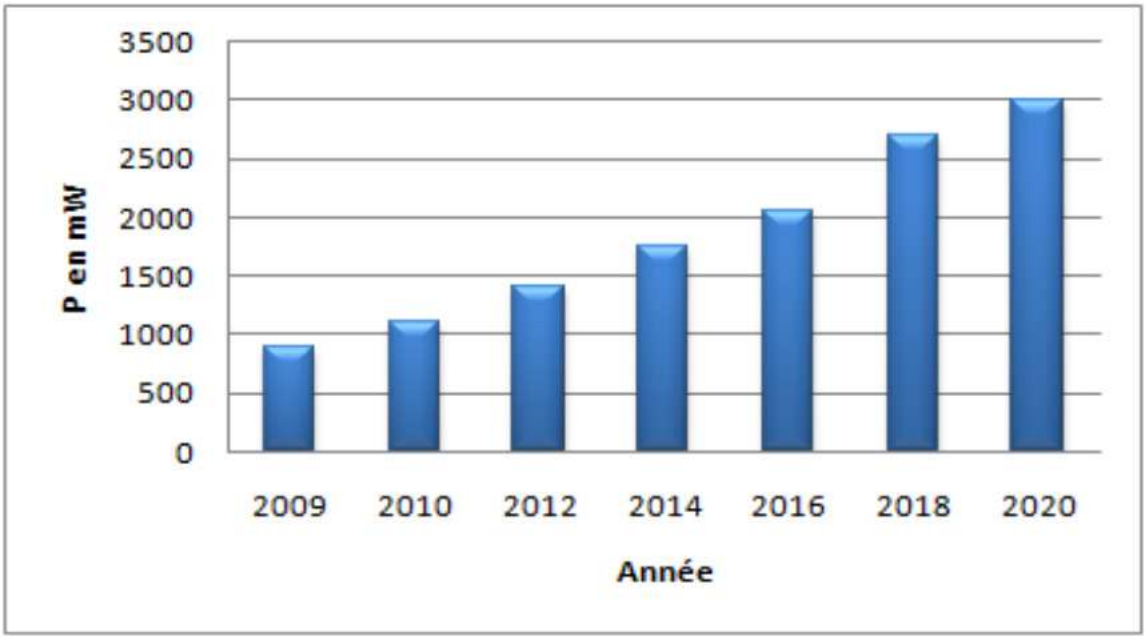

FIGURE 1.1: Évolution de la consommation des systèmes embarqués (type téléphone portatif) prévue par l'ITRS [74].

on observe une certaine effervescence de la communauté scientifique autour de la thématique de l'estimation de la performance et de la consommation d'une plate-forme mais également de l'exploration de l'espace de conception. Cette estimation est d'autant plus importante dans un système temps réel où les contraintes temps-réel doivent absolument être respectées et il faut être certain du choix des composants et paramètres (fréquence par exemple) que l'on a fait sans pour autant sur-dimensionner la plate-forme ce qui engendrerait un surcoût et une sur-consommation du système.

Dans ce contexte, Thales Communications & Security, leader dans les systèmes de communications radio sécurisés pour le domaine civil et militaire, constate effectivement que le choix des composants d'une plate-forme matérielle devient de plus en plus difficile. En effet, Thales n'est pas un fournisseur de processeur et par conséquent utilise très souvent des composants COTS (Commercial Off-The-Shelf) dans l'architecture de ses produits. La problématique actuelle qui se retrouve dans l'industrie des équipementiers et systémiers est de savoir par quels moyens il est possible de valider et d'optimiser les choix d'architecture et de partitionnement (logiciel/matériel). Ces choix de conception sont contraints par plusieurs facteurs. Les plus importants sont souvent le respect des contraintes temps-réel et la minimisation de la consommation électrique.

L'émergence de nouvelles architectures matérielles (multicore, manycore...) offrant plus de puissance de calcul sont attractives mais sont complexes à exploiter efficacement. Pour les caractériser et les comparer, les architectes utilisent le plus souvent les informations fournies par les "datasheets". Malheureusement peu d'informations précises sur les performances et sur la consommation d'énergie y sont présentes. Des campagnes d'expérimentations longues et fastidieuses sont donc généralement nécessaires avant de valider le choix d'une solution. Ces expérimentations permettent d'avoir une meilleure vision des performances réelles des composants du marché mais prennent du temps, nécessitent de disposer des composants (les plates-formes de tests ne sont d'ailleurs pas toujours disponibles suffisamment tôt dans le cycle de conception) et de développer des logiciels pour les tester. Pour ces raisons, il est de plus en plus nécessaire de proposer des méthodologies accompagnées d'outils pour évaluer et valider les choix architecturaux, ceci de façon plus formelle qu'avec de simples estimations basées sur les "datasheets" constructeurs, mais également de façon

moins onéreuses et moins longues qu'avec des expérimentations sur cibles.

Le sujet de thèse défini entre Thales et le LEAT s'inscrit dans cette thématique. Le sujet exact s'intitule : "Caractérisation et optimisation de la performance temporelle et de la consommation électrique de systèmes embarqués basés sur des plates-formes multiprocesseurs/cœurs et mettant en œuvre du logiciel temps réel". La problématique réside dans un premier temps dans l'identification des paramètres caractéristiques des composants logiciels et matériels impactant la performance et la consommation du système. Ces travaux ont pour but d'aboutir au développement d'une "boite à outil" logicielle permettant d'obtenir des estimations sur les performances/consommation de plates-formes mono-processeur & multi-processeurs Les exigences de Thales par rapport à l'approche développée sont les suivantes :

- La possibilité de comparer différentes solutions du marché en terme de performance et de consommation, mais aussi d'optimiser l'utilisation d'une solution.
- Les modèles doivent être rapides à développer et ne doivent comporter que des informations de haut niveau (présentes dans les datasheets).
- L'obtention des estimations pour une architecture donnée doit être possible en un temps très rapide, inférieur à une minute.
- L'exploration d'architecture doit pouvoir s'effectuer en quelques minutes, 5 typiquement.
- Les erreurs d'estimation par rapport aux mesures doivent être inférieures à 20%.

Le second chapitre de cette thèse aborde les différentes méthodes d'estimations de la performance et consommation d'énergie des systèmes embarqués ainsi que certains langages associés. Nous décrivons ensuite les problèmes liés à l'exploration d'architectures puis comparons certains "frameworks" existants.

Le troisième chapitre porte sur les différents paramètres architecturaux impactant la performance et la consommation d'énergie des plates-formes. Nous nous appuierons sur des expérimentations effectuées sur des plates-formes réelles afin de s'assurer de la criticité des paramètres.

Le quatrième chapitre décrit avec précision la méthodologie développée ainsi que l'outil FORECAST associé. Le flot est décrit dans l'ordre d'enchaînement des différents étages fonctionnels en commençant par les entrées de l'outil, puis les étapes d'estimations et la génération automatique de code. Enfin, nous abordons ce que fournit FORECAST en terme de résultats ainsi que la partie exploration d'architectures.

Le cinquième chapitre regroupe les différents résultats obtenus tout au long de cette thèse. Il compare les estimations effectuées avec les valeurs mesurées sur les plates-formes réelles mais aussi avec deux autres approches issues de projets collaboratifs.

Ce mémoire se termine en concluant sur les travaux effectués et en essayant de dégager des perspectives basées sur la méthodologie développée.

Chapitre 2

Exploration d'architectures : modèles, simulations ou estimations ?

De plus en plus de systèmes embarqués utilisent des processeurs multicoeurs homogènes/hétérogènes plutôt que des processeurs à haute fréquence. Ils permettent en effet d'obtenir de meilleures performances tout en limitant la consommation d'énergie. Le nombre de plates-formes multicoeurs et multiprocesseurs étant croissant et les méthodes trop simplistes transposées du monocoeur ne fonctionnant plus sur des architectures si complexes, il devient donc nécessaire de trouver une méthode outillée pour définir précisément les performances et la consommation électrique d'un système contenant des processeurs, des mémoires et des périphériques. La possibilité de faire de l'exploration d'architecture devient aussi de plus en plus importante. Dans ce manuscrit, nous appellerons architecture un ensemble de processeurs ainsi que leurs mémoires associées. Nous ne traiterons pas du cas des périphériques ou des GPU et many-cores.

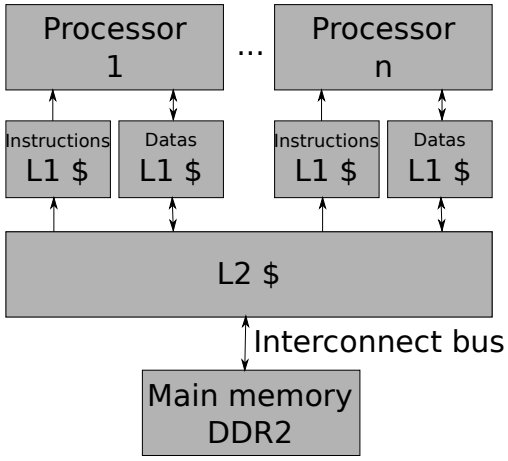

FIGURE 2.1: Exemple simple d'architecture matérielle que l'on étudiera.

La figure 2.1 montre un exemple simple de n-processeurs ainsi que leurs mémoires. C'est ce type d'architecture qui sera étudiée dans ce document.

2.1 Méthodes d'estimation de la performance et de la consommation d'énergie

La performance d'une plate-forme embarquée est primordiale étant donné la complexité grandissante des systèmes embarqués (logiciel et matériel). Dans cette section, nous abordons successivement les simulations, les modèles analytiques et les estimations. Au préalable, on fera un récapitulatif des benchmarks (logiciels de test) car ils sont nécessaires pour définir un type de logiciel qui s'exécutera sur la plate-forme. Nous aborderons finalement les différents modèles de calculs utilisés dans l'embarqué, les langages de description haut-niveau et les méthodes de profiling aidant à créer des modèles d'applications.

Les travaux précédents montrent une variété de différentes approches depuis la simulation au niveau cycle et le niveau RTL jusqu'aux méthodes purement analytiques à un haut niveau d'abstraction. Suivant la méthode d'évaluation choisie, la phase de "déploiement"(mapping) dans le Y-chart (Fig. 2.2) permettant de déterminer les performances peut être très complexe, faisant des appels spécifiques à un compilateur ou à une phase de synthèse. D'autres méthodes, quant à elles, représentent la décision de déploiement implicitement en faisant varier le jeu de paramètres. Le problème concernant l'exploration de l'espace des conceptions en ayant des résultats de performance pour chaque point individuel sera abordé dans la section suivante.

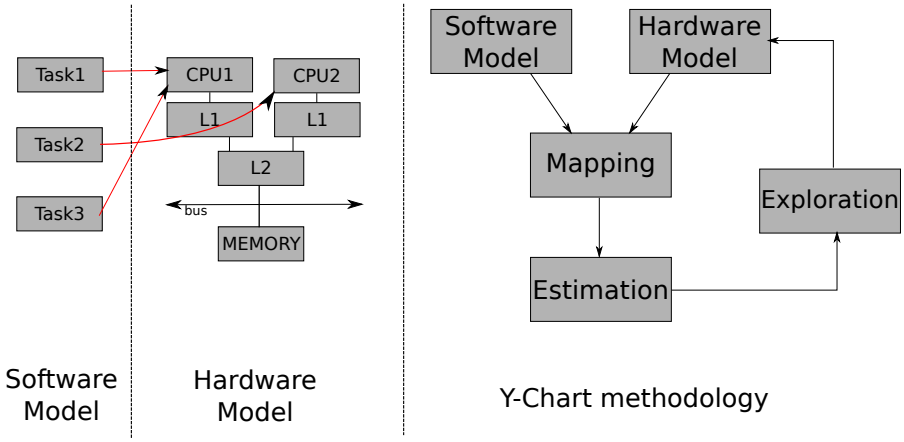

FIGURE 2.2: Représentation Y-Chart (à droite) et déploiement logiciel sur une plate-forme matérielle (à gauche).

2.1.1 Évaluation par programme de test (benchmarks)

Dans le cas où les plates-formes matérielles à évaluer sont disponibles sur le marché, une première méthodologie consiste à exécuter des logiciels de test (appelés benchmarks) sur les plates-formes. Ceci permet à la fois d'évaluer leurs performances mais aussi de les comparer.

Dans le but d'obtenir des valeurs de performance comparables et ayant du sens, toutes les méthodes d'évaluation nécessitent de définir des benchmarks de différentes natures qui décrivent la charge imposée au système évalué et qui permettront de vérifier la validité des modèles. Pour avoir des résultats reproductibles, un benchmark inclut une description de l'application et de l'architecture testée, des contraintes de charge (définies par l'environnement de travail du système embarqué), une répartition de l'application sur l'architecture, ainsi que des métriques et des fonctions de coût. Les benchmarks disponibles peuvent, dans un premier temps, être classés selon leur domaine d'application. On retrouve par exemple les domaines du "Network processing" ([141][98]), "General purpose computing" (SPEC, `http://www.spec.org`), "Embedded systems" ([60][24]) ou Multimedia ([85]).

Le problème des benchmarks est de savoir exactement ce qui est testé dans chaque programme. En effet, les benchmarks ont plutôt tendance à tester une performance globale du système sans forcément évaluer une partie précise (coeur de processeur, cache, etc). Nous avons donc classé les benchmarks avec plusieurs niveaux possibles :

- "Bas niveau" : permettent de tester des points caractéristiques du processeur, comme par exemple la latence d'interruption, le débit mémoire etc ...
- "Niveau intermédiaire" : permettent de tester des petites applications (ou morceaux d'applications) type comme la compression/décompression, le calcul flottant etc...
- "Haut niveau" : permettent d'évaluer les performances globales du système complet en exécutant une application complète sur la plate-forme cible.

Les benchmarks de haut niveau et intermédiaire sont généralement implémentés avec un OS déjà présent sur la plate-forme (souvent Linux) ce qui n'est pas le cas des benchmarks de bas niveau. Ces benchmarks ne subissent donc pas les variations dues à l'OS (par exemple le noyau peut reprendre la main et polluer les mémoires cache).

La difficulté qui apparaît alors pour les benchmarks bas niveau est l'écriture des drivers permettant d'activer les zones à tester. De plus, ces logiciels sont souvent dépendants de l'architecture sur laquelle ils sont écrits puisque de l'assembleur est souvent nécessaire.

La suite de benchmark nbench [24] représente ce que l'on appelle un benchmark de niveau intermédiaire. Elle effectue des tests de plusieurs paramètres du processeur comme les accès mémoires ou le traitement pur. La figure 2.3 schématise le type de benchmark pour les différents tests de nbench [24]. De plus, les tests "Fourier coefficients", "Neural net" et "LU decomposition" effectuent une évaluation de la capacité du processeur à traiter des types de données flottantes.

Afin d'évaluer une partie de nos résultats, nous avons utilisé les benchmarks "Numeric sort", "String sort", "Bitfield", "Emulated floating point" et "Huffman compression" de nbench, ainsi que le benchmark codeur jpeg de MiBench. D'autres applications complètes et fonctionnelles ont aussi été utilisées pour évaluer notre approche.

Il apparaît donc que lorsque les plates-formes que l'on souhaite évaluer sont disponibles, il est intéressant d'utiliser des benchmarks de plusieurs niveaux différents afin d'obtenir des informations pertinentes.
Lorsque les plates-formes ne sont pas disponibles, il est alors toujours utile d'avoir des benchmarks de références qui serviront à évaluer les modèles et la qualité des estimations. Dans ce cas, il est alors nécessaire d'évaluer les plates-formes à l'aide de simulation, de modèles analytiques ou toute autre méthode d'estimation.

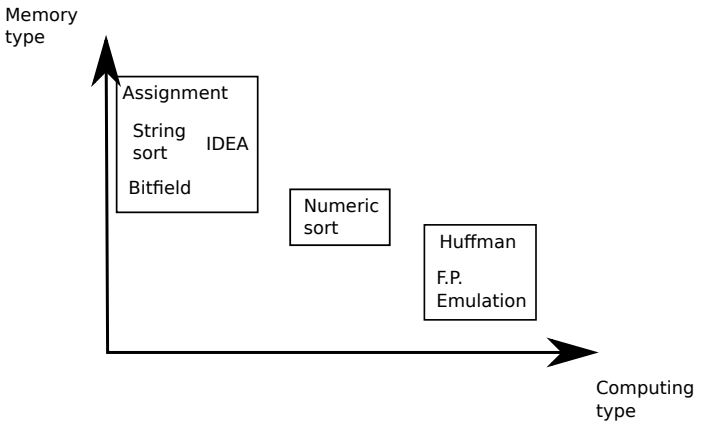

FIGURE 2.3: Les différents benchmarks classés dans deux catégories.

2.1.2 L'évaluation fondée sur la simulation

L'évaluation d'un système grâce à la simulation s'avère très utile lorsque l'on souhaite estimer un "cas-moyen" d'exécution. En effet, la simulation permet d'évaluer un modèle du système grâce à un jeu de stimuli (application réelle, modèle d'application) défini. Grâce à cela, il est possible d'effectuer une exécution qui reflète le comportement typique du système. L'inconvénient en revanche, est le besoin d'obtenir un modèle exécutable qui peut s'avérer difficile dans les phases amont de la conception. Les simulations sont très utiles pour analyser les effets dynamiques et sporadiques apparaissant dans le système.
Deux types de simulation vont nous intéresser ici. Tout d'abord la simulation au niveau système, qui va permettre de simuler un système complet (dont les processeurs et les mémoires). Ensuite nous aborderons la simulation au niveau cycle qui permet d'obtenir des résultats très précis.

La simulation au niveau système. Grâce à cette méthodologie, il est possible de modéliser et de simuler l'interaction entre les composants du système en utilisant différents modèles de calcul (MoC) par exemple.
Un exemple de modèle de calcul très utilisé dans le domaine de l'exploration d'espace de conception est le modèle KPN pour "Khan process networks" [77]. Les modèles KPN supposent un réseau de processus autonomes qui communiquent à travers des canaux FIFO. Ces processus communiquent de point-à-point et utilisent des synchronisations à lecture bloquante. Chaque processus du réseau est spécifié comme un programme séquentiel qui s'exécute de manière concurrente avec les autres processus. Un KPN utilise les caractéristiques suivantes :

- Le modèle KPN est déterministe, ce qui veut dire, qu'indépendemment de l'ordonnancement choisi pour évaluer le réseau, il existera toujours la même relation d'entrée/sortie. Cela nous donne une grande liberté lorsque l'on lie les processus au matériel ou au logiciel.
- La synchronisation inter-processus est faite avec des lectures bloquantes. C'est un protocole de synchronisation très simple qui peut être réalisé facilement et efficacement dans le matériel ou le logiciel.
- Les processus s'exécutent de manière autonome et se synchronisent via les lectures bloquantes. Lorsque l'on lie des processus sur du matériel comme un FPGA, on obtient des "îlots" autonomes sur le FPGA qui sont synchronisés seulement par des lectures bloquantes.
- Comme le contrôle est distribué à chaque processus individuel, il n'y a pas de scheduler global. De ce fait, partitionner un KPN au travers un nombre de composants reconfigurables ou microprocesseurs est une tâche simple.
- Comme l'échange des données est distribué au travers de FIFOs (Fig. 2.4), il n'y a pas de notion de mémoire globale accédée par de multiples processus. Il n'y a donc pas de contention qui apparaît.

L'un des intérêts des KPN en synthèse de systèmes embarqués est qu'ils permettent de décrire des

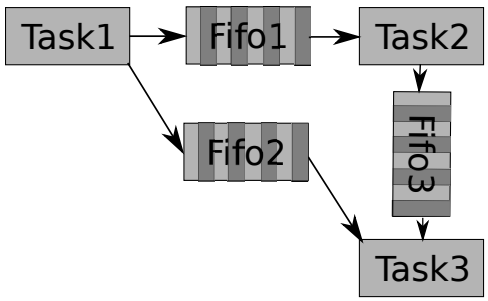

FIGURE 2.4: Illustration KPN de trois tâches communicant ensemble.

systèmes réactifs comportant du parallélisme, tout en se prêtant à une analyse "par dépendances" analogue à celle qui permet de traiter les codes séquentiels.

La simulation niveau cycle. Pour permettre d'évaluer avec précision un système, il est nécessaire de raffiner les modèles. La simulation au niveau cycle permet d'obtenir une précision au cycle d'horloge prés. Les modèles du matériel peuvent être faits de deux façons :

- En logiciel, en modélisant les latences et le comportement du matériel.
- A l'aide d'un langage de description matériel tel que le VHDL (Very High Description Language), permettant de plus un prototypage rapide sur FPGA (Field-Programmable Gate Array) si nécessaire.

A ce niveau, il est nécessaire d'obtenir l'application réelle qui doit être évaluée et non juste un modèle comportemental non fonctionnel. Il faut donc que l'application soit écrite dans un langage de programmation haut-niveau ou de l'assembleur.
Il est ensuite possible d'utiliser l'application sur un simulateur de matériel que l'on appellera de la co-simulation, ou alors d'utiliser une simulation de l'application avec un modèle logiciel du matériel.

Le langage SystemC essaie de tirer profit de ce principe. Ce langage sera décrit dans la section 2.1.6.

Les modèles au cycle-près sont très souvent utilisés pour étudier les micro-architecture des processeurs. Des simulateurs, comme SimpleScalar [21] et SimOS [120], sont spécialisés dans certaines catégories de processeurs, tels que les MIPS ou les ARM. Il est alors possible d'estimer les effets des caches, prédictions de branchement et les largeurs de bus, qui n'affectent que les mécanismes d'exécution, sans avoir besoin de recompiler l'application pour une nouvelle architecture.

D'autres approches visent à améliorer la vitesse d'exécution des simulations. Par exemple le projet COMCAS [31] utilise l'émulateur QEMU [119][13] permettant de traduire dynamiquement les instructions d'une architecture cible vers l'architecture de l'hôte. Dans sa version standard, QEMU ne permet pas d'obtenir des informations sur les temps d'exécution. Pour cela, il est nécessaire d'inclure les instances de QEMU (permettant de simuler le fonctionnement des processeurs) dans un wrapper SystemC pour obtenir des informations temporelles.
La figure 2.5 montre le schéma global de la plate-forme utilisée dans le projet COMCAS. Comme on peut le voir, chaque processeur de la plateforme est simulé par un émulateur QEMU. Ces différents émulateurs sont imbriqués dans un "wrapper" SystemC permettant d'ajouter une notion de temps et de les connecter aux composants externes (interconnexion, mémoires, timers) eux aussi en SystemC. Afin de connaître le temps d'exécution des processeurs sous QEMU, chaque instruction décodée est annotée avec un temps (donné par la datasheet constructeur). Cette méthode permet un gain important de temps au détriment de la précision de l'estimation (qui baisse à environ 10%).
De manière équivalente, [103] crée une connexion de QEMU vers des périphériques SystemC afin d'évaluer un système complet. Malheureusement, seule la partie SystemC est "timée" ce qui ne permet pas vraiment d'évaluation de performance de tout le système mais plutôt de détecter et corriger les problèmes fonctionnels. Une partie des résultats montrent l'overhead de QEMU par rapport à une plate-forme réelle. Pour un

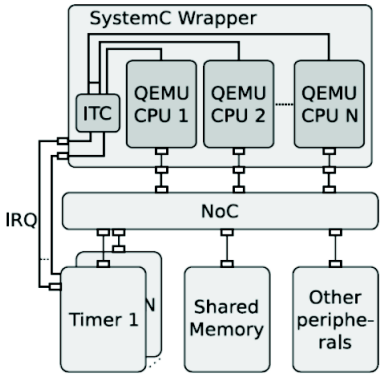

FIGURE 2.5: L'encapsulation de QEMU dans SystemC [52].

décodage MPEG2 sur une plate-forme ARM9, le temps d'exécution est de 1.5 fois supérieur. En revanche pour l'émulation d'un PC i386, le temps d'exécution est 7 fois supérieur. QEMU est donc efficace mais dans une certaine mesure, si la plate-forme évaluée est très puissante, QEMU aura un overhead important.
D'autres projets concernent QEMU et CoWare [95] ce qui permet de modéliser le GPP sous QEMU et un DSP ou d'autres IP matérielles sous CoWare en SystemC ou RTL. Les résultats de ce projet montrent qu'il n'y a pas beaucoup de pertes de vitesse à utiliser QEMU+Coware au lieu d'utiliser uniquement Coware (pertes entre 0 et 50%).

L'approche par simulation apporte plusieurs niveaux de précisions. Tout d'abord, le niveau système permet d'évaluer rapidement un système complet et de simuler l'interaction entre les différents composants. Ce niveau est de plus souvent basé sur des modèles de calcul simples à mettre en place comme des machines d'états, des réseaux de Pétri, des graphes de tâches ou des KPN. Ces simulations restent peu précises (environ 10-15% d'erreur) mais sont rapides et permettent la simulation de tout le système.
D'un autre coté, les simulations au cycle-près permettent d'obtenir des précisions très correctes (entre 0 et 3%). En contrepartie, ces simulations peuvent être très longues à s'exécuter, et le plus souvent seule une partie du système est modélisée. On aura donc ici des outils surtout utiles aux constructeurs de SoC (comme Texas Instruments, STmicroelectronics, ...) plutôt qu'à un utilisateur final de SoC ou un fournisseur de systèmes.
Afin de convenir à nos besoins, une approche basée sur des simulations au niveau système est bien plus adaptée.

2.1.3 Les approches purement analytiques

Contrairement à l'approche par simulation qui fait la plupart du temps une estimation du "cas-moyen" d'exécution, les approches analytiques sont très utiles si l'on cherche à estimer un comportement au pire-cas. Un comportement déterministe est aussi nécessaire afin d'être capable d'obtenir des estimations concluantes. En effet, des évènements dynamiques vont poser des problèmes pour les estimations purement analytiques. Plusieurs méthodes analytiques sont alors possibles :

Profiling statique. Plusieurs méthodologies existent pour effectuer le profiling statique d'une application. L'analyse de la complexité des algorithmes permet, à partir d'une analyse de l'algorithme d'obtenir une performance temporelle. En effet, en analysant le nombre de boucles nécessaire ainsi que les structures de données, il est possible de déduire la complexité de l'algorithme et d'en déduire son temps d'exécution ou sa consommation électrique.
Voici quelques résultats typiques :

- Pour des algorithmes simples, la fréquence maximum des opérations basiques est utilisée. Par exemple, la complexité est en $O(n^2)$ si l'algorithme possède deux boucles imbriquées, alors que la complexité est $O(n^3)$ lorsqu'il y a trois boucles.
- Pour des algorithmes standards, comme quick sort et heap sort, la complexité est $O(n \cdot log(n))$ alors qu'un tri linéaire aura comme complexité $O(n)$.

L'analyse des dépendances d'un ordonnancement statique de tâches ou d'un graphe d'appel de fonctions peut aussi permettre d'extraire le pire-cas d'exécution. Ici, les évènements externes sporadiques et non-déterministes sont exclus. En effet, les interruptions, la récursivité de l'application, et les effets du système d'exploitation sont impossibles à estimer avec cette méthode.
Une dernière méthode relativement simple consiste à compter les opérations basiques apparaissant dans le pseudo-code de l'application. L'inconvénient de cette méthode est qu'il est très difficile d'estimer des effets dynamiques comme par exemple les boucles ou les branchements conditionnels.

Modèles analytiques basés sur un flux d'évènements. Pour certains domaines d'applications spécifiques (dédiés calcul, automobile, ...), des modèles de tâches et de charge de travail sont disponibles. Dans la littérature sur l'analyse du temps-réel, il y a quatre principaux évènements. Le plus simple et efficace est l'évènement périodique. Un simple paramètre T traduisant la période, permet d'exprimer cet évènement. Ensuite, il est souvent possible d'autoriser l'évènement à dériver légèrement dans le temps. On ajoute pour cela un paramètre J (jitter). Le troisième modèle utilisé est un modèle avec un burst d'évènements. Le dernier évènement possible est l'évènement sporadique, qui lui n'est défini que par un temps minimum entre chaque occurrence. Ces évènements permettent d'effectuer une analyse d'ordonnancement statique du système afin d'évaluer le pire cas d'exécution.

Les méthodes analytiques sont un bon moyen d'obtenir des estimations très tôt dans la conception. De plus, elles ne nécessitent pas de modèle exécutable et cherchent à estimer le pire-cas, ce qui est souvent utile dans le cas de systèmes temps réel. Les erreurs d'estimation sont par contre assez importantes (supérieur à 10%) car les éléments dynamiques sont rarement pris en compte. Elles sont adaptées aux systèmes déterministes.

2.1.4 Les méthodes combinant simulation et approche analytique

Afin de réduire le surcoût lié à la simulation d'un système complet sous évaluation, les méthodes suivantes essayent de réduire le temps de simulation en réunissant toutes les caractéristiques qui sont communes entre les conceptions en cours d'évaluation et qui ne sont pas soumises à l'exploration d'architecture, dans une seule simulation initiale. Les informations extraites de la simulation initiale peuvent être réutilisées par toutes les évaluations. L'évaluation temporelle se réduit au temps qu'il faut pour évaluer les caractéristiques distinctives.

L'analyse de performance basée sur les traces. Cette méthode d'analyse est fortement utilisée pour évaluer les structures de mémoires et les bus [140]. Elle consiste à exécuter le programme une première fois, et d'extraire tous les accès mémoires pour les stocker dans une trace. Ensuite, suivant le modèle de cache, de mémoire et de bus, la trace est utilisée pour calculer les performances globales, le taux d'occupation des bus, ou encore le taux de cache-hit et cache-miss.
Le but de cette méthodologie est de réduire le temps d'estimation (et plus tard d'exploration) en n'effectuant qu'une seule fois une simulation coûteuse en temps. Une fois cette simulation effectuée, il est possible de réutiliser les traces obtenues avec différentes structures de cache ou de bus. Des exemples d'études et d'outils qui utilisent cette méthode pour l'analyse de la performance, l'analyse énergétique et piloter l'exploration d'architecture d'un sous-système mémoire sont par exemple Fornaciari et al. [43][44] et Givargis et al. [50].

Modèles analytiques avec simulation de calibration initiale. Dans le domaine des processeurs de réseau, Franklin et Wolf [45] développent une méthodologie nécessitant une première caractérisation à l'aide de simulations d'un benchmark spécifique. Des paramètres sont ainsi tirés des simulations effectuées avec SimpleScalar [21], Wattch [19] et CACTI [102].

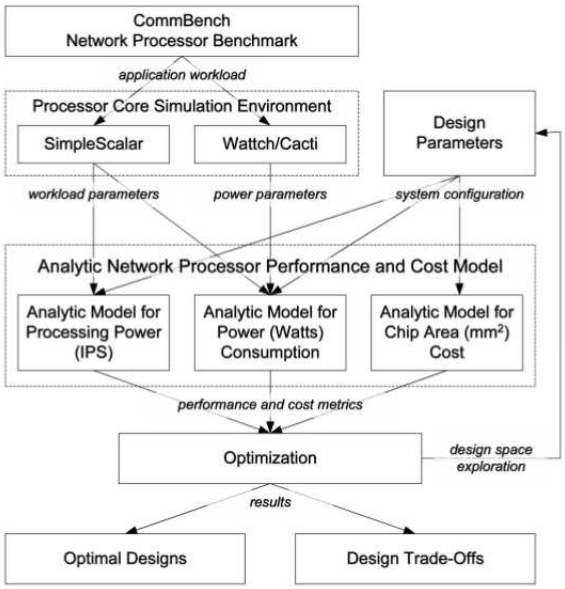

FIGURE 2.6: Approche par calibration initiale suivie de modèles analytiques.[45]

Comme on peut le voir sur la figure 2.6, une partie analytique (au centre) va ensuite être utilisée pour estimer la performance et la consommation d'énergie. Cette partie utilise les paramètres tirés des simulations et de la configuration de la plate-forme.
Cette méthode permet d'avoir accès à des informations dynamiques (grâce à la calibration initiale par simulation) puis d'explorer rapidement des architectures à l'aide de modèles analytiques.

2.1.5 Les modèles de calcul

Comme dit précédemment, KPN est l'un des différents modèles de calcul existant. De nombreux autres modèles de calcul (MoC) ont été définis dans la littérature, chacun apportant ses spécificités. La principale difficulté réside dans la modélisation du temps et de la concurrence [47]. Le type d'application du système va conditionner le choix du modèle de spécification. Par exemple, si il est plutôt orienté contrôle ou données.

Si le concepteur considère qu'un système est dominé par le contrôle et possède les caractéristiques suivantes :

- Réactions à des événements discrets
- Pas d'hypothèse sur l'occurrence des événements
- Tous les événements doivent être pris en compte
- Contrôle/commande de processus

On utilisera alors plutôt des MoC de type Système à événements discrets, Système réactif synchrone, Machine d'états finis, Réseaux de Petri, Processus concurrents et communicants (Graphe de tâches).

Dans le cas contraire, lorsque le système est plutôt dominé par les données et possède les caractéristiques suivantes :

- Les sorties sont fonctionnellement dépendantes des entrées
- Les occurrences des valeurs sur les entrées sont périodiques
- Traitements intensifs
- Traitement du signal et des images

on s'orientera plutôt vers les MoC de type Kahn Process Networks, Dataflow Process Networks.

Les travaux de [38] établissent ces concepts et énumèrent différents MoC couramment utilisés, que l'on retrouve aussi dans [30] et une synthèse de Jantsch et Sander [75][76]. Nous résumerons les différents MoC ainsi :

Synchronous Data Flow (SDF) Les flots de données synchrones [96] sont des modèles statiques dans lesquels le nombre d'éléments de données produits ou consommés est connu lors de la conception. Les nœuds de traitement (ou tâches) communiquent à travers des files d'attente et peuvent être ordonnancés statiquement selon le flux d'arrivée des données (synchronisation implicite par les données). Généralement, les applications de traitement du signal se prêtent bien à ce genre de modélisation. On peut les spécifier à l'aide de DFG (Data Flow graph), où chaque nœud représente un élément de calcul et les arcs des chemins de données. On retrouve ce modèle pour décrire aussi bien des applications logicielles que pour décrire des systèmes d'accélérateurs matériels.

Continuous Time (CT) les éléments de calcul communiquent avec des signaux en temps continu. Ces systèmes sont typiquement utilisés pour représenter des équations différentielles comme dans les systèmes physiques et l'électronique analogique. Les outils et langages associés sont typiquement ceux de Matlab/Simulink.

Processus Séquentiels Communicants(CSP) qui communiquent via des messages synchrones comme le "message passing" ou les "rendez-vous" [71]. Les langages CSP, Occam, Lotos se prêtent bien à ce genre de formalisme.

Réseaux de processus (PN) comme les Kahn PN Comme expliqué précédemment, les réseaux de processus de Kahn [77] communiquent via des messages asynchrones transitant par des queues (FIFO) de taille infinie. C'est un modèle courant pour les applications de traitement du signal.

Finite State Machine (FSM) ou machine à états finis ces FSM sont représentées par des graphes de flots de contrôle (Control-Flow Graph (CFG)) et d'états (statecharts de Harel [72]), dans lesquels les nœuds représentent des états et les arcs sont les traitements déclenchés par des éléments booléens que l'on nomme gardes. On les utilise souvent pour décrire les automates (logiciels ou matériels) mais aussi pour décrire les états des tâches logicielles dans un RTOS. Certains [131] définissent même un modèle abstrait basé sur les FSM pour le co-design, nommé ACFSM (Abstract codesign finite-state machine).

Discrete-Events (DE) les nœuds de calcul interagissent via des files d'évènements. On les utilise souvent

pour décrire des circuits numériques asynchrones. On y associe aisément les langages de design matériel (HDL) comme Verilog et VHDL.

Petri Net les nœuds d'un réseau de Pétri effectuent des calculs asynchrones avec des points de synchronisation explicites.

Ces différents modèles montrent bien l'importance de choisir son MoC en fonction de ce que l'on souhaite modéliser. Ici, nos besoins en terme de temps réel, asynchronisme et parallélisme nous poussent à nous orienter vers un modèle de type KPN.

2.1.6 Langages de description système haut-niveau

De nombreux langages de description de système existent dans la littérature et nous faisons ici une description rapide des plus notables.

UML

UML fournit un jeu de diagrammes permettant de décrire des structures logicielles graphiquement. Ces diagrammes aident les architectes logiciels à mettre en place des structures logicielles complexes. Bien que ces diagrammes individuels soient utiles pour décrire les structures logicielles, l'UML ne peut pas totalement définir de relation entre les diagrammes. Les diagrammes sont développés en tant qu'entités séparées qui expriment différents aspects du logiciel, et non comme une partie commune d'une construction. Comme résultat, la consistance entre les diagrammes est totalement laissée au concepteur.
Malgré ce problème, UML a été largement adoptée en raison de la façon dont il reflète les préoccupations et les besoins de communication des programmeurs et des concepteurs de logiciels.

Récemment, la communauté UML a travaillé sur le fait d'effectuer des analyses multiples afin de prouver des propriétés du système modélisé. Deux de ces travaux orientés vers les systèmes embarqués sont SysML et le profile MARTE (Modeling and Analysis of Real-Time and Embedded systems). SysML ajoute la possibilité de capturer des interactions avec le monde physique en utilisant des modèles mathématiques et des propriétés de vérification associées. MARTE est prévu pour ajouter des capacités de modélisation afin de vérifier des propriétés temps-réel comme l'ordonnançabilité.

SysML

SysML [108] fournit deux nouveaux diagrammes fondamentaux :

1. Diagramme des besoins
2. Diagramme paramétrique

Ces extensions modifient les diagrammes principaux plutôt que d'utiliser des mécanismes d'extension. Le diagramme des besoins cible la tracabilité et les besoins de manière plus importante. Le diagramme paramétrique exprime les relations entre le logiciel et l'environnement.
En utilisant le diagramme paramétrique, un concepteur peut modéliser de manière mathématique la relation logiciel/environnement afin de vérifier par exemple, si le logiciel peut contrôler son environnement ou d'autres propriétés. Donc la modélisation en utilisant le diagramme paramétrique est focalisée sur le temps continu où les calculs sont instantanés. Cette optique devient un désavantage quand le modèle idéal est transcrit dans un logiciel réel exécutable, où les exécutions non-instantanées compromettent la validité des analyses.

MARTE

Le but du profil MARTE [122] est d'ajouter l'analyse des propriétés temps-réel en utilisant la théorie "rate-monotonic" (les tâches possédant la période la plus petite ont la priorité la plus élevée) et la génération de code en présence de différents systèmes d'exploitations. Dans MARTE, de multiples stéréotypes sont définis. Les nouveaux stéréotypes spécifient les éléments pour modéliser trois aspects :

1. Modèle de ressource logicielle
2. Modèle de ressource matérielle
3. L'allocation du modèle logiciel sur le modèle matériel

Cette modélisation de ressource est basée sur la théorie "rate-monotonic" et inclue un mapping entre une API d'OS générique et des API d'OS spécifiques afin d'être capable de générer du code automatiquement.

En utilisant MARTE, un concepteur modélise le système avec de multiples diagrammes fonctionnels et matériels. Ensuite, les connexions entre les diagrammes sont utilisées pour modéliser les allocations d'entités d'un diagramme à l'autre. Bien sur, encore une fois la consistance entre les diagrammes est laissée à la responsabilité du concepteur.

Le profile MARTE incorpore l'expérience de la communauté AADL afin de viser la modélisation du runtime et des architectures matérielles. De plus, certains membres du comité de standardisation AADL font parti du comité de MARTE.

AADL

AADL vient traditionnellement d'un langage de programmation plutôt que d'une tradition de diagramme comme UML. AADL, comme son prédécesseur MetaH, produit des artefacts basés sur un langage de modélisation. AADL a été développé comme étant un langage de programmation, non seulement pour définir une représentation textuelle d'une architecture logicielle, mais aussi (plus important) pour définir formellement une syntaxe et une sémantique. En plus de la représentation textuelle, AADL donne la possibilité au concepteur logiciel de décrire le système graphiquement.

Les descriptions en AADL peuvent être vérifiées par des analyseurs syntaxiques et sémantiques afin d'assurer que les descriptions sont analysables et consistantes. En d'autres termes, la construction d'un modèle est analysée par le compilateur pour vérifier qu'elle est "légale" (ex : un thread ne peut pas contenir de process). La vérification d'une description se s'effectue de la même manière que lorsqu'un compilateur vérifie qu'un programme est proprement structuré, consistant et sémantiquement correct pour être capable de produire du code exécutable.
N'importe quelle description AADL est analysable et a une interprétation non-ambiguë.

SystemC/TLM

La première tendance dans la modélisation au niveau système a été d'étendre des langages existants. Le but des langages de description de niveau système (SLDL) tel que SpecC et SystemC [123] est d'enrichir le langage C/C++ pour les spécifications d'extensions (bibliothèques/langages) fournissant les concepts systèmes nécessaires, comme modéliser le temps, la concurrence et les liens de communications. Ce genre de concept est particulier à la modélisation de systèmes numériques matériels.

SystemC est un langage de description de haut niveau, puisqu'il permet une modélisation de systèmes au niveau comportemental. Conservant les fonctionnalités du C++, il reste possible de décrire des fonctions purement logicielles. SystemC permet donc de modéliser des systèmes matériels, logiciels, mixtes ou même non-partitionnés. Il est donc particulièrement approprié à la conception de systèmes de type SoC (System On Chip).
SystemC permet également de simuler le modèle conçu, puis, par raffinements successifs, d'aboutir à une représentation implémentable. SystemC est compatible avec de la simulation au cycle-près grâce à ses multiples niveaux de raffinement (TLM (Transaction Level Model), Cycle-accurate, ...).

Le langage Waveperf

Le langage Waveperf [84] a été développé dans les laboratoires de Thales Communications and Security afin de décrire les applications de type radio utilisées dans l'entreprise. Il est associé à un outil permettant de générer du code exécutable sur cible. Ce langage est basé sur les modèles de calcul KPN.
A partir d'une spécification utilisant des fichiers texte de configuration, il est possible de générer du code exécutable C++. Un fichier de configuration est requis pour chaque composant logiciel ainsi que pour la description haut niveau de l'architecture. La figure 2.7 montre l'interconnexion entre les fichiers. Les fichiers

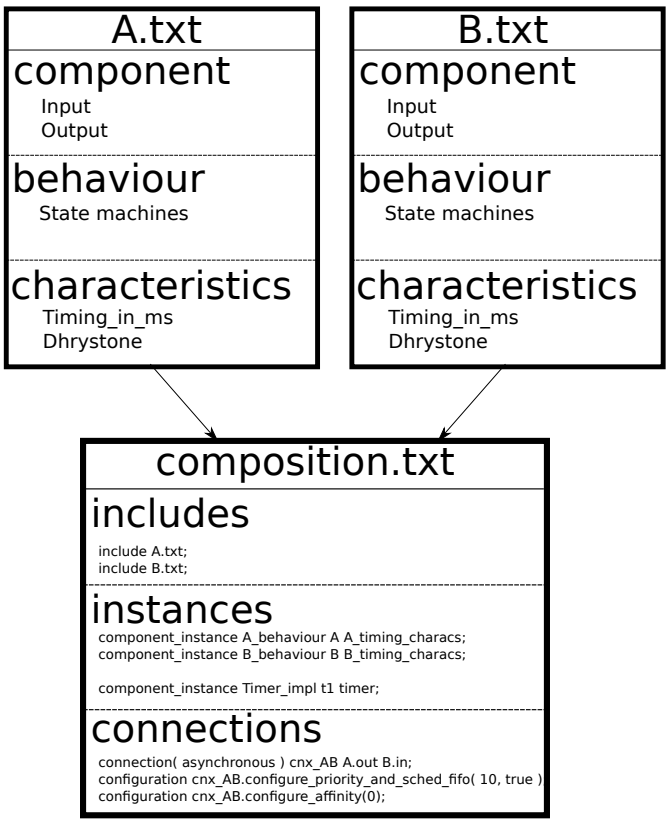

FIGURE 2.7: Utilisation des différents fichiers nécessaires à waveperf.

A.txt et B.txt représentent tous les deux une description d'un bloc logiciel (tâche). Le fichier composition.txt représente la configuration générale de l'application modélisée, c'est à dire l'instanciation et l'interconnexion entre les tâches. La syntaxe des différents fichiers sera décrite par la suite (Section 4.4).
Les fichiers de configuration d'un bloc logiciel sont divisés en trois parties distinctes comme le montre la figure 2.7 dans le cas général.

- Component : décrit la vue externe du composant avec la définition de signaux d'entrées et de sorties.
- Behavior : définit le comportement du composant lorsqu'un signal d'entrée est reçu. Pour ce faire, un état (si une machine d'état est définie), des signaux de sorties (et leurs nombre d'activation) doivent être spécifiés.
- Characteristics : définit le temps de calcul CPU ou le nombre d'opérations à exécuter lorsqu'un signal d'entrée est reçu.

Trois différents types de *Characteristics* sont possibles dans le langage. Tout d'abord, un temps durant lequel la tâche va s'exécuter. Ensuite, un temps durant lequel la tâche va être endormie. Et enfin un nombre d'opérations à exécuter par la tâche.

Les fichiers d'architecture (par exemple composition.txt) définissent la façon dont sont connectés les composants. Après avoir inclus les différents fichiers de configuration des composants, chaque composant doit être instancié avec un comportement et une caractéristique. Ensuite, les connexions entre les composants doivent être spécifiées à travers les signaux d'entrées/sorties.
Dans les fichiers d'architecture, il est aussi possible d'implémenter des composants simulant des timers. Les timers sont très utiles lorsque l'on crée des applications temps-réel.
Afin d'ajouter la possibilité de tester des comportements non-déterministes (interruptions), le port ethernet peut être utilisé pour activer une tâche. Cette fonctionnalité écoute la connexion sur le port ethernet et se

réveille quand des packets arrivent depuis le LAN (par exemple un ping fait à l'adresse IP de la plate-forme). De plus, à la fin du benchmark le nombre d'octets reçu sur le port est affiché.

Bien évidement, un timer peut aussi créer un comportement plus ou moins aléatoire étant donné qu'une tâche peut être activée à n'importe quel moment. Le fichier d'architecture permet également de connecter les différents composants afin de gérer les dépendances. Ces connexions peuvent être synchrones ou asynchrones. Une connexion synchrone est bloquante pour une tâche (A dans notre exemple 2.8) qui démarre l'exécution d'une autre tâche (e.g. B). La première conséquence est que les deux tâches s'exécutent séquentiellement sur le même processeur. Le comportement d'une connexion synchrone peut alors être assimilé à un appel de fonction. Une tâche hérite aussi de la priorité de sa tâche mère. D'un autre coté, une connexion asynchrone

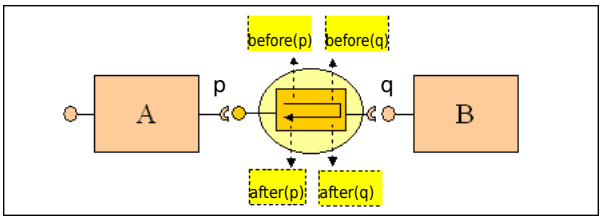

FIGURE 2.8: Connexion synchrone de la tâche A vers la tâche B.

permet l'exécution de tâches en parallèle. La figure 2.9 illustre cette exécution parallèle. Comme on peut le voir, une FIFO est insérée entre les deux tâches (A et B dans notre exemple) pour symboliser la connexion. La tâche A va tout d'abord copier les données nécessaires à la tâche B dans la FIFO puis continuer sa propre exécution. La tâche B va ensuite lire les données présententes dans la FIFO et les utiliser en parallèle. Quand une connexion asynchrone est créée, il est possible de configurer la nouvelle tâche (B dans notre exemple) avec une priorité qui sera utilisée pour l'ordonnancement. Le modèle KPN apparaît bien avec l'utilisation de

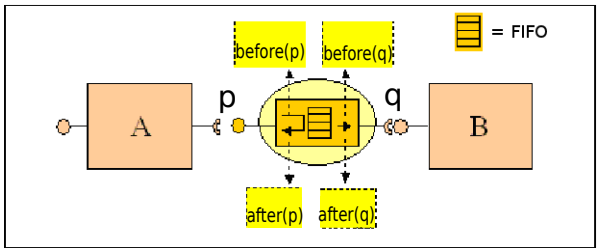

FIGURE 2.9: Connexion asynchrone de la tâche A vers la tâche B.

connexions asynchrones. Il a toutefois été amélioré avec l'ajout de connexions synchrones.
Lorsque l'on effectue les connexions entre les composants, on se doit de respecter deux règles :

- Une sortie d'un composant doit obligatoirement être connectée à l'entrée d'un autre composant. Il ne peut pas y avoir de sortie non connectée.
- Une sortie ne doit posséder qu'une seule connexion.

Il est à noter que :

- Une entrée d'un composant peut ne pas être connectée.
- Une entrée peut avoir autant de connexion que l'on souhaite.

Une fois les modèles écrits, un code exécutable est généré et permet d'exécuter le modèle de l'application soit sur une plate-forme embarquée, soit sur un ordinateur hôte.

Conclusion

Ces langages permettent d'effectuer des modélisations haut-niveau du logiciel et/ou du matériel. Par exemple, les langages basés sur UML, AAL et SystemC permettent à la fois de modéliser le logiciel et le matériel alors que Waveperf ne permet que de modéliser la partie logiciel.

L'avantage de Waveperf vient de sa capacité à générer une application exécutable à partir des modèles. De plus, le langage permet de modéliser des applications complexes utilisant des interruptions, des timers ainsi que des machines d'états à l'intérieur des tâches. La simplicité du langage (comme nous le verrons dans la section 4.4) permet de créer des nouveaux modèles rapidement.

Le choix du langage de description de l'application a été fortement influencé par les contraintes liées à mon activité de recherche en milieu industriel. En effet, dès le début de ma thèse j'ai pu me rendre compte que le langage Waveperf était largement utilisé à Thales Communications and Security pour décrire et simuler des applications. Aussi, le choix du langage de description, sans m'être imposé, s'est porté assez naturellement vers Waveperf.

2.1.7 Les techniques de profiling de l'application

Les méthodes de profiling sont très utiles afin d'estimer des performances et de la consommation d'une application. En effet, elles permettent d'obtenir des informations précises sur des caractéristiques d'exécution (nombre d'instructions, taux de cache-miss,...) et donc d'effectuer des estimations plus précises.
Nous présentons dans la suite cinq techniques de profiling.

Profiling statique

Comme vus précédemment dans la section 2.1.3, il existe plusieurs méthodologies afin d'effectuer le profiling statique d'une application. L'analyse de la complexité des algorithmes permet, à partir d'une analyse de l'algorithme d'obtenir une performance temporelle. En effet, en analysant le nombre de boucles nécessaires ainsi que les structures de données, il est possible de déduire la complexité de l'algorithme et par conséquent son temps d'exécution ou sa consommation électrique
L'analyse des dépendances d'un ordonnancement statique de tâches ou d'un graphe d'appel de fonctions peut aussi permettre d'extraire le pire-cas d'exécution. Ici, les évènements externes sporadiques et non-déterministes sont exclus. En effet, les interruptions, la récursivité de l'application, et les effets du système d'exploitation sont impossibles à estimer avec cette méthode.
Une autre méthode relativement simple consiste à compter les opérations basiques apparaissant dans le pseudo-code de l'application. L'inconvénient de cette méthode est qu'il est impossible d'estimer des effets dynamiques comme par exemple les boucles ou les branchements conditionnels.

Pour ces différentes raisons, il apparaît que cette méthode n'est pas adaptée à notre utilisation étant donné les contraintes associées (pas de prise en compte des effets dynamiques). Il est donc nécessaire d'effectuer du profiling dynamique.

Gprof

L'outil Gprof disponible sous GNU/Linux [54] produit un profil d'exécution d'une application en C, Pascal ou Fortran. Afin d'effectuer le profiling, il est nécessaire de compiler l'application avec une option précise (*-pg*). Ensuite, un graphe d'appel est créé lors de l'exécution de l'application indiquant les dépendances entre fonctions ainsi que le nombre d'appel de chaque fonction. Il y a aussi une estimation du temps passé dans chaque fonction ce qui permet de rapidement identifier les fonctions les plus gourmandes en temps de calcul (et sur lesquelles il faudra concentrer le travail d'optimisation).

Bien que ces métriques soient utiles, des informations additionnelles comme le nombre d'accès mémoire ou le nombre d'instructions exécutées sont nécessaires pour une estimation précise.

Valgrind

Valgrind[105] est un environnement d'instrumentation pour construire des outils d'analyse dynamique. La distribution Valgrind inclut pour le moment six outils :

- Un détecteur d'erreurs mémoire
- Deux détecteurs d'erreurs au niveau des threads
- Un profiler de cache et de branch-prediction
- Un générateur de call-graph incluant le profiler de cache et de branch-prediction
- Et un profiler de pile.

