Prédiction de performance d'algorithmes de traitement d'images sur différentes architectures hardwares

Nicolas Soucies

Thèse CIFRE

Université Pierre et Marie Curie / CNRS

Institut des Systèmes Intelligents et de Robotique (ISIR)

Groupe ALTEN

SOUCIES Nicolas

Prédiction de performance d'algorithmes de traitement d'images sur différentes architectures hardwares

---

Direction : Stéphane REGNIER

Encadrement : Nizar OUARTI

JURY

M. AMMI M.	Maître de conférences à l'Université Paris SUD	Rapporteur
M. CHARVILLAT V.	Professeur à l'ENSEEIHT Université de Toulouse	Rapporteur
M. WENDLING L.	Professeur à l'Université Paris Descartes	Examinateur
M. ZARADER J.L.	Professeur à l'Université Pierre et Marie Curie	Examinateur
M. REGNIER S.	Professeur à l'Université Pierre et Marie Curie	Examinateur
M. OUARTI N.	Maître de conférences à l'Université Pierre et Marie Curie	Examinateur

INVITE

M. BAUDET J.C.	Directeur R&D du Groupe ALTEN	Examinateur

Résumé

Dans le contexte de la vision par ordinateur, le choix d'une architecture de calcul est devenu de plus en plus complexe pour un spécialiste du traitement d'images. Le nombre d'architectures permettant de résoudre des algorithmes de traitement d'images augmente d'année en année. Ces algorithmes s'intègrent dans des cadres eux-mêmes de plus en plus complexes répondant à de multiples contraintes, que ce soit en terme de capacité de calculs, mais aussi en terme de consommation ou d'encombrement. A ces contraintes s'ajoute le nombre grandissant de types d'architectures de calculs pouvant répondre aux besoins d'une application (CPU, GPU, FPGA).

L'enjeu principal de l'étude est la prédiction de la performance d'un système, cette prédiction pouvant être réalisée en phase amont d'un projet de développement dans le domaine de la vision. Dans un cadre de développement, industriel ou de recherche, l'impact en termes de réduction des coûts de développement, est d'autant plus important que le choix de l'architecture de calcul est réalisé tôt.

De nombreux outils et méthodes d'évaluation de la performance ont été développés mais ceux-ci, se concentrent rarement sur un domaine précis et ne permettent pas d'évaluer la performance sans une étude complète du code ou sans la réalisation de tests sur l'architecture étudiée. Notre but étant de s'affranchir totalement de benchmark, nous nous sommes concentrés sur le domaine du traitement d'images pour pouvoir décomposer les algorithmes du domaine en éléments simples ici nommées briques élémentaires.

Dans cette optique, un nouveau paradigme qui repose sur une décomposition de tout algorithme de traitement d'images en ces briques élémentaires a été conçu. Une méthode est proposée pour modéliser ces briques en fonction de paramètres software et hardwares. L'étude démontre que la décomposition en briques élémentaires est réalisable et que ces briques élémentaires peuvent être modélisées. Les premiers tests sur différentes architectures avec des données réelles et des algorithmes comme la convolution et les ondelettes ont permis de valider l'approche. Ce paradigme est un premier pas vers la réalisation d'un outil qui permettra de proposer des architectures pour le traitement d'images et d'aider à l'optimisation d'un programme dans ce domaine.

Mots clés : Prédiction de performance, Traitement d'images, Architecture Hardware, CPU, Calculateur embarqué, Modélisation.

Abstract

In computer vision, the choice of a computing architecture is becoming more difficult for image processing experts. Indeed, the number of architectures allowing the computation of image processing algorithms is increasing. Moreover, the number of computer vision applications constrained by computing capacity, power consumption and size is increasing. Furthermore, selecting an hardware architecture, as CPU, GPU or FPGA is also an important issue when considering computer vision applications.

The main goal of this study is to predict the system performance in the beginning of a computer vision project. Indeed, for a manufacturer or even a researcher, selecting the computing architecture should be done as soon as possible to minimize the impact on development.

A large variety of methods and tools has been developed to predict the performance of computing systems. However, they do not cover a specific area and they cannot predict the performance without analyzing the code or making some benchmarks on architectures. In this works, we specially focus on the prediction of the performance of computer vision algorithms without the need for benchmarking. This allows splitting the image processing algorithms in primitive blocks.

In this context, a new paradigm based on splitting every image processing algorithms in primitive blocks has been developed. Furthermore, we propose a method to model the primitive blocks according to the software and hardware parameters. The decomposition in primitive blocks and their modeling was demonstrated to be possible. Herein, the performed experiences, on different architectures, with real data, using algorithms as convolution and wavelets validated the proposed paradigm. This approach is a first step towards the development of a tool allowing to help choosing hardware architecture and optimizing image processing algorithms.

Keywords: Performance prediction, Image processing, Hardware Architectures, CPU, Embedded Systems, model.

Remerciements

Ce travail de thèse est né de la collaboration entre l'Institut des Systèmes Intelligents et de Robotique (CNRS/UPMC) et de la société ALTEN. Il n'aurait pu avoir lieu sans le soutien de nombreuses personnes de ces deux entités et j'aimerais donc remercier tous ceux qui se sont impliqués directement ou non à l'aboutissement de cette thèse.

J'aimerais ensuite remercier les deux rapporteurs, M. AMMI Mehdi et M. CHARVILLAT Vincent, pour le temps consacré à la relecture de ma thèse. Je remercie également M. ZARADER Jean-Luc et M. WENDLING Laurent d'avoir accepté de faire partie des membres du jury de ma thèse.

Je voudrais remercier mon directeur de thèse, M. Stéphane REGNIER d'avoir été présent tout au long de ces années.

J'ai eu la chance d'être encadré par deux personnes aux nombreuses qualités humaines et scientifiques. Nizar OUARTI m'a permis de grandir en tant qu'homme et en tant que chercheur. Je voudrais le remercier pour son soutien et son implication dans mon projet de thèse. J'ai énormément apprécié travailler et échanger avec lui tout au long de ces trois années de thèse. Jean-Claude BAUDET, personnalité au grand cœur, a été un des grands acteurs du lancement de ma thèse. Son soutien et son expérience m'ont permis d'avancer et de grandir pour arriver au bout de la rédaction de mon manuscrit.

Au sein de la société ALTEN, de nombreuses personnes m'ont permis d'évoluer et d'avancer tout au long de ma thèse mais j'aimerais tout particulièrement remercier Stéphane JEANTY pour son aide et sa bonne humeur lors de la finalisation de ma thèse.

Mes remerciements vont également aux collègues du groupe interaction que j'ai pu rencontrer au cours de mes années de thèse au sein de l'ISIR. Je voudrais particulièrement remercier Abdenbi, Antoine, Bruno, Jean, Laura, Mokrane, Soukeyna, Tianming, Tiantian, Zhenjiang, … Je ne saurai oublier le personnel de l'ISIR qui assure le bon fonctionnement du laboratoire, pour son soutien tout au long de ma thèse.

Enfin, je voudrais tout particulièrement adresser mes remerciements à ma compagne, ma mère, mon père, mes frères et sœur, ainsi que toute ma famille pour leur encouragement, leur aide et leur énorme soutien tout au long de ma thèse.

Table des matières

Table des figures

Introduction générale

Contexte

Le projet de thèse s'inscrit à la suite de recherches menées sur le développement d'une solution palliative pour des personnes atteintes de DMLA (Déficience Maculaire Liée à l'Âge) dirigée par Nizar Ouarti à l'ISIR. La solution en question s'appuie sur des algorithmes de traitement d'images nécessitant une capacité de calcul importante. Outre le développement des algorithmes de traitement d'images, un des problèmes majeurs du projet qui s'est posé, a été le choix d'une architecture hardware de calculs. Il aurait été intéressant à ce moment-là de disposer d'un outil qui permette d'aider à choisir la meilleure architecture possible par rapport aux contraintes. L'absence d'un tel outil a donc exprimé le besoin d'en mettre un au point.

De manière plus générale, dans le contexte de la vision par ordinateur, le choix d'une architecture de calcul est devenu de plus en plus complexe pour un spécialiste du traitement d'images. En effet, le nombre d'architectures permettant de résoudre des algorithmes de traitement d'images augmente d'années en années. De plus, les applications de vision s'intègrent dans des cadres de plus en plus complexes répondant à de multiples contraintes, que ce soit en terme de capacité de calculs, mais aussi en terme de consommation ou d'encombrement.

A ces contraintes s'ajoute le nombre grandissant de types d'architectures de calculs pouvant répondre aux besoins d'une application, que ce soit des processeurs « classiques » CPU (Central Processing Unit), des processeurs graphiques GPU (Graphic Processor Unit), des DSP (Digital Signal Processor) ou encore des circuits programmables FPGA (field-programmable gate array). Mais on peut aussi considérer deux types de traitements que sont, le traitement en local et le traitement déporté. En effet, dans certaines conditions, le traitement déporté peut offrir une capacité de calculs presque illimitée mais les contraintes porteront davantage sur la capacité et le temps de transmission des données. En ce qui concerne le CPU, de plus en plus de plates-formes destinées à l'embarqué ou non voient le jour et il serait intéressant d'avoir un outil permettant de prédire les performances sur ces architectures.

Un spécialiste de la vision peut avoir du mal à apprécier la performance d'une application sur un système de calculs donné. En effet, ces derniers sont rarement des

spécialistes des architectures et connaissent peu les particularités des différents systèmes de calcul. Il apparaît aussi évident que l'obtention des plateformes qui pourraient convenir et la réalisation de tests sur chacune d'elles, peuvent s'avérer coûteuses, lourdes et fastidieuses. Partant de ce constat, il paraît intéressant pour les spécialistes en traitement d'images et vidéo de disposer d'un outil qui les aide dans le choix d'une architecture de calcul.

L'enjeu principal de notre étude est de pouvoir s'inscrire dans la phase amont d'un projet dans le domaine de la vision. En effet, pour un industriel ou même pour un chercheur, il paraît évident que, plus le choix de l'architecture de calcul est réalisé tôt, moins l'impact sur le développement sera important. Si l'on attend d'avoir effectué des tests sur une architecture et d'avoir développé tout ou partie de l'application, il sera beaucoup plus complexe de recommencer à zéro sur une nouvelle plateforme.

Bien évidemment, au-delà de cet enjeu principal, l'outil a pour objectif d'intervenir tout au long de la vie du développement d'une application, pour permettre de se réorienter le cas échéant. Le développement d'un tel outil rencontre un certain nombre de verrous technologiques et d'aléas qui seront développés dans les différents chapitres de la thèse.

En effet, on peut s'apercevoir que de nombreux modèles ou méthodes ont été développés pour prédire la performance d'un programme mais, à notre connaissance, ceux-ci nécessitent d'avoir une description très fine des algorithmes étudiés, voir même de connaitre le code déjà implémenté pour fournir une évaluation de la performance. De plus, les solutions totalement déterministes sont souvent fastidieuses à utiliser. Pour se placer très en amont dans le développement d'une solution, nous nous sommes concentrés sur le domaine de la vision par ordinateur pour pouvoir décomposer ce domaine facilement. C'est en se concentrant sur ce domaine que nous avons pu nous démarquer des solutions existantes en ayant des résultats convaincants tout en se plaçant très en amont dans la réalisation d'une solution de traitement d'images.

Notre outil pourrait donc permettre à un utilisateur de décrire son algorithme de traitement d'images de façon très macroscopique et lui proposer des architectures de calculs en fonction de contraintes qu'il fournirait, comme par exemple, le nombre d'images par seconde nécessaire, le temps de calcul maximum ou encore la consommation du système ou son encombrement.

L'outil devrait, aussi, déterminer le temps de calcul d'une application sur une architecture hardware donnée, s'apparentant ainsi à une simulation. Il paraît aussi évident que d'autres fonctions pourraient être ajoutées dans l'aide au perfectionnement d'une application. En effet, en se basant sur les performances d'un

algorithme, l'outil pourrait indiquer à l'utilisateur les points critiques de son application, pour lui permettre d'améliorer sa performance. Pour cela l'utilisateur pourra retravailler ces points ou réaliser quelques concessions.

Positionnement

Nous avons défini un **système de calcul** (Figure 1) comme étant un couple constitué d'un **software** qui, dans notre contexte, est défini comme l'algorithme qui sera appliqué aux données d'entrées (une image pour des applications dans le domaine de la vision par ordinateur) et d'une **architecture hardware** représentant la plateforme qui va réaliser les calculs. C'est ce couple qui va devoir être modélisé de façon croisée pour déterminer la performance du système et répondre aux objectifs de l'outil.

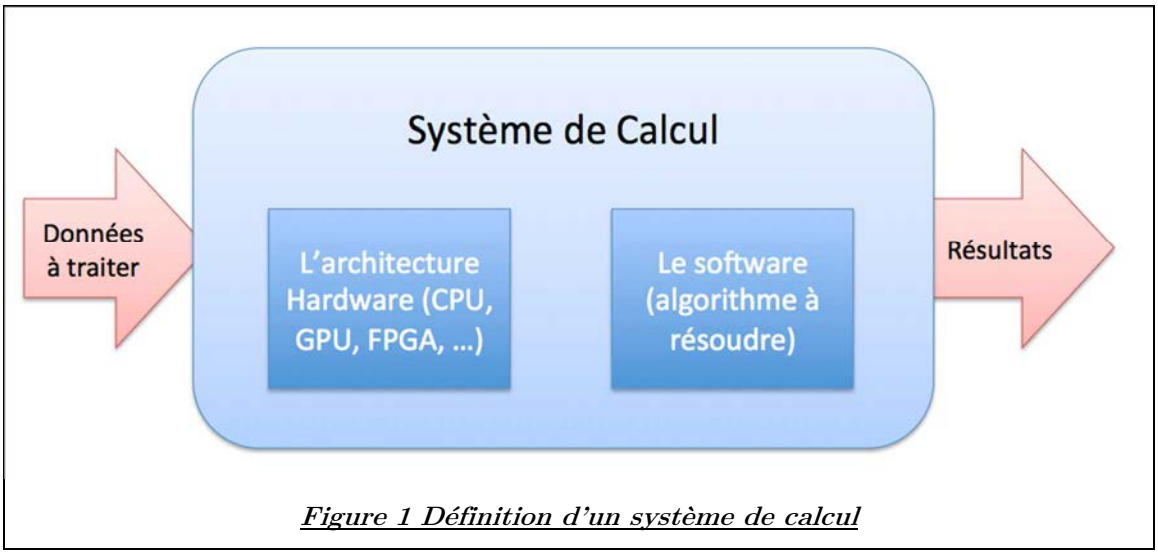


Figure 1 Définition d'un système de calcul

Dans l'objectif de développement de cet outil, nous avons mené un certain nombre de recherches sur la modélisation d'un algorithme de traitement d'images et sur la modélisation d'architectures hardwares.

Pour cette étude, nous avons décidé de nous focaliser sur les applications de traitement d'images sur des architectures CPU. Comme nous le verrons, de nombreuses architectures de calculs peuvent servir à la résolution d'algorithmes de traitement d'images comme les GPU, les FPGA ou encore les DSP. Mais, comme nous le montrerons par la suite, nous avons décidé, dans un premier temps de nous concentrer sur le CPU, qui est une architecture plus flexible que les autres et qui est apparue prioritaire dans le domaine de l'embarqué. De plus, comme énoncé

précédemment, la performance d'un système de calcul pour des applications de vision, regroupe plusieurs aspects que sont :

- Le temps de résolution de l'algorithme par l'architecture qui peut s'exprimer en seconde mais aussi en images par seconde ou encore en flops.
- La consommation énergétique du système exprimée en Watt.
- La température exprimée en degré.

Dans notre étude, nous considérerons la performance comme étant seulement le temps de résolution d'un algorithme sur une architecture de calcul donnée.

Le but de l'approche est de faire en sorte que l'utilisateur arrive à faire totalement abstraction de benchmark pour modéliser son algorithme. L'idée qui a été retenue dans cette étude, a été de s'appuyer sur des briques élémentaires pour décrire un algorithme, qui elle-même repose sur des modèles qui ont pu être défini par des benchmarks. L'approche peut être qualifiée d'hybride. En effet, les paramètres libres des différents modèles reposent sur des micro-benchmarks. Ce n'est pas une approche totalement benchmark qui nécessiterait à chaque fois de recommencer tous les calculs pour connaître le temps de calcul d'un algorithme, ce n'est pas non plus une approche totalement prédictive qui n'aurait besoin d'aucun benchmark, d'où le terme d'approche hybride. Pour présenter cette approche, le manuscrit se divise en trois chapitres distincts.

Le premier chapitre présente en détails le système de calcul que l'on désire modéliser, mais aussi les différents modèles et méthodes qui ont été développés pour répondre à cette problématique d'évaluation de la performance. Ce chapitre montre bien que les solutions existantes se place très rarement sur un domaine précis. En conséquence, elles interviennent très en aval dans le développement d'une solution et utilisent souvent des benchmark pour évaluer la performance d'un système. A notre connaissance, les outils d'évaluation de la performance d'un système de calcul n'interviennent pas en amont dans le développement d'une solution.

Le deuxième chapitre présente un premier paradigme permettant de modéliser et évaluer la performance d'un algorithme de traitement d'images sur une architecture CPU très en amont dans le développement d'un système de calcul. Pour cela, les algorithmes sont décomposés en briques élémentaires et chacune de ces briques est modélisée de façon indépendante mais avec des paramètres identiques. Ce chapitre présente également une première version d'un outil qui permettrait d'utiliser la modélisation de la performance d'un algorithme de traitement d'images qui a été mise en œuvre. Cet outil nommé Métis, permet de mieux comprendre les applications concrètes qui découlent du paradigme scientifique proposé.

Le troisième chapitre présente la validation expérimentale du paradigme scientifique sur une brique élémentaire de traitement d'images. Cela permet de valider que la modélisation d'une brique élémentaire très utilisée dans le domaine du traitement d'images est possible. Puis, la suite des résultats expérimentaux permet de prouver qu'un algorithme « complexe » comme les ondelettes est décomposable en briques élémentaires et que l'addition de la modélisation de ces briques correspond bien à l'ensemble de l'algorithme.

Enfin, **la conclusion** rappelle les points clés de la thèse et permet de comprendre les perspectives d'évolution de l'étude et les apports scientifiques qui pourront être développés par la suite.

Chapitre 1 Caractérisation d'un système de calcul

Un système de calcul, au sens où nous allons l'entendre dans cette étude, est composé d'une partie hardware et d'une partie software, comme présenté dans l'introduction. Dans ce chapitre, une étude de la complexité algorithmique et du parallélisme va permettre de faire ressortir les problématiques liées à la modélisation d'un algorithme en général. En effet, il existe plusieurs types et plusieurs niveaux de complexité et de parallélisme selon les instructions. Ce sont ces degrés de parallélisme et de complexité (deux notions intimement liées) qui vont influencer la performance d'un algorithme.

Ensuite, une étude des architectures permettant la résolution d'algorithmes de traitement d'images ainsi qu'un état de l'art des benchmarks comparant ces différentes architectures, vont permettre de faire ressortir les caractéristiques majeures influençant la performance et de comprendre les difficultés à modéliser une architecture de calcul.

Après avoir défini le couple hardware - software qui compose ce que l'on appelle un système de calcul, ce chapitre présentera les différents outils et modèles de l'état de l'art qui permettent d'instrumenter et caractériser la performance d'un système de calcul. Comme cela a été défini dans l'introduction, la performance n'est autre que le temps de résolution de l'algorithme.

Cet état de l'art permet de comprendre quelles sont les limites des paradigmes et outils existants et pourquoi le problème développé dans l'introduction n'a pas été résolu.

1.1. Un système de calcul : un software combiné à un hardware

1.1.1. La complexité algorithmique et la parallélisation des calculs

1.1.1.1. Définition

La complexité algorithmique s'exprime sous deux formes distinctes : la complexité en mémoire et la complexité en temps [1].

La complexité en mémoire, s(n), est la fonction qui calcule le nombre de mots mémoires nécessaires à l'implémentation de l'algorithme sur une machine. Par exemple, si on trie des entiers, S(n) = n, si on trie des noms de 20 caractères, sur une machine 32 bits, S(n) = 5n (1 caractère = 8 bits) etc.

La complexité en temps, t(n), est la fonction qui calcule le nombre de pas de calculs élémentaires nécessaires à l'exécution de l'algorithme sur une machine. Par exemple, si on trie des entiers avec la méthode du tri à bulles : $t(n) = n^2$.

La complexité est la fonction t entre le nombre d'opérations et la taille des entrées n : le nombre d'opérations à traiter est fonction de t(n). L'augmentation de la puissance de calcul ne permet donc pas de faire diminuer de façon linéaire le temps de calcul. Si l'on prend le problème dans le sens inverse, il est facile d'en comprendre la raison : pour un temps donné, le nombre de données traitées n'augmente pas linéairement avec la puissance de calcul, mis à part dans un cas très précis : lorsque t(n) = n. En effet, si l'on considère un algorithme ayant une complexité caractérisée par la fonction $t(n) = n^3$ et que l'on multiplie notre puissance de calcul par 1000 :

$(n')^3 = 1000 * n^3$ On obtient donc n' = 10 n .

n' (nombre de données traitées après l'augmentation de la puissance)

En conclusion, avec cette complexité algorithmique, lorsque l'on multiplie la puissance de calcul par 1000, le nombre de données traitées est multiplié par 10 et non pas aussi par 1000, si l'on supposait que le temps d'exécution de x opérations est inversement proportionnel à la puissance de calcul.

Un certain nombre de complexités ont été fixées [1] dont voici quelques exemples :

- La comparaison de deux entiers : O(n)
- L'addition de deux entiers : O(n)
- La multiplication de deux entiers : $O(n^2)$

- La division euclidienne de deux entiers : O(n)
- L'algorithme d'Euclide pour des entiers de taille au plus n : $O(n^2)$

« O() » étant la notation qui définit la fonction que va suivre la performance par rapport au nombre d'éléments à traiter, pour deux entiers de taille n.

Si on met cela en perspective du domaine du traitement d'images, « n » n'est bien évidemment pas la taille de l'image, mais la caractérisation d'un pixel d'une image.

1.1.1.2. Complexité et parallélisme

Les complexités séquentielles t(N) et s(N) sont dépendantes de la taille N du problème à traiter. Pour étudier la complexité parallèle, on considère des classes d'architectures parallèles très simples qui peuvent être caractérisées par un seul paramètre : le nombre d'éléments de parallélisation.

La complexité parallèle est alors dépendante de deux variables ; la taille du problème (N) et le nombre d'éléments de parallélisation (P) utilisés : t(N, P) et t(N, P) notées généralement : Tp(N) et Sp(N). L'étude va se concentrer sur la complexité en temps et laisser de côté la complexité en mémoire.

Aujourd'hui, lorsque l'on parle de performances algorithmiques et de parallélisation, il paraît évident de parler de la loi d'Amdahl [2] qui, même si elle peut être vue comme triviale de nos jours, est considérée comme l'un des fondement de la caractérisation de la parallélisation algorithmique.

La loi d'Amdahl, écrite en 1967, définit que si N est le nombre de processeur (cœur aujourd'hui), s est le temps passé (par un processeur mono-cœur) sur une partie d'un programme en série, et p est le temps passé (par un processeur mono-cœur) sur la partie du programme qui peut être réalisée en parallèle, alors le gain de parallélisation est donné par :

$$\boldsymbol{Gain} = \frac{(\boldsymbol{s}+\boldsymbol{p})}{\left(\boldsymbol{s}+\frac{\boldsymbol{p}}{\boldsymbol{N}}\right)} \text{ [3]} \qquad (1)$$

Depuis, cette loi a été réévaluée par Gustafson en 1988 [3] et étendue pour correspondre aux problématiques actuelles des nouvelles technologies multi-coeur par Hill et Marty en 2008 [4]. Ces études théoriques permettent de décrire des phénomènes mais pas de réaliser une prédiction sur des résultats expérimentaux.

La complexité du parallélisme d'un algorithme est, de nos jours, complétement dépendante de l'architecture sur laquelle l'algorithme va être résolu. En effet, comme nous le verrons dans la partie suivante (1.1.2), une architecture Von Neumann ne va

pas s'étudier de la même façon qu'une architecture Harvard et il existe donc différents types de « modèles » qui décrivent ces types de parallélisme.

Une ou plusieurs instructions, un unique flux de données :

- SISD (Single Instruction Single Data)
- MISD (Multiple Instructions Single Data)

Dans ces types de modèles, un ou plusieurs processeurs partagent une même mémoire et exécutent chacun, des instructions différentes en parallèle mais sur un unique flot de données. Ce modèle ne s'applique pas au traitement d'images et ne convient pas vraiment aux architectures étudiées.

Une ou plusieurs instructions, plusieurs flux de données :

- SIMD (Single Instruction Multiple Data)
- MIMD (Multiple Instructions Multiple Data)

Une des méthodes les plus utilisées pour le calcul de la complexité des algorithmes parallèles est le modèle PRAM [5]. Le modèle PRAM (Parallel Random Access Memory) est un modèle issu du modèle RAM, qui permet de parfaitement représenter la parallélisation des calculs sur différentes données.

Il s'agit d'une machine théorique où, comme dans le modèle RAM, on considère que toute instruction prend exactement un cycle. Il existe deux types de PRAM :

Le PRAM-MIMD (Multiple Instructions Multiple Data)

Le modèle PRAM (Parallel Random Acess Memory)-MIMD est une tentative possible de la modélisation des architectures MIMD : chaque PE exécute une instruction de son programme à chaque cycle sur des données différentes. Pour l'instant, il existe peu d'architectures permettant de réaliser du MIMD (beaucoup d'architectures sont développées dans le but de traiter une grande quantité de données et ne peuvent donc appliquer qu'une seule et même instruction à toutes ces données).

Le PRAM-SIMD (Single Instruction Multiple Data)

Ici, chacun des processeurs va exécuter la même instruction sur des flots de données distincts et sur une mémoire partagée.

Pour agir sur n flots de données, il existe deux façons : la mémoire partagée et le réseau d'interconnexion. Le PRAM, quant à lui, utilise une mémoire partagée.

Comme ce modèle est à mémoire centrale partagée, le problème qui se pose est de définir comment les PEs (Parrallel Elements) accèdent à cette mémoire. En effet, chacun des PEs n'agit pas sur le même flux de données mais cela n'empêche pas le fait que sur un cycle, deux PEs peuvent avoir à lire ou écrire sur la même case mémoire. Snir a donc proposé une classification des PRAMS en quatre catégories selon que les accès (en lecture et en écriture) soient effectués de manière concurrente ou de manière exclusive (séquentielle) :

- EREW (Exclusive Read Exclusive Write) : seul un processeur peut lire et écrire à un moment donné sur une case donnée de la mémoire partagée.
- CREW (Concurrent Read Exclusive Write) : plusieurs processeurs peuvent lire en même temps une même case, par contre, un seul à la fois peut y écrire. C'est un modèle proche des machines réelles (et de ce que l'on a vu à propos des threads JAVA).
- CRCW (Concurrent Read Concurrent Write) : c'est un modèle étonnamment puissant et assez peu réaliste, mais qui est intéressant à considérer d'un point de vue théorique. Plusieurs processeurs peuvent lire ou écrire en même temps sur la même case de la mémoire partagée. Il nous reste à définir ce qui se passe lorsque plusieurs processeurs écrivent au même moment; on fait généralement l'une des trois hypothèses suivantes :
  - mode consistant : tous les processeurs qui écrivent en même temps sur la même case, écrivent la même valeur.
  - mode arbitraire : c'est la valeur du dernier processeur qui écrit, qui est prise en compte.
  - mode fusion : une fonction associative (définie au niveau de la mémoire), est appliquée à toutes les écritures simultanées sur une case donnée. Cela peut être par exemple, une fonction maximum, un bit ou bit à bit etc.

Ces différents modèles permettent déjà d'établir un lien entre la complexité algorithmique et les typologies d'architectures hardwares.

1.1.2. Architecture hardware et traitement d'images

Aujourd'hui, il existe de nombreuses architectures qui permettent de résoudre des algorithmes de traitement d'images. Cette étude va présenter succinctement les trois grands groupes d'architectures hardwares que sont le CPU (Central Processing Unit), le GPU (Garphics Processing Unit) et le FPGA (Field Programmable Gate Array). De nombreux chercheurs ont d'ailleurs comparé la performance de différents algorithmes de traitement d'images sur ces types d'architectures pour analyser leurs

comportements. L'analyse se concentrera ensuite exclusivement sur les différents types de CPU.

1.1.2.1. Les types d'architectures

Le processeur « générique » : CPU (Central Processing Unit)

Ce que l'on appelle souvent les CPU, ne sont autres que des GPP (General Purpose Processor). Il existe d'autres types de processeurs comme les DSP spécialisés dans le traitement du signal mais ceux qui nous importent ici, sont les GPP. Il existe plusieurs types d'architectures CPU et ces typologies ont une influence sur la performance. Les CPU ont commencé à être développé sur la base d'une architecture Von Neumann (Figure 2) mais ont ensuite évolué vers une architecture plus performante, l'architecture Harvard (Figure 3).

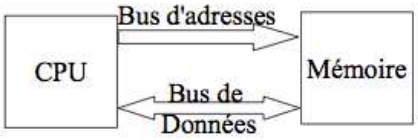


Figure 2 Architecture Von Neumann

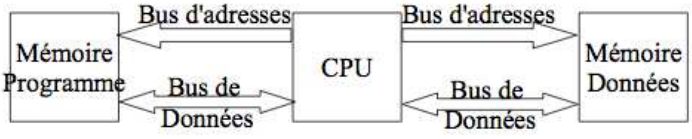


Figure 3 Architecture Harvard

Sur une architecture Von Neumann, la mémoire contient les instructions et les données. Le CPU charge donc les instructions depuis la mémoire et il possède deux types de registres : PC (« Program Counter » : instruction à exécuter), et SP (sommet de pile de registres généraux).

Sur une architecture Harvard, les mémoires de données et des instructions du programme sont séparées, on peut alors accéder en parallèle aux deux types de mémoire. Sur la base de l'architecture Harvard, il existe deux grands types d'architectures qui peuvent influencer sur la performance d'un système de calcul : CISC et RISC.

Les processeurs avec une architecture RISC (Reduced Instruction Set Computer) ont dans l'absolu, un temps d'exécution plus rapide que les processeurs à architecture CISC (Complex instruction Set Computer), cette dernière étant plus complexe. Mais le même code C aura au moins cinq fois plus d'instructions Assembleur sur RISC que sur CISC. L'efficacité de l'architecture CISC se retrouve dans le fait que les codes ont une empreinte mémoire plus faible et les accès mémoires sont donc réduits. Dans une architecture RISC, on observe que pour une opération donnée, le nombre d'instructions va être généralement plus élevé que dans le cas d'une architecture CISC, ainsi l'empreinte mémoire des programmes est plus importante mais l'accessibilité est accrue.

Les deux solutions de GPP les plus avancées et populaires, que sont l'architecture X86 utilisée dans la majeure partie des ordinateurs personnels aujourd'hui, ainsi que l'architecture ARM énormément utilisée dans le monde de l'embarqué, sont présentées ci-dessous.

Architecture X86

L'architecture X86 CISC est donc très utilisée et est apparue en 1978 avec le processeur 8086. Cette architecture est actuellement celle qui est utilisée en grande majorité dans les ordinateurs personnels et fournit par Intel et AMD notamment. Les processeurs sont pour la plupart multi-cœur. Chaque cœur a une fréquence d'exécution et possède souvent un à deux niveaux de cache (Premier niveau avec un cache d'instructions et un cache de données) qui leur est propre ainsi qu'un niveau de cache partagé entre les cœurs (Figure 4).

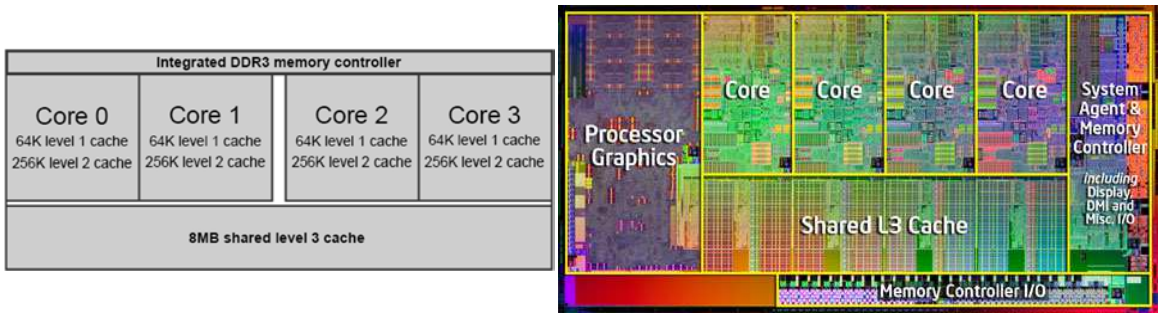


Figure 4 Exemple d'architecture X86 Intel quad core

L'avantage majeur de l'architecture X86 est sa grande puissance de calculs, avec des fréquences d'horloges et des tailles de mémoires caches les plus importantes de tous les processeurs. Ces architectures sont, aujourd'hui, pour la plupart multi-core, mais elles dépassent rarement les 16 cœurs, mise à part certains développements spécifiques pour des supercalculateurs.

Architecture ARM

L'architecture ARM est apparue en 1985 sous la forme de prototype créé par la société Acorn. La société ARM, du nom de l'architecture, a été fondée en Novembre 1990. ARM conçoit différents cœurs de microprocesseurs RISC et vend ensuite des licences de conception à ses partenaires fabricants de semi-conducteurs. Les partenaires en question fabriquent et vendent à leurs clients des circuits intégrés, basés sur ces cœurs RISC car en effet, la société ARM ne fabrique pas de circuits intégrés.

Elle développe un ensemble de technologies pour aider les développements autour de ses architectures : outils logiciels, cartes, outils de mise au point, logiciels applicatifs,

architectures de bus, périphériques, etc. Cette architecture de processor est la plus répandue dans le monde de l'embarqué, du fait de sa faible consommation et son faible coût.

Les avantages du ARM sont assez nombreux :

- Une faible consommation : jusqu'à 1200 MIPS / W (0.25µm @ 0.9 V)
- Une bonne performance pour de l'embarqué: meilleure que les microprocesseurs 8/16-bit
- Un code haute densité : plus de programmes dans une taille mémoire limitée
- Faible surface de puce : 1 mm² (Leff= 0.25µm)
- Un dégagement de chaleur très faible
- Un coût limité

Elle n'est pour l'instant pas intégrée dans les ordinateurs personnels car elle reste limitée en puissance de calcul comparée à l'architecture X86.

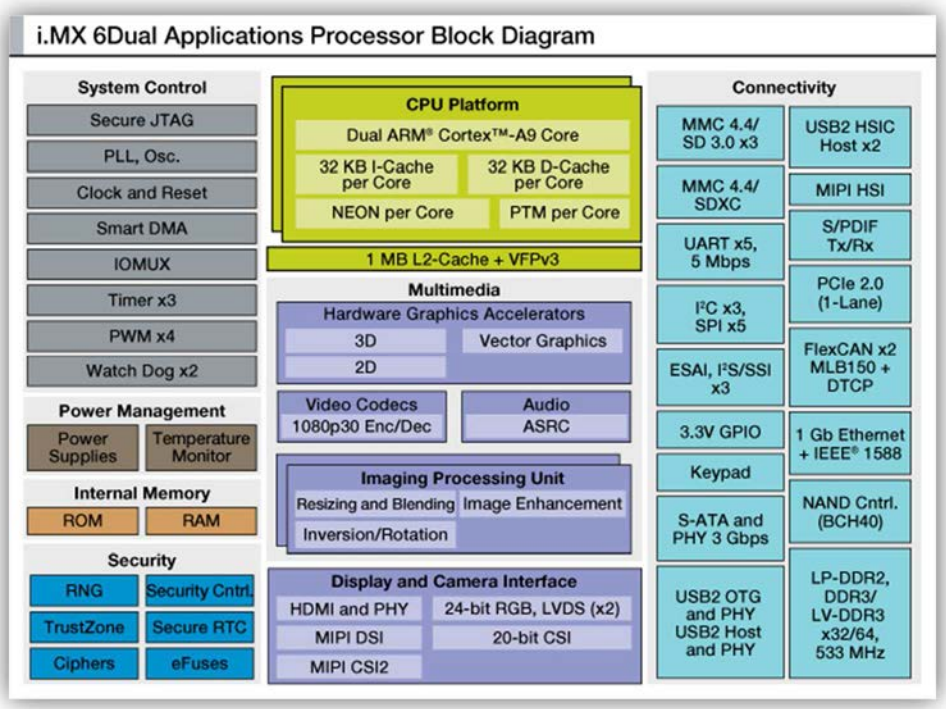


Figure 5 Exemple d'architecture ARM Freescale

Plus spécifiquement pour le traitement vidéo/d'images, les constructeurs tels que Freescale conçoivent des System on chips basés sur des processeurs ARM, qui permettent d'avoir un grand nombre de connectivités, des interfaces vidéos multiples, ainsi que des accélérateurs graphiques matériels et des traitements d'images implémentés en hardware. Ils intègrent même des Codec audio et vidéo tel que l'on

peut voir sur le block diagramme de l'IMX6 ARM 9 dual Core (Figure 5). Cet exemple représente bien l'esprit de l'architecture ARM, qui est non seulement un processeur mais également, un ensemble d'outil regroupé dans un SOC avec le CPU.

Conclusion

Au-delà des types d'architectures (Von Neumann ou Harvard), il ressort que les paramètres de performance les plus significatifs sont bien évidemment, la fréquence du processeur mais aussi le nombre de cœurs, la taille et le nombre de niveaux de mémoire cache, la vitesse d'accès à la mémoire cache ainsi que le type d'instructions (CISC ou RISC).

Pour notre étude, on comprend bien que tous ces paramètres vont influer sur la performance d'un algorithme de traitement d'images de façon différente selon les caractéristiques de celui-ci.

