République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

Université Colonel Hadj-Lakhdar Batna
Faculté des sciences de l'ingénieur
Département d'Informatique

Mémoire

En vue de l'obtention du diplôme de

MAGISTERE

Option

Informatique Industrielle

Optimisation des Ressources dans les Systèmes Embarqués

Présenté et soutenu par

Belkhiri Hadda

Date de soutenance : *13/10/2010*

Composition du jury :

1. ***Dr. A. Bilami (Université de Batna)*** .... ***Président du Jury***
2. ***Pr. M. Benmohammed (Université de Constantine)*** .... ***Rapporteur***
3. ***Dr. B. Belattar (Université de Batna)*** .... ***Examinateur***
4. ***Dr. S. Merniz (Université de Constantine)*** .... ***Examinateur***
5. ***Dr A. Chaoui (Université de Constantine)*** .... ***Examinateur***

Année universitaire : 2010-2011

Remerciements

Avant tout, je remercie dieu le tout puissant pour la force, la volonté et la patience qu'il m'a donné pour réaliser ce travail.

J'exprime mon extrême gratitude à mon encadreur Pr. Benmouhamed Mohamed pour son aide, sa compréhension, sa patience et ses judicieux conseils tout au long de ce travail.

Je tiens à remercier Dr. A. Bilami, maître de conférences à l'université de Batna pour l'honneur qu'il me fait en acceptant d'être le président de Jury.

J'adresse également mes remerciements à messieurs : Dr. B. Belattar, Dr. S. Merniz et Dr. A. Chaoui d'avoir accepté d'être présents dans le jury.

Mes remerciements s'adressent également à Mer. Nekkache Mabrouk de l'université de Sétif pour son aide appréciable lors de la configuration des différents simulateurs sous Linux.

Enfin, je remercie mes chers parents et mon mari Yacine pour leurs encouragements, soutien et aide tout au long de ce travail.

Mme Belkhiri Hadda

Dédicace

Je dédie ce modeste travail à :

- A l'âme de ma chère et regrettée fille **Maria Malak** que le bon dieu a choisi pour être avec les anges du paradis.
- Mes chers et adorables parents (que dieu les garde inchallah)
- Mon mari Yacine
- Mes chers frères et chères sœurs
- Mes beaux frères et belles sœurs
- A tous mes neveux et nièces
- A toutes mes amies du CFPA Batna3 et plus précisément à : Naziha Sahraoui, Sihem bendris, Samira Bahaz, Samia Aitouche, Amina Boulekfouf , Habiba Bengouga, Zidani Ghania et Amel Yahia Bey.

Mme Belkhiri Hadda

TABLE DES MATIERES

Abstract iv

Liste des figures v

Liste des tableaux viii

INTRODUCTION GENERALE 1

CHAPITRE I

INTRODUCTION AUX SYSTEMES EMBARQUES

Introduction 5

I.1. Structure d'un système embarqué 6

I.2. Caractéristiques d'un système embarqué 7

I.3. Méthodologies de conception des systèmes embarqués 9

I.3.1. Approches de conception des systèmes embarqués 10

I.3.1.1. Approche conventionnelle du Codesign 10

I.3.1.2. Approche basée sur Modèle 13

I.3.2. Démarches de conception 15

I.3.2.1. La conception ascendante (bottom up) 15

I.3.2.2. La conception descendante (Top down) 16

I.4. Synthèse des circuits 16

I.4.1. Niveaux d'abstraction 16

I.4.2. Les différents niveaux de synthèse 17

CHAPITRE II

LES MEMOIRES DANS LES SYSTEMES EMBARQUES

Introduction 21

II.1. Classification fonctionnelle des mémoires 21

II.1.1. Les mémoires mortes 21

II.1.2. Les Mémoires vives 22

II.1.3. La mémoire Flash 24

II.2. Les nouvelles alternatives des RAM 24

II.3. La hiérarchie mémoire 26
II.3.1. Approches de conception d'une hiérarchie mémoire 27
II.3.1.1. Mémoire cache gérée de façon matérielle 28
II.3.1.2. Mémoire cache gérée de façon logicielle 33
II.3.1.3. Création d'une hiérarchie mémoire dédiée 35
II.3.2. Mesures de performances du cache 36
II.3.3. Interaction et influence des différents paramètres 37
II.4. Techniques d'optimisation du cache dans les systèmes embarqués 38
II.4.1. Augmentation de la capacité du cache et la taille de la ligne du cache 40
II.4.2. Augmentation de l'associativité 40
II.4.3. Le cache victime (victim cache) 41
II.4.4. Cache partitionné 42
II.4.5. La mémoire scratch pad 42
II.4.6. Le préchargement 44
II.4.7. Cache multi niveaux 45
II.5. Optimisation de la mémoire et placement dans le flot de conception 45
II.6. Stratégie globale de la synthèse de la mémoire 46
II.6.1. Choix d'une hiérarchie mémoire 48
II.6.2. Distribution des structures de données 49
II.6.3. Placement des structures de données 49
II.6.4. Ordonnancement sous contraintes mémoires 49
II.6.5. Génération des adresses 50
II.7. Applications cibles 50

CHAPITRE III

APPROCHES DU PARTITIONNEMENT DU DATA CACHE

Introduction 53
III.1. Architectures du cache partitionné 54
III.1.1. Partitionnement du cache selon la localité spatiale 54
III.1.1.1. Split Temporal/Spatial Cache (STS Cache) 54
III.1.1.2. Dual Data Cache 54
III.1.1.3. Selective and dual data cache 55
III.1.1.4. Array Cache 55

III.1.2. Partitionnement du cache selon la localité temporelle ... 55
III.1.2.1. Assist cache ... 55
III.1.2.2. Non Temporal Streaming Cache (NTS cache) ... 56
III.2. Partitionnement du Data cache pour les applications multimédia embarquées ... 56
III.3. Méthodes d'analyse du cache ... 58
III.3.1. Hardware Monitoring ... 58
III.3.2. La modélisation analytique ... 58
III.3. 3. Simulation ... 59
III.3. 3.1. La simulation dirigée par trace ... 59
III.3. 3.2. La simulation dirigée par exécution ... 63

CHAPITRE IV

ETUDE EXPERIMENTALE

Introduction ... 65
IV.1. Choix du simulateur ... 65
IV.2. Paramètres du Cache ... 66
IV.2.1. Capacité du cache ... 66
IV.2.2. Taille de bloc ... 68
IV.2.3. Associativité ... 69
IV.3. Mesures de performance ... 70
IV.3.1. Taux des défauts du cache ... 70
IV.3.2. Temps d'accès au cache ... 70
IV.3.3. Surface du cache ... 71
VI.3.4. Consommation d'énergie ... 71
IV.4. Les benchmarks ... 71
IV.5. Résultats expérimentaux d'un cache unifié (scalaire et tableau) ... 73
IV.5.1. Taux de défauts du cache ... 73
IV.5.2. Temps d'accès au cache ... 80
IV.5.3. Surface du cache ... 83
IV.5.4. Energie consommée par accès ... 84
IV.5.5. Calcul de l'énergie totale consommée par les benchmarks ... 86
IV.5.6. Interaction entre les paramètres: énergie, espace et temps d'accès au cache ... 89

IV.6. Résultats expérimentaux d'un cache partitionné (Scalar cache et Array cache) 91

IV.6.1. Paramètres du cache partitionné 91

IV.6.2.Mesures de performance 92

IV.6.3.Résultats expérimentaux du cache partitionné...93

IV.6.3.1. Cache des scalaires (Scalar Cache) 93

IV.6.3.2. Cache des Tableaux (Array Cache) 97

IV.6.3.3. Comparaison entre le cache partitionné (Array Cache & Scalar Cache) et le cache unifié 101

CONCLUSION 103

BIBLIOGRAPHIE

ANNEXES

ABSTRACT

Recent years have witnessed the emergence of microprocessors that are embedded within a plethora of devices used in everyday life. Embedded architectures are customized through a meticulous criteria and time consuming design process to satisfy stringent constraints with respect to performance, area, power, and cost. In embedded systems, the cost of the memory hierarchy limits its ability to play as central a role. This is due to stringent constraints that fundamentally limit the physical size and complexity of the memory system.

Embedded systems are increasingly using on-chip caches as part of their on-chip memory system. The existing cache organization suffers from the inability to distinguish different types of localities. This causes unnecessary movement of data among the levels of the memory hierarchy and increases in miss ratio. In our work we propose split data cache architecture for the embedded multimedia applications that will group memory accesses as scalar or array references according to their inherent locality and will subsequently map each group to a dedicated cache partition.

Keywords: Embedded systems, memory hierarchy, Cache misses, Low power design.