Ici, nous nous intéressons au générateur de call-graph avec le profiler de cache et de branch-prediction. Ceci permet de connaître, pour chaque fonction, un grand nombre d'informations.

L'utilisation de cet outil avec une exécution du code à profiler permet d'obtenir un grand nombre d'informations très utiles :

- Le nombre d'instructions exécutées
- Le nombre d'accès mémoire (lecture et écriture)
- Le nombre de cache-miss (suivant la configuration de cache donnée à l'outil)
- Le nombre de mauvaises prédictions de branchements

La figure 2.10 montre un exemple d'exécution de l'outil de call-graph sur une application de décodeur vidéo H.264. Le logiciel "Kcachegrind" est utilisé afin de visualiser les résultats.

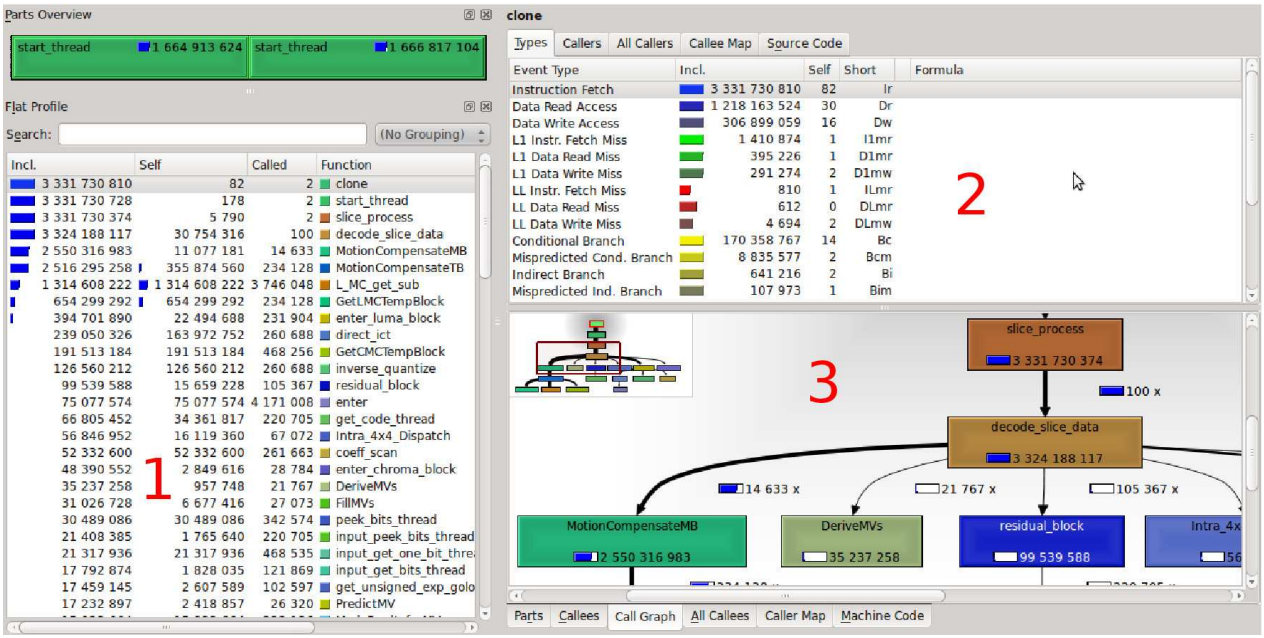

FIGURE 2.10: Exemple de la sortie de Valgrind.

Trois parties sont présentes sur la figure :

- 1 : La liste des fonctions présentes dans l'application
- 2 : Les caractéristiques de la fonction sélectionnée (nombre d'instructions, d'accès mémoires, de cache miss, etc...)
- 3 : Le call-graph de l'application.

Il est donc possible d'obtenir le call-graph pour créer notre modèle au niveau tâche (couplé à la connaissance de l'ingénieur de l'application modélisée) ainsi que la valeur des paramètres caractéristiques associés.

A partir de la, il est possible de faire différents choix de modélisation. Tout d'abord, si aucune plate-forme réelle du même type que la plate-forme que l'on cherche à modéliser n'est en notre possession, il est possible de faire du profiling sur un PC. Nous obtiendrons alors le nombre d'instructions et d'accès mémoire (load/store) pour l'architecture X86 ce qui peut entraîner (comme nous le verrons dans la section 4.1.2) des différences importantes par exemple pour le nombre d'instructions ou d'accès mémoire.

Des plates-formes comme la BeagleBoard [11] (OMAP3530) ou la PandaBoard [116] (OMAP44xx) sont facilement disponibles, il est alors relativement aisé de faire du profiling sur une architecture embarquée (de type ARM par exemple).

L'outil Valgrind est donc capable de nous fournir beaucoup d'informations sur une application ce qui permettra de compléter nos modèles logiciels avec précision. Mais il existe aussi d'autres outils, comme on va le voir par la suite.

Plate-forme virtuelle

Une autre approche permettant de faire du profiling d'une application consiste à utiliser une plate-forme virtuelle instrumentée afin d'obtenir les paramètres nécessaires. Par exemple, la plate-forme QEMU modifiée et développée durant le projet Européen COMCAS peut être utilisée dans ce but. QEMU est plutôt adapté aux architectures GPP (ARM, MIPS, PowerPC, ...) et permet de simuler du vrai code. Grâce à cette plate-forme, il est possible d'exécuter du code cross-compilé pour une plate-forme cible, et de connaître le nombre d'instructions, d'accès mémoire et de cache miss en ayant instrumenté le code préalablement (lecture de registres spéciaux destinés au profiling).
Le tableau 2.1 montre la différence du nombre d'instructions entre le profiling sur cible réelle et le profiling sur QEMU simulant la même architecture.

Nom de la plateforme	Plateforme réelle (+valgrind)	QEMU	Erreur (%)
Nombre d'instructions	1524018057	1468401460	3.65

TABLE 2.1: Comparaison du profiling sur une plate-forme réelle et sur QEMU pour l'application vidéo décodeur H.264.

L'erreur étant faible, il est tout à fait envisageable d'utiliser QEMU pour effectuer le profiling. Cependant, comme nous souhaitons faire des estimations pour différentes plates-formes matérielles, nous avons privilégié l'utilisation de Valgrind pour sa rapidité de mise en oeuvre.
De la même manière, une approche basée sur OVPSim [110][124] tente d'instrumenter la simulation afin d'en récupérer les éléments clés (nombre d'instructions, d'accès mémoire, de cache-miss, etc...).

Utilisation des compteurs de performance présents sur les cibles

Une approche un peu similaire peut-être effectuée en utilisant les compteurs de performance de plus en plus présents sur les processeurs. En effet, il est possible, sur les plates-formes embarquées, d'utiliser des compteurs spécialisés (compteur de cycle, compteur d'accès au cache, compteur de cache miss, ...) pour profiler son application (afin d'en déterminer le nombre d'instructions de chaque fonction) et en créer un modèle. Il faut dans ce cas aussi instrumenter le code (écriture de routines bas niveau permettant d'accéder aux registres des compteurs) pour permettre de lire et écrire dans les registres processeurs correspondants.

Conclusion

Lorsqu'une application réelle doit être modélisée, il est nécessaire d'effectuer une étape de profiling afin d'obtenir les informations dynamiques de l'application que l'on souhaite modéliser (nombre d'instructions exécutées par chaque tâche,...). Étant donné que nous souhaitons avoir des informations précises sur les caractéristiques et non pas seulement un workload de l'application comme dans beaucoup d'approches haut-niveau, il n'est pas possible d'utiliser du profiling statique ou l'outil Gprof.

Plusieurs solutions s'offrent donc à l'utilisateur afin d'effectuer le profiling dynamique de son application. Nous avons décidé d'utiliser Valgrind pour plusieurs raisons :

- Nous possédons plusieurs plates-formes réelles, il n'est donc pas judicieux (pour des raisons de rapidité d'obtention des résultats) d'utiliser un simulateur tel que QEMU.
- Nous préférons une solution sans instrumentation du code afin de ne pas interférer en performance ou consommation avec l'exécution de l'application.
- Valgrind offre une rapidité et une simplicité d'utilisation importante comparé aux plates-formes virtuelles.

2.1.8 Conclusion

Comme nous l'avons vu, plusieurs méthodes d'estimation sont possibles afin d'évaluer un système. Les modèles analytiques permettent le plus souvent une estimation rapide mais possèdent une erreur assez élevée et surtout ne prennent pas en compte les effets dynamiques des applications complexes, ce qui ne convient pas à nos contraintes.
Il est donc nécessaire de prendre en compte à la fois les effets dynamiques internes à l'application (boucles,...) en effectuant par exemple une simulation/profiling préliminaire mais aussi les effets dynamiques externes (interruptions) en effectuant des simulations de haut-niveau.

Nous avons alors vu les différents MoC qu'il est possible d'utiliser et celui nous semble le plus pertinent (en prenant en compte nos application) à été le KPN pour sa capacité à modéliser des systèmes asynchrones.

Différents langages de modélisation ont alors été analysés et notre choix s'est porté sur le langage Waveperf. En effet, ce langage était déjà utilisé au sein de Thales Communications and Security et se caractérise par une simplicité de modélisation des systèmes complexes.

Enfin, afin d'obtenir des informations pertinentes pour la création des modèles, nous avons choisi d'utiliser un profiler dynamique qui permet d'obtenir de nombreuses informations. En effet, Valgrind permet d'obtenir des informations de profiling dynamique et peut s'exécuter sur différentes plates-formes.

2.2 Méthodes pour explorer l'espace de conception (DSE)

Thales Communications and Security étant un équipementier, il est important de pouvoir explorer plusieurs architectures disponibles sur le marché afin d'en évaluer les capacités et de choisir la plus adaptée aux besoins du produit ciblé sans pour autant acheter chaque plate-forme.
Dans ce contexte, après avoir discuté des méthodes utilisées pour évaluer un point de conception unique, cette section donne un aperçu des algorithmes utilisés afin de raisonnablement couvrir l'espace de conception. Explorer l'espace de conception est un processus itératif qui est habituellement basé sur une approche Y-chart [80]. La figure 2.11 montre les différentes étapes d'une approche en Y-chart pour l'exploration d'architecture. Ici, les descriptions de l'application et de l'architecture sont explicitement liées l'une à l'autre dans une étape de mapping et évaluées par la suite. Le mapping pourrait inclure des phases de compilation et de synthèse afin de permettre l'analyse des performances. Les résultats de l'évaluation de ce point de conception particulier pourraient alors être utilisés pour mieux guider l'exploration en faisant varier les descriptions d'applications et d'architecture ainsi que la correspondance entre les deux.

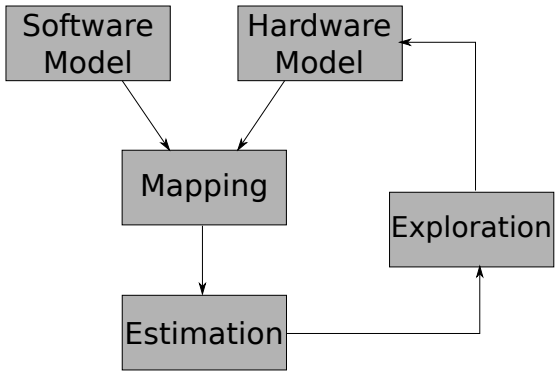

FIGURE 2.11: Représentation Y-chart pour l'exploration de l'espace des conceptions.

Dans les sections suivantes, nous décrivons rapidement les différentes stratégies d'optimisation ainsi que quelques métriques couramment utilisées dans l'embarqué. La dernière section aborde les stratégies de simplification de l'espace des conceptions.

2.2.1 Stratégie d'optimisation

Lorsque plusieurs objectifs sont utilisés pour explorer les différentes architectures possibles, il est nécessaire d'introduire un nouveau terme afin d'être capable de déterminer quelle architecture est la plus adaptée. L'optimalité de Pareto [111] est utilisé afin de prouver qu'une solution (ou un groupe de solutions) est bien la meilleure au vue des objectifs choisis.

Def. 1 (**Le critère de Pareto pour la dominance**). *Soit k objectifs à minimiser sans perte de généralité, et deux solutions (design) A et B avec les valeurs* (a_0 , a_1 , . . . , a_{k-1}) *et* (b_0 , b_1 , . . . , b_{k-1}) *pour tous les objectifs, respectivement, la solution A domine la solution B si et seulement si*

$$\forall_{0\leq i<k}\ \ i : a_i \leq b_i \qquad et \qquad \exists_j : a_j < b_j.$$

Cela signifie que, une solution est supérieure à une autre si elle est meilleure dans au moins un objectif tout en étant au moins le même dans tous les autres objectifs. Une définition plus rigoureuse de la dominance stricte exige que A soit meilleur dans tous les objectifs par rapport à B, alors que la définition moins forte de la dominance faible ne nécessite que la condition $\forall_{0\leq i<k} i : a_i \leq b_i$.

Def. 2 (**Solution Pareto-optimale**). *Une solution est appelée Pareto-optimale si elle n'est pas dominée par une autre solution. Les solutions non-dominées forment un ensemble Pareto-optimale dans lequel aucune des solutions n'est dominée par toute autre solution dans l'ensemble.*

Les méthodes d'optimisation peuvent être classées par rapport aux critères suivants (voir [67] et les références présentes) :

- La prise de décision avant la recherche : le concepteur décide comment regrouper les différents objectifs en une seule et même fonction d'objectif (coût) avant que la recherche à proprement parler soit effectuée.
- Recherche avant la prise de décision : la recherche de solutions optimales est effectuée avec de multiples objectifs à l'esprit qui sont séparés pendant la recherche. Le résultat de la recherche est un ensemble de solutions Pareto-optimale.
- La prise de décision lors de la recherche : cette catégorie est un mélange des deux groupes précédents. Ici, les étapes initiales de recherche peuvent être utilisées pour restreindre davantage l'espace de conception et / ou guider la recherche dans certaines régions de l'espace de conception.

Ainsi, le choix d'un algorithme de recherche mono ou multi-objectif non seulement influe sur le temps où les objectifs de conception sont définis, mais affecte également le processus d'exploration dans son ensemble.

En utilisant une recherche mono-objectif, le résultat de l'optimisation est un point de conception unique. Cela signifie que, les recherches doivent être répétées avec, par exemple, différents poids ou contraintes sur la fonction d'objectif afin d'explorer l'espace de conception et de générer un ensemble de solutions Pareto-optimales. Selon la forme de la fonction d'objectif qui agrège plusieurs objectifs, certaines régions de l'espace de conception peuvent ne pas être atteignable.s

Contrairement à cela, les recherches multi-objectifs sont potentiellement en mesure de trouver les différentes solutions Pareto-optimale dans un cycle d'optimisation unique. Le choix pour l'une des solutions dépend de contraintes supplémentaires ou de fonctions d'objectifs qui s'appliquent en combinaisons des objectifs utilisés pour la recherche.

2.2.2 Les différentes métriques et objectifs dans l'exploration d'architectures

Bien choisir l'objectif à atteindre lors d'une exploration/optimisation d'architecture est très important. Nous allons ici, lister les principaux objectifs possibles pour l'exploration.

Les objectifs d'optimisations représentent souvent les propriétés de l'ensemble du système évalué. Les métriques citées ci-dessous peuvent être utilisées dans les explorateurs du domaine des systèmes embarqués.

- Le coût : Le coût est un objectif très important à étudier. Il est toujours utile d'être capable d'optimiser le coût d'une conception que ce soit pour une entreprise, ou un laboratoire de recherche. On peut par exemple effectuer la somme des divers composants présents dans le système.
- La dissipation d'énergie : La consommation d'énergie étant de plus en plus l'objet de toutes les attentions, cet objectif est fortement utilisé depuis quelques années. Les systèmes étaient plutôt optimisés pour la vitesse, mais les pertes d'énergies et la dégradation de la durée de vie des composants liée au problème thermique a conduit à l'optimisation en dissipation d'énergie. Cependant, plusieurs types de dissipation apparaissent, comme la puissance dynamique et la puissance statique induite par les fuites de courant.
- La performance temporelle : La performance temporelle (souvent appelée vitesse) d'un système est cruciale, particulièrement pour les systèmes embarqués temps réels. Cette métrique peut mesurer différentes performances suivant le domaine d'utilisation. Par exemple, le débit de calcul réalisé, le débit des communications sur les différents bus ou le temps de latence ou de réponse de certains événements.
- La taille physique : La taille et le poids physique d'un système peuvent être d'une importance primaire pour les systèmes embarqués comme par exemple dans le domaine de l'automobile, qui affectent

particulièrement le coût des systèmes.
- Le temps de mise en place du système : Cette métrique prend en compte la complexité du système (le logiciel d'un système many-coeurs va être plus long à mettre ne place que celui d'un mono-coeur par exemple) mais aussi le type d'environnement de développement.

Il est bien entendu possible de combiner certaines métriques afin d'évaluer conjointement plusieurs aspects de l'architecture.

Une fois les métriques bien choisies pour le système, il est primordial d'être capable d'implémenter une méthode d'exploration et/ou de simplification de l'espace afin d'accélérer le processus.

2.2.3 Stratégies de simplification et de parcours de l'espace des conceptions

L'espace des conceptions pouvant être très rapidement important, il est nécessaire d'une part de le réduire, et d'autre part d'essayer de le parcourir le plus intelligemment possible. Nous aborderons donc plusieurs méthodes permettant de réduire l'espace et de couvrir les parties de l'espace les plus intéressantes.

Évaluer exhaustivement chaque point de conception possible. Ceci est l'approche de base pour l'exploration de l'espace des conceptions. Elle consiste à estimer chaque point de l'espace l'un après l'autre. Cette approche est bien évidement inutilisable lorsque l'espace est important et comporte plusieurs paramètres.

Exploration hiérarchique. L'exploration hiérarchique consiste à classer des régions intéressantes de l'espace des conceptions dans un premier temps, puis d'utiliser un modèle plus fin pour explorer ces régions. Ceci permet d'accélérer l'exploration globale en ne faisant qu'une estimation gros grain sur tout l'espace, et de faire une exploration précise uniquement sur un sous-ensemble de l'espace.

Sous-échantillonnage de l'espace des conceptions. La méthode la plus classique et simple à mettre en oeuvre pour réduire l'espace est de le sous-échantillonner. Cela permet d'obtenir une exploration non-biaisée et plus rapide qu'une exploration exhaustive.
Plusieurs types de sous-échantillonnage peuvent alors être utilisés.

Aléatoire : Effectue un sous-échantillonnage de manière totalement aléatoire dans l'espace. Cela permet d'obtenir assez rapidement des points partout dans l'espace mais possède l'inconvénient de ne potentiellement jamais passer dans une zone. Il est nécessaire d'estimer un grand nombre de points afin d'être certain de n'avoir oublié aucune zone de l'espace.

Régulier : Le sous-échantillonnage s'effectue à partir d'une grille régulière. Cette méthode est souvent utilisée en limitant le nombre de valeurs possibles des paramètres. Par exemple, si l'on prend la fréquence comme paramètre d'exploration, il suffit de ne faire que des sauts de 50MHz plutôt que d'explorer chaque fréquence pour avoir un sous-échantillonnage régulier.
Toutes les approches qui quantifient les paramètres de conception pour réduire l'espace des conceptions appliquent un motif de sous-échantillonnage régulier.

Le sous-échantillonnage est une approche efficace et simple qui ne nécessite que peu d'effort à implémenter. C'est pour cette raison qu'il est très souvent utilisé.

Contraindre l'espace des conceptions. Cette tâche simple mais tout de même très importante consiste à identifier les différentes contraintes de l'espace des conceptions. C'est souvent une étape initiale qui est effectuée dans les outils permettant l'exploration. Cela permet de restreindre l'espace grâce par exemple à l'utilisation d'une méthode analytique pour déterminer les pires-cas et ainsi connaître les cas critiques de l'espace. Il est aussi possible de limiter certains paramètres par des bornes.

Recherche non-orientée et la recherche guidée. Les explorations classiques exécutent des recherches non-orientées. C'est à dire qu'elles recherchent dans tout l'espace disponible (que ce soit exhaustif, sous échantillonné ou contraint) jusqu'à l'évaluation de toutes les solutions. La recherche non-orientée va donner

au concepteur, une vision impartiale de l'espace des conceptions. Dans les recherches orientées, on va guider l'exploration pour qu'elle arrive le plus vite possible aux meilleures solutions sans pour autant évaluer toutes les solutions possibles. On va donc devoir ajouter de la connaissance du domaine de l'espace des conceptions pour être capable de réaliser une exploration guidée.

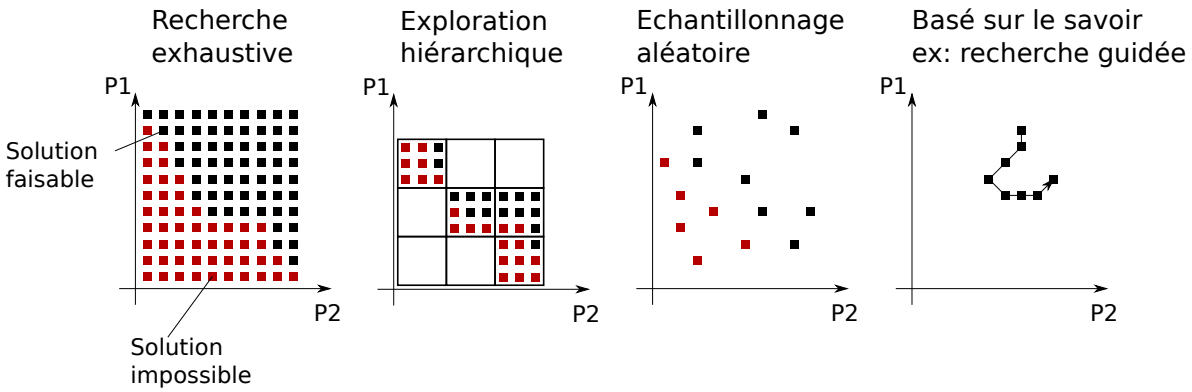

FIGURE 2.12: Approches habituelles pour couvrir l'espace des conceptions. Un espace discret à deux dimensions est présenté.

En résumé, la figure 2.12 décrit graphiquement différentes stratégies de recherche pour couvrir l'espace des conceptions. Un espace de conception discret est défini par deux paramètres de conception (dans l'espace des problèmes) ou deux contraintes de conception (dans l'espace des solutions) P1 et P2.

2.2.4 Conclusion

Nos contraintes nous obligeant à choisir une solution rapide permettant d'analyser plusieurs objectifs, il est impossible d'explorer exhaustivement tout l'espace. De plus, un sous-échantillonnage aléatoire n'est pas non plus envisageable car il est alors possible soit de manquer des solutions intéressantes, soit d'attendre longtemps avant d'obtenir une solution optimale. Le sous-échantillonnage régulier est tout de même envisageable, par exemple pour la fréquence.

Effectuant cette thèse dans un contexte industriel, nous avions déjà une grande connaissance des plates-formes ainsi que des possibilités associées. C'est pour ces raisons que nous préférons utiliser une exploration guidée afin de bénéficier de nos connaissances.

2.3 Les plates-formes d'outils disponibles pour le DSE

De nombreux environnements de conception ont été développés par les universités et les industries. Certains d'entre eux supportent aussi l'exploration de l'espace des conceptions que ce soit au niveau système ou au niveau de la micro-architecture. La prochaine sous-section aborde quelques exemples d'outils permettant de donner une vision des outils et approches existants. Une comparaison entre différents outils sera ensuite effectuée.

2.3.1 Plate-formes d'outils au niveau système

Les outils de cette catégorie permettent de modéliser et d'évaluer les architectures et des applications sur différents niveaux d'abstraction en utilisant différents modèles de description. Ils vont donc aussi souvent permettre de coupler des outils d'évaluation de différentes sources et fournisseurs afin de permettre l'évaluation des architectures hétérogènes au niveau système.

Metropolis [10]. Université de Californie à Berkeley, Politecnico di Torino, et Cadence Berkeley Labs. Metropolis est un framework qui permet la description et le raffinement d'un design à différents niveaux d'abstraction et intègre la modélisation, la simulation, la synthèse et les outils de vérification. La fonction d'un système, telle que l'application, est modélisée comme un ensemble de processus qui communiquent à travers les médias. Un comportement non-déterministe peut-être modélisé et les contraintes peuvent restreindre l'ensemble des exécutions possibles. Les blocs d'architecture sont représentés par des modèles de performance où les événements sont annotés avec des coûts d'intérêt. Des annotations supplémentaires pourraient inclure des informations arbitraires, comme une demande d'accès à une ressource partagée, qui est soumise à un contrôle centralisé par un gestionnaire de quantité. Une correspondance entre les modèles fonctionnels et l'architecture est déterminée par un troisième réseau qui corrèle les deux modèles par des événements de synchronisation (en utilisant des contraintes) entre eux.

Artemis [114]. Universités de Amsterdam, Delft, Leiden, NL, et Philips Research. Artemis est fondé sur une description en réseaux de processus de Kahn de l'application et intègre les idées de SPADE [86], c'est à dire que la co-simulation du système est effectuée en utilisant des traces d'instructions symboliques générées et interprétées lors de l'exécution par les modèles de performance abstraits. Il ajoute des aménagements afin

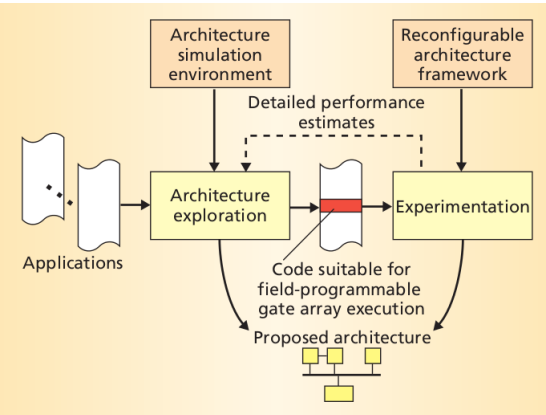

FIGURE 2.13: Représentation graphique du flot Artemis [114].

d'explorer les architectures reconfigurables et pour le raffinement des modèles d'architecture. Dans [115], une couche supplémentaire de processeurs virtuels est introduite entre les applications et les architectures en vue

de résoudre les blocages possibles en raison de décisions de mapping.

Rabbit (Plate-forme basée sur QEMU) Laboratoire TIMA [139] (CNRS). Rabbit est une plate-forme basée sur QEMU et SystemC permettant à la fois d'exécuter du code réel sur la partie processeur, mais aussi d'implémenter des périphériques en SystemC. Cet environnement améliore l'émulateur QEMU afin d'y ajouter la notion de temps. Il est alors possible d'annoter les différents modèles de processeurs disponibles avec le nombre de cycle de chaque instructions. L'avantage de cette méthodologie est d'être capable d'exécuter le vrai code qui s'exécutera sur la plate-forme embarquée. De plus, des modèles de consommation haut-niveau sont aussi implémentables. Différents modèles de raffinement pour les accès mémoire sont aussi possibles, ce qui permet d'effectuer des simulation plus ou moins rapides/précises. Cependant, le temps de simulation augmente fortement si le niveau de précision simulé est très important. En effet, le temps de simulation de 10 benchmarks peut aller de 15 minutes pour un niveau d'abstraction très élevé, jusqu'à une journée lorsque les niveaux de caches et l'interconnexion sont très finement modélisés [83].
L'exploration d'architectures automatique n'est pas disponible dans cet outil mais le nombre important de modèles de processeur permet d'explorer de nombreuses possibilités manuellement.

OpenPeople [132] Ce projet développé par plusieurs universités françaises permet de décrire des SoC dans le langage AADL. Cela donne la possibilité d'ajouter des contraintes et d'analyser les modèles. Cet outil a été développé afin de partager différents modèles de consommation d'énergie dans une même plate-forme. Des modèles de processeurs, mais aussi de DSP ainsi que de FPGA sont disponibles. En plus de la plate-forme de modélisation, un banc de mesure de la consommation d'énergie à distance a aussi été développé, ce qui permet la vérification de ses modèles.

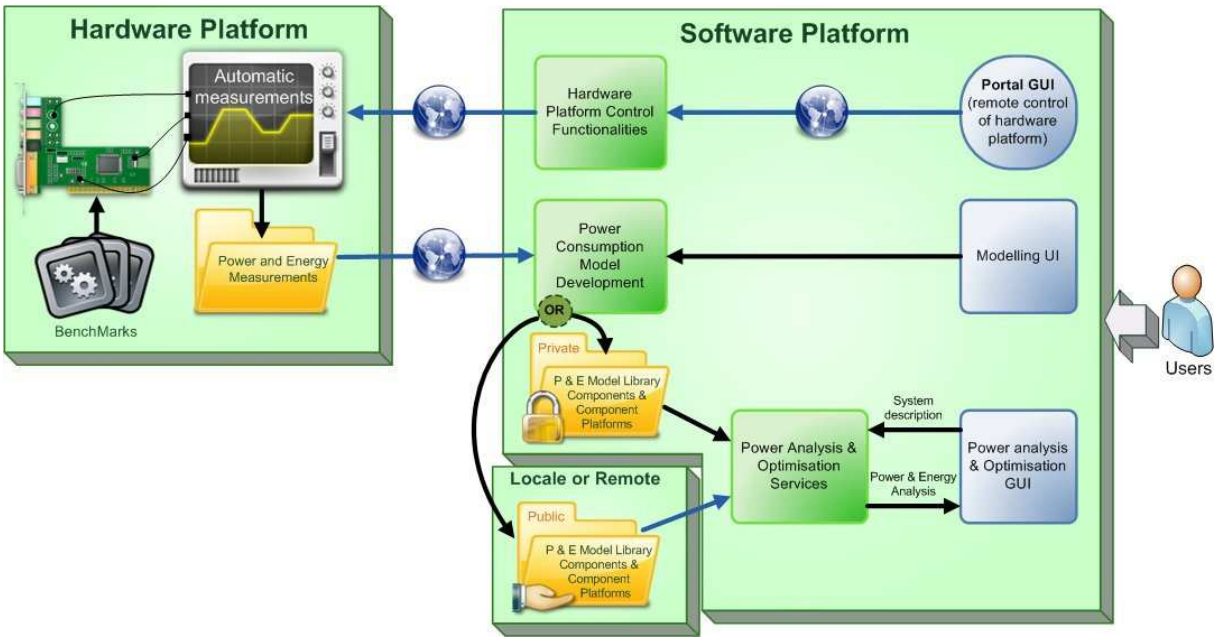

FIGURE 2.14: Illustration globale de la plate-forme Open-PEOPLE. [4]

La figure 2.14 montre le schéma global utilisé dans le projet Open-PEOPLE. On peut y voir d'un côté la plate-forme matérielle permettant d'effectuer des mesures afin de créer des modèles, et d'un autre côté la plate-forme logicielle permettant d'effectuer les modélisations.

Ce projet permet avec un unique langage (AADL) d'estimer la consommation d'énergie d'un système avec différents modèles. Une des méthodologies d'estimation de la consommation d'énergie repose en deux

parties.
Dans un premier temps, des modèles de consommation de la plate-forme sont élaborés. Ces modèles reposent sur la méthodologie FLPA (Functional Level Power Analysis [97]) qui consiste à identifier un jeux de blocs fonctionnels qui influent fortement sur la consommation d'énergie du composant ciblé. Deux types de paramètres sont identifiés dans [124], les paramètres algorithmiques qui dépendent de l'algorithme exécuté (ex : taux de cache-miss et instructions par cycle) et les paramètres architecturaux qui dépendent de la configuration du composant (ex : la fréquence).
Une caractérisation de la variation de consommation d'énergie lorsque les paramètres varient est alors effectuée afin d'en déduire des lois. Ceci est fait à l'aide de programmes élémentaires en assembleur (que l'auteur appelle scénario) ou des vecteurs de test élaborés pour stimuler chaque bloc séparément.

La deuxième partie consiste à décrire l'architecture matérielle à l'aide d'un simulateur (OVPSim). Ce simulateur est modifié et amélioré afin, lorsqu'il exécute l'application, de compter les différentes activités nécessaires à l'estimation de consommation (taux de cache-miss, instructions par cycle...). Le simulateur est alors utilisé ici comme un profiler permettant d'obtenir des informations dynamiques sur l'application.
L'estimation de consommation est alors ensuite faite à l'aide des lois déduites dans la première partie et de la simulation effectuée dans la deuxième partie.

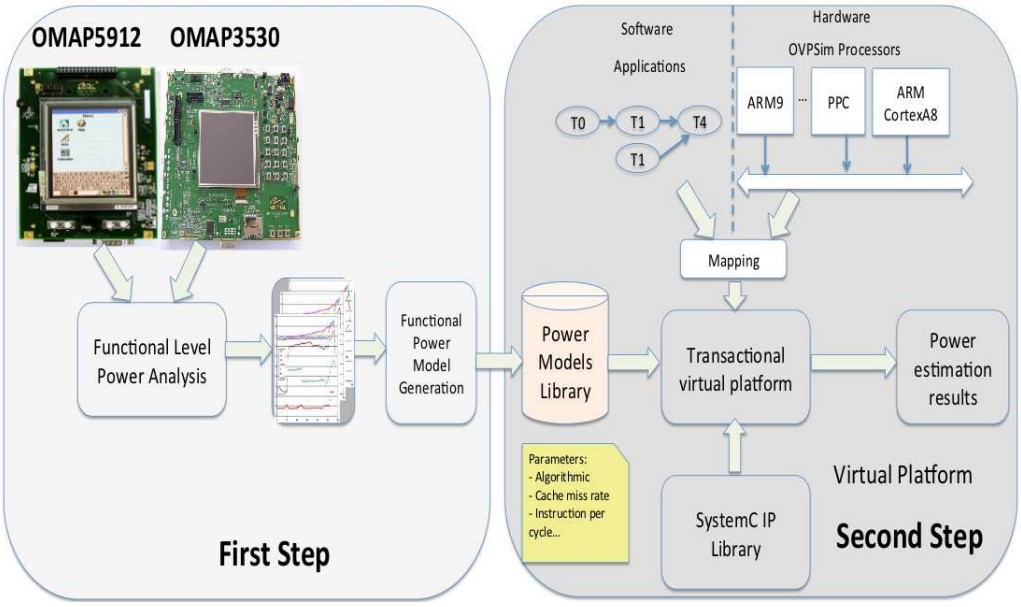

FIGURE 2.15: Une des méthodologies d'estimation de la consommation d'énergie dans le projet OpenPEOPLE. [124]

La figure 2.15 montre les deux différentes étapes de cette approche, la mesure puis la simulation.

L'article [124] déduit des mesures un modèle de consommation pour le processeur ARM Cortex-A8 : $P(mW) = 0.79 \cdot F_{Processor} + 18.65 \cdot IPC + 0.26 \cdot (y_1 + y_2) + 10.13$
Où :
- $F_{Processor}$ représente la fréquence du processeur.
- IPC représente le nombre d'instructions par cycle.
- y représente le taux de cache-miss.

Cette approche est intéressante du fait de sa précision (4% maximum) pour un niveau d'abstraction assez haut.
Ici encore, l'exploration d'architecture est manuelle pour le moment, mais plusieurs travaux visent à automatiser le processus d'exploration.

2.3.2 Comparaison

Nous avons complété le tableau présent dans [58] avec les derniers outils mentionnés. On distingue les catégories suivantes dans le tableau 2.2 :

- *Source/origine du framework* : A(cademique) ou C(ommercial).
- *Description de l'application* : Représentation de l'application utilisée par l'outil, comme les réseaux de processus de Kahn (KPN), modèles de calcul (MoC), les langages de programmation comme C, C++, le flux synchrone ou dynamique des données (SDF et DDF), et machines d'états finis (FSM).
- *Description de l'architecture* : la représentation de l'architecture utilisée, tel que des modèles de performance, la description de la micro-architecture à l'aide des langages de description d'architecture (ADL), ou un langage de description de matériel (HDL). La notation "Abstrait au HDL" signifie que le concepteur dispose de plusieurs options et, selon la représentation d'architecture choisie, le niveau d'abstraction peut varier de modèles de performance abstraits à des descriptions HDL fin. ISS représente un simulateur du jeu d'instructions.
- *Les modes d'exploration* : Les options pour explorer l'espace de conception, par exemple, automatique vs recherche manuelle afin d'évaluer plus d'un design. L'exploration basée sur scripts dans ce contexte signifie que les outils correspondants sont en mesure de découvrir automatiquement les conceptions exhaustives en évaluant toutes les permutations possibles des paramètres de conception. Les paramètres de conception sont exportés, par exemple, par les modules SystemC et représentent des décisions de conception, telles que la vitesse d'horloge, taille du cache, et la largeur en bits des opérations. Les interconnexions, c'est à dire la topologie de la conception, ne peuvent pas être changées automatiquement par ces scripts.
- *Couplage d'outils* : La possibilité de couplage d'outils, en particulier, des simulateurs, des différents framework et des fournisseurs tiers afin d'évaluer un système hétérogène est montrée ici. L'abstraction de modélisation SystemC au niveau transactionnel est souvent utilisée à cette fin.
- *Cible du design* : L'objectif principal de l'outil de conception est affiché dans cette colonne. Centré sur la conception au niveau système ("système") implique que les systèmes hétérogènes multi-processeurs peuvent être décrits. L'accent est mis, en particulier sur le support à différents niveaux d'abstraction et de chemins de raffinement. Il signifie, aussi, souvent, que SystemC est pris en charge et les outils, tels que des simulateurs, à partir de sources différentes, peuvent être combinés pour l'évaluation au niveau système. Des outils centrés sur la micro-architecture ("micro") se concentrent généralement sur les systèmes à processeur unique, où la génération automatique des compilateurs optimisés et des simulateurs devient important.
- *Outils générés* : Les outils basés sur ADL supportent la génération automatique d'outils d'évaluation et de développement, tels que des simulateurs ("sim"), l'assembleur ("asm"), et les compilateurs ("comp").

On pourra noter que nous ne distinguons pas les outils concernant leur méthode d'évaluation. En dehors de EXPO [114], qui est un framework analytique, tous les outils sont basés sur la simulation et se concentrent principalement sur la performance. Habituellement, la précision des modèles d'architecture utilisés détermine la précision de l'évaluation, par exemple, instructions-près vs cycle-près. Si les modèles HDL peuvent être intégrés dans l'évaluation, d'autres résultats de la synthèse, comme la consommation de surface, sont disponibles.

Parmi ces nombreux frameworks, certains ne répondent pas à nos exigences. Par exemple, les frameworks ciblant la micro-architecture comme EXPRESSION et LisaTek et ceux ciblant les mémoires comme Atomium ne correspondent pas à nos besoins d'exploration de systèmes. Ensuite, les outils utilisant un niveau HDL pour modéliser le matériel ne conviennent pas non plus du fait de leur niveau trop précis de modélisation. De plus, le temps pour simuler du HDL est très long. De même, l'utilisation d'ISS ou de simulation précise à l'instruction près nécessiterait trop de temps d'exploration.
Enfin, le besoin d'avoir une exploration automatique nous contraint aussi à éviter les différents outils ne procurant qu'une exploration manuelle.

Name	Source	Application model	Architecture model	Exploration mode	Tool coupling	Design focus	Tool generation	Notes
Artemis	A	KPN	abstract perf. model	manual	no	system	no	Based on SPADE.
ASIP-Meister	A	'C'	ADL	manual	no	micro	sim, comp	Uses CoSy compiler from ACE.
Atomium	C	'C'	N/A	manual	no	memory	no	Optimization of memory subsystem.
Chess/Check.	C	'C'	ADL	manual	no	micro (generated)	sim, comp	
CoCentric	C	FSM, PN, SDF, DDF, Matlab, SystemC	abstract to HDL	manual, scripts	yes	system	no	
COMCAS	A	'C'	instr.-accurate	manual	no	system	no	Based on QEMU
EXPO	A	DAG, task-level	abstract, accumu-lated	evolutionary optimizer	no	system	no	Based on analytical evaluation.
Expression	A	C++	ADL	manual	no	micro (generated)	sim, comp	
LisaTek	A+C	assembler	ADL	manual	no	micro	sim, asm	
MaxCore	C	assembler	ADL	manual	no	micro	sim, asm	
Mescal	A+C	mixed MoC, assembler	ADL	partly automatic (memory)	(planned)	system+ micro (generated)	sim, asm	
Metropolis	A+C	mixed MoC (Meta Model language)	abstract perf. model	manual	yes	system	sim	
MILAN	A	C, Matlab, Java, SDF	abstract to HDL	manual	yes	system	no	
Open-PEOPLE	A	'C'	ADL	manual	(planned)	system	no	
PICO	A+C	'C'	micro-arch. templates	automatic, heuristics	(planned)	system+ micro (generated)	comp, sim	Templates for VLIW, cache, co-processor.
SPADE	A	KPN	abstract, instr.-accurate	manual	no	system	no	
SPW	C	FSM, C++, Matlab, SystemC	abstract to HDL	manual, scripts	yes	system	no	
StepNP	A+C	Click	abstract, ISS	manual	yes	system	no	

TABLE 2.2: Comparaison des outils d'exploration de l'espace des conceptions.

2.4 Conclusion

Nous avons dans cette partie présenté différentes méthodologies permettant principalement d'estimer la performance d'un système donné, mais aussi pour explorer différentes architectures et faciliter son exploration. En effet, obtenir une bonne estimation de performance est un prérequis primordial afin d'effectuer de l'estimation de consommation précise.
Il en est ressorti que pour nos contraintes (obtenir une estimation très rapide sans avoir de plates-formes et en ayant seulement accès aux documents des constructeurs), les méthodologies de simulations au cycle-près avec des modèles très précis ne sont pas envisageables. En effet, la création de tels modèles est très complexe (impossible avec uniquement une datasheet) et les simulations basées sur ces modèles sont longues.
De même, la solution consistant à utiliser une traduction du binaire (émulateur type QEMU) pour accélérer le temps d'exécution et les estimations n'est pas satisfaisante. Le temps d'exécution reste relativement im-

portant alors que la précision des estimations se dégrade fortement.
D'un autre coté, une analyse purement analytique, bien que très rapide à l'exécution, permet difficilement d'analyser un comportement dynamique d'une application (interruption, boucles de contrôle, etc...).
Le profiling dynamique répond à une partie des problèmes cités plus haut. Il permet d'analyser le code tout en prenant en compte le comportement dynamique interne aux tâches (boucles de contrôle). Il ne permet cependant toujours pas d'analyser le comportement exact de l'application comme les interruptions.
La simulation au niveau système (à haut niveau d'abstraction) est la méthode la plus appropriée pour satisfaire nos contraintes. En effet, cette méthodologie permet des temps de simulation rapides et nécessite aussi un effort initial assez faible.

D'un point de vue purement énergétique, l'estimation de la consommation effectuée sur des niveaux bas de la conception est réalisée avec une grande précision mais sont généralement longues et complexes compte tenu de la complexité des architectures actuelles. A l'inverse au niveau système, l'évaluation de la consommation est moins fiable mais s'obtient dans un temps beaucoup plus court. C'est aussi au niveau système que les gains les plus notables sont attendus car les méthodes d'optimisation de l'énergie s'appuient à ce niveau sur un plus grand nombre de mécanismes de gestion de la consommation tout en ayant une connaissance de l'activité de l'ensemble du système. Aussi dans le cadre de mes activités de recherche, j'ai choisi de travailler à ce niveau d'abstraction.

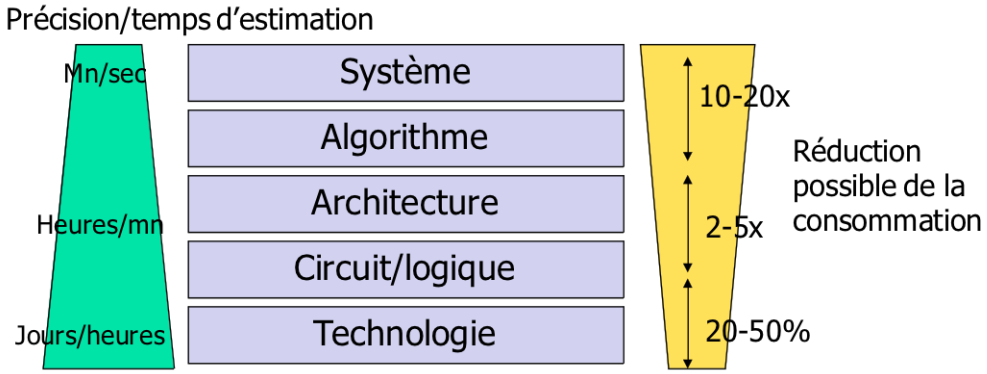

FIGURE 2.16: Évolution du temps d'estimation et de l'optimisation possible en consommation d'énergie suivant le niveau de modélisation.

La figure 2.16 montre d'un coté que la précision et le temps d'estimation vont en augmentant lorsque le niveau de modélisation est plus fin, et d'un autre coté que le gain potentiel dû à l'optimisation est de plus en plus faible.

Nous avons donc choisi dans notre approche de coupler à la fois le profiling dynamique pour nous permettre d'obtenir des informations précises sur l'application (en prenant en compte son comportement dynamique interne comme les boucles de contrôles) et des modèles paramétriques pour obtenir une estimation des performances et de la consommation des différentes tâches. Sur la base de ces informations et de ces modèles, nous avons également utiliser un simulateur de tâches pour exécuter l'interaction entre les différents blocs logiciels ce qui permet d'analyser le comportement dynamique inter-tâches. Une méthodologie en Y-chart a été adoptée afin de permettre de cloisonner le logiciel et le matériel et ainsi de simplifier l'exploration d'architectures.
Pour l'exploration, nous avons choisi tout d'abord d'échantillonner en partie l'espace (pour la fréquence par exemple) puis d'utiliser nos connaissances des systèmes embarqués pour effectuer des recherches guidées.

Il nous a donc été nécessaire dans un premier temps de trouver et choisir les paramètres clés (logiciel et matériel) ayant le plus d'impacts sur la performance et la consommation d'énergie tout en pouvant être extraits de datasheet.

Chapitre 3

Etude des paramètres dimensionnant l'estimation des performances et de la consommation

Notre approche, décrite dans le prochain chapitre, se base sur une série de paramètres matériels permettant la description de l'architecture et l'estimation de ses performances. Le but de notre approche étant de créer des modèles simples et facilement implémentables, il nous fallait étudier le fonctionnement de ces différents paramètres et analyser leurs effets sur les performances et la consommation.

Pour ce faire, des expérimentations ont été menées pour déterminer les paramètres dont l'influence était la plus importante afin de modéliser uniquement les paramètres ayant le plus d'influence sur les performances ou la consommation. La plate-forme OMAP3530 (décrite dans le chapitre Résultats) a été utilisée pour effectuer ces tests. Une représentation simplifiée de cette plate-forme est décrite dans la figure 3.1. Comme

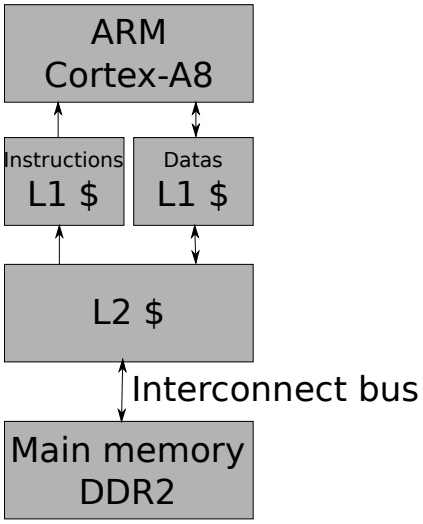

FIGURE 3.1: Description simplifiée de l'OMAP3530 coté GPP.

nous pouvons le voir, le processeur ARM Cortex-A8 possède deux caches de niveau 1, un pour les données,

l'autre pour les instructions. Ces deux caches sont eux-mêmes reliés à un cache de niveau 2, puis à la mémoire externe de type DDR2 via un bus d'interconnexion L3.
Un exemple de test montrant l'influence des divers paramètres est décrit ci-dessous. Le cache L2 est activé ou désactivé, la fréquence du bus L3 est paramétrée de 83MHz à 166MHz et la fréquence du processeur varie de 50 à 800MHz.

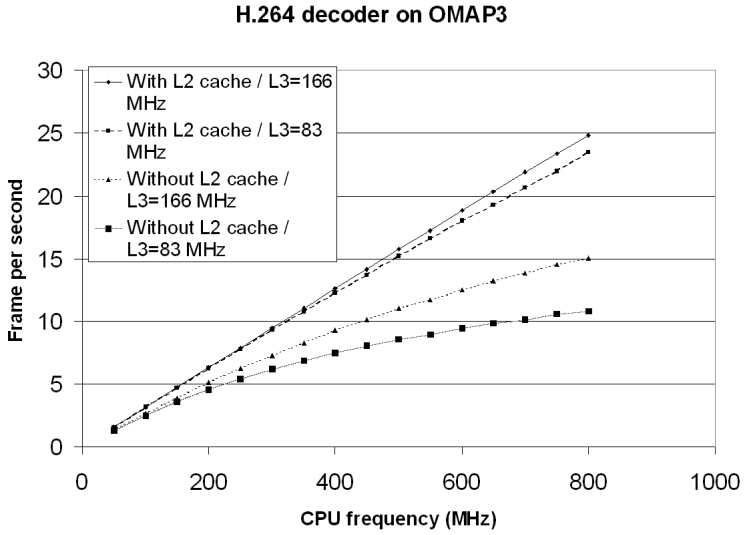

FIGURE 3.2: Décodeur vidéo H.264 s'exécutant sur un processeur ARM Cortex-A8 avec différents paramètres matériel.

La Figure 3.2 montre les résultats obtenus pour différentes configurations d'architecture de la plate-forme OMAP3. Dans le but de mesurer la performance des différentes configurations, le décodeur H.264 (décrit dans la section 6.1.2) essaie de décoder aussi rapidement que possible un maximum d'images, ce qui signifie que le nombre d'images par seconde en entrée n'est pas fixé. Tout d'abord, nous pouvons observer sur la figure 3.2 que lorsque le cache L2 est activé, les performances globales restent à peu près identiques (seulement 0.5% à 6% de perte) quel que soit le choix de la fréquence du bus L3. D'un autre coté, si la fréquence du bus L3 est fixée à 166MHz, le nombre d'images décodées par seconde va être sensiblement diffèrent si le cache L2 est activé ou désactivé. Les performances vont ainsi décroître de 14 à 39% quand le cache L2 est désactivé (par rapport à une solution avec le cache L2 activé). Finalement, nous pouvons voir que la fréquence du bus L3 est un paramètre influençant de manière non négligeable le nombre d'images décodées par seconde quand le cache L2 est désactivé. Comme le montre la figure 3.2, dans ce cas, une baisse de performance de 3.5% à 28% peut être observée. Pour conclure, ces expériences démontrent clairement que la performance (ici en terme de nombre d'image décodées par seconde) de l'application décodeur H.264 dépend énormément de ces paramètres matériels. De plus, tous ces paramètres architecturaux sont interdépendants.

Il apparaît donc qu'une approche globale est nécessaire pour rapidement obtenir des estimations en performance et en consommation pertinentes et ainsi explorer différentes architectures afin de sélectionner la solution qui présente le meilleur compromis performance/consommation tout en respectant les contraintes temps-réel. Le tableau 3.1 présente trois différentes configurations qui respectent une qualité de service de 10 images par secondes. Chaque configuration correspond à différents choix architecturaux. L'objectif de notre approche consiste à définir une méthode qui aidera les ingénieurs système à déterminer les différentes solutions architecturales qui respectent les contraintes de temps. L'environnement ainsi proposé devra également permettre d'aider l'ingénieur système dans le choix de la configuration la plus optimisée en terme de performance et consommation électrique (par exemple, la première configuration du tableau 3.1 dans notre cas).

Configuration name	CPU Frequency	L2 cache	Interconnect Frequency	Power consumption
Config1	300 MHz	Enabled	166 or 83 MHz	954 mJ
Config2	450 MHz	Disabled	166 MHz	1012 mJ
Config3	700 MHz	Disabled	83 MHz	1439 mJ

TABLE 3.1: Différentes possible configurations for 10 FPS QoS.