Processeur graphique : GPU (Graphics Processing Unit)

Le calcul par une unité de processeur graphique (GPU) consiste à utiliser celui-ci en parallèle du CPU, pour accélérer des tâches de calculs polyvalentes de science et d'ingénierie. Les premiers GPU, tels que nous les connaissons aujourd'hui, ont été lancés dans les années 90 par NVIDIA et le GPGPU (General Purpose Computing on the Graphics Processing Unit) et sont apparus dans les années 2000, avec une explosion de ces technologies vers 2005-2006 grâce à l'arrivée de standard comme l'Open CL et des technologies plus propriétaires comme CUDA de Nvidia ou ATI Stream de AMD. Tout cela a permis de lancer des calculs basiques sur de nombreuses données en parallèle. Le calcul par le GPU s'est imposé comme un standard de l'industrie. Des millions d'utilisateurs en profitent dans le monde entier et la majorité des constructeurs de matériel l'ont adopté.

Le calcul par le GPU permet de paralléliser les tâches et d'offrir un maximum de performances dans de nombreuses applications : le GPU accélère les portions de code les plus lourdes en ressources de calcul, le reste de l'application restant affectée au CPU. Les applications des utilisateurs s'exécutent donc bien plus rapidement.

Aujourd'hui, les processeurs graphiques les plus évolués contiennent plus de 1000 cœurs graphiques pour le calcul en parallèle. Cette parallélisation des calculs permet de gagner beaucoup de temps dans le domaine du traitement d'images. En effet, en utilisant les bibliothèques appropriées (comme CUDA chez Nvidia), on peut accélérer considérablement les calculs.

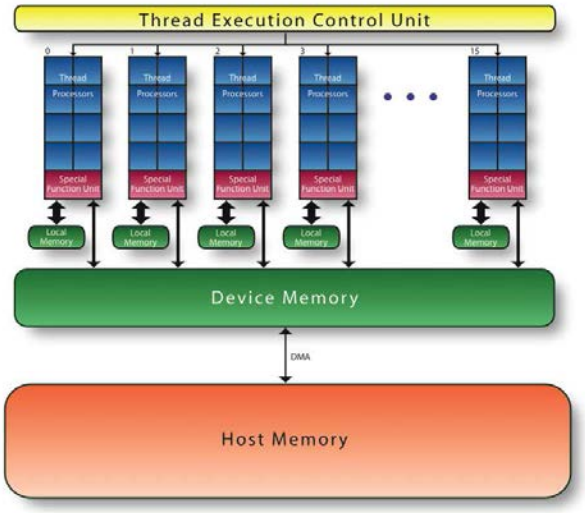


Figure 6 Architecture d'un GPU

Nous retrouvons les nombreux cœurs en bleu (Figure 6) et on peut voir qu'ils possèdent une mémoire par groupe de cœurs (ou multiprocesseur) et une mémoire partagée avec tous les multiprocesseurs. Cette mémoire partagée est reliée à l' « Host Memory » ou mémoire de l'hôte.

FPGA : Field Programmable Gate Array

Les FPGA (Field Programmable Gate Arrays ou "réseaux logiques programmables") sont des composants entièrement reconfigurables, ce qui permet de les reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs.

L'avantage de ce genre de circuit est sa grande souplesse, qui permet de les réutiliser à volonté dans des algorithmes différents en un temps très court.

Le progrès de ces technologies permet de faire des composants toujours plus rapides et à plus haute intégration, ce qui offre la possibilité de programmer des applications importantes.

Les circuits FPGA sont constitués (Figure 7) d'une matrice de blocs logiques programmables, entourés de blocs d'entrées/sorties programmables. L'ensemble est relié par un réseau d'interconnexions programmables. Les FPGA sont bien distincts des autres familles de circuits programmables tout en offrant le plus haut niveau d'intégration logique.

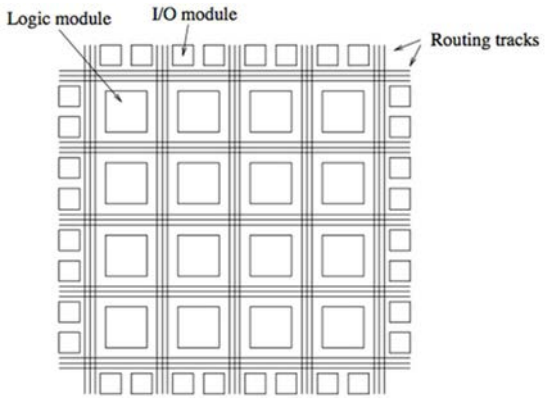


Figure 7 Architecture générique d'un FPGA

Le fait de programmer directement les portes logiques, permet de réaliser certains calculs beaucoup plus rapidement et d'être très souple. On peut faire jouer au FPGA le rôle d'un DSP, d'un Frame Grabber ou encore d'un calculateur d'algorithmes parallélisés.

1.1.2.2. Comparaison de performance par les benchmarks en traitement d'images

Comme décrit précédemment, un certain nombre d'études comparatives ont été menées sur les performances d'algorithmes de traitement d'images sur des architectures hardwares différentes (CPU, GPU, FPGA). Ces études permettent de faire ressortir les points faibles et les points forts de ces architectures pour du traitement d'image mais aussi de montrer quelles sont les spécificités qui influent le plus sur la performance.

Etude comparative de performance en traitement d'images par Cope et al. :

Dans [6], le but de Cope et al. est de comparer les performances de CPU, GPU et FPGA sur de l'accélération de traitement vidéos. Pour cela, il s'agit d'implémenter un certain nombre d'algorithmes de traitement vidéos ayant des caractéristiques très diverses, pour permettre d'évaluer sur quels types de traitements, les architectures sont les plus efficaces. D'un point de vue théorique, ces architectures ont certains avantages et inconvénients les unes par rapport aux autres, comme cela est résumé sur les Figure 8 et Figure 9.

	Cycles per Output	Iteration Level Parallelism	Clock Rate
General Purpose Processor	Low	Low	High
Graphics Processor	Medium	Medium–High	Medium
Reconfigurable Logic	High	High (Flexible)	Low

Figure 8 Comparaison CPU/GPU/FPGA [6]

General Purpose Processor	GPU
Large Overhead in Extracting Parallelism from an Algorithm	Parallelism is Inherent to the Fixed Data path of Multi-Processors
Instruction Inefficiency (Program Counter (PC), Flow Control, Data Load/Store)	SPMD (shared PC), *Increased Flow Control Overheads*, Data Load/Store Hidden by Multi-Threads (*Sometimes*)
Large Instruction Count	Specialised Vector ISA[1]
High Ratio of Cache to ALUs (Low Computational Density)	Low Ratio of Cache to ALUs (High Computational Density)

[1] The GeForce 8 and 9 generations of GPUs support only a scalar ISA

Figure 9 Comparaison CPU/GPU [6]

Pour ce faire, ils utilisent les matériels les plus performants dans les années 2005-2006 :

- 3 FPGA : Spartan 3 (gamme low cost), Virtex II Pro (haut de gamme ancienne génération), Virtex 4 (haut de gamme génération 2005-2006).
- 2 GPU : Geforce 6800 GT (ancienne gamme), GeForce 7900 GTX (haut de gamme génération 2006).

Différents types d'algorithmes sont étudiés pour connaître l'influence des paramètres sur les performances. Ces cinq algorithmes (Primary color correction, 2D convolution, Video Frame resizing, Histogram Equalization and 3 step Non-Full-Search Motion Vector Estimation) sont très différents et permettent d'avoir un maximum de cas qui influencent de façon différente les trois caractéristiques que sont : la complexité algorithmique, les accès mémoires et les dépendances de données (Figure 10).

	Arithmetic Complexity			Memory Access Requirements				
Algorithm	Math	Branch	Vector	Number (per output†)	Memory Intensity	Pattern	Reuse Potential	**Data Dependence**
Primary Colour Correction	low	low	medium					
Input Correction (IC)	medium	low	medium	1	low	predictable	$\frac{0}{1}$	null
Histogram Correct (HC)	medium	high	low					
Colour Balance (CB)	medium	medium	medium					
2D Convolution	null	null	medium	N^2	high	predictable	$\frac{N(N-1)}{N^2}$	low
Video Frame Resizing	null	null	high	16	medium	predictable	$\frac{0}{16}$ to $\frac{16}{16}$ (variable)	low
Histogram Equalisation								
Generation (256 bins)	null	null	low	$XY^{\ddagger}$		predictable	$\frac{XY}{XY}$	high
Intensity Equalisation	null	null	null	$1+1^{\S}$	high	random	$\frac{0}{1}+\frac{\sigma^{\sharp}}{1^{\S}}$	null
3-step Non-Full-Search Motion Vector Estimation	low	medium	high	$16^2(9+\ldots 8+8)+16^2$	medium	locally random	$\frac{44(44-16)}{44^2}$, $\frac{0}{16^2}$	medium

† The term output is used to indicate that for each function the output may not be a pixel in an output frame. For Histogram Equalisation Generation there are 256 bin outputs from the algorithm. For 3-step NFS-MVE there are $\frac{X}{16} \times \frac{Y}{16}$ motion vector outputs from the algorithm. For all other algorithms the outputs are individual pixels in an output frame. ‡ The product XY is the input frame resolution. § A single memory access from a 256 location lookup table. ♯ The variable σ indicates a dependence on the variance of the intensity when accessing the 256 location lookup table mentioned in note §.
Key: N - dimensionality of 2D convolution ($N \times N$), X,Y - respectively the width and height of the video (measured in pixels).

Figure 10 Description des architectures étudiées par Cope et al. [6]

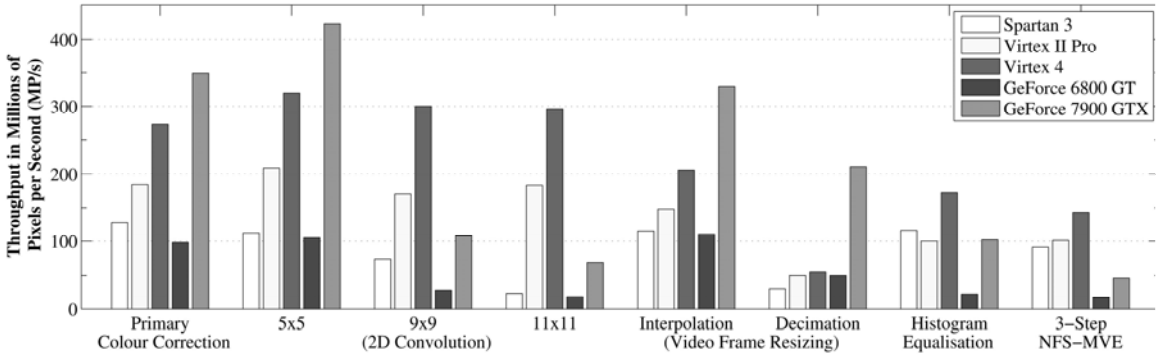


Figure 11 Résultats de performance des algorithmes de traitements de Cope et Al. [6]

La capacité d'un GPU ou d'un FPGA à accomplir des opérations plus rapidement que le GPP est dépendante de sa capacité de parallélisme. Cope et al. [6] ont analysé les paramètres influençant les performances dans le but de mettre en relief des résultats.

L'influence de la complexité algorithmique

Le nombre d'instruction opérationnelle requis d'un GPP est moins important que celle d'un GPU. Bien que le GPU possède un nombre d'instruction plus important (50% de plus), il a par contre deux fois moins de CPI (cycle per instruction), et douze fois moins de CPO (cycle per output) que le GPP. Cela étant dû au fort potentiel de parallélisation du GPU.

Le nombre d'opérations requis pour un FPGA est trois à quatre fois supérieur que celui requis pour le GPP. Mais le nombre d'instructions pour un GPP est quatre fois supérieur à celui requis pour un FPGA et ce nombre d'instructions est même encore inférieur pour le GPU par rapport au FPGA. Par contre, quand le GPU est au maximum 280 fois plus rapide que le GPP, le FPGA quant à lui est 445 fois plus rapide que le GPP. Cette différence est due à l'inefficacité du GPU ainsi qu'au fort niveau de parallélisation du FPGA.

On constate que l'inefficacité du GPU est bien inférieure à celle du GPP. Mais on peut aussi voir que l'évolution du CPI et l'inefficacité confrontée à l'augmentation de la taille de la convolution 2D (algorithme de traitement d'images testé par Cope et al.) ne permettent pas de justifier les résultats de performance obtenus. C'est l'étude des accès mémoires (données constructeurs) qui permettra de les justifier.

L'influence des accès mémoires

L'utilisation de l'On-Chip ou de l'Off-Chip Memory : La vitesse de transfert vers la mémoire interne (on-chip memory) est plus importante sur le FPGA. Par exemple,

le Virtex 4 a une vitesse de 288GB/s et le GPU 7900 GTX de 68GB/s. Ce qu'il est important de noter, c'est que les accès mémoires requis pour un même algorithme peuvent être différents sur GPU et sur FPGA. En effet, le FPGA va plus se servir de son on-chip memory et moins de son off-chip memory que le GPU, du fait des dépendances de données.

La bande passante : Le nombre d'accès mémoires est le même sur FPGA que sur GPU. Mais les performances du GPU sont fortement influencées par ce nombre d'accès mémoires, du fait de sa plus faible bande-passante. Le FPGA, quant à lui, n'est pas influencé par cette bande-passante. Cependant, on constate une légère baisse de performance due à un temps de latence qui augmente. La performance est pratiquement linéaire pour le FPGA grâce à sa flexibilité de parallélisation et la capacité de son on-chip memory supérieure au GPU.

La réutilisation des données : le potentiel de réutilisation des données dans l'algorithme « Resizing » permet de constater son influence. En effet, celui-ci a une influence sur les accès mémoires, sur l'IBW (Input memory BandWidth) et donc sur le CPO. Malgré le fait que le CPO du FPGA soit bien inférieur à celui du GPU, celui-ci ne permet pas de compenser l' « Iteration Level Parallelism » (ItLP) qui est 24 fois plus important sur le GPU.

L'influence des dépendances de données :

Ce paramètre est une caractéristique de l'algorithme et non de l'architecture. Si chaque étape dépend du résultat de la précédente, la dépendance de données sera très importante et à contrario, si aucune des étapes n'est dépendante des résultats des précédentes alors il n'y a aucune dépendance de données.

Ces méthodes qui permettent de réduire les accès mémoires sur le GPU ne compensent pas la capacité du FPGA à allouer la mémoire interne dont elle a besoin pour avoir le moins possible d'accès à la mémoire externe. Et même si l'ItLP du FPGA n'est que de quatre pour l'algorithme « 3-step NFS-MVE », cette capacité permet d'avoir une performance bien supérieure sur FPGA que sur GPU.

Comparaison du choix d'une architecture pour une application bien précise

Dans [7], Grozea et al. étudient différentes possibilités de calculateurs hautes performances « low cost » pour une application bien précise. Il est question d'implémenter un réseau d'IDS (Intrusion Detection System) qui détecte des anomalies (buffer over- flow attacks or web application exploits). Le but est d'accélérer la séquence d'algorithme de comparaison en utilisant différentes architectures hardwares que sont les FPGA, les CPU multi-cœur (2, 4, 8) et les GPU

comprenant de nombreux cœurs graphiques (64 128 240). Il s'agit de traiter 84 000 paquets par seconde.

Trois matériels ont été utilisés pour représenter les trois architectures hardwares décrites plus haut : un FPGA Xilinx Virtex-5 XC5VFX70T sur une board d'évaluation ML507, d'une fréquence maximum de 0.55 GHz et 70 miles logic cells ; un Dell Precision T7400 avec deux CPU Xeon 5472 quad-core, fréquence 3GHz et 16GB de RAM ; deux GPU Nvidia Quadro FX 5600, avec 1.5 GB de RAM, 128 shaders CUDA chacun et une fréquence de 1.35 GHz.

Ces trois configurations correspondaient aux configurations les plus haut de gamme de leur époque. Pour ce qui est du processeur, la librairie openMP a été utilisée pour le calcul en parallèle sur une base de code C/C++. Quant au GPU, la librairie CUDA propre aux cartes graphiques a été utilisée pour répartir les calculs sur les différents cœurs graphiques. Le FPGA était connecté à un PC host en PCIe qui lui envoyait les données à traiter et qui recevait ensuite les résultats.

L'implémentation de l'algorithme de comparaison a été décrite comme plus compliquée par Grozea et al. En effet, deux algorithmes ont été implémentés sans succès (« The complex sort-merge sorting » et « The bitonic sort »), avant que l'algorithme « The insertion sort » fonctionne. Les algorithmes demandaient trop de puissance.

Au final, il est apparu que la solution sur FPGA était trop lente, dû au fait de la communication entre le CPU et la board FPGA (1Gbit/s), même si le temps d'exécution sur la board était rapide. De plus, l'implémentation sur FPGA est très flexible mais elle est la plus difficile à prendre en main. Aujourd'hui, l'implémentation du PCI Express 8x 2.0 est possible, contrairement au 1X utilisé dans l'expérience et cela permettrait donc d'avoir une bande passante bien supérieure.

La solution GPU ne convenait pas pour l'application étudiée, dû au temps de latence trop long. Mais pour des calculs plus longs, où le temps de latence est négligeable devant le temps de calcul total, les performances du GPU sont alors bien supérieures. Le bémol signalé dans cet article porte sur la librairie CUDA qui selon eux, n'est pas très stable et difficile à prendre en main.

Enfin, la solution sur CPU est celle qui convenait le mieux car le temps de traitements répondait totalement aux attentes et la librairie OpenMP est très simple à prendre en main. Cette solution a permis d'avoir une bande passante importante et un temps de latence faible.

L'analyse de Grozea et al. montre bien les difficultés de comparer la performance d'une application sur des plateformes très différentes et permet de comprendre que ces tests de comparaison auront souvent une certaine subjectivité selon les préférences

du testeur. En effet, l'implémentation ne sera pas forcément du même niveau selon le type de plateforme car l'équipe qui s'en occupe est rarement composée d'experts de niveau identique dans chacun des domaines. On voit donc que le choix d'architecture contient une part de subjectivité.

1.1.3. Conclusion

Les deux comparaisons de performance d'architectures hardwares [6] et [7], comme beaucoup d'autres d'ailleurs [8] [9] [10] [11] [12] [13], montrent bien la difficulté de comparer des architectures si différentes que sont le CPU, le GPU et le FPGA. Il paraît donc indispensable de se concentrer sur un premier modèle du système de calcul (défini sur la Figure 1) qui ne prenne en compte qu'une seule grande famille d'architecture.

Ces différentes études mettent aussi en avant que l'implémentation sur des architectures de types GPU et FPGA influe plus sur la performance. Il y a une certaine part de subjectivité dans l'implémentation d'algorithme de traitement d'images sur ce type de plateforme. Le facteur humain semble être un facteur important dans le choix d'une architecture.

L'architecture CPU parait être le choix le plus logique pour initialiser une étude, de par sa simplicité vis-à-vis des architectures GPU et FPGA.

1.2. Modélisation de la performance

L'évaluation et la modélisation de la performance d'un système de calcul est un sujet traité par de nombreux chercheurs. Nous allons voir dans un premier temps, ce qui ressort de l'état de l'art sur la mesure, au sens instrumentation, et sur la caractérisation d'un système de calcul. Nous avons, ensuite, regroupé les méthodes et paradigmes proposant d'évaluer la performance, en deux ensembles. Nous présentons, d'un côté, les méthodes stochastiques et de l'autre, les méthodes déterministes. Nous verrons qu'un certain nombre de ces méthodes sont plutôt hybrides, surtout les méthodes que nous considérons comme déterministes en théorie mais qui le sont rarement à 100% dans les faits.

1.2.1. Performance d'un système de calcul : Mesure et caractérisation

La mesure de la performance a toujours été importante pour identifier les problèmes de celle-ci dans un système de calcul et prévenir les futurs problèmes de performance du système en question, qui devront être gérés.

Les constructeurs de processeurs sont les premiers à mesurer la performance, souvent à l'aide de benchmark dans un environnement très contrôlé. Ils veulent démontrer la haute performance de leurs systèmes et aider leurs utilisateurs à configurer au mieux leurs applications. Evidemment, les chercheurs universitaires mènent également de nombreux travaux sur la mesure de la performance. Ces études sont souvent menées de manière plus approfondie, les systèmes proposés sont implémentés, testés et étudiés de façon empirique. L'étude et la caractérisation de la performance permettent également de pouvoir valider un modèle d'évaluation de performance algorithmique. Heidelberger et Lavenberg [14] se sont concentrés sur trois sujets d'étude de la performance que sont, l'instrumentation, la caractérisation de la charge de calcul et l'étude statistique de la performance.

1.2.1.1. L'instrumentation

L'instrumentation d'un système peut se faire de façon hardware ou software. Un système de contrôle hardware est généralement externe. Il n'interfère donc pas dans le système mesuré et n'altère pas sa performance. Le système de contrôle software, pour sa part, est un ensemble d'instructions ajoutées au système qui permet de rassembler des données de performance. Par exemple, il lit les emplacements mémoires, détecte les évènements, les accès mémoires ou encore les interruptions des entrées/sorties. Contrairement au système de contrôle hardware, le système de

contrôle software va altérer les performances du système. Il faut donc le prendre en compte dans l'évaluation de la performance.

De ce fait, le système de contrôle hardware est précis et rapide tandis que le software est flexible et facilite l'accès aux données du software étudié. Des systèmes hybrides ont donc été développés pour combiner les avantages des systèmes de contrôle hardware et software.

Heidelberger et Lavenberg citent de nombreux systèmes de monitoring comme le Multics System, le DIAMOND, le XRAY, le S6000ou encore le Zahlmonitor III.

1.2.1.2. La caractérisation de la charge de calcul

Que ce soit pour l'étude de la mesure de performance ou pour la modélisation, la caractérisation de la capacité de traitements est un élément important à prendre en considération. On peut distinguer trois types de capacité de traitements : « live workload », « executable or real workload » et « synthetic workload ».

Pour caractériser une charge de traitement « réelle », Heidelberger et Lavenberg proposent cinq étapes :

1. La sélection de l'élément à caractériser,
2. La sélection des spécifications utilisées qui peuvent être hardware (instructions processeurs, espace mémoire,…) ou software (nombre d'appel au compiler, gestionnaire de fichiers,…),
3. La mesure de la charge de traitement (obtenue à l'aide d'un grand nombre d'éléments),
4. L'analyse des données (ressortir les spécifications de chaque élément étudié),
5. Le groupement des données (partitionnement des éléments dans des groupes ayant les mêmes spécifications).

1.2.2. Les modèles stochastiques

Simonetta Balsamo et Al. [15] ont répertorié et classifié les méthodes permettant d'améliorer la performance d'un software en la prédisant de manière stochastique, le plus en amont possible dans le processus de développement d'un logiciel. Ils ont répertorié la majorité des ensembles de méthodes qui permettent de modéliser ou simuler la performance d'un logiciel développé. L'idée est d'intervenir à différents niveaux de cycles de développement d'un logiciel considérant le modèle générique décrit sur la Figure 12.

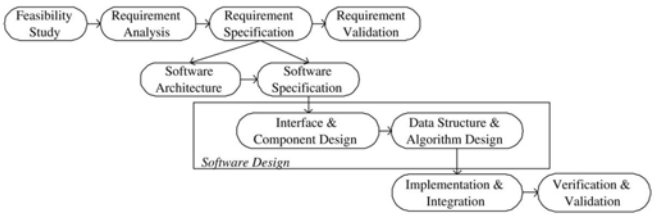


Figure 12 Modèle générique du cycle de vie d'un logiciel [15]

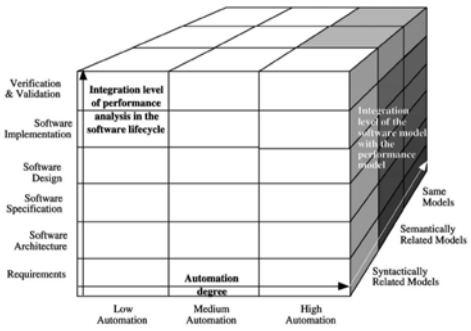


Figure 13 Classification des approches de performance logicielle[15]

En plus du niveau du cycle de développement, cette étude se base sur deux indicateurs que sont le niveau d'intégration du modèle logiciel avec le modèle de performance et le degré d'automatisation de la méthodologie. Ce qui nous donne donc une classification à trois dimensions (Figure 13) des différentes méthodes.

Les méthodes sont classées selon cinq grandes catégories décrites ci-dessous.

1.2.2.1. Méthodes basées sur la théorie des files d'attente (Queueing Network)

Parmi les méthodes de Queueing Network, il existe quatre grandes familles que sont l'approche « SPE » (Software Performance Engineering), le patron de conception (« Architectural Pattern »), les analyses de traces et l'UML pour la performance.

Approche SPE (Software Performance Engineering)

La première méthode basée sur la SPE a été proposée par William et Smith [16] (**M1**) pour évaluer les performances d'une architecture logicielle en utilisant le langage UML (Diagrammes de Classe, diagrammes de séquence, etc.). Cette méthode qui s'applique dans la phase Software Design (Figure 12) a été concrètement utilisée dans l'outil SPE.ED[17] permettant de modéliser la performance d'une architecture logicielle. L'utilisateur décrit les étapes de traitements, spécifie le nombre de ressources logicielles requises pour chaque étape et l'outil génère automatiquement un modèle QN (Queueing Network). Ensuite, il évalue le temps de traitement complet, le temps passé pour chacun des process et l'utilisation du système. D'autres modèles basés sur la SPE ont été développés comme le PRIMA-UML [18][19] (**M2**), qui génère un modèle de performance à partir de différents diagrammes UML (« Deployment Sequence » et « Use Case Diagrams »). Ces diagrammes UML sont

enrichis avec respectivement, la distribution des charges de travail et les paramètres des architectures hardwares. Une extension au modèle PRIMA-UML [20] (**M3**) a aussi été développée pour supporter le cas de l'architecture logicielle mobile.

Enfin, une dernière méthode basée sur l'approche SPE (**M4**) a été développée par Cortellessa et Al. [21] à partir du modèle LQN (Layered Queueing Network) en utilisant des outils CASE (Computer Aided Software Performance Evaluation) standards.

« Patterns » de conception

Ces méthodes utilisent des « Patterns » pour décrire une architecture logicielle et chaque pattern est qualifié par sa structure (ses composants) et son comportement (comment il interagit). Gomaa et Menascé ont présenté la première approche basée sur les « architectural Pattern ».

Petriu et Wang (**M5**) ont proposé trois concepts similaires à [22] qui décrivent l'architecture logicielle à l'aide des « architectural patterns ». Ces trois approches suivent également la méthode SPE décrite précédemment. La première méthode utilise l'UML (collaboration, sequence, deployment and use case) pour spécifier l'architecture, la deuxième ajoute des profils de performance et le dernier utilise le XSLT (eXtensible, Stylesheet, Language Transformation).

Gomaa et Menascé ont aussi présenté une approche basée sur le CLISSPE (**M6**) [23] (CLIent/Server Software Performance Evaluation). Le système CLISSPE permet de générer un modèle QN (Queueing Network), d'estimer les paramètres de ce dernier ainsi que de fournir le modèle de performance associé.

L'analyse de traces

Ces approches sont basées sur la génération et l'analyse de séquences d'évènements/d'actions, basées elles-mêmes sur une description dynamique du système. Une des approches (**M7**) permet de générer des modèles QN à partir d'une description logicielle réalisée à l'aide de « Message Sequence Chart » [24] et de « Labeled Transition Systems » [25]. Woodside et al. (**M8**) ont décrit, pour leur part, une méthodologie qui permet de générer un modèle LQN à partir de l'outil PAMB (Performance Analysis Model Builder) [26] qui s'applique généralement sur des logiciels temps-réel. Après avoir résolu le modèle, l'environnement donne des résultats de performance par des graphes de prédictions et des MSC (Message Sequence Chart).

Plus récemment, Woodside et al. (**M9**) ont présenté une approche basée sur des modèles de performances LQN (Layered Queueing Network) décrits par des Use Case Maps (UCM) [27]. Un outil nommé UCM2LQN permet d'optimiser cette méthode.

L'UML pour la performance

Les précédentes approches utilisent beaucoup l'UML pour construire un modèle, qui leur permet de dégager la performance d'un système. Mais il est aussi possible d'utiliser l'UML pour directement caractériser la performance d'un logiciel. Le « UML Profile for Scheduling, Performance and Time » a été créé dans ce but. Kähkipuro (**M10**) a introduit une approche utilisant le profil UML [28] qui se découpe en trois étapes. Dans un premier temps, l'architecture logicielle est représentée en UML, elle est par la suite, automatiquement caractérisée par une représentation textuelle qui ne retient que les aspects de performances. L'architecture logicielle sera finalement transformée en un modèle QN. Ensuite, les résultats peuvent être retraduits dans l'autre sens jusqu'aux diagrammes UML. Cette méthode a été partiellement implémentée dans l'outil OAT (Object oriented performance modeling and Analysis Tool).

1.2.2.2. Méthodes basées sur les Process-Algebra

Pour décrire les propriétés fonctionnelles et de performance d'un logiciel, il est possible d'utiliser des processus stochastiques algébriques (SPA = Stochastic Process Algebra) en utilisant la méthodologie décrite Figure 14 par Hermanns et Al. [29].

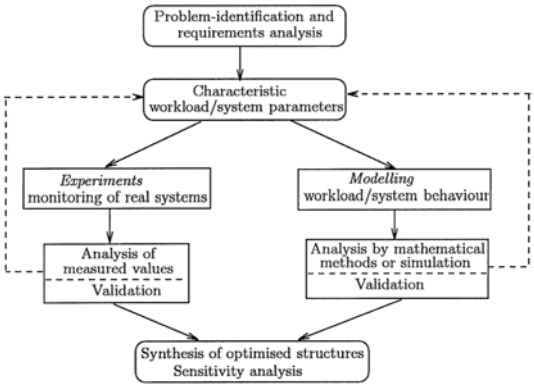


Figure 14 Méthodologie d'évaluation de la performance par des concepts algébriques[29]

De nombreuses approches algébriques stochastiques (**M11**) ont été définies pour décrire cette performance, comme par exemple le TIPP (TIme Processes and

Performability evaluation), l'EMPA (Extended Markovian Process Algebra) ou le PEPA (Performance Evaluation Process Algebra). Chacune de ces approches a d'ailleurs un outil associé.

1.2.2.3. Méthodes basées sur les réseaux de Petri

Les méthodes de réseau de Petri stochastiques (SPN=« Stochastic Petri Nets ») (**M12**), permettant de réaliser de la prédiction de performance, sont assez nombreuses (GreatSPN, HiQPN, DSPNExpress 2000, etc.). Elles permettent d'analyser les propriétés d'un système de façon fonctionnelle et non-fonctionnelle pour le modéliser. Récemment, des approches ont été proposées pour combiner l'UML et le réseau de Petri [30][31]. On peut citer le GSPN (General Stochastic Petri Net), dérivé des diagrammes de cas d'utilisation (« Use Cases Diagram »), qui est une combinaison de diagrammes de communication (« collaboration diagrams ») et de diagrammes d'états (« Statechart diagrams »).

1.2.2.4. Autres Méthodes

Miguel et al. (**M13**) proposent une approche basée sur une extension des diagrammes UML [32], qui permet de prendre en compte les besoins temporels et les ressources utilisées, pour qu'elle soit appliquée sur des systèmes temps-réel. Arief et Speirs (**M14**) ont proposé un Framework [33] nommé SimML (Simulation Modeling Language) qui génère automatiquement un programme de simulation Java en permettant à l'utilisateur de dessiner les diagrammes de classes et de séquences de son logiciel.

Lindemann et Al. (**M15**) ont proposé une approche [34] pour passer d'un diagramme d'état ou d'activité vers un système stochastique d'évènement discret automatiquement nommé semi-Markov. Elle a été implémentée en utilisant l'outil DSPNexpress 2000.

1.2.2.5. Comparaison des méthodes

Nombre de ces méthodes utilisent l'UML pour modéliser l'algorithme car cela permet d'avoir une vue d'ensemble de celui-ci et faire ressortir les problèmes. Le réseau de files d'attente est le modèle de performance le plus utilisé. Toutes les méthodes utilisant le SPE supposent d'avoir les performances accessibles à chaque phase de vie du logiciel. Grâce à cette étude, il est possible de se représenter la phase du projet sur laquelle les différentes méthodes peuvent être utilisées, leur degré d'automatisation ou encore leur domaine d'application. La Figure 15 permet de

visualiser quel est le champ d'application de chacune des méthodes décrites (M1 à M15).

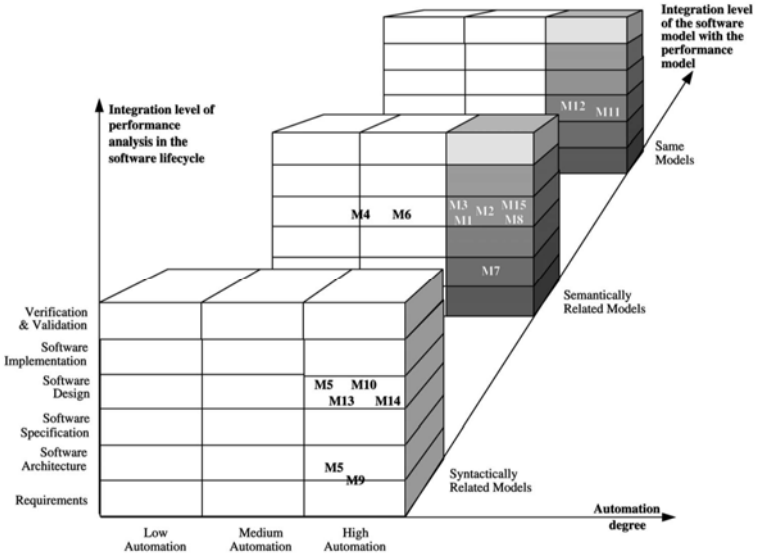


Figure 15 Classification des méthodes d'évaluation de la performance [15]

1.2.2.6. Autre exemple d'application d'un modèle stochastique

Ould-Ahmed-Vall et al. [35] utilisent un modèle de régression statistique pour évaluer la performance d'un système basé sur l'étude des « Hardware events count » pendant l'exécution de différents calculs. Différents outils de mesure permettent de les utiliser comme par exemple l'outil HP Caliper [36], proposé par Robert Hundt de la société Hewlett-Packard. Cette méthode, basée sur un modèle statistique, essaye de résoudre deux sous-problèmes que sont l'identification des problèmes de performance pour donner des conseils d'amélioration et l'estimation du gain de performance obtenue grâce aux modifications réalisées à partir de ces conseils.

La solution envisagée se base sur le modèle statistique M5', qui est elle-même, basée sur un arbre de décision. Cette méthode M5' partitionne de manière récursive l'espace d'entrée en sous-ensembles homogènes, de sorte que l'ajustement linéaire au niveau des nœuds puisse expliquer la variabilité restante.

La métrique de performance Y choisie est dépendante d'un set de prédicteur (X_1 à X_n). Y est exprimé en nombre de cycles par instruction (CPI).

$Y = f(X_1, X_2, \ldots, X_n) +$ avec étant l'erreur.

Les arbres de décision présentent de nombreux avantages. Ils offrent une prédiction précise ; ils fournissent non seulement les problèmes de performance mais aussi leur

gravité et ils offrent également la possibilité d'avoir un grand nombre de caractéristiques étudiées.

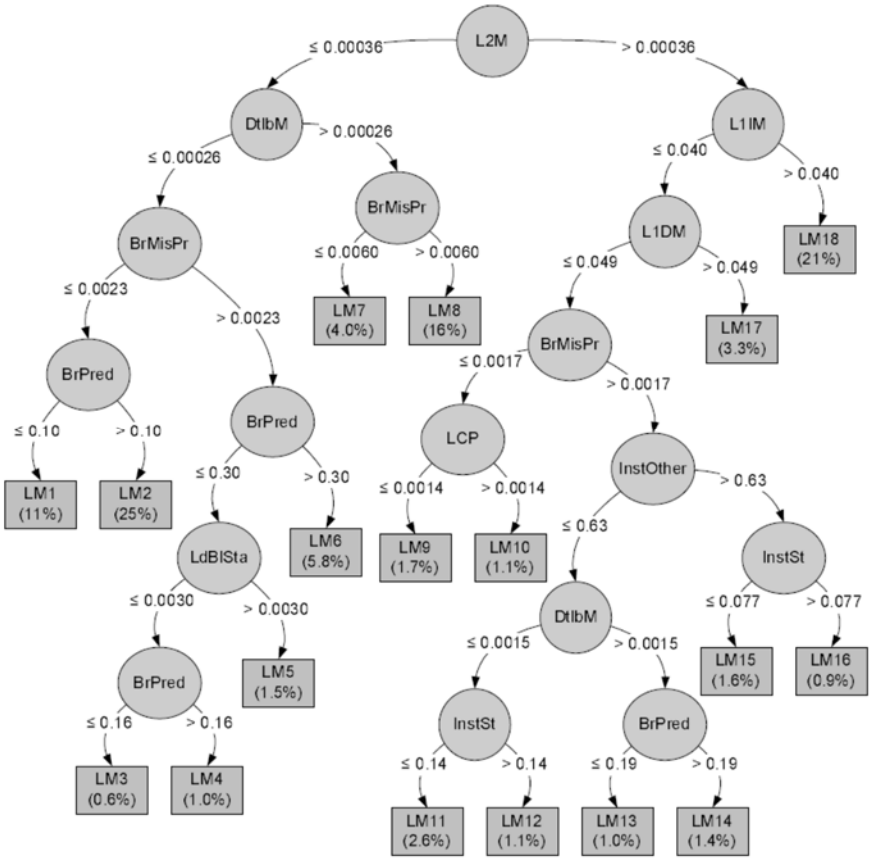


Figure 16 Exemple d'arbre M5' pour l'analyse de performance[35]

Sur la Figure 16, on peut voir un exemple d'arbre permettant l'analyse de la performance. Chaque nœud correspond à une métrique bien définie tandis que chaque feuille correspond aux résultats découlant du résultat des métriques. L2M correspond par exemple au nombre de « L2 cache misses » par instruction. L1IM et L1DM correspondent respectivement aux « L1 cache misses » de données et d'instructions par cycle.