LISTE DES FIGURES

Figure I.1. Evolution de l'écart de performances entre CPU et mémoire 1

Figure I.2. Evolution de la surface des SoC 2

Figure I.3 Energie et temps d'accès pour mémoires 2

Figure I.4 : Flot de conception 10

Figure I.5 : Approche conventionnelle du codesign 12

Figure I.6 : Approche de conception Codesign basé sur modèle 15

Figure I.7 : Diagramme en Y de Gajski 20

Figure II.1 : Hiérarchie mémoire 26

Figure II.2 : Cache à adressage direct 28

Figure II.3 : Cache associatif par ensembles 29

Figure II.4 : Cache complètement associatif 30

Figure II.5 : Mémoire Scratch pad 34

Figure II.6: Croissance de la consommation avec la capacité du cache 37

Figure II.7 : Variation de la consommation d'énergie, cycle moyen d'exécution et le taux des défauts avec l'augmentation de la capacité du cache. 38

Figure II.8 : Hiérarchie mémoire avec inclusion du cache victime 41

Figure II.9 : Exemple de cas d'utilisation du scratchpad 44

Figure III.1 : Architecture d'un data cache partitionné en Scalar cache et Array cache 57

Figure III.2. Principe de fonctionnement d'un simulateur dirigée par trace 60

Figure III.3. Les simulateurs de SimpleScalar 64

Figure IV.1 : Taux de défauts pour un cache de données de 8 KOctets 73

Figure IV.2 : Taux de défauts pour un cache de données de 16 KOctets 74

Figure IV.3: Taux de défauts pour un cache de données de 32Koctets 74

Figure IV.4: Taux de défauts pour un cache de données de 64 KOctets 75

Figure IV.5: Taux de défauts pour une taille du bloc de 16 Octets 76

Figure IV.6: Taux de défauts pour une taille du bloc de 32 Octets 77

Figure IV.7: Taux de défauts pour une taille du bloc de 64 Octets........ 77

Figure IV.8: Taux de défauts de cache de données en fonction de la taille du bloc et la capacité du cache 79

Figure IV.9: Variation du temps d'accès du cache en fonction de la taille du bloc 80

Figure IV.10: Variation du temps d'accès au cache des benchmarks 82

Figure IV.11: Variation de la surface en fonction de la taille du bloc du cache 83

Figure IV.12: Variation de l'énergie consommée par accès en fonction de la taille du bloc du cache 85

Figure IV.13: Variation de l'énergie consommée lors des accès au cache par les benchmarks........ 88

Figure IV.14: Interaction entre temps d'accès, surface du cache et énergie consommée 90

Figure IV.15: Principe de fonctionnement du Cache Partitionné 91

Figure IV.16: Taux de défauts pour un Scalar Cache de 8 KOctets 93

Figure IV.17: Taux de défauts pour un Scalar Cache de 16 KOctets 93

Figure IV.18: Taux de défauts pour un Scalar Cache de 32 KOctets 94

Figure IV.19: Taux de défauts du Scalar Cache pour une taille du bloc de 8 Octets 95

Figure IV.20: Taux de défauts du Scalar Cache pour une taille du bloc de 16 Octets 95

Figure IV.21: Taux de défauts du Scalar Cache pour une taille du bloc de 32 Octets 96

Figure IV.22: Taux de défauts du Array Cache pour une capacité de 4 KOctets 97

Figure IV.23: Taux de défauts du Array Cache pour une capacité de 8 KOctets 97

Figure IV.24: Taux de défauts de l' Array Cache pour une capacité de 16 KOctets 98

Figure IV.25: Taux de défauts de l' Array Cache pour un bloc de 16 Octets 99

Figure IV.26: Taux de défauts de l' Array Cache pour un bloc de 32 Octets 99

Figure IV.27: Taux de défauts de l' Array Cache pour un bloc de 64 Octets 100

Figure IV.28: Variation du taux de défauts d'un cache de données unifié de 16 Koctets
par rapport à un cache partitionné en Scalar Cache et Array Cache(8Koctets et 4 Koctets) 101
Figure IV.29: Variation du taux de défauts d'un cache de données unifié de 32 Koctets par rapport
à un cache partitionné en Scalar Cache et Array Cache (16Koctets et 8 Koctets) 101

LISTE DES TABLEAUX

Tableau IV. 1 : Quelques capacités du data cache pour quelques processeurs embarqués 67

Tableau IV.2 : Quelques tailles de ligne utilisées dans les caches pour quelques processeurs embarqués 69

Tableau IV.3 : Description de la suite Mediabench 72

Introduction Générale

INTRODUCTION GENERALE

La croissance des capacités d'intégration a permet à des milliers de transistors de se cohabiter sur la même puce ce qui a entrainé des systèmes de plus en plus complexes. Si les capacités de calcul augmentent fortement suite à cette intégration, il n'en va pas de même pour les mémoires. Cette situation a crée un fossé de performance entre mémoire et unité de traitement, ce fossé n'est pas récent, en effet dans l'ordinateur IBM System / 360 Model 50 introduit en 1964 le rapport de performance entre mémoire et processeur est de 4 avec une période d'horloge du processeur de 500 *ns* alors que la mémoire opère a 2 μs [1]. Ceci montre que l'écart de performance entre processeur et mémoire existait depuis les premières ères des systèmes informatiques et ne cesse de croître au fil des années. Actuellement, si on prend une DRAM de type Corsair DDR2 ayant une latence de 5.9 *ns* et un processeur tel qu'un Pentium 4 ayant une fréquence de 3.8 GHZ qui opère à une période d'horloge de 0.26 *ns* on obtient ainsi un rapport de performance de 22.7 [1]. Ainsi les mémoires constituent un goulet d'étranglement qui délimite les performances du système vu que le processeur ne peut exécuter une application plus rapidement que le rythme de production de l'information par la mémoire au processeur (Figure I.1). Des études récentes montrent que les performances du processeur évoluent de 50 à 100% par année, alors que celles des mémoires ne dépassent pas 7 % durant la même période [2].

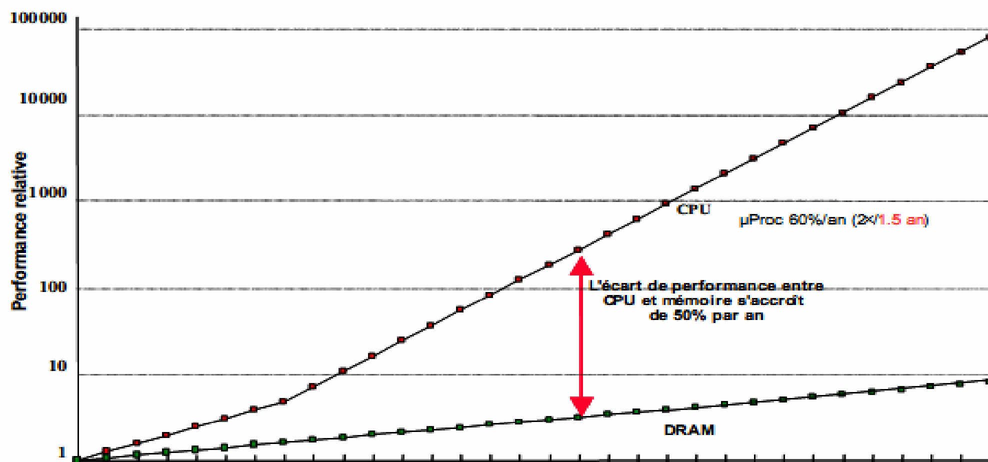


Figure I.1. Evolution de l'écart de performances entre CPU et mémoire [2]

Ce goulet d'étranglement ne se limite pas aux performances, la prédominance des mémoires sur la surface et la consommation globale des systèmes s'accentuent avec l'augmentation des quantités d'information traitées. Ainsi, le problème devient plus délicat lorsqu'on parle des systèmes on chip. D'après les estimations du *2000 International Technology Roadmap for Semiconductor* près de la moitié de la surface des systèmes actuels est dédiée à la mémoire, cette proportion devrait passer à 94% aux alentours du 2014 (Figure 1.2).

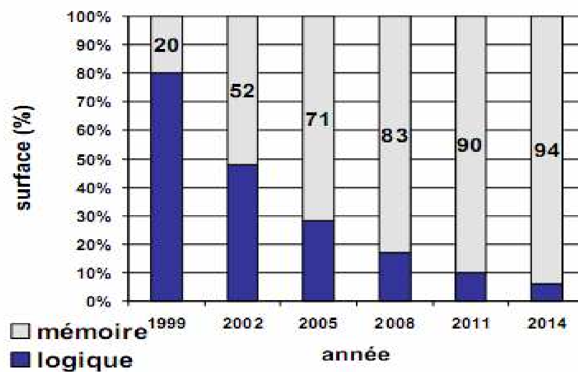


Figure I.2. Evolution de la surface des SoC [3]

Pour la consommation en puissance des mémoires, elle représente actuellement 10 à 15% de la consommation du système et peut atteindre près de 50 % de la consommation globale pour des applications de télécommunication ou multimédia [3].

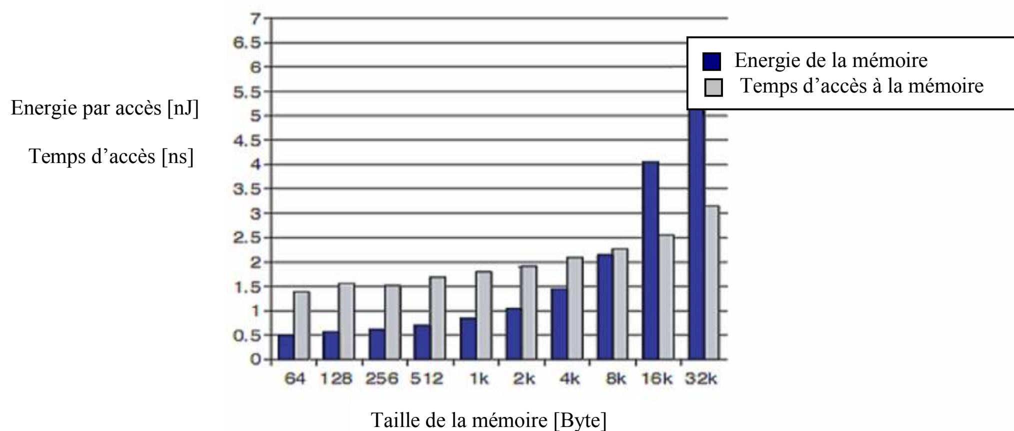


Figure I.3 : Energie et temps d'accès pour mémoires [2]

Vu les opportunités offertes par la mémoire dans l'amélioration de la performance, consommation et espace du système, celle-ci a fait objet, ces dernières années, à plusieurs méthodes, démarches et techniques d'optimisation.