D'autres expérimentations ont été effectuées afin de mesurer l'impact des paramètres du cache de niveau 1 sur le nombre de cache miss. Pour cela, l'outil de profiling Valgrind [105] a été utilisé pour permettre d'extraire le nombre de cache miss de l'exécution de l'application vidéo décodeur H.264. Cet outil peut être configuré avec différentes tailles de cache, différentes tailles de ligne de cache, et différents degrés d'associativité. Les

Event Type	Incl.	Self	Short
Instruction Fetch	3 331 730 810	82	Ir
Data Read Access	1 218 163 524	30	Dr
Data Write Access	306 899 059	16	Dw
L1 Instr. Fetch Miss	1 410 874	1	I1mr
L1 Data Read Miss	395 226	1	D1mr
L1 Data Write Miss	291 274	2	D1mw
LL Instr. Fetch Miss	810	1	ILmr
LL Data Read Miss	612	0	DLmr
LL Data Write Miss	4 694	2	DLmw
Conditional Branch	170 358 767	14	Bc
Mispredicted Cond. Branch	8 835 577	2	Bcm
Indirect Branch	641 216	2	Bi
Mispredicted Ind. Branch	107 973	1	Bim

FIGURE 3.3: Paramètres fournis par Valgrind pour chaque tâche.

résultats fournis par cet outil nous permettent d'obtenir différents paramètres pour les tâches s'exécutant sur le processeur comme le nombre d'instructions, le nombre d'accès mémoire ou le nombre de cache-miss. On peut voir sur la figure 3.3 les différents paramètres fournis par Valgrind ici associés à l'exécution d'une application de décodage vidéo H.264.

La figure 3.4 montre le nombre de cache miss pour différentes configurations de cache. On peut observer que seule la configuration avec un degré d'associativité a un comportement sensiblement différent des autres configurations. Dans les autres cas, on remarque que seule la taille du cache a vraiment un impact important sur le nombre de cache-miss. En étudiant les processeurs présents sur le marché, il est intéressant de noter que généralement seule la taille varie de l'un à l'autre, alors que le nombre de way reste le même. C'est pour cette raison que nous avons décidé de ne pas modéliser le nombre de way. Nos modèles sont ainsi plus simples tout en étant largement représentatifs des configurations de caches des processeurs que l'on peut trouver sur le marché.

Nous avons ensuite étudié l'impact du prédicteur de branchements sur les performances d'une application vidéo décodeur H.264. Pour cela, nous avons exécuté le décodeur sur la plate-forme OMAP3530 avec une fréquence fixe de 600MHz en activant ou désactivant le prédicteur de branchements. Les résultats obtenus montrent que lorsque l'on utilise le prédicteur de branchements, la performance atteinte par l'application est de 19.3 images décodées par seconde contre 18.4 images décodées par seconde lorsque le prédicteur est désactivé. Il apparaît donc que le prédicteur a une influence non négligeable (4.6% dans cet exemple) sur les performances.

Un autre axe de recherche peut être la différence entre les plates-formes mono-coeur et multi-coeurs. En effet, la charge de traitement à effectuer peut facilement dépasser la capacité d'un processeur unique, et

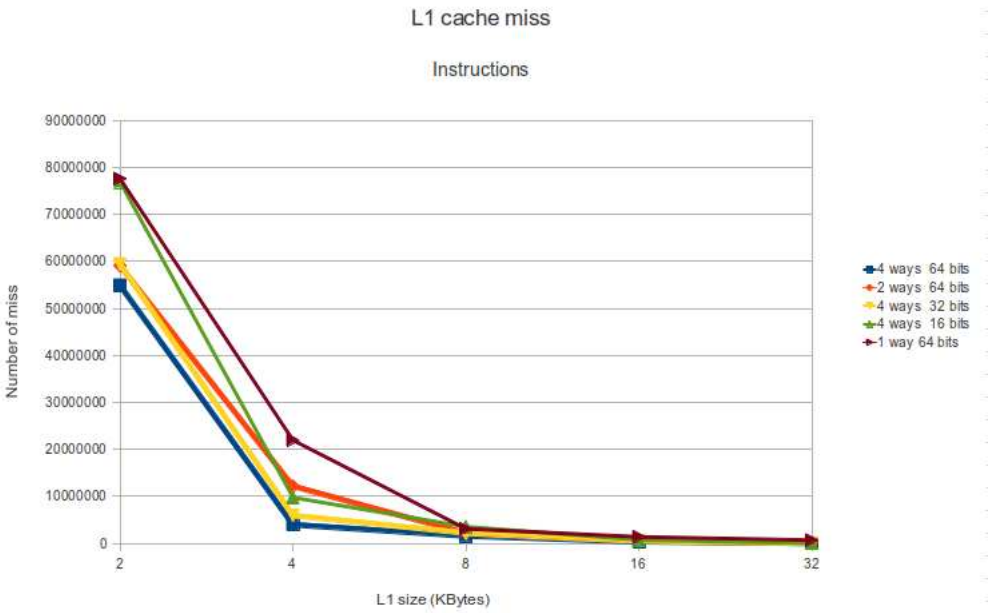

FIGURE 3.4: Impact de la configuration du cache sur le nombre de cache miss pour un cache L1.

un multiprocesseur peut être nécessaire pour réaliser une exécution efficace, i.e. respectant les contraintes temps-réel. De plus, les architectures multiprocesseurs sont plus rentables que celles mono-processeur (à puissance équivalente) parce que le coût d'un système de k-processeurs est significativement moins cher que celui d'un processeur qui est k fois plus rapide (si un processeur de cette vitesse est en effet disponible).
De même, la consommation électrique d'un système de k-processeur est moins importante que celle d'un processeur qui est k fois plus rapide. Par exemple, nous avons exécuté un décodeur vidéo H.264 sur une plate-forme double-coeur en utilisant soit un, soit deux coeurs. Nous avons dans un premier temps choisi de garder un même niveau de performance de l'application sur les deux configurations matérielles.

- Pour la plate-forme n'utilisant qu'**un seul processeur** à 1000MHz, nous avons observé une performance de 13.1ms pour décoder une image. La consommation associée pour décoder une image est de 8.48mJ.
- Pour la plate-forme utilisant **deux processeurs** à 600MHz, nous avons observé une performance de 12.9ms pour décoder une image. La consommation associée pour décoder une image est de 6.16mJ.

Cet exemple montre que posséder deux processeurs au lieu d'un seul permet dans notre cas de réduire la consommation d'énergie de 27% tout en gardant une performance quasiment équivalente.
Dans un second temps, nous avons utilisé le processeur simple à 600MHz et le double coeur à 300MHz afin de garder la même capacité de calcul globale du système.

- Pour la plate-forme n'utilisant qu'**un seul processeur** à 600MHz, nous avons observé une performance de 21.9ms pour décoder une image. La consommation associée pour décoder une image est de 5.93mJ.
- Pour la plate-forme utilisant **deux processeurs** à 300MHz, nous avons observé une performance de 26ms pour décoder une image. La consommation associée pour décoder une image est de 4.42mJ.

De même dans ce cas, passer à un double processeur a permis un gain en consommation de 34% mais a aussi causé une perte de performance de 15.7% (car les applications ne sont jamais exécutées 100% en parallèle). Les observations ci-dessus soulignent clairement l'importance croissante des multi-processeurs dans les systèmes temps réel et le besoin de prendre en compte le nombre de processeurs présents dans l'architecture.

Grâce à ces différentes expérimentations, nous avons mis en évidence plusieurs paramètres importants à prendre en compte dans le processus d'estimation. Il s'agit, entre autres, de la fréquence des processeurs, la taille des mémoires caches mais aussi la taille du pipeline ainsi que le nombre de mauvaises prédictions

de branchements. Si les deux premiers paramètres paraissaient évidents, les deux derniers pouvaient sembler plus négligeables.

Comme on a pu le voir dans ce chapitre, différents paramètres influent sur la performance et la consommation d'un système. On peut classer ces paramètres en deux catégories, les paramètres matériels et les paramètres logiciels. Le tableau 3.2 représente les différents paramètres caractéristiques retenus :

Paramètres matériels	Paramètres logiciels
Fréquence processeur Taille des caches	Nombre d'instructions Nombre d'accès mémoire en lecture Nombre d'accès mémoire en écriture Nombre d'erreurs de prédiction de branchement Nombre de cache-miss d'instruction en fonction de la taille du cache Nombre de cache-miss de données en fonction de la taille du cache Le taux d'utilisation du pipeline

TABLE 3.2: Les différents paramètres importants retenus dans notre approche.

En plus de ces paramètres qu'il nous a été possible d'expérimenter sur plate-forme réelle, il semble nécessaire d'ajouter certains paramètres dont l'impact sur les performances où la consommation n'est pas mesurable à l'aide d'une plateforme matérielle. Citons par exemple :

MIPS/MHz des processeur qui correspond à la capacité de traitement (nombre d'instructions par seconde) du processeur

Taille de pipeline des processeurs qui est utile lorsqu'on l'associe aux erreurs de prédiction de branchement afin de connaître le nombre de cycles de pénalité nécessaire

Ces deux paramètres sont de plus présent dans les "datasheets" constructeurs.

Le chapitre suivant présente la méthodologie utilisée ainsi que l'outil FORECAST développé.

Chapitre 4

Description précise de la méthodologie et de l'outil FORECAST

Ce chapitre propose une description précise de chaque partie de la méthodologie ainsi que les outils associés. Dans un premier temps, nous évoquerons la méthodologie générale ainsi que les outils externes utilisés. Ensuite, nous développerons dans l'ordre le formalisme d'entrée de FORECAST (coté logiciel et matériel), les estimateurs utilisés, l'outil et le langage Waveperf puis ce que fournit FORECAST en terme de résultats. Enfin, nous aborderons l'explorateur développé dans le cadre de cette thèse permettant d'aider les architectes système.

4.1 Description générale de l'approche et ses outils associés

La méthodologie (et l'outil) développée repose d'une part sur la modélisation de la partie logicielle, d'autre part, sur celle de la partie matérielle. Une étape supplémentaire est effectuée pour lier les deux parties ensemble. Ceci permet de ne pas faire une modélisation dépendante du matériel et de pouvoir facilement faire de l'exploration d'architectures. De plus, les paramètres choisis, que ce soit pour la partie applicative, ou pour la partie matérielle, sont simples à estimer et/ou à déterminer à partir des "datasheets" constructeur. Dans une première partie, nous proposons une description globale de la méthodologie d'estimation développée, puis nous présenterons la méthode de profiling utilisée pour déterminer les informations dynamiques de l'application ainsi que l'une des exploitations possibles.

4.1.1 Le flot de l'outil d'estimation et d'exploration

La méthodologie développée se décompose en 5 grandes parties :

1. La modélisation de la plate-forme matérielle et de l'application ainsi que l'étape de mapping des tâches logicielles sur les différents processeurs
2. L'estimation de la performance statique de chaque tâche
3. La génération de code et l'exécution de la simulation
4. L'utilisation des traces dynamiques pour l'estimation des performances globales et de la consommation d'énergie

5. L'exploration des architectures

La figure 4.1 montre la manière dont s'exécutent les différentes parties. En entrée de FORECAST, il faut

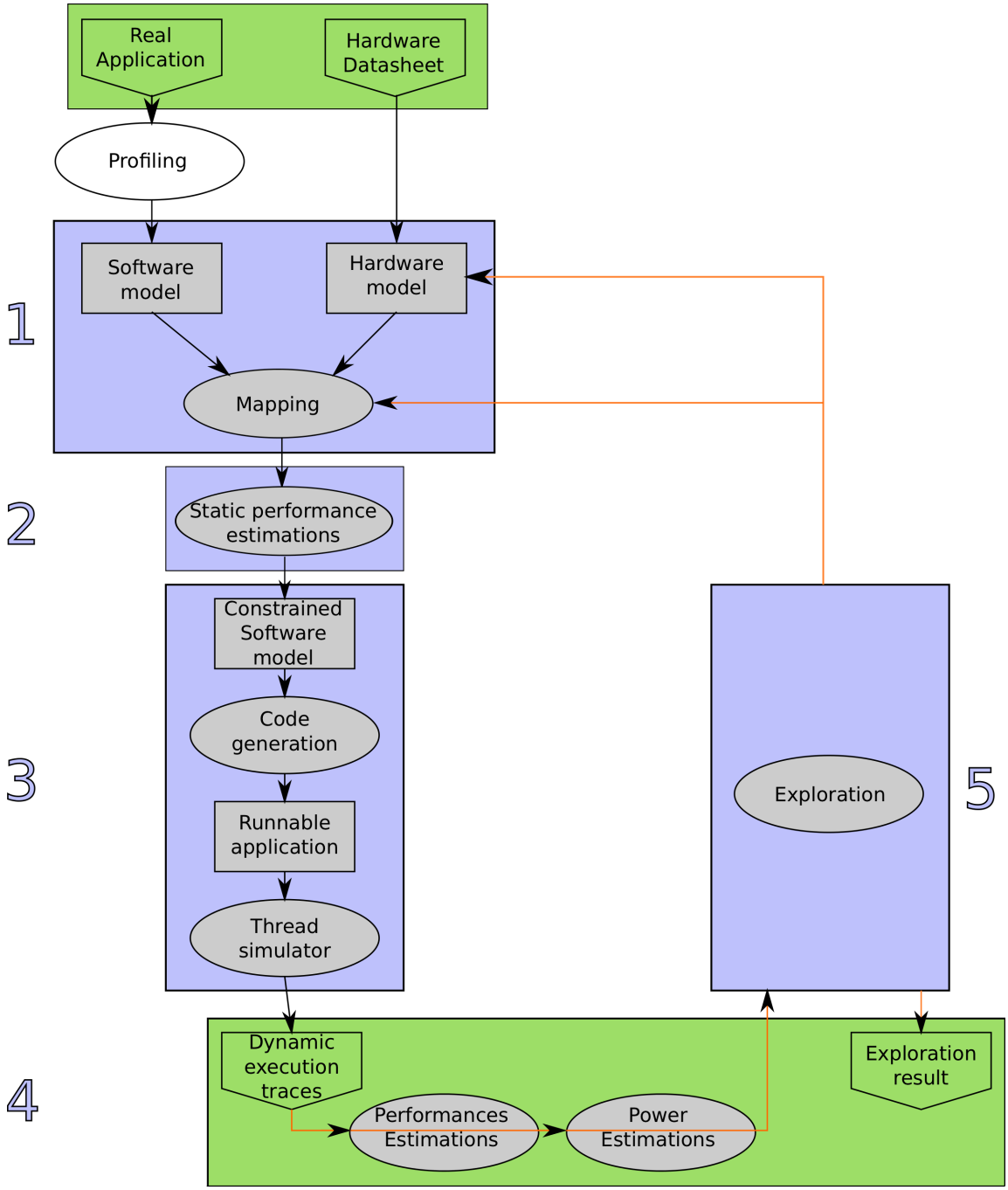

FIGURE 4.1: Description globale de l'outil FORECAST.

posséder :

pour la partie matérielle : une datasheet de composants matériels à modéliser

pour la partie logicielle : une description réelle ou basée sur un modèle théorique de l'application

Dans le cas d'une application réelle, une étape supplémentaire de profiling est nécessaire. Cette étape sera décrite dans la section suivante étant donné que nous utilisons quasi-systématiquement une application réelle (une application ou benchmark déjà existant) en entrée. Si l'application n'existe pas encore, on peut utiliser certaines méthodes d'estimation du nombre d'accès mémoire suivant la complexité de l'application, comme par exemple dans [104].
Notre approche permet alors, à partir de ces deux modèles, de créer un modèle de l'application contrainte par le matériel considéré pour l'estimation de performance ou de consommation. C'est ce que l'on appelle, l'étape de mapping. C'est à ce moment qu'interviennent les estimateurs de performance permettant de calculer statiquement (i.e. les effets dynamiques telle que la préemption ne sont pas pris en compte) le temps d'exécution de chaque tâche suivant l'unité de calcul choisie. Ensuite, le modèle est transformé en code C++ exécutable sur n'importe quel ordinateur utilisant la norme POSIX (toute plate-forme sous Linux).
Lorsque ce code généré est lancé sur une machine, l'ordonnancement, les priorités et les préemptions liées aux différentes tâches de l'application sont également considérées. De plus, une trace d'exécution est générée. Cette trace contient le début et la fin de chaque tâche permettant ainsi d'analyser le comportement fonctionnel et dynamique de l'application comme par exemple la charge du ou des processeurs, le respect de la périodicité ou de l'ordonnancement des tâches.
Comme on peut le voir sur la figure 4.1, une analyse des performances et de la consommation est faite à posteriori. Le premier processus permet d'analyser les différentes charges CPU, le second (appelé "Power Estimations" sur la figure 4.1) analyse la consommation associée à l'exécution et retourne la consommation de chaque élément, mémoire et processeur.
Enfin, FORECAST permet également d'explorer différentes architectures en modifiant les paramètres/composants matériels (changement des fréquences, de la taille des caches, du processeur, ...) ainsi que la répartition des tâches (affinité des tâches) sur les différents processeurs. L'explorateur utilise les valeurs calculées de charge CPU ainsi que la consommation estimée.

Afin de nous aider à créer les modèles logiciels, une étape de profiling est nécessaire lors de l'utilisation d'une application réelle. Cette étape est décrite dans la section suivante.

4.1.2 Le profiling de l'application

Lorsqu'une application réelle doit être modélisée, il est nécessaire d'effectuer une étape de profiling afin d'obtenir des informations dynamiques pour cette application. Étant donné que nous souhaitons avoir des informations précises sur les caractéristiques et non pas seulement un "workload" comme dans beaucoup d'approches haut-niveau, nous avons décidé d'utiliser l'outil Valgrind [105].

A partir de la, il est possible de faire différents choix de modélisation. Tout d'abord, si nous ne disposons pas d'une plate-forme réelle du même type que la plate-forme que l'on cherche à modéliser, il est possible de faire du profiling sur un PC. Nous pouvons alors obtenir le nombre d'instructions et d'accès mémoire (load/store) pour l'architecture X86. Comme le montre la table 4.1 dans le cas de l'application H.264 par exemple, la différence pour le nombre d'instructions mais surtout pour le nombre d'accès mémoire peut-être non négligeable.

Platform name	x86	ARM	difference (%)
With filter			
total_nb_insn	3712418033	3302398898	11
nb_w	1279783228	958862304	25
nb_r	670673474	526536751	21.5
Without filter			(%)
total_nb_insn	1529944972	1524018057	0.4
nb_w	508337779	381628784	25
nb_r	247282385	208018352	15.9

TABLE 4.1: Comparaison de quelques paramètres de profiling obtenus par Valgrind sur différentes architectures pour l'application décodage vidéo H.264.

Des estimations de performances ont été effectuées à partir du profiling de deux différentes architectures (ARM et x86). Les estimations basées sur un profiling effectué sur une plateforme différente de la plateforme réelle montrent une erreur d'estimation de 5 à 10% supplémentaire. Par exemple, une estimation de la performance de l'application décodeur de vidéo H.264 donne en moyenne 12% d'erreur par rapport à la plate-forme réelle dans le cas du profiling X86, et 6% dans le cas du profiling ARM (cf. Table 4.2).

Platform name	x86 (error)	ARM (error)
600 MHz	12.6 %	6.07 %
500 MHz	16.3 %	6.35 %
250 MHz	12.4 %	6.22 %
125 MHz	10.0 %	9.44 %

TABLE 4.2: Comparaison de l'erreur d'estimation de la performance suivant l'architecture utilisée pour le profiling.

Il apparaît donc nécessaire, dans la mesure du possible, d'effectuer l'étape de profiling avec une architecture la plus proche possible de l'architecture que l'on souhaite estimer ou explorer. Comme le montre nos tests, les résultats en seront d'autant plus précis.

Des plates-formes réelles comme la BeagleBoard [11] (OMAP3530) ou la PandaBoard [116] (OMAP44xx) sont facilement disponibles et ceci pour un montant tout à fait raisonnable (environ 200$). Par conséquent, il est relativement aisé de faire du profiling sur ce type d'architecture embarquée (de type ARM dans ce cas). Une fois les paramètres clés obtenus pour un processeur, une extrapolation est effectuée afin de généraliser

les résultats de profiling aux processeurs ayant une même classe d'architecture. Par exemple, nous faisons le profiling sur un ARM Cortex-A8, et extrapolons les résultats obtenus pour tous les autres processeurs ARM.

Dans notre approche, l'outil de Valgrind permet aussi de profiler le nombre de cache-miss de l'application en spécifiant une taille de cache pour le cache L1-instructions, le cache L1-données et le L2.

```
valgrind --tool=callgrind --cacheuse=yes --I1=32768,4,64 --D1=32768,4,64 --LL=262144,8,128 ./
    nbench -cCOM.DAT
```

Listing 4.1: Exemple de ligne de commande exécutant Valgrind sur l'application nbench

La procédure que nous avons choisie d'utiliser consiste à faire tourner trois fois le profiler avec trois tailles de cache différentes. Par exemple, pour le cache de niveau 1 : 8KByte, 16KByte et 32KByte. Le listing 4.1 présente un exemple d'exécution de Valgrind avec une taille de cache L1 de 32KByte (ainsi qu'une associativié de 4 et une largeur de 64 bits) et une taille de cache L2 de 256Kbyte (ainsi qu'une associativié de 8 et une largeur de 128 bits).
En examinant le nombre de cache-miss de chaque tâches, nous avons observé un comportement différent du nombre de cache-miss pour le cache d'instructions ou le cache de données.
En effet, sur la base de plusieurs expérimentations, il s'est avéré que le nombre de cache-miss dans le cache d'instruction évolue telle une puissance ($a \cdot x^b$, où a et b sont des constantes, et x la taille de la mémoire cache cf : Fig.4.2) tandis que le nombre de cache-miss dans le cache de données évolue de manière exponentielle ($a \cdot e^{b \cdot x}$ cf : Fig.4.3).
Il est donc possible d'utiliser une technique d'interpolation pour calculer les valeurs des paramètres a et b, ou de façon plus pragmatique un logiciel de résolution d'équations tel que GnuPlot ou Microsoft Excel en lui précisant le type d'équation souhaitée pour trouver notre fonction représentant le nombre de cache-miss instruction et données. Afin de n'utiliser aucun logiciel propriétaire ou payant, nous avons décidé d'utiliser Gnuplot. Gnuplot nous permet à la fois de calculer la courbe de tendance, mais aussi de les tracer pour vérifier visuellement que l'estimation est proche de ce que l'on souhaite.

```
gnuplot> f(x)=a*x**b
gnuplot> fit f(x) 'il1_subsampled.dat' via a,b
gnuplot> plot "il1.dat" title 'Real data', a*x**b title 'fit curve'
```

Listing 4.2: Ligne de commande Gnuplot permettant de trouver la fonction d'approximation du nombre de cache-miss

Le listing 4.2 montre les instructions Gnuplot permettant d'estimer la courbe $f(x) = a \cdot x^b$ puis de la tracer sur un graphique avec les mesures réelles pour pouvoir les comparer.

Prenons par exemple le nombre de cache miss d'instruction pour le cache de niveau 1 pour un décodeur vidéo H.264. Nous avons dans un premier temps fait la mesure du nombre de cache miss pour plusieurs tailles de cache. Puis, nous avons cherché à extrapoler les résultats sur la base d'un sous-ensemble des points. Les points rouges sur la figure 4.2 montrent les mesures réelles de cache miss. La courbe verte montre la fonction d'approximation du nombre de cache miss pour n'importe quelle taille de cache. Cette courbe a été obtenue à l'aide de seulement 3 points de mesure (8KByte, 16KByte et 32KByte).

De la même manière, la figure 4.3 montre l'évaluation de la fonction de prédiction du nombre de cache miss pour les données (donc cette fois en $a \cdot e^{b \cdot x}$) calculée à partir de 3 points. Nous avons choisi de n'utiliser que trois valeurs réelles pour, d'une part accélérer la construction du modèle, et d'autre part car les essais effectués avec plus de points ne permettent pas d'obtenir de meilleures précisions.

L'outil Valgrind est donc capable de nous fournir beaucoup d'informations sur une application et cela permet d'une part de renseigner les modèles logiciels mais aussi d'évaluer la courbe de tendance des caches-miss suivant la taille du cache.

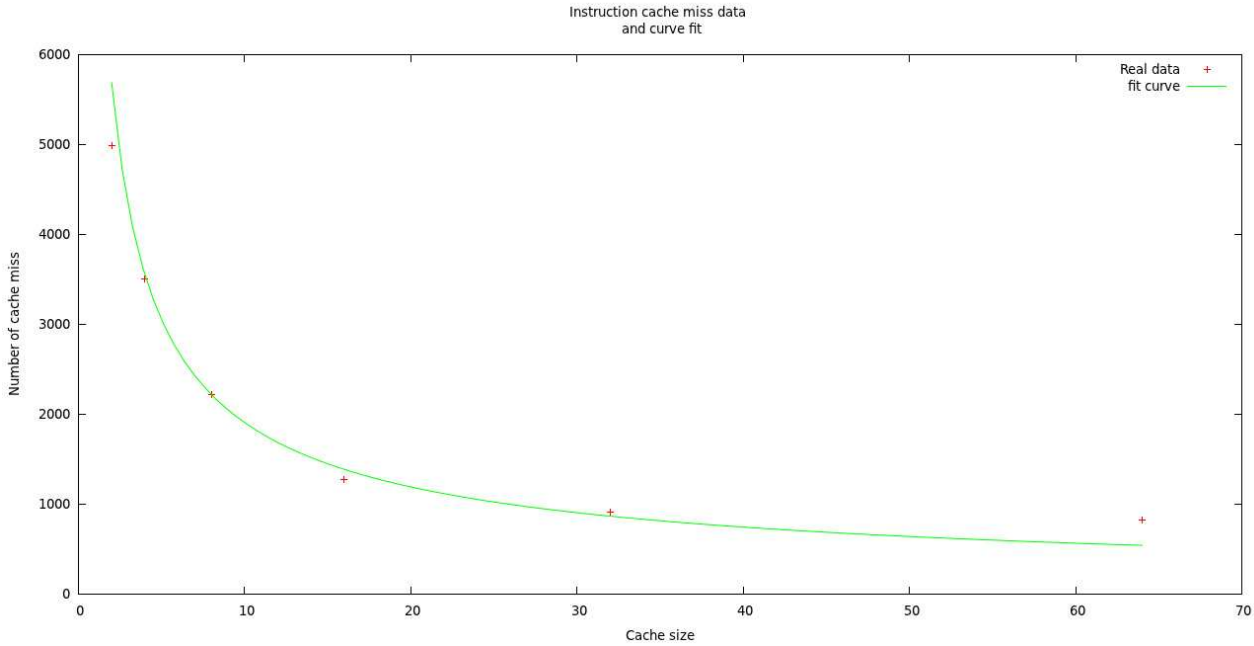

FIGURE 4.2: Icache fit.

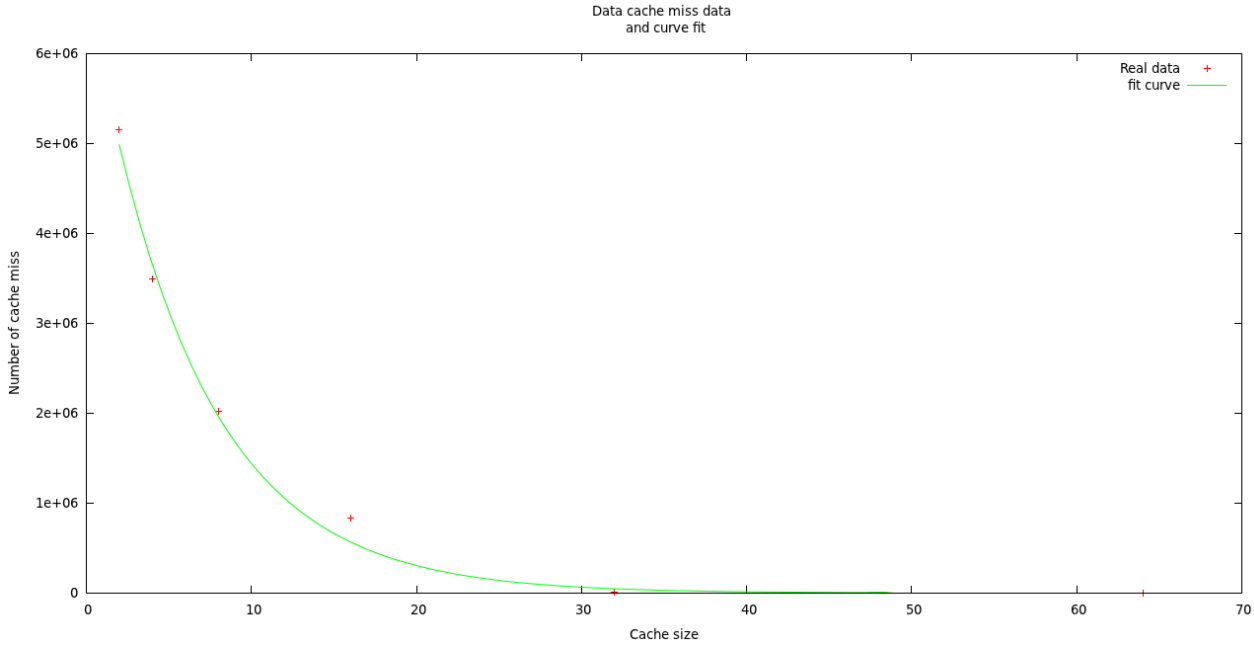

FIGURE 4.3: Data read cache fit.

4.2 Les entrées

Comme le montre la figure 4.4, notre méthodologie nécessite deux modèles distincts en entrée : un modèle représentant l'application, l'autre décrivant la plate-forme matérielle. La simplicité du langage de description utilisé pour ces deux modèles permet de créer de nouveaux modèles/comportements facilement et rapidement. Une étape de mapping est alors nécessaire afin d'assigner les tâches aux différents processeurs.

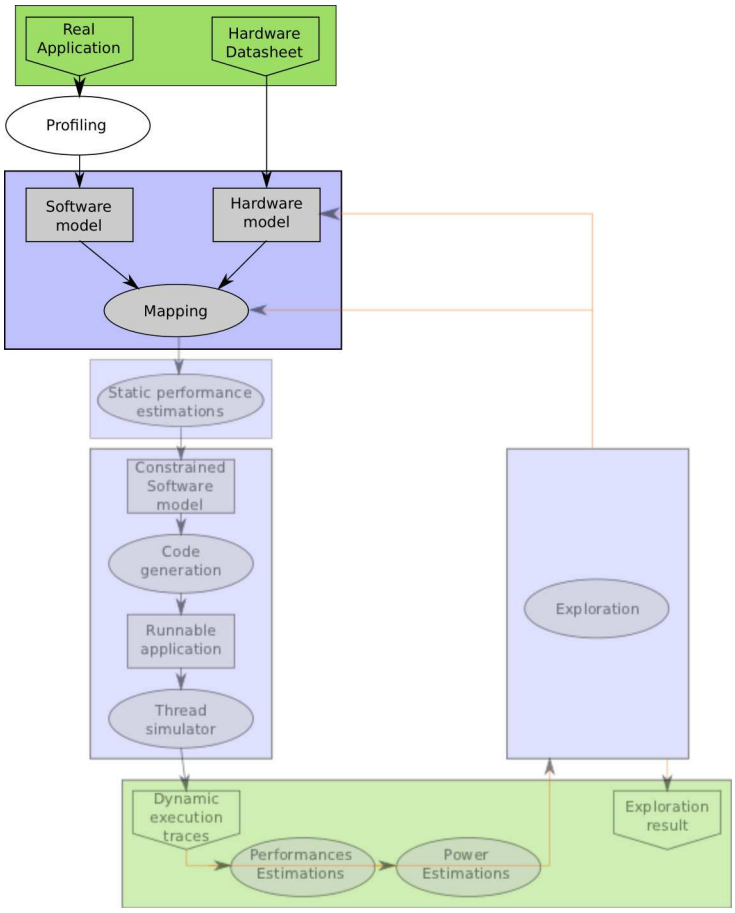

FIGURE 4.4: Graphe représentant les entrées nécessaires à la méthodologie.

4.2.1 Modélisation de l'application logicielle

Concernant le modèle de l'application, nous avons choisi un modèle de haut niveau d'abstraction basé sur la description des processus et des tâches ainsi que leurs interdépendances. Le langage utilisé permet donc de facilement créer des composants logiciels (processus) avec des connexions entrantes (tâches) ainsi que les sorties. Une étape de connexion entre ces composants est ensuite nécessaire afin de décrire les dépendances et le comportement correct du modèle.
La méthodologie étant basée sur l'outil *Waveperf*, le formalisme du modèle logiciel est une extension du formalisme décrit dans la section 4.4.1.

En effet, *Waveperf* n'a pas été créer à l'origine pour d'effectuer de l'estimation de performances, il était

seulement nécessaire de renseigner un temps d'exécution pour chaque tâche. Ici, nous avons enrichi le langage en supprimant ce temps fixe et en le remplaçant par différents paramètres permettant d'estimer un temps d'exécution.
Tout d'abord, nous avons rajouté le mot clé TBC qui signifie "To Be Calculated" et qui permet à l'outil qui va parcourir le fichier texte de connaître l'emplacement de la valeur de temps d'exécution estimée. Ensuite, une série de valeurs (correspondant aux différents paramètres logiciels) issues d'un profiling de l'application sont à ajouter au fichier texte :

alpha : Le taux de parallélisation des instructions

i : Le nombre d'instructions à exécuter

r : Le nombre de lectures de la tâche

w : Le nombre d'écritures de la tâche

il1 : Ces deux paramètres servent à estimer le nombre de cache miss du niveau 1 pour les instructions. Ils représentent les deux paramètres a et b de l'équation $a \cdot x^b$ vue dans le chapitre précédent

rl1 : Ces deux paramètres servent à estimer le nombre de cache miss du niveau 1 pour les lectures. Ils représentent les deux paramètres a et b de l'équation $a \cdot e^{b \cdot x}$ vue dans le chapitre précédent

wl1 : Ces deux paramètres servent à estimer le nombre de cache miss du niveau 1 pour les écritures. Ils représentent les deux paramètres a et b de l'équation $a \cdot e^{b \cdot x}$ vue dans le chapitre précédent

il2 : Ces deux paramètres servent à estimer le nombre de cache miss du niveau 2 pour les instructions. Ils représentent les deux paramètres a et b de l'équation $a \cdot x^b$ vue dans le chapitre précédent

rl2 : Ces deux paramètres servent à estimer le nombre de cache miss du niveau 2 pour les lectures. Ils représentent les deux paramètres a et b de l'équation $a \cdot e^{b \cdot x}$ vue dans le chapitre précédent

wl2 : Ces deux paramètres servent à estimer le nombre de cache miss du niveau 2 pour les écritures. Ils représentent les deux paramètres a et b de l'équation $a \cdot e^{b \cdot x}$ vue dans le chapitre précédent

bm : Le nombre de mauvaises prédictions de branchement fait par la tâche

Comme il sera expliqué dans la section suivante, tous ces paramètres permettent de calculer la performance d'une tâche.

```
characteristics( Timing_in_ms ) slice_timing_characs of slice_behaviour
{
  input.run
  {
    (1) { 12.42 0 }
  }
};

characteristics( Architecture_parameters ) slice_timing_characs of slice_behaviour
{
  input.run
  {
    (1) { TBC 0 } [ alpha i r w il1 il1 rl1 rl1 wl1 wl1 il2 il2 rl2 rl2 wl2 wl2 bm]
  }
};
```

Listing 4.3: Exemple illustrant les paramètres à fournir à Waveperf (haut) et les nouveaux paramètres à fournir (bas) au modèle logiciel.

Le listing 4.3 présente dans un premier temps le paramètre temporel qu'il était necessaire de fournir à *Waveperf* et dans un second temps les informations dynamiques nécessaires à la création du modèle d'une tâche pour l'estimation de performance.

Le langage initial a donc été enrichi avec des paramètres permettant de caractériser le modèle applicatif avec précision et afin d'être capable d'explorer différentes architectures. Le formalisme global reste le même

que le langage *Waveperf* (expliqué dans la section 4.4.1) ce qui permet ainsi aux ingénieurs utilisant déjà ce langage de bénéficier de leur expérience.

4.2.2 Modélisation de l'architecture matérielle d'une plate-forme

Après la phase de modélisation de la partie applicative, il est nécessaire de faire une description de l'architecture matérielle de la plate-forme considérée. Afin d'être uniforme avec le langage de modélisation utilisé pour *Waveperf*, un langage similaire à été développé pour décrire des architectures matérielles. Cette extension à *Waveperf* utilise le même type de découpage en fichier (la description des processeurs d'un coté, le lien entre tous les processeurs de l'autre) et une syntaxe similaire pour les fichiers de composition.
Il est tout d'abord nécessaire de modéliser les composants de calcul (GPP, DSP) à l'aide de paramètres architecturaux disponibles dans les datasheets constructeurs.

```
start_frequency: 100
end_frequency: 800
dmips: 2
pipeline: 13
```

Listing 4.4: Description d'un composant processeur ARM CortexA8

Le listing 4.4 montre par exemple quelques paramètres architecturaux caractérisant le composant ARM Cortex-A8. On peut y retrouver quelques paramètres architecturaux comme la fréquence minimale et maximale, le nombre d'instructions par secondes par MHz et la profondeur du pipeline. Les GPP peuvent ainsi être facilement et rapidement modélisés uniquement sur la base des datasheets constructeurs.

FORECAST étant grandement "customisable", des paramètres spécifiques aux DSP pourraient aisément être implémentés et modélisés de la même manière afin d'introduire de l'hétérogénéité dans l'outil.

De la même manière qu'il est nécessaire de connecter les tâches logicielles entre elles, il est également nécessaire de relier les unités de calcul (GPP, DSP) avec leurs mémoires correspondantes (caches, mémoire principale). Un fichier peut ainsi être créé pour décrire les différentes connexions entre les blocs mémoire et les processeurs.

```
component  PU     CPU0  inputfile/cortexA9.txt 150
component  PU     CPU1  inputfile/cortexA9.txt 150
component  PU     CPU2  inputfile/cortexA9.txt 150
component  PU     CPU3  inputfile/cortexA9.txt 150
component  CACHE  L1_0  32 32 4
component  CACHE  L1_1  32 32 4
component  CACHE  L2_0  1024 32 8
component  CACHE  L1_2  32 32 4
component  CACHE  L1_3  32 32 4

connection  CPU1  ->  L1_0
connection  CPU0  ->  L1_2
connection  L1_0  ->  L2_0
connection  CPU2  ->  L1_1
connection  CPU3  ->  L1_3
connection  L1_1  ->  L2_0
connection  L1_2  ->  L2_0
connection  L1_3  ->  L2_0
```

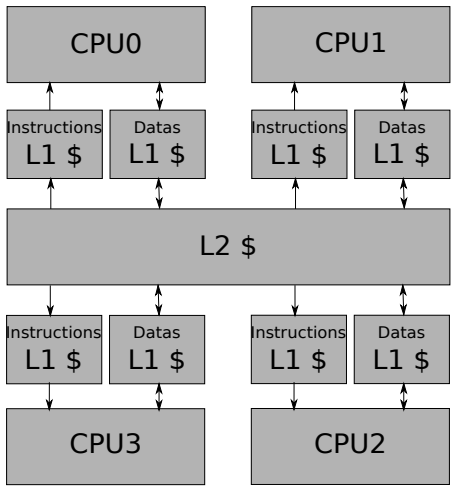

FIGURE 4.5: Description de l'architecture matérielle d'un i.MX6.

La figure 4.5 et le listing associé montrent le fichier permettant de décrire une architecture quadri-coeurs du type Freescale IMX.6.
Le mot clé "component" permet d'instancier un composant processeur ou mémoire. Le mot clé PU (Processing Unit) est utilisé pour instancier un processeur alors que le mot clé CACHE permet lui d'instancier une mémoire cache. Il est aussi possible de donner un nom et de passer des paramètres à chaque instance.

Pour un processeur, les deux paramètres sont le fichier du composant et la fréquence souhaitée pour ce processeur.
Le cache quant à lui nécessite trois paramètres : sa taille, sa largeur de ligne et son degré d'associativité. On remarque qu'il n'y a pas de distinction de cache de données ou d'instructions dans notre fichier de description. En effet, les caches de données et d'instructions possèdent toujours la même taille ce qui ne nécessite pas de décrire deux fois la même chose.

Le mot clé "connection" permet de connecter deux composants à l'aide du symbole "− >". Les connexion s'effectuent toujours du processeur vers la mémoire principale. La connexion s'effectue donc depuis un processeur vers un cache L1, ou d'un cache L1 vers un cache L2.

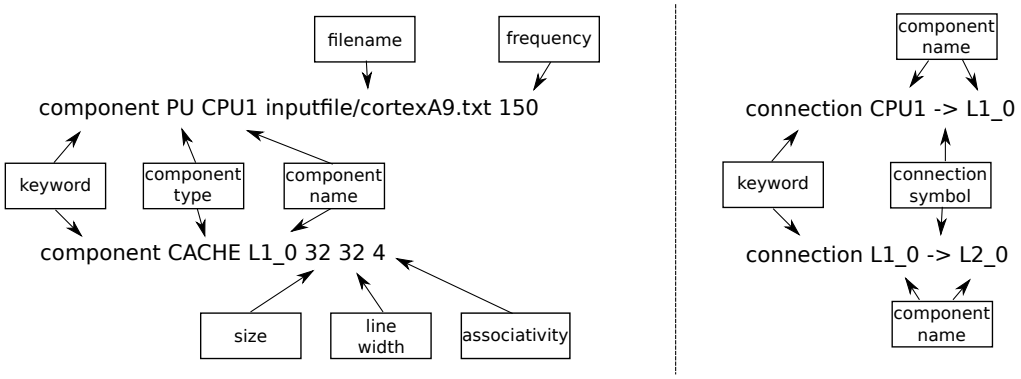

FIGURE 4.6: Syntaxe de la création de composants (à gauche) et des connexions (à droite).

La figure 4.6 reprend les différentes parties du fichier de composition de la plate-forme matérielle en expliquant précisément les valeurs de chaque paramètre. Cette approche par composant nous permet de créer des bibliothèques d'applications par exemple, mais aussi des bibliothèques de plates-formes existantes.

Une dernière étape est nécessaire lors de la création des modèles. Cette étape consiste à mapper le modèle logiciel sur le modèle matériel.

4.2.3 La phase de mapping

Une fois que les tâches (composants logiciel) sont complètement caractérisées (voir section 5.4), il est alors nécessaire de modéliser les connexions entre ces tâches. Ici, seul le formalisme permettant d'allouer une tâche à un processeur est abordé. En effet, il y a différentes possibilités pour l'allocation d'une tâche (due à l'exploration d'architecture). Le listing 4.5 et la figure 4.7 montrent les trois possibilités d'allocation d'une

```
configuration tx_data.configure_affinity( 0 );
configuration tx_data.configure_affinity( ? );
configuration tx_data.configure_affinity( A );
```

Listing 4.5: Un exemple des différentes possibilités d'allocation de tâche.

tâche :

Affinité fixe La tâche est assignée de manière fixe à un processeur. Cette syntaxe doit être obligatoirement utilisée pour effectuer uniquement une estimation de performance ou de consommation (pas d'exploration automatique).
Le chiffre mis entre parenthèses représente l'identifiant du processeur sur lequel la tâche est allouée.

Affinité variable La tâche peut être assignée à n'importe quel processeur. Cette syntaxe autorise la migration de tâches d'un processeur à l'autre dans le cas de plates-formes multiprocesseurs. Cette syntaxe est donc particulièrement utilisée lors de la phase d'exploration d'architecture.
Le point d'interrogation mis entre parenthèses représente l'ensemble des processeurs disponibles sur la plate-forme.

Affinité variable par groupe De la même manière que l'affinité variable, la tâche peut être assignée à n'importe quel processeur par l'explorateur. L'intérêt de cette syntaxe réside dans le fait que plusieurs tâches peuvent être regroupées. En d'autres termes, si l'une d'elles est assignée à un processeur, toutes les autres tâches du même groupe seront également allouées sur ce même processeur.
Une lettre en majuscule mise entre parenthèses représente l'identifiant du groupe de tâches. Pour faire partie d'un même groupe, les tâches doivent avoir la même lettre.

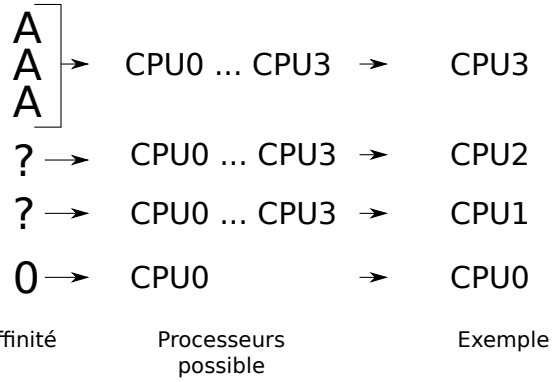

FIGURE 4.7: Définition des différentes affinités processeurs possibles.

Suivant ce que l'on souhaite faire (estimation seule ou exploration), il est alors possible de configurer différemment les tâches en les assignant ou non à des processeurs.

Une fois les modèles de l'application et du matériel décrits, il faut procéder aux estimations. C'est ce dont fait l'objet la prochaine section.

4.3 Les estimateurs

4.3.1 Performance

Comme on peut le voir sur la figure 4.8, il s'agit ici, à partir de modèles génériques d'une application et d'un modèle du matériel, de créer un modèle de l'application contraint par le matériel. FORECAST va estimer la performance de chaque tâche (i.e. chaque entrée d'un composant) pour la plate-forme matérielle cible considérée.

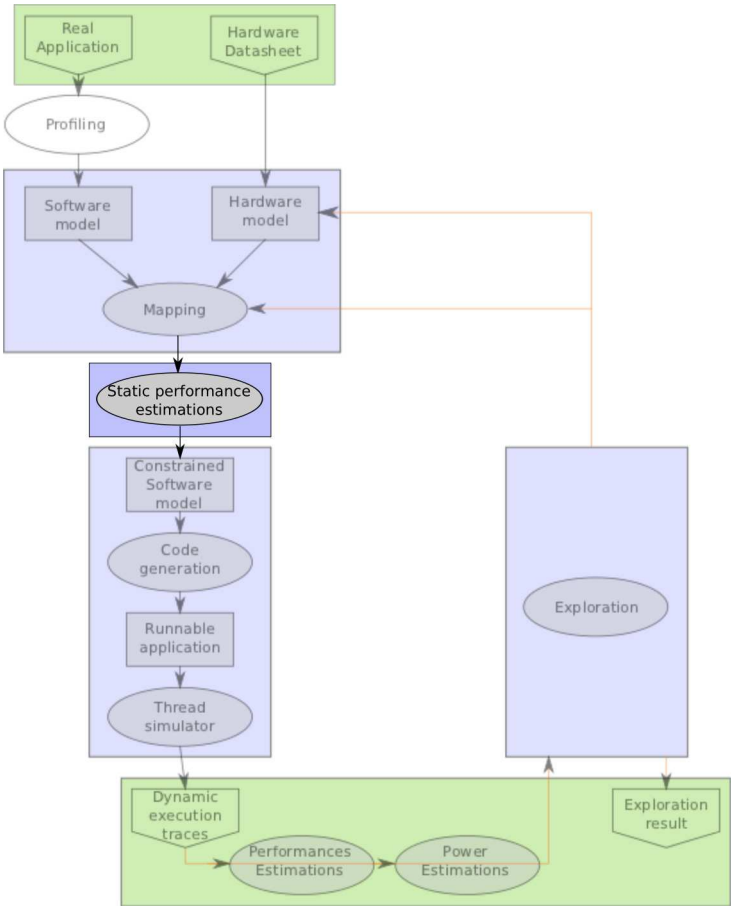

FIGURE 4.8: Transformation des modèles logiciels et matériels vers un modèle *Waveperf* "timé".

Le listing 4.6 montre un exemple de transformation entre les deux modèles. TBC est remplacé par le temps total d'exécution de la tâche calculé (Total_elapsed_time). De plus, les premiers et derniers paramètres (respectivement le taux d'utilisation du pipeline et le nombre de mauvaises prédictions de branchements) sont retirés car ils ne sont nécessaires que pour le calcul du temps d'exécution des tâches.

L'outil d'estimation va donc calculer le temps total d'exécution d'une tâche comme étant la somme du temps passé au niveau du processeur ($CPU\_elapsed\_time$) et du temps passé au niveau des mémoires ($MEM\_elapsed\_time$) :

$$Total\_elapsed\_time = CPU\_elapsed\_time + MEM\_elapsed\_time \tag{4.1}$$

Le paramètre *Total_elapsed_time* représente une estimation du temps total d'exécution de la tâche en mil-

```
    (1) { TBC 0 } [ 90 15039303 3235056 1623677 10500 6470 3247 270 1617 811 66816 ]
vers
    (1) { 26.0 0 } [ 15039303 3235056 1623677 10500 6470 3247 270 1617 811 ]
```

Listing 4.6: Transformation de modèles : paramètres d'architecture étendus / paramètres d'architecture waveperf.

lisecondes. Cette métrique sera souvent comparée au temps d'exécution du système réel dans la section validation. Le *CPU_elapsed_time* est défini de la manière suivante :

$$CPU\_elapsed\_time = \frac{total\_nb\_insn}{perf} + InstrCacheMissPen + MissPredPen \tag{4.2}$$

Ce paramètre représente une estimation du temps d'exécution en millisecondes requis pour exécuter toutes les instructions présentes dans le thread.

- Le paramètre *total_nb_insn* est égal au nombre total d'instructions exécutées pour cette tâche.
- Le paramètre *perf* représente le nombre d'instructions par secondes que le processeur est capable de traiter. Ce paramètre dépend de la fréquence et du nombre d'instructions par seconde par MégaHertz.
- Il faut aussi prendre en compte le temps passé à vider le pipeline lors d'une erreur de prédiction de branche (*MissPredPen*). En effet, lors d'une boucle conditionnelle par exemple, le processeur va précharger les instructions d'une branche ou d'une autre dans son pipeline. Mais si la branche préchargée est la mauvaise, il faut alors vider le pipeline, ce qui prend du temps. Ce paramètre dépend donc du nombre de mauvaises prédictions de branchements (nb_miss) ainsi que de la profondeur du pipeline et du temps de cycle de l'horloge (cycle_time).

$$MissPredPen = nb\_miss \cdot pipeline\_depth \cdot cycle\_time \tag{4.3}$$

- *CPU_elapsed_time* inclut aussi le temps nécessaire pour récupérer les instructions lors d'un défaut de cache d'instructions (*InstrCacheMissPen*).

Le paramètre *MEM_elapsed_time* permet de calculer le temps nécessaire pour accéder aux données en lecture et écriture au travers des caches L1 et L2 si nécessaire. Ce paramètre est alors défini de la façon suivante :

$$MEM\_elapsed\_time = [(nb\_r + nb\_w)] \cdot (rw\_ms + \sum_{i=1}^{n} li\_miss\_rate \cdot li\_pen) \tag{4.4}$$

Finalement, le paramètre *li_pen* représente le temps de pénalité (en millisecondes) lors d'un cache miss ($i = 1$ pour le cache de niveau 1, $i = 2$ pour le cache de niveau 2, etc.), et sont définis comme suit :

$$li\_pen = rw\_ms \cdot li\_nbcycle \tag{4.5}$$

Les taux de caches-miss sont évalués comme nous l'avons expliqué dans la section 4.1.2. Suivant la taille des caches choisis dans le modèle matériel, le taux de cache-miss sera différent (calculé à partir des courbes estimées). Le nombre de cycles de pénalité pour chaque niveau mémoire a été défini par rapport à notre connaissance des architectures processeurs, et par vérification sur des plates-formes embarquées. Le temps d'accès à une donnée présente :

- dans le cache de niveau 1 est d'un cycle d'horloge,
- dans le cache de niveau 2 est de dix cycles d'horloge,
- dans la mémoire principale est de cent cycles d'horloge,

Un benchmark a été développé afin de déterminer le nombre de cycle nécessaire pour chaque niveau mémoire. Voici comment se déroule ce benchmark :

- Un tableau assez grand pour contenir toutes les données contenues dans le cache de niveau deux est créé par le benchmark.

– Le benchmark effectue grâce à une boucle interne la lecture de 128 lignes de cache. Ici, on ne lit toujours qu'une seule valeur par ligne pour provoquer uniquement des cache miss dans le cache de niveau un lorsque l'on veut aller lire dans le cache de niveau deux. En effet, si l'on ne faisait pas de saut et qu'on lisait des valeurs consécutives, il y aurait un certain nombre de cache-hit avant d'avoir un cache-miss.
– une boucle externe permet de répéter l'opération afin de calculer une valeur moyenne.

Ainsi, en faisant varier le nombre de valeurs lues dans la boucle interne, on va pouvoir faire des lectures soit dans le cache de niveau un, soit dans le cache de niveau deux. En effet, si l'on veut lire dans le cache de niveau deux, il suffit alors de faire des lectures dans un tableau plus grand que la taille du cache de niveau un.