L'algorithme pris en exemple par Ould-Ahmed-Vall et al., est non seulement modélisé mais aussi évalué à l'aide de trois métriques, qui comparent les résultats expérimentaux au modèle. Ces trois métriques sont le coefficient de corrélation qui est de 0,98 dans l'exemple, la MAE (mean absolute error) égale à 0,05 et l'erreur absolue relative qui est ici de 7,83%.

1.2.2.7. Conclusion

Toutes ces méthodes permettent de prédire la performance de tel ou tel algorithme mais elles ne sont pas mises en perspective avec les spécifications de l'architecture hardware utilisée. En effet, ces méthodes ne se placent que dans la phase de développement du software, pour rendre ce dernier le plus performant possible de façon intrinsèque. Les méthodologies répertoriées sont très performantes mais elles ne se placent pas du point de vue du hardware. Nous travaillons donc sur un axe différent. De ce fait, nous ne voulons pas remplacer ces méthodologies très avancées dans leur domaine mais en complément, d'essayer de comprendre quels sont les paramètres d'une architecture hardware qui vont influencer sur les performances du software. Finalement, ces méthodes sont très utiles dans la description de l'algorithme. Elles sont donc une perspective intéressante pour la caractérisation d'un algorithme de traitement d'images.

1.2.3. Les modèles déterministes

Un grand nombre de modèle déterministe ont été proposé pour permettre de jauger de la performance d'un système. Un certain nombre de ces modèles sont assez génériques tandis que d'autres sont très centrés sur la parallélisation des calculs (la capacité de parallélisation du hardware mais aussi du software). Ces modèles sont souvent présentés à travers des outils qui permettent d'utiliser le modèle en question.

1.2.3.1. Les méthodes généralistes

Le modèle PAPI

Browne et Al. [37] ont proposé en l'an 2000, un outil (PAPI) qui permet d'évaluer la performance d'un microprocesseur en étudiant les évènements comptés par les registres « counters ». Le principal objectif de PAPI est de fournir une interface qui permette de facilement avoir accès à ces « Performance counters » sur la majorité des plateformes CPU et de fournir aux développeurs des aides pour adapter leur logiciel sur différentes plateformes.

PAPI utilise un certain nombre de métriques permettant de caractériser un système, comme le niveau de hiérarchie mémoire, le « hit rates » des caches L1, L2, le nombre de données TLB (Translation Lookaside Buffer) manquées, l'état du cache MESI dans les environnements SMP, nombres d'instructions « load », « store », « integer », « floating_point » ou encore les différents évènements du pipeline.

L'architecture PAPI se décompose en un certain nombre de couches (Figure 17). La première couche qui se décompose en deux, n'est pas dépendante de la machine étudiée : les interfaces « high level » et « low level ».

L'interface « low level » contient toutes les API permettant d'accéder au gestionnaire d'état, au gestionnaire de mémoire, à la manipulation des structures de données et à la sauvegarde des threads. Elle procure aussi des options avancées comme le profilage ou la gestion de l'overflow. L'interface high level procure une interface simple d'accès et de contrôle des « hardware counters ».

Ces deux interfaces sont indépendantes de l'architecture étudiée. Cependant, pour fournir tous les outils d'analyse, elles doivent se reposer sur une sous-couche (PAPI Machine Dependent Substrate sur la Figure 17) totalement dépendante de la machine étudiée. En effet, pour chaque paire d'architecture/OS étudiée, une sous-couche doit être créée. Selon, Browne et Al., une semaine est souvent nécessaire pour générer cette sous-couche.

La méthode PAPI a été implémentée sur un certain nombre de plateformes que sont :

- les Pentium Pro II et III sur linux 2.x,
- les SGI MIPS R1000 et MIPS R2000 sur IRIX 6.x,
- les IBM power 604, 604e et 630 sur AIX 4.3,
- les COMPAQ ALPHA EV4 et EV5, EV6 sur TRU64 UNIX
- le CRAY T3E sur UNICOS/MK

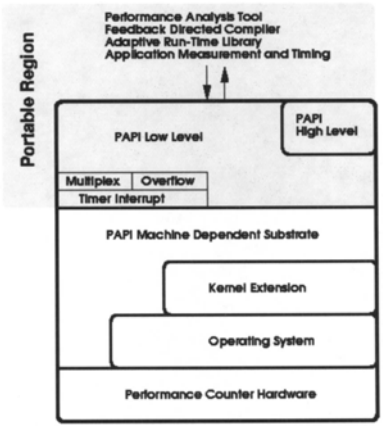


Figure 17 Architecture PAPI [37]

L'interface utilisateur de la méthode PAPI se décompose en deux outils. Le premier outil se nomme Perfometer et permet d'obtenir la performance de chaque métrique PAPI. Le deuxième outil, Profometer permet d'obtenir l'histogramme des occurrences d'Event PAPI pour chaque métrique.

Cette méthode permet d'obtenir un très grand nombre d'informations de performance sur l'architecture étudiée mais il demande énormément de travail préalable pour fonctionner. Par contre, un travail pourrait être réalisé sur la prédiction des résultats de ces outils en se basant sur des résultats d'architectures similaires existantes.

Méthode MANTIS

Une méthode de prédiction de performance d'application smartphone a été présentée à la conférence Annuel USENIX par Know et Al. [38]. Cette méthode possède certaines similitudes avec notre démarche, car elle se restreint à un domaine pour permettre d'être le plus précis possible. Elle se décompose en deux versions « Offline » et « Online ». L'étape « Offline », qui permet de déterminer une fonction d'approximation du temps d'exécution du programme, est décomposée en quatre étapes (Figure 18).

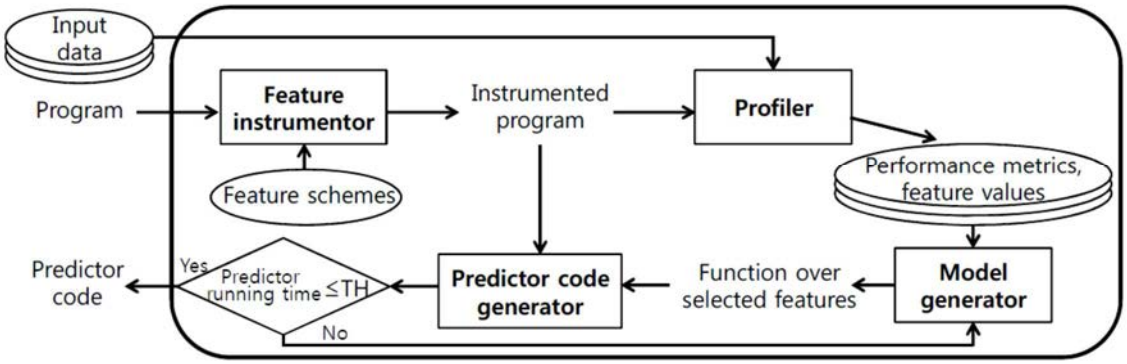


Figure 18 Etapes « offline » de la méthode MANTIS [38]

La première étape, « Feature instrumentor », va analyser le programme pour déterminer le nombre de « branch » (If, Else), le nombre de « loop » (boucle de calculs), le nombre de « method call » (procédure associée à un objet), les « variable values » (définition de la somme et de la moyenne de chaque mise en mémoire) et d'autres spécifications comme les valeurs les plus utilisées pour chaque variable. Cette étape permet d'obtenir le programme « instrumenté ».

La seconde étape, « Profiler », permet de lancer le programme instrumenté avec différentes valeurs d'entrées, fournies par l'utilisateur. Pour chaque donnée d'entrée, l'outil va déterminer certaines spécifications propres à la valeur d'entrée (Features value) et le temps d'exécution avec cette donnée (Performance metric).

La troisième étape, « Model generator », permet de produire une fonction , qui approxime le temps d'exécution avec une partie des spécifications définies précédemment. L'échantillon en question est défini à l'aide de l'algorithme SPORE-

FoBa, qui a été choisi parmi d'autres (FoBa et LASSO) de part sa meilleure performance.

La quatrième et dernière étape, « Predictor code generator », permet d'évaluer automatiquement les spécifications choisies (étape 3) à partir du programme instrumentalisé déduit de l'étape 1. Cette évaluation est réalisée à l'aide de « Static program slicing » [39].

Le Framework basé en quatre étapes, décrit ci-dessus, a débouché sur un prototype qui fonctionne avec des applications Android (Figure 19).

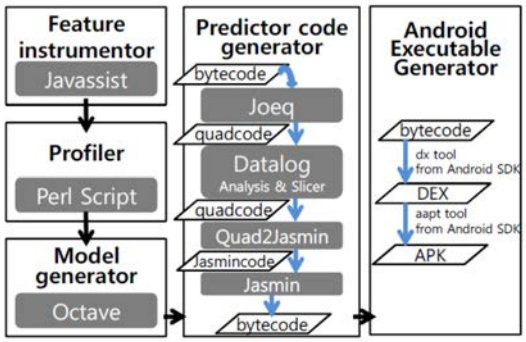


Figure 19 Prototype MANTIS [38]

Know et Al. [38] ont donc testé leur prototype sur six différents algorithmes, Encryptor (Encryptions de fichier avec une matrice jouant le rôle de clé), Path routing (Définition du plus court chemin), Spam Filter (Filtre des messages spams de la messagerie), Chess Engine (Application d'Echec), Ringtone Maker (Génération de sonnerie) et Face Detection (Détection de visages avec OpenCV), qui permettent de couvrir un large panel de fonctionnalités Android. Ces différents algorithmes ont été testés sur trois smartphones différents : Galaxy Nexus (Dual-Core, 1.2Ghz, 1GB RAM, Android 4.1.2), Galaxy S2 (Dual-Core, 1.2Ghz, 1GB RAM, Android 4.0.3), et Galaxy S3 (Quad-Core, 1.4Ghz, 1GB RAM, Android 4.0.4). Chaque algorithme a ensuite été testé avec 1000 inputs différents générés aléatoirement.

Application	Prediction error (%)	Prediction time (%)	No. of detected features	No. of chosen features
Encryptor	3.6	0.18	28	2
Path Routing	4.2	1.34	68	1
Spam Filter	2.8	0.51	55	1
Chess Engine	11.9	1.03	1084	2
Ringtone Maker	2.2	0.20	74	1
Face Detection	4.9	0.62	107	2

Figure 20 Résultats des tests du Mantis [38]

On peut voir dans la Figure 20 que les résultats sont très bons. Non seulement l'erreur de prédiction est faible (entre 2,2 et 11,9%) mais le temps d'exécution du « Predictor code generator » (Figure 19) ne représente qu'aux alentours de 1% du temps

d'exécution de l'application elle-même. Il ressort que le taux d'erreur devient constant au bout de moins de 50 inputs pour quatre algorithmes mais nécessite plus de 100 inputs pour se stabiliser avec les deux derniers algorithmes. Cet outil montre donc sa haute performance sur des algorithmes très différents mais toujours développés sur Android et lancés sur un Smartphone muni d'un CPU ARM. De plus, ce n'est pas un outil qui permet de se passer de l'architecture sur lequel on veut réaliser des tests. Il permet de déterminer le temps d'exécution en fonction de données d'entrée mais ne prend pas en compte des métriques comme la consommation de ressources. Malgré tout, l'équipe en charge de ce projet travaille en ce moment sur ce point.

Représentation Y-Chart

Bontempi et Kruijtzer [40] ont proposé une procédure pour déterminer le modèle de prédiction d'un logiciel basé sur l'analyse des paramètres de l'architecture hardware et du logiciel étudié. Cette procédure est décrite comme étant spécialisée pour des applications embarquées.

Le modèle est basé sur une représentation Y-Chart qui se décompose en quatre étapes pour établir une modélisation : 1. Spécification du domaine fonctionnel, 2. Spécification de l'architecture candidate, 3. Mapping des tâches fonctionnelles et des éléments de l'architecture et 4. L'estimation des résultats du design. A partir de représentation Y-Chart connue, ils ont développé un modèle en huit étapes (Figure 21) qui permet de réaliser le modèle de prédiction.

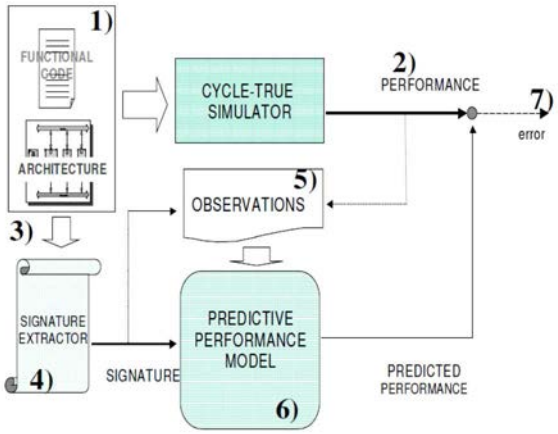


Figure 21 Procédure d'estimation du modèle de prédiction du logiciel [40]

1. La première étape permet de définir la famille d'applications qui va représenter le spectre de fonctions et d'architectures le plus large possible.

2. Définition des critères de performance les plus pertinents.

3. 4. « Signature Extraction » : Extraction des paramètres du système qui représente la performance ou plutôt la capacité de calcul du système à l'aide d'une étude de la complexité.

5. Collecte d'un maximum de données provenant de l'extraction des paramètres (4) et de la définition des critères de performance (2).

6. Modélisation de la relation entre les entrées et les sorties à partir des données collectées en (5).

7. Validation du modèle : Modification du scénario pour constater le comportement du modèle.

Bontempi et Kruijtzer démontrent la véracité de leur procédure avec la réalisation d'un modèle sur la base d'algorithmes de tri et de benchmark multimédia sur un microprocesseur MIPS3000. Lorsque l'on modifie les algorithmes et que l'on conserve la même architecture, l'erreur de prédiction varie entre 0,06% et 19,3%. Par contre, lorsque l'on conserve les mêmes algorithmes et que l'on modifie l'architecture de calcul, en prenant en compte des signatures différentes, l'erreur de prédiction de performance varie alors entre 4,4% et 70,2%.

Méthode basée sur le Monte Carlo

Srinivasan et al. ont mis à jour [41] leur modèle de prédiction de performance basé sur Monte Carlo en 2006, pour inclure les « data-prefetching » et améliorer le modèle mémoire après l'avoir proposé dans [42]. Cette méthode est donc vraiment hybride car elle fait intervenir un modèle statistique. Au départ, ils ont appliqué le modèle Monte Carlo à la prédiction de performance, en prédisant le CPI (Cycle Per Instruction) à l'aide de statistiques de prédiction de branch, de cache ou encore de caractéristiques d'applications.

L'idée principale de ce modèle est de séparer le CPI en deux composantes : le CPI_i et le CPI_{st}. Le CPI_i correspond au nombre de cycle inhérent à l'application testée (théorique), tandis que le CPI_{st} correspond au nombre de cycle induit par une erreur de pipeline du processeur nommé « Processor stalls ». Ce CPI_{st} représente d'ailleurs généralement 60% du CPI et peut même représenter 80% du CPI global sur certaines applications. Différents facteurs influencent le CPI_{st} : la dépendance des données, le mélange des instructions et les évènements hardware (branch mis-prediction, cache misses, TLB misses, etc.).

Pour démontrer la véracité de leur modèle, Srinivasan et al. se sont servis d'une architecture Itanium-2 qui comprend un pipeline VLIW à huit niveaux, capable de résoudre six instructions par cycle et incluant trois niveaux de cache L1, L2 et L3.

Les applications étudiées ne sont pas simulées cycle par cycle (temps de traitement trop important) mais c'est un nombre d'éléments générés qui permet de calculer le nombre de cycle de façon probabiliste. Le modèle proposé est représenté Figure 22 ainsi que l'évolution de celui-ci, Figure 23, qui prend en compte le L2DTLB (Translation Lookaside Buffer) et le Prefetch Logic.

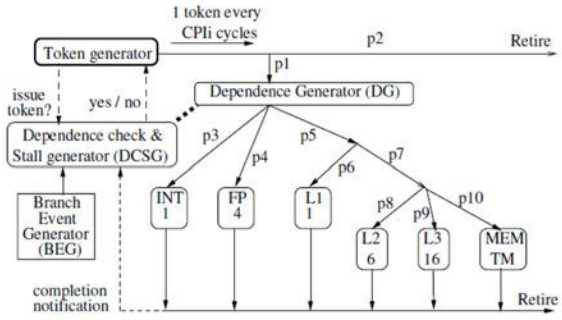


Figure 22 Modèle basée sur Monte Carlo [42]

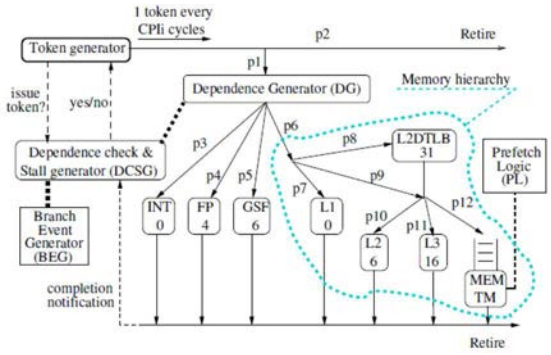


Figure 23 Evolution du modèle basée sur Monte Carlo [41]

Le processeur est représenté par le générateur de jeton (token generator). Un jeton correspond à une instruction normalement émise à chaque cycle. Soit, les jetons se retirent immédiatement (p2), soit ils entrent dans le « delay center » qui regroupe les différentes dépendances qui vont faire prendre du retard à une instruction. Chaque type de retard (integer ALU, memory hierarchy, etc.) a un nombre de cycle associé, ce qui permet de calculer le CPI_{st}.

« p_1 » à « p_{10} » correspondent aux probabilités qu'a un jeton d'aller dans le chemin adéquat (Figure 22). Les probabilités p_1 à p_5 sont calculées à l'aide de l'outil PIN, tandis que les probabilités p6 à p10 sont déduites des études déjà réalisées sur la prédiction analytique des taux de hit/miss des caches car les simulateurs de cache prendraient trop de temps. Au final, les résultats obtenus sont très satisfaisants puisque l'erreur de prédiction moyenne est de 7%, avec un maximum de 19% pour une application précise (Figure 24).

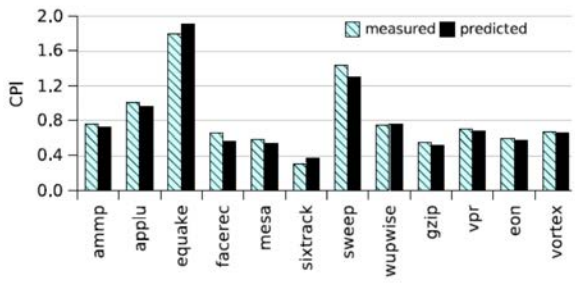


Figure 24 Précision de prédiction du CPI [41]

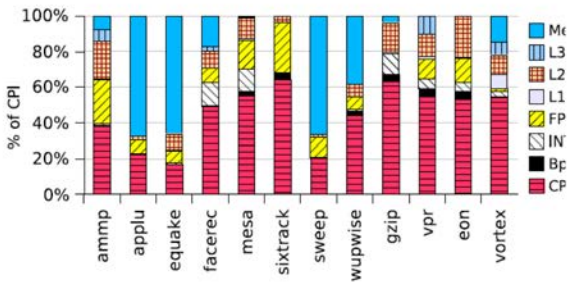


Figure 25 Répartition des différents types de CPI [41]

Ce modèle permet aussi de constater quelles sont les répartitions des différents CPI (Figure 25). On peut voir que le CPI_i est en moyenne de 40% mais varie entre 20% et plus de 60%. La part de CPI_{st} est donc loin d'être négligeable puisqu'elle est le plus souvent majoritaire.

Ce modèle de prédiction est intéressant, de part ces résultats de prédiction performants mais aussi de part l'analyse effectuée sur le nombre de cycles par instructions (CPI) induits par les différents ralentissements processeur.

Modèle paramétrique : trousse à outils de Marin et Mellor-Crummey

Marin et Mellor-Crummey ont présenté une trousse à outils permettant de mesurer et modéliser semi automatiquement les caractéristiques statiques et dynamiques d'une application [43]. L'architecture du toolkit se décompose donc en trois parties principales (Figure 26) que sont : la « Static analysis » permettant l'analyse statique des fichiers binaires, la « Dynamic analysis » permettant d'analyser le comportement d'une application binarisée préalablement instrumentée et le « post processing » qui permet de synthétiser les deux analyses pour générer le modèle.

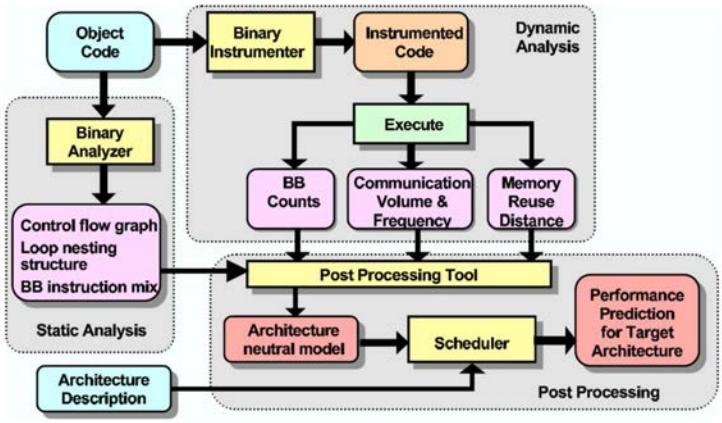


Figure 26 Procédure d'estimation du modèle de prédiction d'un software [43]

L'analyse statique

Cette analyse des fichiers binaires se décompose en trois analyses que sont :

1) la construction de CFG (Control Flow Graph) pour chaque routine,

2) la détermination des boucles imbriquées,

3) l'identification des mix d'instructions dans les blocs basiques ou les boucles et l'identification des dépendances d'ordonnancement entre les instructions (deux types de dépendances : la dépendance de registre et la dépendance de mémoire).

L'analyse dynamique

Cette analyse du toolkit va instrumenter le code binaire en le réécrivant pour pouvoir analyser son comportement lors de l'exécution. A partir de cela, l'outil va générer des histogrammes des blocs basiques exécutés, des fréquences d'exécution de ceci et du nombre d'accès des différentes mémoires.

L'outil va donc déterminer la fréquence à laquelle un graphe de contrôle de flux (CFG) est traversé pendant l'exécution. Pour cela, il faut donc instrumenter un certain nombre de routines sélectionnées (Figure 27).

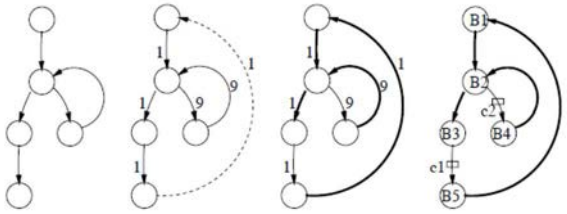


Figure 27 Exemple d'étapes de mise en place de l'évaluation de la fréquence d'exécution [43]

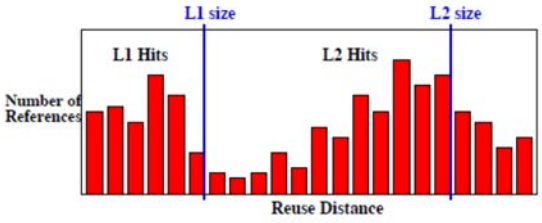


Figure 28 Nombre d'accès mémoire à chacun des niveaux hiérarchiques de la mémoire [43]

Pour caractériser les accès mémoires, le toolkit référence le nombre d'accès mémoire en fonction de leur niveau hiérarchique (Figure 28). Pour cela, un « event handler » va incrémenter une clock à chaque accès mémoire et va stocker le moment de cet évènement (logical time). Ce n'est que lors d'un deuxième accès à la même donnée que l'on pourra connaître la distance mémoire.

La création du modèle

Elle va être réalisée à partir des deux analyses décrites précédemment. Cette étape a, dans un premier temps, pour but de réaliser un modèle indépendant de l'architecture étudiée. Celui-ci prend en compte les facteurs clés qui influent sur les performances d'une application comme le nombre d'instructions exécutées, le nombre d'instructions mixées, le nombre de dépendances d'ordonnancement dans les boucles les plus exécutées et les types d'accès mémoires.

Pour le modèle de fréquence d'exécution, Marin et Mellor-Crummey ont choisi d'utiliser une fonction représentée par une combinaison linéaire de fonctions basiques. Les coefficients de cette fonction sont calculés par rapport aux données collectées lors des phases d'analyses.

Concernant les accès mémoires, le modèle doit prédire l'histogramme de la MRD, « Memory Reuse Distance » (Figure 28) pour chaque taille de problème.

Les résultats expérimentaux :

Pour démontrer la capacité de prédiction du modèle, un certain nombre de benchmarks (ASCI Sweep3D ; BT et SP de NPB 2.3 Serial ; BT, LU et SP de NPB 3.0) ont été étudiés sur la plateforme Origin 2000 qui contient seize CPUs R12000 de 300MHz et 10 GB de RAM. Chaque CPU a un cache de donnée L1 de 32KB et un TLB (Translation Lookaside Buffer) de 64 entrées ayant chacune deux pages (de registre) de 16KB. Ensuite, chaque paire de CPUs partage un cache L2 de 8MB. Le cache L1 a une taille de ligne (taille des blocs reçus) de 32 bytes, tandis que le cache L2 a une taille de ligne de 128 bytes.

La première partie des résultats porte sur le nombre d'accès ratés vers les différents accès mémoires et les TLB. Pour prédire ce nombre d'accès ratés, Marin et Mellor-Crummey ont utilisé la MRD (Memory Reuse Distance) collectée à l'aide de leur toolkit. Cette MRD est étudiée à un certain nombre de tailles différentes entre 20 et 50. A partir de cela, un modèle a été déterminé pour chaque taille différente. Mis à part quelques exceptions, l'erreur entre la prédiction du nombre de ratés mémoires et le nombre mesuré à l'aide d'« hardware performance counter » ne dépasse pas les 10%.

La seconde partie porte sur la prédiction du temps d'exécution. Pour estimer ce temps d'exécution, il faut prendre en compte celui du programme lui-même mais aussi le coût des accès aux différents niveaux mémoires. Pour cela, ils utilisent les histogrammes du nombre d'exécutions nécessaires pour un certain nombre de problèmes élémentaires et l'évaluation du nombre de cycles moyens nécessaires pour accéder aux différents niveaux mémoires (L1, L2, TLB). Au final, les prédictions correspondent très bien aux performances mesurées de quatre des applications. Pour les deux dernières, le temps d'exécution a été sous-estimé de 20% souvent à cause de conflits au niveau du cache L2, non prédit.

Modèles de prédiction et mémoires caches

La bonne utilisation des caches de CPU est un élément critique de la performance d'un système, c'est pourquoi Aleksey et al. ont proposé un outil [48] permettant de repérer les défauts de cache (« cache misses » : recherche dans un cache mémoire qui ne contient pas l'élément recherché) pour aider les programmeur.

Selon Hennessy et Patterson [23], les différents défauts de cache se décomposent en quatre catégories, les défauts de cache obligatoire (première demande de données), les défauts de cache de partage (lorsque deux CPU partagent un cache et que l'un a écrit à un endroit mémoire où l'autre pensait trouver ses données), les défauts de cache conflictuels (deux adresses de la mémoire supérieure écrivent en même temps

sur le cache) et les défauts de cache capacitif (la taille des données nécessaires excèdent la taille du cache).

Aleksey et al. ont donc développé l'outil DProf, qui accumule des traces lors de l'exécution du logiciel étudié à l'aide d'outil de monitoring, pour repérer et catégoriser les défauts de mémoire. Cet outil permet ensuite d'identifier les données qui occasionnent des défauts de cache et les affiches de quatre manières différentes : par profil de données (Data profile), par classification des défauts de mémoires (Miss Classification), par données actives (Working set), par flux de données (Data Flows).

Data profile : Répertorie le nombre de défauts de cache pour chaque profil de données. Cette vue est uniquement intéressante si c'est le type de données qui occasionne un défaut de cache.

Miss Classification : Répertorie le nombre de défauts de cache par catégories (cf. Hennessy et Patterson).

Working Set : Répertorie quel est le type de donnée le plus utilisé à chaque moment de l'exécution du programme.

Data Flow : Indique les flux de données qui vont changer de cœur de traitement. Cela permet de percevoir quelles sont les données qui vont induire des défauts de cache de partage.

Pour générer ces quatre vues, l'outil DProf enregistre et étudie deux types de données que sont les « path traces » et les « address set ». Un « path trace » enregistre tout l'historique de lecture et écriture d'un objet de son allocation à sa libération mémoire. Tandis que les « adress set », permettent d'enregistrer l'adresse et le type de tous les objets alloués en mémoire pendant l'exécution.

D'autres outils étudiant les performances du cache ont été proposés, que ce soit en profilage de code comme Gprof ou Quartz, ou que ce soit en profilage de données comme MemSpy ou Cprof. Cependant, l'outil Dprof fournit de nombreuses données dans les deux domaines et permet d'avoir une réelle vue d'ensemble sur les performances du cache du logiciel étudié.

D'autres recherches ont été menées spécifiquement sur la mémoire cache comme par exemple Hammond et al. dans **[44]** ou encore Li et al. qui avaient analysé le WCET (Worst Case Execution Time) à l'aide de l'analyse du path en supposant que chaque instruction prend le même temps à être exécuter. Pour déterminer le temps d'exécution global, il suffit de déterminer le temps d'exécution de chaque type de tâches. Chaque type de tâche est additionné après avoir été multiplié par son coefficient de redondance. Et pour améliorer cette modélisation, ils ont proposé une modélisation de la mémoire cache d'instruction dans [45], beaucoup plus complète

qui permet de comprendre l'influence proéminente de celle-ci dans la performance d'un système.

1.2.3.2. Les méthodes orientées parallélisme

Méthodologie PAMELA

Van Gemund [46] a proposé une méthode d'évaluation de la performance de systèmes parallèles basée sur le langage de modélisation PAMELA (PerformAnce ModEling LAnguage)[47]. Cette modélisation permet d'unifier les spécifications techniques d'un programme (Lprog) et d'une machine (Lmach) et de réaliser une « serialization analysis » du modèle (T(L)), comme on peut le voir sur la Figure 29.

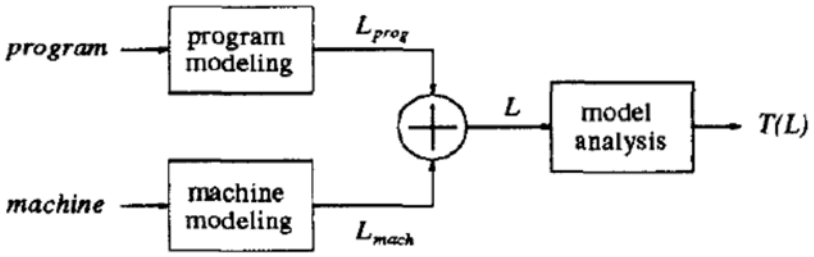


Figure 29 Processus de modélisation de performance PAMELA [46]

Le langage PAMELA permet, dans un premier temps, d'augmenter un algorithme écrit en langage C en définissant les parties séquentielles (seq), les parties parallèles (par), les exponentiations (for, for all), les itérations (while), et les conditions (if, else) mais aussi en ajoutant des conditions de synchronisations à l'aide des opérateurs « wait » et « signal ». Cette première étape va aussi permettre de décomposer le code en fonctions hardwares basiques (load, flop, etc.). Cette étape permet d'obtenir le L_{prog} représenté sur la Figure 29.

La seconde étape permet de déterminer les temps des fonctions basiques : accès mémoires et d'exécution en floating point ($_m$ et $_f$) pour prendre en compte les spécificités de l'architecture (L_{match}). Enfin, en faisant converger les deux premières étapes, on obtient le programme modélisé L.

Le langage PAMELA est une méthodologie très orientée matériel. Si on considère l'exemple d'un système client-serveur avec P clients qui passent N demandes de services à un serveur, que chaque demande prend un temps $_l$ et chaque exécution prend un temps $_s$ sur le server s alors la décomposition en PAMELA est détaillée Figure 30.

Grâce à cette méthodologie, il est relativement simple de modéliser un programme. Par contre, il reste un certain nombre de paramètres qui ne sont pas pris en compte comme le branching et les boucles imbriquées. Pour solutionner ce problème, l'utilisateur va intervenir au moment de la compilation ou directement sur le programme à l'aide d'annotation.

Pour ce qui est de la modélisation de l'architecture, le niveau de détail est beaucoup plus variable que celle du programme. En PAMELA, une architecture est considérée comme une collection de CPUs connectés à un sous-système de mémoire partagée comprenant des switches et de la mémoire respectivement associés aux modèles load et store.

En considérant l'exemple de M modules mémoires accessibles par un réseau d'interconnections composés de S switches, un « load » de la mémoire **a** est modélisé sur la Figure 31.

```
par (p = 1...P)
    seq (i = 1...N) {
        delay(τ_l);
        use(s, τ_s) }
```

Figure 30 Modélisation d'un programme [46]

$$\mathbf{use}(m_{a'}, \tau_m)$$

Figure 31 Modélisation d'un chargement mémoire [46]

Pour analyser le programme modélisé, défini par le langage PAMELA, Van Gemund [46] a initié un certain nombre de règles d'analyses des séquences sérialisées. Ces règles permettent d'analyser et de simplifier au maximum les codes modélisés pour transformer le code L en T(L) (Figure 29).

L'outil PACE

L'outil PACE [48] a été développé pour permettre de prédire la performance d'un système parallèle en fonction du moment du développement du logiciel. En effet, si l'on réfléchit à la performance d'un algorithme au moment de son design, ce ne sera pas la même chose qu'au moment de son implémentation ou de son exécution.

Selon cette étude, l'évaluation de la performance doit intégrer un certain nombre de propriétés que sont le cycle de vie ouvert, abstraction, hiérarchie et modularité.

Pour cela, l'outil se base sur une approche définie par un Framework qui contient quatre couches différentes :

La couche application est le point d'entrée de l'évaluation de la performance dans le Framework. La couche Sub-Task décrit les éléments séquentiels de l'application qui vont être parallélisés. Les « Parallel Template Object » permettent de décrire la parallélisation qui va être effectuée sur les sub-Tasks. La couche hardware décrit les

spécifications de l'architecture utilisée : les paramètres (taille de cache, nombre de processeur, etc.), les résultats de micro-benchmarks, des modèles statistiques, analytiques et heuristiques.

Ce Framework permet d'obtenir un modèle de performance qui contient un certain nombre d'objets contenus dans chacune des couches définies (Figure 32).

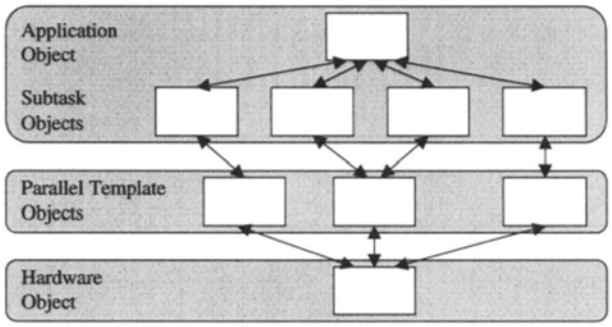


Figure 32 Diagramme des couches du Framework (HLFD hierarchy) [48]

Le système PACE est donc un modèle basé sur ce Framework, qui se décompose en six grandes phases représentées sur la Figure 33 :

Object editor : Accompagné d'« Object Library », il permet de définir l'application à l'aide d'objets de performance.

Source Code Analysis : Le code source est analysé et transcrit en procédures CHIP^{3}S (CHaracterization Instrumentation for Performance Prediction of Parallel Systems) qui peuvent être ensuite incorporées à l'ensemble des objets de performance à l'aide de l' « object editor ».

Compiler : Il a pour responsabilité de transcrire les scripts de performance obtenus par l'analyse du code source en Langage C. Celui-ci est ensuite mis en lien avec l' « evaluation Engine », ce qui permettra de définir les prédictions de performance avec le modèle Hardware.

Hardware Configuration : Il permet de caractériser l'environnement de calcul et se définit par les composants hardware (Hiérarchie mémoire, inputs/outputs, communications, …) et les informations de configuration. Pour cela, un HMCL (Hardware Modeling and Configuration Language) est utilisé.

Evaluation Engine : Les informations obtenues grâce au modèle de définition de l'application sont passées dans le modèle hardware pour obtenir une prédiction de performance. Les sorties peuvent être soit une estimation du temps d'exécution complet de l'application, soit des informations sur le comportement attendu de l'application.

Performance Analysis : Le modèle de performance obtenu est exécuté dans deux cas d'analyses différentes. L'analyse « Off-Line » permet à l'utilisateur d'interagir et de lui fournir un aperçu des performances attendues, tandis que l'analyse « On-the-fly » permet la prise de décision dynamique à l'exécution, pour par exemple, déterminer quel code va être exécuté et sur quelles ressources disponibles du système.