Si la mémoire centrale doit être accédée pour la récupération des données, le gap entre les registres et la mémoire devient plus grand, vu que les registres sont rapides en termes d'accès alors que les mémoires centrales sont lentes. L'une des techniques proposées pour alléger ce gap est la création de niveaux intermédiaires dans l'hiérarchie mémoire. L'instanciation la plus populaire de ce concept est le cache. Le cache est une mémoire de petite taille ayant un temps d'accès rapide placée entre le processeur et la RAM, son principe de fonctionnement repose sur le fait de conserver des informations en vue de leur utilisation dans un futur proche. Leur succès magique dans les systèmes traditionnels a motivé les concepteurs pour les adopter dans la conception des systèmes embarqués. En exploitant les spécificités des ces systèmes, surtout le fait que l'application qui va s'exécuter est connue à l'avance, les chercheurs ne cessent de jour en jour de proposer de nouvelles formes ou de personnaliser les paramètres du cache pour produire des caches dédiés aux systèmes embarqués.

En effet, le problème d'optimisation de la mémoire est un problème multi objectifs et peut être traité comme un problème à deux niveaux d'optimisation. Le niveau externe explore différentes architectures mémoires possibles alors que le niveau interne explore le placement des données au sein des mémoires pour minimiser l'espace [4].

Notre travail se concentre sur l'optimisation du cache des données et son effet sur l'optimisation des ressources d'un système embarqué, on commence par l'étude de l'effet des paramètres du cache tels que la capacité du cache et la taille du bloc sur les performances du système, les métriques de performances prises dans notre étude sont : le taux des défauts du cache, le temps d'accès, la surface et l'énergie consommée par le cache. Puis on va proposer une approche qui consiste à partitionner le cache des données en cache pour les scalaires et un cache pour les tableaux et ceci vu que ces deux types des données présentent des localités différentes d'où le besoin de les séparer pour permettre une meilleure exploitation des caractéristiques de chaque type de données.

Notre mémoire est organisé comme suit :

Le premier chapitre est consacré à une introduction aux systèmes embarqués, où on va présenter la structure ainsi que les caractéristiques de ces systèmes, puis on abordera brièvement les méthodologies de conception d'un système embarqué.

Le deuxième chapitre est désigné pour la présentation des mémoires utilisées dans les systèmes embarqués, là on présentera les types de mémoires utilisées, la hiérarchie mémoire ainsi que les techniques d'optimisation du cache.

Le troisième chapitre aborde les approches de partitionnement du cache ainsi qu'une présentation de l'approche proposée.

Le quatrième chapitre est consacré à l'étude expérimentale, dans lequel on présentera les simulateurs utilisés ainsi que les paramètres adoptés et on termine par une présentation des résultats obtenus lors de notre étude avec une comparaison de l'architecture proposée avec l'architecture standard.

Chapitre I

Introduction aux systèmes embarqués

Introduction

L'évolution de capacité d'intégration des circuits électroniques a permis ces dernières années aux systèmes électroniques embarqués de devenir une réalité très vite perceptible. Au début, les applications des systèmes embarqués étaient surtout dans le milieu militaire à cause d'un ensemble de facteurs, notamment leur prix et l'avantage stratégique d'une telle technologie [5]. Cependant, leur grande potentialité leur a rapidement ouvert les portes vers des applications civiles, ceci est dû principalement au progrès technologique qui a constamment baissé leur prix et augmenté leurs possibilités; aujourd'hui, les systèmes embarqués trouvent des applications partout où la miniaturisation, la mobilité et la puissance de calcul sont nécessaires. Il est intéressant de constater que nous utilisons actuellement, sans nous en rendre compte, plus d'une douzaine de processeurs embarqués dans notre vie courante tels que Téléphones portables, baladeurs, cartes de paiement, appareils photos , électroménager … etc. Ceci est sans parler des applications moins « voyantes » mais pas moins répandues : conduite assistée pour voitures, monitorage du trafic, contrôles d'accès, avionique, systèmes de sécurité, etc. . . Dans ce contexte Patrice Kadionik[1] de l'ENSEIRB (Ecole Nationale Supérieure d'Electronique, Informatique et Radiocommunications de Bordeaux) a écrit :

« *Les systèmes embarqués nous entourent et nous envahissent littéralement, fidèles au poste et prêts à nous rendre service. Ils sont donc partout, discrets, efficaces dédiés à ce que quoi ils sont destinés. Omniprésents, ils le sont déjà et le seront de plus en plus.* » [6].

Il est communément admis que d'une manière générale un système embarqué est tout système conçu pour résoudre un problème ou une tâche spécifique mais n'est pas un ordinateur d'ordre général [4], On qualifiera d'embarqué un système complètement encapsulé dans le dispositif qu'il contrôle. En effet, un système embarqué est intégré dans un système plus large pour lequel il réalise des fonctions de calcul, de surveillance ou de contrôle, il n'est pas visible en tant que tel mais il est intégré totalement dans un équipement doté d'une autre fonction, au point que l'utilisateur le plus souvent n'a pas conscience qu'il utilise un système à base de microprocesseur. On dit que le système est enfoui ce qui traduit plus fidèlement le mot anglais Embedded System.

[1] Patrice Kadionik : maître de conférences responsable de l'option système embarqué à ENSEIRB.

Le système embarqué se présente sous forme d'un système électronique et informatique autonome ne possédant pas des entrées/sorties standards comme un clavier ou un écran d'ordinateur (PC). Contrairement à un PC, l'interface homme-machine d'un système embarqué peut être aussi simple qu'une diode électroluminescente (LED) ou aussi complexe qu'un système de vision de nuit en temps réel [7]. Ils intègrent des parties dites logicielles et des parties matérielles dédiées ou spécifiques de calcul ou de mémorisation. Le système matériel et logiciel sont intimement liés et noyés dans le matériel et ne sont pas aussi facilement discernables comme dans un environnement de travail classique de type PC, ainsi la mise à jour de l'application ne peut être que très limitée par exemple : chargement de nouveaux programmes ou reprogrammation de matériel reconfigurable. Les composantes logicielles peuvent être implantées sur des processeurs à usage général ainsi que sur des processeurs à usage spécifique (DSP, ASIP). Les composantes matérielles sont implantées soit sur des composantes dédiées (ASIC) soit sur des composantes reprogrammables (FPGA) [8].

I.1. Structure d'un système embarqué

Un système embarqué est composé typiquement de matériel exécutant du logiciel, ce qui implique un ou plusieurs processeurs, des mémoires, des interfaces d'E/S et éventuellement des circuits dédiés à des applications spécifiques. Toutes ces parties sont implémentées sur un seul composant constituant ainsi ce qu'on appelle SOC : *System On Chip*.

Vu que l'application qui va être exécutée sur un système embarqué est connue durant la phase de conception, les concepteurs tirent avantage de cette information en complétant généralement cette architecture par des circuits numériques programmables (FPGA), des circuits dédiés à des applications spécifiques (ASIC) ou des modules analogiques jouant le rôle de coprocesseur afin de proposer des accélérations matérielles au processeur et d'optimiser les performances et la fiabilité de ces systèmes pour une application donnée.

I.2. Caractéristiques d'un système embarqué

Les systèmes embarqués exposent généralement des caractéristiques qui permettent de les distinguer des autres systèmes. Les caractéristiques communes des systèmes embarqués comme présentés par Peter Marwedel dans [9]:

- ✓ Les systèmes embarqués sont fréquemment connectés à l'environnement physique par des capteurs utilisés pour collecter les informations et des actionneurs pour contrôler cet environnement.
- ✓ Les systèmes embarqués doivent être fiables : Ces systèmes sont utilisés dans des applications de plus en plus critiques dans lesquelles leur dysfonctionnement peut générer des nuisances, des pertes économiques ou des conséquences inacceptables pouvant aller jusqu'à la perte de vies humaines, c'est le cas par exemple des applications médicales ou celles de transports pour lesquelles une défaillance peut avoir un impact direct sur la vie des êtres humains, ce type de système doit garantir une très haute fiabilité et doit pouvoir réagir en cas de panne de l'un de ses composants.

  La fiabilité du système inclus les aspects suivants :

  - ❖ **Sureté de fonctionnement** : est la probabilité que le système ne tombe pas en panne.
  - ❖ **Maintenance** : Probabilité qu'un système en panne peut être réparé dans un certain temps.
  - ❖ **Disponibilité** : est la probabilité que le système soit disponible.

    La sureté de fonctionnement et la maintenance, toutes deux doivent être très hautes pour assurer une haute disponibilité.
- ✓ **Sécurité** : Ce terme décrit la propriété que des données confidentielles reste confidentielles et qu'une communication légale est garantie.
- ✓ Les systèmes embarqués doivent être efficaces : Les paramètres suivants peuvent être utilisés pour mesurer l'efficacité d'un système embarqué :
  - ✓ **L'Energie** : Les systèmes embarqués autonomes doivent avoir une faible consommation car ils sont alimentés par des batteries. Une consommation excessive augmente le prix de revient des systèmes embarqués car il faut alors des batteries de plus forte capacité.