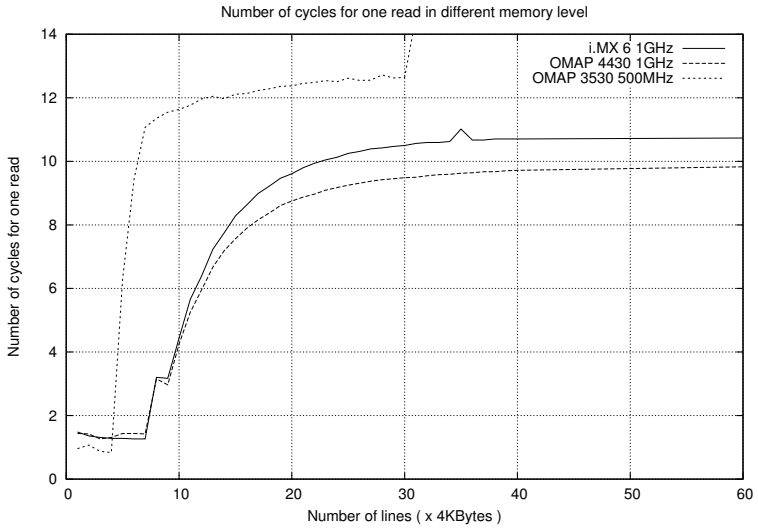

FIGURE 4.9: Nombre de cycles pour lire une donnée dans les différents niveaux de cache.

La figure 4.9 montre dans la première partie (gauche) le temps d'accès en cycles pour accéder à une donnée dans le cache de niveau un. On observe que le temps d'accès, pour trois différentes plates-formes, est d'environ un cycle. Sur le deuxième palier de la courbe, on peut observer le temps d'accès pour une donnée dans le cache de niveau deux. Dans ce cas, le temps d'accès est d'environ dix cycles. Ce benchmark confirme bien les affirmations présentes dans la littérature.

Nous avons donc un modèle paramétrique de la performance, s'appuyant à la fois sur des paramètres logiciels (le nombre d'instructions, le nombres d'accès mémoires...) et sur des paramètres matériels (la fréquence du processeur, la taille des caches,...). Les comportements dynamiques comme le taux de cache miss et le nombre de mauvaises prédictions de branchement sont évalués grâce à l'étape de profiling.
Le fait d'avoir des modèles paramétriques permet aussi de simplement rajouter de nouveaux paramètres si nécessaire (par exemple le nombre d'opérations flottantes).

4.3.2 Consommation électrique

Deux différentes méthodes d'estimation de la consommation du système sont abordées ici. Tout d'abord une méthode gros grain effectuant une estimation moyenne de la consommation d'énergie du bloc processeur/-caches L1/cache L2 et une méthode grain fin séparant l'estimation en deux parties : la consommation des coeurs de processeur et la consommation des mémoires cache.
Pour estimer la consommation électrique d'un coeur de processeur, nous utilisons un modèle de consommation dépendant du nombre d'instructions exécutées (informations récupérées par le profiling puis tracées au cours de l'exécution), du courant de fuite ou "*leakage*" (dépendant de la technologie) et du nombre d'instructions par cycle.
L'estimation de la consommation électrique des mémoires cache, est quant à elle estimée par rapport au nombre d'accès qui sont effectué sur chaque mémoire et du *leakage*.

La consommation d'énergie du système s'appuie sur la trace d'exécution dynamique qui a préalablement été créée comme on peut le voir dans la figure 4.10. En sortie, nous obtenons alors une trace de l'énergie

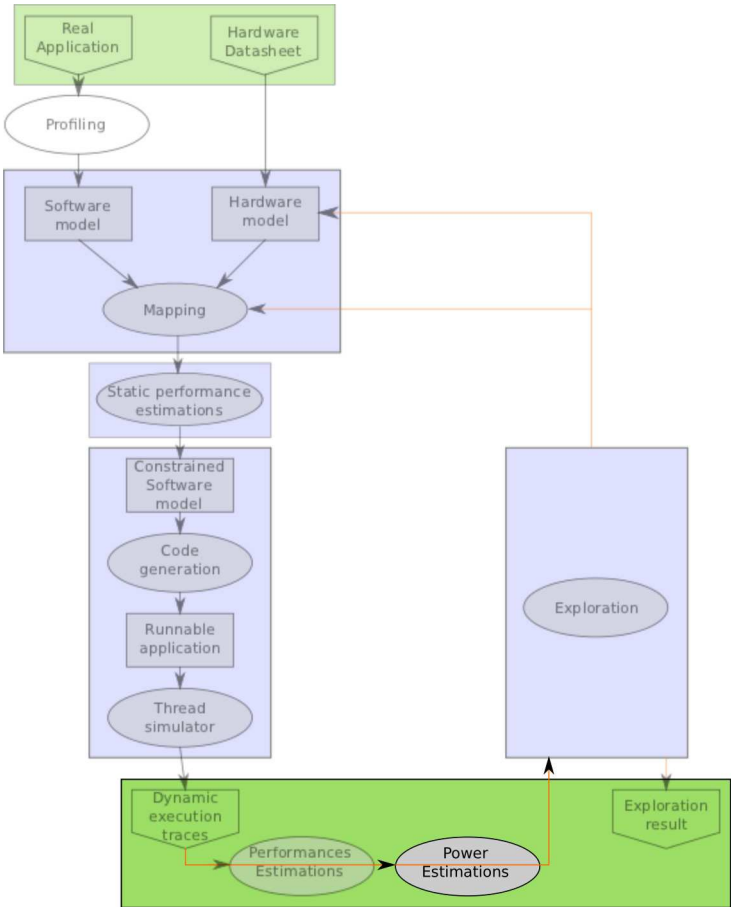

FIGURE 4.10: Flot d'estimation de la consommation d'énergie.

consommée au cours du temps. Il est donc tout d'abord nécessaire de créer les modèles de consommation électrique qui permettront d'estimer la consommation du système à l'aide des traces.

La sonde PowerTrace de Lauterbach et le banc de mesure Open-PEOPLE

Pour créer et valider nos modèles de consommation, nous devons être capable de mesurer précisément la consommation d'un système embarqué. En effet, nos modèles sont basés sur des mesures effectuées préalablement sur une plate-forme matérielle de référence à l'aide de benchmarks bas niveaux spécifiques.
Pour obtenir nos valeurs initiales de consommation, nous avons tout d'abord utilisé une sonde proposée par la société Lauterbach. Cette sonde est très efficace et permet à la fois de debugger le code s'exécutant sur la plate-forme (grâce au bus ETM des architectures ARM par exemple), et de le corréler à la consommation électrique.
On obtient en sortie le temps et l'énergie consommée par chaque tâche s'exécutant sur la plate-forme avec une précision de l'ordre de la dizaine de nanosecondes / nanojoules. En théorie, cette sonde doit donc nous permettre de mesurer très finement la consommation d'énergie et par la suite de valider avec précision nos modèles de consommation.
L'utilisation de cette sonde présente un avantage intéressant : comme il n'est pas nécessaire d'instrumenter le code, l'utilisation de la sonde n'est donc pas intrusive ce qui ne perturbe pas le fonctionnement du système.

Malheureusement, suite à de nombreux problèmes d'utilisation de la sonde Lauterbach (impossibilité de mesurer les faibles courants et difficulté de mise en oeuvre sur certaines plates-formes), nous avons abandonné cette solution et cherché un autre moyen de mesurer la consommation d'énergie.

La deuxième solution de mesure de consommation que nous utilisons pour créer nos modèles est l'utilisation du banc de mesure Open-PEOPLE. En effet, Thales Communications and Security faisant partie du consortium Open-PEOPLE, nous avons accès au banc de mesure à distance disponible grâce à ce projet. Ce banc de mesure permet de mesurer à distance la consommation de différentes plates-formes embarquées. Les plates-formes et les instruments de mesure se trouvent à Lorient. Un logiciel a été développé durant le projet pour permettre d'envoyer une application sur la plate-forme, puis de récupérer automatiquement la consommation électrique et les temps d'exécution mesurés. Cette solution est particulièrement intéressante lorsque l'on ne souhaite pas investir dans des cartes ou des instruments de mesure.

C'est donc finalement la plate-forme du projet Open-PEOPLE qui a été utilisée pour effectuer des mesures de consommations d'énergie. Ces mesures nous ont alors permis de créer et de valider nos modèles de consommation.

Estimations gros grain

Nous avons utilisé deux types d'estimations pour la consommation électrique. La première estimation, appelée gros grain est à un niveau d'abstraction très haut. L'idée est de considérer le processeur et ses mémoires directes (cache de niveau un et deux) comme étant un seul bloc. Cela permet de créer une caractérisation de la consommation du composant très simplement sans avoir d'outillage spécifique.
La caractérisation est basée sur le principe que le processeur est soit actif (en train d'exécuter du code) soit inactif (en attente de code à exécuter). Comme nous allons le voir, cette approche est efficace pour obtenir rapidement et facilement une première estimation de la consommation d'un système. Soit T1 le temps durant lequel le processeur exécute le programme, Cf la consommation électrique associée, T2 le temps durant lequel rien ne s'exécute et Ci la consommation électrique associée. On peut alors calculer l'énergie E de la manière suivante :

$$E = T1 \cdot Cf + T2 \cdot Ci \tag{4.6}$$

Nous avons tout d'abord étudié cette méthodologie à partir d'une application vidéo décodeur H.264 qui présente l'avantage d'être instrumenté afin de connaître le temps d'exécution des tâches. En effet, les temps d'exécution peuvent être mesurés sur cette application grâce à une instrumentation du code. Cette instrumentation consiste à sauvegarder une trace d'exécution dans un buffer à chaque fois que le décodage commence et lorsqu'il s'arrête. Il est donc possible de récupérer le timestamp de chaque début et fin d'exécution (i.e. de repos), ce qui permet de facilement déterminer la durée d'exécution et de repos. Cette approche d'instrumentation est intrusive puisqu'elle augmente légèrement le temps d'exécution du code. Cependant ceci n'a pas énormément d'impact significatif sur la consommation d'énergie comme on peut le voir dans le tableau 4.4.
La procédure est la suivante :
Nous effectuons des expérimentations sur la plate-forme OMAP3530 en considérant le processeur ARM Cortex-A8 et ses mémoires cache comme étant une seul composant. Le tableau 4.3 représente la consommation électrique lorsque le processeur est chargé et lorsqu'il ne fait rien (idle) à différentes fréquences.

Fréquence (MHz)	Puissance d'exécution (W)	Puissance repos (W)
600	0.462	0.167
550	0.364	0.122
500	0.277	0.085

TABLE 4.3: Les différentes consommations du modèle haut niveau.

Ensuite, nous exécutons l'application en l'instrumentant afin de déterminer les moments d'activités ou d'inactivités du code s'exécutant sur la plate-forme. La dernière étape consiste à appliquer les modèles de puissance (exécution ou repos) aux différents segments de temps mesurés durant l'exécution de l'application. Pour finir, nous exécutons une vraie application et nous mesurons la consommation d'énergie grâce à la plate-forme de mesure Open-PEOPLE puis nous comparons les valeurs de consommation ainsi obtenues (Tableau 4.4).

Fréquence (MHz)	Energie estimée (J)	Energie mesurée (J)	Erreur (%)
600	1.849	1.783	3.6
550	1.491	1.436	3.7
550	1.117	1.168	4.3

TABLE 4.4: Comparaison de la consommation d'énergie du décodeur vidéo H.264. (instrumentation du code)

Le tableau 4.4 compare la consommation d'énergie mesurée par la plate-forme de mesure Open-PEOPLE, et la consommation d'énergie estimée avec notre modèle de haut niveau. Les résultats obtenus sont très satisfaisants pour le niveau d'abstraction utilisé.

La méthodologie semblant être efficace, nous avons ensuite effectué des essais avec la méthode d'estimation de consommation gros grain associée aux estimations de performance. Nous avons donc de la même manière évalué le temps d'exécution nécessaire pour décoder 50 images ainsi que le temps de repos du processeur pour respecter la contrainte temps-réel de 8 images par seconde.
Par exemple, pour une fréquence de 600MHz, nous avons estimé (grâce aux estimations de performances présentées précédemment) qu'il est nécessaire d'exécuter le code pendant 2.427 secondes et de rester au repos pendant 3.823 secondes.

Fréquence (MHz)	Energie estimée (J)	Energie mesurée (J)	Erreur (%)
600	1.759	1.783	1.36
550	1.403	1.436	2.35
500	1.089	1.117	2.57

TABLE 4.5: Comparaison de la consommation d'énergie du décodeur vidéo H.264. (estimation)

Le tableau 4.5 compare l'énergie estimée et mesurée par le système pour décoder 50 images. Les résultats montrent que même avec cette méthode assez simple de caractérisation de la consommation électrique et une estimation de la performance de l'application, on obtient une estimation de l'énergie consommée très précise pour trois fréquences du processeur.

Cette méthodologie fonctionne bien sur des applications effectuant beaucoup de calcul. Cependant, cette approche ne permet pas de prendre en compte certains comportements, comme par exemple lorsqu'une application effectue de nombreux cache miss et passe donc beaucoup de temps à attendre des données provenant du niveau de mémoire supérieur (RAM).
Comme nous allons le voir, la consommation du système n'est alors pas toujours la même suivant le code qui est exécuté.
La figure 4.11 montre en rouge, la puissance dissipée par le processeur et ses mémoires caches. La courbe verte représente la consommation de la mémoire externe (DRAM). Les quatre "pics" rouges représentent quatre exécutions de différentes applications. Nous avons créé des applications spécifiques qui ne font que du calcul ou que des accès soit dans la mémoire de niveau un, soit de niveau deux, soit dans la DRAM. Comme on peut le voir sur la courbe, la puissance dissipée par le processeur et ses mémoires est totalement différente suivant l'application exécutée par le processeur (tout en notant que nous avons testé des cas critiques peu probables dans la réalité).

Si l'on souhaite estimer différents types d'applications, il semble donc nécessaire de faire une estimation de la consommation avec un grain plus fin. Les résultats présentés montrent en effet que le modèle gros grain basé sur l'activité ou l'inactivité du processeur peut être insuffisant dans certains cas.

Estimations à grain fin

Comme nous l'avons mentionné précédemment, nous avons souhaité avoir des modèles distincts pour la consommation électrique du coeur de processeur et des mémoires. Or, dans les plates-formes embarquées, il est de nos jours impossible de mesurer séparément les consommations de ces différentes unités matérielles. Par contre, il est tout à fait possible de mesurer en même temps : le processeur, les caches L1 et le cache L2. D'où l'idée d'utiliser des benchmarks spécifiques permettant d'activer qu'une partie de la plate-forme afin d'évaluer séparément la consommation de chaque sous-partie.
La première solution consiste à créer un benchmark n'exécutant que des instructions, puis un benchmark

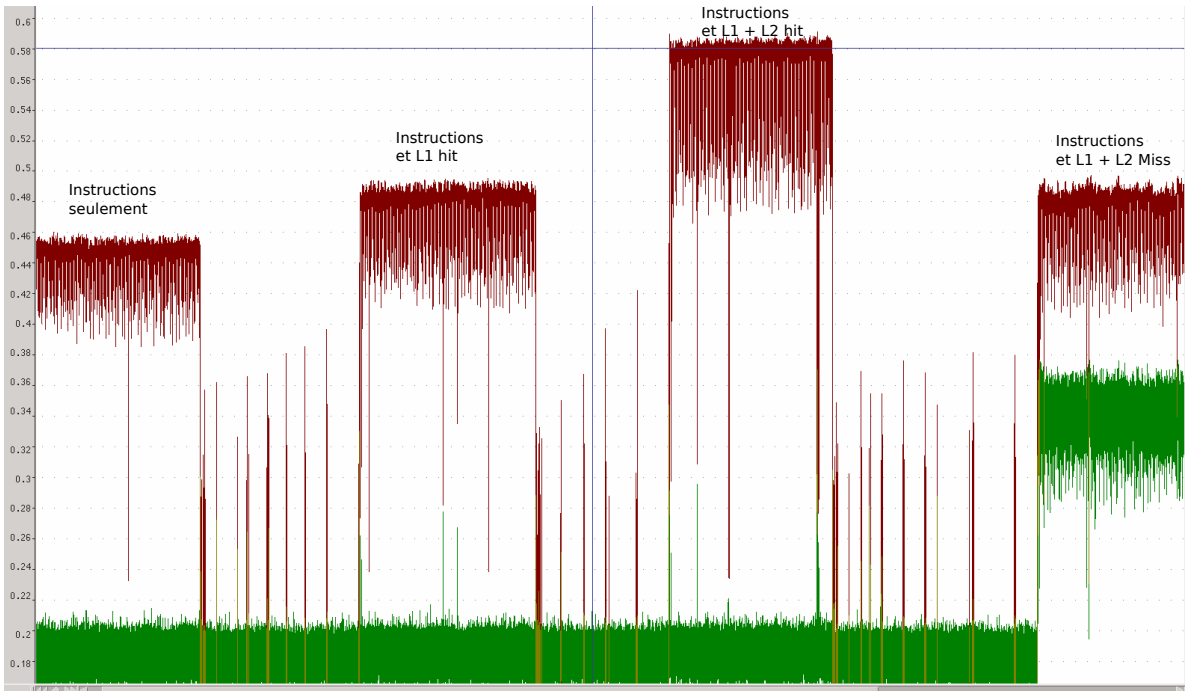

FIGURE 4.11: Différentes puissances dissipées suivant le programme exécuté.

n'exécutant que des lectures dans un petit tableau pour rester dans le cache L1 et enfin un benchmark lisant dans un tableau plus grand pour n'effectuer que des cache miss en L1 et aller lire en cache L2. Le problème pour ce dernier benchmark, est que lors d'un cache miss, le processeur se met en attente pendant le temps nécessaire pour la recherche de la donnée dans les différents niveaux de mémoire. Il n'est donc plus possible de visualiser l'impact du cache L2 puisque le processeur change d'état et consomme beaucoup moins que dans les deux benchmarks précédents.
L'idée est d'effectuer des écritures pour estimer la consommation du cache de niveau deux, à la place des lectures. En effet, lors d'une écriture dans la mémoire, le processeur envoie la donnée à écrire, puis continue directement à exécuter les prochaines instructions, ceci quelque soit le niveau mémoire (cache L1, L2 ou DRAM) où la donnée est écrite. Ce procédé est possible en partie grâce aux buffers présents avec les mémoires cache.
La figure 4.12 montre les trois benchmarks utilisés pour différencier la consommation des différentes parties du système.
Pour bien comprendre ces benchmarks, il est important de connaître les politiques d'écriture des caches. Les deux politiques principales utilisées sont :

Write through no-allocate : Cette politique est très utilisée pour les caches de niveau un. Elle permet, lors d'un hit en écriture, d'écrire la donnée à la fois dans le cache de niveau un et de niveau deux (d'où le terme write through). Cela facilite la cohérence des données dans le cas des architectures multiprocesseurs. Lors d'un cache miss dans le cache de niveau un, la donnée est directement écrite dans le cache de niveau deux, sans rapatrier la valeur dans le cache de niveau un (d'où le terme no-allocate).

Write back allocate : Cette deuxième politique est plutôt utilisée pour les caches de niveau deux. Contrairement à la politique précédente, lorsqu'un cache hit en écriture est effectué dans le cache de niveau deux, la valeur est seulement écrite dans le cache de niveau deux, et non dans la mémoire du niveau supérieur. De plus, lors d'un cache miss, une allocation est effectuée ce qui signifie que la ligne de cache est rapatriée de la mémoire de niveau supérieur et la donnée correspondante est remplacée par la nou-

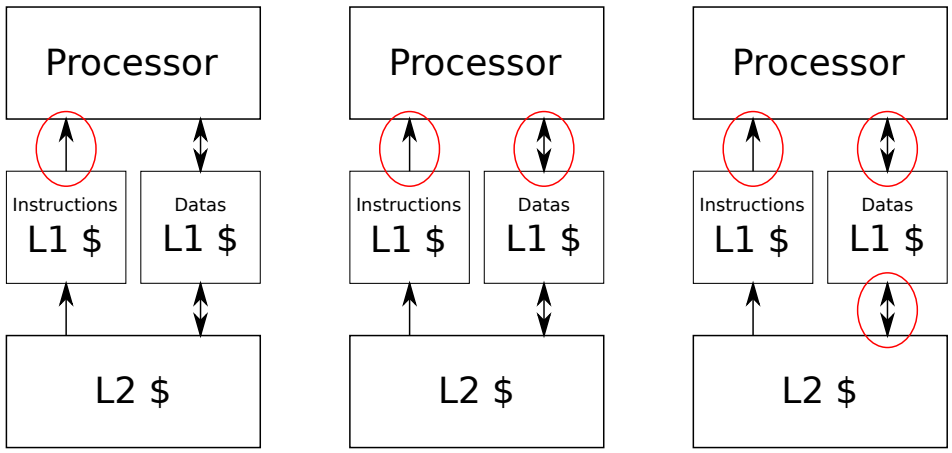

FIGURE 4.12: Définition des benchmarks mémoire. Instructions seulement / Instructions + L1 / Instructions + L1 + L2

velle valeur (si la donnée remplacée a été modifiée par une écriture précédente, elle a été préalablement écrite en mémoire avant son remplacement).

Ces deux politiques sont utilisées dans toutes les plates-formes standards (dont celles que nous utilisons).

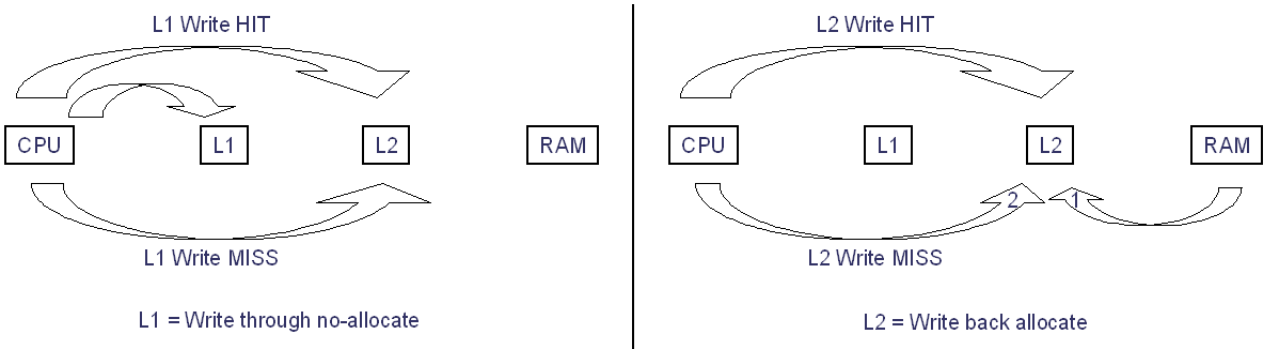

FIGURE 4.13: Les deux politiques de cache pour l'écriture d'une donnée.

La figure 4.13 montre les deux différentes politiques de cache pour l'écriture d'une donnée. Sur la gauche, on retrouve la politique des caches de niveau un (Write through no-allocate), et sur la droite, la politique pour les caches de niveau deux (Write back allocate).

Nous avons défini trois différents benchmarks qui sont :
- Benchmark1 : exécute uniquement des instructions de calcul (des additions),
- Benchmark2 : exécute des instructions et lit des données dans le cache de niveau un,
- Benchmark3 : exécute des instructions et écrit des données dans le cache de niveau un et le cache de niveau deux.

Ces exécutions s'effectuent sans temps de pause du processeur afin de mesurer des valeurs cohérentes.

Considérons les paramètres suivants :
- *leakage* : les fuites en consommation du système,
- *cpu* : la consommation du processeur pour une instruction exécutée,
- $I\$1$: la consommation du cache d'instructions de niveau un pour une instruction exécutée,

- $D\$1$: la consommation du cache de données de niveau un pour accéder à une donnée,
- $L\$2$: la consommation du cache de niveau deux pour accéder à une donnée. (on néglige la lecture des instructions du cache de niveau 2, en effet, ces instructions une fois lues restent dans le cache de niveau 1 pendant la durée du benchmark)

Exécutons à présent les trois benchmarks.

Soit A, le nombre d'instructions exécutées durant le benchmark1, $time1$ le temps d'exécution du benchmark et $E1$ l'énergie consommée par la plate-forme, on a alors :

$$E1 = A \cdot cpu + A \cdot I\$1 + time1 \cdot leakage \tag{4.7}$$

Soit B le nombre d'instructions exécutées durant le benchmark2, C le nombre de lecture en cache, $time2$ le temps d'exécution du benchmark et $E2$ l'énergie consommée par la plate-forme, on a alors :

$$E2 = B \cdot cpu + B \cdot I\$1 + C \cdot D\$1 + time2 \cdot leakage \tag{4.8}$$

Soit B le nombre d'instructions exécutées durant le benchmark3, C le nombre d'écritures en cache, $time2$ le temps d'exécution du benchmark et $E3$ l'énergie consommée par la plate-forme, on a alors :

$$E3 = B \cdot cpu + B \cdot I\$1 + C \cdot D\$1 + C \cdot L\$2 + time2 \cdot leakage \tag{4.9}$$

Nous connaissons les valeurs de E1, E2, E3 (mesures des énergies), de A, B, C (valeurs connues en créant le logiciel), de time1, time2 (les temps d'exécution mesurés) et de *leakage* (la valeur mesurée lorsque rien ne se passe).
Il est donc possible en utilisant les trois équations précédentes, de déduire la consommation de chaque bloc.
Pour le calcul de la consommation du cache de niveau 2, il suffit de calculer $E3 - E2 = C \cdot L\$2$
Donc

$$L\$2 = \frac{E3 - E2}{C} \tag{4.10}$$

Le calcul pour les autres valeurs nécessite un peu plus de calcul. Tout d'abord, il faut calculer le coût d'une instruction pour le processeur ET le cache de niveau un :
$E1 = A \cdot cpu + A \cdot I\$1 + time1 \cdot leakage$
$E1 = A \cdot (cpu + I\$1) + time1 \cdot leakage$
$A \cdot (cpu + I\$1) = E1 - time1 \cdot leakage$

$$(cpu + I\$1) = \frac{E1 - time1 \cdot leakage}{A} \tag{4.11}$$

On peut ensuite remplacer cette valeur dans l'équation 4.8 :
$E2 = B \cdot (cpu + I\$1) + C \cdot D\$1 + time2 \cdot leakage$
$E2 = B \cdot \frac{E1 - time1 \cdot leakage}{A} + C \cdot D\$1 + time2 \cdot leakage$
On peut alors en déduire la valeur pour la consommation du cache de données de niveau 1 : $D\$1$.

$$D\$1 = \frac{E2 - B \cdot \frac{E1 - time1 \cdot leakage}{A} - time2 \cdot leakage}{C} \tag{4.12}$$

Puisque les caches de niveau un pour les données ou pour les instructions sont identiques, on en déduit que leurs consommations sont égales et donc : $I\$1 = D\$1$

On peut ainsi retrouver la consommation du processeur uniquement pour une instruction exécutée grâce à l'équation 4.7 :

$$cpu = \frac{E1 - time1 \cdot leakage}{A} - I\$1 \tag{4.13}$$

Ces trois types de benchmarks ont été expérimentés et validés sur la plate-forme OMAP3530 contenant un ARM Cortex-A8 pour toutes les fréquences disponibles. Les résultats obtenus sont présentés dans la Table 4.6 ci-dessous.

Fréquence (MHz)	Energie L2 (nJ)	Energie L1 (nJ)	Energie proc (nJ)	Fuites (W)
600	0.122	0.085	0.200	0.181
550	0.115	0.084	0.167	0.147
500	0.103	0.072	0.151	0.117
250	0.076	0.056	0.127	0.038
125	0.072	0.052	0.110	0.009

TABLE 4.6: Calcul de la consommation de chaque partie du Cortex-A8.

Ces premières valeurs ont été obtenues grâce aux mesures effectuées sur le banc de mesure du projet Open-PEOPLE. Elles vont dans un premier temps nous permettre d'évaluer la consommation électrique d'une application s'exécutant sur la plate-forme.

On peut en premier lieu observer que l'exécution d'une instruction est l'opération qui consomme le plus dans un bloc processeur. Le second élément le plus gourmand en consommation est l'accès au cache de niveau 2. En effet, cet accès est coûteux en temps, ce qui consomme beaucoup d'énergie (contrairement au traitement d'une instruction qui elle va consommer beaucoup et en peu de temps). Par contre, le nombre d'accès au cache de niveau 2 est souvent très largement inférieur au nombre d'instructions exécutées dans une application. Un accès au cache de niveau 1 est ce qui consomme le moins mais le nombre d'occurrences est très important étant donné que chaque instruction et donnée provient du cache de niveau 1.
Il faut aussi observer que les fuites varient en fonction de la fréquence. En effet, lorsque l'on change la fréquence, la tension varie en même temps afin de minimiser la consommation d'énergie.

Un des inconvénients de cette approche est qu'il n'est pas possible de vérifier ces valeurs puisque la consommation d'énergie de chaque bloc n'est pas mesurable sur les plates-formes actuellement disponibles. Nous comparerons alors dans le chapitre 5 les résultats obtenus de manière globale (comparaison de la consommation de toute la plate-forme) à l'aide de la plate-forme mesure du projet Open-PEOPLE.

4.4 Le générateur de code : Waveperf

La figure 4.14 présente le générateur de code Waveperf au sein de notre méthodologie.

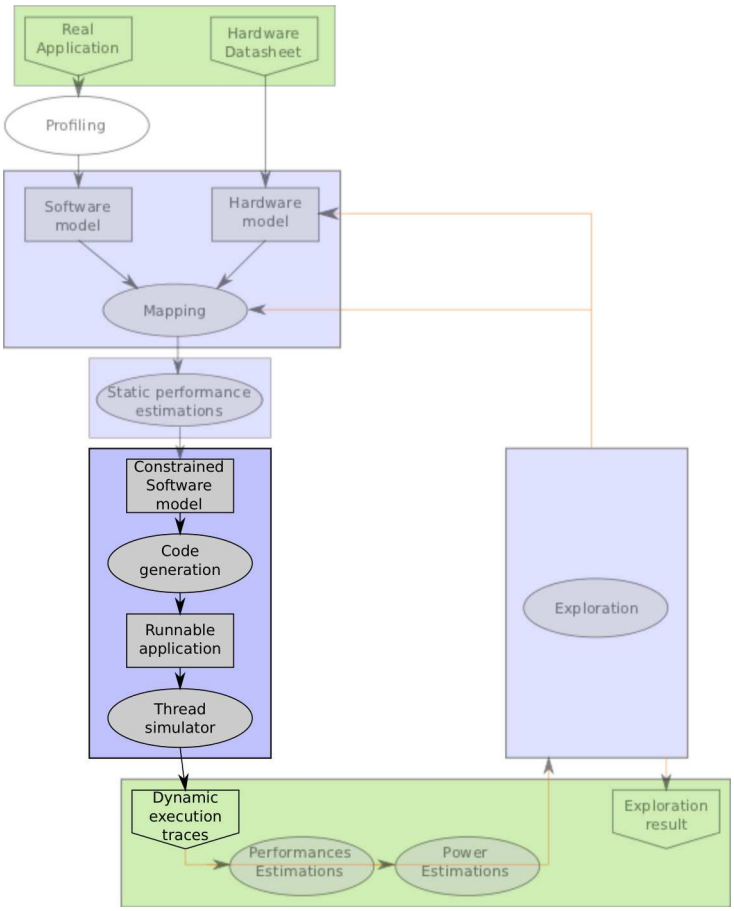

FIGURE 4.14: Les différentes étapes de Waveperf.

Nous allons dans un premier temps décrire la partie permettant d'effectuer la génération de code ainsi que la simulation dynamique du comportement de l'application, puis nous aborderons deux exemples d'utilisation.

4.4.1 Description du langage

Comme nous l'avons vu précédemment, Waveperf permet à partir d'une spécification utilisant des fichiers texte de configuration, de générer du code exécutable C++. Un fichier de configuration est requis pour chaque composant logiciel ainsi que pour la description haut niveau de l'architecture.

Le listing 4.7 représente les différentes syntaxes utilisable lorsque l'on ne souhaite pas utiliser de machine d'état dans la tâche que l'on modélise.
On retrouve bien les trois blocs que nous avions mentionnés dans la section 2.1.6 (component, behaviour, characteristic). Ces trois blocs permettent, comme leurs noms l'indiquent, de décrire le composant de manière externe (les entrées et les sorties), son comportement (ce qu'il se passe lorsqu'il est activé par un signal) et ses caractéristiques d'exécution (nombre d'opérations, temps d'exécution).

```
component [NomDuBloc]
{
    provides Runnable [NomDuSignalEntrant];
    uses     Runnable [NomDuSignalSortant];
};

behaviour [NomDuBehaviourDuBloc] of [NomDuBloc]
{
    [NomDuSignalEntrant].run
    {
        ([NumEtat]) [NbDExecution] { [NomDuSignalSortant].run }
    }
};

characteristics( [typeOperation] ) [NomDesCaracteristiquesDuBloc] of [NomDuBehaviourDuBloc]
{
    [NomDuSignalEntrant].run  { ([NumEtat]) { [TempsAvantSignal] [TempsApresSignal] } }
};
```

Listing 4.7: Fichier de configuration de bloc logiciel sans machine d'état.

Un exemple un peu plus complexe utilisant une machine d'état est montré dans le listing 4.8.

```
component [NomDuBloc]
{
    provides Runnable [NomDuSignalEntrant];
    uses     Runnable [NomDuSignalSortant];
};

behaviour [NomDuBehaviourDuBloc] of [NomDuBloc]
{
    _var_ { [typeVar] [NomDeLaVariable]; };
    _init_ { [NomDeLaVariable] = [ValeurInitDeLaVariable]; };
    _state_ { [NomEtat1] [NomEtat2] }
    _initial_state_ { [NomEtat1] };

    [NomDuSignalEntrant].run
    {
        ([NumEtat]) [NomEtat1] -[ (this_->[NomDeLaVariable] [Condition] ) ]-> [NomEtat2] [
            NbDExecution] { [NomDuSignalSortant].run } ! { this_->[NomDeLaVariable] [affectation
            ]; }
        ([NumEtat]) [NomEtat2] -[ (this_->[NomDeLaVariable] [Condition] ) ]-> [NomEtat1] [
            NbDExecution] { [NomDuSignalSortant].run } ! { this_->[NomDeLaVariable] [affectation
            ]; }
    }
};

characteristics( [typeOperation] ) [NomDesCaracteristiquesDuBloc] of [NomDuBehaviourDuBloc]
{
    [NomDuSignalEntrant].run  { ([NumEtat]) { [TempsAvantSignal] [TempsApresSignal] } }
};
```

Listing 4.8: Fichier de configuration de bloc logiciel avec machine d'état.

Les différents termes du langage ainsi que les options disponibles pour décrire les tâches logicielles sont présentés dans le tableau 7.1 présent en annexe.

Le listing 4.9 représente les différentes possibilités pour créer une architecture logicielle en utilisant le formalisme de Waveperf.
Ce fichier permet d'effectuer l'instanciation des blocs et les différentes connexions entre eux.

```
include [nomDuFichierBloc1].txt;
include [nomDuFichierBloc2].txt;

component_instance [nomDuBehaviourDuBloc1] [nomDeLInstanceDuBloc1] [
    NomDesCaracteristiquesDuBloc1];
component_instance [nomDuBehaviourDuBloc2] [nomDeLInstanceDuBloc2] [
    NomDesCaracteristiquesDuBloc2];

connection( [typeConnection] ) [nomDeLaConnection] [nomDeLInstanceDuBloc1].[NomDuneSortie] [
    nomDeLInstanceDuBloc2].[NomDuneEntree] [nomDuPortDeSynchronisation];
configuration [NomDeLInstanceDuBloc]->configure_priority_and_sched_fifo([prioritee], [boolFifo])
    ;

component_instance Raw_ip_iterface [NomDeLInstanceDuBloc] ip_interface;
configuration [NomDeLInstanceDuBloc]->configure_priority_and_sched_fifo([prioritee], [boolFifo])
    ;

component_instance Timer_impl [NomDeLInstanceDuBloc] timer;
configuration [NomDeLInstanceDuBloc]->configure_timerspec_and_sched_fifo([TempsDepartSec], [
    TempsDepartNSec], [IntervalSec], [IntervalNsec], [boolFifo], [prioritee]);

entry_point [NomDeLInstanceDuBloc].[NomDuneEntree];
final_point [NomDeLInstanceDuBloc].[NomDuneSortie];
```

Listing 4.9: Fichier d'architecture logicielle.

Les différents termes du langage ainsi que les options disponibles pour décrire l'architecture logicielle sont présentés dans le tableau 7.2 présent en annexe.

Afin de pouvoir modéliser des comportements pour des architectures multi-processeurs, un autre paramètre a été ajouté. Ce paramètre, appelé "configure_affinity", permet au concepteur (ou à la partie exploration) de choisir le processeur sur lequel la tâche va s'exécuter. Dans ce cas, nous avons donc une assignation statique des tâches sur les processeurs, puisqu'une tâche assignée à un processeur ne peut pas migrer. Par contre, si l'affinité processeur n'est pas activée, la tâche pourra migrer (grâce au scheduler de l'OS) sur n'importe quel processeur. Dans ce cas cependant, l'activité de chaque processeur restera inconnue lors de la trace d'exécution.

Une fois l'étape de modélisation de l'application achevée, l'outil permet ensuite de vérifier visuellement le modèle créé. La figure 4.15 illustre la représentation graphique d'une application radio générée automatiquement par *Waveperf*. On peut y observer les différents composants ainsi que les connexions entre eux. L'outil *Waveperf* permet de modéliser rapidement et avec précision des applications complexes.

Les sous-sections suivantes montrent deux exemples d'utilisation de *waveperf*.

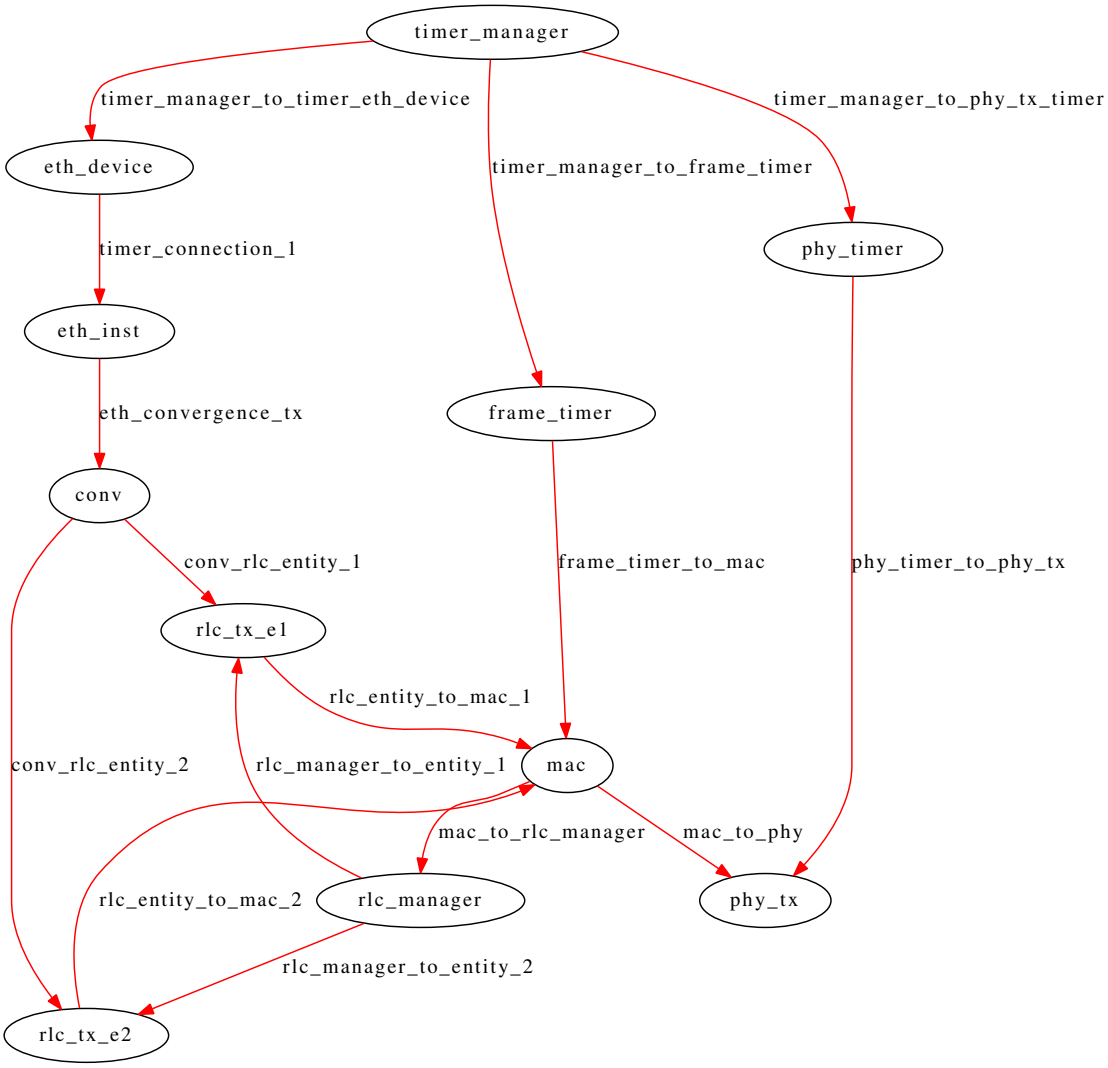

FIGURE 4.15: Représentation graphique du benchmark radio.

4.4.2 Utilisation classique de *Waveperf*

Génération de benchmark

Comme il a été mentionné précédemment, *Waveperf* est capable de générer du code correspondant aux benchmarks spécifiés dans les standards Posix ou Xenomai. Des objets C++ sont créés par l'outil pour chaque composant du système. Ces composants sont exactement les mêmes que ce soit pour le standard Posix ou Xenomai. La différence principale réside dans la création des tâches et des timers. En effet, une librairie est créée pour chaque standard. Dans ces librairies, on peut trouver l'implémentation de la partie *Characteristics*, l'interaction entre les composants (création de thread et activation du composant) et les timers.
Tout d'abord, voyons comment ont été implémentées chaque parties pour le standard Xenomai :

– Pour les connexion asynchrones, au démarrage du benchmark, le générateur utilise "rt_task_create()" puis "rt_task_start()" pour créer et démarrer les tâches, puis crée un sémaphore pour la tâche afin de

la mettre en pause grâce à "rt_sem_create()". Lorsque la connexion est activée par une autre tâche (ou un timer) le sémaphore est envoyé "rt_sem_v()" et la tâche est débloquée.
- Pour la création des timers, la fonction "rt_task_set_periodic()" est utilisée en passant comme paramètre le nombre de nano-secondes de période.
- Pour la Characteristics "Timing_in_ms", la méthode utilisée par *Waveperf* consiste à exécuter un grand nombre de boucles avec des instructions de calcul durant la phase d'initialisation du benchmark. L'application fait ceci pendant 2 secondes et estime alors le temps requis pour faire une boucle. Ensuite, lorsqu'un appel est effectué pour obtenir un "temps d'exécution", le nombre de boucles correctes est exécuté. Ceci permet d'avoir une réelle exécution dynamique lorsqu'une préemption sera faite pendant ces boucles, elles devront continuer par la suite, ce qui ne serait pas le cas si on avait utilisé un "sleep". La calibration est faite avec la fonction "rt_timer_read()" pour obtenir le temps avant et après le grand nombre de boucles.

De la même façon, l'implémentation pour le standard Posix est décrite ci-dessous :
- Les connexion asynchrones sont créées avec les fonctions "pthread_create()" et "pthread_attr_setschedparam()" pour les priorités. Les sémaphores sont envoyés grâce à "sem_post()" pour débloquer les threads.
- Les fonctions "timer_settime()" et "timer_create()" sont utilisées pour créer un timer et lui fixer sa période.
- Enfin, pour obtenir le temps des boucles de calibration, le générateur utilise la fonction "gettimeofday()".

On peut donc assez facilement envisager d'implémenter des bibliothèques pour d'autres OS (VxWorks, RTAI, ...) ou standards puisque seules quelques fonctions ont besoin d'être ré-implémentées (6). Grâce à cela, nous sommes capables de créer des applications exécutables sur des architectures cibles (PC ou embarqués) avec simplement un modèle de haut-niveau en entrée.

Résultats d'utilisation de *Waveperf*

L'analyse des interruptions Un des objectifs principaux du générateur de benchmarks est d'évaluer les performances temps-réel des plates-formes (latence d'interruption, préemptions,...). La boite à outils est pour le moment capable de générer une application pour les configurations suivantes :
- L'utilisation du standard Posix avec Linux
- L'utilisation du standard Posix avec Xenomai
- L'utilisation des drivers natifs Xenomai

Afin de tester l'impact des interruptions sur une plate-forme (temps réel) embarquée, un modèle simple d'une application de radio communication a été créé. Trois tâches principales sont implémentées avec différentes priorités. La tâche ayant la plus haute priorité correspond à la simulation de la couche physique (PHY). Un timer est implémenté pour simuler l'activation de la tâche PHY à une fréquence de 2000Hz. Ensuite, la deuxième tâche simule la couche MAC avec une fréquence de 100Hz. La troisième tâche, ayant la priorité la plus basse, représente le buffer d'entrée de la couche protocolaire ethernet. Une description simplifiée de cette application radio est présentée sur la Figure 4.16. La couche RLC établit la connexion entre l'équipement de l'utilisateur et le contrôleur radio. Elle contient des fonctions classiques du niveau 2 tel que le transfert des données sur l'interface radio. Elle réalise la fonction de segmentation des paquets en des unités de taille prédéfinie. Elle assure aussi le réassemblage des paquets à la réception.

Pour s'assurer du comportement correct de la future application, le thread PHY ne doit pas être interrompu par un autre thread et doit être parfaitement périodique. Cet exemple de modélisation d'application a été créé pour une architecture mono-coeur pour s'assurer que tous les threads s'exécutent sur le même processeur (et que les interruptions arrivent toutes au même endroit).

Le benchmark a donc tout d'abord été généré pour le standard Posix et utilisé sur une plate-forme embarquée utilisant un noyau Linux 2.6.26. Le noyau est de plus configuré avec les timers de haute résolution (améliorant la précision des timers).

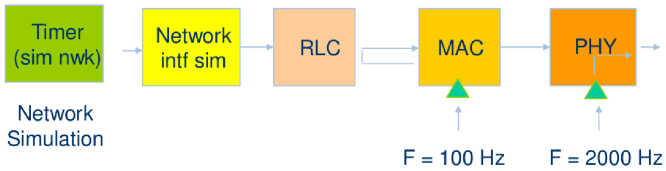

FIGURE 4.16: Description simplifiée du benchmark radio.

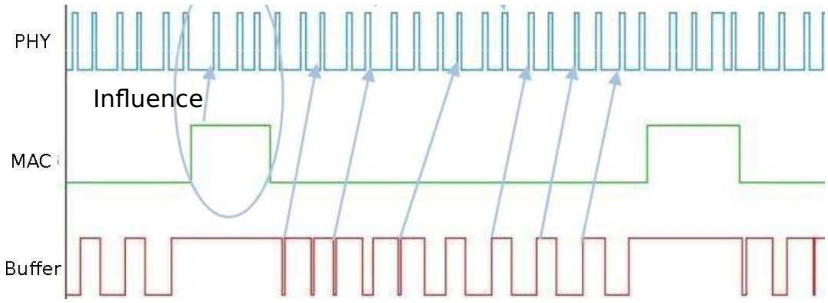

FIGURE 4.17: Résultat de l'exécution du benchmark pour une implémentation Posix.

La figure 4.17 montre la trace d'exécution des tâches de l'application radio générée. L'abscisse représente le temps durant lequel s'exécute l'application, alors que l'ordonnée représente l'exécution des différentes tâches. Lorsqu'un niveau est bas, la tâche est en attente et ne s'exécute pas. Lorsque le niveau est haut, cela signifie que la tâche a commencé à s'exécuter. La tâche PHY (en haut de la figure 4.17) possède la plus haute priorité et doit s'exécuter à une fréquence de 2000Hz. Cependant, comme on peut l'observer sur la figure 4.17, l'exécution de la tâche PHY n'est pas régulière. Ceci est dû à l'exécution/activation des autres threads. Une première analyse du problème montre que le standard Posix couplé à Linux n'est pas suffisant pour nos besoins temps-réel (tâches très courtes avec une fréquence élevée). Ce problème a aussi été observé en utilisant le standard Posix avec un Linux utilisant Xenomai. Le problème vient donc du standard Posix.
Sur la base de ces premiers résultats, nous avons alors généré l'application pour une utilisation native Xenomai afin de tester si le comportement est amélioré.

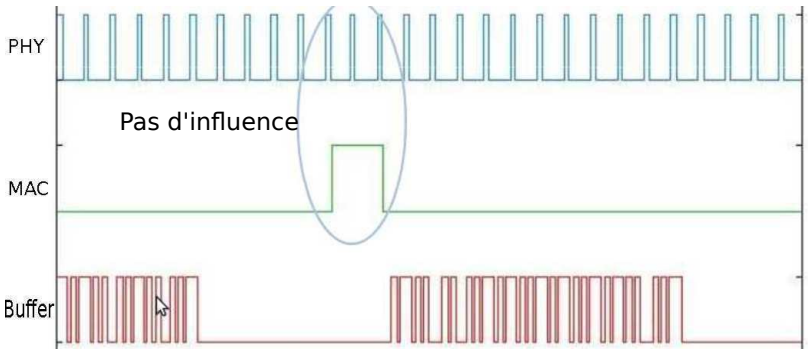

FIGURE 4.18: Résultat de l'exécution du benchmark pour une implémentation native Xenomai.

La figure 4.18 représente le comportement de l'application générée en utilisant les interfaces natives de Xenomai. Comme on peut le voir, le thread PHY s'exécute maintenant de façon parfaitement régulière et n'est plus perturbé par l'exécution/activation des autres threads. Ni le timer de la couche MAC, ni les packets provenant de l'ethernet ne modifient le comportement de la tâche la plus prioritaire.

Le générateur permet donc à l'utilisateur de modéliser différentes applications de test très facilement et ensuite de générer le code correspondant. Ce code peut utiliser un jeu d'interfaces standards (Linux Posix, Xenomai Posix, Xenomai Native). L'application générée permet alors à l'utilisateur de détecter d'éventuels problèmes provenant des interruptions (à cause de l'OS ou des interfaces utilisées) sur une plate-forme embarquée.

Analyse des performances temporelles Un autre objectif du générateur de benchmark est d'évaluer la performance d'un processeur. Pour cela, chaque composant peut utiliser/exécuter une des trois différentes "characteristics" :

- Un nombre d'instructions : le processeur exécute le nombre d'instructions souhaitées.
- Une attente active : le processeur exécute des instructions pendant un certain laps de temps.
- Une attente passive : le processeur s'endort pendant un certain temps.

Le nombre d'instructions est principalement utilisé pour estimer la performance en terme de puissance de calcul d'une plate-forme. Cette valeur peut aussi déterminer si un processeur est capable de respecter les contraintes temps-réel d'une application. Dans [104] par exemple, les auteurs sont capables de déterminer le nombre d'instructions d'une application sans aucun profiling. Les auteurs montrent qu'il est possible d'extraire le nombre d'instructions d'une application radio logiciel simplement grâce à la complexité de son algorithme et son débit. La pause active est utilisée si l'on connaît le temps d'exécution de la tâche à l'avance, ou si seulement les interruptions sont testées. La pause passive est utilisée pour modéliser, par exemple, une latence entre la transmission d'une donnée sur l'interface radio et la réception de cette donnée.

En sortie de *waveperf*, chaque benchmark exécuté est accompagné d'une trace d'exécution montrant l'activité des différents processeurs (chargé lorsque le niveau est haut ou au repos lorsque le niveau est bas), ainsi que l'activité de chaque tâche. La Figure 4.19 représente l'exécution d'une application vidéo décodeur H.264 avec le temps d'exécution de ses différentes tâches. La performance peut alors être mesurée et les problèmes, comme par exemple la violation des contraintes temps-réel ou la surcharge processeur, peuvent facilement être identifiés.

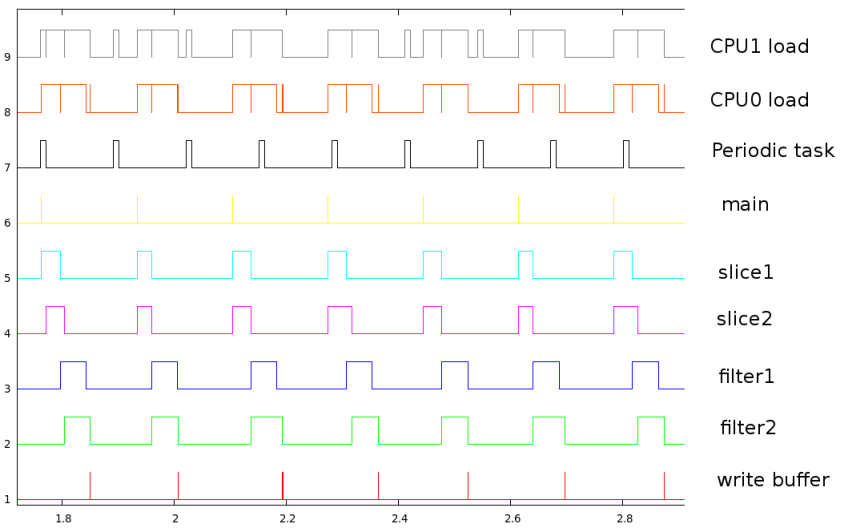

FIGURE 4.19: Trace du modèle de l'application décodeur vidéo H.264 sur un processeur double-coeur.