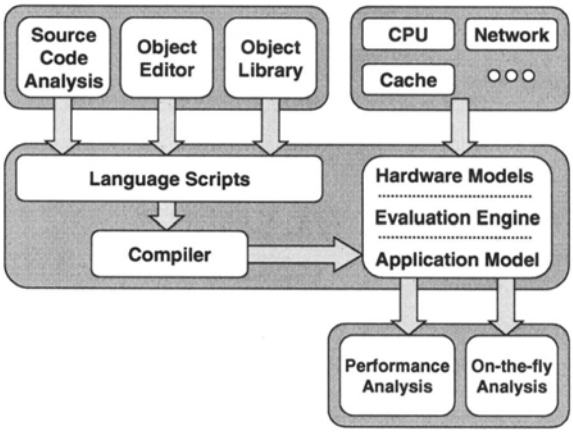


Figure 33 Schéma du système PACE [48]

Nudd et Al. ont décrit précisément chacune des parties de leur outil pour montrer que leur système, très complet, donne des résultats très intéressants dans l'étude de la performance d'un système de calcul parallèle. Par contre, pour fonctionner, l'outil doit analyser précisément le code source de l'utilisateur. Deux modèles intéressants ressortent de cette étude, les modèles software et hardware qui permettent de construire des objets softwares et des objets hardwares. Ces objets permettent de caractériser l'application étudiée et l'architecture sur laquelle elle va être lancée.

Objet Software : Ils sont définis à l'aide de l'ACT (Application Characterization Tool). Pour caractériser automatiquement un code, le SUIF est utilisé (Figure 34).

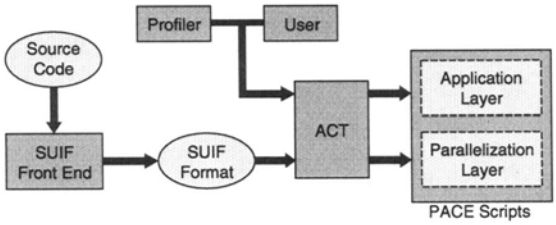


Figure 34 Processus de création d'un modèle avec ACT [48]

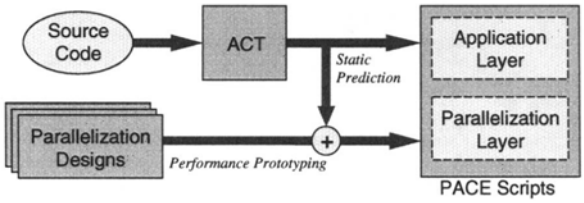


Figure 35 Utilisation d'ACT pour l'estimation de performance statique et le prototypage [48]

Objet Hardware : Les modèles hardwares peuvent prendre plusieurs formes. Cela peut se traduire par des micro-Benchmarks ou encore par des modèles analytiques

plus complexes. Il y a d'ailleurs bien souvent plusieurs modèles pour caractériser un système en particulier.

Chaque objet hardware est subdivisé en sous-modèle hardware (exemple sur la Figure 36).

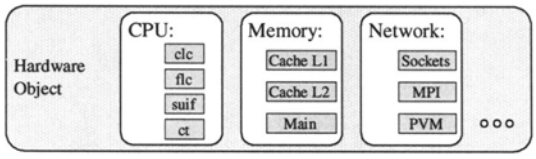


Figure 36 Exemple d'objet hardware [48]

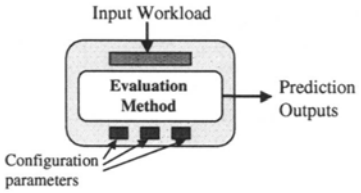


Figure 37 Composition d'un composant du modèle hardware [48]

Chaque modèle de composants hardwares est ensuite décrit à l'aide d'une méthode d'évaluation et de paramètres de configuration (Figure 37). Ceci permet d'établir une prédiction à partir du travail à réaliser (input workload) caractérisée avec les objets softwares.

Applications du modèle PACE

Jarvis et al. [49] se sont posés la question du rôle de la prédiction de performance dans le support de la répartition du travail sur des systèmes de calcul parallèle : ce que les données de performance permettent de décrire (temps d'exécution, capacités de calcul, …), comment ces données sont obtenues (benchmarks, modèles analytiques, …), comment elles sont classifiées (modèles paramétriques, scénarios d'extrapolations, …) et comment elles sont utilisées (répartition des ressources, répartition des tâches). L'exemple de l'outil PACE décrit dans [48] et retravaillé dans [49] par Jarvis et al. (Figure 38) permet de comprendre les enjeux de l'étude de la performance d'un système de calcul parallèle. Cet outil est compliqué à mettre en place mais il permet d'obtenir des niveaux de prédiction très intéressant (5-12% d'erreur). Il a d'ailleurs été utilisé à de nombreuses reprises pour des applications de finance, analyse de performance temps réel ou encore de l'application Sweet3D[50]. Jarvis et al. ont étudié la gestion des ressources à un même niveau à l'aide d'une combinaison d'un co-ordonnanceur (le TITAN [51]) et un ordonnanceur standard (le Condor [52]).

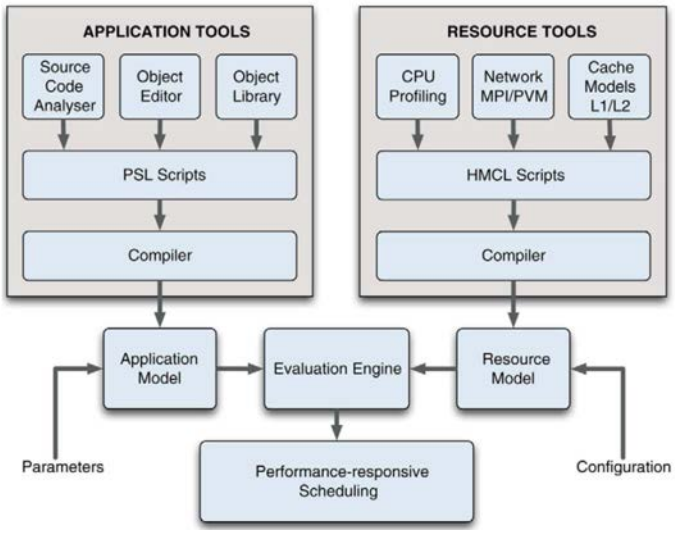


Figure 38 Représentation du PACE tool par Jarvis et Al. [49]

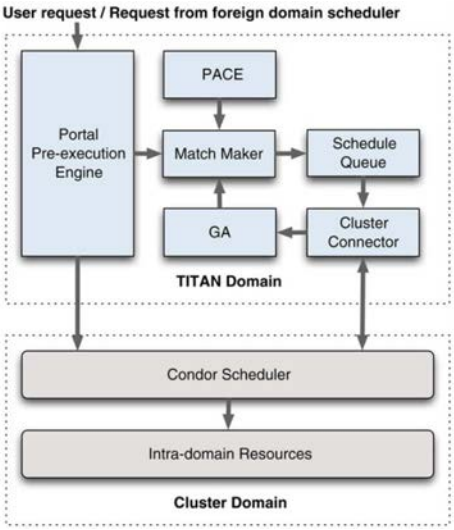


Figure 39 Interaction entre le Condor et le Titan intégrant le modèle PACE [49]

Autres modèles

Ipek et al. ont présenté une solution de modélisation et de prédiction de performance en se focalisant sur le benchmark SMG2000 [53]. Le principe de l'approche est d'utiliser un modèle d'apprentissage sur l'espace de paramètres définis par les entrées du programme et de collecter une grande quantité de données sur une architecture précise. Grâce à ces données, le modèle va apprendre quel est l'impact des différents paramètres sur la performance pour ajuster sa prédiction. Wabding et Arding ont présenté un outil intéressant nommé PAPS [54] qui permet aussi de décrire des systèmes parallèles complexes.

1.3. Positionnement de notre étude et problématique

Comme nous venons de le présenter, de nombreuses recherches ont été menées dans le domaine de la prédiction de performance et de nombreux livres ont été écrits sur le sujet. Comme par exemple « The Art of Computer Systems Performance Analysis » par Raj Jain [55] qui présentent les bases et les erreurs à ne pas commettre dans l'évaluation de la performance ou encore « Computer Architecture : A Quantitative Approach » par Hennessy et Patterson [56] qui explique quelles sont les problématiques de l'amélioration de la performance d'une architecture hardware et qui permet de mieux comprendre les paramètres influençant cette performance.

Il n'est pas question ici de rentrer en concurrence avec toutes les approches ou modèles développés mais de placer l'étude de manière originale par rapport à ce qui existe. Un certain nombre de techniques, utilisées dans la construction des différents modèles présentés, seront d'ailleurs utiles dans la construction du paradigme de notre étude. Mais ces différentes techniques ne pourront en aucun cas être utilisées dans la description de l'algorithme par un utilisateur de notre approche.

La problématique de notre étude porte sur l'analyse de la performance d'un système de calcul, très en amont dans la phase de développement d'une solution algorithmique.

En effet, l'étude d'un algorithme très peu abouti, paraît complexe à entreprendre avec les outils existants car il est nécessaire de connaître très en détails, l'algorithme que l'on désire étudier pour les utiliser. Comme cela a été expliqué dans l'introduction générale, c'est la performance des algorithmes de traitement d'images qui nous a amené à travailler sur la performance algorithmique en générale. De ce fait, c'est sur ces algorithmes en particulier que va donc porter notre étude.

L'idée originale de l'étude repose donc justement sur le fait qu'elle ne porte que sur les algorithmes de traitement d'images, contrairement à la grande majorité des modèles qui permettent d'étudier tout type d'algorithmes. Mais cette restriction permet d'avoir une approche différente des outils et méthodes existants et de pouvoir apporter une aide à un utilisateur dans une phase beaucoup plus amont dans le développement d'une solution.

En effet, de nombreux outils d'évaluation de performance nécessitent un accès au code source de l'algorithme, de réaliser un certain nombre de tests ou nécessitent parfois même que l'utilisateur ait des connaissances en architecture. Alors que le but de notre approche est de permettre à l'utilisateur de décrire son algorithme de traitement d'images, simplement et sans entrer trop profondément dans les détails.

Le concept présenté ici est de caractériser un algorithme de traitement d'images à l'aide d'un jeu de blocs élémentaires, qui pourrait dans l'absolu, caractériser l'ensemble des algorithmes de traitement d'images. Cette méthode permettra à un utilisateur de décrire son algorithme, le plus en amont possible, que ce soit dans un projet industriel ou dans un projet de recherche en laboratoire.

Il est question aussi de faire en sorte que le paradigme présenté, puisse être utilisé par un outil, qui permette à une personne qui désire développer une application de traitement d'images, de simuler son application sur une architecture ou de trouver l'architecture qui lui convienne le mieux pour son application. C'est l'objet du chapitre suivant qui présente le paradigme original mis en œuvre et une esquisse de ce que pourrait être l'outil d'aide au développement d'application de traitement d'images.

Chapitre 2 Présentation de notre paradigme

L'étude se concentre sur la modélisation d'algorithmes de traitement d'images. Ces algorithmes étant très consommateurs en puissance de calcul, on comprend l'impact important d'une modélisation fine du temps de calcul. De plus l'orientation vers le traitement d'images donne le cadre et les contraintes du travail engagé.

Par rapport à cette thématique globale de caractérisation de la performance d'un algorithme de traitement d'images sur une architecture hardware donnée, il a été présenté trois grands types d'architectures qui permettent de traiter ce type d'algorithme : le CPU, le GPU et le FPGA. L'étude va se concentrer sur la caractérisation du CPU. En effet, comme on a pu le voir dans le Chapitre 1, cette architecture est plus simple à appréhender et elle comporte des variables de performance moins aléatoires. De plus, pour le traitement d'images cette architecture est assez majoritaire, en particulier dans les architectures embarquées qui prennent de l'importance d'années en années. Si l'approche envisagée ne fonctionne pas sur ce type d'architecture, il est très difficile d'imaginer qu'elle fonctionnera sur GPU et FPGA. Il est ressorti de l'étude de ces architectures que les choix entrepris dans l'implémentation d'algorithme de traitement d'images a un impact non négligeable sur la performance. La parallélisation sur GPU, qui est l'essence même de cette architecture, n'est pas forcément évidente à prendre en main et le FPGA a une flexibilité qui est le fondement même de sa puissance mais c'est cette flexibilité qui le rend complexe à modéliser.

Bien sûr, à travers la démarche entreprise, nous verrons qu'il y a de nombreuses manières d'intégrer ces différentes problématiques, mais la démarche entreprise, a été de d'abord démontrer que l'approche envisagée fonctionne sur des architectures CPU avant d'envisager de porter la démarche sur d'autres architectures.

Hypothèse de travail N°1 : Etude du CPU

Les CPU ont déjà de nombreux paramètres à intégrer, comme la fréquence d'horloge, la mémoire cache (nombre, taille, vitesse d'accès) ou le nombre de cœur. Il faudra aussi porter attention au type de CPU utilisé (CISC ou RISC).

Pour ce qui est de la parallélisation des traitements, l'étude porte principalement sur de la parallélisation SIMD, qui reste très appropriée pour du traitement d'images et qui peut souvent se résumer à un même traitement sur une quantité importante de

données. Le MIMD étant plus complexe à prendre en compte et moins caractéristique pour le traitement d'images, cette première étude ne le prend donc pas en compte.

Hypothèse de travail N°2 : Prise en compte de la parallélisation SIMD

Les différentes hypothèses de travail proposées permettent de créer une délimitation du travail sur lequel nous allons nous concentrer. Et cela va permettre d'établir un premier paradigme qui prend en compte une partie des architectures hardwares présentées.

2.1. Approche envisagée

2.1.1. Approche générale

Le but de cette étude est de prédire les performances de n'importe quel algorithme de traitement d'images sur une architecture donnée, soit pour simuler cette performance, soit pour proposer des architectures crédibles dans le but de la résolution de cet algorithme. La modélisation présentée est défini comme le paradigme scientifique de notre étude. Cette étude repose toujours sur l'idée de trouver un modèle parcimonieux, c'est-à-dire d'avoir un modèle le plus simple possible, dans la limite de résultats de prédictions intéressants. En effet, le but est d'avoir le moins de paramètres libres possible. Les différents types de paramètres étant, les paramètres hardwares connus par les spécifications techniques des fabricants de composants électroniques, les paramètres définis pour chaque architecture à l'aide de benchmarks et les paramètres libres modélisés à l'aide de tests.

Le paradigme scientifique envisagé en réponse à la caractérisation de la performance d'un algorithme de traitement d'images, repose sur la décomposition de cet algorithme en briques de traitement d'images élémentaires. Ces briques élémentaires seront définies en amont de manière stochastique à l'aide de benchmark, pour déterminer le comportement de ces briques en fonction des caractéristiques d'une architecture hardware. Cela va constituer un ensemble de modèles de briques élémentaires que l'on nommera MBE.

Ensuite, le but est de pouvoir construire tout algorithme de traitement d'images à l'aide de ces MBE. Chaque algorithme modélisé sera donc composé de briques stochastiques arrangées de manière déterministe.

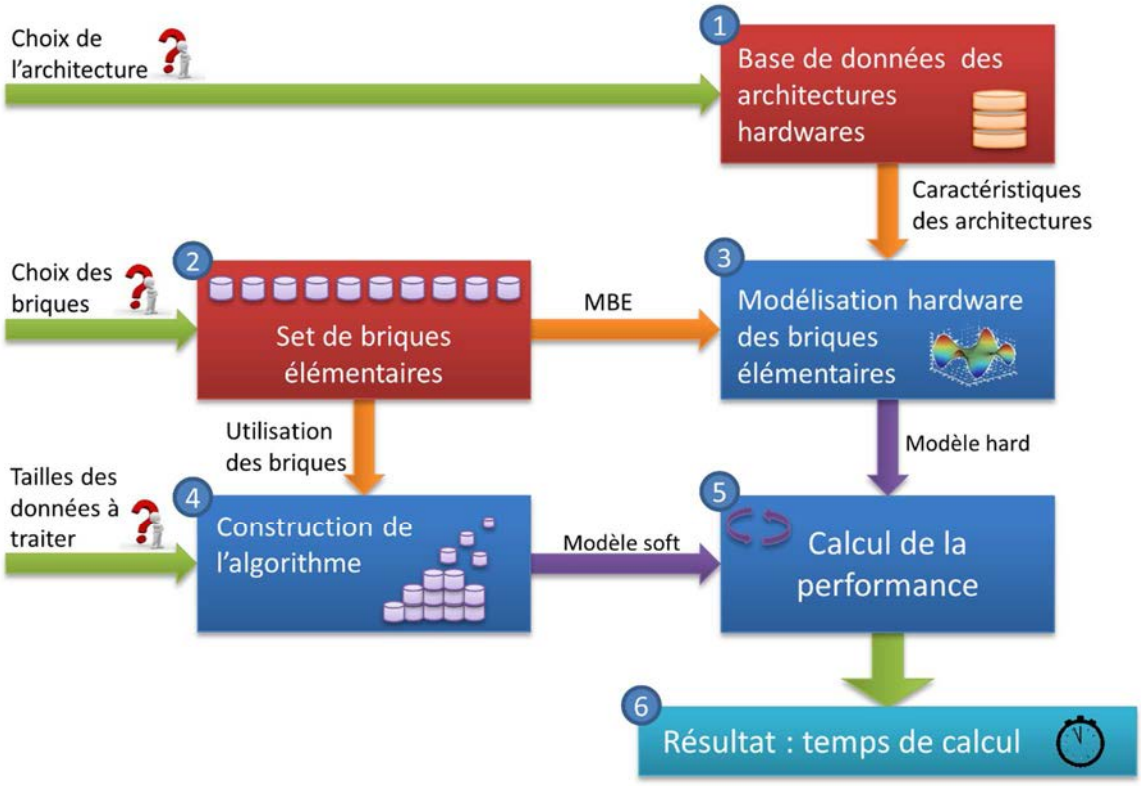


Figure 40 Paradigme d'évaluation de la performance d'un algorithme de traitement d'images

Le paradigme scientifique simplifié, qui est présenté en Figure 40, se décompose en six blocs :

1) La base de données des architectures hardwares : l'idée est d'avoir le choix parmi des architectures ayant un certain nombre de caractéristiques (type, fréquence, nombre de cœur etc.) qui sont nécessaire au paradigme exposé. Il est aussi possible d'alimenter cette base de données en fournissant les caractéristiques nécessaires, de l'architecture que l'on désire étudier. Dans un premier temps, les architectures étudiées se restreignent aux CPU de type GPP et les caractéristiques qui influent sur la performance ont été déterminées à l'aide des résultats observés (voir ci-après).

2) Le set de briques élémentaires : il va permettre la construction de l'algorithme de traitement d'images étudié. Cette partie de la modélisation permet de choisir les briques élémentaires qui sont indispensables à la construction de l'algorithme étudié et de faire ressortir les MBE pour les faire converger vers les caractéristiques des architectures. Ce set de briques élémentaires, permettant de définir l'ensemble des algorithmes de traitement d'images (dans l'absolu), est la base du paradigme d'évaluation de la performance mis en place.

3) La modélisation hardware des briques élémentaires : cette partie permet d'intégrer les caractéristiques de l'architecture hardware aux MBE, pour créer le « modèle

hard » qui permettra de déterminer la performance. En effet, les MBE contiennent des paramètres dépendants directement des caractéristiques des architectures hardwares et des paramètres libres qui sont, dans leur cas, dépendants de benchmarks réalisés sur ces architectures. Pratiquement tous les MBE sont différents.

4) Modélisation software : l'idée est ensuite de construire l'algorithme avec les briques élémentaires choisies et d'assigner à chacune des briques de l'algorithme, la taille des données à traiter qui lui correspond. En effet, chaque étape de l'algorithme de traitement d'images étudié, peut avoir une taille d'image à traiter différente. Cette étape va aboutir au « modèle soft ».

5) Calcul de la performance : la cinquième étape de notre paradigme consiste, à partir des modèles hardwares et softwares, à calculer la performance du système de calcul étudié (Figure 1). A cette étape, les MBE ayant intégré la performance du hardware, ils ne dépendent plus que de la taille de données à traiter. Chaque brique de l'algorithme est donc évaluée et la somme de ces briques donne la performance globale de l'algorithme.

6) Résultat : pour finir, ce calcul de la performance (5) aboutit sur le temps de résolution de l'algorithme. Pour l'instant, cette performance est seulement caractérisée par le temps de calcul mais d'autres paramètres pourront être modélisés par la suite comme par exemple, la consommation électrique, la température dissipée ou même l'encombrement.

2.1.2. Base de données des architectures hardwares

La base de données des architectures hardwares permet d'obtenir toutes les caractéristiques techniques qui influent sur les performances. Il est ressorti de l'étude que les spécifications influençant les performances de ce groupe d'architectures sont : le type d'architecture CPU (CISC ou RISC), le nombre de processeurs parallèles, la fréquence d'horloge, et la performance mémoire, elle-même caractérisée par la taille, le nombre et la vitesse d'accès des mémoires caches (L1, L2, …). Les résultats expérimentaux du chapitre 3 démontrent ce postulat sur un certain nombre d'architectures et d'algorithmes de traitement d'images élémentaires. Pour un certain nombre de CPU (ou GPP pour notre étude), les performances mémoires sont souvent les caractéristiques les moins évidentes à trouver. En effet, les constructeurs ne détaillent pas forcément les caractéristiques des caches mémoires intégrés à leurs CPU.

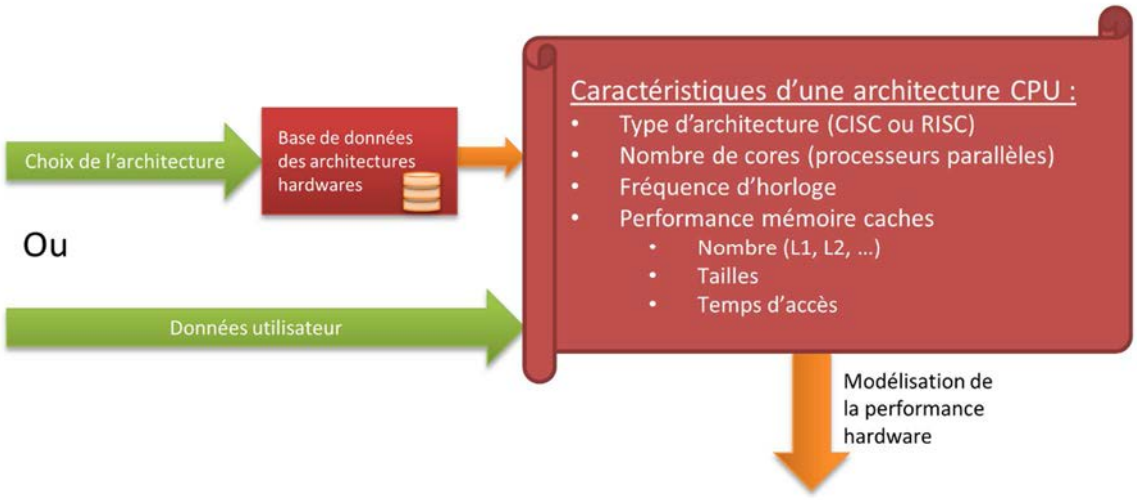


Figure 41 Détails de la première partie du paradigme présenté en Figure 40

La performance des mémoires caches n'est pas nécessairement accessible auprès des constructeurs. On peut par contre émettre l'hypothèse qu'un modèle pourrait être défini en fonction des caractéristiques des mémoires caches. Pour ce qui est du nombre et des tailles de caches mémoires, ces informations peuvent s'obtenir facilement, par contre les temps d'accès et vitesses de transfert entre les différents niveaux de caches sont plus complexes à déterminer. Il est possible d'évaluer la performance mémoire à l'aide de tests. Mais compte tenu des objectifs de l'étude, il est difficile d'imaginer que cette méthode d'évaluation de la performance mémoire soit mise en place dans notre paradigme. Par contre, il est possible qu'une extrapolation de la performance mémoire soit réalisée à partir de tests entrepris sur le panel d'architectures étudiées.

Nous verrons dans le chapitre 3, qu'un logiciel a été développé pour évaluer la performance mémoire sur chacune des architectures hardware testée. A terme, l'objectif est de déterminer le modèle qui régit la performance mémoire en fonction des paramètres de la mémoire cache (Pour ce qui est de la RAM, le modèle pourra éventuellement prendre en compte la taille de RAM maximum autorisée par le CPU ou donnée par l'utilisateur).

Pour obtenir le « modèle hardware » (partie (3) de la Figure 40) d'une MBE pour une architecture précise, les caractéristiques techniques du CPU (choisi ou décrit), sont intégrées aux modèles des briques élémentaires sélectionnés dans le panel (Figure 41).

Ensuite, la modélisation de l'algorithme de traitement d'images (nommée « modèle soft » dans la Figure 40), croisé avec le « modèle hard », permet d'obtenir la performance de l'algorithme en question ; c'est-à-dire son temps de calcul sur l'architecture choisie.

2.1.3. Ensemble de briques élémentaires pour la construction de l'algorithme

L'approche de prédiction de performance présentée dans cette étude, repose en grande partie sur le set de briques élémentaires qui permet de définir le domaine étudié. Pour notre part, nous présentons cette approche appliquée au domaine du traitement d'images car c'est ce domaine qui nous a poussés à mener cette étude. De plus, il rentre parfaitement dans le schéma de décomposition en blocs élémentaires. Cette méthode de caractérisation de la performance d'algorithmes pourrait aussi s'appliquer à d'autres domaines. Il faudrait pour cela, qu'ils puissent être caractérisés et délimités par ce set de briques élémentaires ou que le pourcentage (démontré) de couverture du domaine par ce set soit acceptable.

Une brique élémentaire est définie comme étant un algorithme de traitement d'images qui ne peut pas être défini à l'aide d'autres algorithmes de traitement d'images plus simples. Ces algorithmes élémentaires sont justement intégrés dans d'autres algorithmes plus complexes. On peut dire que ces briques élémentaires doivent avoir une certaine atomicité.

Le but est de pouvoir définir un set de blocs élémentaires qui permet de définir l'ensemble des algorithmes de traitement d'images. La démonstration qu'un set de blocs élémentaires définisse l'ensemble des algorithmes de traitement d'images est complexe à appréhender. On peut donc imaginer, un ensemble d'algorithmes entrant dans un cadre plus restreint. Pour cela, il est possible d'avoir une approche empirique sur la question au lieu de démontrer que le set d'algorithmes représente l'ensemble des algorithmes de traitements d'images. Dans un premier temps, pour notre étude, nous avons donc réalisé une étude avec des experts du domaine, pour converger vers un set qui se rapproche le plus possible de ce que l'on nomme algorithme de traitement d'images.

Ce travail heuristique a permis de déterminer un certain nombre de briques élémentaires qui sont présentés dans la Figure 42. Ce set de briques élémentaires (Upsampling, Decimation, Thresholding, …) permet de caractériser un grand nombre d'algorithmes de traitement d'images, mais il ne caractérise bien sûr pas l'ensemble des algorithmes de traitement d'images. Ce set de briques élémentaires est le premier exemple qui devra être affiné.

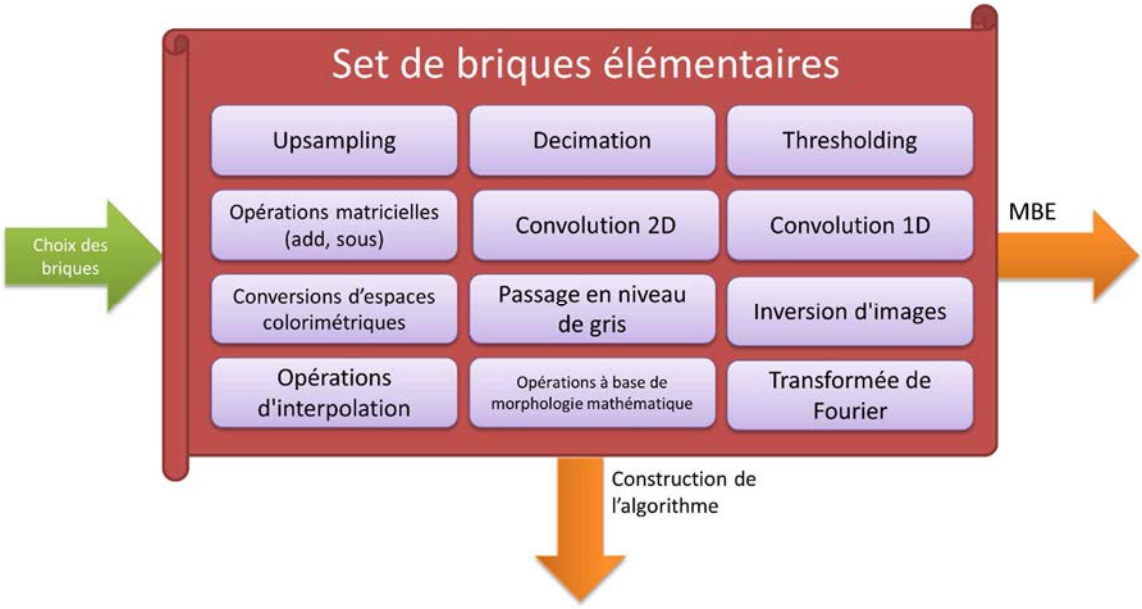


Figure 42 Ensemble non-exhaustif des briques élémentaires (Partie 2 de la Figure 40)

Chaque brique élémentaire répond ensuite à un MBE (Modèle de Brique Elémentaire) qui lui est propre. En effet, la performance d'une brique élémentaire ne réagit pas de la même façon face aux caractéristiques d'une architecture hardware. Par contre, comme nous le verrons dans le chapitre 3, les briques étudiées ont une performance linéaire par rapport à la taille de données traitées. Chaque MBE est caractérisé à l'aide d'une étude, puis il est intégré au paradigme de prédiction de performance.

2.1.4. Caractérisation d'un MBE (Modèle de Brique Elémentaire)

Un modèle de brique élémentaire est défini pour chaque algorithme de traitement d'images choisi comme étant élémentaire. Pour cela, une étude de l'influence de chacun des paramètres hardwares doit être réalisée pour chaque brique élémentaire.

L'idée est d'avoir le moins de paramètres libres possibles. Nous verrons que sur les architectures testées, le paramètre de la fréquence d'horloge influence la performance de façon totalement proportionnelle contrairement au type d'architecture, aux paramètres mémoires et au nombre de cores de l'architecture. En effet, ces trois derniers paramètres influencent la performance de manière différente selon la brique élémentaire de traitement d'images étudiée. Ce postulat sera confirmé dans le chapitre 3.

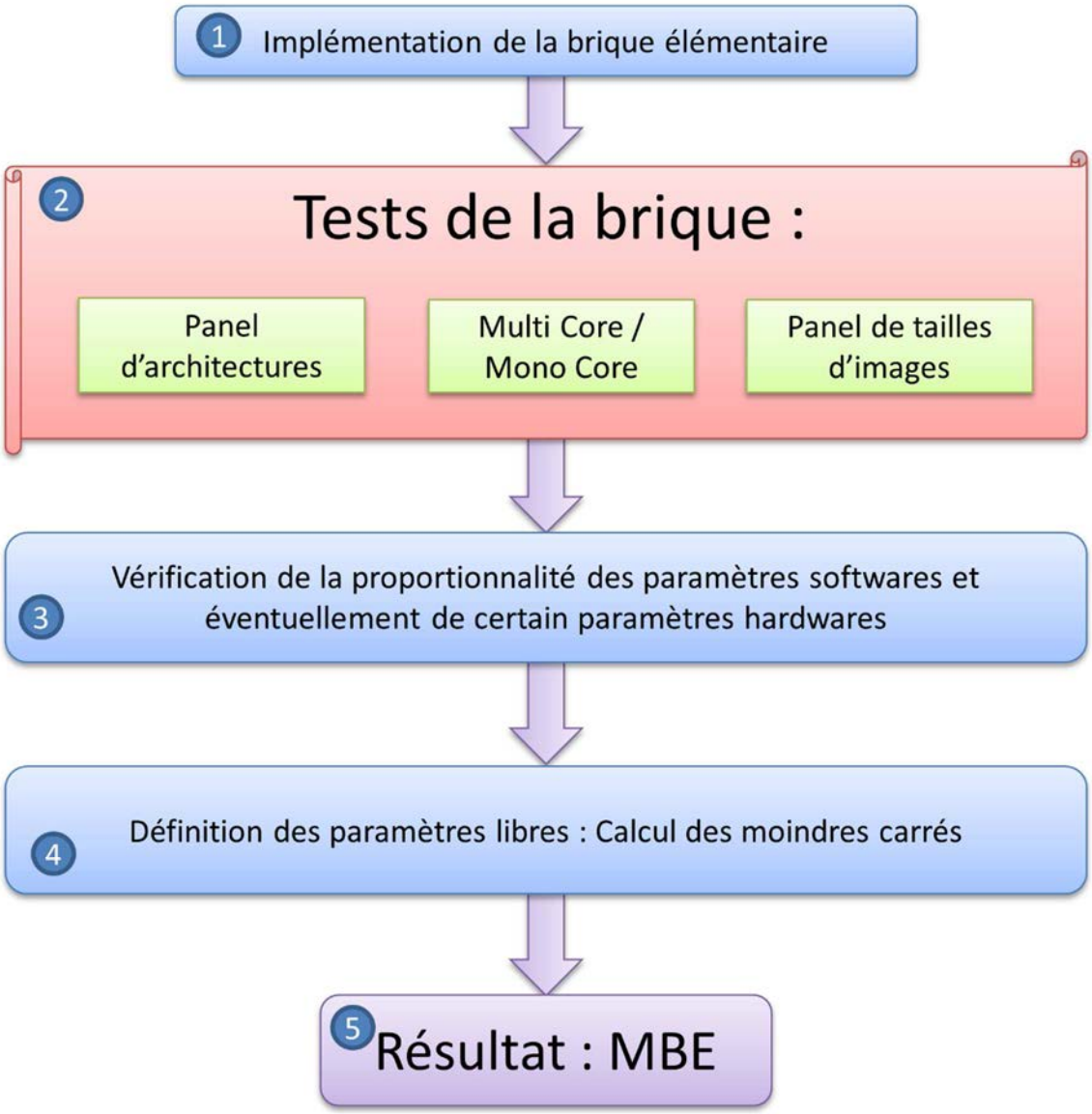


Figure 43 Définition d'un Modèle de Brique Elementaire (MBE)

Chaque brique élémentaire fait l'objet d'une étude pour être modélisée. Cette modélisation est obtenue par le calcul des paramètres libres vu précédemment. Les paramètres libres sont déterminés à l'aide de benchmark réalisé sur un panel d'architectures. Ce panel d'architectures est défini de manière heuristique, pour faire ressortir l'influence de chacun des paramètres hardwares. Le panel en question intègre des CPU de différents types (CISC et RISC), ayant des fréquences d'horloges représentant les processeurs existants.

Ces différents benchmarks permettent également de confirmer l'influence directe (strictement proportionnelle) de la fréquence d'horloge mais aussi des paramètres softwares. La méthode d'évaluation d'un MBE décrite Figure 43 se décompose en cinq étapes :

1) **Implémentation de la brique élémentaire** sur le type d'architectures traitées (seul le CPU sera traité dans un premier temps). Cette partie permet d'adapter l'algorithme étudié sur la typologie d'architectures que l'on souhaite tester (X86 ou ARM, mono core ou multi core).
2) **Tests de la brique élémentaire** sur différentes architectures, en mono/Multi avec différentes tailles d'images. L'objectif est d'obtenir un grand nombre de résultats sur un panel d'architectures représentant au maximum l'ensemble des caractéristiques de cette typologie d'architecture.
3) **Vérification de la proportionnalité** des paramètres softwares et de la fréquence d'horloge. Cette partie permet de vérifier le postulat de l'existence d'une proportionnalité entre la performance et des paramètres softwares. Ces paramètres softwares se limitent souvent à la taille de l'image traitée mais ils peuvent aussi contenir la taille d'un filtre pour une convolution 2D. Enfin, cette étape permet de linéariser le coefficient de proportionnalité de l'algorithme étudié en fonction de la fréquence d'horloge pour chaque architecture du panel.
4) **Définition des paramètres libres** en appliquant la méthode des moindres carrés. En effet, les caractéristiques restantes vont différemment influencer la performance du système selon la brique élémentaire.
5) **Conclusion** : Obtention du modèle de brique élémentaire grâce aux calculs des différents paramètres.

La définition de ces MBE est la base de la modélisation globale mise en place. C'est à dire que le pourcentage d'erreur de prédiction de chacun des MBE, aura une incidence directe sur l'erreur de prédiction globale du problème. Pour ne pas obtenir une erreur de prédiction trop importante, il est nécessaire que l'erreur de chacun des MBE soit, dans un premier temps, au moins inférieure à 10%.

2.1.5. Mise en perspective

Nous avons décomposé les types de modèles en deux groupes : les modèles stochastiques qui se basent sur des calculs statistiques pour déterminer la performance d'un système de calcul et les modèles déterministes qui déterminent la performance d'un système de calculs en fonction d'un certain nombre de paramètres. Les modèles que nous avons étudiés dans le chapitre 1 sont rarement totalement déterministes.

Un grand nombre de méthodes caractérise la performance d'un algorithme en utilisant directement le code ou en se basant sur un algorithme très détaillé. Par exemple, les méthodes stochastiques utilisent souvent des diagrammes UML (use case diagramme ou diagramme de séquence) pour décrire un algorithme. La réalisation de ces différents diagrammes peut être assez laborieuse et trop contraignante pour l'utilisateur. De plus, cela ne permet pas d'intervenir très en amont dans la phase de réalisation d'un projet en traitement d'images. En effet, notre problématique est de pouvoir modéliser notre système de calcul le plus en amont possible, dans les différentes phases de développement d'une solution. Les méthodes stochastiques d'analyse des performances présentées, manquent souvent de structuration et demande bien souvent d'utiliser des benchmark pour chaque architecture et/ou algorithme étudié. Cela donne des contraintes très importantes pour l'utilisateur en termes de temps mais aussi en termes de prix (obtention des architectures testées).

Les méthodes sont rarement totalement déterministes mais, en général, plus elles le sont, moins elles sont performantes. Bien sûr, tout dépend du degré de description de l'algorithme et des restrictions de domaines. En effet, certaines méthodes décortiquent le code de l'algorithme à des niveaux très bas mais cela ne correspond pas aux attentes que l'on a vis-à-vis de notre étude car nous avons la préoccupation d'intervenir très en amont dans le développement d'une solution. De plus, cela ne correspond pas aux compétences d'un spécialiste du traitement d'images.