- ✓ **Taille du code** : Tout le code qui doit s'exécuter dans un système embarqué doit être sauvegardé dans le système. Typiquement, il n'existe pas de disque dur ou le code va être enregistré, ainsi la taille du code de l'application voulue doit être aussi réduit que possible et ceci est spécialement vrai pour les systèmes on chip.
- ✓ **Efficacité** : Le minimum de ressources doit être utilisé pour l'implémentation de la fonctionnalité désirée. On doit être capable de respecter les contraintes de temps en utilisant le moindre nombre de ressources hardware et d'énergie.
- ✓ **Poids** : Les systèmes embarqués portables requièrent le plus souvent un faible encombrement comme les PDA, les téléphones portables…etc. Par conséquent la réalisation d'un packaging en faisant cohabiter dans un faible volume, électronique analogique, électronique numérique et RF sans interférences est une tâche très difficile.
- ✓ **Coût** : Lorsque les systèmes embarqués sont utilisés dans des produits de grande consommation ils sont fabriqués en grande série. Les exigences de coût se traduisent alors en contraintes drastiques sur les différentes composantes du système : utilisation de mémoire de faible capacité et des processeurs à 4 ou 8 bits mais en grand nombre. Ainsi les systèmes embarqués sont sensibles au coût de production et doivent avoir des prix de revient extrêmement faibles.

- ✓ Les systèmes embarqués sont dédiés à une certaine application, ils exécutent généralement une seule fonction ou un ensemble de fonctions d'une manière répétitive.
- ✓ La plupart des systèmes embarqués ne disposent pas d'entrées/sorties standard tel que claviers, souris.
- ✓ Plusieurs systèmes embarqués doivent respecter des contraintes de temps réel, les opérations de calcul sont alors faites en réponse à un évènement externe. La validité et la pertinence d'un résultat dépendent du moment où il est délivré. Une échéance manquée induit une erreur de fonctionnement qui peut entraîner soit une panne du système, soit une dégradation de ses performances
- ✓ Certains systèmes embarqués sont des systèmes hybrides, ils incluent généralement une partie analogique et une partie numérique.

- ✓ Typiquement les systèmes embarqués sont des systèmes réactifs, un système réactif est un système qui répond constamment aux sollicitations de son environnement en produisant des actions sur celui-ci.

I.3. Méthodologies de conception des systèmes embarqués

Le processus de conception des systèmes dits mixtes ou intégrés qui sont des systèmes à deux dimensions, une logicielle et une autre matérielle « est un ensemble d'actions, ordonnées et bien définies, à exécuter par l'outil ou l'opérateur de conception pour satisfaire une spécification d'un système, il permet de passer d'un modèle dit de haut niveau qui est une description fonctionnelle du circuit à un modèle dit de bas niveau correspondant à l'élaboration des plans des masques qui vont définir la topologie des circuits. Ces actions sont étroitement dépendantes de la nature du système en cours de conception » [10].

Le processus de conception accueille à son entrée la spécification d'un produit et génère à sa sortie l'implémentation de celui-ci. Fournie par le client, la spécification englobe toutes ou quelques consignes que l'implémentation du produit doive satisfaire. Au cours des décennies passées, la conception des composantes logicielles et matérielles d'un système embarqué s'effectuait séparément. Puis le logiciel obtenu est exécuté sur le matériel prototype, si les contraintes de la spécification requises par le système ne sont pas satisfaites, le processus de conception est réitéré dans l'espoir de retrouver un bon prototype.

Vu la complexité grandissante sans cesse des systèmes soumis au processus de conception et les diverses contraintes auxquelles doivent faire face les concepteurs, cette technique s'est avérée lourdement coûteuse en termes du temps de réalisation et du coût du système. La nécessité de faire appel à des nouvelles méthodes de conception s'est avérée indispensable, c'est ainsi que le codesign a fait son apparition, le codesign s'efforce de mettre au point un prototype précoce pour valider la spécification et fournir à la clientèle un « feedback » durant le processus de conception en générant une architecture abstraite autour des processeurs communicants qui implémentent la spécification initiale. Cette architecture peut être raffinée davantage afin de produire finalement une architecture hétérogène plus complète, englobant des modules matériels, logiciels et de communication.

L'avantage du codesign réside dans le fait qu'il permet de repousser le plus loin possible dans la conception du système les choix matériels à faire contrairement à l'approche classique ou les choix matériels sont faits en premier lieu.

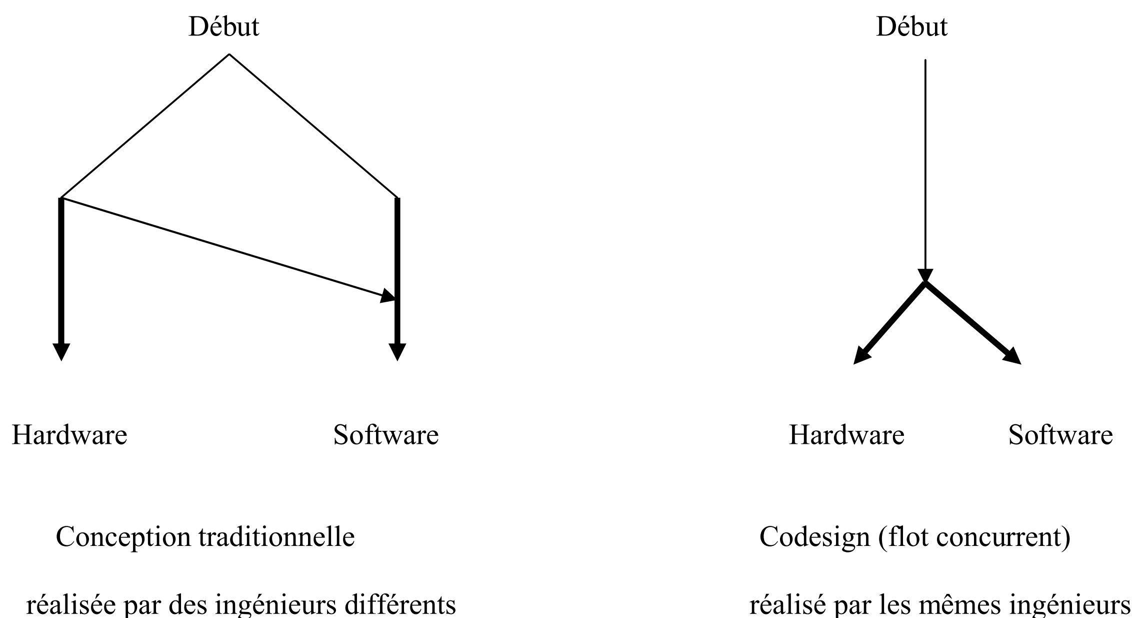


Figure I.4 : Flot de conception

I.3.1. Approches de conception des systèmes embarqués

I.3.1.1. Approche conventionnelle du Codesign

Généralement, le processus de conception codesign comprend les étapes suivantes :

I.3.1.1.1. Spécification : La première étape de conception consiste à élaborer une spécification relativement informelle du système à construire, cette spécification peut s'agir de centaines de pages de description de la fonctionnalité souhaitée. La spécification consiste en une spécification fonctionnelle et une spécification non fonctionnelle. La spécification fonctionnelle est l'algorithme qui décrit l'application, elle est écrite dans un langage séquentiel standard comme le C ou le C++, par contre la spécification non fonctionnelle est l'ensemble des contraintes que le système devra supporter comme la vitesse, la surface ou la consommation. Cette étape est très importante dans le flot de conception et peut avoir un fort impact sur les résultats.

I.3.1.1.2. Modélisation : Une fois la spécification des parties matérielles et logicielles du système est élaborée sous forme textuelle, l'étape de modélisation consiste à produire à partir de la liste de spécification abstraite une formalisation sous forme d'un modèle à l'aide d'un langage spécialisé pour la description des systèmes matériels. La spécification fonctionnelle est partitionnée en un ensemble de tâches, ainsi le système est modélisé comme un ensemble de tâches communicantes.

I.3.1.1.3. Partitionnement : Durant cette étape le choix de l'architecture cible intervient et l'affectation des tâches dans l'architecture du système se fait en analysant et définissant pour chacune d'elles laquelle sera réalisée en matériel ou en logiciel et par quel processeur ou coprocesseur.

Une simulation dans un simulateur au niveau architecture de système est ensuite effectuée. En effet les performances d'un circuit à l'intérieur d'un système intégré sont difficilement évaluables autrement que par la simulation. Le simulateur utilisera les modèles fonctionnels des composants matériels du système et sera capable de simuler les communications entre modèles. Le concepteur doit pouvoir tenter plusieurs architectures matérielles de son système afin de pouvoir choisir le meilleur compromis entre les performances et le coût du circuit.

I.3.1.1.4. Synthèse et optimisation : Une fois l'exploration de l'espace de conception effectuée, les décisions d'implantation peuvent être prises et la synthèse du matériel commence. La synthèse consiste en un processus de génération du modèle physique à partir du modèle comportemental, elle consiste en un raffinage d'un modèle abstrait à un modèle plus détaillé de plus bas niveau d'abstraction. Cette translation peut être réalisée en utilisant les outils tels que les compilateurs et les compilateurs croisés. Les outils de synthèse du matériel par exemple peuvent lire la description matérielle écrite en format VHDL ou Verilog et génèrent des plans de masque. D'une manière similaire les compilateurs croisés compilent des programmes écrits en langage de haut niveau en un ensemble d'instructions natives du processeur embarqué.

Durant cette étape, le modèle HDL obtenu est optimisé à la lumière de certaines contraintes telles que le temps d'exécution, la surface du circuit et son coût. Pour le faire, des outils de synthèse permettent de produire des descriptions du système de moins en moins abstraites et de plus en plus détaillées, en vu de leur réalisation matérielle.