Différents types de benchmarks ont été modélisés en utilisant *waveperf*. Par exemple, une application vidéo décodeur H.264 (tableau 4.7) ou une couche physique de radio logicielle. Comme illustré sur le tableau 4.7 nous sommes capables de générer une application en seulement une journée avec une erreur d'estimation de performance d'environ 10% par rapport à l'application réelle.

Platform name	Real Application	Benchmark Model Application	error (%)
With filter OMAP3 @ 600MHz	8.9 FPS	9.73 FPS	9.7
Without filter OMAP3 @ 600MHz	19.3 FPS	21.2 FPS	9.8
Man's month to develop application	6	0.05 (1 day)	

TABLE 4.7: Comparaison de performance entre le benchmark généré et l'application vidéo décodeur H.264 réelle.

Waveperf est capable de générer des benchmarks pour différents systèmes d'exploitation comme Linux, LynxOS ou Xenomai et pour toutes les plates-formes matérielles supportant ces OS. Par exemple, les benchmarks générés ont été exécutés (après recompilation) sur les plates-formes ARM Cortex-A8, ARM Cortex-A9, Intel x86 et Freescale QorIQ.

Conclusion Le langage *Waveperf* et son générateur associé permettent de très rapidement modéliser et générer une application multi-tâche uniquement dépendante de l'OS. Ceci nous permet de simuler le comportement d'une application comportant plusieurs threads, tout en utilisant différentes priorités, ou des allocations processeurs ainsi que différentes préemptions entre tâches.
Outre son utilisation pour la création de benchmarks pour nos plates-formes embarquées, cet outil va aussi être au coeur de notre méthodologie d'exploration d'architectures. En effet, il va être utilisé pour générer du code qui s'exécutera sur un processeur hôte (PC) pour simuler l'exécution d'une application comme si elle s'exécutait sur la plate-forme matérielle réelle.

4.5 L'execution et la sortie

Nous allons maintenant aborder les différentes parties qui sont fournies en sortie de l'outil d'estimation (l'exploration sera abordée dans la section suivante) comme le montre la figure 4.20. Comme nous l'avons vu

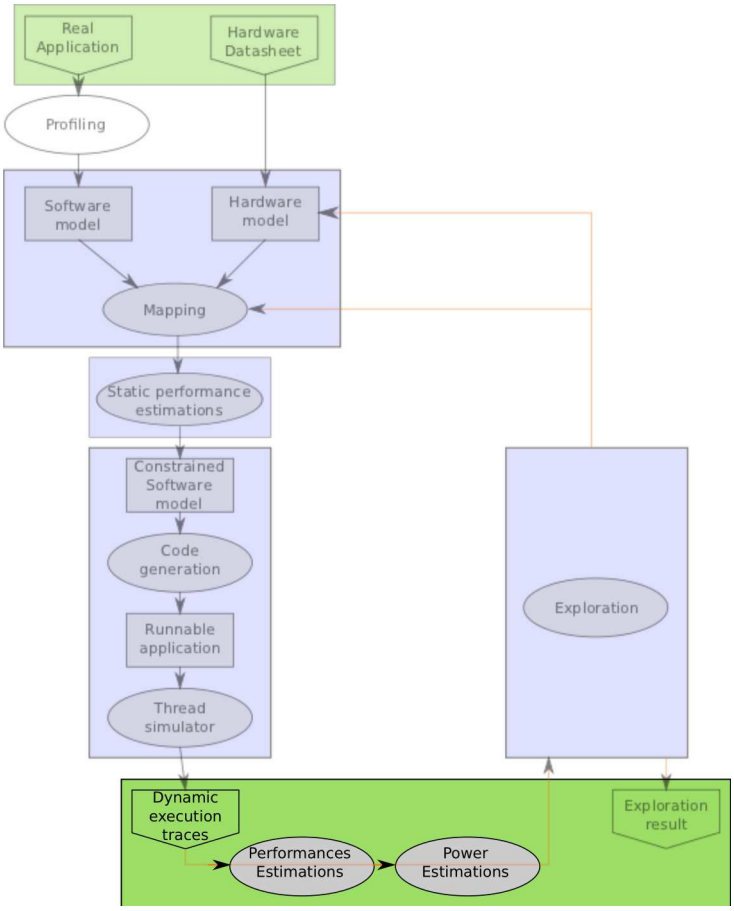

FIGURE 4.20: Graphique de la partie exécution et traces de sorties.

précédemment, *Waveperf* est capable de générer du code exécutable d'un modèle d'application pour n'importe quel processeur utilisant POSIX. Nous combinons donc les estimations de performance et le profiling des tâches avec *Waveperf* afin d'être capable de simuler des tâches s'exécutant sur la plate-forme cible. Le but est de simuler le comportement dynamique de l'application, en d'autres termes les interactions entre les différentes tâches. Le code généré est exécuté sur l'ordinateur hôte utilisant Linux et la norme POSIX. Chaque tâche ayant un temps d'exécution calculé pour le processeur embarqué sur lequel elle doit s'exécuter, la simulation se comporte comme si l'application réelle s'exécutait sur la plate-forme réelle.
Les différents processeurs de l'ordinateur hôte sont en fait utilisés comme si ils faisaient partis des différentes unités de calcul (GPP, DSP) de la plate-forme embarquée. En particulier, le parallélisme de l'application est simulé de la même manière que sur la plate-forme cible.
La figure 4.21 représente sur la droite le modèle de l'application vidéo decodeur H.264. Les différentes tâches y sont représentées ainsi que leurs dépendances et les processeurs auxquels elles sont assignées. De plus, une tâche supplémentaire (periodic_task) est ajoutée afin d'effectuer des tests de préemptions. Le graphique de gauche montre l'exécution de chaque tâche.
Sur la figure 4.21, nous avons modélisé une application constituée de plusieurs tâches sur deux processeurs

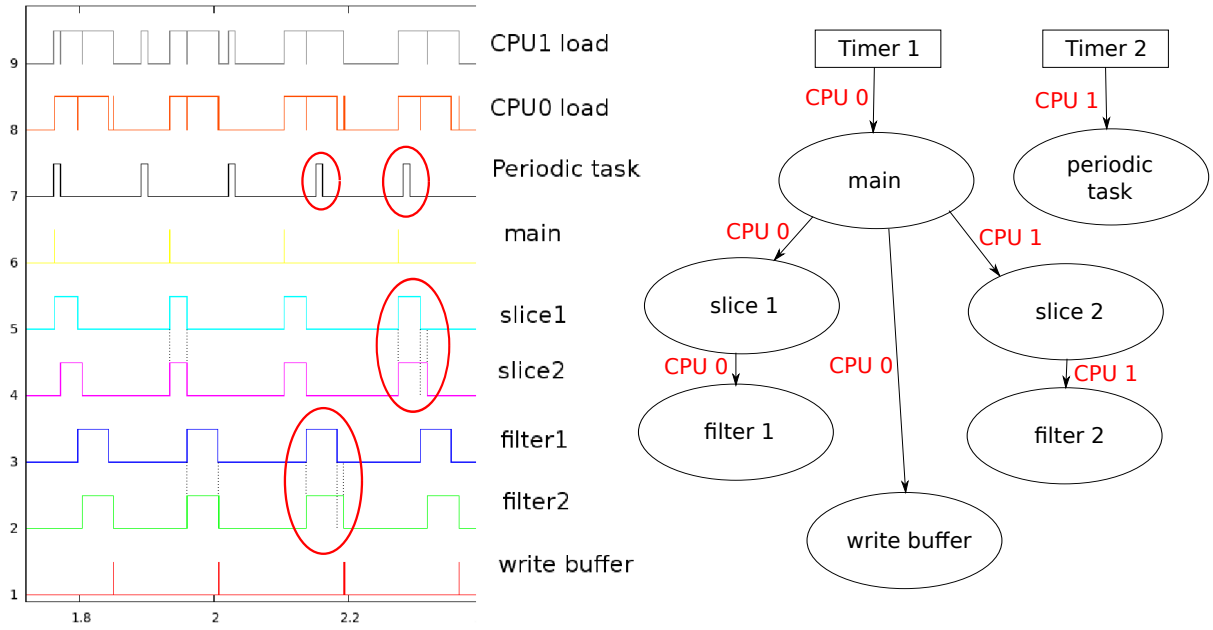

FIGURE 4.21: Parallélisme des tâches et préemptions.

différents. On observe bien sur le graphique de gauche, que deux tâches sont capables de s'exécuter en parallèle (par exemple : slice 1 et slice 2). De même, lorsque la tâche "Periodic task" préempte une tâche qui s'exécute sur son processeur (slice 2 ou filter 2), le temps d'exécution de cette dernière est augmenté par rapport à son temps d'exécution "normal". On est donc bien capable d'exécuter des applications parallèles et d'assigner (statiquement dans un premier temps) des tâches à certains processeurs.

Les opérations liées à l'ordonnanceur, comme les préemptions et les priorités, sont exécutées par le système d'exploitation de la machine hôte (typiquement un PC). En effet, en utilisant Posix, les définitions des parties propriétés d'ordonnancement sont standards. Ainsi, lorsque les tâches sont prêtes à être exécutées, le système d'exploitation (Linux) va alors les ordonnancer dynamiquement suivant leurs priorités et leur affinité de processeurs.

FORECAST est capable de fournir plusieurs traces d'exécutions. Tout d'abord, comme nous l'avons déjà vu plusieurs fois, il est possible de tracer l'exécution des différentes tâches de l'application qui nous intéressent. Ceci permet de visualiser le comportement fonctionnel de l'application, et de s'assurer que ce fonctionnement est bien celui recherché.
Ensuite, il est aussi possible de tracer l'activité de chaque processeur. Ceci permet à la fois de visualiser la charge des différents processeurs, mais aussi d'évaluer le niveau de parallélisme de l'application. En effet, si il apparaît que les processeurs sont rarement occupés ensemble, c'est que le parallélisme n'est pas satisfaisant et qu'il n'est peut être pas nécessaire d'avoir plusieurs processeurs.

D'autre part, FORECAST est aussi capable de tracer les accès aux différentes mémoires de la plate-forme. Étant donné que l'on connaît le nombre d'accès effectué par chaque tâche, dès qu'une tâche est déclenchée, on trace le nombre d'accès mémoire pendant la durée de la tâche. Ceci peut être très utile pour évaluer les mémoires les plus utilisées, si les caches sont correctement dimensionnés, ou encore si il y a des pics d'accès mémoire.
La figure 4.22 montre un exemple de trace des accès mémoire obtenu pour l'application vidéo décodeur H.264 décodant les vidéos à 8 images par seconde sur une plate-forme ARM Cortex-A8. Le graphique permet de visualiser les accès pour chaque mémoire, et ainsi d'analyser que la mémoire la plus sollicitée est le cache

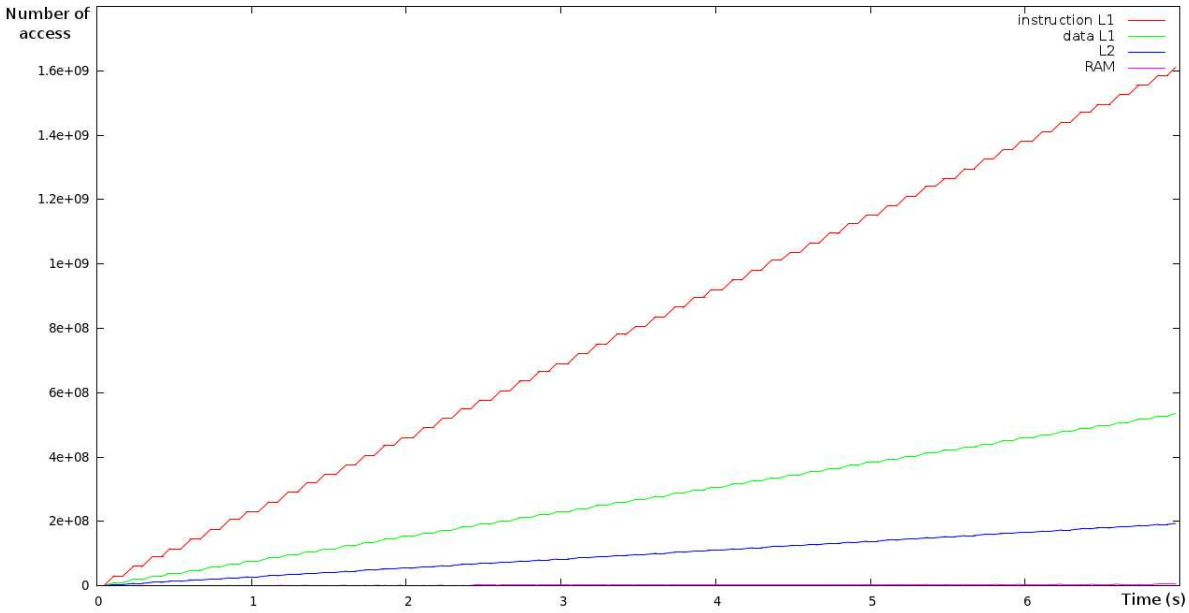

FIGURE 4.22: Graphique permettant de visualiser le nombre d'accès dans les différentes mémoires.

d'instructions de niveau un, puis le cache de données de niveau un. Viennent ensuite le cache de niveau deux, et enfin la mémoire principale (RAM) qui est très peu utilisée.

Ces traces sont aussi nécessaires afin d'effectuer des estimations de la consommation d'énergie. Comme on l'a vu dans la section 4.3.2, nous avons utilisé deux types de modèles de consommation : les estimations gros grain et les estimations à grain fin.
Pour les estimations haut niveau (gros grain), nous n'utilisons que la courbe des activités processeurs pour déterminer si le processeur est en activité ou si il ne fait rien. Ceci nous permet de calculer l'énergie dépensée par le système au bout d'un certain temps.

Pour les estimations grain fin, nous utilisons des modèles de consommation basés sur la consommation de fuite (nécessite le temps d'exécution) mais aussi la consommation d'un élément de base (par exemple un accès pour les mémoires ou l'exécution d'une instruction pour le processeur). Il est alors nécessaire d'utiliser des traces plus complexes que celles présentées précédemment.
Comme nous savons caractériser les accès aux différents éléments mémoire, il est alors possible de modéliser la consommation d'énergie de chaque mémoire. Il est aussi possible d'obtenir la consommation liée à l'exécution des instructions.

En conclusion, grâce à FORECAST, nous sommes donc capables d'obtenir un grand nombre d'informations portant à la fois sur l'ordonnancement, les performances ou la consommation électrique.

4.6 L'exploration de l'espace de conception

L'exploration de l'espace de conception permet de manière automatique d'évaluer plusieurs alternatives architecturales et de converger vers celle qui est la plus appropriée au problème posé. Comme on peut le voir sur la figure 4.23, l'exploration se base sur les traces de sortie. Après une analyse de ces traces, l'outil d'exploration va modifier la configuration du modèle matériel ainsi que le mapping des tâches afin d'améliorer les performances et la consommation du système.

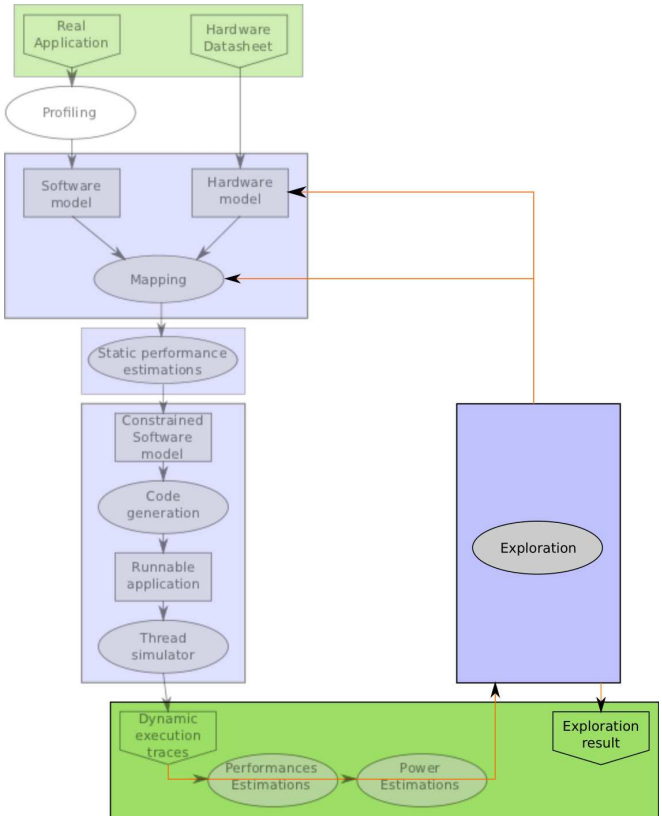

FIGURE 4.23: Intégration de la partie exploration dans le flot global.

4.6.1 Les objectifs

Un premier algorithme a été développé afin de trouver pour une plate-forme existante la meilleure solution respectant les contraintes temps-réel tout en minimisant la consommation d'énergie. Les objectifs appliqués au système peuvent être répartis en deux catégories : primordiaux (qu'il est nécessaire de respecter) et optionnels (qu'il est préférable de respecter, mais non obligatoire) :
Primordial :

- Le temps d'exécution de certaines tâches peut être contraint et devra alors rester inférieur à cette limite.
- La consommation doit être minimisée.

Optionnel :

- Le taux de charge des processeurs peut être contraint par une borne minimum et une borne maximum.

Afin de converger vers une solution optimisée, l'explorateur dispose de deux leviers sur lesquels il peut jouer :

- La fréquence des processeurs.
- La répartition des tâches sur les différents processeurs.

Comme nous utilisons l'explorateur pour une architecture matérielle existante il est uniquement possible de modifier la fréquence ou le nombre de processeurs actifs. En effet, il n'est pas possible par exemple de modifier la taille du cache ou le type de processeur (même si l'explorateur serait capable de le faire) car ces paramètres sont fixes pour cette plate-forme. De même, du coté logiciel, il est seulement possible de déplacer les tâches d'un processeur à l'autre.

4.6.2 La méthode d'exploration

Comme nous l'avons vu dans la section 2.2.4, plusieurs méthodes d'explorations sont possibles afin d'optimiser le temps d'exploration. Effectuant cette thèse dans un contexte industriel, nous avions déjà une grande connaissance des plates-formes ainsi que des possibilités associées. C'est pour ces raisons que nous préférons utiliser une exploration guidée afin de bénéficier de nos connaissances pour atteindre plus rapidement une solution étant Pareto-optimale.

La méthode utilisée lors de l'exploration est la suivante. Le nombre de tâches à assigner est analysé ainsi que le nombre de processeurs disponibles sur la plate-forme. Dans un premier temps, l'algorithme essaie au maximum de répartir les tâches sur les différents processeurs.
Le but est donc d'avoir le plus de processeurs possibles actifs mais avec la fréquence la plus basse afin d'obtenir une consommation la plus faible possible. Cette méthode se base sur des expérimentations menées avec un ou deux cœurs que l'on a vu précédemment dans le chapitre 3.

L'idée d'essayer de paralléliser un maximum en premier lieu afin d'utiliser une fréquence la plus faible possible est donc tout à fait justifiée. A partir de la, l'explorateur essaie de minimiser la fréquence tout en respectant les contraintes fixées.
Cette première étape va permettre de satisfaire les contraintes de charge des processeurs.

Une fois les charges de processeurs respectées, si des contraintes temps-réel ont été spécifiées, l'explorateur va alors jouer sur les fréquences des processeurs dont les tâches ne respectent pas leurs contraintes.
La figure 4.24 résume ces étapes dans un diagramme. Bien évidement, des conditions d'arrêts existent, par exemple si les fréquences des processeurs dépassent le maximum, ou si le temps d'exploration devient trop long. Nous avons estimé que si l'explorateur n'a pas fini au bout de 5 minutes (une itération durant 6 secondes), c'est que l'architecture matérielle ne peut répondre aux contraintes logicielles de l'application considérée.

Au niveau des contraintes (objectifs) à respecter il est possible dans un premier temps de n'utiliser que la charge processeur. En renseignant une charge processeur minimale et maximale, l'explorateur va exécuter des itérations afin de converger vers une solution où chaque processeur rentre dans les bornes, ou à défaut, restent sous la borne autorisée. La charge processeur est calculée en effectuant la moyenne entre le temps passé à exécuter du code et le temps passé à attendre au cours de l'exécution complète de l'application.
Si aucune solution respectant les contraintes n'est trouvée, l'explorateur recommencera du début en allouant les tâches de manière différente (tirage aléatoire pour la première itération puis répartition suivant les processeurs les moins chargés) sur les processeurs.
La figure 4.25 montre le résultat d'une exploration utilisant 4 processeurs et une application de décodage vidéo H.264. Chaque groupe d'histogrammes représente une itération de l'explorateur. Chaque histogramme à l'intérieur d'un groupement (de quatre) représente la charge d'un processeur. On remarque bien que la première étape consiste à répartir un maximum les tâches sur les différents processeurs. Ensuite, la fréquence est augmentée afin de réduire la charge processeur afin de respecter les contraintes. On observe aussi que le processeur 0 nécessite une plus haute fréquence pour tenir dans les bornes, ceci s'explique car c'est ce processeur qui exécute aussi la partie non-parallèle de l'application. Il a donc besoin de plus de puissance de

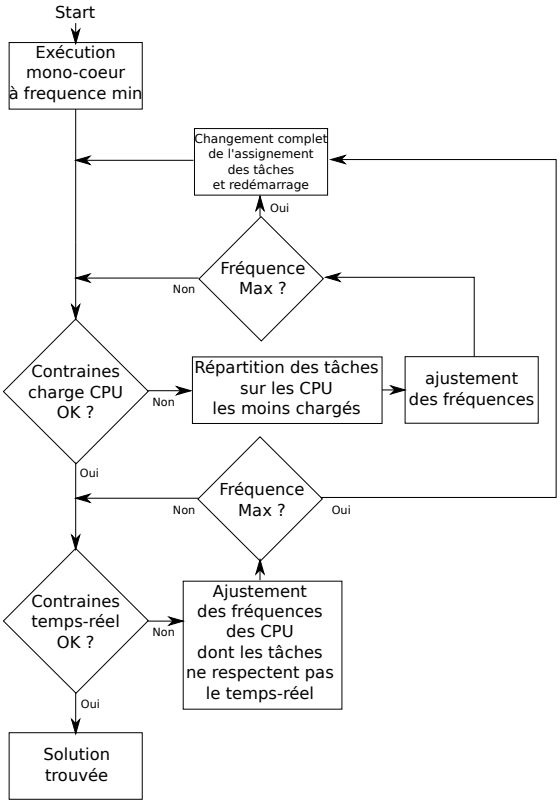

FIGURE 4.24: Algorithme utilisé lors de nos explorations d'architectures.

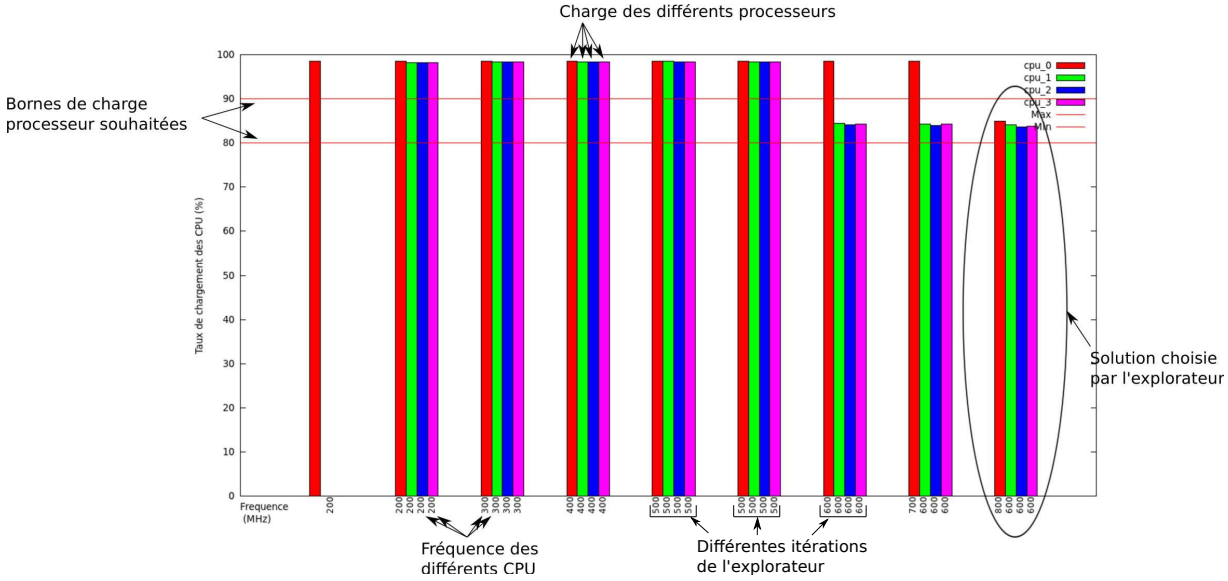

FIGURE 4.25: Exploration avec 4 processeurs et des bornes de charge processeur entre 80 et 90%.

calcul pour atteindre une charge inférieure à 90%.

Dans un deuxième temps, il est possible de rajouter des contraintes “temps réel” à certaines tâches. En effet, il est tout à fait possible que certaines tâches nécessitent de s'exécuter en très peu de temps, même si cela laisse le processeur inutilisé pendant longtemps. Une autre possibilité est la préemption de tâches.

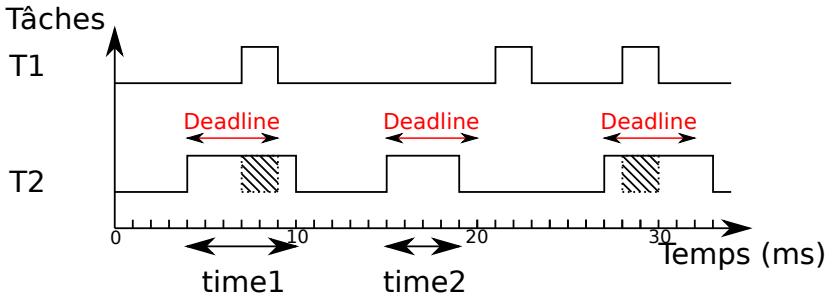

FIGURE 4.26: Exemple de temps d'exécution de deux tâches temps-réel.

En effet, comme on peut le voir dans la figure 4.26, il est possible que la tâche T2 se fasse préempter par la tache T1 ce qui va faire que sa *deadline* sera dépassée, mais la charge processeur n'a pas pour autant atteint 100%. Se baser uniquement sur la charge processeur n'est alors pas suffisant pour assurer le bon déroulement de l'application, il est aussi nécessaire de vérifier le temps d'exécution de chaque tâche temps-réel.

Un graphique est également disponible en sortie lorsque l'on utilise des contraintes temps-réel sur les tâches. Ce dernier permet de visualiser le temps d'exécution des différentes tâches contraintes ainsi que leur temps maximum d'exécution autorisé.

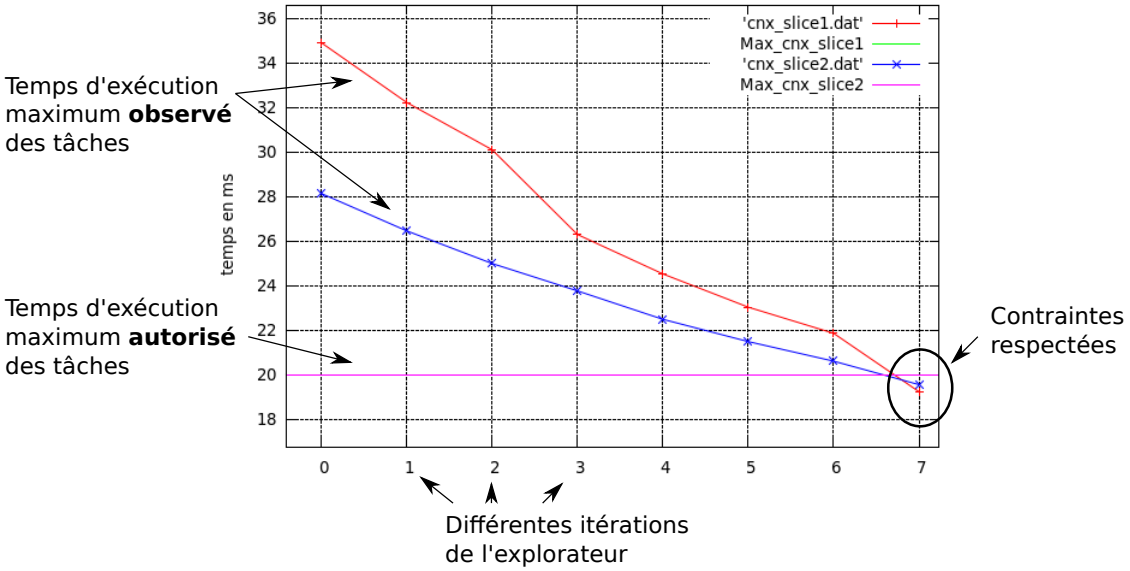

FIGURE 4.27: Temps d'exécution des deux tâches contraintes en temps.

La figure 4.27 montre le temps d'exécution au cours du temps de deux tâches contraintes en temps (20ms maximum d'exécution). On observe bien que le temps d'exécution des deux tâches réduit de plus en plus jusqu'à passer sous la contrainte de 20ms.

L'étape d'analyse de la consommation résultante n'a pas été implémentée par manque de temps. Cependant, les décisions qui ont été prises pour l'algorithme d'exploration, comme la répartition maximale sur

les cœurs afin d'utiliser la fréquence minimale, devrait permettre de trouver des solutions minimisant la consommation d'énergie du système.

Comme on peut le voir, nous avons utilisé une méthode d'exploration basée sur une connaissance à priori des solutions les plus appropriées. En effet, faire une recherche exhaustive de toutes les solutions afin de trouver la meilleure aurait été beaucoup trop long et une recherche orientée est plus efficace. Prenons par exemple l'expérimentation précédente possédant 4 processeurs avec chacun 20 fréquences disponibles (de 200 à 1200MHz par saut de 50MHz) et 10 tâches pouvant être réparties sur les différents processeurs. Cela signifie environ $1.67 \cdot 10^{11}$ possibilités ce qui n'est pas envisageable avec une itération de 6 secondes.

La méthodologie étant totalement décrite, nous allons maintenant décrire les différents résultats obtenus, que ce soit pour des plates-formes mono-processeur, ou des plates-formes multi-processeurs. Nous aborderons dans un premier temps les différentes plates-formes matérielles et applications de test, puis les comparaisons des estimations avec les plates-formes réelles, le projet COMCAS et le projet Open-PEOPLE.

Chapitre 5

Resultats et évaluations

Ce chapitre va permettre d'évaluer la pertinence de notre approche et des outils développés en comparant nos estimations avec des résultats de mesures obtenus sur plates-formes réelles, ainsi que des estimations effectuées avec d'autres méthodologies. Pour cela, plusieurs applications ont été exécutées sur différentes plates-formes. Nous présentons dans un premier temps les différentes plates-formes utilisées, puis nous aborderons les différentes applications étudiées. Enfin nous étudierons les résultats obtenus au cours de cette thèse.

5.1 Description des plates-formes et des cas d'études

5.1.1 Les plates-formes matérielles

Atmel AT91SAM9263

Dans le cadre d'une étude menée pour Thales Communications and Security, visant à implémenter un décodeur vidéo sur un de leur produit, nous avons utilisé un composant Atmel AT91SAM9263. Ce composant contient un processeur ARM926e-js cadencé de 125 à 200MHz. Il possède aussi des mémoires cache de niveau un de 8KBytes chacune.
Ce processeur a été grandement utilisé dans l'industrie et le monde de l'embarqué de part sa faible consommation et sa puissance de calcul honorable. Il offre aussi la possibilité d'utiliser des instructions DSP ainsi qu'un accélérateur pour le langage JAVA.
Il n'existe cependant qu'en version mono-processeur, et possède un ratio de DMIPS / MHz de 1.1.

Texas Instruments OMAP3530

La première architecture cible qui a été choisie durant la thèse est la plate-forme OMAP3530 de Texas Instruments [106]. Ce processeur possède un GPP ARM Cortex-A8 et un DSP C64x+. Pour le cas d'une modélisation monoprocesseur, seul le processeur ARM a été utilisé.

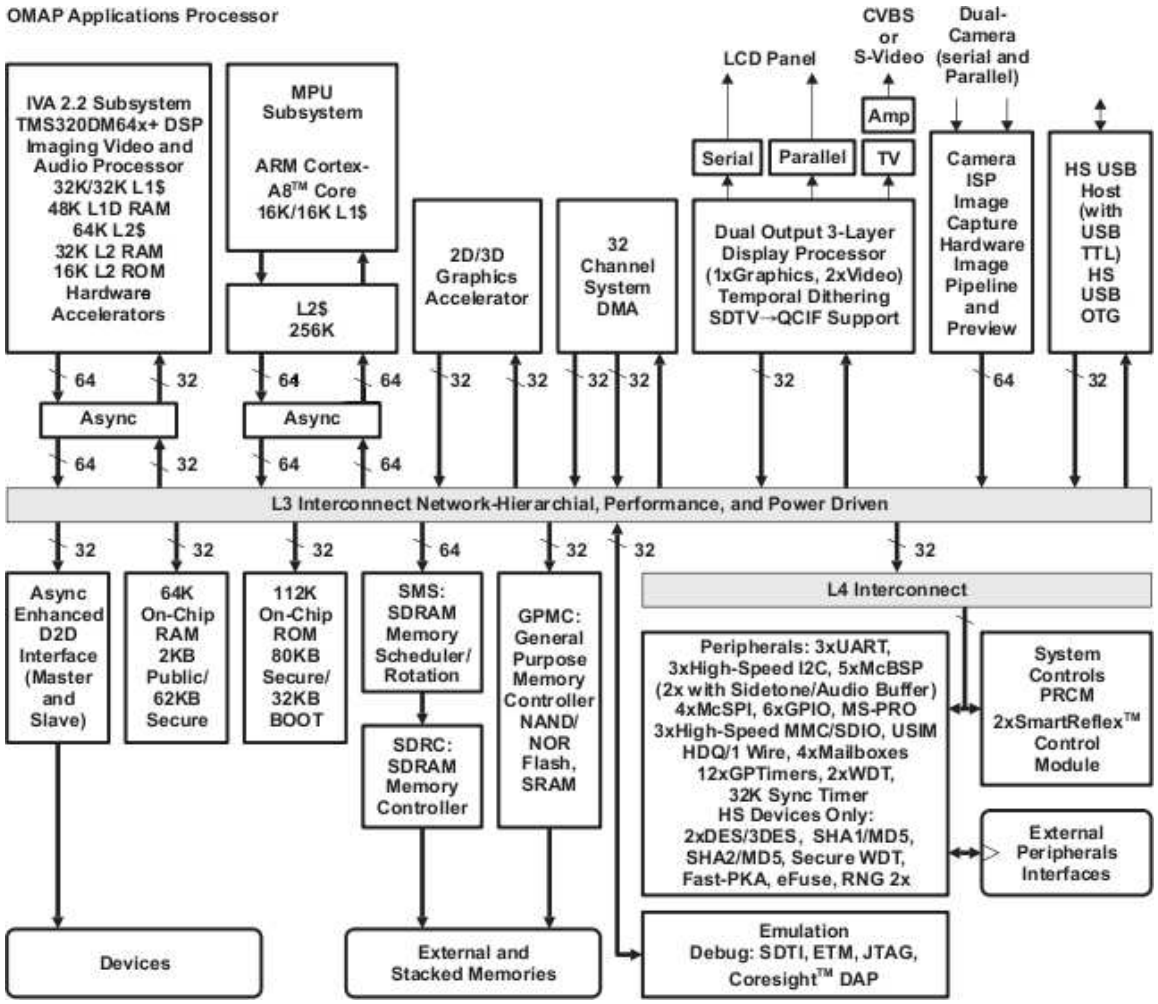

FIGURE 5.1: Schéma bloc de l'OMAP3530

La figure 5.1 montre le schéma bloc du chip OMAP3530. Cette plate-forme, récente et disponible dès le début de la thèse, a également été choisie pour sa simplicité de mise en oeuvre et sa configurabilité (quoique partielle). En effet, il possède une large gamme de fréquences processeur allant de 100 à 800MHz, une fréquence de bus mémoire modifiable (83 ou 166MHz), et des caches désactivables. Le processeur possède 16KBytes de cache L1 d'instructions et 16KBytes de cache L1 de données ainsi qu'un cache L2 de 256Kbytes. L'ARM Cortex-A8 possède un ratio DMIPS / MHz de 2.0. La technologie de gravure utilisée pour ce composant est de 65 nm.
Enfin, la carte d'évaluation permet de mesurer la puissance consommée par le coeur de processeur ainsi que par la RAM. C'est donc une très bonne carte adaptée à l'évaluation et la mesure des performances et de la consommation d'énergie.

Freescale I.MX31

La série i.MX3x est une famille de processeurs basée sur l'architecture ARM11 (ARM1136JF-S), conçue avec un processus CMOS 90nm. Le processeur possède 16KBytes de cache L1 d'instructions et 16KBytes de cache L1 de données ainsi qu'un cache L2 de 128Kbytes. Etant un processeur plus ancien, l'ARM1136 possède un ratio DMIPS / MHz de 1.18.

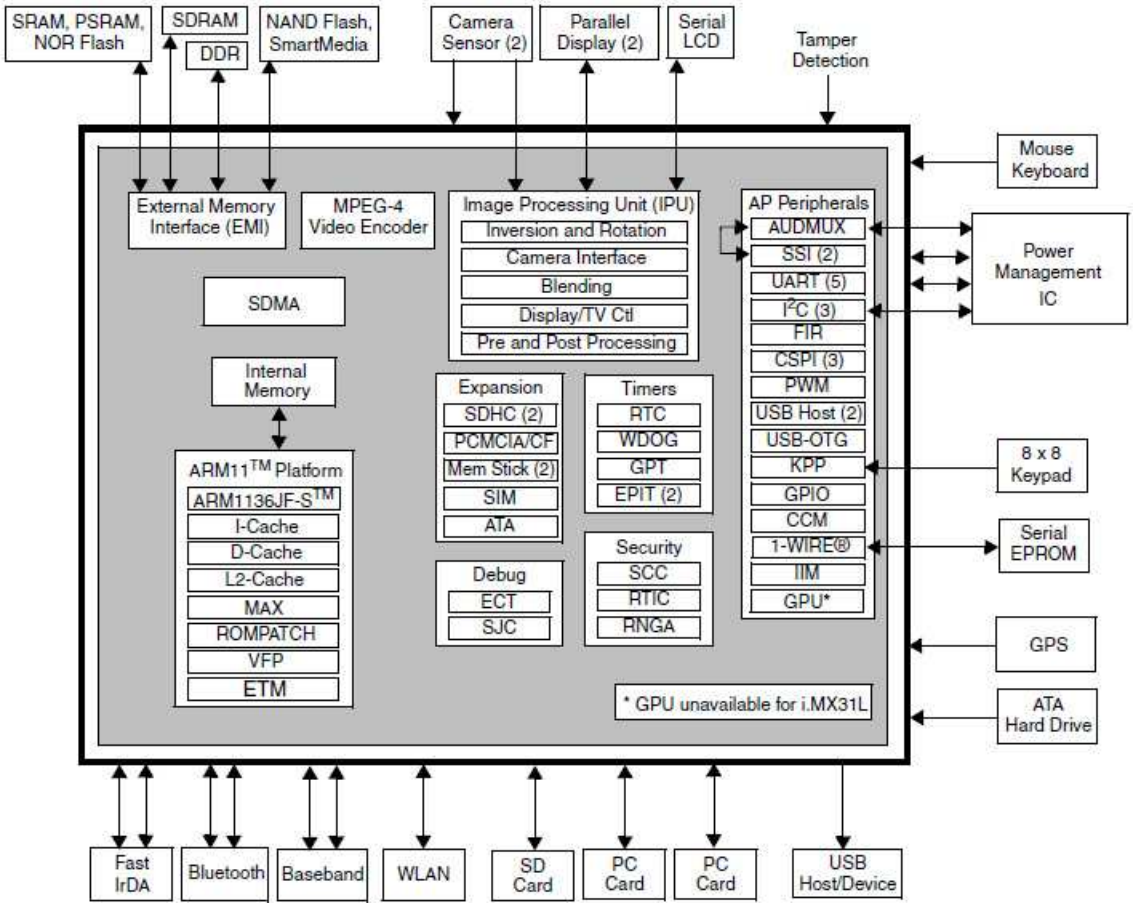

FIGURE 5.2: Schéma bloc de l'i.MX31

La figure 5.2 montre le schéma bloc du composant i.MX 31. Pour notre étude, nous n'utilisons que la partie processeur ARM. Ce processeur permet également une utilisation en multicoeur.

Freescale QorIQ P2020

La série P2 des plate-formes QorIQ, qui incluent les processeurs P2020 et P2010, délivre une bonne performance par watt pour une large variété d'applications comme la télécommunication, le militaire ou l'industrie. Les séries donnent accès à un simple ou double coeur avec une horloge allant jusqu'à 1.2 GHz avec une technologie 45 nm basse consommation.

Les séries QorIQ P2 consistent en des produits (dual ou simple coeur) dont le boîtier est compatible avec les produits QorIQ P1, ce qui offre cinq solutions interchangeables. Ces solutions vont du simple coeur à 533MHz (P1011) au dual coeur à 1.2GHz (P2020).

L'architecture haute performance des plates-formes P2020 (dual) ou P2010 (mono) est composée de coeurs e500 allant jusque 1.2 GHz, de 32KBytes de cache de niveau un et de 512KBytes de cache de niveau deux. Il possède aussi un contrôleur de mémoire DDR2/DDR3.

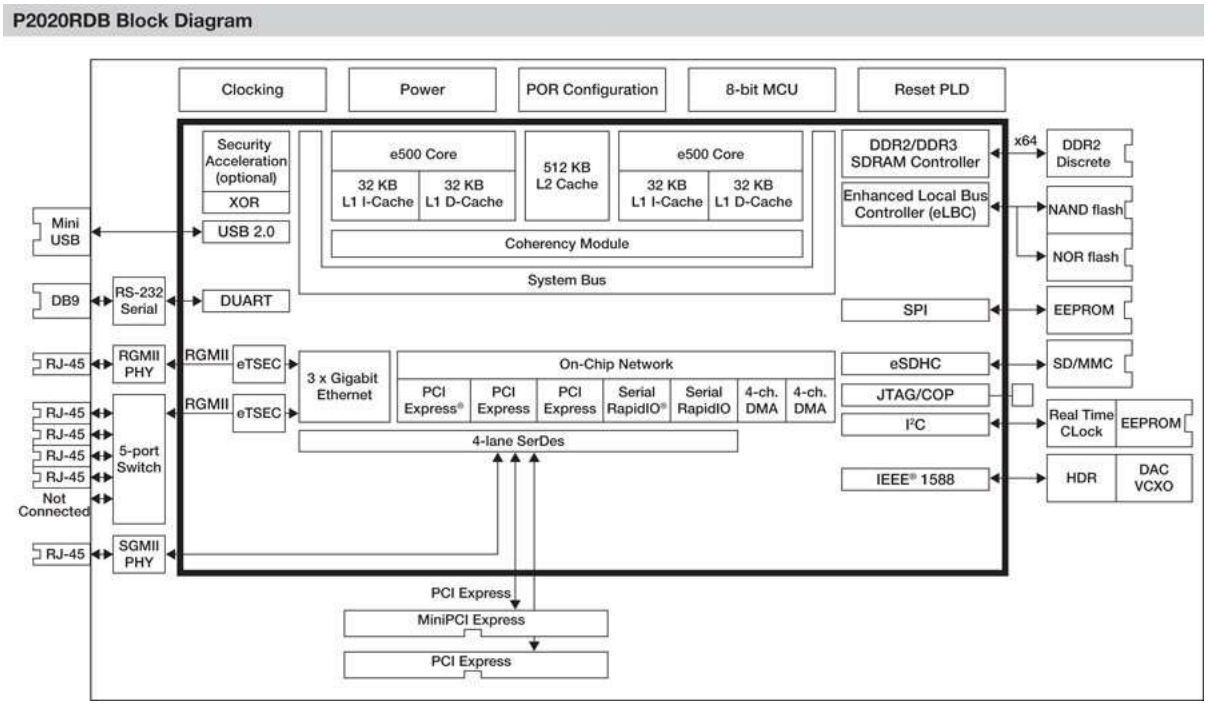

FIGURE 5.3: Schéma bloc du QorIQ P2020

Texas Instruments OMAP44xx

La série des OMAP44xx [107] utilise le processeur double-coeur ARM Cortex-A9. Ce processeur symétrique permet d'obtenir un ratio DMIPS / MHz de 2.5 par coeur. Ce composant a été conçu pour être utilisé dans les smartphones et tablettes tactiles du marché.
Plusieurs gammes ont été développées, allant de l'OMAP4430 tournant à 1GHz maximum, jusqu'à l'OMAP4470 tournant jusque 1.8GHz. L'amélioration de performance théorique par rapport à l'ARM Cortex-A8 est d'environ 150%. La finesse de gravure a aussi été améliorée et la technologie utilisée est du CMOS 45 nm. Chaque

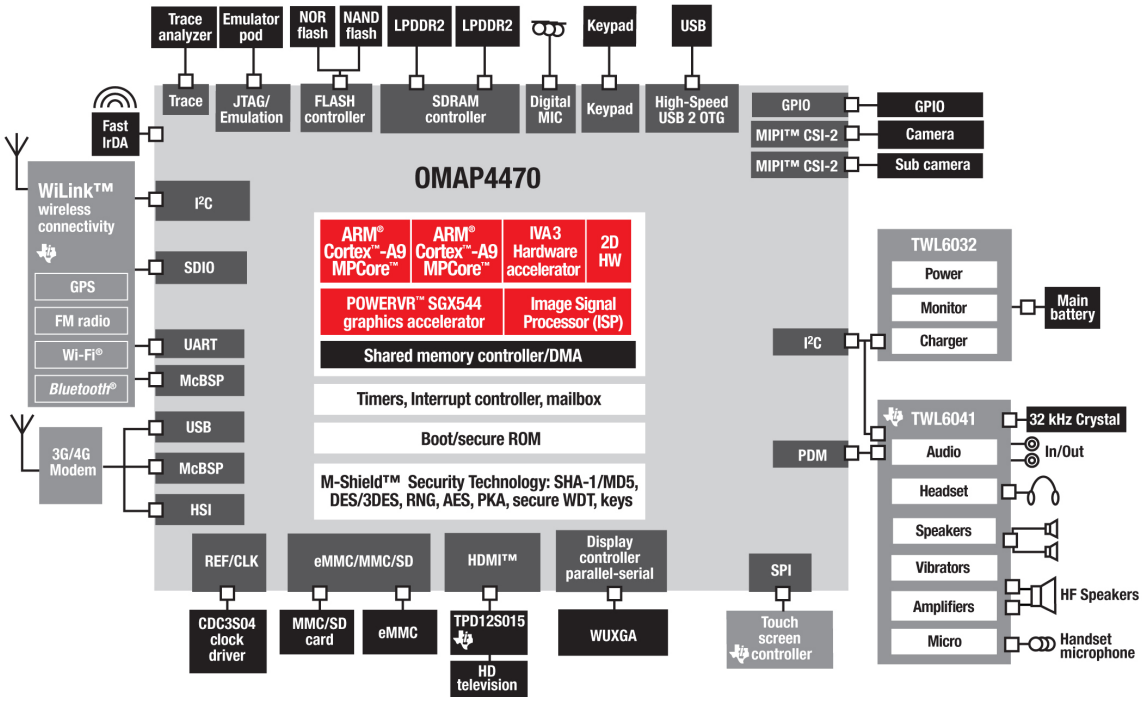

FIGURE 5.4: Schéma bloc de l'OMAP_44x

processeur possède ses caches de niveau un de 32KBytes chacun, et un cache de niveau deux commun de 1024KBytes.

Freescale I.MX6

La série des i.MX6x est la dernière du portfolio Freescale. Elle est basée sur le processeur ARM Cortex-A9 en mono, dual ou quadri-coeur. La technologie utilisée est un processus CMOS 40 nm. Les fréquences d'utilisation de l'i.MX6 vont de 400 à 1.2GHz. De même que l'OMAP44xx, l'i.MX6 possède des caches de niveau un de 32KBytes et un cache de niveau deux de 1024KBytes.

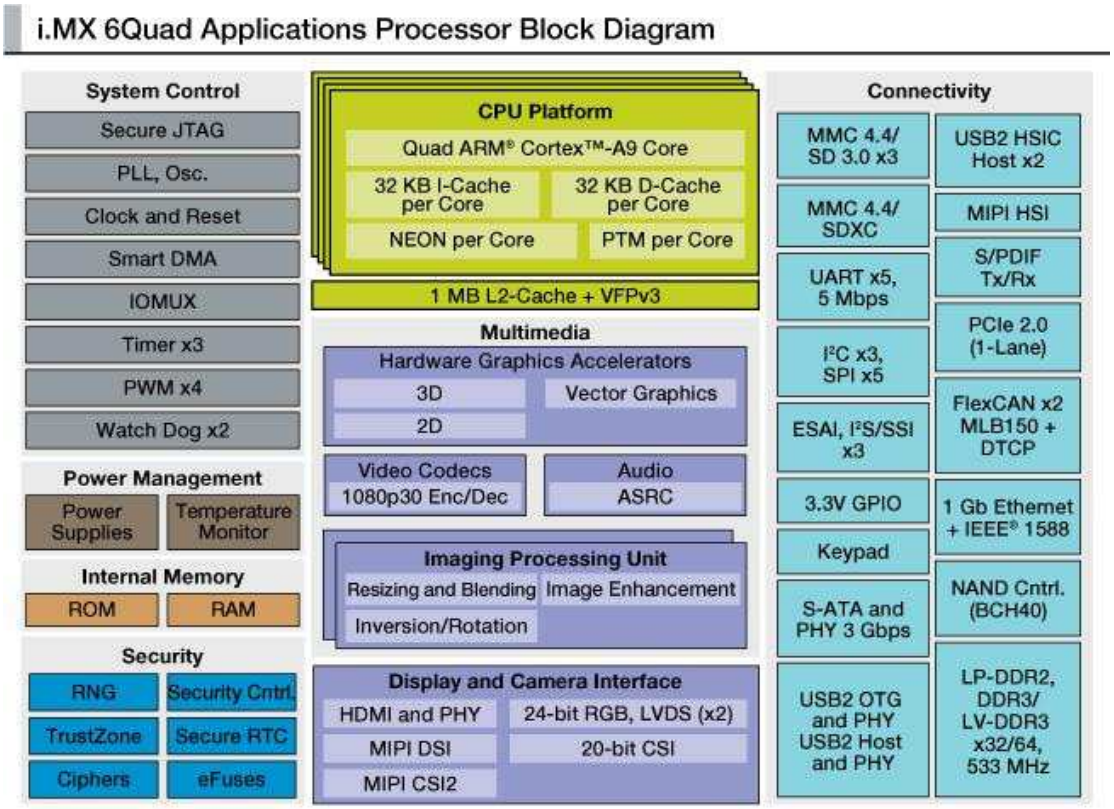

FIGURE 5.5: Schéma bloc de l'i.MX6

Récapitulatif

Cinq plates-formes matérielles ont été utilisées pour valider les différents modèles d'estimation de performance et de consommation. Chaque plate-forme ayant ses propres paramètres suivant la date de sortie et le processeur présent.

Nom	Fréquences (MHz)	DMIPS / MHz	Profondeur pipeline	Nombre de coeurs	Taille cache L1	Taille cache L2
AT91SAM9263	125 - 200	1.1	5	1	2 x 8 KB	0 KB
i.MX 31	532 - 665	1.18	8	1 - 4	2 x 16 KB	128 KB
OMAP3530	100 - 800	2.0	13	1	2 x 16 KB	256 KB
OMAP44xx	300 - 1800	2.5	8	1 - 2	2 x 32 KB	1024 KB
i.MX 6	400 - 1200	2.5	8	1 - 4	2 x 32 KB	1024 KB
QorIQ	533 - 1200	2.5	7	1 - 2	2 x 32 KB	512 KB

TABLE 5.1: Récapitulatif des différentes plates-formes et de leurs paramètres.