Par rapport aux différents modèles et méthodes existants, notre paradigme se place de manière déterministe tout en ayant des briques stochastiques. L'approche peut être considérée comme hybride. En effet, la construction des MBE se fait plutôt de manière stochastique avec des benchmarks. Mais une fois que les MBE sont définis, la prédiction de performance d'un algorithme utilisant les briques définies est réalisée de manière totalement déterministe. Et l'avantage de notre étude vient du fait que l'algorithme étudié n'aura pas besoin d'être très détaillé car la description de celui-ci se fait de manière très macro.

2.2. Représentation et implémentation

2.2.1. Contexte et objectifs

Un outil permettant d'exploiter le paradigme présenté précédemment a été développé. L'idée est de pouvoir construire son algorithme à l'aide d'une interface simple et intuitive. Pour suivre le paradigme présenté, l'utilisateur devra décomposer son algorithme en briques élémentaires de traitement d'images. Ces briques sont représentées par des blocs ayant un certain nombre de caractéristiques à définir par l'utilisateur.

L'outil ne s'adresse pas à des personnes spécialistes des architectures hardwares, mais plutôt à des experts du traitement d'images qui désirent tester les performances de leurs algorithmes sur plusieurs architectures hardwares. L'idée, est de pouvoir faire une évaluation très en amont dans les différentes phases de développement de leurs systèmes de calculs que ce soit dans un cadre industriel ou académique. Cette évaluation permettra d'éventuellement rectifier certains choix algorithmiques (software), mais surtout de déterminer vers quels types d'architectures ils vont s'orienter.

Le premier objectif de l'outil est de fournir une estimation du temps de calcul d'un algorithme de traitement d'images sur une architecture hardware fixée par l'utilisateur. Mais d'autres évolutions, tout autant intéressantes et même peut être à plus haute valeur ajoutée, pourront ensuite être intégrées. Ces évolutions pourront guider l'utilisateur dans son choix d'architectures en fonction de contraintes de performance (au sens large) qu'il aura fixé. Les contraintes de performance en question pourront contenir la performance brute (temps de calcul ou nombre d'images par seconde) mais aussi la consommation, la température ou même l'encombrement. L'idée est donc d'avoir un outil complet qui permet d'aider l'utilisateur dans son choix d'architecture hardware et de lui donner des pistes pour améliorer les points critiques de son algorithme. Ces idées seront développées dans la partie perspective de la conclusion.

2.2.2. Choix technologiques

Le choix des langages de programmation utilisés pour réaliser cet outil est conditionné par le public visé. En effet, l'outil doit être accessible par toute personne voulant tester un algorithme et ce sur le plus grand nombre de plateforme possible.

Une manière simple de répondre à ces exigences est de permettre à l'utilisateur d'accéder à l'outil depuis son navigateur internet. Le choix s'est rapidement porté

sur le langage JAVA et (au moins dans un premier temps) sur une applet Java. En effet, ce choix semble assez pertinent, du fait de sa compatibilité avec un maximum de plateformes (Windows, Linux, Mac OS) et de son accessibilité. L'idée étant ensuite de pouvoir bénéficier du retour d'expériences des utilisateurs pour faire évoluer l'outil.

2.2.3. Présentation de l'outil Métis

L'interface présentée (Figure 44) est une première version de Métis développée dans le but de répondre au paradigme défini. Elle permet de créer son algorithme à partir de blocs représentant les briques élémentaires (Figure 45) de la modélisation présentée. Chaque bloc est donc un algorithme de traitement d'images élémentaire contenant des options qui lui sont propres (exemple : taille de filtre pour une convolution 2D) et des options génériques telle que la taille de l'image à traiter. Chaque bloc doit avoir un type d'algorithme élémentaire assigné (Figure 46). L'utilisateur peut ensuite exporter la description d'un l'algorithme de traitement d'images au format XML, pour pouvoir le réimporter par la suite. Le choix de XML est apparu évident pour sa symbiose avec le JAVA et sa modification possible par l'utilisateur.

Enfin, l'interface graphique offre la possibilité à l'utilisateur de choisir l'architecture hardware étudiée parmi un certain nombre d'architectures (stockées dans une base de données) ou permet à l'utilisateur d'ajouter une architecture en intégrant les paramètres de celle-ci.

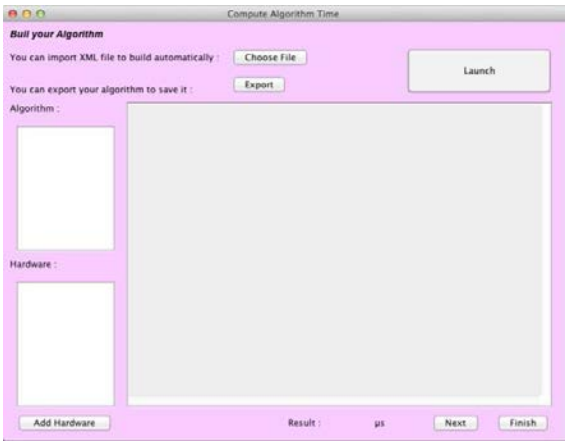


Figure 44 Interface Métis : démarrage

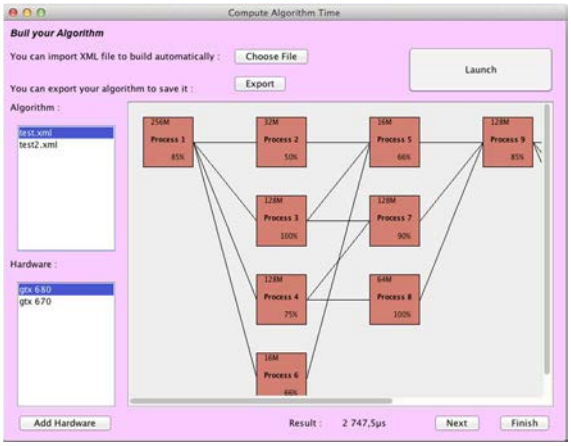


Figure 45 Interface Métis : exemple de construction d'un algorithme

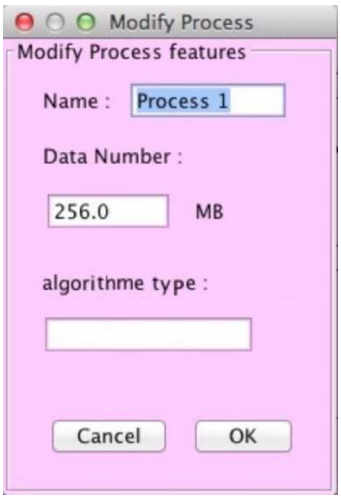


Figure 46 Interface Métis : options d'une brique élémentaire

Ensuite l'algorithme décrit peut être simulé (Figure 47) soit de manière décomposée, bloc par bloc pour obtenir le temps de traitement étape par étape, soit en une seule fois (Figure 48) pour obtenir directement le temps de traitement globale de l'algorithme.

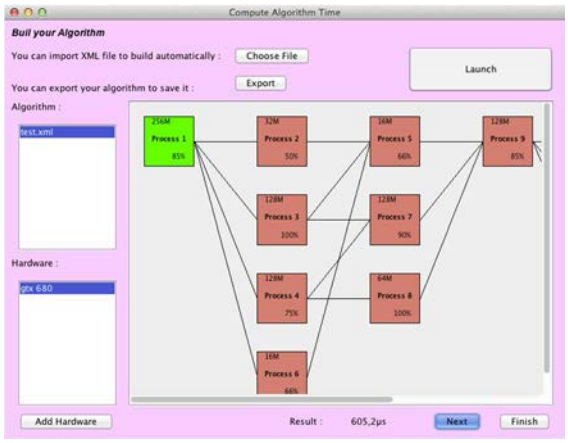


Figure 47 Interface Métis : lancement de la simulation

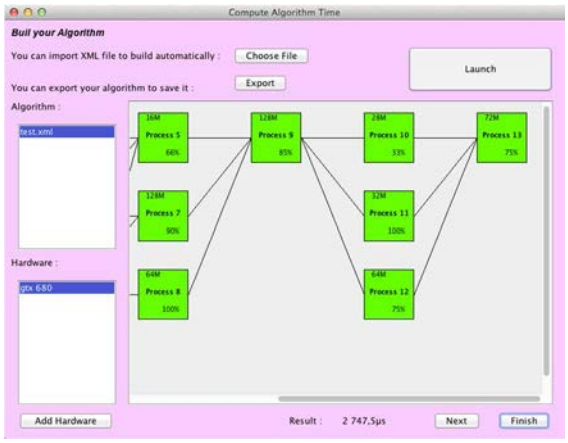


Figure 48 Interface Métis : algorithme traité

Enfin, nous verrons dans le dernier chapitre (Conclusion et Perspectives) que l'outil Métis est destiné à évoluer tant d'un point de vue modélisation (en fonction de l'évolution du paradigme scientifique) que des fonctionnalités offertes (points critiques de l'algorithme, consommation, etc.).

2.3. Démarche de validation

Pour effectuer la validation du paradigme proposé, nous avons décomposé notre étude en deux étapes. Il s'agit, dans un premier temps, de valider une brique élémentaire très utilisée dans le domaine de la vision par ordinateur et dans un second temps, de valider une combinaison de briques élémentaires. Notre validation expérimentale représente une première étape dans la validation de notre paradigme qui lui-même pourra ensuite s'étendre.

Comme décrit dans la partie « caractérisation d'un MBE », la démarche entreprise consiste à d'abord modéliser la partie software du système de calcul et d'ensuite intégrer les paramètres hardwares dans cette modélisation. Sur les architectures de type CPU, l'un des paramètres softwares ayant le plus d'impact sur un algorithme de traitement d'images est bien évidemment la taille de l'image à traiter.

Le but est de d'abord valider qu'il est possible de dégager un MBE sur un des algorithmes les plus utilisés en traitement d'images.

Ensuite, l'idée est d'étudier un algorithme de traitement d'images plus complexe, très connu dans le domaine, qui serait composé d'un certain nombre de briques élémentaires. Cette démarche va permettre, dans un premier temps, de valider la possibilité de modéliser un certain nombre d'algorithmes de traitement d'images élémentaires de manière software (toutes les briques qui composent l'algorithme en question).

Puis, cela va surtout permettre de vérifier que la somme de briques élémentaires permette bien de modéliser un algorithme plus complexe. L'idée est de vérifier si l'addition de chaque modèle software de brique élémentaire donne les mêmes résultats que la résolution de l'algorithme plus complexe. Le but est ici d'estimer la différence entre les résultats modélisés et les résultats expérimentaux.

Enfin, chaque brique élémentaire pourra intégrer les paramètres hardwares des architectures étudiées. Cela permettra d'obtenir une validation d'un premier ensemble de briques élémentaires de traitement d'images.

Les architectures hardwares utilisées pour la validation expérimentale devront représenter au mieux les architectures de type CPU communément utilisées. Pour cela le panel de tous les paramètres définis (Famille, Fréquence d'horloge, Nombre de cores, Performance mémoire) devra être le plus large possible pour que l'influence de chacun de ces paramètres puisse être déterminée.

Chapitre 3 Validation expérimentale

3.1. Validation d'une brique élémentaire

3.1.1. Choix de la brique élémentaire

Dans ces travaux, nous proposons de nous concentrer dans un premier temps, sur un algorithme simple mais incontournable en traitement d'images qui est la convolution 2D. Cet algorithme de base est très utile en traitement d'images, puisqu'il permet le filtrage des images. Différents algorithmes de plus haut niveau en font usage, comme le débruitage par filtre gaussien ou bien la compression en ondelettes.

Cet algorithme transforme une image (de taille Length x Width) par l'intermédiaire d'une matrice N x N (avec N < Length/2 et N< Width/2). La valeur de chaque pixel est donc calculée en multipliant la matrice de convolution avec la matrice centrée, sur le pixel de l'image que l'on veut calculer et de taille N x N (Figure 49).

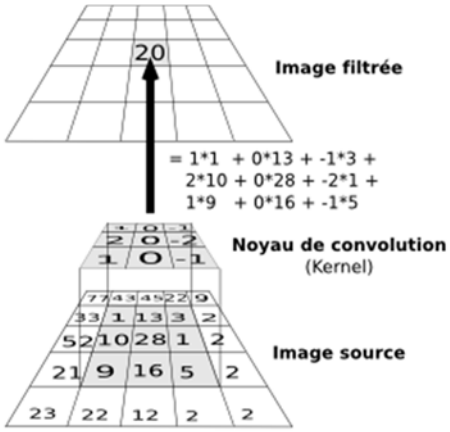


Figure 49 Application de l'algorithme de convolution 2D sur un pixel

Le choix de l'algorithme de la convolution 2D a été fait pour de multiples raisons. Tout d'abord, la résolution de cet algorithme est très demandeur en puissance de calcul car plusieurs opérations sont répétées pour chaque pixel de l'image traitée. D'autre part, sa parallélisation n'est pas aisée car l'opération ne se fait pas point à point. En effet, pour le traitement d'un pixel, il faut accéder aux pixels voisins plus ou moins loin selon la taille du masque. Ces accès aux pixels voisins pourraient, sur une parallélisation qui ne tiendrait pas compte de cela, entraîner des problèmes

mémoires car il pourrait y avoir des accès à une même zone mémoire dans différents processus. Une autre problématique provient du fait que les données accédées en mémoire ne sont pas forcément contiguës, ce qui rajoute de la difficulté à les récupérer dans la mémoire cache. Cependant, malgré ces difficultés, l'opération de convolution est relativement simple et se répète à l'identique pour chaque point de l'image. Etant donné ces informations, on peut comprendre aisément que plusieurs facteurs hardwares peuvent influencer le temps de calcul de la convolution : le pipeline, la taille des caches mémoire, le temps d'accès mémoire, le parallélisme, la fréquence du CPU. L'étude porte donc sur l'influence de ces différents facteurs et pour cela nous avons utilisé différentes architectures hardwares CPU, qui représentent au mieux le panel d'architectures étudiées.

3.1.2. Matériels & méthodes

3.1.2.1. L'algorithme

Nous avons développé un algorithme de convolution en deux dimensions NxN afin d'avoir des accès contigus et non contigus en mémoire. Pour chaque pixel placé en (x,y) sur l'image, notre algorithme consiste en la réalisation du calcul suivant :

$$\boldsymbol{Res[x][y]} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{N}} \sum_{\boldsymbol{l=0}}^{\boldsymbol{N}} \boldsymbol{Filtre[k][l]} \times \boldsymbol{Img}\left[\boldsymbol{x} - \left(\frac{\boldsymbol{N-1}}{\boldsymbol{2}}\right) + \boldsymbol{k}\right]\left[\boldsymbol{y} - \left(\frac{\boldsymbol{N-1}}{\boldsymbol{2}}\right) + \boldsymbol{l}\right] \qquad (2)$$

x allant de [(N-1)/2] à [Height - (N-1)/2], **N** = Taille de la convolution,

y allant de [(N-1)/2] à [Width - (N-1)/2], **Filtre** = Filtre de la convolution,

Res = Image traitée par le filtre de convolution, **Img** = Matrice de l'image,

Width = Largeur de l'image à traiter, **Height** = Hauteur de l'image à traiter.

3.1.2.2. Les cibles

Les tests ont été effectués sur un certain nombre de CPU X86 et ARM ayant les caractéristiques suivantes [1] :

- Un quadri processeur Intel Intel Xeon W3565 3,2 GHz sur un Dell Z4000, contenant trois niveaux de mémoire cache :
  - L1 (4 x 32 KB instruction caches et 4 x 32 KB data caches),
  - L2 (4 X 256 KB),
  - L3 (8MB shared cache).

- Un biprocesseur Intel Core i3 2120T 2,6GHz monté sur mini-itx H67 motherboard, contenant trois niveaux de mémoire cache :
  - L1 (2 x 32 KB instruction caches et 2 x 32 KB data caches),
  - L2 (2 X 256 KB),
  - L3 (3MB shared cache).
- Un biprocesseur Intel Celeron SU2300 1,2GHz monté sur Mini-itx Zotac ION PE Series, contenant deux niveaux de mémoire cache :
  - L1 (2 x 32 KB instruction caches et 2 x 32 KB data caches),
  - L2 (1MB shared cache).
- Un monoprocesseur ARM 11 700MHz, sur Raspberry Pi Model B, contenant deux niveaux de mémoire cache :
  - L1 (16 KB instruction caches et 16 KB data caches),
  - L2 (128KB).
- Un quadri processeur ARM Cortex A15 2,3GHz, sur carte Jetson Tegra K1, contenant deux niveaux de mémoire cache :
  - L1 (2 x 32 KB instruction caches et 2 x 32 KB data caches),
  - L2 (2MB shared cache).

La dernière architecture présentée (Jetson Tegra K1) sera étudiée à pleine puissance (2,3GHz) et à puissance réduite (1,2GHz) pour que l'impact de la fréquence du processeur puisse être jugé de façon totalement unitaire. Ensuite, mis à part pour le Raspberry Pi qui a sa propre distribution linux, toutes les autres plateformes étaient sous linux Ubuntu version 13. Le programme de test a, quant à lui, été écrit en langage C++ et compilé avec g++ v4.x.

Le choix des cibles étudiées a été effectué de façon totalement heuristique dans le but d'obtenir une bonne diversité des cibles hardwares.

3.1.2.3. Les tests

L'algorithme de la convolution 2D a été testé sur les cibles hardwares présentées plus haut en faisant varier leurs paramètres. Ces différents paramètres algorithmiques ont été choisis dans des cas d'utilisations « classiques » qui ont été définis par un travail heuristique :

- la taille de l'image (640 x 480 – 800 x 600 – 1024 x 768 – 1920 x 1080 – 2560 x 1440 – 2880 x 1620 – 3840 x 2160 – 6720 x 3780),
- la taille du filtre (3 x 3 – 5 x 5 – 7 x 7 – 9 x 9 – 11 x 11),
- le traitement Multi-Cœur (Oui – Non).

Pour ce dernier paramètre, nous avons utilisé la librairie Open MP pour le calcul parallèle. Cela nous donne donc 60 possibilités testées pour chacune des architectures ou éventuellement moins si elles ne sont pas multi-cœur.

Dans le but de faire des tests statistiques, plusieurs répétitions (10) de chaque test ont été réalisées. Cela permet aussi de contrôler que l'écart type entre ces différentes occurrences ne soit pas trop important.

3.1.3. Etude de la brique élémentaire : résultats & modélisation

3.1.3.1. Modélisation des paramètres softwares

Les temps de calcul présentés ici, ont résulté de tests effectués sur les différentes architectures décrites ci-dessus. Tous les résultats ne sont pas présentés car nous nous sommes concentrés sur la présentation des résultats les plus caractéristiques. Les conclusions qui résultent des études sur les architectures non présentées seront développées sans qu'il n'y ait besoin de s'étendre sur les résultats car ils sont très similaires aux autres architectures.

Il est important de noter que les écarts entre les différentes occurrences de tests effectués sont très faibles (- de 5% entre min et max). Une moyenne a donc été appliquée sur les dix occurrences pour obtenir les résultats ci-après.

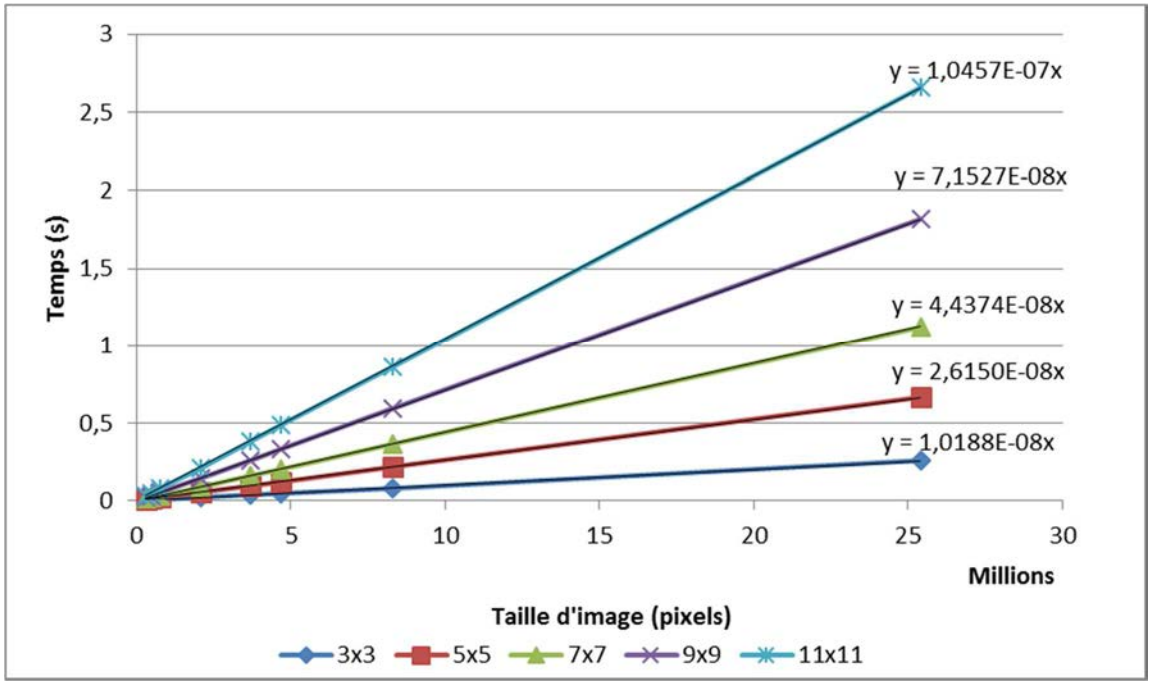


Figure 50 Résultats des tests sur Intel Xeon avec cinq tailles de filtres en Mono Core

La Figure 50 montre l'évolution du temps de calcul d'une convolution 2D en fonction des différentes tailles d'images testées sur un Intel Xeon. Les cinq courbes présentées correspondent aux cinq différentes tailles de filtres étudiés (9, 25, 49, 81 et 121).

Les figures ci-après correspondent à une partie des résultats obtenus mais les représentent parfaitement. En effet, il a été observé une grande similarité entre les résultats quelles que soit les architectures et que ce soit en Mono Core ou en Multi Core.

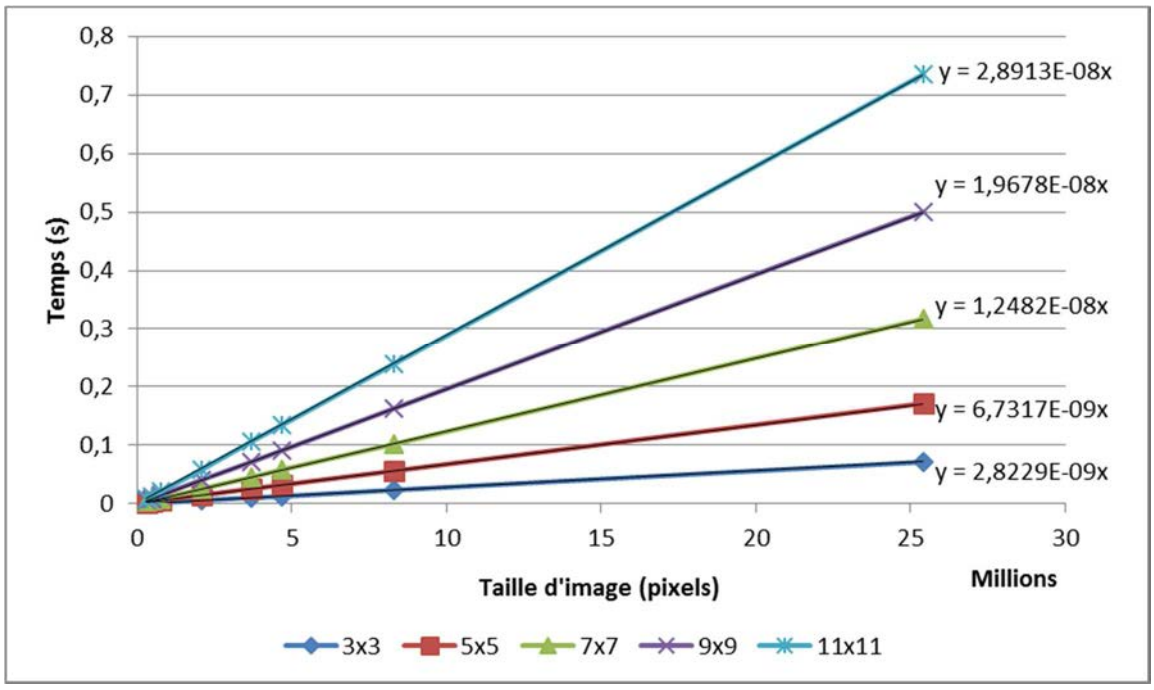


Figure 51 Résultats des tests sur Intel Xeon avec cinq tailles de filtres en Multi Core

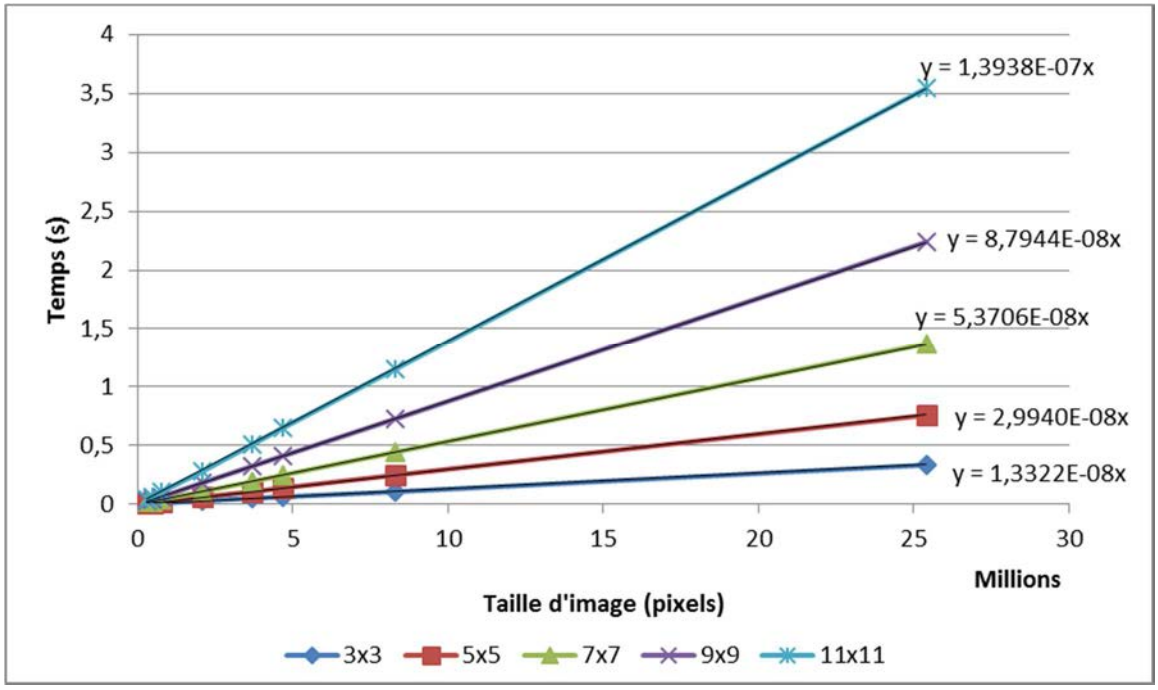


Figure 52 Résultats des tests sur Intel Core I3 avec cinq tailles de filtres en Multi Core

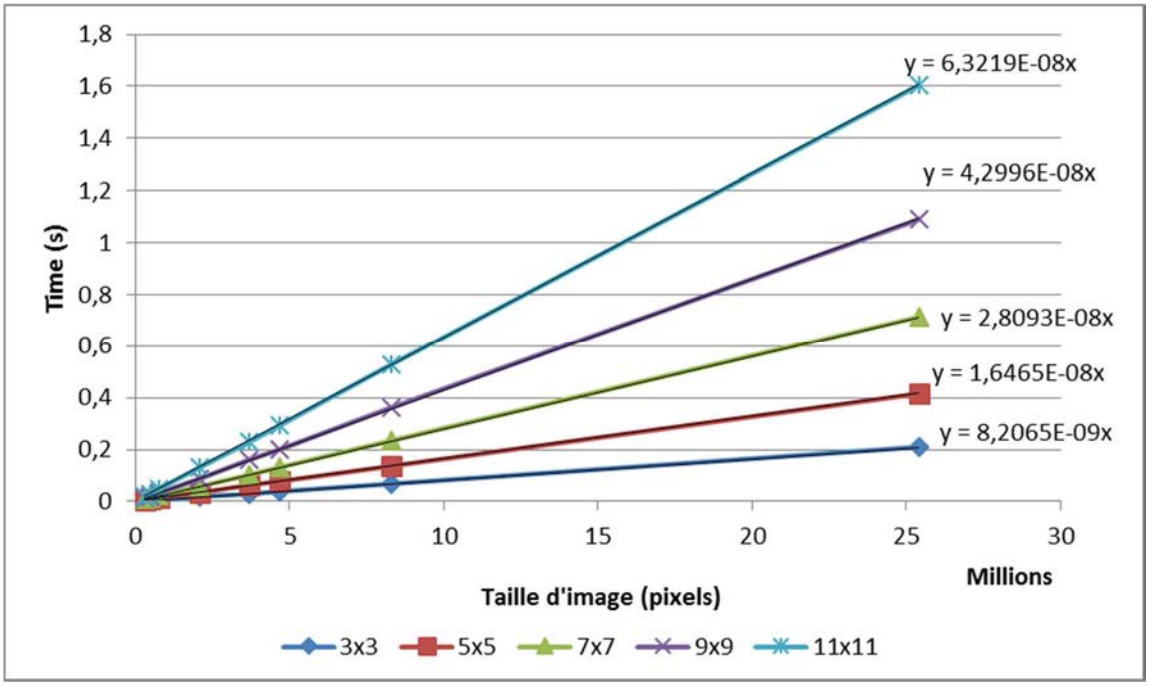


Figure 53 Résultats des tests sur Intel Core I3 avec cinq tailles de filtres en Mono Core

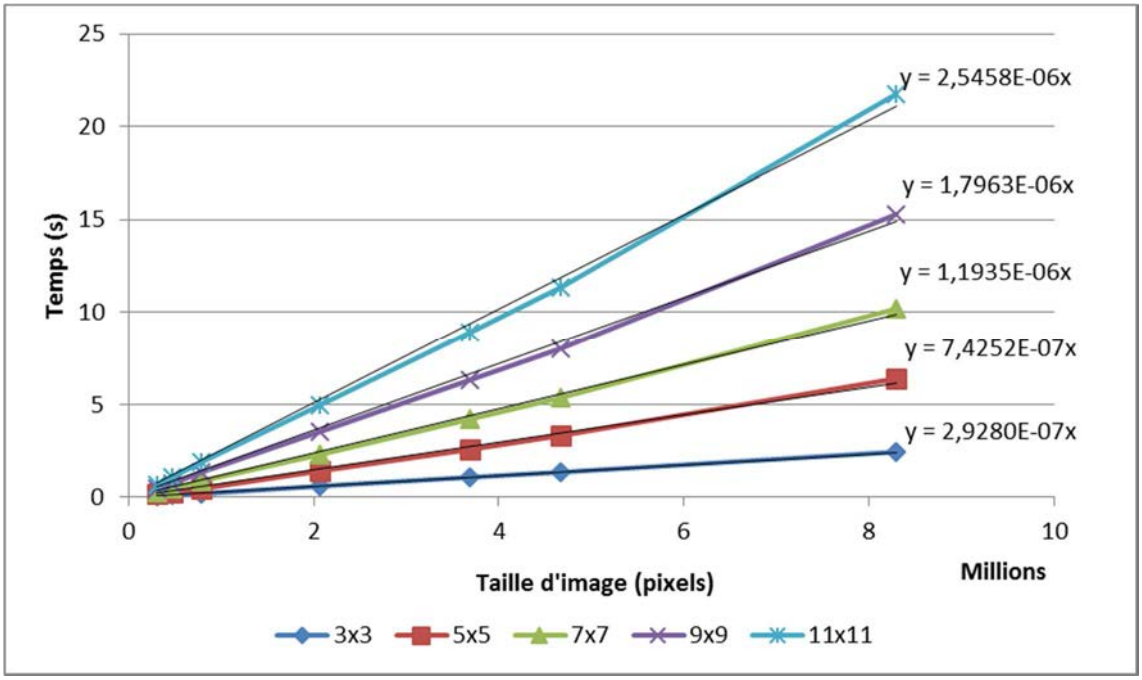


Figure 54 Résultats des tests sur Raspberry Pi avec cinq tailles de filtres en Mono Core

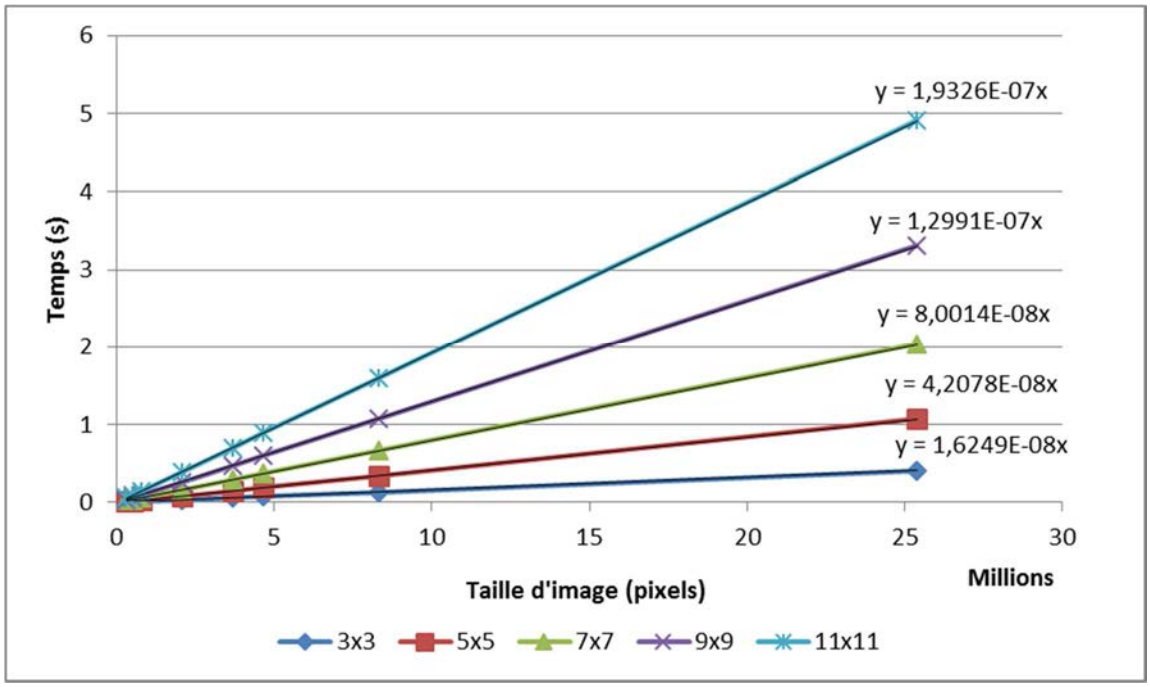


Figure 55 Résultats des tests sur Tegra K1 2,3 Ghz avec cinq tailles de filtres en Mono Core

A l'aide de ces différentes courbes (Figure 50 à Figure 55), on peut aisément arriver à la conclusion que le temps de résolution de l'algorithme est proportionnel à la taille

de l'image. En effet, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur ces différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98, quelle que soit la taille du kernel et quelle que soit l'architecture hardware utilisée. Ces résultats sont aussi vrais en mono core et en multi core.

Conclusion : la formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$:

$$\boldsymbol{Time(s) = a \times ImageSize} \tag{3}$$

Avec « a » différent pour chaque taille de kernel.

Les différents coefficients directeurs « a » de ces droites sont répertoriés pour l'Intel Xeon dans le Tableau 1. Ces différents coefficients sont présentés sur les graphes ci-dessous (Figure 56 à Figure 58) en fonction de la taille du kernel, utilisé pour l'algorithme.