I.3.1.1.5. Validation et cosimulation : La cosimulation est une tâche très difficile et elle fait objet de plusieurs sujets de recherches, elle permet de vérifier si le modèle implémenté reproduit exactement la fonctionnalité du modèle spécifié. Il existe des plates formes commerciales de cosimulations disponibles qui peuvent simuler des modèles synthétisés de hardware et de software dans un environnement intégrés. Mais typiquement les concepteurs simulent les modèles synthétisés séparément puis interprètent les résultats ou génèrent des prototypes qui peuvent être simulés. Toutefois la deuxième méthode tend à être couteuse car ce processus doit être répété plusieurs fois avec modification à chaque fois de la conception originale.

I.3.1.1.6. Intégration et Test de l'intégration : Les résultats de la cosimulation sont vérifiés par rapport aux besoins fonctionnels et les contraintes de la spécification. Si la performance du système est aussi validée à cette étape le système est intégré et un test est effectué sur l'intégration.

Si le système ne répond pas aux exigences, tout le processus à partir de l'étape de partitionnement est répété.

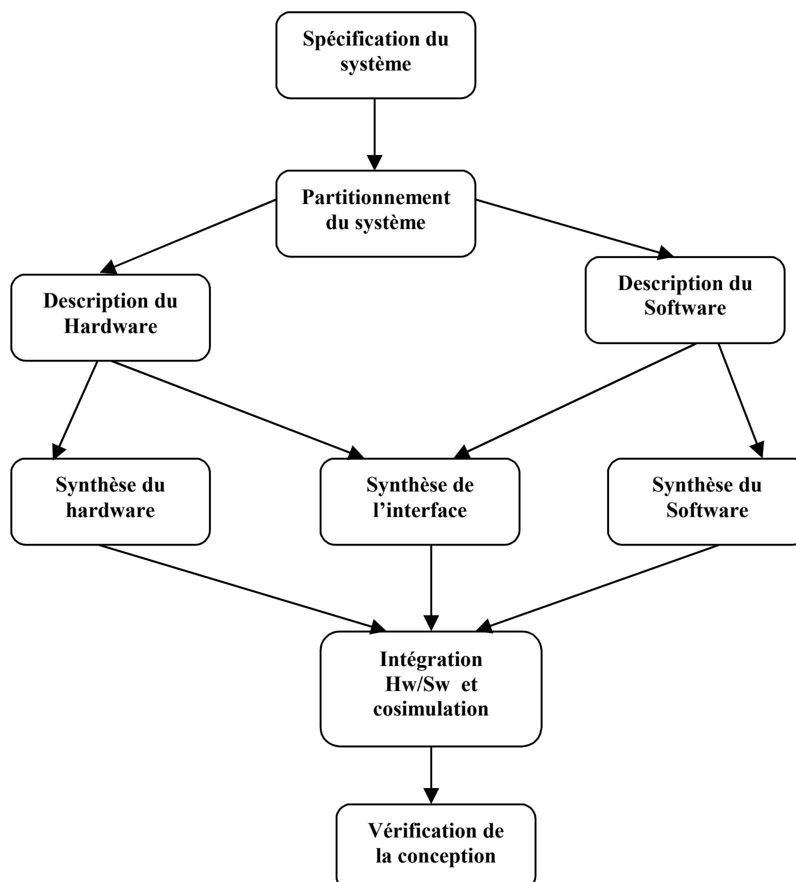


Figure I.5 : Approche conventionnelle du codesign [11]

I.3.1.2. Approche basée sur Modèle

Dans l'approche conventionnelle décrite précédemment, le partitionnement du système est réalisé très tôt dans le processus de conception, ceci réduit la flexibilité du concepteur et aussi l'efficacité de la conception finale. Un autre schéma est proposé, c'est l'approche basée modèle (model based approach), les étapes de cette approche comme présentées dans [11] sont :

Dans cette approche, la spécification est toujours la première étape dans le processus de conception.

I.3.1.2.1. Modèle du système (System Model)

Le modèle du système est une représentation comportementale du système écrite sous forme d'un ensemble d'instructions. Le degré du détail du modèle est appelé la granularité du modèle. Il existe plusieurs degrés de granularité, au plus haut niveau le système peut être modélisé par un algorithme, le plus bas niveau est atteint par l'écriture d'une description du niveau registre (register level description) et niveau porte (gate level description). Un bon modèle doit contenir suffisamment d'informations pour permettre sa simulation ultérieurement et les résultats de la simulation doivent être identiques aux résultats prévus du système final.

Les langages de descriptions des modèles de système ont évolué durant ces années, un langage idéal décrit le comportement exact de n'importe quelle partie du système sans se soucier si elle est implémentée en hardware en software. SystemC est un très bon exemple de ce type de langage.

I.3.1.2. 2. Bibliothèque de Modèle

Le modèle peut être soit construit, soit choisis parmi les modèles déjà existants dans la bibliothèque. Des modèles des différents composants du système qui ont été complètement testés sont ajoutés à la bibliothèque pour faciliter leurs réutilisations. Comme la bibliothèque s'accroit avec le temps, le temps de conception est largement réduit.

Pour SystemC, cette bibliothèque est celle du C++ contenant des classes et des méthodes qui représentent des composants de système.

I.3.1.2.3. Raffinement du modèle et validation

Les sorties de la simulation sont utilisées pour valider le modèle. La validation ici signifie la vérification fonctionnelle et n'implique pas de contraintes de temps, d'espace ou d'énergie. Le résultat de la simulation est comparé avec les valeurs attendues.

I.3.1.2.4. Extraction du graphe et annotation

A partir du modèle raffiné on construit un graphe modèle du processus (Process Model Graph : PMG), c'est un graphe orienté ou chaque nœud représente un processus dans le modèle et les arcs représentent des signaux. Les modèles de processus sont des tâches concurrentes dans le système. Une fois le PMG construit, le graphe sera ensuite annoté avec des contraintes de temps, vitesse, délai…etc.

I.3.1.2.5. Partitionnement Hardware/Software

Le partitionnement hardware/Software est aussi appelé la technologie de l'affectation. Le principe de base est le même expliqué dans l'approche conventionnelle. L'idée est d'affecter autant de nœuds possibles au software selon la capacité du processeur embarqué tout en essayant de respecter les contraintes spécifiées auparavant. Ce problème est similaire à celui de la théorie des graphes ou on doit trouver le plus grand sous graphe qui satisfait certaines conditions.

I.3.1.2.6. Synthèse du système et validation

Le système partitionné et ensuite synthétisé en plusieurs entités. Les outils de cosimulation sont alors utilisés pour valider le système. Ce processus peut être ensuite optimisé pour des contraintes d'espace, d'énergie…etc.

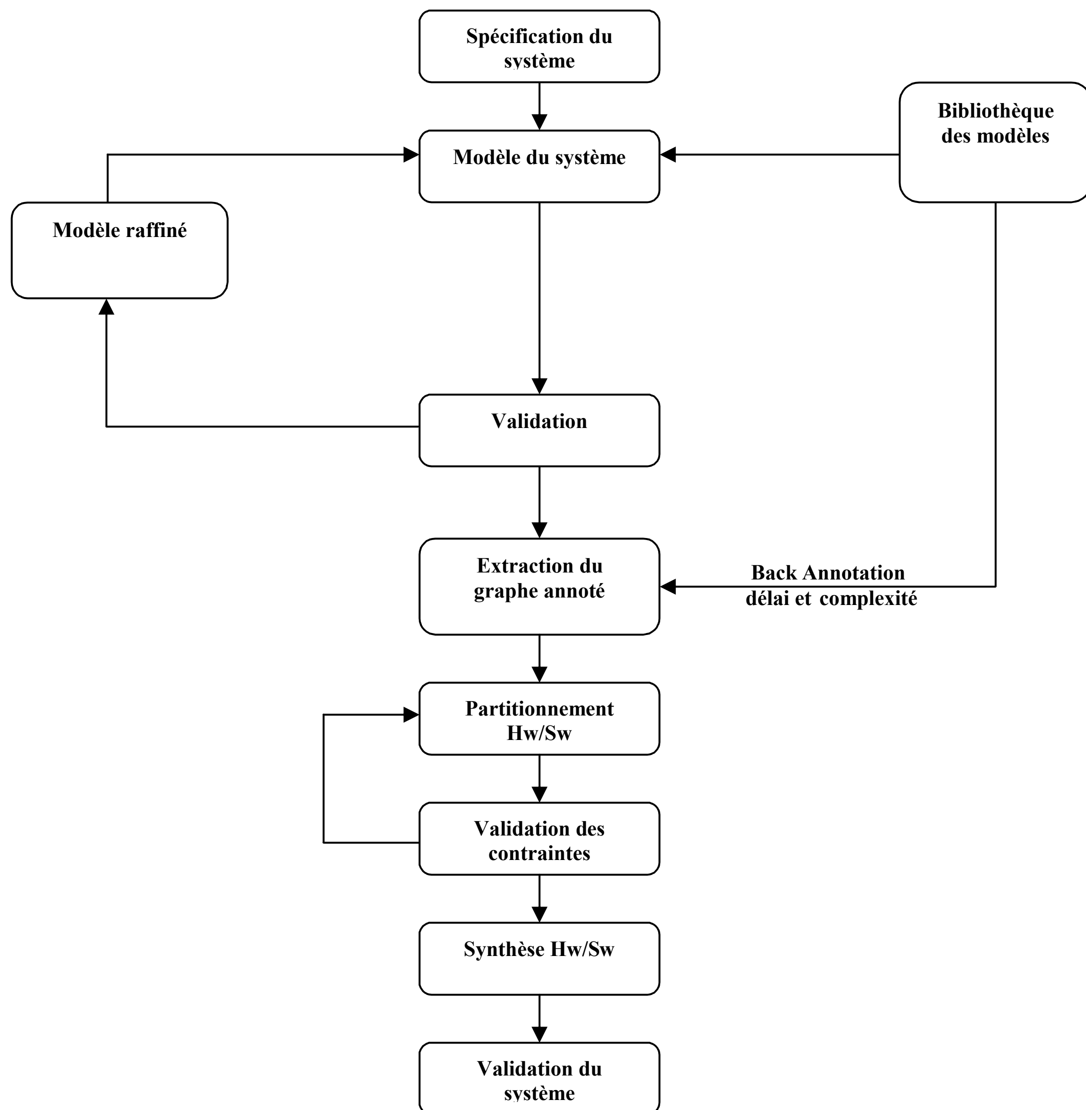


Figure I.6 : Approche de conception Codesign basé sur modèle [11]

I.3.2. Démarches de conception

Le chemin de conception peut être réalisé de deux façons :

I.3.2.1. La conception ascendante (bottom up)

Elle est utilisée dans le cas ou le concepteur prend connaissance relativement tôt qu'il devra utiliser certaines briques de base (comme par exemple les Unités Arithmétique et Logique), le concepteur peut commencer son travail par leur élaboration, et ensuite les assembler progressivement entre elles de façon à monter de niveau d'abstraction.