Le tableau 5.1 propose une synthèse des différents paramètres pour les plates-formes matérielles considérées.

5.1.2 Les applications test

Pour valider nos modèles et nos estimations, plusieurs applications test (*testcases*) ont été utilisées durant cette thèse.

Décodeur vidéo H.264

Tout d'abord, l'application que nous avons le plus utilisée est un décodeur H.264 développé dans le laboratoire de Thales. Cette application permet de décoder des vidéo H.264 encodées avec différents formats (QCIF, CIF, 720p) tout en lui spécifiant une contrainte en termes de nombre d'images par seconde à décoder. Ceci permet d'avoir une information concrète et visuelle de ce qui est estimé (l'oeil humain voit une image non saccadée à partir de 25 images par secondes).

Cette application présente un cas intéressant de parallélisation pour des architectures multi-processeurs. En effet, dans l'algorithme H.264, chaque slice (ou section) d'une image est indépendant et peut être calculée séparément (voir figure 5.6).

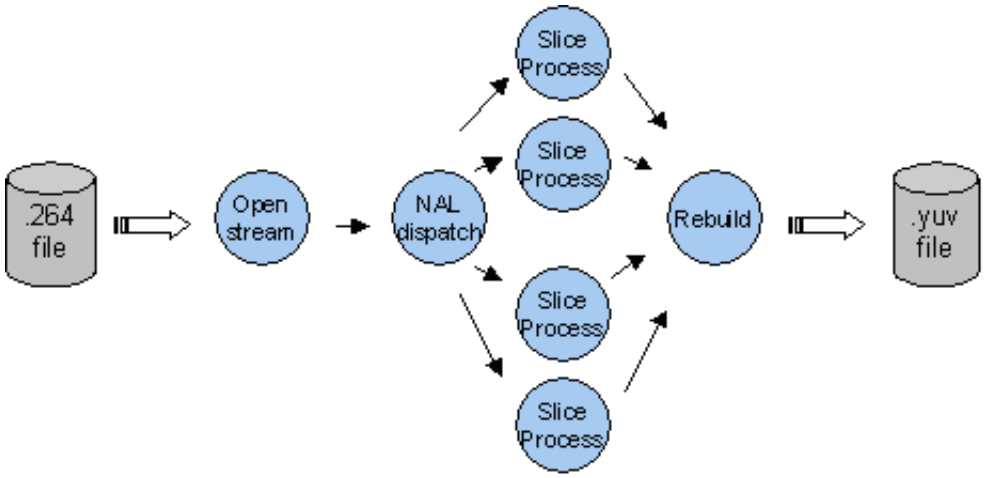

FIGURE 5.6: Version du décodeur H.264 parallélisé (en 4 slices).

La parallélisation sur les slices permet une répartition des traitements égale sur les unités de calcul. L'architecture logicielle qui a été utilisée est présentée sur la figure 5.6. Cette version correspond à un flux contenant 4 slices par image. Il est possible d'augmenter le nombre de slices dans une image ce qui augmentera d'autant le nombre de tâches. L'inconvénient majeur de cette méthode est que plus le nombre de slices est important et moins la qualité de l'image sera bonne.

Voici les différentes tâches présentes dans l'application H.264 :

Open stream : Cette tâche est responsable de l'acquisition des données du système. Elle lit le flux H.264 et le sauvegarde dans un buffer circulaire qui est envoyé à la tâche suivante. Cette tâche exécute aussi le décodage des deux premières NALs (Network Abstraction Layer) qui contiennent les paramètres de la vidéo (nombre d'images, taille d'une image, nombre de slices par image...).

Nal dispatch : Cette tâche est responsable de l'extraction de la NAL. Elle lit le flux et décode les marqueurs de démarrage et de fin. Une fois que la NAL est détectée (représentant une slice), elle est envoyée à une des taches "Slice process" au travers d'un buffer.

Slice process : Cette tâche effectue le décodage H.264 complet sur la slice qui a été reçue. Elle envoie ensuite sa section d'image à la prochaine tâche.

Rebuild : Cette tâche récupère les différentes slices de l'image. Elle reconstruit ensuite l'image décodée complète à partir des différentes parties puis la stocke dans un buffer de sortie.

De plus, une tâche de filtrage optionnelle peut être appliquée en fin de traitement afin de réduire les artefacts caractéristiques du codage/décodage avec la transformation en bloc.

Application audio : G.726

Introduction Un signal de parole est typiquement numérisé sur 8 bits avec une fréquence d'échantillonnage de 8kHz, soit un débit de 64 kbit/s. A la sortie d'un convertisseur analogique-numérique, le signal est en effet encodé au format PCM (Pulse Code Modulation) ou MIC en français (Modulation par Impulsion et Codage), en utilisant la loi A ou la loi μ. Les relations entre les signaux à fréquences vocales et les lois de codage et de décodage PCM sont standardisées dans la Recommandation G.711.

Les communications de données ou de paroles demandent de plus en plus de capacité de transmission sans une dégradation de la qualité du signal transmis. Une solution pour cela est d'utiliser des algorithmes de compression. Pour le signal de parole de nombreux algorithmes ont été proposés et sont très utilisés par exemple dans les communications mobiles GSM ou les répondeurs téléphoniques. L'algorithme de compression G726, standardisé par l'ITU (International Transmission Union) permet de convertir un signal de parole encodé à 64 kbit/s en un signal à 16, 24, 32 ou 40 kbit/s [61], et vice-versa. L'algorithme de compression G726 permet donc de compresser un signal au format PCM en un signal au format ADPCM (Adaptative Differential Pulse Code Modulation) avec un taux de compression de 2, 3, 4 ou 5. L'ADPCM est aussi appelé Modulation par Impulsions et Codage Différentiel Adaptatif (MICDA) en français. Une différence notable entre la norme G726 et bien d'autres algorithmes de compression (G728, Enhanced Full-Rate, AMR...), est que G726 effectue une conversion échantillon par échantillon. Dans beaucoup d'autres vocoders, la compression/décompression s'effectue souvent sur des trames d'échantillons représentant par exemple 20ms de signal de parole (soit 160 échantillons).

La norme G726 est une solution présentant une complexité algorithmique réduite (par rapport à d'autres solutions). Le bon compromis entre taux de compression, qualité et complexité, explique que cette norme est toujours relativement utilisée sur de petits systèmes nécessitant une compression d'un signal de parole (typiquement des répondeurs téléphoniques par exemple).

L'encodeur G.726 La fonction de l'encodeur est de recevoir un signal PCM en loi A ou μ et de le convertir en un signal de sortie ADPCM à 40, 32, 24 ou 16 kbit/s.

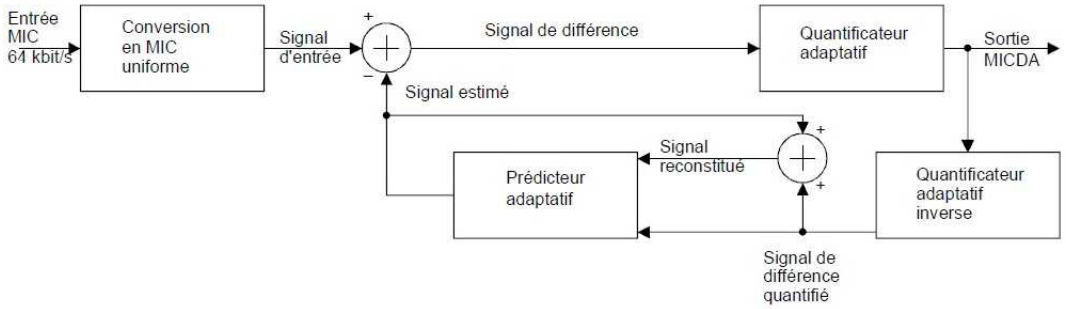

FIGURE 5.7: Schéma de principe simplifié du codeur ADPCM (ou MICDA).

Le signal ADPCM de sortie (MICDA sur la figure) est un signal de différence obtenu en soustrayant du signal d'entrée (préalablement converti de la loi A ou la loi μ en MIC uniforme) une valeur estimée de ce signal. Un quantificateur adaptatif à 31, 15, 7 ou 4 niveaux permet d'attribuer, respectivement, cinq, quatre, trois ou deux éléments binaires à la valeur de ce signal de différence en vue de sa transmission jusqu'au décodeur. Un quantificateur inverse produit le signal de différence quantifié à partir de ces mêmes cinq, quatre, trois ou deux éléments binaires. La valeur estimée du signal est ajoutée au signal de différence quantifié pour fournir une version reconstituée du signal d'entrée. Un prédicteur adaptatif, qui agit tant sur le signal reconstitué que sur le signal de différence quantifié, fournit une estimation du signal d'entrée, ce qui ferme la boucle de contre-réaction.

Le décodeur G.726 La fonction du décodeur est de recevoir un signal d'entrée ADPCM à 40, 32, 24 ou 16 kbit/s et de le convertir en un signal PCM en loi A ou μ à 64 kbit/s.

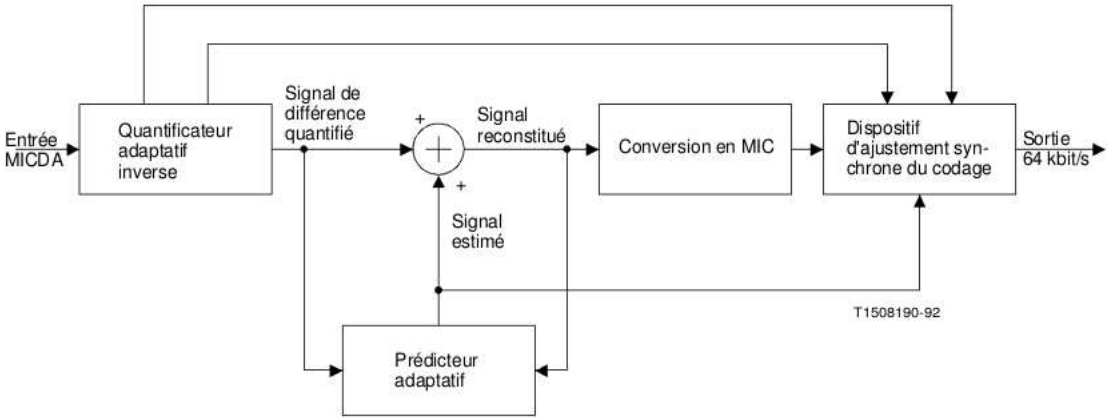

FIGURE 5.8: Schéma de principe simplifié du décodeur ADPCM (ou MICDA).

Le décodeur comporte une structure identique à celle de la boucle de contre-réaction du codeur ainsi qu'un dispositif de conversion du code MIC uniforme en loi A ou en loi μ et un dispositif d'ajustement synchrone du codage. Le dispositif d'ajustement synchrone du codage empêche l'accumulation de la distorsion que l'on pourrait observer avec des codages synchrones en cascade (connexions numériques MICDA-MIC MICDA, etc.) dans certaines conditions. On parvient à ce résultat en ajustant le code de sortie MIC de façon à tenter d'éliminer la distorsion de quantification dans l'étage de codage MICDA suivant.

Encodeur d'image JPEG

La norme JPEG est une norme qui définit le format et l'algorithme de codage et décodage pour une représentation numérique compressée d'une image fixe. L'application utilisée provient de la suite de benchmarks "MiBench". Pour nos essais, nous n'avons utilisé que l'encodeur. Ce format d'image étant un standard, il est intéressant d'utiliser une application permettant le codage dans cette norme afin d'utiliser une application connue de la plupart des gens.
La figure 5.9 montre les différentes étapes afin d'effectuer une compression et une décompression JPEG. Nous pouvons voir que le codage se divise en plusieurs blocs différents et nous verrons par la suite qu'une différence de consommation se fera sentir suivant le bloc en cours d'utilisation.

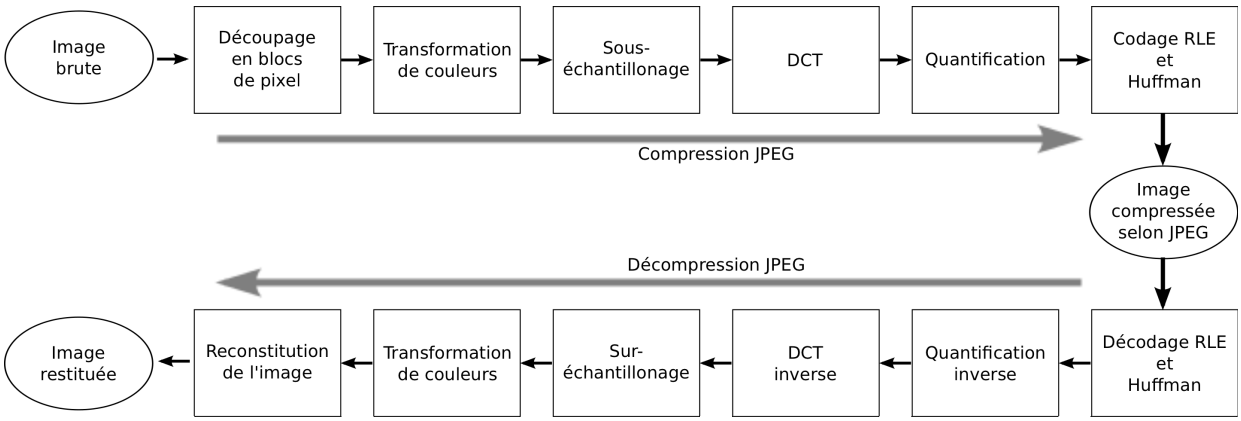

FIGURE 5.9: Schéma bloc de la compression et décompression JPEG.

Suite de benchmarks : nBench

Nous utilisons aussi une partie des applications présentes dans nBench pour valider notre approche. Les applications utilisées sont : "Numeric sort", "String sort", "FP Emulation", "Bitfield" et "Huffman". Ceci nous permet d'avoir un type d'application diversifié sans avoir à créer de nouvelles applications. Voici la description des benchmarks utilisés :

Numeric sort déplace des données de 32 bits et montre à la fois la performance des mouvements mémoire et la performance non-séquentielle des caches.

String sort teste la performance des mouvements mémoire. Similaire au test Numeric sort mais il déplace des données de 8 bits.

Bitfield exécute un grand nombre d'opérations basées sur les bits comme effacer des bits consécutifs, écrire une série de bits etc.

Emulated floating point teste des opérations flottantes en l'absence d'un coprocesseur et montre une bonne mesure de la performance globale du processeur.

Huffman compression exécute une combinaison d'opérations sur les caractères nécessaires à la compression et décompression.

Ceci nous a aussi permet de comparer nos résultats au projet Européen COMCAS qui utilise ces applications pour estimer les erreurs de leur plate-forme d'estimation.

Application radio : DRHD

L'application appelée DRHD est un logiciel de simulation de traitement du signal développée par Thales Communications and Security et utilisée comme application de test dans le projet Open-PEOPLE.
Il s'agit de faire fonctionner un logiciel représentatif des traitements numériques du signal et des données tels qu'ils existent dans toutes les radiocommunications modernes. Pour évaluer la consommation d'énergie de chaque bloc (par rapport à un processeur inactif), chaque bloc pourra être activé ou non, individuellement, par l'utilisateur. Un bloc non activé ignore ses données d'entrée et fournit des données de sortie fictives mais d'un volume cohérent avec ce qu'il produit en fonctionnement, soit une table de constantes de taille adaptée.

Ce logiciel est une application type, issue de l'expertise de Thales et assure l'émission de communications radio en bande HF élargie (Fréquences allant jusqu'aux alentours de 80MHz). Il s'agit plus précisément d'une couche PHY (i.e. Physique).

Le logiciel peut être paramétré, et comporte 3 possibilités de chiffrement et 5 possibilités d'adaptation au canal radio.

Lors du fonctionnement normal de ce logiciel, le choix des algorithmes de chiffrement est une décision opérationnelle, et il est possible de ne pas chiffrer du tout.

Scénarios envisagés Le but de ces scénarios prédéfinis par Thales est de paramétrer l'exécutable de manière à obtenir 2 scénarios opérationnels et réalistes. Les hypothèses qui ont été faîtes pour réaliser ces scénarios sont :

- Couche PHY OFDM pour de l'UHF
- Largeur de bande instantanée : 5 MHz
- Durée d'un palier : 1ms

Scenario 1 (fichier scenario1.sh) :

- Réseau centralisé (comprendre qu'il existe une station de base)
- Transmission de data haut débit
- Transmission de 4 images à 4 utilisateurs différents
- Downlink, c'est à dire que c'est la station de base qui émet vers les 4 terminaux
- Pas de sécurité requise

Scenario 2 (fichier scenario2.sh) :

- Réseau Ad-Hoc
- Transmission de voix
- Transmission d'urgence et très sécurisée
- 4 utilisateurs placés de la manière suivante (comprendre visibilité) dans le réseau :
  user $1 <-> $ user $2 <-> $ user $3 <-> $ user 4

L'intérêt des deux scénarios est que le scénario 2 sollicite plus le processeur (avec des calculs intensif lié au cryptage) alors que le scénario 1 sollicite d'avantage les mémoires.

5.2 L'estimation appliquée à des plates-formes mono-processeur

Pour valider et comparer nos modèles avec des plates-formes réelles, nous avons tout d'abord utilisé des processeurs mono-coeur. Les plates-formes utilisées pour cela sont donc : Freescale i.MX31 et Texas Instruments OMAP3530. Certaines plates-formes multi-coeur ont aussi été utilisées pour nos comparaisons mais en n'utilisant qu'un seul processeur (Texas Instruments OMAP4430). Un cas réel d'utilisation pour un produit Thales sera aussi abordé. Dans un deuxième temps, nous comparerons nos résultats à des projets collaboratifs en cours comme le projet Européen COMCAS et le projet ANR Open-PEOPLE.

5.2.1 Comparaison avec les plates-formes réelles

L'étape la plus importante dans la validation des résultats est de se comparer aux plates-formes réelles. Le but de la méthodologie étant d'estimer rapidement les performances et la consommation d'une application s'exécutant sur une plate-forme réelle, des comparaisons régulières ont été effectuées tout au long de la thèse. Rappelons qu'ici, l'objectif est d'obtenir une erreur d'estimation inférieur à 20% étant donné les modèles très haut-niveau utilisés et le peu d'informations requises.

Une bonne estimation de la performance étant nécessaire afin d'estimer précisément la consommation d'énergie, nous nous attarderons dans un premier temps sur l'estimation de performance. Nous étudierons tout d'abord les modèles préliminaires que nous avons utilisés, puis les modèles finaux.

Estimations préliminaires

Dans un premier temps, nous avons choisi de faire les premières expérimentations avec un sous-ensemble des paramètres évoqués précédemment afin de vérifier l'importance de ces paramètres. Seuls les paramètres les plus critiques ont été utilisés :

- Pour le matériel : La fréquence CPU, le DMIPS/MHz et la taille des caches (en data)
- Pour l'application : Le nombre d'instructions et le nombre de lectures/écritures par tâche

Nous n'utilisons donc pas ici le nombre de cache-miss en instruction ainsi que les erreurs de prédictions de branchement.

Nous décrivons dans un premier temps un exemple d'estimation d'une tâche de l'application vidéo décodeur H.264. La figure 5.10 montre le découpage en tâches de l'application.
Comme on peut le voir, chaque tâche est annotée avec les informations issues du profiling effectué sur l'application. Une fois cette étape accomplie, l'estimateur de performances entre en jeu et détermine le temps d'exécution de chaque tâche.
Par exemple, calculons le temps d'exécution de la tâche "slice1" s'exécutant sur un OMAP3530 à 600MHz. La première étape consiste à calculer le temps de pénalité d'un cache miss de donnée en L1 et en L2 :
$l1\_pen = rw\_ms \cdot l1\_nbcycle$
$l1\_pen = 0.000001667 \cdot 10$
$l1\_pen = 0.00001667ms$

$l2\_pen = rw\_ms \cdot l2\_nbcycle$
$l2\_pen = 0.000001667 \cdot 100$
$l2\_pen = 0.0001667ms$

Une fois ces calculs effectués, nous pouvons calculer le temps nécessaire à la lecture/écriture des données en mémoire :
$MEM\_elapsed\_time = [(nb\_r + nb\_w)] \cdot (rw\_ms + \sum_{i=1}^{\infty} li\_miss\_rate \cdot li\_pen)$
$MEM\_elapsed\_time = [(nb\_r + nb\_w)] \cdot (rw\_ms + l1\_miss\_rate \cdot l1\_pen + l2\_miss\_rate \cdot l2\_pen)$
$MEM\_elapsed\_time = [(5063044 + 2462933)] \cdot (0.000001667 + \frac{0.1}{100} \cdot 0.00001667 + \frac{0.05}{100} \cdot 0.0001667)$

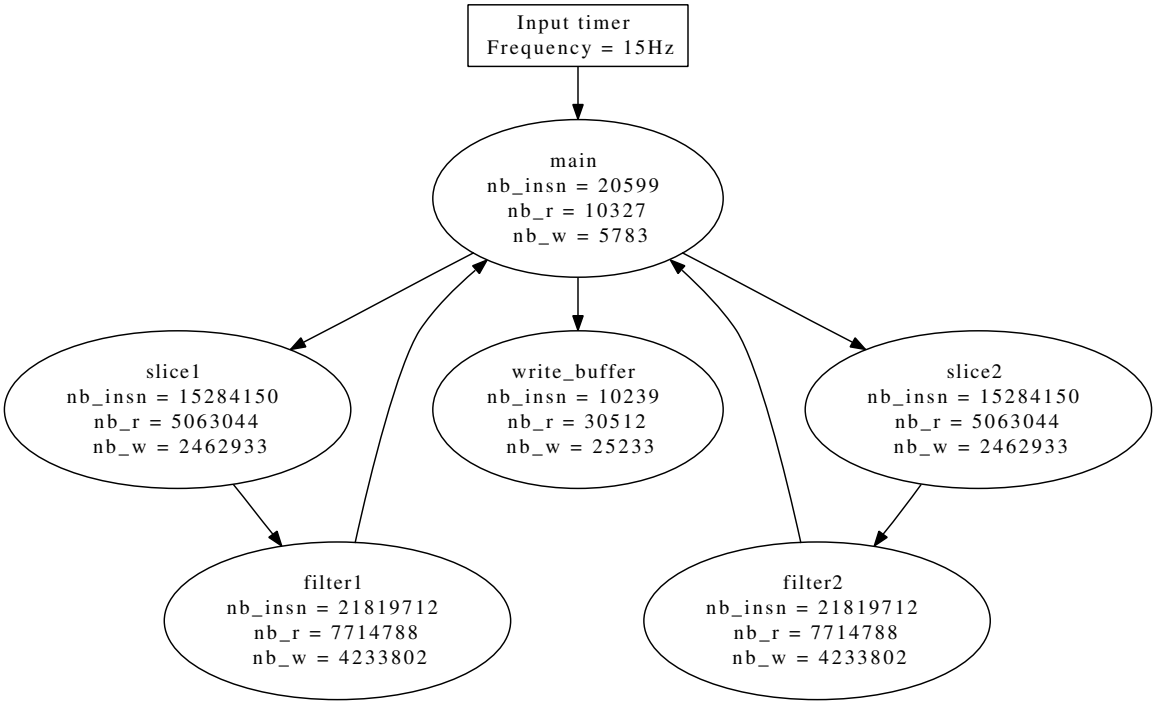

FIGURE 5.10: Modèle de l'application décodeur vidéo H.264.

$MEM\_elapsed\_time = 13.3ms$

Ensuite, il est nécessaire de calculer le temps requis pour exécuter des instructions ("perf" étant égal à 1200 car la fréquence du processeur est de 600MHz et le DMIPS/MHz vaut 2) :
$CPU\_elapsed\_time = \frac{total\_nb\_insn}{perf}$
$CPU\_elapsed\_time = \frac{15284150}{1200}$
$CPU\_elapsed\_time = 12.7ms$

Il ne reste plus qu'à faire la somme des deux pour obtenir le temps d'exécution de la tâche slice1 :
$Total\_elapsed\_time = CPU\_elapsed\_time + MEM\_elapsed\_time$
$Total\_elapsed\_time = 12.7 + 13.3$
$Total\_elapsed\_time = 26ms$

La procédure est répétée pour chaque tâche de l'application. Une fois le modèle temporel de l'application créé, la génération de code est effectuée et la simulation est lancée. On obtient alors une trace d'exécution, qui après analyse permet d'obtenir le tableau 5.2.

La comparaison entre les mesures obtenues sur la plate-forme réelle et les estimations fournies par FORECAST est faite par rapport au nombre d'images par seconde (FPS) décodées. Les résultats montrent une erreur d'estimation moyenne de 9.3% et une erreur maximum de 12.25% par rapport à la plate-forme réelle. Pour chaque exécution, FORECAST estime un temps d'exécution plus rapide que l'exécution réelle. Ceci provient en partie du fait que nous ne prenons pas en compte certains paramètres comme les cache miss en instruction et l'erreur de prédiction de branchement.
D'autre part l'estimation de la performance de l'application lorsque le filtre de déblocage est activé semble meilleure. En analysant les différences entre les deux implémentations, il apparaît que l'activation du filtre

Platform name	Real platform	custom tool (first version)	error
With filter	(FPS)	(FPS)	(%)
OMAP3 @ 600MHz	8.9	9.44	-6.07
OMAP3 @ 500MHz	7.4	7.87	-6.35
OMAP3 @ 250MHz	3.7	3.93	-6.22
OMAP3 @ 125MHz	1.8	1.97	-9.44
Without filter	(FPS)	(FPS)	(%)
OMAP3 @ 600MHz	19.3	21.56	-11.71
OMAP3 @ 500MHz	16.1	17.96	-11.55
OMAP3 @ 250MHz	8.1	8.98	-10.86
OMAP3 @ 125MHz	4	4.49	-12.25
Max. error			12.25
Mean error			9.3

TABLE 5.2: Comparaison des performances réelles et estimées par FORECAST du décodeur vidéo H.264.

entraîne beaucoup plus d'accès mémoire (2.5 fois plus que sans le filtre) que d'instructions (2.1 fois plus que sans le filtre). Cette analyse montre clairement que nous avons une plus grande erreur sur l'estimation de la performance des instructions que des accès mémoire. Ceci peut s'expliquer par le fait que nous avons omis le taux de parallélisme du pipeline ainsi que les cache miss d'instruction.
Après ces premiers résultats tout à fait encourageants, nous avons procédé de même pour une plate-forme i.MX31 avec l'application H.264, et deux nouvelles applications G.726 et JPEG, ce qui donne les résultats décrits dans le tableau 5.3.

Platform name	Real platform	custom tool (first version)	error (%)
H.264 decoder With filter	(FPS)	(FPS)	
IMX.31 @ 532MHz	5.5	5.95	-8.18
H.264 decoder Without filter	(FPS)	(FPS)	
IMX.31 @ 532MHz	12.6	14.12	-12.06
G.726 @ 32kbits/sec	(ms)	(ms)	
OMAP3 @ 600MHz	760	604	20.53
OMAP3 @ 500MHz	910	725	20.33
OMAP3 @ 250MHz	1840	1450	21.20
OMAP3 @ 125MHz	3700	2901	21.59
JPEG encoder	(ms)	(ms)	
OMAP3 @ 600MHz	240	188	21.67
OMAP3 @ 500MHz	280	225	19.64
OMAP3 @ 250MHz	560	450	19.64
OMAP3 @ 125MHz	1110	900	18.92

TABLE 5.3: Comparaison entre les performances sur plate-forme réelle et les estimations de différentes applications.

Comme nous pouvons le constater, les erreurs d'estimations pour le décodeur H.264 sont à peu près similaires que pour la plate-forme i.MX31.
En ce qui concerne les deux autres applications (G.726 et JPEG), l'estimation ne fournit pas des résultats convenables. En effet l'erreur est très proche, voir dépasse les 20% à plusieurs reprises. En fait, les applications ADPCM et JPEG exécutent un grand nombre d'opérations de base "complexe" (telle que des multiplications et divisions) qui nécessitent plus de cycle pour s'exécuter que des opérations de base "simple" (addition,

soustraction...). Un paramètre précisant le taux de parallélisme des instructions semble donc nécessaire afin d'améliorer nos estimations. De plus, FORECAST estime que l'application s'exécute plus rapidement que sur la plate-forme réelle. L'erreur étant supérieur à 20%, il n'est pas possible de rester dans cette configuration avec si peu de paramètres.

Ces premiers résultats, certes encourageants, n'ont pas une précision suffisante. Nous avons donc décidé d'ajouter d'avantage de paramètres afin d'améliorer la précision des estimations dans la version finale de l'estimateur. Nous avons identifié plusieurs paramètres manquants lors de notre première version :

Le taux de parallélisme des instructions dans le pipeline permet de prendre en compte que certaines instructions sont plus compliquées à exécuter que d'autres comme par exemple les multiplications ou les divisions (ces instructions utilisent plus de cycle CPU).

Le nombre de cache-miss d'instruction : permet de prendre en compte que lire une instruction ne se fait pas toujours immédiatement (ce qui était le cas dans notre modèle précédent), même si le plus souvent les lectures d'instruction sont consécutives.

Le nombre de mauvaises prédictions de branchement : lorsqu'une boucle ou un branchement conditionnel est effectué dans l'application, le processeur peut potentiellement se tromper d'instructions à charger. Il est alors nécessaire de vider le pipeline pour le remplir avec les instructions à exécuter et le temps d'exécution subit alors une pénalité.

La profondeur du pipeline est utilisée en complément du nombre de mauvaises prédictions de branchement. En effet, il est nécessaire de connaître la profondeur du pipeline pour savoir combien de cycles vont être nécessaires pour vider le pipeline.

Résultats des estimations finales

Nous avons effectué de nouvelles estimations qui prennent en compte ces nouveaux paramètres. Le tableau 5.4 résume les différentes estimations obtenues.

Platform name	Real platform	custom tool (first version)	custom tool (final version)	error (final version) (%)
H.264 decoder With filter	(FPS)	(FPS)	(FPS)	(%)
IMX.31 @ 532MHz	5.5	5.95	5.71	-3.82
H.264 decoder Without filter	(FPS)	(FPS)	(FPS)	
IMX.31 @ 532MHz	12.6	14.12	12.77	-1.35
G.726 @ 32kbits/sec	(ms)	(ms)	(ms)	
OMAP3 @ 600MHz	760	604	675	11.18
OMAP3 @ 500MHz	910	725	810	10.99
OMAP3 @ 250MHz	1840	1450	1621	11.90
OMAP3 @ 125MHz	3700	2901	3243	12.35
JPEG encoder	(ms)	(ms)	(ms)	
OMAP3 @ 600MHz	240	188	209	12.92
OMAP3 @ 500MHz	280	225	250	10.71
OMAP3 @ 250MHz	560	450	503	10.18
OMAP3 @ 125MHz	1110	900	1005	9.46

TABLE 5.4: Comparaison entre la plate-forme réelle (i.MX31) et la version de FORECAST avec tous les paramètres.

Comme on peut le voir, les résultats sont sensiblement améliorés lorsque ces nouveaux paramètres sont pris en compte. En effet, le parallélisme des instructions ainsi que les effets des prédictions de branchements améliorent à la fois l'estimation de la complexité des instructions mais aussi les effets dynamiques de l'application. Cela permet d'avoir une estimation inférieure à 20% d'erreurs avec un pire cas à 13%.

Une partie des erreurs restante provient probablement du modèle de latence des mémoires cache utilisé. En effet, nous avons prit un modèle générique de 10 cycles de latence pour un cache-miss en L1, mais il est tout à fait possible que ce soit en réalité 11 cycles. La figure 4.9 montre qu'il existe une légère variation suivant les plates-formes.
Une autre source d'erreur provient certainement de l'OS qui n'est pas pris en compte dans nos estimations et qui perturbe sûrement l'exécution de l'application sur le plate-forme cible.

Toujours afin de valider nos modèles, nous avons utilisé deux autres plates-formes matérielles afin d'exécuter l'application décodeur vidéo H.264. De plus, nous avons aussi utilisé le décodeur H.264 avec une vidéo haute-définition afin d'estimer si le changement de taille de vidéo impacte notre estimateur.

Le tableau 5.5 montre les différents résultats obtenus lors de ces expérimentations. Il apparaît que l'estimation faite sur différentes architectures processeurs (ARM / PowerPC) fournit également des résultats corrects. Pour le QorIQ, on pourrait s'attendre à une estimation désastreuse puisque le profiling n'a pas été effectué sur cette architecture. Cependant, comme ces deux plates-formes appartiennent à la même classe d'architecture, les estimations démontrent qu'il est tout à fait possible d'utiliser les résultats de profiling effectués sur un ARM pour une architecture QorIQ.

D'autre part, nous pouvons constater que les résultats d'estimations obtenus pour une résolution d'images nettement supérieure (1280x720 pixels contre 360x280 pixels) sont toujours très précis. Ceci indique que les

Platform name	Real platform	custom tool	error
H.264 decoder Without filter	(FPS)	(FPS)	(%)
QorIQ @ 1000MHz	36	40.84	-13.45
H.264 decoder With filter	(FPS)	(FPS)	(%)
QorIQ @ 1000MHz	20	17.26	-13.70
H.264 decoder Without filter	(FPS)	(FPS)	(%)
OMAP4430 @ 1000MHz	36.5	39.8	-9.04
OMAP4430 @ 300MHz	10.2	11.64	-14.12
H.264 decoder Without filter (720p)	(FPS)	(FPS)	(%)
OMAP3530 @ 500MHz	5.8	6.29	-8.45
OMAP4430 @ 1000MHz	12.6	13.36	-6.03

TABLE 5.5: Comparaison entre la plate-forme réelle (QorIQ) et FORECAST pour l'application H.264.

nombreux accès mémoire supplémentaires requis pour une résolution haute définition n'ont pas d'influence notable sur la précision des estimations.

Nous avons enfin effectué une estimation de performances pour un encodeur et décodeur G.726 à 40kbits/sec. Nous estimons la performance, en millisecondes, de la tâche servant à décoder, ou encoder les échantillons.

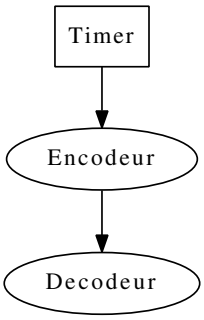

FIGURE 5.11: Modélisation de l'application G.726.

La figure 5.11 montre les deux tâches à estimer. Le timer sert à déclencher la réception des données avec une période de 125 μs (i.e. 8kHz).

Platform name	Real platform	custom tool	error
Encoder	(milliseconds)	(milliseconds)	(%)
OMAP3530 @ 250MHz	29.5	27.28	7.53
OMAP3530 @ 125MHz	59.2	54.57	7.82
Decoder	(milliseconds)	(milliseconds)	(%)
OMAP3530 @ 250MHz	30	29.63	1.23
OMAP3530 @ 125MHz	60	59.26	1.23

TABLE 5.6: Comparaison entre la plate-forme réelle (OMAP3530) et FORECAST pour l'application G.726.

Le tableau 5.6 présente les résultats des estimations ainsi que les résultats d'estimation de FORECAST. Il apparaît encore une fois que les estimations sont concluantes avec une erreur d'estimation inférieur à 10% pour les différents cas d'utilisation.

Les modèles de performances montrant des estimations satisfaisant nos contraintes avec différentes plates-formes et différentes applications, nous avons ensuite procédé aux estimations de consommation d'énergie.

Estimation de la consommation d'énergie

Comme nous l'avons évoqué dans la section 4.3.2, les estimations sont effectuées à partir de valeurs de consommation de différents éléments de base (processeur, L1, L2, et fuites) comme le montre le tableau 5.7.

Fréquence (MHz)	Energie L2 (nJ)	Energie L1 (nJ)	Energie proc (nJ)	Fuites (W)
600	0.122	0.085	0.200	0.181
550	0.115	0.084	0.167	0.147
500	0.103	0.072	0.151	0.117
250	0.076	0.056	0.127	0.038
125	0.072	0.052	0.110	0.009

TABLE 5.7: Calcul de la consommation de chaque partie du Cortex-A8.

Le tableau 5.7 récapitule les différentes consommations évaluées lors de la phase de modélisation de la plate-forme ARM Cortex-A8. Couplées aux informations de profiling de l'application (nombre d'instructions et d'accès mémoires) et aux traces d'exécutions, nous avons utilisé ces informations pour effectuer une estimation de la consommation. L'application de décodeur vidéo H.264 a été simulée afin de décoder 8 images par secondes pendant 50 images. Les résultats pour une fréquence de 600MHz sont les suivants :

- énergie mesurée : 1.736Joules
- énergie estimée : 1.584Joules
- erreur : 8.76%

Nous obtenons une erreur assez faible sachant que l'erreur d'estimation de la performance était déjà d'environ 8% sur cette application et que l'estimation de consommation dépend forcement de l'estimation de performance. Le tableau 5.8 présente d'autres résultats d'estimation de consommation d'énergie obtenus pour l'application H.264 et le décodeur JPEG avec différentes fréquences de fonctionnement.

Fréquence (MHz)	Energie mesurée (J)	Energie estimée	Erreur (%)
H.264 decoder @ 8FPS	(J)	(J)	
600	1.736	1.584	8.76
550	1.340	1.357	-1.27
500	1.020	1.109	-8.73
JPEG encoder	(mJ)	(mJ)	
600	107.57	67.89	36.9
500	77.25	53.21	31.12

TABLE 5.8: Comparaison de la consommation d'énergie mesurée et estimée de deux applications sur la plate-forme ARM Cortex-A8.

Le tableau 5.8 montre que les résultats obtenus dépendent assez fortement de la fréquence. En effet pour le décodeur H.264, bien que les estimations restent dans la borne souhaitée (erreur inférieure à 20%) l'erreur passe de +8.8% à -8.8%. Ces écarts d'estimations importants pour différentes fréquences d'utilisation ne sont bien entendu pas souhaitables. En effet, si l'estimation reste toujours optimiste et dans des bornes d'erreur acceptables, il est alors possible de faire confiance à l'estimateur en sachant que nous aurons toujours un résultat optimiste par rapport à la réalité. Malheureusement, il est ici impossible d'interpréter de manière fiable les résultats fournis par l'estimateur (optimiste ou pessimiste).

D'autre part le tableau 5.8 montre une erreur d'estimation de consommation d'énergie de l'encodeur JPEG sur la plate-forme ARM Cortex-A8 très importante (supérieure à 30%) et sortant des bornes souhaitées.

Par conséquent, nos modèles de consommation à grain fin dans l'état actuel de précision (choix des paramètres du modèle) ne répondent donc pas à nos objectifs de départ (erreur inférieure à 20%).

Nous avons alors utilisé les modèles gros grain afin d'évaluer l'impact sur les résultats d'estimation. Le tableau 5.9 présente les nouvelles valeurs d'estimation obtenues pour les applications H.264 et JPEG avec le modèle gros grain.

Fréquence (MHz)	Energie mesurée (J)	Energie estimée	Erreur (%)
H.264 decoder @ 8FPS	(J)	(J)	
600	1.736	1.759	1.32
550	1.340	1.403	4.7
500	1.020	1.07	4.9
JPEG encoder	(mJ)	(mJ)	
600	107.57	91.85	14.61
500	77.25	66.09	14.45

TABLE 5.9: Comparaison de la consommation d'énergie mesurée et estimée (avec la méthode gros grain) des deux applications sur la plate-forme ARM Cortex-A8.

Comme on peut le voir, l'estimation de consommation d'énergie est bien meilleure avec la méthode gros grain. Le fait d'utiliser une moyenne de consommation lorsque le processeur effectue du calcul réduit la marge d'erreur possible lors des estimations. On se retrouve ici avec approximativement la même erreur que pour l'estimation de performances ce qui montre que peu d'erreur est ajoutée par les modèles gros grain.

Quelle que soit l'application considérée, l'erreur d'estimation reste inférieure à 15% avec pour l'application vidéo décodeur H.264 une erreur inférieure à 5% quelle que soit la fréquence d'exécution de la plate-forme..

Ces résultats tentent à démontrer que les modèles haut-niveau permettent de limiter l'erreur d'estimation globale. En fait, lorsque le processeur exécute des instructions, sa consommation est globalement la même à quelques exceptions près. Par exemple, lorsque beaucoup d'accès mémoire sont effectués (avec beaucoup de cache miss) le processeur exécutera moins d'instructions (il sera en attente d'une donnée) et consommera moins en instantané. De même pour certaines instructions qui nécessitent plus de cycles, le processeur peut consommer plus que la moyenne.

La figure 5.12 montre par exemple, la consommation instantanée pour les deux applications utilisées précédemment.

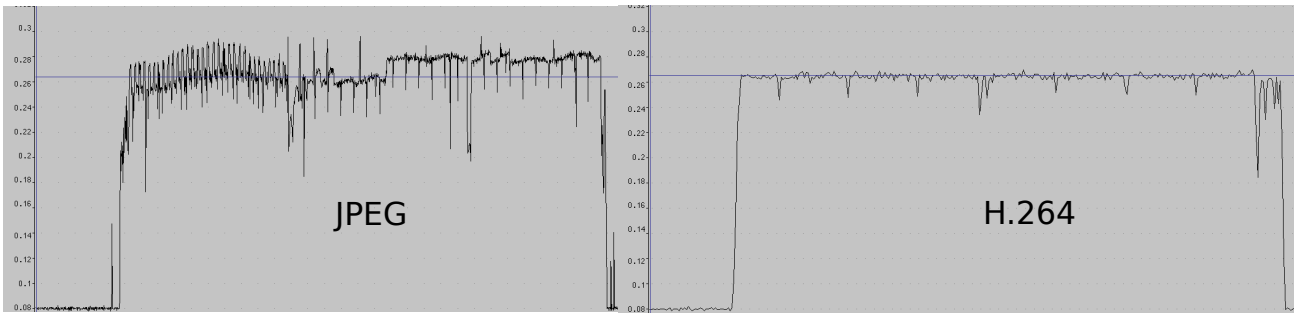

FIGURE 5.12: Consommation instantanée des deux applications (JPEG et H.264).

Comme nous pouvons l'observer, le profil de consommation est très différent entre ces deux applications.

La consommation de l'application JPEG varie beaucoup au cours du temps, alors que l'application H.264 présente un profil en consommation beaucoup plus régulier. Il est également intéressant de constater sur la figure 5.12 que les deux applications ont à peu prés la même consommation moyenne (265mW). C'est ce qui explique que l'estimation de consommation gros grain donne des résultats satisfaisants.

Nous pouvons donc conclure qu'il est très difficile d'effectuer des estimations précises de consommation à grain fin (modélisant la consommation de chaque instruction et accès mémoire) sans une modélisation extrêmement détaillée en performance ou en consommation de chaque sous-système (caches, processeurs, etc.). Sans cela, une modélisation gros grain (processeur actif ou inactif) est moins sensible aux problèmes de précision dans le cas d'une approche rapide.

5.2.2 Cas réel : Etude de faisabilité d'un produit Thales Communications and Security

Dans le cadre d'une étude pour Thales Communications and Security, nous avons procédé à une estimation de la performance du décodeur vidéo que nous possédons pour plusieurs configurations possibles et en fonction des différentes contraintes de formats vidéos que nous avions pour ce produit :

- décodage d'une vidéo au format CIF sans filtre de déblocage,
- décodage d'une vidéo au format CIF avec filtre de déblocage,
- décodage d'une vidéo au format QCIF sans filtre de déblocage,
- décodage d'une vidéo au format QCIF avec filtre de déblocage.

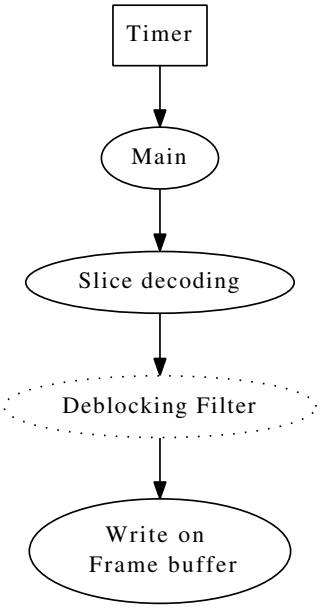

FIGURE 5.13: Modèle de l'application H.264 pour l'étude de cas réel.

La figure 5.13 montre le modèle de l'application utilisé. Le "Deblocking filter" est en pointillé car il n'est pas obligatoire : il peut être omis mais la qualité de l'image sera moins bonne.

La figure 5.14 montre la différence de performance entre l'exécution du décodeur vidéo H.264 sur la plate-forme réelle (ARM926e-js) et l'estimation obtenue avec FORECAST.
Une partie de l'erreur d'estimation vient du fait que l'application affiche l'image décodée sur un écran LCD. Or, les performances pour envoyer ces données au LCD ne sont pas prises en compte avec suffisamment de précision dans l'estimation. En effet, nous estimons que l'envoi des données LCD revient au même qu'une écriture en mémoire, ce qui n'est évidemment pas vraiment le cas. En réalité, l'écriture sur le LCD est un peu plus longue qu'une écriture en mémoire. Notre estimation est donc plus optimiste que le cas réel mais reste très proche de la valeur réelle mesurée sur la plate-forme comme le montre la Figure 5.14, ceci quelque soit le format d'image considéré.

La précision des estimations que nous avons obtenus à partir d'un modèle de haut niveau du système (gros grain) montre l'intérêt de ces estimations dans la phase amont de choix d'un processeur pour un produit. En effet, ce modèle haut niveau nous permet d'obtenir une estimation de la performance du décodeur vidéo H.264 sur un produit réel avec moins de 13% d'erreur ce qui convient parfaitement pour une étude préliminaire.

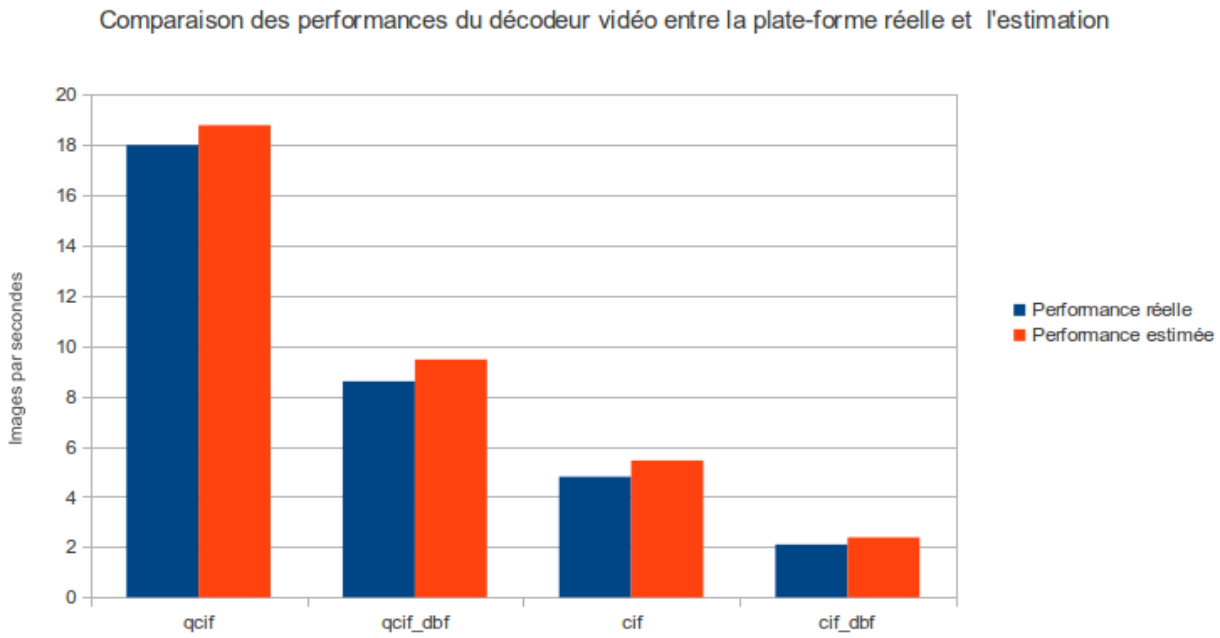

FIGURE 5.14: Comparaison des performances du décodeur vidéo entre la plate-forme réelle et l'estimation.

5.2.3 Comparaison avec une approche se basant sur QEMU

Un des objectifs du projet Européen COMCAS est d'estimer la performance et la consommation électrique d'une application sur une architecture embarquée. Pour ce faire, une méthode basée sur un émulateur d'instructions (QEMU) est utilisée. Un wrapper SystemC permettant d'ajouter des informations temporelles et des modèles de périphériques matériels a également été défini et développé par le laboratoire du TIMA à Grenoble.
Les applications de test utilisées dans ce projet font parties de la suite de benchmark "nbench". Pour nous comparer à QEMU, nous avons donc utilisé le même jeu de test. Les applications utilisées sont : "Numeric sort", "String sort", "FP Emulation", "Bitfield" et "Huffman", ce qui nous permet d'avoir un ensemble d'applications relativement diversifiées.
La figure 5.15 représente l'erreur d'estimation des différents benchmarks, en utilisant le projet COMCAS ou notre approche pour une plateforme OMAP4. Nous avons aussi utilisé plusieurs fréquences de processeur pour effectuer les comparaisons.

- Les diagrammes nommés "High level model" montrent l'erreur d'estimation de notre approche par rapport aux performances obtenues avec la plate-forme réelle.
- Les diagrammes nommés "COMCAS" montrent l'erreur d'estimation obtenue à partir de QEMU par rapport à la performance de la plate-forme réelle.

On observe tout d'abord que les pires estimations sont obtenues pour le benchmark "String sort". Ce benchmark est assez particulier et manipule des chaînes de 8 bits. Que ce soit notre méthode haut niveau, ou l'outil QEMU basé sur la traduction des instructions, les résultats d'estimation ne sont pas satisfaisants (supérieure à 20%).
Dans un second temps, on observe que toutes les autres estimations fournies par FORECAST ont une erreur comprise entre 5 et 16%, et sont toujours optimistes (comme nous l'avons déjà vu précédemment) par rapport aux valeurs réelles. Ces estimations sont néanmoins tout à fait acceptables compte tenu de nos contraintes de précisions.
En ce qui concerne la plate-forme d'estimation COMCAS, certains benchmarks, comme "Numéric sort" ou

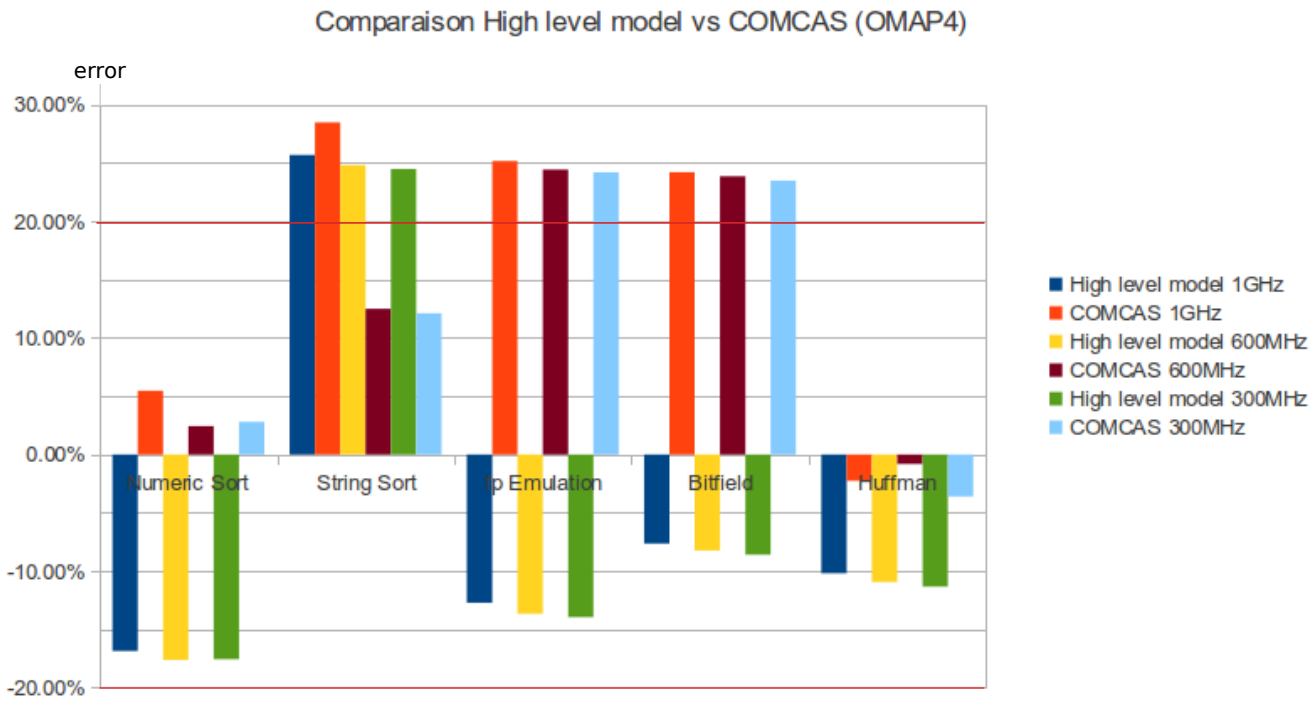

FIGURE 5.15: Comparaison des résultats de COMCAS (QEMU) et de notre approche.