Tableau 1 Coefficient directeur des courbes de linéarité pour l'Intel Xeon :

a (*10^-6)	3x3	5x5	7x7	9x9	11x11
Mono Core	10,188	26,150	44,374	71,527	104,570
Multi Core	2,823	6,732	12,482	19,678	29,913

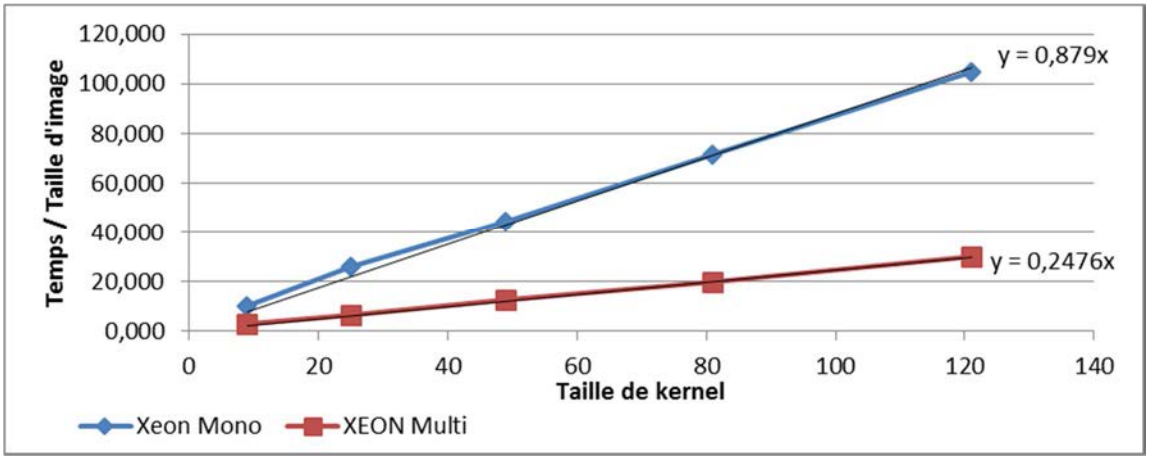


Figure 56 Evolution des coefficients directeurs pour l'Intel Xeon

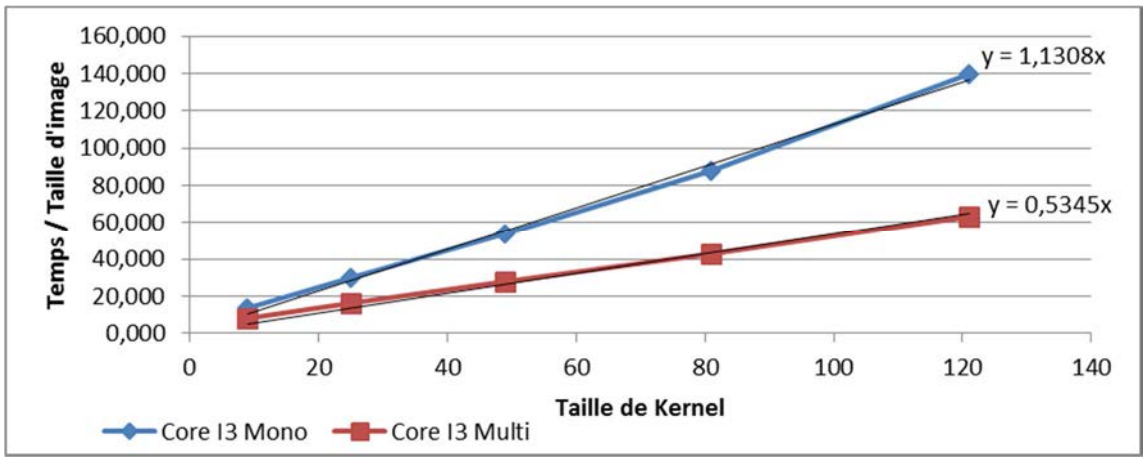


Figure 57 Evolution des coefficients directeurs pour l'Intel Core I3

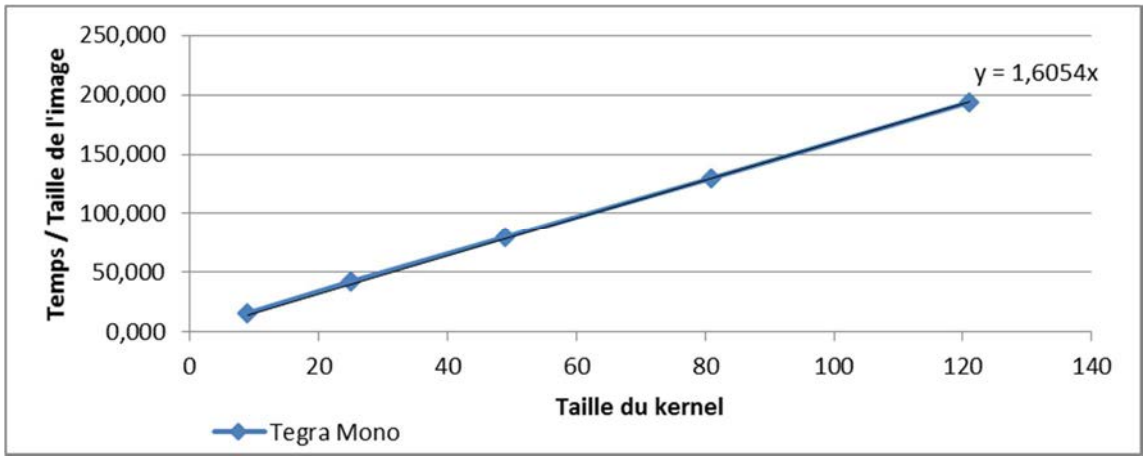


Figure 58 Evolution des coefficients directeurs pour le Tegra K1 à 2,3Ghz

Les résultats obtenus permettent aisément d'arriver à la conclusion que le temps de résolution de l'algorithme est proportionnel au coefficient directeur « a ». En effet, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur ces différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98, quelle que soit l'architecture hardware utilisée. Comme on a pu le constater, ces résultats sont vrais en multi core, comme en mono core.

Conclusion : la formule qui régit donc le temps de calcul en fonction de « a » est de la forme $y = a\,x$ et on peut donc, en partant de l'équation (3) poser l'équation suivante :

$$\boldsymbol{Time(s) = A \times KernelSize \times ImageSize} \qquad (4)$$

Avec : $a = A \times KernelSize$

Tableau 2 Coefficients (A) de l'équation (3) pour toutes les architectures étudiées

A (*10^-9)	XEON	Core i3	SU2300	Raspberry Pi	Tegra K1 (2,3Ghz)	Tegra K1 (1,2Ghz)
Mono Core	0,879	1,131	2,680	21,935	1,605	3,218
Multi Core	0,248	0,535	1,493	-	-	-

Linéarité sur des tailles d'images et filtres très grands : on observe, que même lorsque l'on augmente la taille du filtre de façon très importante (jusqu'à 31 x 31) ou la taille de l'image (56MPixel), la performance des architectures reste proportionnelle à ces différentes tailles.

Les coefficients obtenus dans le Tableau 2 permettent d'obtenir le temps de calcul de la convolution 2D sur une architecture donnée. Il s'agit maintenant de reconstruire le modèle à partir des caractéristiques techniques des différentes architectures étudiées.

3.1.3.2. Modélisation des paramètres hardwares

Il paraît déjà évident que la fréquence du processeur ne peut pas justifier à lui seul les différences de performance. Par exemple, le Core I3 a une fréquence 2,16 fois supérieure au SU2300 mais il est 2,37 fois plus rapide que lui. Un exemple encore plus flagrant est la différence entre le Core I3 et le Raspberry Pi. En effet, le Core I3 a une fréquence 3,7 fois supérieure au Raspberry Pi mais il a une performance 19,4 fois supérieure.

Néanmoins, la fréquence d'horloge du processeur reste l'un des paramètres les plus incontournables influençant la vitesse de calcul. Le temps de calcul est en effet inversement proportionnel à la fréquence d'horloge. Cela se constate d'ailleurs avec les deux résultats obtenus sur la Tegra K1. En effet, en multipliant la fréquence d'horloge par deux (2,3/1,2 = 1,92), on obtient bien un temps de calcul deux fois moins important (3,218/1,605 = 2) sur la même architecture ; les autres caractéristiques de l'architecture étant fixées.

On arrive donc à la conclusion que pour une taille d'image et un filtre de convolution donnés, on peut fixer « $B = A \times Fréquence$ » et en repartant de l'équation (4), obtenir l'équation suivante :

$$\boldsymbol{Time(s)} = \frac{\boldsymbol{B}}{\boldsymbol{Frequency}} \times \boldsymbol{FilterSize} \times \boldsymbol{ImageSize} \qquad (5)$$

Dans un premier temps, l'étude porte sur les résultats Mono-Core puis le paramètre du traitement multi Core sera traité par la suite. Les différents calculs du coefficient B sont intégrés au Tableau 3.

Tableau 3 Coefficients B de l'équation (5) pour toutes les architectures étudiées

Mono Core	XEON	Core i3	SU2300	Raspberry Pi	Tegra K1 (2,3 GHz)	Tegra K1 (1,2 GHz)
A (*10^-9)	0,879	1,131	2,680	21,935	1,605	3,218
Fréquence(GHz)	3,2	2,6	1,2	0,7	2,3	1,2
B	2,813	2,940	3,216	15,355	3,692	3,862

Il a été mis en évidence dans le chapitre 1, que les caractéristiques d'un CPU qui influent le plus sur les performances d'un algorithme de traitement d'images sont essentiellement la fréquence d'horloge du processeur, la taille, le nombre et la vitesse d'accès des différentes mémoires caches.

Nous avons développé un software qui permet d'évaluer la vitesse des accès mémoires. Cet outil permet de déterminer la vitesse de transfert à différents niveaux d'utilisation mémoire. Les architectures utilisées ont été testées à un grand nombre de niveau d'utilisation et il ressort qu'à chaque fois que la taille mémoire, nécessaire au test, dépasse la taille d'un niveau de cache, la performance diminue. Par exemple, pour le Core I3, lorsque la mémoire nécessaire est supérieure à 32KB (taille de la mémoire cache L1), la vitesse d'accès passe de 1000MB/s à 8900 MB/s et ainsi de suite lorsque le niveau dépasse L2 et L3 (Voir Tableau 4).

Tableau 4 Résultats des tests mémoires à différents niveaux d'utilisation

Mémoire utilisée	XEON (Mo/s)	Core I3 (Mo/s)	SU2300 (Mo/s)	RaspberryPi (Mo/s)
≤ 32 KB	13500	10000	4700	500
32 KB → 128 KB	10400	8900	2700	240
128 KB → 256 KB	10400	8900	2700	120
256 KB → 1MB	10400	7500	2700	120
1 MB → 3MB	10400	7500	1350	120
3MB → 8MB	10400	4780	1350	120
> 8MB	3900	4780	1350	120

Comme pour la fréquence, le temps de calcul est inversement proportionnel à la performance mémoire. Mais cette performance mémoire n'affecte pas directement le

temps de calcul, en tout cas, pas de façon linéaire mais de façon affine ($y = a \times x + b$). En reprenant l'équation (5), on peut en déduire que :

$$\boldsymbol{B = \frac{C}{MemoryPerf} + D} \tag{6}$$

Dans l'hypothèse où les coefficients C et D de l'équation (6) sont communs à toutes les architectures, on obtient les équations suivantes :

$$\begin{aligned} \mathbf{2,940} &= \frac{\boldsymbol{C}}{\mathbf{4780}} + \boldsymbol{D} \\ \mathbf{3,216} &= \frac{\boldsymbol{C}}{\mathbf{1350}} + \boldsymbol{D} \\ \mathbf{15,355} &= \frac{\boldsymbol{C}}{\mathbf{120}} + \boldsymbol{D} \end{aligned} \tag{7}$$

Cette équation a été résolue à l'aide de la méthode des moindres carrés pour obtenir C =1558,7 et D = 2,347.

Tableau 5 Différence entre les coefficients B obtenus dans le Tableau 3 et ceux prédits à l'aide des coefficients C et D

	Core I3	**SU2300**	**Raspberry Pi**	**XEON**
B obtenu	2,940	3,216	15,355	2,813
B prédit	2,673	3,501	15,366	2,746

La différence entre le coefficient B obtenu et le coefficient B prédit est raisonnable. Mais surtout, on peut s'apercevoir que le résultat du coefficient B sur Intel Xeon, qui n'a pas été pris en compte dans le calcul des moindres carrés, est prédit avec une erreur de 2,4%.

On obtient donc l'équation (8) suivante comme MBE pour l'algorithme de la Convolution 2D en mono core :

$$\boldsymbol{Time(s) = \frac{\frac{1558,7}{MemoryPerf} + 2,347}{Frequency} \times FilterSize \times ImageSize} \tag{8}$$

3.1.3.3. Conclusion

Finalement, il a donc été démontré qu'il est possible de prédire la performance d'une brique élémentaire, comme nous l'avons défini dans le chapitre 2 (ici la convolution 2D) et qu'elle répond à une même équation, quelles que soient les paramètres de l'algorithme et quelle que soit l'architecture CPU utilisée. Un premier MBE de la convolution 2D est donc défini.

Une généralisation a été proposée avec un modèle linéaire qui donne des résultats convaincants. Il pourra être intéressant, par la suite, d'intégrer aussi le paramètre référant au type d'architecture (RISC ou CISC). En effet, le fait d'utiliser une architecture ARM ou X86 peut avoir une incidence importante sur la performance d'un système de calcul.

La question est donc maintenant de savoir si cette méthode peut s'appliquer à toutes les « briques » élémentaires de traitement d'images et surtout si une combinaison de briques élémentaires peut être déterminée par la modélisation de chacune d'elles.

3.2. Validation d'une combinaison de briques élémentaires

Maintenant que la modélisation d'une brique élémentaire a été validée avec un algorithme fortement utilisé en traitement d'images, qu'est la Convolution 2D, il s'agit alors de voir si la composition de modèles de briques élémentaires peut permettre d'obtenir la performance d'un algorithme plus complexe regroupant les briques en question.

3.2.1. Choix de l'algorithme étudié :

Il s'agit ici d'étudier un algorithme de traitement d'images bien connu dans le domaine et qui regroupe un certain nombre de briques élémentaires. Notre choix s'est rapidement porté sur l'algorithme des ondelettes. En effet, cet algorithme utilise des briques élémentaires très utilisées par ailleurs dans le traitement d'images. De plus il est itératif dans le sens où il est basé sur une répétition d'instructions à différentes échelles. Pour toutes ces raisons, notre choix s'est porté sur cet algorithme regroupant plusieurs algorithmes essentiels et incontournables dans le traitement d'images comme la convolution, le seuillage, l'Upsampling ou la Decimation. La convolution 2D a fait l'objet d'une étude. Il reste donc maintenant à appliquer notre méthode de test sur les autres algorithmes pour évaluer la prédictibilité de ceux-ci.

L'algorithme de traitement d'images se décompose donc en différents blocs élémentaires que sont :

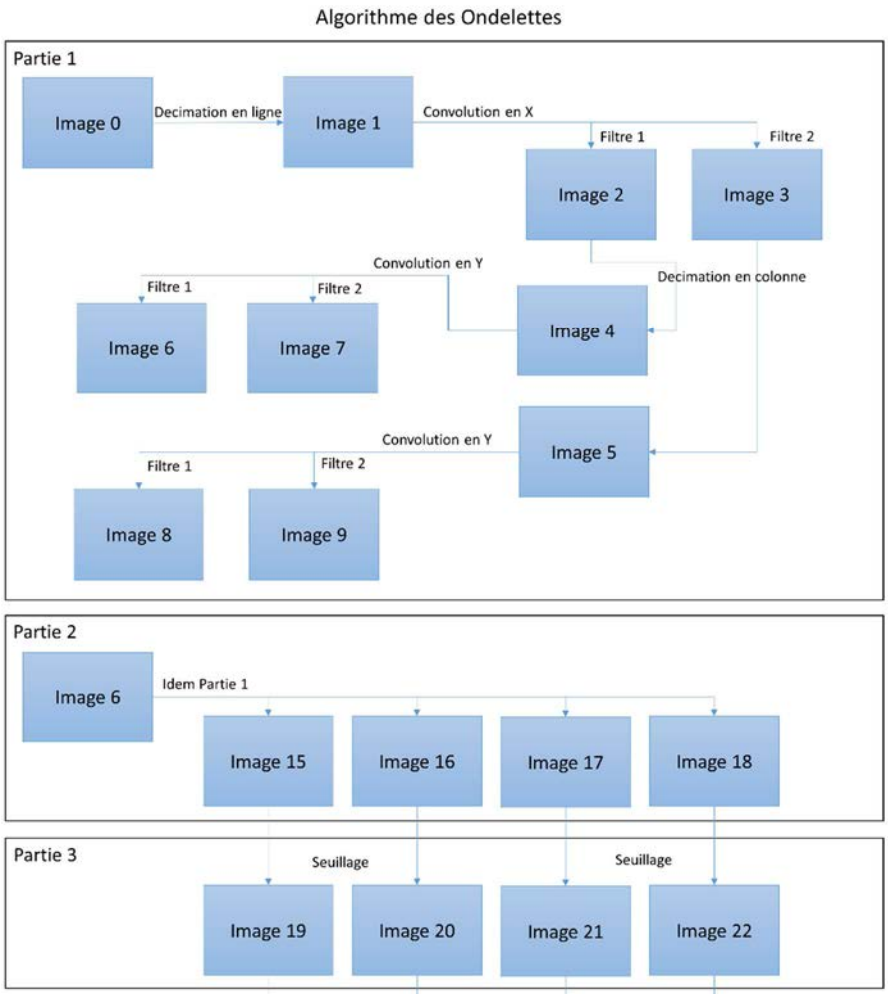
- Le Decimation : Une colonne ou une ligne sur deux de l'image est conservée. Cela divise donc par deux le nombre de ligne ou le nombre de colonnes de l'image.
- L'Upsampling : C'est l'opération inverse de la Decimation, qui consiste à rajouter des zéros une ligne sur deux ou une colonne sur deux. Cela multiplie donc par deux le nombre de lignes ou le nombre de colonnes.
- Addition de deux matrices : Addition de chaque pixel d'une image à son homologue de même taille.
- Seuillage : Algorithme répondant à l'équation suivante : $Seuillage\,(x) = signe(x) \times \max[abs(x) - T, 0]$ avec x correspondant aux valeurs de la matrice et T étant le seuil.
- Convolution : Algorithme similaire à la convolution 2D mais avec un filtre de convolution, se restreignant à une seule dimension, soit horizontale (en x), soit verticale (en y).

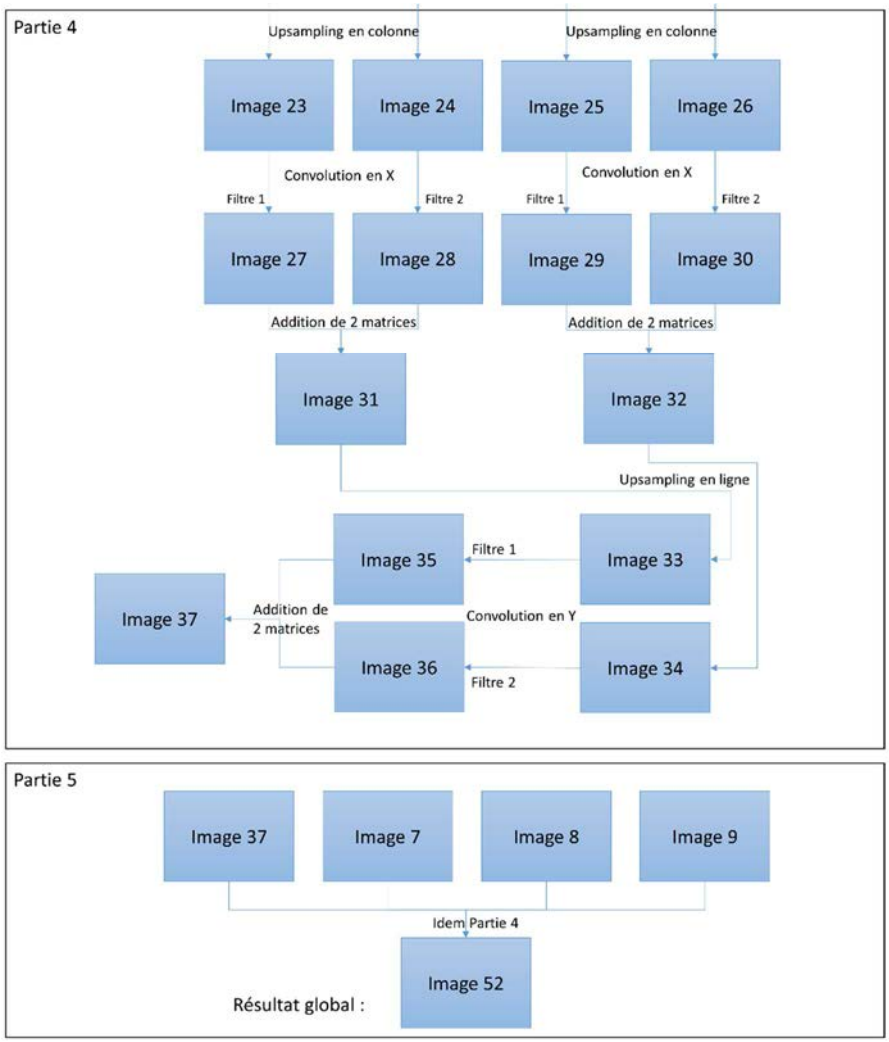
3.2.2. Matériels et méthodes

Le but est d'étudier chacun des algorithmes séparément pour déterminer le modèle de chacun et obtenir sa performance en fonction des spécifications de l'architecture hardware étudiée. Ensuite, l'idée est de comparer ces résultats avec les performances de l'algorithme des ondelettes qui regroupent les algorithmes étudiés.

3.2.2.1. L'algorithme

L'algorithme global des ondelettes se décompose en cinq parties avec l'image de départ définie comme « Image 0 » avec une taille W x H (Weight x Height) :





3.2.2.2. Les cibles

Les cibles étudiées sont identiques à celles utilisées pour l'analyse de la convolution 2D :

- Un quadri processeur Intel Intel Xeon W3565 3,2 GHz sur un Dell Z4000, contenant trois niveaux de mémoire cache.
- Un biprocesseur Intel Core i3 2120T 2,6GHz monté sur mini-itx H67 motherboard, contenant trois niveaux de mémoire cache,
- Un biprocesseur Intel Celeron SU2300 1,2GHz monté sur Mini-itx Zotac ION PE Series, contenant deux niveaux de mémoire cache,
- Un monoprocesseur ARM 11 700MHz, sur Raspberry Pi Model B, contenant deux niveaux de mémoire cache,

- Un quadri processeur ARM Cortex A15 2,3GHz, sur carte Jetson Tegra K1, contenant deux niveaux de mémoire cache.

3.2.2.3. Tests

Comme pour l'algorithme de la convolution 2D, chaque test logiciel a été effectué dix fois sur chaque architecture (présentées plus haut). Ensuite, pour chaque test logiciel, il a fallu faire varier les paramètres suivant :

- la taille de l'image (640 x 480 – 800 x 600 – 1024 x 768 – 1920 x 1080 – 2560 x 1440 – 2880 x 1620 – 3840 x 2160 – 6720 x 3780),
- le traitement Multi-Cœur (Oui – Non).

Pour ce dernier paramètre, nous avons utilisé la librairie Open MP pour le calcul parallèle. Cela nous donne donc 60 possibilités testées pour chacune des architectures ou éventuellement moins si elles ne sont pas multi-cœur.

Pour chacun de ces paramètres, il a fallu tout d'abord tester chaque brique élémentaire de l'algorithme, c'est-à-dire une convolution en X, une convolution en Y, un Upsampling, une Decimation, une addition de deux matrices et un seuillage. Puis, il a ensuite fallu lancer l'algorithme complet des ondelettes avec chacun de ces paramètres.

Dans un premier temps, il s'agit de modéliser les paramètres softwares de chacune des briques d'un côté et de modéliser les paramètres softwares de l'algorithme des ondelettes de l'autre, afin de pouvoir comparer les deux modélisations. En effet, il faut que l'addition des briques modélisées corresponde au modèle des ondelettes, pour pouvoir ensuite espérer modéliser un tout autre algorithme utilisant les briques élémentaires modélisées.

Ce n'est qu'après cela, que l'on pourra se pencher sur la modélisation hardware de chaque brique élémentaire ; c'est-à-dire de modéliser ces briques en fonction des paramètres des différentes architectures étudiées.

3.2.3. Etude d'une combinaison de briques élémentaires : résultats et modélisation

Comme pour la convolution 2D, les résultats des différentes occurrences de tests ont un écart type très faible (<5%), ce qui a permis de considérer les résultats comme une moyenne de ces différentes occurrences. Dans un premier temps, l'étude se concentre sur la modélisation software pour valider la méthode des briques puis enchaîne sur la modélisation hardware.

3.2.3.1. Modélisation software des briques élémentaires

Il s'agit tout d'abord de modéliser de manière software chaque brique élémentaire d'un côté et de modéliser aussi de manière software l'algorithme des ondelettes d'un autre côté pour tout d'abord vérifier que l'addition de chaque brique élémentaire donne bien le même résultat que l'algorithme complet.

Modélisation de l'addition de matrice

L'addition de deux images est un algorithme simple mais qui ne donne pas forcément des résultats similaires à la convolution 2D (Exemples de résultats Figure 59 et Figure 60).

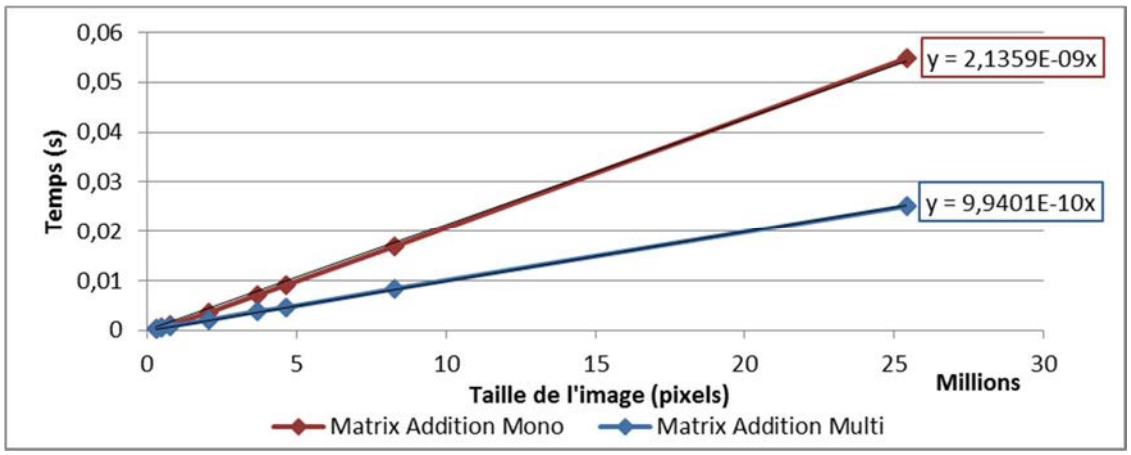


Figure 59 Résultats d'additions de deux images sur Intel Core I3

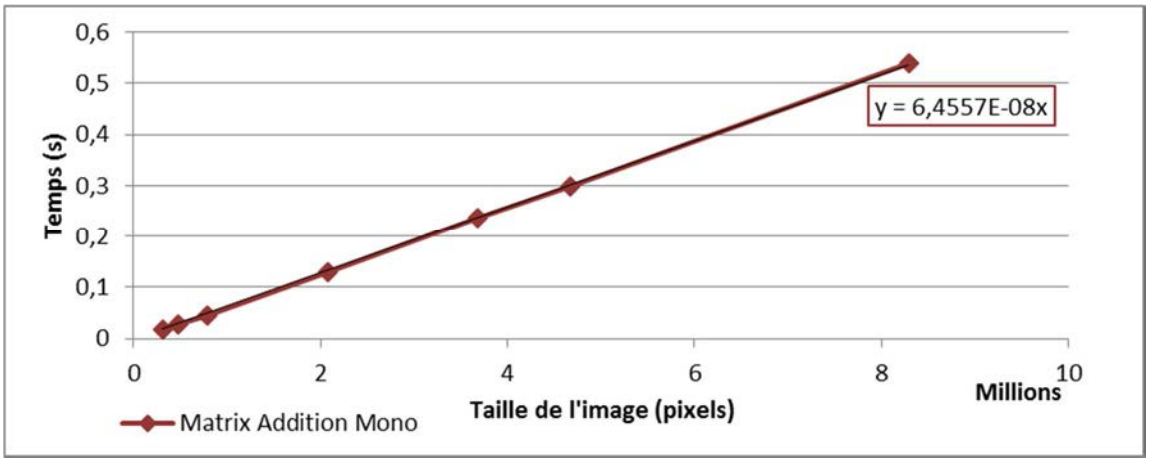


Figure 60 Résultats d'additions de deux images sur Raspberry Pi

Lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours

supérieurs à 0,98, quelle que soit l'architecture hardware utilisée. Ces résultats sont aussi vrais en mono core qu'en multi core. Le temps de calcul est donc proportionnel au nombre de pixels traités.

Conclusion : la formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = Coef_{Add} :

$$\boldsymbol{Time_{Add}(s) = Coef_{Add} \times ImageSize} \tag{9}$$

Tableau 6 Coefficients (Add) de l'équation (9) pour toutes les architectures étudiées

CoefAdd	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	1,897	2,136	6,550	64,557	13,328	26,092
Multi Core	1,259	0,994	4,472	-	-	-

Les coefficients du Tableau 6 sont différents selon les architectures et selon le nombre de cœurs utilisés pour la résolution du problème. On peut constater avec le Tegra K1 que la performance parait être proportionnelle à la fréquence d'horloge (fréquence 2x supérieure => temps de calcul 2x moins important). Mais par contre, contrairement à la convolution 2D, le nombre de cœur n'influence pas la performance de manière proportionnelle. La modélisation « hardware » sera traitée par la suite.

Modélisation de l' « Upsampling » en colonne

L'Upsampling en colonne se résume à des opérations mémoires. Pour moitié il s'agit de recopie mémoire (recopie de chaque pixel une colonne sur deux) et pour l'autre de définition de location mémoire (Valeur « 0 » une colonne sur deux).

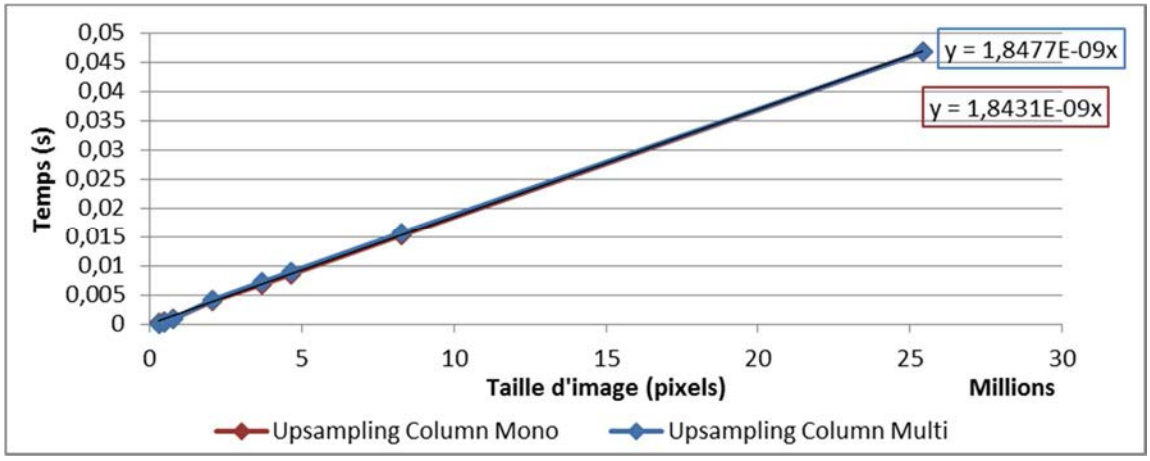


Figure 61 Résultats de l' « Upsampling » en colonne sur Intel Xeon

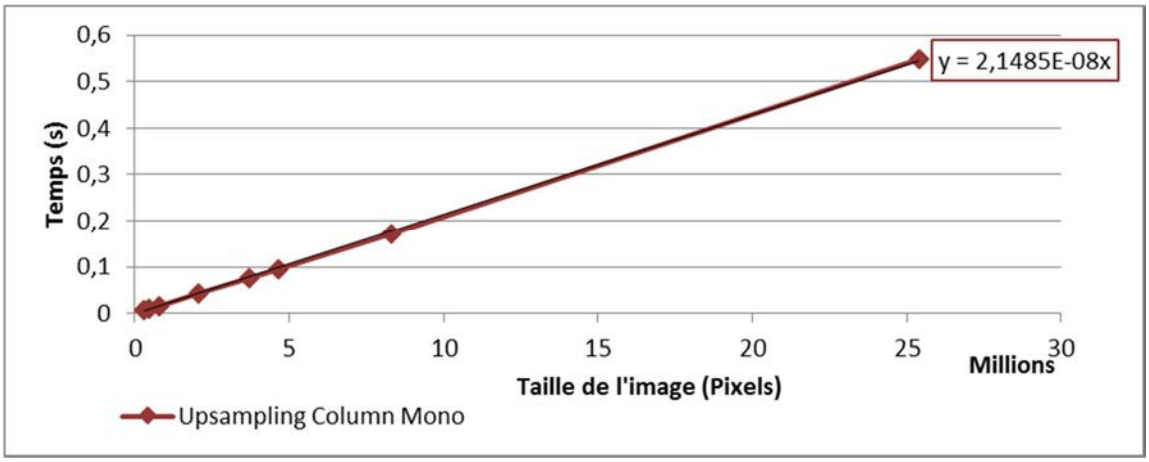


Figure 62 Résultats de l' « Upsampling » en colonne sur Tegra K1 réduit à 1,2 Ghz

De même que précédemment, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98.

La formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = Coef_{UpC} :

$$\boldsymbol{Time_{UpC}(s) = Coef_{UpC} \times ImageSize} \tag{10}$$

Tableau 7 Coefficients (UpC) de l'équation (10) pour toutes les architectures étudiées

Coef_{UpC}	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	1,843	1,289	5,377	38,677	10,891	21,485
Multi Core	1,848	1,2594	5,306	-	-	-

Les coefficients du Tableau 7 sont différents selon les architectures et selon le nombre de cœurs utilisés pour la résolution du problème. On peut constater avec le Tegra K1 que la performance parait être proportionnelle à la fréquence d'horloge (fréquence 2x supérieure => temps de calcul 2x moins important). Mais par contre, contrairement à la convolution 2D, le nombre de cœur n'influence pas la performance de manière proportionnelle. La modélisation « hardware » sera traitée par la suite.

Modélisation de l' « Upsampling » en ligne

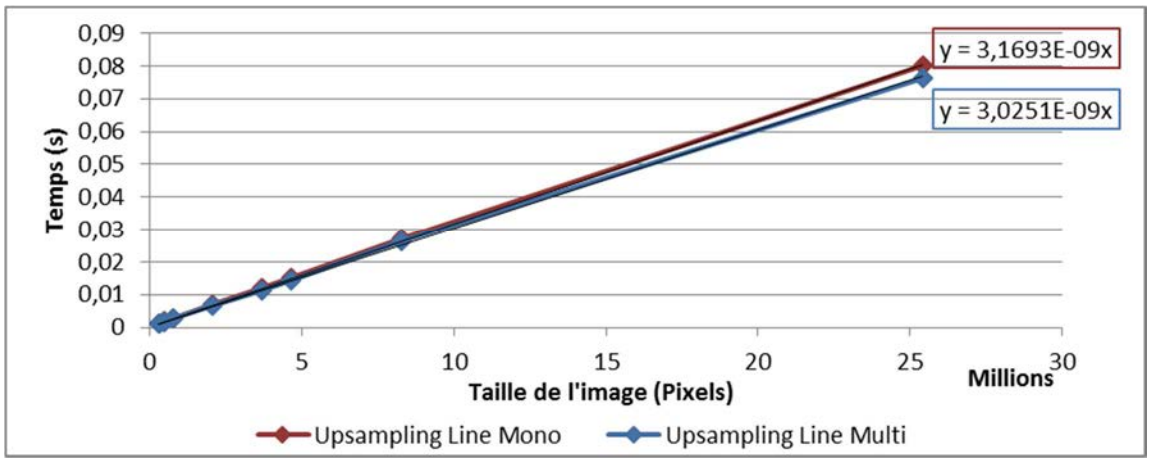


Figure 63 Résultats de l' « Upsampling » en ligne sur Intel SU2300

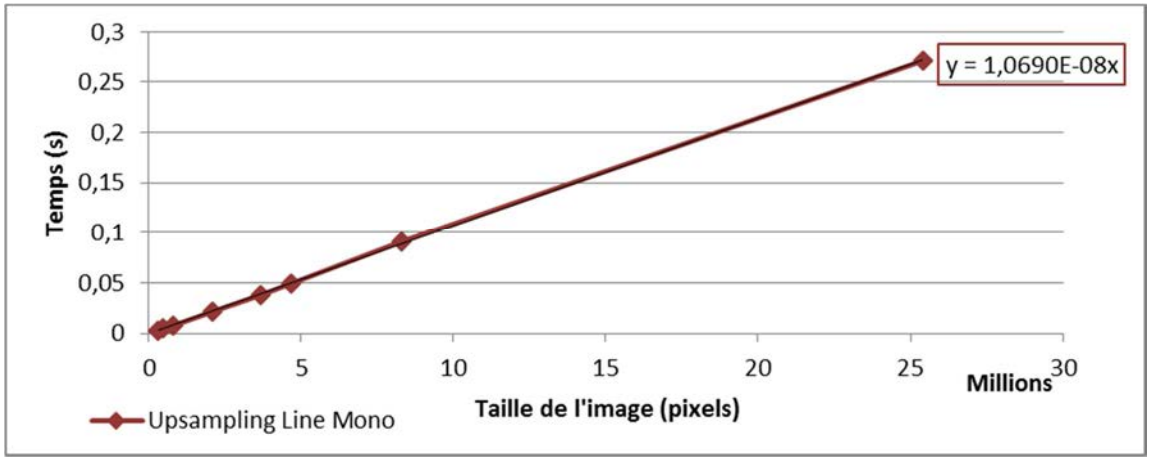


Figure 64 Résultats de l' « Upsampling » en ligne sur Tegra K1 à 2,3 Ghz

De même que précédemment, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98. La formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = Coef_{UpL} :

$$\boldsymbol{Time_{UpL}(s) = Coef_{UpL} \times ImageSize} \tag{11}$$

Cette modélisation permet d'obtenir un coefficient pour chaque architecture hardware en Mono Core et en Multi Core :

Tableau 8 Coefficients (UpL) de l'équation (11) pour toutes les architectures étudiées

$Coef_{UpL}$	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	1,118	0,925	3,169	38,253	10,690	20,445
Multi Core	1,134	0,770	3,025	-	-	-

Les coefficients du Tableau 8 sont différents selon les architectures et selon le nombre de cœurs utilisés pour la résolution du problème. On peut toujours constater avec le Tegra K1 que la performance parait être proportionnelle à la fréquence d'horloge (fréquence 2x supérieure => temps de calcul 2x moins important). Le nombre de cœur parait par contre avoir peu d'influence sur la performance. La modélisation « hardware » sera traitée par la suite.

Modélisation du « Decimation » en colonne

Pour sa part, l'algorithme de Decimation en colonne n'est finalement que la copie de la moitié des colonnes d'une image dans une seconde (Exemples de résultats Figure 65 et Figure 66).