I.3.2.2. La conception descendante (Top down)

Une démarche descendante va partir des spécifications pour arriver, à une représentation informatisée du produit visé [12]. En général, dans une telle démarche le concepteur du système embarqué obtient d'un client des spécifications du système requis. Il commence par les diviser en blocs fonctionnels, c'est à dire une description de haut niveau. Chacun de ces blocs peut alors être spécifié plus finement. En bas de l'échelle d'abstraction, le concepteur arrive finalement à écrire des circuits, c'est à dire des ensembles de portes logiques reliées entre elles par des fils.

Pour bénéficier des avantages des deux il vaut mieux parler de « Meet in the Middle » dans une démarche globalement descendante comportant des boucles d'ascendance [12].

I.4. Synthèse des circuits

L'évolution de la technologie des semi conducteurs a permet l'intégration de milliers de transistors sur une seule puce. Cet accroissement a rendu le processus de conception « à la main » des systèmes complexes une tâche fastidieuse et difficile à réaliser, vu que le raisonnement en portes et transistors est devenu impossible d'où la nécessité de faire appel à une abstraction du niveau de conception et une automatisation du processus de conception. Ainsi un circuit peut être représenté dans un niveau abstrait décrivant son comportement puis transformé à travers une succession d'étapes en un produit concret physiquement implémenté, cette transformation doit être faite d'une manière automatique.

L'abstraction est un concept très important dans la conception des systèmes complexes, il est basé sur le fait d'ignorer ou de supprimer dans un modèle certains détails du système original afin de simplifier sa taille (du modèle), d'en déduire des conclusions d'ordre général et aussi pour réduire le volume du travail requis [10]. On distingue plusieurs niveaux d'abstraction.

I.4.1. Niveaux d'abstraction

I.4.1.1. Niveau système : on essaye à ce niveau d'identifier l'architecture globale du système en spécifiant ses principaux éléments et leurs communications bilatérales avec l'environnement. A ce niveau, on ne s'intéresse pas aux détails des composants du système, une vue globale suffit.

I.4.1.2. Niveau comportemental : la fonctionnalité d'un composant à ce niveau est décrite depuis ses ports d'entrée jusqu'à ses ports de sortie. Le « timing » n'est pas requis dans le modèle; cependant, il peut y être considéré afin de servir de repère pour certaines exécutions.

I.4.1.3. Niveau transfert de registres (RTL) : à ce niveau, on suppose que le système est synchrone. Des contraintes sur le « timing » et les ressources du système sont définies. Le comportement de chaque composant est donc décrit pour chaque cycle d'horloge. Les opérations des flux de contrôle et de données sont accomplies à l'aide des registres.

I.4.1.4. Niveau porte : le système à ce niveau est globalement vu comme étant un circuit séquentiel. Ce dernier pourrait comprendre des bascules et des portes logiques. Un tel modèle peut être dérivé d'une description RTL tout en faisant usage des librairies des différentes technologies (TTL, CMOS, etc.). Ces dernières, créées sous une optique de réutilisation, regroupent des modèles « clé en main » de bascules et de portes.

I.4.1.5. Niveau transistor et mise en carte : à ce niveau, le système paraît comme étant une carte portant plusieurs composants actifs (transistors) et passifs (résistances, capacités, etc.). Une optimisation des emplacements des composants sur la surface offerte s'en suit.

Les outils de synthèse cherchent tout d'abord à compiler une vue comportementale en un modèle cible adapté à la synthèse et indépendant du langage utilisé. Puis ils optimisent ce modèle avant d'en effectuer une projection structurelle sur un niveau d'abstraction immédiatement inférieur. Il existe donc plusieurs niveaux de synthèse qui correspondent chacun au passage d'un niveau d'abstraction n à un niveau n-1.

I.4.2. Les différents niveaux de synthèse

I.4.2.1. La synthèse système

La première étape du flot de synthèse se situe au niveau système, elle consiste à partir de la spécification de découper le système en sous systèmes (processus) communicants. Un sous système peut être partitionné à son tour en un ensemble de processus communicants. Ce découpage peut être guidé par plusieurs paramètres tels que : la fonctionnalité, la synchronisation, la concurrence des

tâches…etc. Le niveau système est utilisé pour spécifier des systèmes entiers comprenant des parties logicielles et des parties matérielles.

I.4.2.2. La synthèse comportementale

La synthèse comportementale, appelée aussi synthèse de haut niveau ou architecturale, est la transformation d'une description de type algorithmique en une description de type transfert entre registres : *Register Transfert Level*. Une description algorithmique ne comporte aucune information architecturale et temporelle, elle représente simplement l'algorithme à compiler sur le silicium.

La synthèse de haut niveau a été développée pour réduire le temps de conception des solutions matérielles en élevant le niveau d'abstraction [3], en effet l'automatisation du flot de synthèse haut niveau permet de réduire considérablement le temps entre la spécification et le silicium et d'augmenter la fiabilité de conception. Toute fois elle constitue un problème très complexe, l'outil de synthèse doit prendre beaucoup de décisions et respecter de nombreuses contraintes pour produire un résultat acceptable [13]. Deux critères principaux permettent de juger la qualité d'un outil de synthèse : la richesse de son langage d'entrée et la puissance de ses méthodes d'optimisation qui donnent la qualité de l'implémentation. Quel que soit le niveau d'abstraction, la synthèse prend en compte différents objectifs d'optimisation telle que la minimisation du délai, de la surface, la consommation du circuit…etc.

Vu la complexité de la synthèse d'un circuit, elle est subdivisée en plusieurs étapes :

La sélection : L'étape de sélection consiste à choisir la nature des ressources matérielles (opérateurs) qui réaliseront les opérations. Le choix des composants se fait sur des critères tels que la surface, la vitesse ou la consommation.

L'allocation : L'étape d'allocation détermine, pour chaque type d'opérateur sélectionné, le nombre de ressources utilisées dans l'architecture finale.

L'ordonnancement : L'étape d'ordonnancement affecte une date d'exécution à chacune des opérations en tenant compte d'une part des dépendances de données et d'autre part des contraintes.

Assignation : L'étape d'assignation associe à chaque opération un opérateur.

I.4.2.3. La synthèse *RTL*

Le point d'entrée de la synthèse *RTL* est une description comportementale synchrone spécifiant le comportement d'un circuit cycle par cycle. Les états du circuit sont complètement définis et l'ensemble des transitions d'un état à un autre est décrit sous la forme d'un ensemble de fonctions de transfert entre des registres. Cependant les registres matériels ne sont pas nécessairement identifiés.

La description comportementale est généralement constituée d'un ensemble d'affectations concurrentes et d'un ensemble de processus séquentiels communicants. Elle défini le comportement d'éléments matériels interconnectés, tels que des automates d'états, des mémoires, des opérateurs arithmétiques et des registres. Un outil de synthèse *RTL* peut cependant modifier l'allocation de ces ressources matérielles. Ce niveau est le niveau d'entrée des outils industriels tels que : *Design Compiler* de *Synopsys*, *Synergy* de *Cadence*…etc.

D'une manière générale, la tâche principale d'un outil de synthèse *RTL* est de transformer une telle description comportementale en une interconnexion d'éléments matériels connus de l'outil, tout en respectant des contraintes d'optimisation. Ces éléments matériels sont par exemple des portes logiques, des bascules ou bien encore des mémoires ou des opérateurs arithmétiques.

I.4.2.4. La synthèse logique

L'entrée de la synthèse logique est une description définissant le comportement d'un circuit sous la forme d'un ensemble d'affectations concurrentes représentant un réseau booléen. Cette description peut être vue comme le résultat intermédiaire d'une synthèse *RTL* : c'est à dire juste avant d'effectuer la projection structurelle, et après avoir ramené à un niveau booléen tous les opérateurs complexes. Ce type de description est donc facilement représentable sous la forme d'une interconnexion de portes logiques simples, de bascules et de barrières trois états.

La synthèse logique doit d'une part optimiser le réseau booléen, elle doit d'autre part le transformer en une interconnexion de portes d'une bibliothèque en un réseau de logique programmable ou encore en un réseau de portes pré-diffusées. C'est le niveau de sortie de tous les outils industriels.

Il est important de noter que l'ordre des étapes de synthèse peut varier selon les outils et les contraintes supportées. En effet, certaines techniques de synthèse effectuent l'ordonnancement puis l'assignation, d'autres l'assignation puis l'ordonnancement et dans certains cas l'ordonnancement et l'assignation de façon simultanée.

Les niveaux d'abstraction et les différents niveaux de synthèse sont bien décrits dans le diagramme en Y présenté par Gajski et Kuhn.