"Huffman" ont une erreur d'estimation très faible (inférieur à 5%). Par contre, les autres benchmarks ont une erreur assez élevée (entre 20 et 25%). Cet outil a d'autre part plutôt tendance à faire une estimation pessimiste du temps d'exécution des applications.

Une des sources d'erreurs de l'estimation provient certainement du fait que le dual-pipeline du Cortex-A9 n'est pas modélisé dans l'émulateur QEMU. Ceci induit donc une erreur dans l'estimation des performances de la plate-forme qui peut être différente suivant les applications. De plus, la politique de remplacement des caches est aussi une source d'erreur. En effet, les constructeurs ne fournissent généralement pas d'informations sur le fonctionnement réel de leur politique. Par exemple le remplacement pseudo-random des processeurs ARM n'est pas documenté et une politique purement random est appliquée.
La figure 5.16 montre les différences de temps de lecture d'une donnée dans le cache de niveau un. On observe que l'erreur entre la plate-forme QEMU et la plate-forme réelle peut dépasser les 20%.
L'outil QEMU nécessite donc des informations très précises sur le fonctionnement des plates-formes pour fournir des estimations précises de performances. Or ces informations, permettant une modélisation grain fin du système, sont malheureusement rarement disponibles.

L'avantage du projet COMCAS utilisant un traducteur dynamique d'instructions (QEMU) réside dans le fait que l'application peut être déployée telle quelle dans leur simulateur. De plus, cette approche ne nécessite aucune modification du code ni de phase de profiling. Il est aussi possible de débugger l'application et de récupérer certaines informations comme le nombre d'accès aux mémoires caches.

Cependant, un des inconvénients majeur de QEMU est que la plate-forme est relativement lente à simuler. Le temps d'exécution des 5 benchmarks est d'environ 26 minutes : démarrage du Linux (1 minute) + exécution des benchmarks (5 minutes par benchmark dans le cas de nbench). Notre outil d'estimation FORECAST permet quant à lui d'effectuer l'estimation d'un seul benchmark en 6 secondes : 1 seconde de

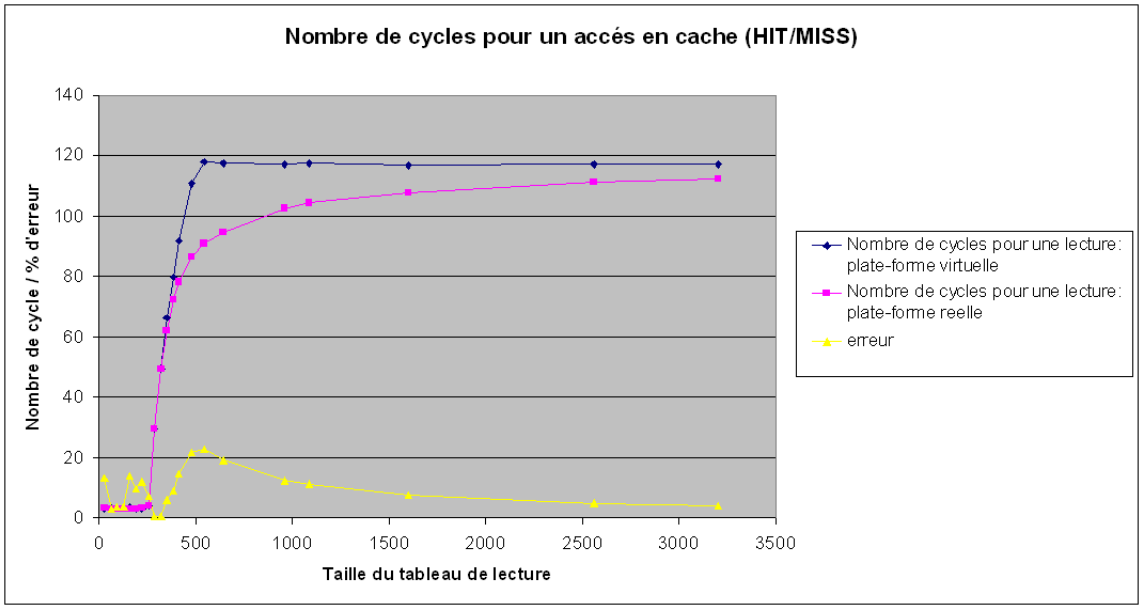

FIGURE 5.16: Différence du temps pour lire une donnée dans le cache de niveau 1 entre QEMU et la plate-forme réelle.

génération de code + 2 secondes de boucles d'initialisation + 3 secondes d'exécution. C'est un avantage non négligeable de FORECAST pour effectuer une exploration rapide d'architectures.
De plus, les erreurs d'estimations de performance sont globalement équivalentes à celles obtenues avec QEMU. Sur les benchmarks exécutés, une erreur maximale de 28.5% est observée sur le projet COMCAS, alors qu'une erreur maximale de 25.7% est obtenue avec notre méthodologie. Les erreurs moyennes sont respectivement 14.4% et 14.9%.

Un autre inconvénient d'une approche basée sur QEMU est la complexité de mise en oeuvre d'un nouveau modèle de processeur. Construire un nouveau modèle n'est en effet pas trivial, surtout lorsqu'il s'agit de modéliser les multiples pipelines internes.

La plate-forme développée dans le projet COMCAS est donc un très bon outil pour effectuer des développements logiciels et valider le comportement fonctionnel d'une application. Il permet également d'obtenir des estimations de performances assez larges du système, mais nécessite le développement d'une plate-forme virtuelle complète. Ce système de modélisation est donc intéressant lorsque le choix de plate-forme cible a déjà été effectué. La plate-forme virtuelle permet en effet de développer les éléments logiciels avec des facilités de débug tout en permettant des estimations de performances. Cependant, notre approche est plus pertinente pour effectuer des choix d'architectures et permettre d'orienter la structuration logicielle. La rapidité d'assemblage des modèles, une simulation rapide et un faible coût de développement permettent son insertion dans les phases d'architecture et de réduction des risques.

5.2.4 Comparaison avec une approche en Y-Chart basée sur le langage AADL

Estimation de performance

Dans un premier temps, il est nécessaire de connaître la performance maximale de l'application de test pour ensuite en déduire sa consommation d'énergie. Dans le projet Open-PEOPLE, l'application utilisée est une application radio. Cette application contient deux scénarios différents, comportant chacun une partie de synchronisation et une partie pour l'envoi de données. Nous rappelons que :

- le scénario 1 effectue principalement l'envoi de données importantes (image) sans cryptage.
- le scénario 2 effectue principalement le cryptage de petites données.

Nous avons tout d'abord estimé la performance des différents scénarios afin de se comparer à l'application réelle. Nous utiliserons ici la plate-forme OMAP3530 (plate-forme utilisée dans le cadre du projet Open-PEOPLE) s'exécutant à une fréquence de 600MHz.

Application name	Real platform	custom tool	error
	(milliseconds)	(milliseconds)	(%)
scenario1 synchro	10.96	12.15	10.84
scenario1 data	220	192.5	12.50
scenario2 synchro	11.03	12.27	11.28
scenario2 data	43	35.8	16.76

TABLE 5.10: Comparaison entre la plate-forme réelle (OMAP3530) et FORECAST pour l'application radio.

Le tableau 5.10 montre l'erreur obtenue en estimant les différents scénarios sur la plate-forme OMAP3530. On obtient une erreur moyenne de 13% et une erreur maximale de 16.8%.

Afin de visualiser la différence entre les deux scénarios, les deux figures suivantes (Fig. 5.17 et Fig. 5.18) présentent les diagrammes d'exécution estimés de chaque scénario. Seules les tâches les plus importantes sont représentées sur les figures.
La figure 5.17 représente la trace d'exécution de l'application radio en utilisant les paramètres du second scénario. L'estimation est effectuée en modélisant chaque bloc fonctionnel de l'application. Ceci permet à la fois d'estimer la performance globale de l'application, mais aussi de déterminer le bloc qui consomme le plus de ressources. Cela peut être utile si l'on souhaite paralléliser l'application.
Nous pouvons observer que lors de la transmission de donnée, le module de cryptage s'exécute presque autant de temps que le codeur ou le modulateur alors que pour la synchronisation, le bloc de modulation est la tâche la plus longue.
La figure 5.18 montre la trace d'exécution de l'application radio en utilisant les paramètres du scénario un. Cette trace permet de visualiser très facilement les différences dans la gestion des données entre les 2 scénarios. En effet, dans ce scénario, 8 slots de données sont envoyés après chaque slot de synchronisation afin de permettre un débit utile plus important dû à l'envoi d'images (et non uniquement de voix). Nous observons aussi que le bloc "cypher" pour les données est très faible. Ceci est dû au fait que dans ce scénario, les données ne sont pas cryptées. Cet effet est visible sur la figure 5.18 car lorsque le bloc "cypher" démarre, il laisse la main au "coder" quelques microsecondes après et attend uniquement la fin de tous les blocs suivants pour retourner au repos. Le "coder" et le "modulator" effectuent donc ici le gros du traitement.

Estimation de la consommation électrique du système

Dans un premier temps, nous avons comparé les estimations de consommation obtenues grâce à notre approche, avec la consommation réelle du système. Basées sur les estimations de chaque élément indépendant (instructions processeurs, accès mémoires, courant de fuites), les estimations sont générées à la suite de

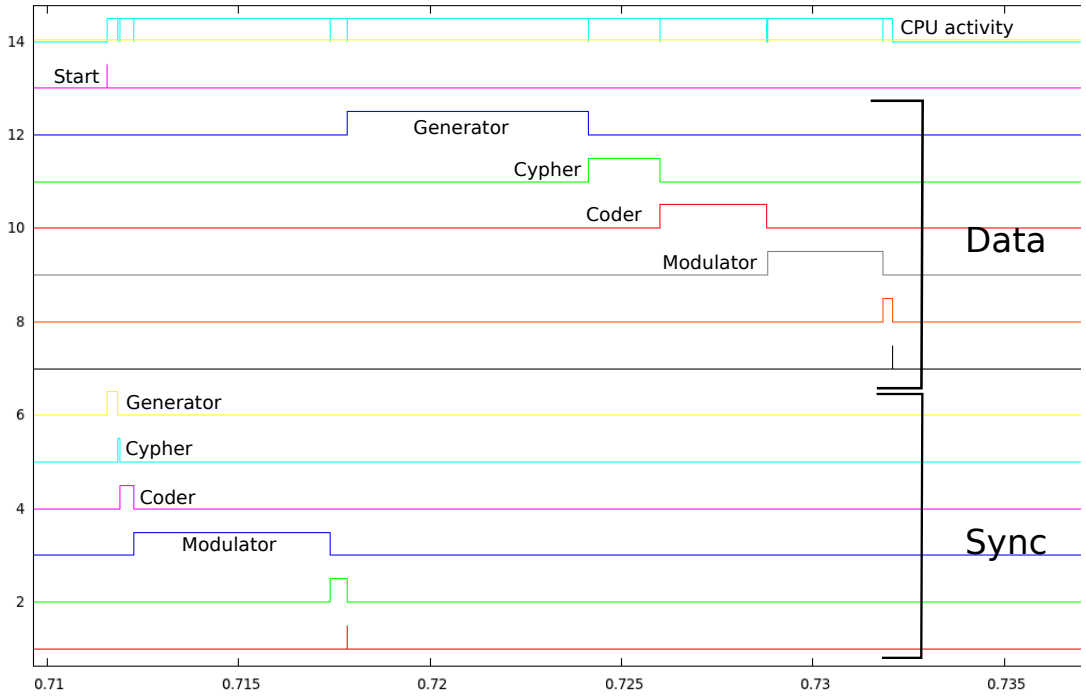

FIGURE 5.17: Exemple d'exécution du scénario deux.

l'exécution de chaque scénario. Nous avons aussi utilisé la méthodologie de l'estimation "gros grain" afin de comparer les deux méthodes.
La figure 5.19 montre l'estimation d'énergie consommée par chaque scénario et pour chaque fréquence disponible sur la plate-forme réelle. Bien évidemment, pour chaque fréquence le temps d'exécution est différent, ce qui signifie que la puissance consommée est moindre pour une fréquence plus basse, mais le programme prend plus de temps pour s'exécuter. De plus, du fait que la tension diminue lorsque la fréquence diminue, l'énergie consommée par le programme est plus faible lorsque la fréquence est moindre.
En comparant les estimations avec la mesure réelle, il apparaît que l'erreur est très importante pour l'estimation à "grain fin" avec une erreur entre 20 et 30%. D'un autre coté, l'erreur d'estimation pour la méthodologie "gros grain" est comprise entre 2 et 19%. Contrairement à ce que l'on pourrait espérer, les estimations haut niveau ont une erreur en moyenne plus faible que l'estimation grain fin. Il apparaît donc que vouloir à tout prix utiliser des modèles très précis avec beaucoup de paramètres n'est pas immédiat et simple à mettre en place, et donne souvent de moins bons résultats que d'utiliser des modèles simples.

On observe, de plus, que le scénario 1 présente une plus grande erreur d'estimation. La figure 5.20 illustre la consommation instantanée lors de l'exécution du scénario 1. Il apparaît que la consommation varie énormément pour cette application, ce qui fait que la consommation est plus élevée que la consommation moyenne utilisée dans notre modèle (droite bleue sur la courbe).
Il est donc normal que la précision de l'estimation soit moins bonne (19% maximum) que les deux précédentes application évaluées (5% pour H.264 et 15% pour JPEG).

Nous avons également comparé nos estimations par rapport à la méthodologie utilisée dans le projet Open-PEOPLE.
Ce projet ne vise pas à estimer la performance mais seulement la consommation d'énergie. La performance des différentes fonctions/tâches est alors mesurée sur la plate-forme réelle afin d'alimenter les modèles de consommation d'énergie.

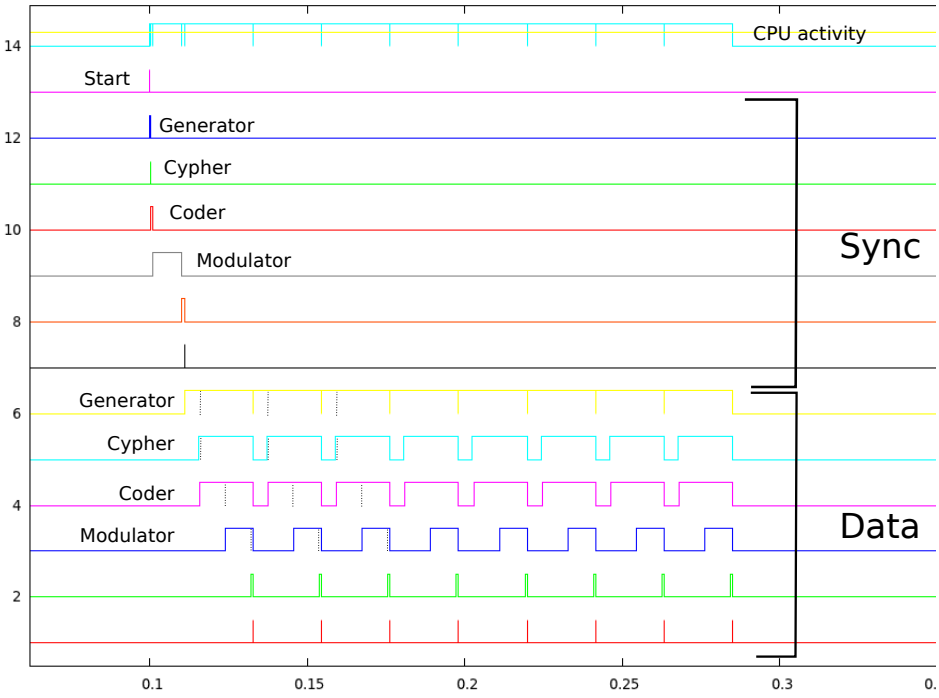

FIGURE 5.18: Exemple d'exécution du scénario un.

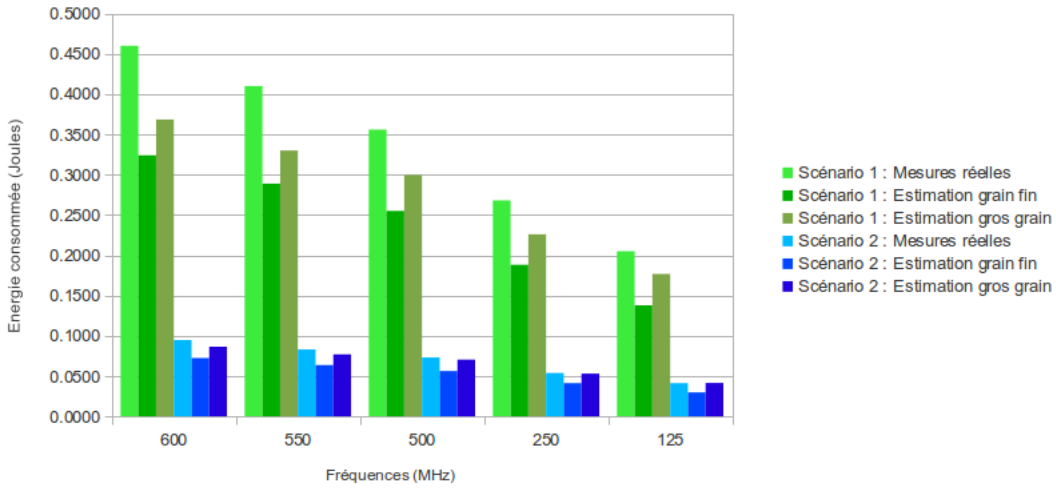

FIGURE 5.19: Comparaison de la consommation d'énergie des deux scénario en utilisant différentes méthodes d'évaluation.

Rappelons tout d'abord le modèle de consommation déduit des mesures sur le processeur ARM Cortex-A8 présent dans l'article [124] : $P(mW) = 0.79 \cdot F_{Processor} + 18.65 \cdot IPC + 0.26 \cdot (y_1 + y_2) + 10.13$
Où :

- $F_{Processor}$ représente la fréquence du processeur.
- IPC représente le nombre d'instructions par cycle.

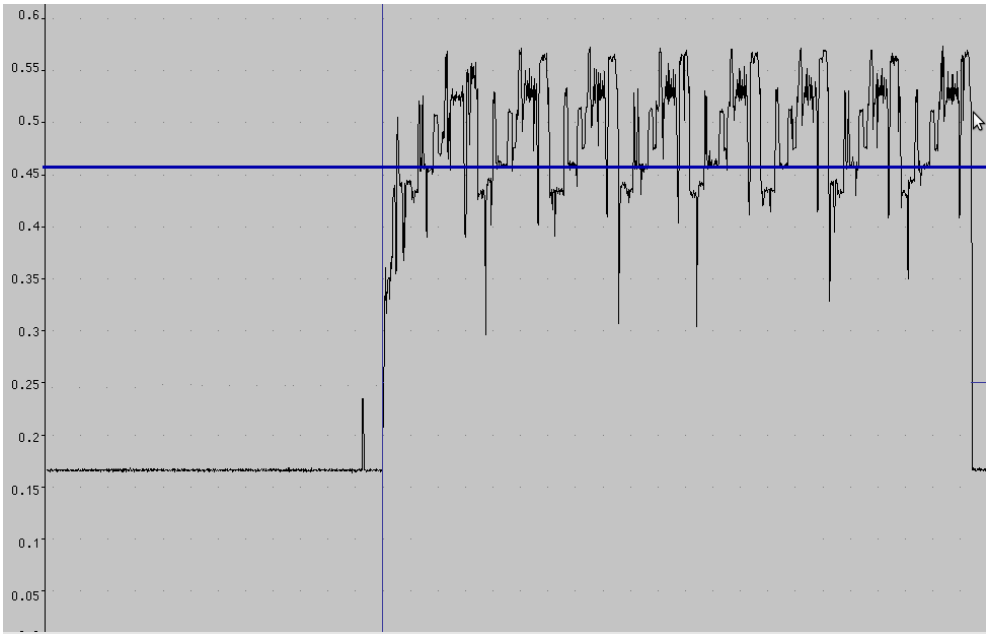

FIGURE 5.20: Consommation électrique des 8 slots de données du scénario 1.

– y représente le taux de cache-miss.

L'erreur maximal observé lors des expérimentations effectuées dans cet article est de 4% ce qui s'avère être très précis. Il est tout de même possible de s'interroger sur certains points.
En effet, les paramètres de cache-miss possèdent un constante très faible (0.26) ce qui signifie que les caches miss n'ont pas beaucoup d'impacts sur la consommation. En général, le taux de cache miss d'une application est inférieur à 5% ce qui veut dire que les valeurs de consommation dépendant du cache miss vont varier entre 0 et 1.3 mW. Ce qui est donc négligeable comparé aux $0.79 \cdot 500 = 395$mW pour une fréquence processeur de 500MHz. De plus, le taux de cache du niveau 1 et du niveau 2 possèdent le même impact sur la consommation, ce qui n'est pas logique.
En analysant de plus prés tous les paramètres, il apparaît en fait que la majeure partie de la variation de consommation suivant l'application vient du nombre d'instructions par cycle (IPC). En effet, lorsque le taux de cache miss est élevé, alors l'IPC va être très faible car le processeur va attendre très souvent les données, ce qui va baisser la consommation.
On peut alors s'interroger sur l'utilité des paramètres y_1 et y_2 étant donné que la plus grande partie de leur impact semble plutôt dans le paramètre IPC.

Le temps de simulation pour l'application radio est comprise entre 4.77 secondes et 6.74 secondes, ce qui est plutôt rapide.

D'après les résultats obtenus et les temps de simulations correspondants, la méthodologie semble intéressante à étudier. En effet, le calcul de la puissance consommée et non de l'énergie consommée comme nous avons souhaité le développer dans notre méthodologie semble moins soumis aux erreurs. Il apparaît que c'est un bon compromis entre les méthodes gros grain et fin grain.

Cependant, quelques inconvénients majeurs pour la méthodologie globale apparaissent comme par exemple :

– La difficulté de prise en main d'OVPSim . Il est en effet nécessaire de modifier la plate-forme virtuelle afin d'être en mesure de récupérer les activités de chaque bloc.
– Il n'est pas possible d'exécuter un OS sur la plate-forme virtuelle.

– L'estimation de consommation est basée sur la mesure réelle du temps d'exécution et non sur des estimations, ce qui complique l'exploration d'architectures. Il serait néanmoins possible d'utiliser nos estimations de performance dans un premier temps puis d'appliquer leur méthodologie d'estimation de consommation.

Pour conclure, un des avantages de l'approche proposée par le projet Open-PEOPLE vient de son coté ouvert et communautaire. En effet, il est possible à quiconque d'ajouter de nouveaux modèles de consommation dans l'environnement de partage, ce qui permet de pouvoir utiliser différents modèles plus ou moins complexes et précis.
Un autre point important est la possibilité de modéliser à la fois les processeurs du type GPP ou DSP, mais aussi des plates-formes à base de FPGA.

5.3 L'estimation appliquée à des plates-formes multi-processeurs

Un de nos objectifs est de pouvoir estimer la performance et la consommation de plates-formes multi-processeurs. Or, comme le générateur de code est utilisable pour des applications s'exécutant sur plusieurs processeurs, il a été relativement aisé d'étendre notre méthodologie pour des architectures multi-processeurs. Nous avons dans un premier temps effectué des comparaisons avec de vraies plates-formes, afin d'évaluer la précision de FORECAST pour différentes applications, puis nous avons comparé notre approche avec le projet COMCAS.

5.3.1 Comparaison avec de vraies plates-formes

Estimation

Le décodeur vidéo H.264 étant actuellement la seule application parallélisée dont nous disposons, nous l'avons utilisée pour comparer nos estimations avec les mesures effectuées sur des plates-formes réelles. Nous sommes capables grâce à cette application de paralléliser les traitements sur autant de coeurs que l'on souhaite (deux ou quatre dans notre cas).

Le premier cas d'étude cible le dual-coeur, avec une architecture de type ARM Cortex-A9 (OMAP4430) et une architecture de type Freescale e500 (QorIQ).

Platform name	Real platform	custom tool	error
H.264 decoder Without filter	(FPS)	(FPS)	(%)
OMAP4430 @ 1000MHz	67.2	73.3	-9.08
OMAP4430 @ 300MHz	19.8	22.0	-11.11
QorIQ @ 1000MHz	70	77.6	-10.8

Table 5.11: Comparaison entre les plates-formes réelle multi-coeurs et FORECAST.

Le tableau 5.11 montre l'erreur d'estimation pour l'application s'exécutant sur la plate-forme OMAP4430. On peut observer une erreur d'environ 10% que ce soit pour une fréquence de 300 ou 1000MHz. L'analyse des traces d'exécution a permis de vérifier que les tâches étaient bien exécutées en parallèle (ce qui est logique étant donné la performance atteinte). On remarque aussi une erreur d'estimation quasiment équivalente pour la plate-forme QorIQ P2020.

La figure 5.21 montre les résultats des expérimentations menées pour l'application vidéo décodeur H.264 sur la plate-forme i.MX6 avec deux ou quatre coeurs. On peut voir sur le graphique que la différence de performance entre le réel et l'estimé est assez faible. Pour l'architecture contenant 2 coeurs, une erreur moyenne de 3% est observée alors que pour l'architecture 4 coeurs, une erreur moyenne de 4% est observée dans l'estimation.

Pour ce qui est de la consommation d'énergie, nous avons effectué des essais avec la méthode gros grain. Nous avons utilisé la plateforme OMAP44xx comme plateforme de test afin de comparer les estimations de la consommation du vidéo décodeur H.264 avec la mesure réelle.
L'application est programmée pour décoder 30 images par seconde (avec 50 images à décoder). La simulation est effectuée pour différentes fréquences afin d'analyser la consommation pour chacune d'elles. Nous utilisons aussi 1 ou 2 coeurs afin d'évaluer la consommation de chaque configuration.

Le tableau 5.12 montre les résultats de la mesure de consommation d'énergie ainsi que l'estimation faite par FORECAST pour les configurations double coeurs. L'erreur d'estimation est au maximum de 14.5% ce qui

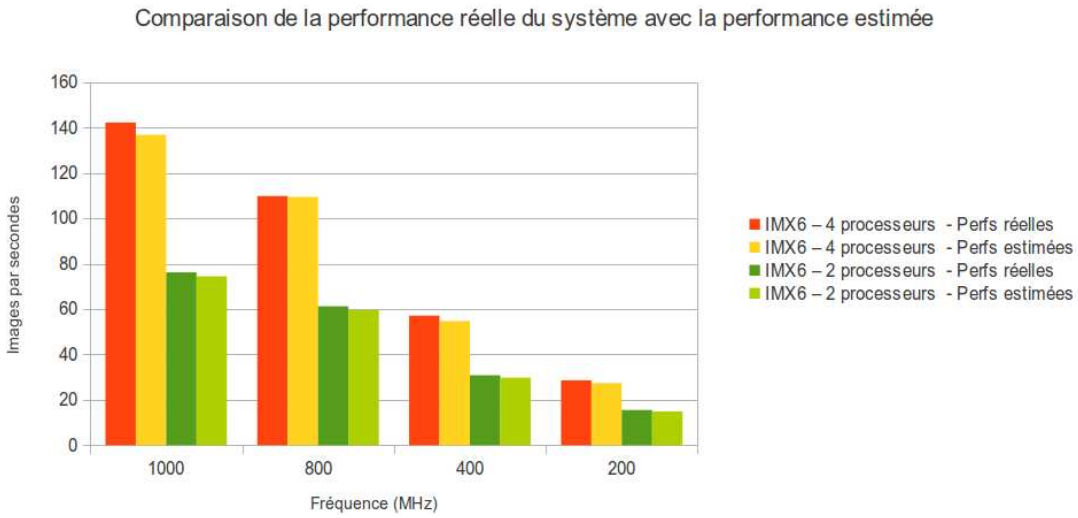

FIGURE 5.21: Comparaison entre l'estimation de la performance et la performance réelle du système (i.MX6 + vidéo décodeur H.264).

Fréquence (MHz)	Consommation mesurée(J)	Consommation estimée(J)	Erreur (%)
1000	1.26	1.08	14.24
800	0.98	0.85	13.39
600	0.64	0.57	11.41

TABLE 5.12: Consommation d'énergie de l'OMAP4 en double-coeurs.

est légèrement plus élevé que l'erreur purement associée à la performance (11%). En effet, les deux erreurs s'accumulent et il est donc normal que l'estimation de consommation soit plus élevée que l'estimation de performance.

Fréquence (MHz)	Consommation mesurée(J)	Consommation estimée(J)	Erreur (%)
1000	1.08	0.93	13.61
800	0.94	0.79	16.38

TABLE 5.13: Consommation d'énergie de l'OMAP4 en mono-coeur.

Le tableau 5.13 présente les résultats de consommation d'énergie pour les configurations mono-coeur afin de comparer la consommation des différentes architectures. Ici, on peut remarquer qu'il n'y a que deux fréquences. En effet, FORECAST estime que les *deadlines* seront respectées uniquement pour ces deux fréquences. En réalité lorsque le processeur est à 800MHz, la contrainte temps réel n'est strictement pas respectée puisque 29.2 images par secondes peuvent être décodées. On remarque tout de même que l'erreur d'estimation reste inférieure à notre borne (20%).

Tant qu'on ne sature pas les processeurs, on peut aussi observer qu'à fréquence équivalente, et si la contrainte temps réel est respectée, il est plus intéressant de n'avoir qu'un seul coeur en fonctionnement pour optimiser la consommation d'énergie. En effet, si on prend par exemple une fréquence de 1000MHz, la consommation en double coeur est de 1.26Joules alors que la consommation en mono coeur est de 1.08Joules. Le passage d'une architecture mono-coeur à une architecture multi-coeur est justifié dans deux cas : soit lorsque la performance n'est pas atteinte par la plate-forme mono-coeur, soit lorsque la réduction de la

fréquence des deux coeurs permet un gain en consommation. En effet, si l'on prend la plate-forme et que l'on baisse la fréquence à 600MHz, la consommation d'énergie est réduite tout en permettant de respecter les contraintes temps-réels.

Exploration

Afin de valider le bon fonctionnement de l'exploration d'architectures, nous avons procédé ainsi :
- Comparaison des résultats par rapport à l'i.MX6
- Utilisation de l'application vidéo décodeur H.264
- Recherche de l'architecture la plus optimisée en performance (et en conso) pour différentes vitesses d'arrivées des images à traiter.

Dans un premier temps, nous avons souhaité uniquement exécuter l'application à la vitesse souhaitée sans chercher à contraindre le temps d'exécution d'une tâche (i.e. aucune *deadline* par tâche).
La plate-forme en double coeurs est tout d'abord utilisée. La contrainte appliquée est de 50 images par seconde. Nous décidons également de limiter la charge des processeurs entre 80 et 90% (comme le laisse entendre la figure 5.22) afin de s'assurer de leur bon fonctionnement et de palier à d'éventuels surcharges occasionnelles.

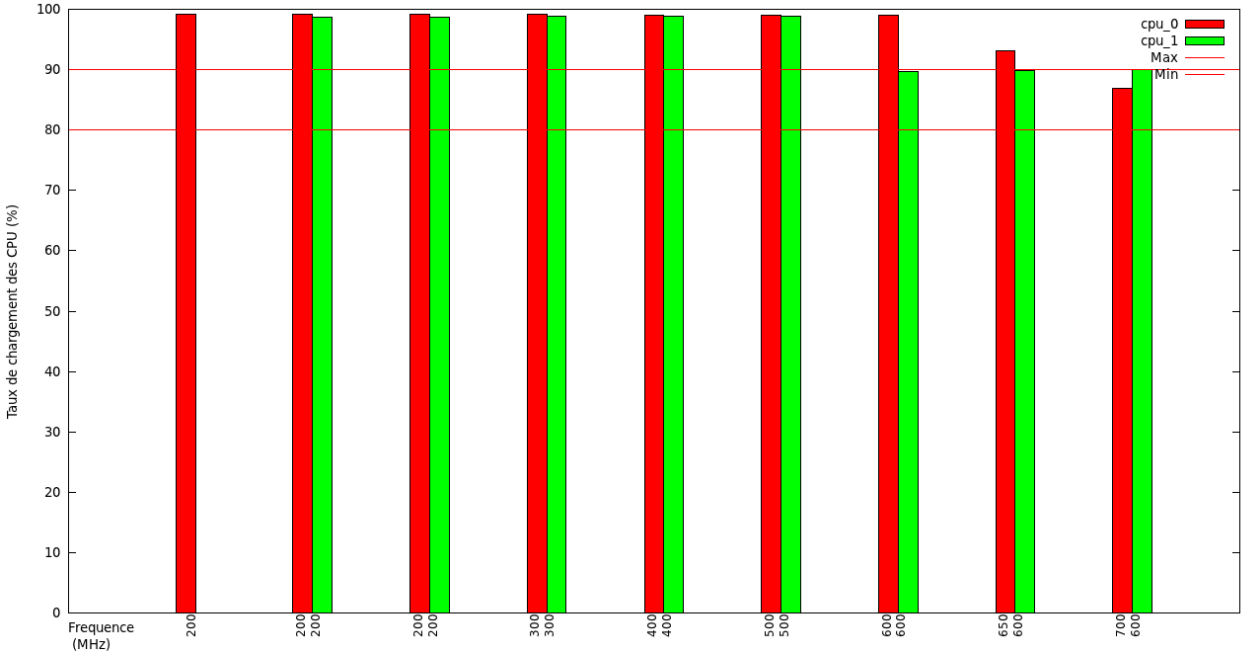

FIGURE 5.22: Exploration avec 2 processeurs.

La figure 5.22 montre le taux de charge de chaque processeur pour chaque itération de l'explorateur. On peut voir que l'explorateur converge vers une solution utilisant le premier processeur à 700MHz et le deuxième à 600MHz. Cette solution permet de respecter la limite de charge de 90% pour les deux processeurs.

Comparons maintenant la solution choisie par l'explorateur avec les performances de l'application sur la plate-forme réelle. La plate-forme réelle n'offrant pas toutes les fréquences disponibles mais seulement quatre (200, 400, 800, 1000MHz), nous avons extrapolé entre les différentes fréquences. De plus, les fréquences doivent être les mêmes pour chaque processeur sur la plate-forme réelle.
La figure 5.23 compare la solution choisie par l'explorateur, avec les performances réelles du système. Sur l'axe des Y sont notés le nombre d'image par seconde décodées par la plate-forme. Comme nous avons choisi une contrainte de 50 images par seconde, nous recherchons donc la plate-forme qui correspond à cette performance. La courbe rouge représente la performance réelle du système en fonction de la fréquence (pour les deux coeurs). L'intersection des deux courbes donne donc la plate-forme optimale afin de satisfaire la contrainte.

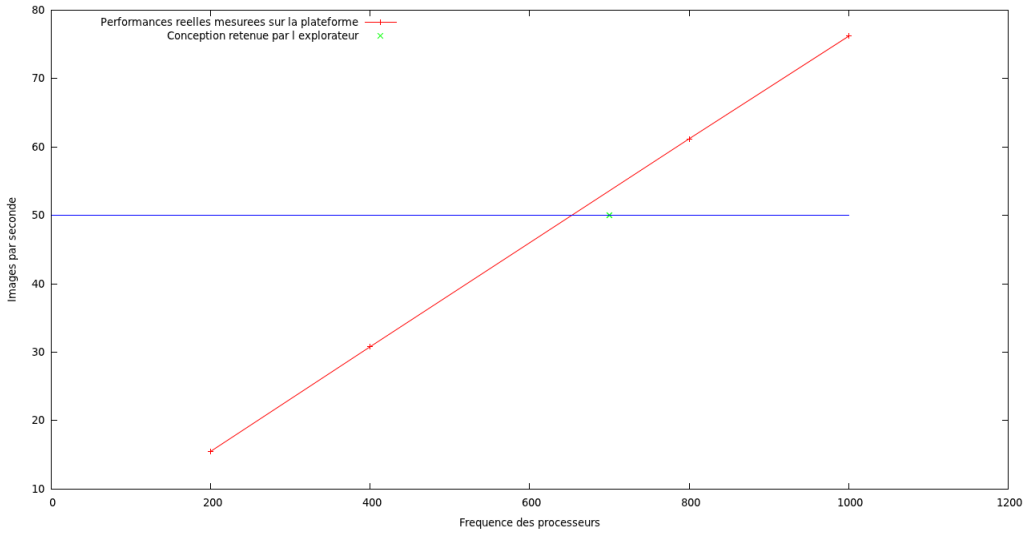

FIGURE 5.23: Comparaison des performances mesurées avec la solution choisie par l'explorateur.

Le point vert représente la configuration choisie par l'explorateur. Comme on peut le voir, la solution choisie par l'explorateur n'est pas très éloignée de la solution optimale. En fait, l'explorateur trouve une solution avec des valeurs de fréquences légèrement plus élevées car nous l'avons contraint afin que les processeurs ne soient pas chargés à plus de 90%. Or, pour la vraie plate-forme, il n'y a pas de possibilité de limiter ce facteur et la plate-forme s'approche des 100% de charge (le processeur le plus chargé des deux).
L'intérêt est donc de permettre à l'architecte d'évaluer la configuration nécessaire de la plate-forme pour une contrainte donnée sans posséder la plate-forme réelle.

Dans un second exemple, nous avons utilisé une plate-forme quadri-processeurs et fixé une contrainte de 100 images par seconde à décoder. Nous décidons également lors de l'exploration de limiter la charge des différents processeurs entre 80 et 90%.
La figure 5.24 montre le déroulement de l'exploration pour quatre processeurs. Comme on peut le voir, l'explorateur commence par paralléliser les tâches sur les différents processeurs. Les contraintes spécifiées n'étant pas respectées, l'explorateur augmente alors la fréquence des différents processeurs dont la charge est supérieure 90%.
L'explorateur converge vers une solution consistant à utiliser un processeur à 800MHz et les autres processeurs à 600MHz. Le premier processeur exécutant une partie de l'application "non parallélisée", il est normal qu'il nécessite une fréquence plus élevée afin de compenser ce manque de parallélisme. Les processeurs sont alors tous à environ 85% de charge.

Nous avons alors comparé ce résultat avec la performance de l'application s'exécutant sur la plate-forme réelle. De même que précédemment, les performances ont été extrapolées entre les fréquences disponibles sur la plate-forme (200, 400, 800, 1000MHz). De plus, les fréquences doivent être les mêmes pour chaque processeur sur la plate-forme réelle.
La figure 5.25 montre la comparaison entre la solution choisie par l'explorateur, et les performances réelles de l'application sur la plate-forme. La courbe bleue représente la contrainte temporelle (100 images par secondes) alors que la courbe rouge représente les performances réelles. Le croisement des deux représente donc la solution optimale pour une contrainte de 100 images par secondes. Cette solution est atteinte pour une fréquence des processeurs d'environ 725MHz.
La solution choisie par l'explorateur est indiquée par une croix verte et représente une solution pour une fréquence de 800MHz (nous choisissons le processeur avec la plus haute fréquence). L'erreur dans le choix de la solution provient du fait que l'on force la plate-forme à garder entre 10 et 20% de charge processeur libre

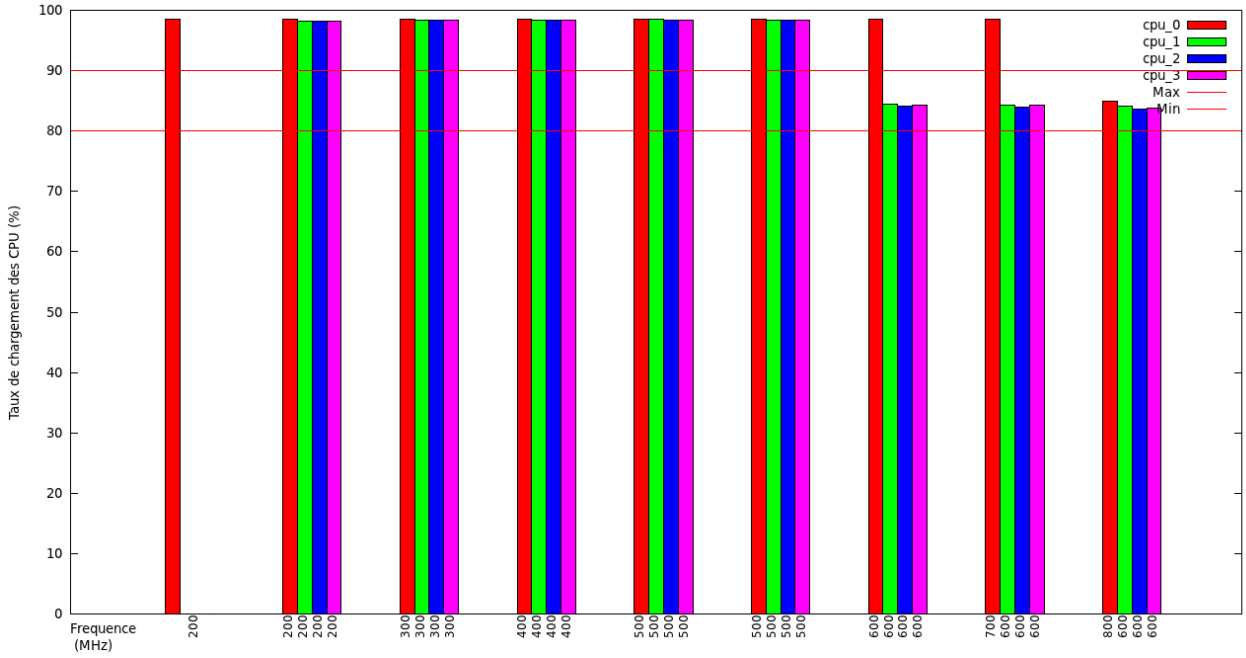

FIGURE 5.24: Exploration avec 4 processeurs.

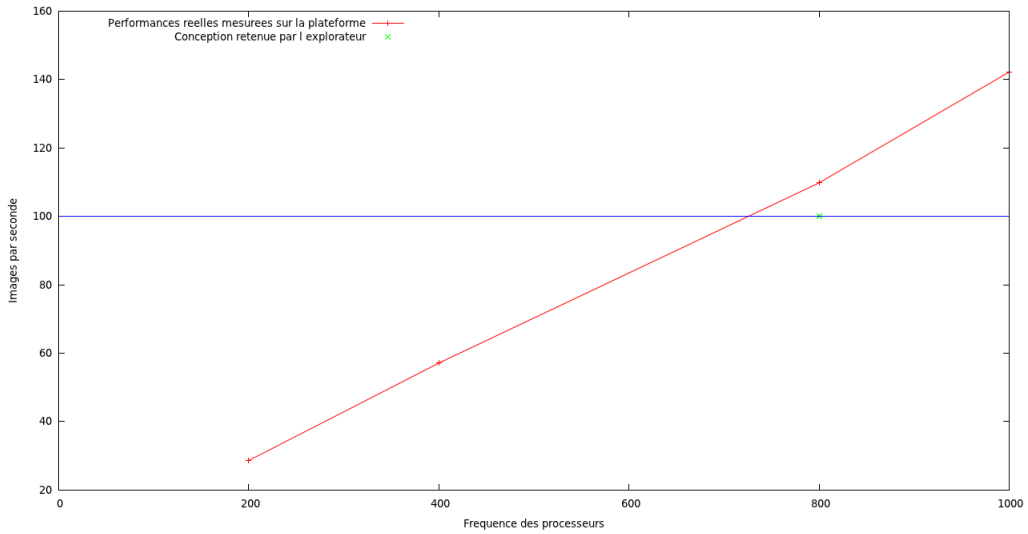

FIGURE 5.25: Comparaison des performances mesurées avec la solution choisie par l'explorateur.

afin de s'assurer du bon fonctionnement de l'application. Charger un processeur à 100% est en effet risqué dans le cas d'applications temps-réels.
En extrapolant les performances estimées par l'explorateur dans le cas où le processeur serait à 100% de charge, on trouve une fréquence de fonctionnement de 675MHz (800 - 15%) ce qui nous rapproche de la valeur réelle.
Ces deux expérimentations ont montré que l'explorateur est capable d'aider l'architecte en trouvant une

solution adaptée aux besoins de l'application choisie.

Nous avons également validé notre explorateur dans le cas où des contraintes temps-réel par tâche sont spécifiées. La procédure suivante a été appliquée afin de visualiser le comportement de l'explorateur pour l'application H.264 pour une plate-forme quadri-processeurs :

- Les processeurs démarrent à des fréquences différentes.
- Les images à décoder arrivent toutes les 12 millisecondes.
- Les tâches de décodage sont contraintes à 6 millisecondes maximum de temps d'exécution.

Cette forte contrainte pour les tâches de décodage devrait permettre de bien visualiser l'effet de la contrainte temps-réel sur le processus d'exploration.

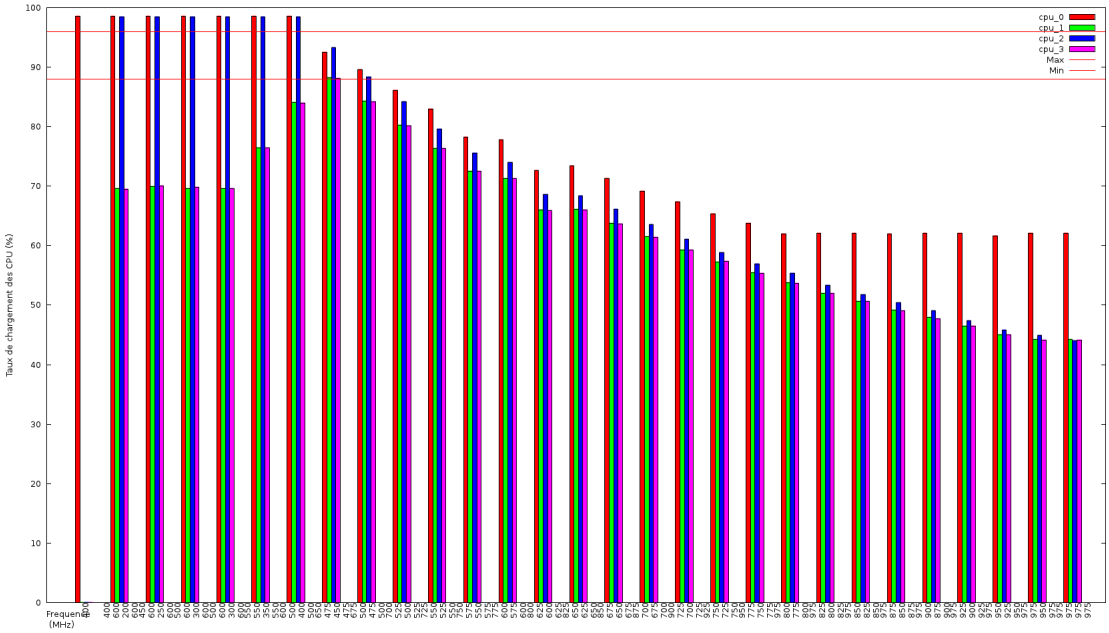

FIGURE 5.26: Évolution de la charge des processeurs au cours de l'exploration lorsque les tâches sont contraintes.

La figure 5.26 représente les différentes itérations de l'explorateur afin de trouver une solution optimale pour notre cas d'étude. Tout d'abord, l'explorateur essaie lors des huit premières itérations d'atteindre la contrainte de charge de processeur en parallélisant et en modifiant les fréquences. On observe également que les processeurs démarrent avec des fréquences différentes. Ici, le processeur 2 commence avec une fréquence de 200MHz, le processeur 0 avec une fréquence de 400MHz et les processeurs 1 et 3 démarrent avec une fréquence de 600MHz. A la 8ème itération, on peut vérifier que les quatre processeurs ont tous une charge de calcul comprise dans les bornes fixées (entre 88 et 95%).
Dans une seconde partie, l'explorateur va chercher des solutions permettant de respecter les contraintes temps-réel. Bien évidemment, ces contraintes ne sont pas respectées dans le cas où les processeurs se trouvent dans les bornes de charge souhaitées (entre 88 et 95%). Le débit d'entrée des images étant de 12 millisecondes, cela signifie que les tâches s'exécutent entre 11 et 12 millisecondes pour décoder chaque image. L'explorateur augmente alors la fréquence des processeurs dont les tâches ne respectent pas les contraintes.
Comme on peut le voir sur la figure 5.26, la solution finale proposée par l'explorateur indique que les quatre processeurs doivent fonctionner à 975MHz. On observe alors que les processeurs sont chargé à environ 50% ce qui est logique étant donné qu'une image arrive tous les 12 millisecondes et qu'on la traite en moins de 6 millisecondes. Il apparaît aussi que le processeur 0 possède une charge processeur plus élevée, car c'est ce processeur qui exécute la partie non-parallèle de l'application.

Une dernière expérimentation a été effectuée afin d'utiliser des contraintes différentes pour chaque tâche. Dans cet exemple, deux tâches doivent s'exécuter en moins de 6 millisecondes, la troisième en moins de 8 millisecondes et la dernière en moins de 10 millisecondes. Cet exemple doit permettre de montrer que l'explorateur traite chaque tâche indépendamment.

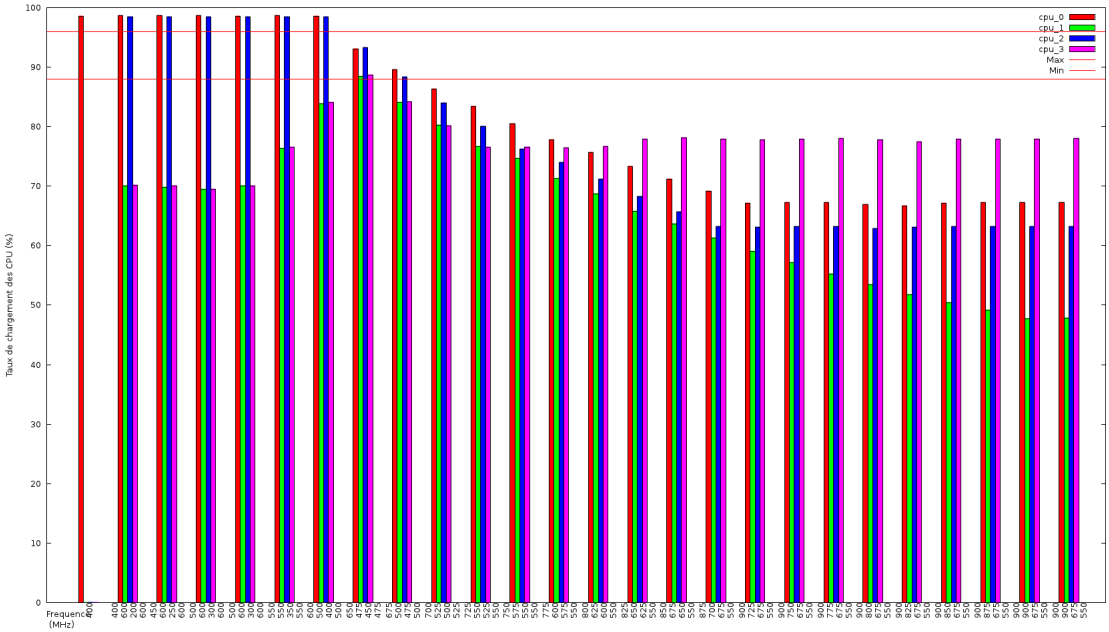

FIGURE 5.27: Évolution de la charge processeurs au cours de l'exploration lorsque les tâches sont contraintes avec des valeurs différentes.

Comme on peut le voir sur la figure 5.27, les taux de charge des processeurs 2 et 3 sont plus importants que dans l'exemple précédent. En effet, sur le processeur 2 se trouve la tâche ayant un temps maximum d'exécution de 8 millisecondes et sur le processeur 3 la tâche ayant un temps maximum de 10 millisecondes. La figure 5.28 montre le temps maximum d'exécution de chaque tâche contrainte en temps. Ce graphique est fourni par l'explorateur afin de s'assurer que les tâches respectent bien les contraintes souhaitées. Les trois courbes horizontales représentent les temps maximum autorisés pour les tâches, soit 6, 8 et 10ms. On observe ensuite le temps maximum d'exécution observé de chaque tâche pour chaque itération de l'explorateur. Le temps d'exécution décroît à chaque itération jusqu'à passer sous le seuil de la contrainte temps-réel fixée. Lorsque toutes les contraintes sont respectées, l'explorateur s'arrête.