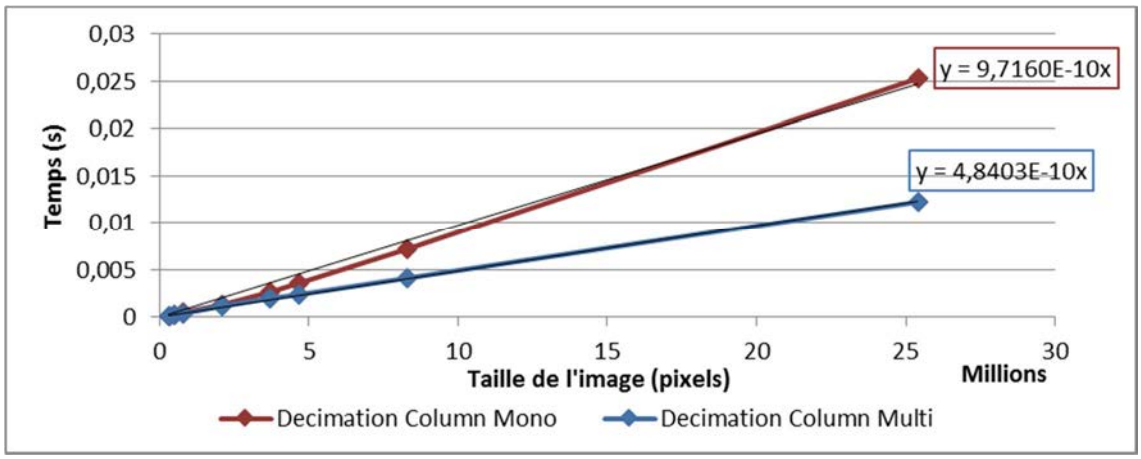


Figure 65 Résultats de « Decimation » en colonne sur Intel Core I3

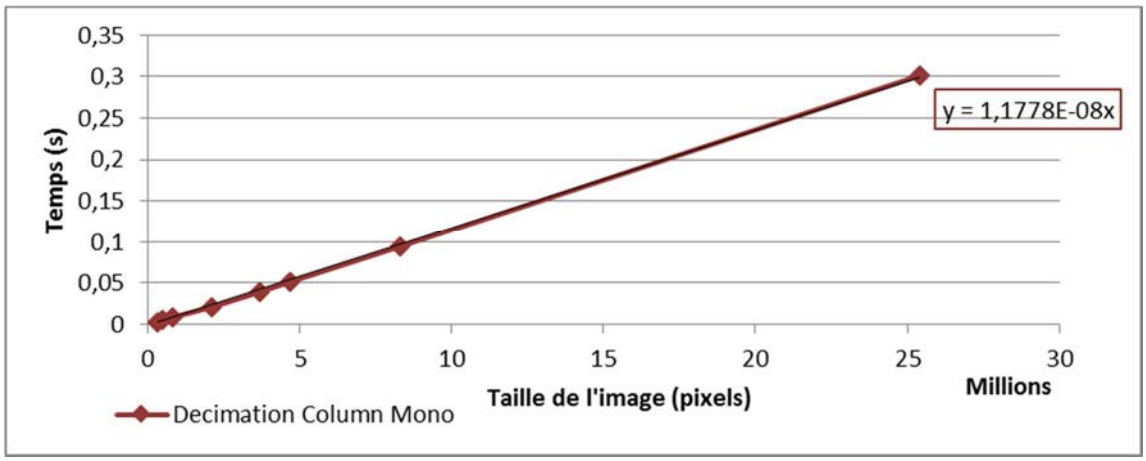


Figure 66 Résultats de « Decimation » en colonne sur Tegra K1 réduit à 1,2 Ghz

De même que précédemment, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98. La formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = $Coef_{DeC}$:

$$\boldsymbol{Time_{DeC}(s) = Coef_{DeC} \times ImageSize} \qquad (12)$$

Cette modélisation permet d'obtenir un coefficient pour chaque architecture hardware en Mono Core et en Multi Core :

Tableau 9 Coefficients (DeC) de l'équation (12) pour toutes les architectures étudiées

CoefDeC	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	0,862	0,972	3,023	34,926	6,067	11,778
Multi Core	0,642	0,484	2,151	-	-	-

Les coefficients du Tableau 9 sont différents selon les architectures et selon le nombre de cœurs utilisés pour la résolution du problème. On peut toujours constater avec le Tegra K1 que la performance parait être proportionnelle à la fréquence d'horloge. Contrairement à ce que l'on a pu relever avec l'Upsampling, le nombre de cœur a une forte influence sur la Decimation. La modélisation « hardware » sera traitée par la suite.

Modélisation du « Decimation » en ligne

L'algorithme de Decimation qui n'est finalement que la copie de la moitié des lignes d'une image (Exemples de résultats Figure 67 et Figure 68).

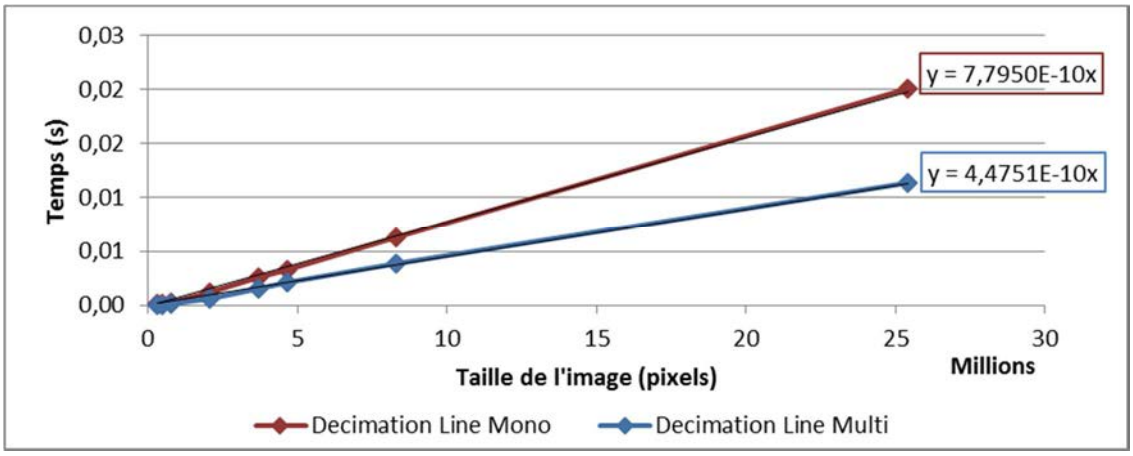


Figure 67 Résultats de « Decimation » en ligne sur Intel Xeon

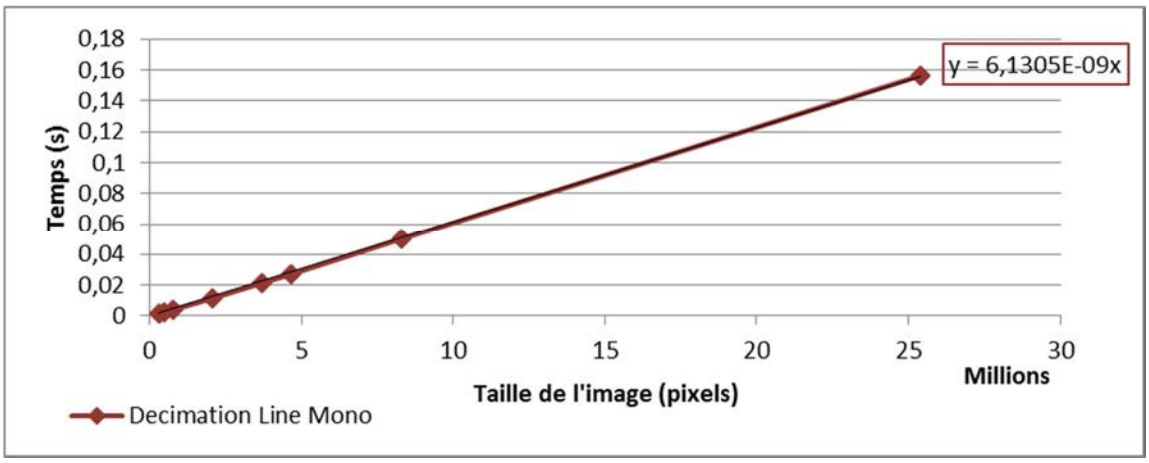


Figure 68 Résultats de « Decimation » en ligne sur Tegra K1 à 2,3 Ghz

De même que précédemment, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98. La formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = $Coef_{DeL}$:

$$\boldsymbol{Time_{DeL}(s) = Coef_{DeL} \times ImageSize} \tag{13}$$

Cette modélisation permet d'obtenir un coefficient pour chaque architecture hardware en Mono Core et en Multi Core :

Tableau 10 Coefficients (DeL) de l'équation (13) pour toutes les architectures étudiées

$Coef_{DeL}$	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	0,780	1,016	2,882	22,585	6,131	11,941
Multi Core	0,448	0,417	1,896	-	-	-

Les coefficients du Tableau 10 sont différents selon les architectures et selon le nombre de cœurs utilisés pour la résolution du problème. Les résultats paraissent assez similaires à la Decimation en colonne mais la modélisation « hardware » sera traitée par la suite.

Modélisation du seuillage

Le seuillage donne des résultats très différents selon les architectures et l'utilisation du multi core mais il est reste tout de même proportionnel aux nombres de pixels traités (Exemples de résultats Figure 69 et Figure 70).

De même que précédemment, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98.

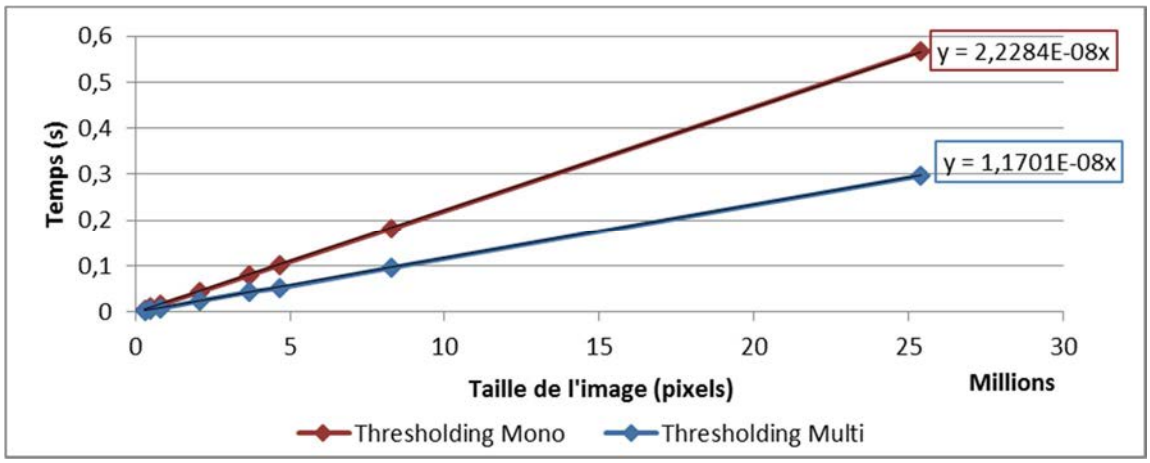


Figure 69 Résultats du seuillage sur Intel SU2300

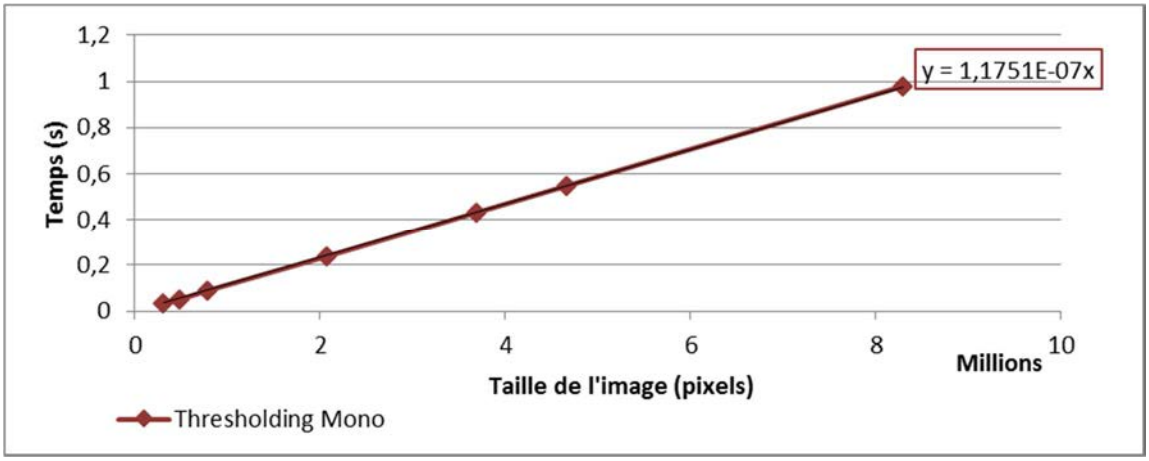


Figure 70 Résultats du seuillage sur Raspberry Pi

La formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = $Coef_{Seu}$:

$$\boldsymbol{Time_{Seu}(s) = Coef_{Seu} \times ImageSize} \tag{14}$$

Cette modélisation permet d'obtenir un coefficient pour chaque architecture hardware en Mono Core et en Multi Core :

Tableau 11 Coefficients (Seu) de l'équation (14) pour toutes les architectures étudiées

$Coef_{Seu}$	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	3,835	6,847	22,284	117,510	40,835	79,619
Multi Core	1,776	5,227	11,701	-	-	-

Les coefficients du Tableau 11 sont différents selon les architectures et selon le nombre de cœurs utilisés pour la résolution du problème. On peut constater avec le Tegra K1 que la performance parait être proportionnelle à la fréquence d'horloge. Par contre, le nombre de cœurs n'influence pas la performance de manière proportionnelle, néanmoins la performance mémoire combinée pourra expliquer ces écarts. Le seuillage est un algorithme plus complexe qui demande un travail de modélisation supérieur. La modélisation « hardware » sera traitée par la suite.

Modélisation de la convolution en X

La Convolution en X (filtre à une dimension de manière verticale) donne des résultats très similaires à l'étude de la convolution2D (Exemples de résultats Figure 71 et Figure 72).

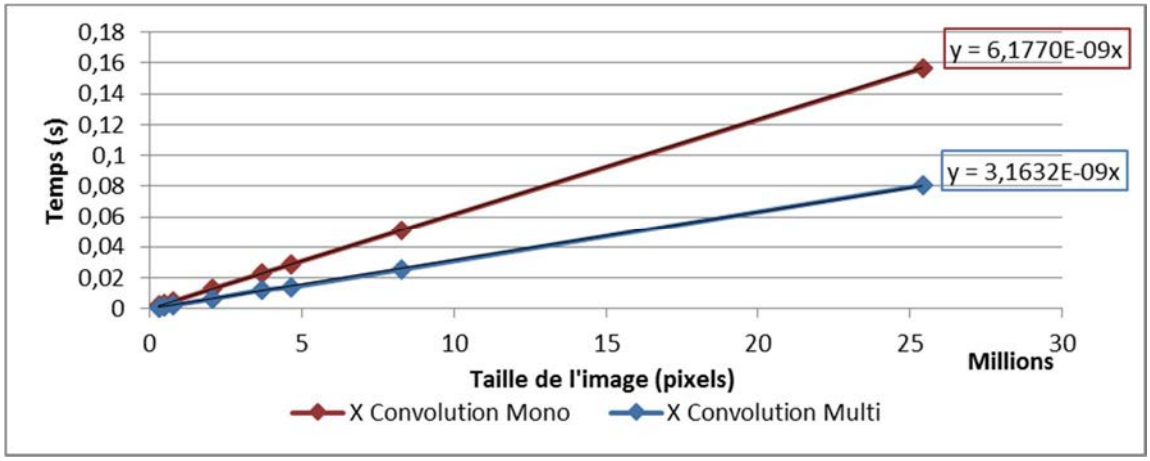


Figure 71 Résultats de la convolution en X sur Intel Core I3

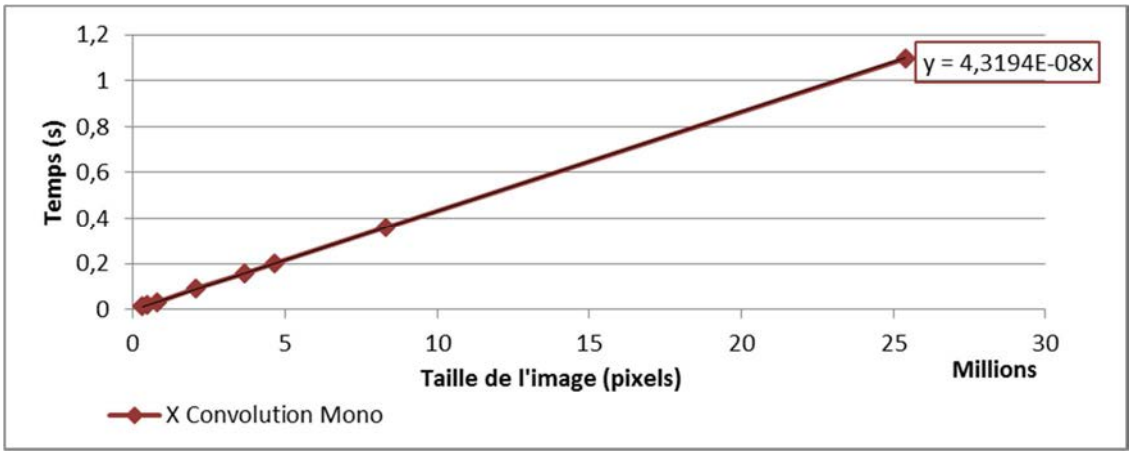


Figure 72 Résultats de la convolution en X sur Tegra K1 à 2,3 Ghz

De même que précédemment, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98. La formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = $Coef_{CoX}$:

$$\boldsymbol{Time_{CoX}(s) = Coef_{CoX} \times ImageSize} \tag{15}$$

Cette modélisation permet d'obtenir un coefficient pour chaque architecture hardware en Mono Core et en Multi Core :

Tableau 12 Coefficients (CoX) de l'équation (15) pour toutes les architectures étudiées

$Coef_{CoX}$	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	3,556	6,177	14,195	159,310	43,194	84,147
Multi Core	1,254	3,163	8,678	-	-	-

Les coefficients du Tableau 12 sont différents selon les architectures et selon le nombre de cœurs utilisés pour la résolution du problème. Les résultats obtenus semble, dans l'esprit, très proche de ce que nous avons obtenus pour la Convolution 2D. La modélisation « hardware » sera traité par la suite mais il semble que la modélisation sera très semblable à celle effectué sur la convolution 2D.

Modélisation de la convolution en Y

La Convolution en Y (filtre à une dimension de manière verticale) donne des résultats très similaires à l'étude de la convolution2D (Exemples de résultats Figure 73 et Figure 74).

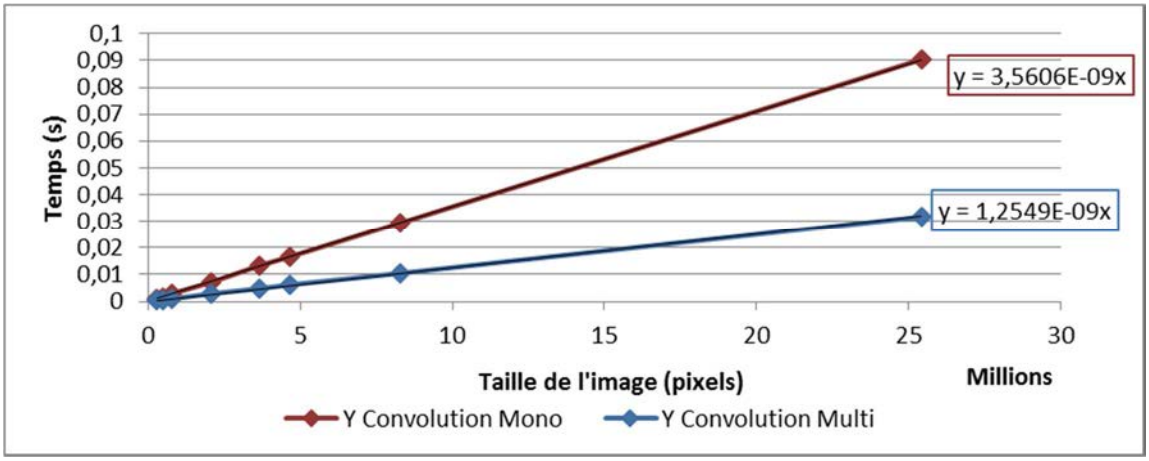


Figure 73 Résultats de la convolution en Y sur Intel Xeon

Figure 74 Résultats de la convolution en Y sur Raspberry Pi

De même que précédemment, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98. La formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = $Coef_{CoY}$:

$$\boldsymbol{Time_{CoY}(s) = Coef_{CoY} \times ImageSize} \tag{16}$$

Cette modélisation permet d'obtenir un coefficient pour chaque architecture hardware en Mono Core et en Multi Core :

Tableau 13 Coefficients (CoY) de l'équation (16) pour toutes les architectures étudiées

$Coef_{CoY}$	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	3,561	6,072	14,240	154,430	43,168	83,686
Multi Core	1,255	3,136	8,673	-	-	-

Les coefficients du Tableau 13 sont différents selon les architectures et selon le nombre de cœurs utilisés pour la résolution du problème. Les résultats obtenus sont similaires à ceux de la Convolution en Y. La modélisation « hardware » sera traitée par la suite.

3.2.3.2. Modélisation des ondelettes par les briques élémentaires et validation

Chaque brique a été modélisée de façon « software », c'est-à-dire en fonction des paramètres d'entrée de l'algorithme. Il reste donc maintenant à valider que l'algorithme des ondelettes peut être modélisé grâce aux modèles de ces briques élémentaires.

Pour cela, l'algorithme complet a été testé sur les différentes architectures hardware et a pu être modélisé par rapport à la taille de l'image traitée (Figure 40 et Figure 76).

En effet, le temps de calcul de l'algorithme des ondelettes est bien proportionnel à la taille de l'image. De même que pour les briques élémentaires, lorsque l'on applique une courbe de tendance linéaire passant par (0 ; 0) sur les différentes courbes, il ressort que les coefficients de détermination R^2 sont toujours supérieurs à 0,98.

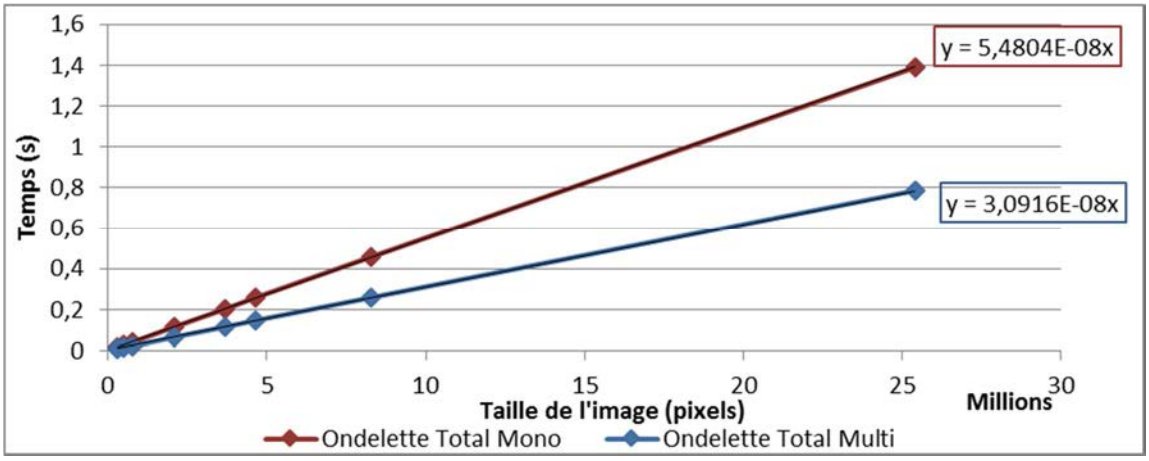


Figure 75 Résultats des ondelettes sur Intel Core I3

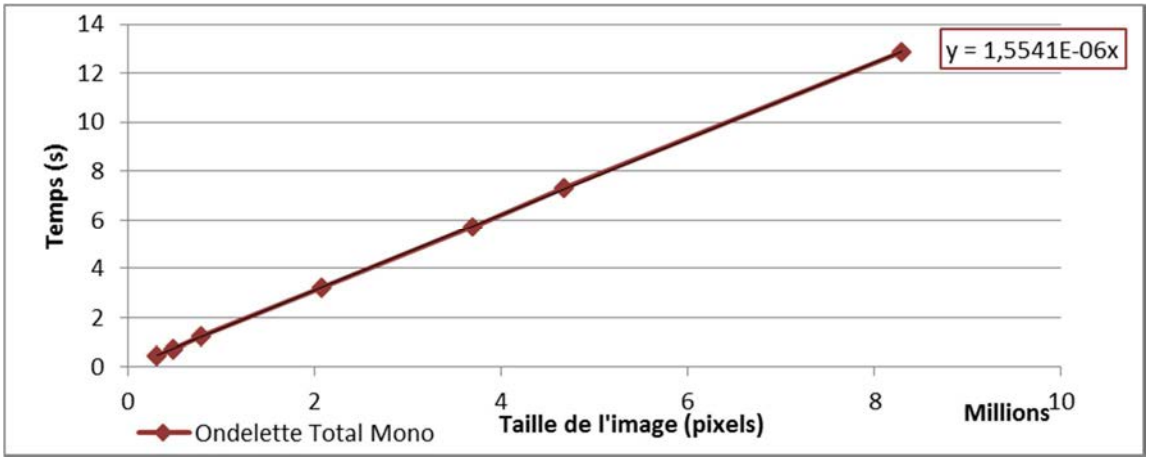


Figure 76 Résultats des ondelettes sur Raspberry Pi

La formule qui régit donc le temps en fonction de la taille de l'image est de la forme $y = a\,x$ avec a = Coef_{Ond} (coefficient différent pour chaque architecture):

$$\boldsymbol{Time_{Ond}(s) = Coef_{Ond} \times ImageSize} \tag{17}$$

Cette modélisation permet d'obtenir un coefficient pour chaque architecture hardware en Mono Core et en Multi Core dans le Tableau 14.

Tableau 14 Coefficients (Ond) de l'équation (17) pour toutes les architectures étudiées

Coef_{Ond}	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Mono Core	36,43	54,80	136,65	1554,10	403,99	787,11
Multi Core	18,97	30,92	91,80	-	-	-

La modélisation de la performance de l'algorithme des ondelettes peut être réalisée à l'aide des coefficients des différentes briques étudiées. En effet, en reprenant l'algorithme détaillé en 3.2.2.1, on obtient l'équation suivante :

$$\begin{aligned}\boldsymbol{Coef_{OndTheo}} = &\boldsymbol{\frac{5}{2}Coef_{Add} + \frac{5}{4}(Coef_{UpC} + Coef_{UpL} + Coef_{DeC} + Coef_{DeL})} \\ &\boldsymbol{+ \frac{1}{4}Coef_{Seu} + \frac{15}{4}(Coef_{CoX} + Coef_{CoY})}\end{aligned} \tag{18}$$

Avec :

$$\boldsymbol{Time_{OndTheo}(s) = Coef_{OndTheo} \times ImageSize} \tag{19}$$

Grâce à l'équation (18), on obtient les coefficients théoriques ($\text{Coef}_{\text{OndTheo}}$) qui correspondent à chaque architecture hardware étudiée, qui sont répertoriés dans les tableaux suivants :

Tableau 15 Comparaison des Coefficients (Ond) de l'équation (17) et des Coefficients (OndTheo) de l'équation (18) en Mono Core

Mono Core	XEON	Core I3	SU2300	Raspberry	Tegra K1 2,3	Tegra K1 1,2
Coef_{Ond}	36,43	54,80	136,65	1554,10	403,99	787,11
$\text{Coef}_{\text{OndTheo}}$	38,14	58,24	146,64	1535,35	409,61	796,57
Variation	+4,71%	+6,26%	+7,31%	-1,21%	+1,39%	+1,20%

Tableau 16 Comparaison des Coefficients (Ond) de l'équation (17) et des Coefficients (OndTheo) de l'équation (18) en Multi Core

Multi Core	XEON	Core I3	SU2300
$Coef_{Ond}$	18,97	30,92	91,80
$Coef_{OndTheo}$	18,09	31,08	94,64
Variation	-4,67%	+0,52%	3,10%

Le Tableau 15 et le Tableau 16 comparent les coefficients « théoriques » obtenus à l'aide de la modélisation de chacune des briques élémentaires et les coefficients expérimentaux obtenus avec les tests de l'algorithme des ondelettes complet sur les différentes cibles. On peut constater que l'on obtient des variations inférieures à 8% entre la modélisation de l'algorithme complet et la modélisation de chacun des blocs composant l'algorithme additionné. On suppose donc que si l'on peut modéliser chacun des sous blocs en fonction des spécifications de l'architecture hardware que l'on veut tester et non pas seulement en fonction des paramètres de l'algorithme (taille de l'image, taille du masque si convolution, etc.), on pourra alors modéliser l'algorithme complet avec les paramètres softwares et hardwares du système de calcul utilisé.

Chaque bloc que compose l'algorithme est à modéliser de façon différente en fonction des spécifications d'une architecture hardware : fréquence d'horloge, nombre de Cœur, performance mémoire, architecture RISC ou CISC etc.

3.2.3.3. Modélisation hardware des briques élémentaires

Il s'agit maintenant d'intégrer les paramètres hardwares dans les différentes briques élémentaires, pour voir si cette modélisation n'impacte pas trop la variation observée entre les coefficients théoriques et les coefficients expérimentaux du Tableau 15 et du Tableau 16.

Il a été défini que les paramètres hardwares à prendre en compte dans la modélisation sont, bien évidemment, la vitesse d'horloge du CPU et le nombre de processeurs sur lesquels il est possible de paralléliser les calculs, mais aussi les performances de caches mémoires (nombre, taille et vitesse d'accès) et le type d'architecture (CISC ou RISC).

La comparaison des résultats sur les différentes architectures permet de déterminer quelle est l'influence de chaque paramètre sur les performances. En effet, en fonction de l'algorithme, l'influence de certains paramètres va être plus ou moins importante.

Les résultats nous permettent d'intégrer le paramètre de la fréquence d'horloge de manière proportionnel. Le Tableau 17 permet d'observer les résultats obtenus pour chaque brique élémentaire.

Tableau 17 Prise en compte de la fréquence d'horloge pour les Coefficients de chaque brique élémentaire en Mono Core

B	XEON	Core I3	SU2300	Raspberry	Tegra K1	Tegra K1 1,2
Addition d'image	6,070	5,553	7,860	45,190	30,654	31,310
Upsampling colonne	5,898	3,352	6,452	27,074	25,049	25,782
Upsampling ligne	3,579	2,405	3,803	26,777	24,587	24,534
Decimation colonne	2,759	2,526	3,627	24,448	13,953	14,133
Decimation ligne	2,494	2,643	3,458	15,810	14,100	14,329
Seuillage	12,270	17,802	26,741	82,257	93,920	95,542
Convolution en X	11,378	16,060	17,034	111,517	99,346	100,98
Convolution en Y	11,394	15,786	17,088	108,101	99,286	100,42

Les Coefficients B du Tableau 17 peuvent être intégrés aux différentes équations (9) à (16) pour obtenir l'équation générique suivante :

$$\boldsymbol{Time_{"Algo"}(s) = \frac{B}{Frequency} \times ImageSize} \tag{20}$$

Avec :

$$\boldsymbol{B = Coef_{"Algo"} \times Frequency} \tag{21}$$

Dans les équations 18 et 19, « Algo » correspond aux noms des briques élémentaires étudiées.

Les résultats obtenus doivent ensuite être étudiés de manière indépendante pour intégrer la performance mémoire obtenue dans le Tableau 4 grâce au software développé. Comme pour l'étude de la convolution 2D, il faut trouver les coefficients C et D de l'équation (22).

$$\boldsymbol{Time(s) = \frac{\frac{C}{MemoryPerf} + D}{Frequency} \times ImageSize} \tag{22}$$

A l'aide de la méthode des moindres carrés, on obtient les coefficients C et D du Tableau 18.

Tableau 18 Prise en compte de la mémoire pour les Coefficients de chaque brique élémentaire en Mono Core

	C	D
Addition d'image	1555	2,37
Upsampling colonne	2758	4,12
Upsampling ligne	2969	2,08
Decimation colonne	2705	1,89
Decimation ligne	1632	2,21
Seuillage	8035	15,60
Convolution en X	1217	1,00
Convolution en Y	1175	1,00

On obtient donc finalement une équation (22) pour chaque brique élémentaire. Ces équations correspondent aux MBE composant l'algorithme des ondelettes.

3.3. Validation du paradigme et perspectives

Ces premiers résultats nous ont donc permis de valider qu'il est possible de définir le MBE (Modèle de Brique Elémentaire) d'un certain nombre d'algorithmes de traitement d'images très utilisés dans le domaine de la vision sur des architectures CPU. Notre modélisation se déroule en deux étapes : modélisation software et modélisation hardware.

Pour la modélisation software, nous avons montré que, pour toutes les briques élémentaires, le paramètre de la taille de l'image agissait de manière linéaire sur la performance. Toutes les courbes sont de la forme : $y = a\,x$.

De son côté, l'algorithme des ondelettes nous a permis de valider qu'il est possible de décomposer un algorithme en briques élémentaires et de prédire sa performance en additionnant les performances de chacune de ces briques.

Nous avons ensuite intégré les paramètres hardwares du CPU dans la modélisation, pour, au final, obtenir un MBE pour chacune des briques élémentaires étudiées.

Chapitre 4 Discussion & Conclusion

4.1. Résultats de l'approche et bornes de précision

Nous avons défini un premier paradigme scientifique de prédiction de la performance d'algorithmes de traitement d'images sur des architectures CPU. Ce paradigme décompose le domaine du traitement d'images en briques élémentaires. C'est l'idée directrice de notre étude : le fait de pouvoir décrire un algorithme de traitement d'images de manière macroscopique à l'aide de ces briques élémentaires pour pouvoir faire une évaluation de la performance très en amont dans le développement d'une solution de vision par ordinateur. L'objectif est aussi de s'abstraire totalement de benchmarks et d'études du code pour l'utilisateur final, comme c'est souvent le cas pour les méthodes existantes. Ce paradigme définit aussi comment modéliser ces briques élémentaires en fonction des paramètres hardwares de l'architecture utilisée et des paramètres softwares de la brique tels que la taille d'image traitée et d'autres paramètres spécifiques à l'algorithme étudié comme la taille du Kernel pour la convolution 2D. Cette modélisation a été définie comme un MBE (Modèle de Brique Elémentaire). Dans un premier temps, notre étude s'est concentrée sur les architectures CPU et nous a permis de faire ressortir un certain nombre de paramètres influençant la performance tels que le type de CPU, le nombre de core, la fréquence d'horloge ou encore la performance mémoire.

La validation expérimentale nous a permis, dans un premier temps, de valider notre paradigme sur une brique élémentaire parmi les plus utilisées en traitement d'images (la convolution 2D). Dans un second temps, à travers l'étude des ondelettes, nous avons pu valider que le principe de décomposition en briques élémentaires peut fonctionner puisque nous avons obtenu une erreur tolérable (inférieure à 8%) entre les résultats de l'algorithme complet et l'addition des résultats des briques élémentaires composant l'algorithme des ondelettes. La démonstration que ce principe fonctionnait au moins sur cet algorithme très utilisé, était nécessaire pour prouver l'hypothèse d'étude. Cela nous a aussi permis de faire ressortir le MBE (Modèle de Brique Elémentaire) des différentes briques composant les ondelettes.

Pour l'instant, le nombre de paramètres hardwares (caractéristiques d'architectures) pris en compte peut paraître restreint. Le but n'est pas de rajouter des paramètres libres ou des paramètres obtenus par benchmark, si cela ne permet pas d'affiner le modèle de manière significative. L'objectif était de rester sur un modèle qui soit simple, sans être simpliste pour autant, pour permettre de l'utiliser dans un outil d'aide au choix d'architectures, qui soit accessible et utilisable très en amont dans un projet de réalisation d'une solution de traitement d'images.

Une première version de l'outil Métis décrit dans le chapitre 2 a été réalisée. Cette première version permet de décrire un algorithme de traitement d'images à l'aide de blocs correspondant aux briques élémentaires du modèle. Il permet ensuite de lancer la simulation de l'algorithme testé pour obtenir son temps de calcul. Métis permettra de tester la modélisation mise en place auprès d'experts du domaine, pour nous permettre d'avoir un retour d'expériences et améliorer notre paradigme. Ces améliorations seront ensuite reversées dans l'outil.

4.2. Perspectives d'évolution

Les résultats expérimentaux ont permis de valider l'approche, et il sera nécessaire de compléter cette validation expérimentale. L'objectif est bien sûr de valider toutes les briques élémentaires définies pour que la modélisation soit « complète » et concrètement utilisable dans l'outil d'aide au développement. Il faudra bien évidemment valider le principe de décomposition en briques élémentaires sur d'autres algorithmes de traitement d'images « complexes » (comme nous l'avons fait sur l'algorithme des ondelettes).

Nous avons posé l'hypothèse que le set de blocs élémentaires que nous avons défini de manière heuristique (par une étude auprès d'experts du domaine) permet de modéliser n'importe quel algorithme de traitement d'images. L'objectif sera de démontrer que ce set de blocs élémentaires permet bien de définir l'ensemble des algorithmes de traitement d'images. Pour cela, il faudra soit définir précisément ce que l'on entend par algorithme de traitement d'images et que cela converge avec le set défini, soit définir à l'aide du set, un ensemble d'algorithmes, de manière à couvrir le plus possible l'ensemble des algorithmes de traitements d'images.