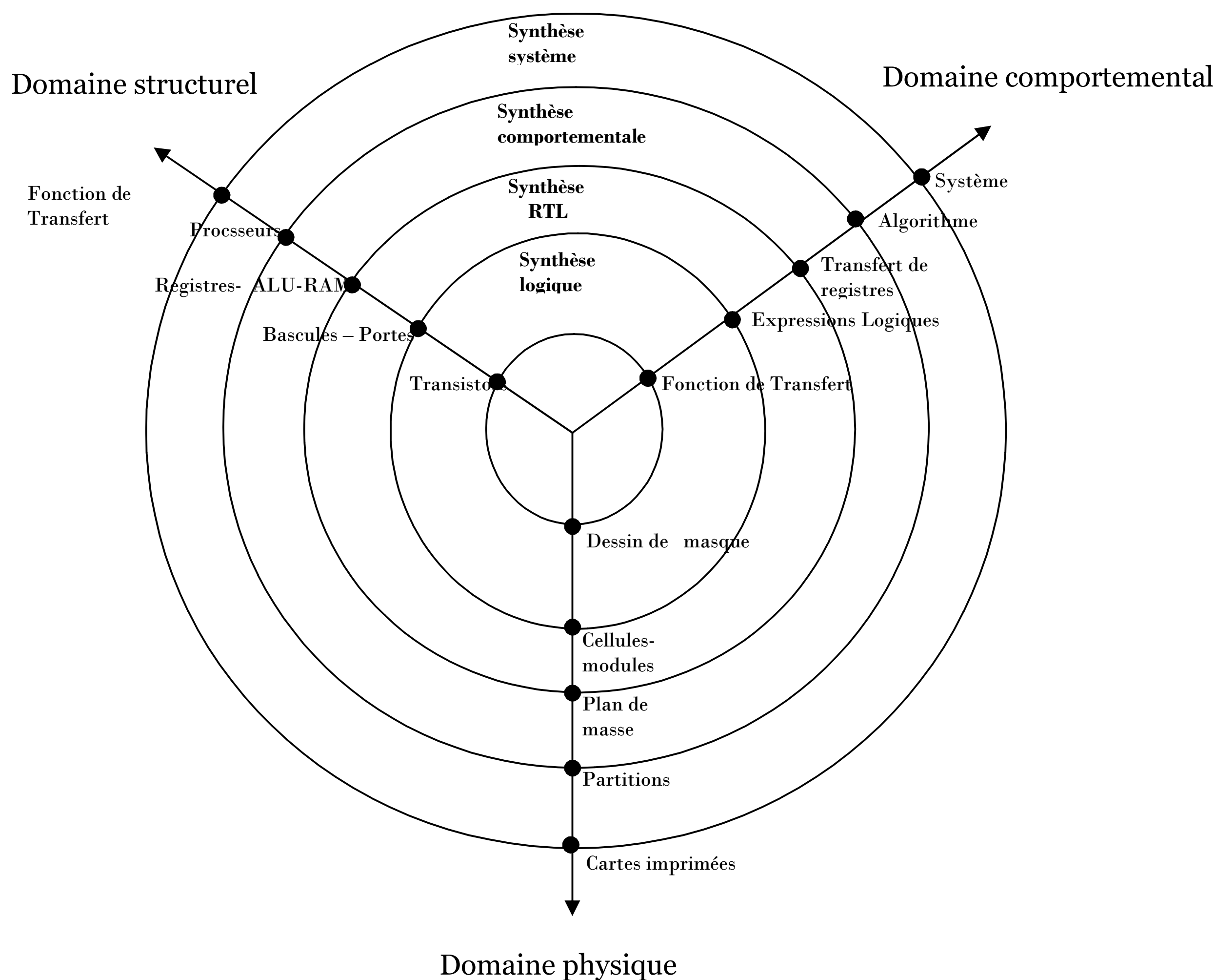


Figure I.7 : Diagramme en Y de Gajski

Dans ce diagramme, les trois domaines comportemental, structurel et physique dans lesquels un système peut être spécifié sont représentés par les trois axes. Les cercles concentriques représentent les niveaux d'abstraction. Une tâche de synthèse peut être vue comme une transformation d'un axe à un autre et/ou comme une transformation d'un niveau d'abstraction à un autre niveau.

Chapitre II

Les mémoires dans les systèmes embarqués

Introduction

Dans les systèmes embarqués, on trouve diverses informations à mémoriser. On distingue les programmes qui doivent être exécutés par les processeurs et les données de l'application. On utilise généralement plusieurs types de mémoires pour le stockage de ces informations Ram, Rom, flash…etc.

Il est clair que la rapidité d'un programme fonctionnant sur un ordinateur n'est pas liée exclusivement aux calculs. En effet, les accès à la mémoire qui sont réalisés pour la récupération des données peuvent donner lieu à des durées non négligeables, ceci vient du fait que la mémoire est organisée en différents niveaux, et que les temps d'accès à ces niveaux ne sont pas identiques [14]. Tout d'abord, on présente un aperçu sur les différents types de mémoires utilisés dans les systèmes embarqués.

II.1. Classification fonctionnelle des mémoires

Selon le type d'opération que l'on peut effectuer sur les mémoires (Lecture, Ecriture) on peut distinguer deux types de mémoires. Les mémoires mortes n'autorisant que les accès en lecture, et les mémoires vives permettant les accès en lectures et en écriture.

II.1.1. Les mémoires mortes

La principale caractéristique des mémoires mortes est qu'elles ont la particularité de garder l'information après une coupure d'alimentation. Ces mémoires sont utilisées pour stocker les informations nécessaires au démarrage et au fonctionnement du système (programmes d'initialisation, système d'exploitation et application), le microcontrôleur utilise la ROM pour stocker les instructions du programme qu'il exécute [15]. On peut distinguer plusieurs types :

- ✓ **Les ROM** : (Read Only Memory) le contenu de ces mémoires est défini lors de la fabrication et n'acceptent que les accès en lecture. Une fois écrite le contenu ne peut plus être changé.
- ✓ **Les PROM** : (Programmable Read Only Memory) sont des mémoires Rom qui offrent à l'utilisateur la possibilité de les programmer une seule fois. Quand le contenu de la mémoire ne convient plus, on programme un autre composant.

- ✓ **Les EPROM** (Erasable PROM) sont des PROM effaçables c.a.d qu'on peut modifier leur contenu. Pour le faire il faut retirer EPROM de l'appareil et la soumettre à un rayonnement ultra-violet pour l'effacer. Par contre l'écriture se fait électriquement avec une tension d'alimentation plus élevée.
- ✓ **Les EEPROM** (Electrically EPROM) : Ces mémoires offrent la possibilité d'effacer électriquement le contenu d'un boîtier mémoire tout en restant sur la carte qui le contient, sans être obligé d'extraire les boîtiers pour les exposer à une source d'ultraviolets comme dans le cas des EPROM. Ainsi les EEPROM peuvent être écrites et effacées plusieurs fois (de 100 000 à 1 000 000 de fois) et peut être lue à l'infini [16].

II.1.2. Les Mémoires vives

Contrairement aux mémoires mortes, les mémoires vives sont des mémoires volatiles, ces mémoires nécessitent une tension d'alimentation permanente pour assurer leur fonction de stockage car l'information mémorisée s'efface même en absence très brève de la tension d'alimentation. Dans les mémoires vives adressables, ces informations sont organisées en mots de taille fixe, désignés par une adresse. Les mémoires adressables sont appelées des RAM (Random Acces Memory : mémoire à accès aléatoire) car on peut avoir accès à tous les mots indépendamment de ceux auxquels on a accédé précédemment. Ces mémoires sont généralement définies par une adresse et un mode de lecture ou d'écriture et elles sont généralement utilisées pour le cas ou on manipule des structures de données très grandes, des fonctions récursives ou réentrantes [15].

Suivant la structure des points mémoires, on distingue deux types de RAM : Les RAM statiques (SRAM) et les RAM dynamiques (DRAM).

- ✓ Les **SRAM** sont qualifiées de statiques car elles permettent de garder l'information enregistrée pendant une durée illimitée tant que le circuit est sous tension. Elles peuvent offrir des temps d'accès très courts (quelques ns). Les SRAM sont utilisées généralement pour constituer les éléments mémoires embarqués sur la puce de silicium comme les registres ou les mémoires caches. Elles offrent les meilleures performances mais ont un coût relativement élevé. En plus les SRAM se caractérisent par leur consommation d'énergie plus élevée et leur intégration qui est moindre qu'avec des mémoires dynamiques, ceci est dû à la complexité des points

mémoires qui occupent beaucoup de place, chaque point mémoire est composé d'au moins quatre transistors.

- ✓ Les **DRAM** : sont constituées de cellules élémentaires instables dans laquelle l'information est stockée sous forme de charge électrique dans un condensateur réalisé physiquement par un transistor. C'est la présence du condensateur qui constitue une des limitations de ces mémoires. En effet les condensateurs ne sont pas parfaits et nécessitent un rafraîchissement périodique de leur charge afin de compenser les pertes dues à leur imperfection. Ce rafraîchissement consiste à venir lire la cellule à intervalles fixes et réécrire l'information avant que la charge stockée ne se dégrade totalement. Les constructeurs indiquent que la mémoire doit être rafraîchie toutes les 64 ms [17]. Pour cette raison, ces mémoires sont qualifiées de dynamiques. Si tous les bits d'une mémoire devraient être lus puis réécrits individuellement les DRAM de grande taille seraient en permanence en train d'être rafraîchies et seraient indisponibles pour les processeurs, mais dans la pratique ces mémoires ne sont pas directement reliées aux bus du microprocesseurs mais interfacées par des circuits spécifiques charger de la gestion des accès et des rafraîchissements ainsi la mémoire n'est indisponible que pendant un peu moins de 1% des cycles actifs de la DRAM [17]. En plus la lecture de l'état du condensateur décharge celui-ci, une cellule doit être rechargée à chaque fois qu'on lui accède on lecture. Cela implique un temps d'accès plus long et des temps de latence plus importants.