Ces différents exemples montrent la méthodologie mise en oeuvre afin d'effectuer l'exploration d'espace des conceptions. L'exploration permet à l'architecte de choisir très rapidement une architecture processeur ainsi que la répartition du logiciel associée sans pour autant posséder de plate-forme. Le gain en temps est alors important puisqu'il est possible d'étudier différentes contraintes temps-réel. De plus, l'explorateur permet l'utilisation d'une plate-forme dans des conditions qui consomment le moins d'énergie possible. De plus, le simulateur fournissant un grand nombre d'informations en sortie, il est aisé d'ajouter d'autres algorithmes d'exploration basés sur ces informations.

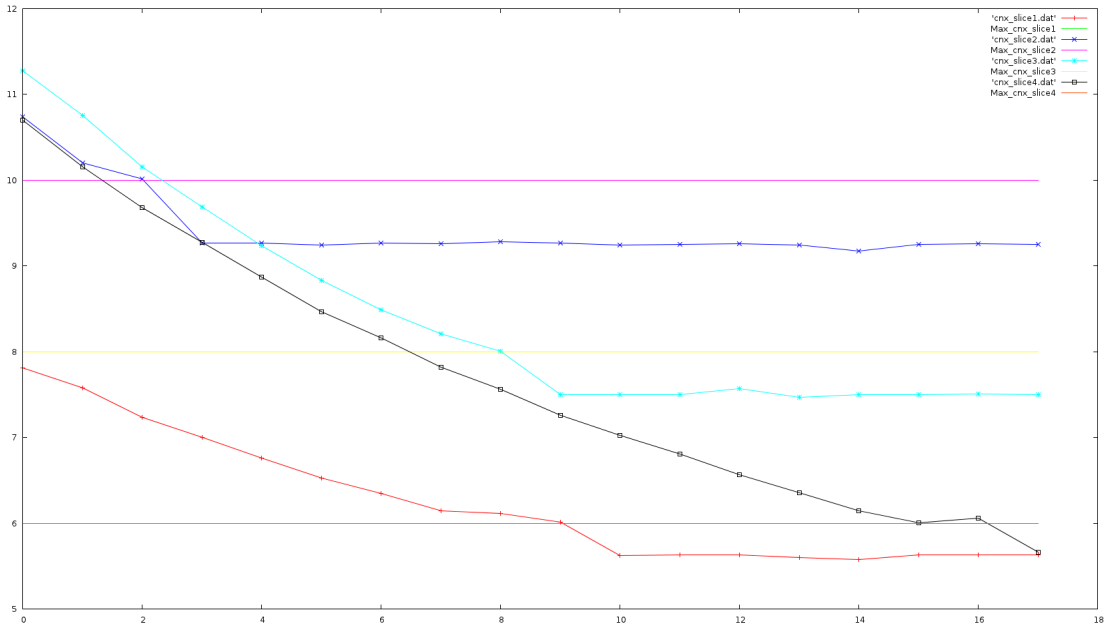

FIGURE 5.28: Évolution du temps maximum d'exécution de chaque tâche au cours des itérations de l'explorateur.

5.3.2 Comparaison à QEMU (projet COMCAS)

Après avoir comparé les résultats de notre approche avec l'approche utilisée dans le projet COMCAS (QEMU + SystemC) pour les plates-formes mono-coeur, nous avons décidé de comparer les processeurs multi-coeurs. Pour cela, nous avons étudié les erreurs d'estimation en performance de l'application H.264. La figure 5.29 montre l'erreur d'estimation des deux approches pour différentes fréquences d'utilisation des processeurs. Dans un premier temps, une estimation est effectuée dans le cas où un seul coeur de la plate-forme est utilisé. Bien que l'erreur d'estimation de la plate-forme QEMU soit acceptable pour des fréquences de 300 et 600MHz (inférieur à 20%), l'erreur est très importante pour une fréquence de 1GHz (33%).
On remarque quasiment le même comportement pour le cas où l'on utilise les deux coeurs. L'estimation est satisfaisante pour une fréquence de 600MHz mais dérive fortement pour les autres fréquences. En effet, lorsque l'on passe d'une fréquence de 300MHz à 1000MHz, le facteur multiplicateur est de 3.3 alors que la performance estimée n'est augmentée que 2 fois. Cela provient en partie du fait que les pénalités pour les accès mémoires (cache-miss) sont en temps et non en cycles processeur.
La figure 5.32 montre au contraire que nos estimations, basées sur des modèles gros grain, ont une erreur comprise entre 9 et 13%. Ces résultats montrent que notre approche permet de fournir des estimations respectant la limite qui avait été fixée au début de ces travaux. reste assez constante (entre 9 et 13%) et ne dépasse jamais la limite autorisée.

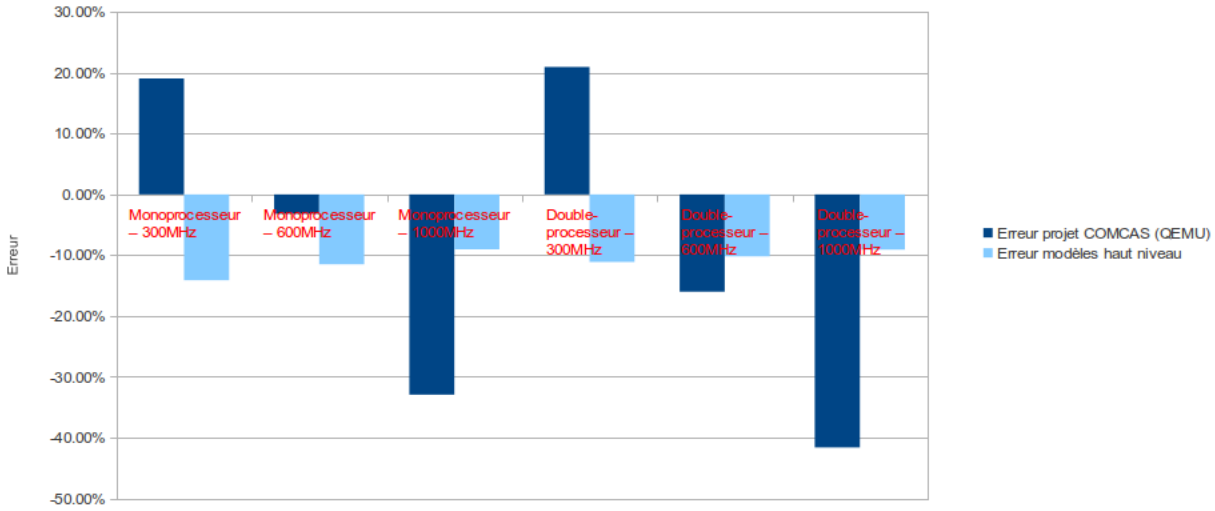

FIGURE 5.29: Comparaison de l'erreur d'estimation de la plate-forme QEMU et de notre approche haut niveau.

5.4 Conclusion

Pour conclure, les multiples comparaisons nous ont permis de valider notre méthodologie et les modèles utilisés. Pour l'estimation de la performance, l'erreur moyenne avoisine les 10% alors que l'erreur maximale est de 17% ce qui correspond à nos attentes. Six applications ont été testées ainsi que six plates-formes matérielles, ce qui permet de couvrir un large spectre de systèmes.

L'estimation de consommation d'énergie a posé plus de difficultés. En effet, le modèle fin grain ne nous a pas permis d'obtenir des résultats d'estimations respectant nos contraintes de précisions. Nous avons avancé l'hypothèse que des paramètres architecturaux significatifs manquaient à nos modèles. Une autre hypothèse serait que la calibration initiale est perfectible.
Le modèle gros grain a quant à lui montré de bon résultats. En effet, une erreur maximale de 19% et une erreur moyenne de 12% ont été observé pour les différentes applications testées.

Nous avons montré au cours de ces expérimentations que les modèles très fins sont souvent très compliqués à mettre en place et peuvent mener à des erreurs d'estimations importantes. D'un autre coté, les modèles haut niveau sont plus aisés et rapide à mettre en oeuvre et ont moins de chance que l'erreur maximale dérive. Par contre, l'erreur moyenne sera logiquement plus élevé qu'un bon modèle à grain fin.

Un avantage certain de FORECAST est sa rapidité de mise en oeuvre et d'exécution. En effet, le fait d'utiliser de la génération de code avec exécution sur un ordinateur hôte permet d'effectuer une simulation en 6 secondes environ quelque soit l'application et la plate-forme. Cette rapidité de simulation a été exploité pour l'exploration d'architecture qui n'aurait pas été possible si la durée d'une simulation avaient été de plusieurs minutes.
Dans les expérimentations présentées dans ce chapitre, les explorations auront demandé entre 1 et 3 minutes pour trouver une solution respectant les contraintes. Ceci n'aurait pas été possible avec des approches à base d'ISS par exemple du fait du temps de simulation important de ces approches.

Chapitre 6

Conclusion et perspectives

6.1 Bilan

La décision d'une architecture matérielle et logicielle lors du lancement d'un nouveau projet est une tâche qui peut s'avérer fastidieuse et complexe. Pour assister les architectes systèmes et logiciels dans les phases de conception, il existe aujourd'hui de nombreux outils. Dans certains d'entre eux, on commence à trouver des fonctionnalités permettant d'obtenir des estimations de performance et de consommation d'énergie. Malheureusement, comme nous l'avons vu, la plupart possèdent des inconvénients majeurs comme le temps de prise en main, le temps de développement des modèles, le temps de simulation, la précision des résultats, ou encore la richesse de composants déjà prédisponible. On note également que les estimations de performance et la consommation ne sont souvent pas réunis dans un seul outil. C'est dans ce contexte et au vu de ce constat que nous avons développé une méthodologie et des outils associés permettant d'évaluer différentes conceptions logicielles et matérielles.

L'approche qui a été proposée repose sur un langage de description haut-niveau (qui a été étendu), qui se veut simple d'utilisation permettant à la fois de décrire une application mais aussi une plate-forme matérielle. Ensuite, un flot d'estimations évalue le temps d'exécution de chaque tâche puis FORECAST effectue une exécution dynamique du modèle.
Grâce à la simulation et aux possibilités de l'ordonnanceur de la machine hôte (systèmes de préemptions, de priorités et de parallélismes), il est alors possible d'observer le comportement dynamique du système complet.

Les estimations sont basées sur des paramètres qu'il est facilement possible d'obtenir, que ce soit pour le matériel ou le logiciel. Par exemple, du coté logiciel il est nécessaire de fournir le nombre d'instructions ou d'accès mémoire, ce qui est faisable avec un outil de profiling, et du coté matériel les paramètres sont présents dans les datasheets constructeurs (fréquence, taille de pipeline, DMIPS...).

Grâce à l'utilisation du standard POSIX et de logiciels libres (Gnuplot, Valgrind), nous nous assurons une compatibilité et une ré-utilisabilité sur d'autres plates-formes. En effet, nous ne souhaitions pas utiliser de logiciel propriétaire qui nécessiterait des licences ou ne serait pas compatible d'un ordinateur à l'autre.

Nous avons été capables de valider le bon fonctionnement de la méthodologie grâce à différentes applications, dont nous avons comparé les estimations avec les valeurs réelles de performance. Les outils ont aussi été utilisés dans le cadre d'un projet d'étude interne à Thales Communications and Security. Enfin, nous avons comparé notre approche à deux projets de recherche COMCAS (projet Européen) et Open-PEOPLE (projet ANR).
Tout ceci a permis de voir que par rapport aux objectifs fixés (outil rapide et simple à mettre en oeuvre dont

l'erreur n'excède pas 20%), notre méthodologie fonctionne correctement. D'une part la modélisation se fait rapidement et simplement et l'exécution est rapide (environ 5 secondes). D'autre part, l'erreur d'estimation reste correcte dans tous les cas présentés avec une erreur moyenne autour de 10% et une erreur maximale de 17%. Pour le cas du multi-coeurs, des expérimentations plus poussées sont nécessaires afin de valider totalement les premiers résultats satisfaisants obtenus.

Par la suite, un explorateur a aussi été développé afin d'ajouter la possibilité d'effectuer automatiquement des itérations permettant de trouver le meilleur compromis pour un système logiciel/matériel. En effet, grâce à la spécification de contraintes (taux de charge des processeurs, temps maximum d'exécution de certaines tâches) l'explorateur va exécuter des itérations en essayant de trouver le meilleur compromis de répartition des tâches sur le système avec les fréquences d'exécution les plus faibles afin de consommer le moins d'énergie possible.
Il est de plus tout à fait possible de créer d'autres algorithmes d'exploration étant donné que notre simulateur ressort un grand nombre d'informations utiles (taux de charge des processeurs, nombre d'instructions et d'accès mémoire de chaque tâche, temps d'exécution de chaque tâche...).

Le coeur de FORECAST étant un générateur de code exécutable, il permet aussi de générer des benchmarks utilisables directement sur des plates-formes embarquées à partir de modèles haut niveau [84]. Cela permet de générer des applications de test fin d'évaluer différentes plates-formes facilement sans avoir à créer d'applications réelles. Il est aussi possible d'évaluer différentes architectures logicielles (parallélisme, priorité des tâches, affinité processeur) afin de déterminer la plus adaptée à la plate-forme.

Pour conclure, le choix d'utiliser un langage haut niveau afin de modéliser le système, couplé à de la génération de code exécutable s'avère un bon choix pour l'estimation de performance et consommation en phase amont d'un projet. Ceci facilite le choix des architectes tout en étant rapide à mettre en place grâce à la possibilité de créer des bibliothèques de composants logiciels et matériels.
Un des problèmes de la méthodologie réside dans le fait d'utiliser l'ordonnanceur présent sur la machine exécutant la simulation (ordonnanceur Linux étant souvent par priorité) ce qui nous empêche d'utiliser des ordonnanceurs exotiques. Cette limitation n'a cependant pas été gênante dans notre contexte car dans les produits qui ont été prospectés, la plupart utilisent un ordonnanceur par priorité.
De plus, nous utilisons les coeurs de la machine hôte afin de simuler le fonctionnement des coeurs de la plate-forme embarquée, ce qui limite le nombre de processeurs que l'on peut simuler. Mais les serveurs 8 ou 16 coeurs étant de plus en plus présent dans les entreprises/laboratoires et les plates-formes embarqués n'ayant souvent que 2 ou 4 coeurs, il reste encore de la marge avant d'atteindre les limites de l'approche. De plus, les cibles de cette thèse sont les plates-formes mono-coeur et multi-coeurs, et non pas les plates-formes many-coeurs ou GPU.

Des améliorations peuvent être proposées à la suite des travaux de la thèse, en particulier l'ajout de modèle pour des unités de calcul de type DSP et la consolidation des modèles de consommation d'énergie.

6.2 Perspectives

Afin de pérenniser FORECAST et de permettre son utilisation par le plus grand nombre, une première perspective réside dans le fait d'enrichir la bibliothèque de composants logiciels (différentes tâches de base utilisées dans les applications) et de composants matériels (les différents processeurs utilisés dans les produits). Ceci nous permettrait aussi de posséder un plus large choix de composants pré-enregistrés lors de futurs études.

L'amélioration de l'erreur d'estimation pourrait se faire en prenant en compte le système d'exploitation (OS) comme développé dans [109]. En effet, l'OS utilise une part des ressources de calcul (pour passer d'une tâche à l'autre par exemple) et consomme de l'énergie à chaque appel système.

Dans la perspective d'utiliser un plus large spectre de plate-forme et afin d'ajouter des plates-formes

hétérogènes, il serait intéressant d'implémenter un flot d'estimation pour les DSP. On pourrait s'inspirer des travaux de l'outil SoftExplorer [130] qui permet d'évaluer la consommation d'énergie de différents DSP à l'aide de paramètres logiciels et matériels. De plus, FORECAST a été créé de telle manière qu'il est possible simplement de rajouter des nouveaux types d'unité de calcul.
Il serait aussi intéressant de rajouter la consommation due aux communications inter-process pour les multi-coeurs hétérogènes. En effet, Chun-Hao Hsu [70] montre une modélisation de l'énergie consommée par un OMAP5912 (double-coeur hétérogène) avec une précision de moins de 5%. Il a été nécessaire pour cela, de modéliser à la fois la consommation du processeur ARM, du DSP mais aussi de la communication inter-process.

Enfin, l'exploration des différentes architectures possibles pourrait être améliorée à l'aide d'algorithmes plus complexes prenant en compte d'autres métriques comme la consommation d'énergie (pour l'instant la recherche étant orientée afin de minimiser la fréquence et donc la consommation d'énergie mais la consommation n'étant pas un objectif à part entière), le type de processeur (et la difficulté d'implémentation du code), etc... FORECAST pourrait alors être le coeur de différents explorateurs qui pourraient être sélectionnés suivant ce que l'on cherche à optimiser.

Chapitre 7

Publications et autre participations

[SAM10] Joffrey Kriegel, Florian Broekaert Alain Pegatoquet, Michel Auguin, "Power optimization technique applied to real-time video application". In *SAME Forum*, Sophia-Antipolis, France, October 2010. Demonstration and poster.

[NAN10] Joffrey Kriegel, Marius Gligor, Fabien Colas-Bigey, Frédéric Petrot "QEMU-based virtual prototyping", In *Nano Electronic Forum*, Madrid, Spain, November 2010. Demonstration and poster.

[WUP11] Joffrey Kriegel, Florian Broekaert, Alain Pegatoquet, Michel Auguin "Power optimization technique applied to real-time video application", In *Workshop on Ultra-Low Power Sensor Networks (WUPS)*, Como, Italy, February 2011. Tutorial and demonstration.

[DAC11] Joffrey Kriegel, Alain Pegatoquet, Florian Broekaert, Michel Auguin "A High-Level Benchmarks Generator for Multi-Core Platforms Running Real-Time Applications", In *Design and Automation Conference (DAC)*, San Diego, United States, June 2011. Work In Progress.

[DOC11] Joffrey Kriegel, Alain Pegatoquet, Florian Broekaert, Michel Auguin "A Performance Estimation Flow for Embedded Systems with Mixed Software/Hardware Modeling", In *Journée des doctorants du LEAT*, Sophia Antipolis, France, June 2011. Paper.

[SAM11] Joffrey Kriegel, Alain Pegatoquet, Michel Auguin, Florian Broekaert "A Performance Estimation Flow for Embedded Systems with Mixed Software/Hardware Modeling", In *International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS)*, Samos, Greece, July 2011. Paper.

[JEL11] Joffrey Kriegel, Alain Pegatoquet, Florian Broekaert, Michel Auguin "A Performance Estimation Flow for Embedded Systems with Mixed Software/Hardware Modeling", In *Journées électroniques 2011*, Montpellier, France, October 2011. Poster.

[EWL12] Joffrey Kriegel, Alain Pegatoquet, Florian Broekaert, Michel Auguin "Waveperf : A Benchmark Generator for Performance Evaluation", In *Embedded with Linux (EwiLi)*, Lorient, France, June 2012. Paper.

[NEW12] Joffrey Kriegel, Alain Pegatoquet, Florian Broekaert, Michel Auguin "A High Level Mixed Hardware/Software Modeling Framework for Rapid Performance Estimation", In *10th IEEE International NEWCAS Conference*, Montreal, Canada, June 2012. Paper.

Annexes

Nom du Symbole	Signification
[NomDuBloc]	Nom choisi par l'utilisateur pour nommer son bloc
[NomDuSignalEntrant]	Nom choisi par l'utilisateur pour un signal de communication arrivant vers le bloc
[NomDuSignalSortant]	Nom choisi par l'utilisateur pour un signal de communication repartant du bloc
[NomDuBehaviourDuBloc]	Nom choisi par l'utilisateur pour nommer le comportement de son bloc
[NumEtat]	Numéro de l'état dans lequel effectuer la ligne dans le cas d'une machine d'état. S'il n'y a pas de machine d'état, ce paramètre vaut 1 par défaut.
[NbDExecution]	Indique le nombre de fois que doit être exécutée l'action qui suit
[typeOperation]	Décrit si les spécifications qui suivent seront décrites en temps ou en opération dhrystone. Peut prendre les valeurs : Timing_in_ms — Dhrystone_number_ops — Sleep_in_ms
[NomDesCaracteristiquesDuBloc]	Nom choisi par l'utilisateur pour nommer les caractéristiques de son bloc
[TempsAvantSignal]	Dans le cas de Timing_in_ms ou Sleep_in_ms : Temps écoulé entre l'arrivée du signal entrant et l'envoi du signal sortant Remplacé par un nombre d'opération Dhrystone le cas échéant
[TempsApresSignal]	Dans le cas de Timing_in_ms ou Sleep_in_ms : Temps écoulé entre après l'envoi du signal sortant et avant l'appel du signal suivant Remplacé par un nombre d'opération Dhrystone le cas échéant
var	Signal la déclaration d'une variable
[typeVar]	Type de la variable à déclarer. Exemple : int
[NomDeLaVariable]	Nom choisi par l'utilisateur pour nommer sa variable
init	Signal l'initialisation d'une variable
[ValeurInitDeLaVariable]	Valeur choisie par l'utilisateur pour initialiser sa variable. Exemple : 0 ou NULL
state	Indique la déclaration des états de la machine d'état
[NomEtat]	Nom choisi par l'utilisateur pour nommer son état de machine d'état
_initial_state_	Indique quel état est choisi pour démarrer la machine d'état
[Condition]	Condition sur la variable qui va déterminer le passage ou non à l'état suivant, indiqué après]-¿. Exemple : !=0 ou ¡10 ou ==2
[affectation]	Affectation d'une valeur à la variable après l'exécution de l'état et avant de passer à l'état suivant. Ne dépend pas de l'exécution de l'état suivant. Exemple : = 2 ou ++

TABLE 7.1: Description des différents éléments présents dans la description des blocs logiciels.

Nom du Symbole	Signification
[nomDuFichierBloc]	Fichier .txt contenant la description du composant (bloc)
[nomDuBehaviourDuBloc]	Nom donnée à la partie behaviour du composant dans son fichier de desciption
[nomDeLInstanceDuBloc]	Nom de l'instance du bloc dans ce fichier de description d'architecture. Ce nom peut être choisi arbitrairement.
[NomDesCaracteristiquesDuBloc]	Nom donné à la partie characteristics du composant dans son fichier de description
[typeConnection]	Type de la connexion entre deux E/S de deux blocs. Peut prendre les valeurs : synchronous ou asynchronous ou deferred_synchronous
[nomDeLaConnection]	Nom de la connexion dans ce fichier de description d'architecture. Ce nom peut être choisi arbitrairement.
[NomDuneSortie]	Nom d'une E/S de type uses dans le fichier de description du composant
[NomDuneEntree]	Nom d'une E/S de type provides dans le fichier de description du composant
[NomDuPortDeSynchronisation]	Uniquement dans le cas d'une connexion deferred_synchronous Indique sur quel port du premier bloc de la connexion doit se faire la synchronisation. Permet d'attendre le résultat d'un second bloc et de se synchroniser avec lui.
Raw_ip_interface	Composant optionnel particulier défini par défaut. Permet de réceptionner ou générer du trafic réseau
ip_interface	Nom des characteristics de Raw_ip_interface
configure_priority_and_sched_fifo	Fonction devant toujours être appelée après l'instanciation de Raw_ip_interface ou d'une connexion asynchrone ou d'une connexion deferred_synchronous. Défini la politique d'ordonnancement et la priorité du thread réseau
[prioritee]	Nombre entier définissant la priorité du thread
[boolFifo]	Booleen définissant si la politique d'ordonnancement du thread est SCHED_FIFO ou non
Timer_impl	Composant optionnel particulier définit par défaut. Permet d'implémenter un timer extérieur au système, pour le déclancher par exemple.
timer	Nom des characteristics de Timer_impl
configure_timerspec_and_sched_fifo	Fonction devant toujours être appelée après l'instanciation de Timer_impl
[TempsDepartSec]	Entier représentant le temps en seconde avant le premier tick timer
[TempsDepartNSec]	Entier représentant le temps en nano seconde avant le premier tick timer
[IntervalSec]	Entier représentant le temps en seconde entre deux coups de timer
[IntervalNsec]	Entier représentant le temps en nano seconde entre deux coups de timer
entry_point	Définit le point entrée du système, c'est-à-dire sur quelle entrée d'un bloc va être envoyé un signal pour démarrer le système
final_point	Définit le point de sortie du système, c'est-à-dire sur quelle entrée d'un bloc va être envoyé un signal pour arrêter le système. Envoyé automatiquement au bout de 3 secondes.

TABLE 7.2: Description des différents éléments présent dans la description de l'architecture logiciels.

Table des figures

Liste des tableaux

Listings

Bibliographie

[1] I. Ahmad, M. Dhodhi, and C. Chen. "Integrated scheduling, allocation and module selection for design-space exploration in high-level synthesis." *IEE Proceedings - Computers and Digital Techniques*, 142(1) :65–71, Jan. 1995.

[2] G. Ascia, V. Catania, and M. Palesi. "Design space exploration methodologies for IP-based system-on-a-chip." In *IEEE International Symposium on Circuits and Systems*, volume 2, pages 364–367, May 2002.

[3] G. Ascia, V. Catania, and M. Palesi. "A framework for design space exploration of parameterized VLSI systems." In *ASP-DAC/VLSI Design 2002*, pages 245–250, Jan. 2002.

[4] R. B. Atitallah, E. Senn, D. Chillet, M. Lanoe and D. Blouin. "An efficient Framework for Power-Aware Design of Heterogeneous MPSoC". *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, © IEEE PRESS*, accepted for publication, 2012.

[5] M. Auguin, L. Capella, F. Cuesta, and E. Gresset. "CODEF : a system level design space exploration tool." In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 1145–1148, 2001.

[6] J. Axelsson. "Architecture synthesis and partitioning of real-time systems : a comparison of three heuristic search strategies." In *5th Int. Workshop on Hardware/Software Codesign (CODES/CASHE)*, pages 161–165, Mar. 1997.

[7] A. Baghdadi, N. Zergainoh, W. Cesario, T. Roudier, and A. Jerraya. "Design space exploration for hardware/software codesign of multiprocessor systems." In *11th International Workshop on Rapid System Prototyping (RSP)*, pages 8–13, 2000.

[8] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. "Hardware-Software Co-Design of Embedded Systems : The Polis Approach." Number 404 in *International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 1997.

[9] F. Balarin, M. Chiodo, A. Jurecska, L. Lavagno, B. Tabbara, and A. Sangiovanni-Vincentelli. "Automatic generation of a real-time operating system for embedded systems". In *5th International Workshop on Hardware/Software Co-Design (Codes/CASHE)*, Mar. 1997.

[10] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Paserone, and A. Sangiovanni-Vincentelli. "Metropolis : an integrated electronic system design environment". *IEEE Computer*, 36(4) :45–52, Apr. 2003.

[11] BeagleBoard :http ://beagleboard.org/

[12] M. Becker, G. Di Guglielmo, F. Fummi, W. Mueller, G. Pravadelli, T. Xie "RTOS-Aware Refinement for TLM2.0-based HW/SW Designs", In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2010

[13] Bellard Fabrice, "QEMU, a Fast and Portable Dynamic Translator", *FREENIX Track : 2005 USENIX Annual Technical Conference*.

[14] G. Berry and G. Gonthier. "The Esterel Synchronous Programming Language : Design, Semantics, Implementation." *Science of Computer Programming*, 19(2) :87–152, 1992.

[15] T. Blickle, J. Teich, and L. Thiele. "System-level synthesis using evolutionary algorithms." *Design Automation for Embedded Systems, Kluwer Academic Publishers*, 3(1) :23–58, Jan. 1998.

[16] S. Blythe and R. Walker. "Toward a practical methodology for completely characterizing the optimal design space." In *9th International Symposium on System Synthesis*, pages 8–13, Nov. 1996.

[17] S. Blythe and R. Walker. "Efficiently searching the optimal design space." In *Ninth Great Lakes Symposium on VLSI*, pages 192–195. IEEE Comput. Soc, Mar. 1999.

[18] S. Blythe and R. Walker. "Efficient optimal design space characterization methodologies." *ACM Transactions on Design Automation of Electronic Systems*, 5(3) :322–336, July 2000.

[19] D. Brooks, V. Tiwari and M. Martonosi . "Wattch : a framework for architectural-level power analysis and optimizations." *In Proceedings of the 27th International Symposium on Computer Architecture, 2000.*

[20] D. Bruni and A. B. L. Benini. "Statistical design space exploration for application-specific unit synthesis." In *38th Design Automation Conference (DAC)*, pages 641–646, 2001.

[21] D. Burger and T. M. Austin. "The SimpleScalar tool set, version 2.0." Technical Report 1342, Computer Sciences Department, University of Wisconsin-Madison, June 1997.

[22] L. Cai, D. Gajski, and M. Olivarez. "Introduction of system level architecture exploration using the SpecC methodology." In *IEEE International Symposium on Circuits and Systems*, volume 5, pages 9–12, May 2001.

[23] L. Cai, S. Verma, and D. D. Gajski. "Comparison of SpecC and SystemC languages for system design." Technical Report CECS 03-11, University of California, Irvine, May 2003.

[24] S. Cho and Y. Kim "Linux BYTEmark Benchmarks : A Performance Comparison of Embedded Mobile Processors", *IEEE The 9th International Conference on Advanced Communication Technology*, Feb. 2007

[25] S. Chakraborty, S. Künzli, and L. Thiele. "A general framework for analysing system properties in platform-based embedded system design." In *Design, Automation and Test in Europe (DATE)*, Munich, Germany, Mar. 2003.

[26] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. "Performance evaluation of network processor architectures : Combining simulation with analytical estimation." *Computer Networks, Elsevier Science*, 41(5) :641–665, Apr. 2003.

[27] P. Chandra, F. Hady, R. Yavatkar, T. Bock, M. Cabot, and P. Mathew. "Benchmarking network processors." In P. Crowley, M. Franklin, H. Hadimioglu, and P. Onufryk, editors, *Network Processor Design : Issues and Practices*, volume 1, pages 11–25. Morgan Kaufmann Publishers, Oct. 2002.

[28] K. Chatha and R. Vemuri. "An iterative algorithm for hardware-software partitioning, hardware design space exploration and scheduling." *Design Automation for Embedded Systems, Kluwer Academic Publishers*, 5(3-4) :281–293, Aug. 2000.

[29] S. Chaudhuri, S. Blythe, and R. Walker. "A solution methodology for exact design space exploration in a three-dimensional design space." In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 5, pages 69–81, Mar. 1997.

[30] D. Culler, J.P. Singh, and A. Gupta. "Parallel Computer Architecture : A Hardware/Software Approach." *Morgan Kaufmann, 1st edition, August 1998.* The Morgan Kaufmann Series in Computer Architecture and Design.

[31] Projet COMCAS http ://www.comcas.eu/news.html

[32] E. A. de Kock, G. Essink, W. J. M. Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, P. Lieverse, and K. A. Vissers. "YAPI : Application modeling for signal processing systems." In *37th Design Automation Conference (DAC)*, pages 402–405, June 2000.

[33] B. De Smedt and G. Gielen. "WATSON : a multi-objective design space exploration tool for analog and RF IC design." In *IEEE 2002 Custom Integrated Circuits Conference*, pages 31–34, 2002.

[34] R. P. Dick and N. K. Jha. "MOCSYN : Multiobjective core-based single-chip system synthesis." In *Design, Automation and Test in Europe Conference (DATE)*, pages 263–270, 1999.

[35] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha "Analysis of Power Dissipation in Embedded Systems Using Real-Time Operating Systems", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and systems*, Vol. 22, No. 5, May 2003

[36] R. Dutta, J. Roy, and R. Vemuri. "Distributed design space exploration for high-level synthesis systems." In *29th Design Automation Conference (DAC)*, pages 644–650, June 1992.

[37] DSPBIOS/LINK software, Texas Instruments. Dallas, Texas. http ://processors.wiki.ti.com/index.php/Category :BIOSLink

[38] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli. "Design of Embedded Systems : Formal Models, Validation, and Synthesis." In *Proceedings of the IEEE 85(3)*, pages 366–390, March 1997.

[39] L. Eeckhout, K. de Bosschere, and H. Neefs. "Performance analysis through synthetic trace generation." In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 1–6, 2000.

[40] EEMBC : http ://www.eembc.org/home.php

[41] A. Fauth, J. Van Praet, and M. Freericks. "Describing instruction set processors using nML." In *European Design and Test Conference (ED&TC)*, pages 503–507, Mar. 1995.

[42] A. Ferrari and A. Sangiovanni-Vincentelli. "System design : Traditional concepts and new paradigms." In *International Conference on Computer Design (ICCD)*, pages 2–12, Oct. 1999.

[43] W. Fornaciari, D. Sciuto, C. Silvano, and V. Zaccaria. "A design framework to efficiently explore energy-delay tradeoffs." In *Ninth International Symposium on Hardware/Software Codesign (CODES)*, pages 260–265, 2001.

[44] W. Fornaciari, D. Sciuto, C. Silvano, and V. Zaccaria. "A sensitivity-based design space exploration methodology for embedded systems." *Design Automation for Embedded Systems*, Kluwer Academic Publishers, 7(1-2), Sept. 2002.

[45] M. A. Franklin and T. Wolf. "A network processor performance and design model with benchmark parameterization." In P. Crowley, M. Franklin, H. Hadimioglu, and P. Onufryk, editors, *Network Processor Design : Issues and Practices*, volume 1, pages 117–139. Morgan Kaufmann Publishers, Oct. 2002.

[46] M. A. Franklin and T. Wolf. "Power considerations in network processor design." In *Second Workshop on Network Processors at the 9th International Symposium on High Performance Computer Architecture (HPCA9)*, Feb. 2003.

[47] C. A. Furia, D. Mandrioli, A. Morzenti, and M. Rossi. "Modeling time in computing : A taxonomy and a comparative survey." *ACM Comput. Surv.*, 42(2) :1–59, 2010.

[48] D. Gajski, F. Vahid, S. Narayan, and J. Gong. "System-level exploration with SpecSyn." In *35th Design and Automation Conference (DAC)*, pages 812–817, June 1998.

[49] L. Gauthier, S. Yoo, and A. Jerraya. "Automatic generation and targeting of application specific operating systems and embedded systems software." In *Design, Automation and Test in Europe (DATE)*, pages 679–685, Mar. 2001.

[50] T. Givargis, J. Henkel, and F. Vahid. "Interface and cache power exploration for core-based embedded system design." In *International Conference on Computer-Aided Design (ICCAD)*, Nov. 1999.

[51] T. Givargis, F. Vahid, and J. Henkel. "System-level exploration for Pareto-optimal configurations in parameterized system-on-a-chip." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4) :416–422, Aug. 2002.

[52] M. Gligor, N. Fournel, F. Pétrot "Using Binary Translation in Event Driven Simulation for Fast and Flexible MPSoC Simulation". In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. Grenoble, France, October 2009.

[53] F. Glover and M. Laguna. "Tabu Search." *Kluwer Academic Publishers*, July 1997.

[54] S. Graham, P. Kessler, M. McKusick. "gprof : A Call Graph Execution Profiler". In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No 6, pp. 120-126, June 1982.

[55] M. Gries. Algorithm-Architecture Trade-offs in Network Processor Design. PhD thesis, Diss. ETH No. 14191, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, July 2001.

[56] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. "Comparing analytical modeling with simulation for network processors : A case study." In *Design, Automation and Test in Europe (DATE)*, Munich, Germany, Mar. 2003.

[57] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. "Exploring trade-offs in performance and programmability of processing element topologies for network processors." In *Second Workshop on Network Processors at the 9th International Symposium on High Performance Computer Architecture (HPCA9)*, Mar. 2003.

[58] M. Gries "Methods for Evaluating and Covering the Design Space during Early Design Development", *the VLSI Journal*, 2003.

[59] T. Grotker, S. Liao, G. Martin, and S. Swan. "System Design with SystemC." *Kluwer Academic Publishers*, May 2002.

[60] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. "MiBench : A free, commercially representative embedded benchmark suite." In *IEEE 4th Annual Workshop on Workload Characterization*, pages 3–14, Dec. 2001.

[61] G726 http ://www.itu.int/rec/T-REC-G.726

[62] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau. "EXPRESSION : A language for architecture exploration through compiler/simulator retargetability." In *Design, Automation and Test in Europe (DATE)*, pages 485–490, 1999.

[63] T. Harriss, R. Walke, B. Kienhuis, and E. Deprettere. "Compilation from Matlab to process networks realized in FPGA." *Design Automation for Embedded Systems*, Kluwer, 7(4) :385–403, Nov. 2002.

[64] C. Haubelt, J. Teich, K. Richter, and R. Ernst. "System design for flexibility." In *Design, Automation and Test in Europe (DATE)*, pages 854–861, Mar. 2002.

[65] G. Hekstra, G. L. Hei, P. Bingley, and F. Sijstermans. "TriMedia CPU64 design space exploration." In 1999 *IEEE International Conference on Computer Design : VLSI in Computers and Processors*, pages 599–606, 1999.

[66] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, and H. Meyr. "A methodology for the design of application specific instruction set processors (ASIP) using the machine description language LISA." In *International Conference on Computer Aided Design (ICCAD)*, San Jose, CA, Nov. 2001.

[67] J. Horn. "Multicriterion decision making." In T. Bäck, D. Forgel, and Z. Michalewicz, editors, Handbook of Evolutionary Computation. *Institute of Physics Publishing*, Bristol, UK, 1997.

[68] H. Hsieh, F. Balarim, L. Lavagno, and A. Sangiovanni-Vincentelli. "Efficient methods for embedded system design space exploration." In *37th Design Automation Conference (DAC)*, pages 607–612, 2000.

[69] X. Hu, G. Greenwood, S. Ravichandran, and G. Quan. "A framework for user assisted design space exploration." In *36th Design Automation Conference (DAC)*, pages 414–419, 1999.

[70] C.-H. Hsu, J. J. Chen, and S.-L. Tsao "Evaluation and Modeling of Power Consumption of a Heterogeneous Dual-Core Processor", In *International Conference on Parallel and Distributed Systems*, 2007

[71] C. A. R. Hoare. "Communicating Sequential Processes." *Commun. ACM*, 21(8) :666–677, 1978.

[72] D. Harel and A. Naamad. "The STATEMATE Semantics of State-charts." *ACM Trans. Softw. Eng. Methodol.*, 5(4) :293–333, 1996.

[73] N. Halbwachs, F. Lagnier, and C. Ratel. "Programming and Verifying Real-Time Systems by Means of the Synchronous Data-Flow Language LUSTRE." *IEEE Trans. Softw. Eng.*, 18(9) :785–793, 1992.

[74] ITRS. "2009 Report - System Drivers." Technical report, International Technology Roadmap for Semiconductors, 2009

[75] A. Jantsch and I. Sander. "Models of Computation and Languages for Embedded System Design." *IEEE Proceedings on Computers and Digital Techniques*, 152(2) :114–129, March 2005. Special issue on Embedded Microelectronic Systems ; Invited paper.

[76] A. Jantsch. "Models of Embedded Computation." In *Richard Zurawski, editor, Embedded Systems Handbook*. CRC Press, 2005. Invited contribution.

[77] G. Kahn. "The semantics of a simple language for parallel programming." In *Proceedings of the IFIP Congress*, pages 471–475. North-Holland Publishing Co., 1974.

[78] I. Karkowski and H. Corporaal. "Design space exploration algorithm for heterogeneous multi-processor embedded system design." In *35th Design and Automation Conference (DAC)*, pages 82–87, 1998.

[79] V. Kathail, S. Aditya, R. Schreiber, B. R. Rau, D. Cronquist, and M. Sivaraman. "PICO : automatically designing custom computers." *IEEE Computer*, 35(9) :39–47, Sept. 2002.

[80] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf. "An approach for quantitative analysis of application-specific dataflow architectures." In *Application-Specific Systems, Architectures*, and Processors (ASAP), July 1997.

[81] J. Kin, C. Lee, W. Mangione-Smith, and M. Potkonjak. "Power efficient mediaprocessors : design space exploration." In *36th Design Automation Conference (DAC)*, pages 321–326, 1999.

[82] I. Karkowski and H. Corporaal. "Design space exploration algorithm for heterogeneous multi-processor embedded system design." In *35th Design and Automation Conference (DAC)*, pages 82–87, 1998.

[83] J. Kriegel, A. Pegatoquet, M. Auguin, F. Broekaert "A Performance Estimation Flow for Embedded Systems with Mixed Software/Hardware Modeling", In *International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS)*, Samos, Greece, July 2011.

[84] J. Kriegel, F. Broekaert, M. Auguin, A. Pegatoquet "Waveperf : A Benchmark Generator for Performance Evaluation", In *Embedded with Linux (EwiLi)*, Lorient, France, June 2012.

[85] C. Lee, M. Potkonjak, and W. Mangione-Smith. "MediaBench : a tool for evaluating and synthesizing multimedia and communications systems." In *Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture*, pages 330–335, Dec. 1997.

[86] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere. "A methodology for architecture exploration of heterogeneous signal processing systems." *Kluwer Journal of VLSI Signal Processing*, 29(3) :197–207, Nov. 2001.

[87] R. Leupers and P. Marwedel. "Retargetable Compiler Technology for Embedded Systems - Tools and Applications." *Kluwer Academic Publishers*, Oct. 2001.

[88] D. Lanneer, J. Van Praet, A. Kifli, K. Schoofs, W. Geurts, F. Thoen, and G. Goossens. "CHESS : Retargetable code generation for embedded DSP processors." In P. Marwedel and G. Goossens, editors, *Code Generation for Embedded Processors*, volume 317 of SECS, pages 85–102. Kluwer Academic Publishers, 1995.

[89] C. Liem and P. Paulin. "Compilation techniques and tools for embedded processor architectures." In J. Staunstrup and W. Wolf, editors, *Hardware/Software Co-Design : Principles and Practise*. Kluwer Academic Publishers, 1997.

[90] Y.-T. S. Li, S. Malik, and A. Wolfe. "Performance estimation of embedded software with instruction cache modeling." *ACM Transactions on Design Automation of Electronic Systems*, 4(3) :257–279, July 1999.

[91] J.-Y. Le Boudec and P. Thiran. "Network Calculus : A Theory of Deterministic Queuing Systems for the Internet." *Number 2050 in LNCS*. Springer Verlag, 2001.

[92] K. Lahiri, A. Raghunathan, and S. Dey. "System-level performance analysis for designing on-chip communication architectures." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(6) :768–783, June 2001.

[93] K. Lahiri, A. Raghunathan, and S. Dey. "Efficient exploration of the SoC communication architecture design space." In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 424–430, Nov. 2000.

[94] K. Lahiri, A. Raghunathan, and S. Dey. "Performance analysis of systems with multi-channel communication architectures." In *Proceedings of 13th International Conference on VLSI Design*, pages 530–537, Jan. 2000.

[95] Jing-Wun Lin, Chen-Chieh Wang, Chin-Yao Chang, Chung-Ho Chen, Kuen-Jong Lee, Yuan-Hua, Chu, Jen-Chieh Yeh, and Ying-Chuan Hsiao "Full System Simulation and Verification Framework", 2009 *Fifth International Conference on Information Assurance and Security* IAS '09.

[96] E.A. Lee and D.G. Messerschmitt. "Synchronous Data Flown." *Proceedings of the IEEE*, 75(9) :1235–1245, September 1987.

[97] J. Laurent, N. Julien, and E. Martin. "Functional level power analysis : An efficient approach for modeling the power consumption of complex processors." In *Proceedings of the Design, Automation and Test in Europe Conference*, Munich, 2004.

[98] G. Memik, W. H. Mangione-Smith, and W. Hu. "NetBench : A benchmarking suite for network processors." In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2001.

[99] P. Mishra, N. Dutt, and A. Nicolau. "Functional abstraction driven design space exploration of heterogeneous programmable architectures." In *International Symposium on System Synthesis*, pages 256–261, Oct. 2001.

[100] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation." In *Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, June 2002.

[101] A. Mihal, C. Kulkarni, K. Vissers, M. Moskewicz, M. Tsai, N. Shah, S. Weber, Y. Jin, K. Keutzer, C. Sauer, and S. Malik. "Developing architectural platforms : A disciplined approach." *IEEE Design & Test of Computers*, 19(6) :6–16, 2002.

[102] N. Muralimanohar, R. Balasubramonian and N. P. Jouppi. "CACTI 6.0 : A Tool to Model Large Caches." In *in International Symposium on Microarchitecture*, Chicago, Dec 2007.

[103] M. Monton, A. Portero, M. Moreno, B. Martinez, J. Carrabina "Mixed SW/SystemC SoC Emulation Framework", In *IEEE International Symposium on Industrial Electronics (ISIE)*, 2007.

[104] J. Neel, P. Robert, J. Reed. "A Formal Methodology for Estimating the Feasible Processor Solution Space for a Software Radio." In *Proceeding of the SDR 05 Technical Conference and Product Exposition*, 2005.

[105] N. Nethercote. "Dynamic Binary Analysis and Instrumentation." *PhD Dissertation*, University of Cambridge, November 2004.

[106] OMAP3530 Application processor, Texas Instruments. Dallas, Texas. http ://focus.ti.com/docs/prod/folders/print/omap3530.html

[107] OMAP4430 Application processor, Texas Instruments. Dallas, Texas. http ://www.ti.com/lit/ml/swpt034b/swpt034b.pdf

[108] OMG. Systems Modeling Language (SysML) Specification. OMG document : ad/2006-03-08-01, version 1. Draft, April 2006.

[109] B. Ouni, C. Belleudy, and E. Senn. "Accurate Energy Characterization of OS Services in Embedded Systems". In *EURASIP Journal on Embedded Systems*, 2012.

[110] Open Virtual Platform Initiative. OVPsim instruction set simulators. available at : http ://www.ovpworld.org/.

[111] V. Pareto. Cours d'Economie Politique. *F.Rouge, Lausanne*, 1896.

[112] S. Pees, A. Hoffmann, and H. Meyr. Retargeting of compiled simulators for digital signal processors using a machine description language. In *Design, Automation and Test in Europe Conference (DATE)*, pages 669–673, 2000.

[113] S. Pees, A. Hoffmann, V. Zivojnovic, and H. Meyr. "LISA-machine description language for cycle-accurate models of programmable DSP architectures." In *36th Design Automation Conference (DAC)*, pages 933–938, June 1999.

[114] A. Pimentel, L. Hertzberger, P. Lieverse, P. van der Wolf, and E. Deprettere. "Exploring embedded-systems architectures with Artemis." *IEEE Computer*, 34(11) :57–63, Nov. 2001.

[115] A. Pimentel, S. Polstra, F. Terpstra, A. van Halderen, J. Coffland, and L. Hertzberger. "Towards efficient design space exploration of heterogeneous embedded media systems." In *Embedded processor design challenges. Systems, architectures, modeling, and simulation - SAMOS*, volume 2268 of LNCS, pages 57–73. Springer-Verlag, 2002.

[116] PandaBoard :http ://pandaboard.org/

[117] Patterson, http ://www.hpts.ws/papers/2007/TechTrendsHPTSPatterson2007.ppt

[118] G. Qu, N. Kawabe, K. Usaini, and M. Potkonjak "Function-Level Power Estimation Methodology for Microprocessors", *IEEE Design Automation Conference*, June 2000

[119] QEMU : http ://wiki.qemu.org.

[120] M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod. "Using the SimOS machine simulator to study complex computer systems." *ACM Transactions on Modeling and Computer Simulation*, 7(1) :78–103, Jan. 1997.

[121] D. S. Rao and F. Kurdahi. "Hierarchical design space exploration for a class of digital systems." IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 1(3) :282–295, Sept. 1993.

[122] L. Rioux, T. Saunier, S. Gerard, A. Radermacher, R. De Simone, T. Gauthier, Y. Sorel, J. Forget, J.-L. Dekeyser, A. Cuccuru, C. Dumoulin, and C. Andrè. "MARTE : A New OMG Profile RFP for the Modeling and Analysis of Real-Time Embedded Systems." In *DAC 2005 Workshop UML for SoC Design*, UML-SoC'05, June 2005.

[123] A. Rose, S. Swan, J. Pierce, and J.-M. Fernandez. "OSCI White paper : Transaction Level Modeling in SystemC ." available at : http ://www.systemc.org.

[124] S. K. Rethinagiri, R. B. Atitallah, E. Senn, J.-L. Dekeyser, and S. Niar. "An Efficient Power Estimation Methodology for Complex RISC Processor based Embedded Platforms", *22nd Great Lakes Symposium on VLSI (GLSVLSI 2012)*, May 2012, Salt Lake City, Utah, USA.

[125] M. Schwiegershausen and P. Pirsch. "A system level design methodology for the optimization of heterogeneous multiprocessors." In *Eighth International Symposium on System Synthesis*, pages 162–167, 1995.

[126] R. Szymanek and K. Kuchcinski. "Design space exploration in system level synthesis under memory constraints." In *25th EUROMICRO Conference*, volume 1, pages 29–36, 1999.

[127] G. Schirner, A. Gerstlauer, and R. Domer. "Abstract, Multifaceted Modeling of Embedded Processors for System Level Design." *In Design Automation Conference, 2007.* ASP-DAC '07. Asia and South Pacific, pages 384 –389, 2007.

[128] G. Snider. "Spacewalker : Automated design space exploration for embedded computer systems, HPL-2001-220." *Technical report*, HP Laboratories Palo Alto, Sept. 2001.

[129] A. Sinha and A.Chandrakasan, "JouleTrack – A Web Based Tool for Software Energy Profiling," *Proc. 38th Design Automation Conference*, June 2001.

[130] E. Senn, J. Laurent, N. Julien and E. Martin. "SoftExplorer : Estimating and Optimizing the Power and Energy Consumption of a C Program for DSP Applications." In *Eurasip Journal on Applied Processing*, .

[131] M. Sgroi, L. Lavagno, and A. Sangiovanni-Vincentelli. "Formal Models for Embedded System Design." *IEEE Des. Test*, 17(2) :14–27, 2000.

[132] E. Senn, C. Belleudy, D. Chillet, A. Fritsch, R. Ben Atitallah and O.Zendra. "Open-PEOPLE : Open Power and Energy Optimization PLatform and Estimator". *15th EUROMICRO Conference on Digital System Design (DSD'2012)*, September 2012, Cesme, Izmir, Turkey.

[133] M. Tsai, C. Kulkarni, C. Sauer, N. Shah, and K. Keutzer. "A benchmarking methodology for network processors." In P. Crowley, M. Franklin, H. Hadimioglu, and P. Onufryk, editors, *Network Processor Design : Issues and Practices*, volume 1, pages 141–165. Morgan Kaufmann Publishers, Oct. 2002.

[134] H. Theiling, C. Ferdinand, and R. Wilhelm. "Fast and precise WCET prediction by separate cache and path analyses." *Real-Time Systems*, Kluwer, 18(2-3) :157–179, May 2000.

[135] L. Thiele, S. Chakraborty, M. Gries, and S. Kunzli. "Design space exploration of network processor architectures." In P. Crowley, M. Franklin, H. Hadimioglu, and P. Onufryk, editors, *Network Processor Design : Issues and Practices*, volume 1, pages 55–89. Morgan Kaufmann Publishers, Oct. 2002.

[136] L. Thiele, S. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert. "Embedded software in network processors - models and algorithms." In *First Workshop on Embedded Software (EMSOFT)*, pages 416–434, Oct. 2001.

[137] T. K. Tan A. Raghunathant, G. Lakshminarayanat, N. K. Jha "High-level Software Energy Macro-modeling", *IEEE Design Automation Conference*, June 2001

[138] T. K. Tan A. Raghunathan , and N. K. Jha "Embedded Operating System Energy Analysis and Macro-modeling", *International Conference on Computer Design*, 2002

[139] Rabbit project of TIMA - http ://tima-sls.imag.fr/www/research/rabbits

[140] R. Uhlig and T. Mudge. "Trace-driven memory simulation : A survey." *ACM Computing Surveys*, 29(2) :128–170, June 1997.

[141] T. Wolf and M. Franklin. "CommBench - A telecommunications benchmark for network processors." In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 154–162, Apr. 2000.

[142] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. "The SPLASH-2 programs : Characterization and methodological considerations." In *22nd International Symposium on Computer Architecture (ISCA)*, pages 24–36, June 1995.

[143] S. J. Weber, M. W. Moskewicz, M. L Low, and K. Keutzer. "Multi-view operation-level design supporting the design of irregular ASIPs." *Technical Report UCB/ERL M03/12*, Electronics Research Laboratory, University of California at Berkeley, Apr. 2003.

[144] C. Ykman-Couvreur, J. Lambrecht, D. Verkest, F. Catthoor, A. Nikologiannis, and G. Konstantoulakis. "System-level performance optimization of the data queueing memory management in high-speed network processors." In *39th Design Automation Conference (DAC)*, June 2002.

[145] V. Zivkovic, E. Deprettere, P. van der Wolf, and E. de Kock. "Design space exploration of streaming multiprocessor architectures." In *IEEE Workshop on Signal Processing Systems (SIPS)*, pages 228–234, 2002.