Par rapport aux limites de l'approche évoquée, il faudra faire évoluer le modèle destiné à l'évaluation de performance sur CPU. Mais, il faudra surtout étendre ce modèle à

d'autres architectures comme le GPU (Graphics Processing Unit) ou le FPGA (Field-programmable Gate Array). En effet, ces deux types d'architectures sont apparus dans notre étude, comme étant incontournables dans le domaine du traitement d'images.

Les architectures GPU et FPGA possèdent un certain nombre de paramètres qui influencent les performances d'un algorithme de traitement d'images. L'esprit du GPU se rapproche fortement du CPU alors que le FPGA a des paramètres très différents. En effet, sur le GPU, même si les paramètres vont agir de manière différente (une parallélisation beaucoup plus importante), ou s'il faudra prendre en compte d'autres paramètres comme les copies mémoires entre le CPU et le GPU, le modèle devrait rester très similaire. Pour les FPGA, l'étude de l'influence des paramètres hardwares sera sûrement plus complexe à entreprendre. En effet, l'optimisation sur FPGA est plus complexe car elle est propre à chaque algorithme. Une approche un peu différente devra être entreprise pour l'étude de l'addition de plusieurs briques élémentaires. En effet, sur un FPGA, il est tout à fait vraisemblable d'avoir un temps de traitement identique pour une brique élémentaire et pour l'addition de deux briques, pour peu que les deux briques élémentaires n'aient pas besoin de plus d'éléments logiques que ce qui est disponible sur le FPGA. Il y aura forcément des ajustements à faire sur le modèle, plus ou moins importants selon les architectures étudiées, mais le principe sera identique.

De son côté, l'outil Métis devra intégrer les évolutions du paradigme, mais surtout, au-delà des aspects purement IHM (Interface Homme Machine) et de la nécessité de sa simplicité d'utilisation, il pourra intégrer d'autres fonctionnalités. Certaines fonctionnalités pourront découler du paradigme mis en place, comme par exemple le fait de déterminer les points critiques d'une application ou encore le fait de proposer une architecture hardware en fonction de contraintes déterminées par l'utilisateur. Par contre, d'autres fonctionnalités comme l'extension de la notion de performance pourront avoir un impact sur le paradigme. En effet, la performance pourrait par exemple intégrer la notion de consommation énergétique. Avec l'explosion des applications de systèmes embarqués, la notion de consommation énergétique est une contrainte de plus en plus importante dans le développement d'un système de calculs dans le domaine de la vision par ordinateur.

Notre étude se restreint au domaine du traitement d'images et c'est de cette manière que nous avons pu développer notre méthode. Mais cette méthode pourrait s'appliquer à d'autres domaines que le traitement d'images. Il s'agirait alors de pourvoir définir un ensemble de blocs qui permette de caractériser ce nouveau domaine.

Finalement, cette étude aux nombreuses perspectives, est une première étape qui pose le socle d'un paradigme ambitieux, qui doit être poursuivi pour démontrer toute sa valeur. A ce titre, des travaux sont actuellement engagés sur l'amélioration du paradigme CPU et sur l'extension de celui-ci, sur GPU.

Annexes

Smart Moving Nightstand For Medical Assistance of Elderly People: an Open Project

Durant la thèse, de nombreux travaux, autres que ceux présentés dans le manuscrit, ont été menés. Ces travaux ont notamment abouti à une publication portant sur le développement d'une table de nuit intelligente d'aide aux personnes âgées.

Ce projet porté par la société Alten a pour vocation de travailler sur un robot d'assistance permettant de surveiller des personnes âgées, de les suivre d'un point vue médicale et de leur apporter des fonctionnalités multimédias. Ces travaux ont donné lieu à une publication lors de la conférence internationale HEALTH INF 2014 :

Nicolas Soucies[1,2], Jeremie Girouard[2] and Nizar Ouarti[1]
[1]*Institute des Systemes Intelligents et de Robotique, Universite Pierre et Marie Curie, 4 place jussieu, Paris, France*
[2]*Alten, 221 Boulevard Jean Jaurès, Boulogne-Billancourt, France*

{ouarti, soucies}@isir.upmc.fr, jgirouard@webmail.alten.fr

Keywords: ELDERLY PEOPLE, HEALTH, REMOTE SENSORS, ZIGBEE, ROBOT, OPEN PLATFORM

Abstract: We present an open mobile platform that aims to benefit from versatile wireless sensors. This mobile assistant is a robot that can monitor different physiologic data for elderly people. Moreover it has the ability to determine the distance and potentially the position of the elderly person who use it. As an assistant it can transport some "objects" as glasses or drugs. Preliminary results show the proof of concept of our approach with a remote sensor that measures the temperature of the subject. We also present a method to assess the quality of the RSSI signal in order to determine the distance of a zigbee module attached to the arm of the subject. These results are the first steps towards a totally autonomous system that is an open platform. In this platform it will be easy to highlight the interaction or the correlation between the different physiological data and to move the robot properly in case of alert. It is possible to program different services and to integrate new sensors remotely. This platform can be convenient for developers and researchers involved in health technology.

1 INTRODUCTION

In recent years the number of elderly people isolated at home continues to grow. For instance, isolated elderly people represent 11 percent of the population in France. These people often have need for medical monitoring and have difficulty to open outward. One of the solutions is to develop connected medical sensors to remotely monitor physiological and medical data. The variety of available physiological sensors allows to consider a large number of diseases, including non-exhaustively:

Automatic blood pressure sensors are now available which send the data, either wired or wireless to a smartphone or a computer [58].

Any apparatus that measures the concentration of sugar in the blood are using the same techniques; it is imperative to remove the patient's blood on a test strip. Some of these devices can transmit data via a USB cable or wirelessly.

There are two ways to achieve the pulse measurement, with an electrocardiogram (ECG) or an oximeter. The ECG is generally composed of a transmitter unit includes a belt worn around the chest and comprising electrodes for sensing the heart beat and to transmit information to a receiver, which can be worn the wrist like a watch. Using this system can be complicated for elderly people. The alternative is pulse oximeter which measures the quantity of oxygen in the blood at the finger. The measurement of the variation of oxygen in the blood is an indirect indicator of the pulse rate.

Thermometers are classic physiological sensors widely used at home. But there is an added value to be able to record temperature regularly for monitoring the evolution of temperature. Moreover some new thermometers were recently design that can measure the temperature with no contact. The principle is simply to record the infrared radiation from the heat source and to convert it in temperature.

The main drawback of these systems is the constraint to access the data one by one. Based on this issue, some researchers proposed the project the E-Health with the purpose to connect several physiological sensors to one platform. It consists of a "Cooking Hacks" card, which is used to interface all medical sensors, associated with either a "Arduino" card or a "Raspberry Pi" card. On this platform, it is possible to connect up to nine medical sensors. Physiological data can be sent via WiFi, Bluetooth, 3G, GPRS or ZigBee. In addition to providing a hardware architecture, the project also provides all the software part implemented in C + +. To ensure code compatibility between the two architectures, developers have chosen to use the library "ArduPi".

▪

The principal drawback of this project is the connection with specific sensors that are dedicated to the project. Moreover, even if the data can be processed remotely, the acquisition by sensors is done with wires. These connections not provide a convenient ergonomics and limit the number of possible sensors. Another limitation is the absence of procedure in the case the patient is experiencing a serious crisis not allowing him to call for help.

Moreover a disease can be detected earlier by the correlation of different physiological data. An automatic system allowing to one hand to centralize and process the data, and to another hand to move to the patient to observe his condition, would allow to send an alert to a health's professional or the family to alert them. Another possibility is that the patient can talk with his relatives and health's professional to reassure them in case of false alarm. This kind of task could usefully be done by a robotic assistant.

Today there are many robots that were developed to assist people.

Some robots are human's assistant, but are not natively equipped with physiological sensors. This is the case of JAZZ robot, whose main application is telepresence. It may, in some cases, allow a doctor to visually observe a patient without being physically present on the site. The robot Kompaï, for its part is designed to support the elderly in a home environment. This robot focuses on multimedia features and non-medical application. ASIMO is a robot equipped with technologies that give it a genuinely independent action. Its name is an acronym for Advanced Step in Innovative Mobility. Asimo is an autonomous robot capable of determining his behavior in unpredicted situation. Thus, it can coexist with humans. Autonomy allows him to decide to change his path to avoid a collision with another person. In some circumstances, his faculties are superior to those of men. For example, it is able to track multiple conversations simultaneously [59][60].

In contrast, other robots are connected to physiological sensors. This is the case of the robot RP-Vita Remote Presence is a medical robot mobile telepresence designed to be used primarily

in a hospital and communicate with medical instruments connected to it. It helps to have several medical officers in connection who may have access to all information on the equipped patients. This comprehensive platform is exclusively available to hospitals. It requires that the hospital has to be equipped with hardware that can communicate with the robot and the platform is not suitable for home use. In this category of robots one can quote HealthBots that is a project aiming to measure some physiologic data, but the drawback is that the robot uses some dedicated sensors[61][62]. Another orientation is the one taken by Robo MD which is to combine the mobility of a Nao robot with sensors networks. This approach is mainly oriented to provide an alert in case of falls situations[63].

The aim of this study is to develop a medical assistant robotics for elderly people. It seems essential for such an assistant to have physiological sensors in large numbers. We decided to connect the physiological sensors with a ZigBee connection, this strategy has many advantages. Firstly it allows to be connected wirelessly with the platform, it also multiplies the number of accessible sensors, and then it can detect diseases more accurately by the correlation of physiological signals by identifying the type of sensor. An assistant must be able to both: use multimedia resources in an emergency to communicate with the older person or to observe his condition visually. Moreover, given the constraints due to the humanoid form that greatly complicates the mechanics of a system; we opted for a more rudimentary design that can be better accepted by the elderly. This is the concept of "smart moving nightstand". This platform is developed to allow an elderly person to be autonomous while being connected to the outside world (family and doctors). It would also carry essential items such as eyeglasses or medications of any user by monitoring its essential physiological variables. It is a new open platform for developers interested to compute physiological data and offers the services of wireless robotic assistant. This open project will make available all the hardware and the software sources and could be adapted to the versatility of final usages.

2 SYSTEM OVERVIEW

The system can be divided into 6 main parts (see Figure 77: Hardware block diagram of the platform.):

- The wireless communications with Zigbee and WiFi.
- Medical sensors (scalable to the needs of the user)
- The multimedia part that plays the role of user interface.
- The processor (SOC ARM 32 bit)
- The motor control to move the robot.
- The power to the battery and charging station.

In this study we focus on the first two issues.

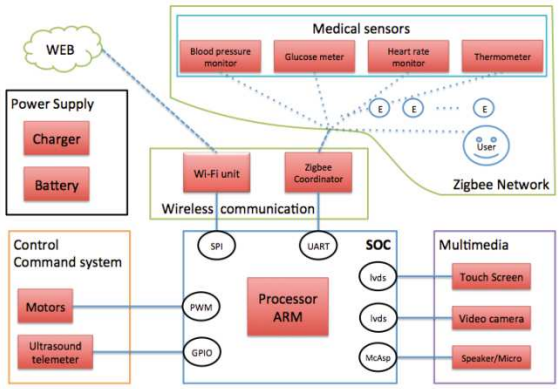


Figure 77: Hardware block diagram of the platform.

2.1 Concept

The principle behind this intelligent platform is that it can blend in with its environment. The aim is to have a "smart nightstand" that can move independently in a home environment (apartment or house). Our goal is that this assistant can navigate, locate and track the user's system when needed. This medical assistant may allow a relative or a doctor to remotely monitor the physiologic data of the user (thermometers, blood glucose meters, blood pressure, pulse oximeters, etc.). It also helps when an alert occurs to visually inspect the patient and if necessary, communicate with him. The raw data are sent wirelessly to the robot and processed on-board; some relevant information can also be store on it. This platform should also allow the user to access a number of multimedia features such as video conferencing, play multimedia content (movies, music, etc..). The technological system choices were made with the idea that this platform must be accessible to the greatest number of elderly people.

A first prototype has been developed which embodies the concept discussed. We present here the different technologies and techniques to meet the expected functionality.

2.2 Zigbee for localization and sensors communication

2.2.1 Zigbee

The Zigbee technology, based on the 802.15.4 standard and it works on the same frequency band as WiFi, 2.4GHz. This technology has the advantage of having very low power consumption, one have also the opportunity to significantly expand a mesh network (65,000 end-devices). It therefore provides parallel information from many sensors. Additionally, this technology can be used in domotic, allowing the platform to control compatible equipment. Indeed, one of the advantages of ZigBee is its interoperability. The ZigBee Alliance [64] has created this standard to create a consistent communication between multiple devices. This standard is actually a layer (ZigBee Pro Stack 2007 [65]) which is placed on top of the 802.15.4 MAC layer handles addresses. It allows to manage the network (topology, security, communication, etc). The Zigbee standard formats the messages sent between the devices. Each device is classified into categories and sub-category (Cluster) and is able according to the categories on which it depends to send or receive specific messages. All these categories are stored in different standardized norms. For our part, we will focus especially on the "Home Automation" standard and the XBEE hardware that is constituted of many analogic channels. The standard home automation can control lights, heating, electrical outlets, smoke detectors or can provide alarms. The XBEE hardware is interesting to collect analogic data from different sensors, in our case medical sensors.

2.2.2 Zigbee for localization

The localization of the platform in its environment is achieved by Zigbee. For Zigbee network, XBee Pro module was used as a system coordinator. Then, effective way to equip an apartment with Zigbee, would be to place an electrical outlet with Zigbee on each power socket. This will allow it to have a large mesh network covering the whole apartment and allow it to achieve an effective localization. To allow a Zigbee module to be wear by the user, the Zigbee medical solution ZCare of CLEODE was chosen. This sensor allows can monitor: pulse, possible falls of the person, and includes a button to call emergency. Here we can see the interest of Zigbee with the interoperability of a system developed by third parties. This solution allows to locate the person and to perform medical monitoring at the same time.

The localization of a ZigBee module for its part can be determined based on the signal power (RSSI: Receive Signal Strength Indication) sends to other modules.
The signal strength varies with the distance [66]:

$$P_{RX} = P_{TX} \cdot G_{TX} \cdot G_{RX} \left(\frac{\lambda}{4\pi d} \right)^2 \quad (1)$$

$$RSSI = 10 \cdot \log \frac{P_{RX}}{P_{Ref}} \quad [RSSI] = dBm \quad (2)$$

With: P_{TX} = Transmission power of sender,
P_{RX} = Remaining power of wave at receiver,
G_{TX} = gain of transmitter,
G_{RX} = Gain of receiver,
λ = wave length,
d = Distance betweens sender and receiver,
P_{REF} = power reference (Typically 1mW)
RSSI in dBm.

The RSSI values range between -45 and -100 dBm and therefore it is possible (for a signal on 100m) to trace the evolution of the theoretical RSSI function of distance[67].

A number of researches have been done on the indoor localization Zigbee [68][69]. Based on triangulation algorithms it seems conceivable given the location of a Zigbee module in a room provided by Zigbee module with an error margin of 2m.

In this project, the ZigBee can be used in order to know in what room of the apartment is the user through a Zigbee bracelet, and know where the platform is. It would be useful to determine how accurately our platform could determine the position of the user in the room with the RSSI signal to come and watch, and whether this accuracy is maintained in outdoor conditions.

2.2.3 Zigbee for sensors communication

Communication with XBee modules on our platform is done by receiving messages with UART written in hexadecimal. The X-CTU software is used to configure the XBee card. There are two possible modes, the transparent mode and API mode. The API mode is more indicated for a network or identification of multiple devices.
In this project we choose to apply the API mode that will help for the computation of different physiological signal by identifying which sensor

sends the data. Moreover another type of application could be the localisation of a lost sensor which can be an interesting issue with elderly people.

2 PRELIMINARY RESULTS

Some preliminary results were obtained with our first prototype to show the potential of our approach. In this section we show the potential of our method that can either be a tool for localization and for sending data.

2.1 Distance estimation with RSSI

It is possible to estimate the distance between two ZigBee modules with the help of the RSSI signal (Received Signal Strength Indication). A relation exist between the value of the RSSI and the distance.
In the first experiment, we decided to estimate this relation with our specific hardware. We decide to estimate the distance between the ZigBee module and our robotic platform.

2.2.1 Static determination of distance (indoor)

As state in the introduction, RSSI signal is perturbed by noise but the RSSI signal is more clear and discriminant at short distance (<1.5m).
We realised the following measurement at different distances: 5, 10, 20, 30, 40, 50, 70, 100, 130, 160, 200, 250 and 300 cm. These measures were repeated 10 times at each position to obtain statistically exploitable data. The mean value obtained is represented at Figure 2. This study was conducted in an office room which can be likened to a domestic environment. It can be observed that the RSSI curve is approximately bijective, meaning that the distance can be evaluated. But there is an exception at 160 cm that can be explained by occlusion and reflection of the ZigBee wave due to the objects included in the room.

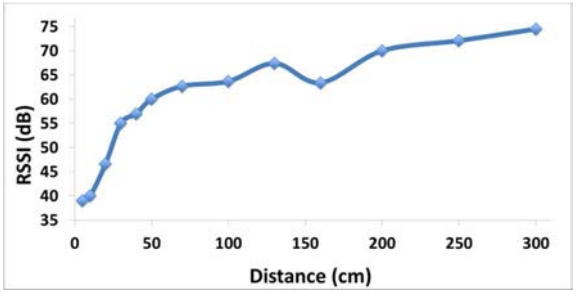


Figure 78: Static recording of RSSI (indoor)

2.2.1 Dynamic determination of distance (outdoor)

In this section we tested with two repetitions the RSSI value in indoor condition when the mobile platform was moving to the target (i.e. Zigbee module) to find whether the motion can lower the accuracy of RSSI signal. We propose a new manner to assess the quality of the obtained points by computing the coefficient of determination (R^2) between these points and a logarithmic curve that fits the points. If the points follow a bijective and logarithmic function as expected ideally, the R^2 will be close to 1. If the R^2 is further to 1 it means that the quality is low and certainly the RSSI signal is perturbed by occlusion, reflection or low intensity signal. In this experiment the R^2 is equal to 0.92 (see Figure 3) that is compared to 0.96 in the static condition.

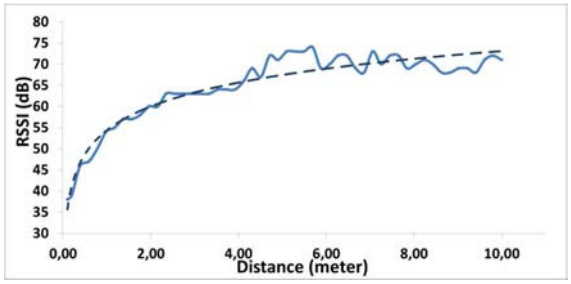


Figure 79: Dynamic recording of RSSI (outdoor)

This result has to be compared to the RSSI data obtained in indoor in dynamic condition. We obtained a R^2 of 0.87 (see Figure 4). We can observed that in this case the curve is much noisy compared to outdoor condition and also compared to static condition where 10 samples were recorded.

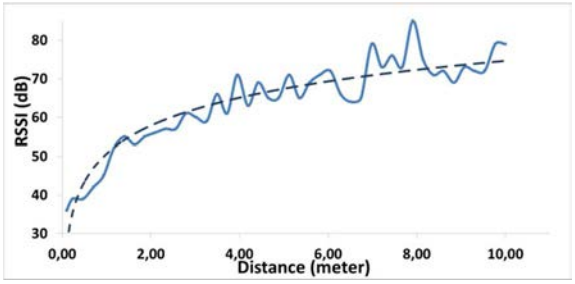


Figure 80: Dynamic recording of RSSI (indoor)

2.2 Send data from a sensor

The XBEE technology can be connected to a shield that sends data to a UART port. We developed a software that reads the data from this UART and record it on our platform. As explained before the user holds a sensor using the home automation protocol. Moreover we send wirelessly the data of temperature during a moment when the user grasps the thermometer. This thermometer is a device that we developed for the experiment (see Figure 5).

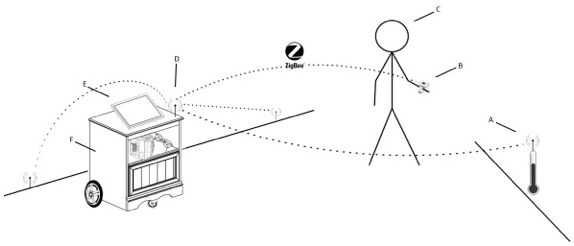


Figure 81 : Communication between our robot and the user. A : Zigbee Thermometer, B : ZCare device, C : User, D : Zigbee Coordinator, E : Touch screen and F : Robot.

The temperature of the body where captured (hand temperature) and sent remotely to the robot (see Figure 6). One can notice the ambient temperature around 24° C, the progressive increase around 27°C, the temperature of the hand during the grasping and a progressive return to the former value of ambient temperature.

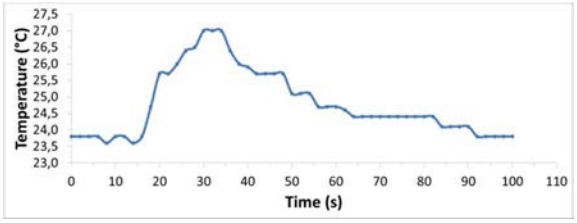


Figure 82: Temperature measurement with ZigBee transmission

3 DISCUSSION AND FUTURE WORK

We propose an open platform allowing to access the robot at low level. We demonstrated a first proof of concept by sending remotely temperature data. We also presented a method to assess the quality of the RSSI signal. And we showed that outdoor dynamic measurement is more reliable than indoor measurement, probably due to occlusion and reflection of the wave signal. The determination of the distance between the robot and a wireless sensor is essential because this can lead to the localization of the user that wearing the sensor. Another important issue is the localization of a lost sensor which can be solved with the same method. Our ongoing researches are focused on different type of sensors. The development of specific algorithms based on the different sensors is the challenge that we want to promote with the help of the community of developers and researchers involved in health technology. Depending of a known pathology, it could be advantageous to monitor the data all the time or occasionally, depending of the situation of the person, it would be more appropriate to contact the relatives or a medical center. All these scenarios have to be programmed, and the need for an intuitive setting of the robot is crucial in order to have an effective benefit.

Bibliographie

[1] L. Kronsjö, *Algorithms: their complexity and efficiency (2nd ed.)*. New York, NY, USA: John Wiley & Sons, Inc., 1987.

[2] G. M. Amdahl, “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities,” in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, New York, NY, USA, 1967, pp. 483–485.

[3] J. L. Gustafson, “Reevaluating Amdahl’s Law,” *Commun ACM*, vol. 31, no. 5, pp. 532–533, May 1988.

[4] M. D. Hill and M. R. Marty, “Amdahl’s Law in the Multicore Era,” *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[5] R. M. Karp, “A Survey of Parallel Algorithms for Shared-Memory Machines,” University of California at Berkeley, Berkeley, CA, USA, 1988.

[6] B. Cope, P. Y. K. Cheung, W. Luk, and L. Howes, “Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study,” *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 433 –448, Apr. 2010.

[7] C. Grozea, Z. Bankovic, and P. Laskov, “FPGA vs. Multi-core CPUs vs. GPUs: Hands-On Experience with a Sorting Application,” in *Facing the Multicore-Challenge*, vol. 6310, R. Keller, D. Kramer, and J.-P. Weiss, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 105–117.

[8] R. Kalarot and J. Morris, “Comparison of FPGA and GPU implementations of real-time stereo vision,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2010, pp. 9 –15.

[9] T. R. Savarimuthu, A. Kjær-Nielsen, and A. S. Sørensen, “Real-time medical video processing, enabled by hardware accelerated correlations,” *J. Real-Time Image Process.*, vol. 6, no. 3, pp. 187–197, Dec. 2010.

[10] J. Fowers, G. Brown, P. Cooke, and G. Stitt, “A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications,” in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, New York, NY, USA, 2012, pp. 47–56.

[11] S. Asano, T. Maruyama, and Y. Yamaguchi, “Performance comparison of FPGA, GPU and CPU in image processing,” in *International Conference on Field Programmable Logic and Applications, 2009. FPL 2009*, 2009, pp. 126 –131.

[12] B. Cope, P. Y. K. Cheung, W. Luk, and S. Witt, “Have GPUs made FPGAs redundant in the field of video processing?,” in *2005 IEEE International Conference on Field-Programmable Technology, 2005. Proceedings*, 2005, pp. 111–118.

[13] J. Chase, B. Nelson, J. Bodily, Z. Wei, and D.-J. Lee, “Real-Time Optical Flow Calculations on FPGA and GPU Architectures: A Comparison Study,” in *16th International Symposium on Field-Programmable Custom Computing Machines, 2008. FCCM ’08*, 2008, pp. 173 –182.

[14] P. Heidelberger and S. S. Lavenberg, “Computer Performance Evaluation Methodology,” *IEEE Trans. Comput.*, vol. C–33, no. 12, pp. 1195–1220, Dec. 1984.

[15] S. Balsamo, A. di Marco, P. Inverardi, and M. Simeoni, “Model-based performance prediction in software development: a survey,” *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 295–310, May 2004.

[16] L. G. Williams and C. U. Smith, “Performance Evaluation of Software Architectures,” in *Proceedings of the 1st International Workshop on Software and Performance*, New York, NY, USA, 1998, pp. 164–177.

[17] C. U. Smith and L. G. Williams, “Performance engineering evaluation of object-oriented systems with SPE·ED TM,” in *Computer Performance Evaluation Modelling Techniques and Tools*, R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, Eds. Springer Berlin Heidelberg, 1997, pp. 135–154.

[18] V. Cortellessa, G. Lazeolla, and R. Mirandola, “Early generation of performance models for object-oriented systems,” *Softw. IEE Proc. -*, vol. 147, no. 3, pp. 61–72, Jun. 2000.

[19] V. Cortellessa and R. Mirandola, “Deriving a Queueing Network Based Performance Model from UML Diagrams,” in *Proceedings of the 2Nd International Workshop on Software and Performance*, New York, NY, USA, 2000, pp. 58–70.

[20] V. Grassi and R. Mirandola, “PRIMAmob-UML: A Methodology for Performance Analysis of Mobile Software Architectures,” in *Proceedings of the 3rd International Workshop on Software and Performance*, New York, NY, USA, 2002, pp. 262–274.

[21] V. Cortellessa, A. D’Ambrogio, and G. Iazeolla, “Automatic derivation of software performance models from CASE documents,” *Perform. Eval.*, vol. 45, no. 2–3, pp. 81–105, Jul. 2001.

[22] D. C. Petriu and X. Wang, “From UML Descriptions of High-Level Software Architectures to LQN Performance Models,” in *Applications of Graph Transformations with Industrial Relevance*, M. Nagl, A. Schürr, and M. Münch, Eds. Springer Berlin Heidelberg, 2000, pp. 47–63.

[23] D. A. Menasce and H. Gomaa, “A method for design and performance modeling of client/server systems,” *IEEE Trans. Softw. Eng.*, vol. 26, no. 11, pp. 1066–1085, Nov. 2000.

[24] F. Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi, “Deriving Performance Models of Software Architectures from Message Sequence Charts,” in *Proceedings of the 2Nd International Workshop on Software and Performance*, New York, NY, USA, 2000, pp. 47–57.

[25] F. Aquilani, S. Balsamo, and P. Inverardi, “Performance analysis at the software architectural design level,” *Perform. Eval.*, vol. 45, no. 2–3, pp. 147–178, Jul. 2001.

[26] M. Woodside, C. Hrischuk, B. Selic, and S. Bayarov, “Automated performance modeling of software generated by a design environment,” *Perform. Eval.*, vol. 45, no. 2–3, pp. 107–123, Jul. 2001.

[27] D. C. Petriu and M. Woodside, “Software Performance Models from System Scenarios in Use Case Maps,” in *Computer Performance Evaluation: Modelling Techniques and Tools*, T. Field, P. G. Harrison, J. Bradley, and U. Harder, Eds. Springer Berlin Heidelberg, 2002, pp. 141–158.

[28] P. Kähkipuro, “UML-Based Performance Modeling Framework for Component-Based Distributed Systems,” in *Performance Engineering*, R. Dumke, C. Rautenstrauch, A. Scholz, and A. Schmietendorf, Eds. Springer Berlin Heidelberg, 2001, pp. 167–184.

[29] H. Hermanns, U. Herzog, and J.-P. Katoen, “Process algebra for performance evaluation,” *Theor. Comput. Sci.*, vol. 274, no. 1–2, pp. 43–87, Mar. 2002.

[30] P. King and R. Pooley, “Derivation of Petri Net Performance Models from UML Specifications of Communications Software,” in *Computer Performance Evaluation.Modelling Techniques and Tools*, B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith, Eds. Springer Berlin Heidelberg, 2000, pp. 262–276.

[31] S. Bernardi, S. Donatelli, and J. Merseguer, “From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models,” in *Proceedings of the 3rd International Workshop on Software and Performance*, New York, NY, USA, 2002, pp. 35–45.

[32] M. de Miguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, and S. Piekarec, “UML Extensions for the Specification and Evaluation of Latency Constraints in Architectural Models,” in *Proceedings of the 2Nd International Workshop on Software and Performance*, New York, NY, USA, 2000, pp. 83–88.

[33] L. B. Arief and N. A. Speirs, “A UML Tool for an Automatic Generation of Simulation Programs,” in *Proceedings of the 2Nd International Workshop on Software and Performance*, New York, NY, USA, 2000, pp. 71–76.

[34] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. P. Waldhorst, “Performance Analysis of Time-enhanced UML Diagrams Based on Stochastic Processes,” in *Proceedings of the 3rd International Workshop on Software and Performance*, New York, NY, USA, 2002, pp. 25–34.

[35] E.-M. Ould-Ahmed-Vall, J. Woodlee, C. Yount, K. A. Doshi, and S. Abraham, “Using Model Trees for Computer Architecture Performance Analysis of Software Applications,” in *IEEE International Symposium on Performance Analysis of Systems Software, 2007. ISPASS 2007*, 2007, pp. 116–125.

[36] R. Hundt, “HP Caliper: a framework for performance analysis tools,” *IEEE Concurr.*, vol. 8, no. 4, pp. 64–71, Oct. 2000.

[37] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, “A Portable Programming Interface for Performance Evaluation on Modern Processors,” *Int. J. High Perform. Comput. Appl.*, vol. 14, no. 3, pp. 189–204, Aug. 2000.

[38] Y. Kwon, S. Lee, H. Yi, D. Kwon, S. Yang, B.-G. Chun, L. Huang, P. Maniatis, M. Naik, and Y. Paek, “Mantis: Automatic Performance Prediction for Smartphone Applications,” in *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, Berkeley, CA, USA, 2013, pp. 297–308.

[39] F. Tip, “A Survey of Program Slicing Techniques.,” CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands, 1994.

[40] G. Bontempi and W. Kruijtzer, “A Data Analysis Method for Software Performance Prediction,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, Washington, DC, USA, 2002, p. 971–.

[41] R. Srinivasan, J. Cook, and O. Lubeck, “Ultra-Fast CPU Performance Prediction: Extending the Monte Carlo Approach,” in *18TH International Symposium on Computer Architecture and High Performance Computing, 2006. SBAC-PAD ’06*, 2006, pp. 107–116.

[42] R. Srinivasan, J. Cook, and O. Lubeck, “Performance modeling using Monte Carlo simulation,” *Comput. Archit. Lett.*, vol. 5, no. 1, pp. 38–41, Jan. 2006.

[43] G. Marin and J. Mellor-Crummey, “Cross-architecture Performance Predictions for Scientific Applications Using Parameterized Models,” in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, 2004, pp. 2–13.

[44] J. R. Hammond, S. Krishnamoorthy, S. Shende, N. A. Romero, and A. D. Malony, “Performance characterization of global address space applications: a case study with NWChem,” *Concurr. Comput. Pract. Exp.*, vol. 24, no. 2, pp. 135–154, 2012.

[45] Y.-T. S. Li, S. Malik, and A. Wolfe, “Performance Estimation of Embedded Software with Instruction Cache Modeling,” *ACM Trans Autom Electron Syst*, vol. 4, no. 3, pp. 257–279, Jul. 1999.

[46] A. J. C. van Gemund, “Performance Prediction of Parallel Processing Systems: The PAMELA Methodology,” in *Proceedings of the 7th International Conference on Supercomputing*, New York, NY, USA, 1993, pp. 318–327.

[47] A. J. C. van Gemund, *Performance Modeling with PAMELA: An Introduction*. 1992.

[48] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, “Pace—A Toolset for the Performance Prediction of Parallel and Distributed Systems,” *Int. J. High Perform. Comput. Appl.*, vol. 14, no. 3, pp. 228–251, Aug. 2000.

[49] S. A. Jarvis, D. P. Spooner, H. N. Lim Choi Keung, J. Cao, S. Saini, and G. R. Nudd, “Performance prediction and its use in parallel and distributed computing systems,” *Future Gener. Comput. Syst.*, vol. 22, no. 7, pp. 745–754, Aug. 2006.

[50] J. Cao, D. K. Kerbyson, E. Papaefstathiou, and G. R. Nudd, “Performance modeling of parallel and distributed computing using PACE,” in *Performance, Computing, and Communications Conference, 2000. IPCCC ’00. Conference Proceeding of the IEEE International*, 2000, pp. 485–492.

[51] D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini, and G. R. Nudd, “Local grid scheduling techniques using performance prediction,” *IEE Proc. - Comput. Digit. Tech.*, vol. 150, no. 2, p. 87, 2003.

[52] M. J. Litzkow, M. Livny, and M. W. Mutka, “Condor-a hunter of idle workstations,” in *, 8th International Conference on Distributed Computing Systems, 1988*, 1988, pp. 104–111.

[53] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, “An Approach to Performance Prediction for Parallel Applications,” in *Euro-Par 2005 Parallel Processing*, J. C. Cunha and P. D. Medeiros, Eds. Springer Berlin Heidelberg, 2005, pp. 196–205.

[54] H. Wabnig and G. Haring, “Performance Prediction of Parallel Systems with Scalable Specifications&Mdash;Methodology and Case Study,” *SIGMETRICS Perform Eval Rev*, vol. 22, no. 2–4, pp. 46–62, Apr. 1995.

[55] “Book Review: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling by Raj Jain (John Wiley & Sons 1991),” *SIGMETRICS Perform Eval Rev*, vol. 19, no. 2, pp. 5–11, Sep. 1991.

[56] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2006.

[57] N. Anheier, C. Bassow, J. Culver, and S. Emery, “CPU WORLD.” [Online]. Available: http://www.cpu-world.com/index.html.

[58] R. Isais, K. Nguyen, G. Perez, R. Rubio, and H. Nazeran, “A low-cost microcontroller-based wireless ECG-blood pressure telemonitor for home care,” in *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2003*, 2003, vol. 4, pp. 3157–3160 Vol.4.

[59] B. Mutlu, S. Osman, J. Forlizzi, J. Hodgins, and S. Kiesler, “Perceptions of ASIMO: an exploration on co-operation and competition with humans and humanoid robots,” in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, New York, NY, USA, 2006, pp. 351–352.

[60] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura, “The intelligent ASIMO: system overview and integration,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*, 2002, vol. 3, pp. 2478–2483 vol.3.

[61] C. Jayawardena, I. H. Kuo, U. Unger, A. Igic, R. Wong, C. I. Watson, R. Q. Stafford, E. Broadbent, P. Tiwari, J. Warren, J. Sohn, and B. A. MacDonald, “Deployment of a service robot to help older people,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 5990–5995.

[62] C. Jayawardena, I. Kuo, C. Datta, R. Q. Stafford, E. Broadbent, and B. A. MacDonald, “Design, implementation and field tests of a socially assistive robot for the elderly: HealthBot version 2,” in *2012 4th IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 2012, pp. 1837–1842.

[63] A. A. J. van de Ven, A. A. G. Sponselee, and B. A. M. Schouten, “Robo M.D.: a home care robot for monitoring and detection of critical situations,” in *Proceedings of the 28th Annual European Conference on Cognitive Ergonomics*, New York, NY, USA, 2010, pp. 375–376.

[64] A. Zigbee, “ZIGBEE ALLIANCE.” [Online]. Available: www.zigbee.org.

[65] Zigbee Alliance, “ZigBee Specification - Document 053474r17.” Jan-2008.
[66] J. Blumenthal, R. Grossmann, F. Golatowski, and D. Timmermann, “Weighted Centroid Localization in Zigbee-based Sensor Networks,” 2007, pp. 1–6.
[67] M. Sugano, T. Kawazoe, Y. Ohta, and M. Murata, “Indoor Localization System using RSSI Measurement of Wireless Sensor Network based on ZigBee Standard,” presented at the Wireless Sensor Networks.
[68] S.-Y. Lau, T.-H. Lin, T.-Y. Huang, I.-H. Ng, and P. Huang, “A measurement study of zigbee-based indoor localization systems under RF interference,” in *Proceedings of the 4th ACM international workshop on Experimental evaluation and characterization*, New York, NY, USA, 2009, pp. 35–42.
[69] F. Thomas and L. Ros, “Revisiting trilateration for robot localization,” *IEEE Trans. Robot.*, vol. 21, no. 1, pp. 93–101, 2005.

Liste des publications

Publiées :

- N. SOUCIES, J. GIROUARD and N. OUARTI, (2014). Smart Moving Nightstand, For Medical Assistance of Elderly People: an Open Project. In Proc. 7th International Conference on Health Informatics: HEALTHINF 2014.

- N. SOUCIES and N. OUARTI. (2014). Performance prediction of visual algorithms on different hardware architectures. International Symposium on Optomechatronic Technologies : ISOT 2014.

En cours :

- N. SOUCIES and N. OUARTI. (2015). Performance prediction of visual algorithms on different hardware architectures (Convolution2d). En cours de soumission au Journal.

- N. SOUCIES and N. OUARTI. (2015). Performance prediction of complex visual algorithms on different hardware architectures. En cours de soumission au Journal.