Ce type de mémoire est plus dense que les mémoires statiques (un transistor par bit), mais il est plus délicat à employer à cause des circuits de rafraîchissement. Au début il n'était pas embarqué sur le silicium mais dernièrement les premières DRAM embarquées ont déjà fait leur apparition et permettent d'avoir des mémoires de grande capacité à un faible coût, au détriment d'un temps d'accès nettement supérieur aux mémoires SRAM.

II.1.3. La mémoire Flash

La mémoire flash est une mémoire à semi-conducteurs, non volatile et réinscriptible, c'est-à-dire une mémoire possédant les caractéristiques d'une mémoire vive mais dont les données ne se volatilisent pas lors d'une mise hors tension. Ainsi la mémoire flash stocke les bits de données dans des cellules de mémoire, mais les données sont conservées en mémoire lorsque l'alimentation électrique est coupée.

En raison de sa vitesse élevée vu que l'écriture et l'effacement se font de manière très rapides, d'où son nom de mémoire flash, de sa durabilité et de sa faible consommation, la mémoire flash est idéale pour de nombreuses applications embarquées et elle est principalement utilisée pour les cartes mémoires des appareils numériques. De plus ce type de mémoire ne possède pas d'éléments mécaniques, ce qui leur confère une grande résistance aux chocs.

II.2. Les nouvelles alternatives des RAM

Les mémoires présentées jusqu'à présent reposent sur le principe de circulation électrique pour la sauvegarde de l'information, de nouvelles technologies sont apparues, ces dernières utilisent la lumière et le magnétisme comme moyen de véhiculer et de stocker l'information.

- ✓ *La mémoire magnétorésistive* MRAM (Magnetoresistive Random Access Memory) est une nouvelle technologie pour les mémoires qui possède le potentiel de devenir la mémoire universelle. C'est une mémoire qui stocke les données en utilisant des charges magnétiques au lieu des charges électriques de la RAM classique. Elle est non volatile même sans tension d'alimentation, elle nécessite une faible alimentation électrique uniquement pour la modification de la polarité des éléments mémoires d'une puce. Les premières mémoires commercialisées en 2006 possèdent les caractéristiques suivantes : cycle de lecture/écriture de seulement 35 nanosecondes, temps d'accès de l'ordre de 10 nanosecondes, les débits sont de l'ordre du Gigabit par seconde et la capacité des chips est de 512 Ko, mais elle devrait fortement augmenter pour atteindre des niveaux plus élevés vers 2010 [18].

  Comparée à la DRAM, le système de consommation de la MRAM est réduit de manière significative, et non volatile il suffit de le couper lorsqu'elle est inactive, et donc ne nécessite pas de refroidissement. De plus, elle dispose d'une grande simplicité d'intégration. Comparée à la SRAM, c'est surtout le coût qui sera retenu car de conception récente sa taille est nettement plus

réduite. Sans oublier qu'elle est non volatile, ce qui par opposition permet de l'utiliser sans disposer d'un système de batteries.

Ce type de mémoire permet d'avoir des mémoires de grande capacité comme celles obtenues avec les DRAM tout en ayant les performances des mémoires SRAM ainsi qu'une très faible consommation [17].

- ✓ *La mémoire holographique* est un autre concurrent désignée comme étant la prochaine génération de stockage optique des données. Avec un temps d'accès de quelques microsecondes, un débit de quelques milliards de bit/s et une capacité de stockage de l'ordre de 1012 octets, la mémoire holographique, pour accomplir ces prodiges, utilise la troisième dimension, les données étant stockées sous forme d'hologrammes à l'intérieur d'un cristal photosensible et non plus seulement en surface comme c'est le cas pour les disques optiques. Contrairement aux mémoires magnétiques leur utilisation ne propose pas de solution basse consommation. En effet une mémoire de ce type est construite grâce à un système coûteux regroupant un laser de très haute précision, une matrice LCD et un capteur CCD. Les équipes IBM prévoient la production des premiers systèmes de stockage holographiques avant 2010 et d'après Inphase Technologies on pourrait enregistrer simultanément jusqu'à un million de bits. La vitesse de transfert atteindrait 160 Mbps, La durée de vie serait supérieure à 50 ans. Le nombre de cycles de lecture serait d'environ 10 millions. Ces mémoires offrent la possibilité de stocker un Téra octet sur un support cristallin de la taille d'un morceau de sucre [18], les recherches actuelles visent à atteindre des performances de 1 To de stockage et 1 Go par seconde de débit et surtout une utilisation possible dans les systèmes embarqués [18].

II.3. La hiérarchie mémoire

Les DRAM sont les mémoires les moins chers à produire en coût par bit. Elles sont utilisées pour avoir de grandes capacités mais ne peuvent pas être utilisées directement pour les échanges avec le processeur lors des calculs car elles constitueraient un sévère goulet d'étranglement en terme de performance.

Cet écart de performance a nécessité la mise en œuvre de techniques de gestion pour réaliser des accès rapides à la mémoire. La hiérarchie mémoire est une solution qui a été proposée très tôt (à la fin des années 60) par les concepteurs de processeurs pour palier la lenteur de la mémoire, comparée aux vitesses grandissantes des processeurs. La mémoire a été hiérarchisée en différents niveaux de tailles et de temps d'accès décroissant au fur et à mesure que l'on se rapproche du processeur (Figure II.1). Les hiérarchies mémoire peuvent être composées de plusieurs niveaux hiérarchiques et distribuées sur plusieurs bus. Le niveau inférieur celui le plus proche du processeur utilise des technologies plus coûteuses, plus petites et plus rapides que le niveau supérieur.

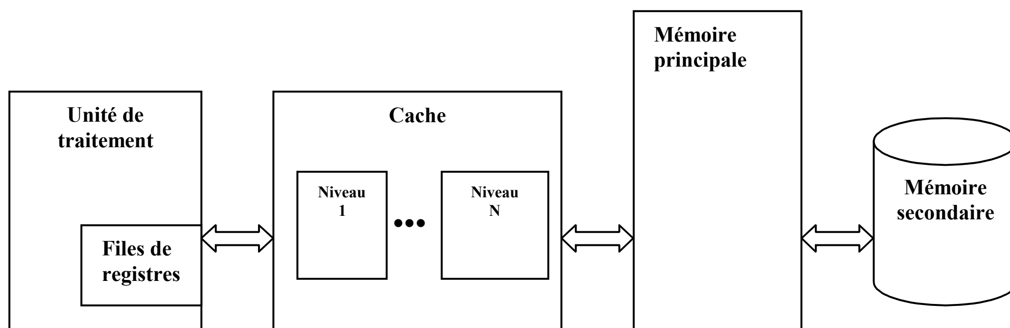


Figure II.1 : Hiérarchie mémoire [3]

Dans une hiérarchie mémoire classique, les données ne peuvent être recopiées d'un niveau à un autre que s'ils sont adjacents. La Figure II.1 représente une architecture simple avec *n* niveaux de hiérarchie mémoire. Elle est composée d'une mémoire cache et d'une mémoire principale.

Chaque requête du processeur est adressée au cache et non directement à la mémoire principale. Si une copie des données est disponible dans le cache il peut répondre immédiatement on dit qu'on a un *hit*. Dans le cas contraire, le cache doit accéder à un niveau supérieur de la hiérarchie, copier le bloc contenant la donnée et ensuite répondre à la requête, dans ce cas on dit qu'on a un défaut de cache

(*miss*). Le principe du cache repose sur les propriétés de *localités temporelles et spatiales* des accès aux données dans les programmes [3].

La *localité spatiale* indique que l'accès à une donnée située à une adresse X va probablement être suivi d'un accès à une zone toute proche de X. De la même manière le principe de *localité temporelle* indique que l'accès à une zone mémoire à un instant donné a de fortes chances de se reproduire dans la suite immédiate du programme, ainsi les données accédées séquentiellement font apparaître la localité spatiale pouvant être exploitée dans la gestion des accès. Par contre les programmes contiennent pour la plupart des boucles et il est donc fortement probable que les instructions et les données sont accédées de façon répétée, ce qui génère une localité temporelle élevée [17].

II.3.1. Approches de conception d'une hiérarchie mémoire

Un cache est une petite mémoire qui possède des temps d'accès relativement rapides par rapport à la mémoire centrale. Le cache est conçu spécialement pour alléger l'écart des performances entre la mémoire et le processeur. La mémoire centrale est généralement de type DRAM alors que le cache est de type SRAM. Le coût d'un accès à une donnée dans l'hiérarchie dépend du niveau auquel on accède. Un cache est décomposé en blocs qui contiennent un ensemble de données fixes. Les transferts entre deux niveaux sont effectués par blocs appelés aussi ligne du cache afin de prendre en compte la localité spatiale, la localité temporelle étant gérée par la politique de remplacement de ces blocs. Les méthodes mises en œuvre pour gérer le remplacement des blocs jouent un rôle primordial dans les performances d'une hiérarchie.

L'objectif de la conception d'une hiérarchie mémoire est double. D'une part, il s'agit de réduire les temps d'accès à la mémoire pour diminuer la latence des applications ; d'autre part, il s'agit de réduire la puissance dissipée par accès notamment pour les applications embarquées.

Les approches proposées dans la littérature pour la conception d'un cache peuvent être classées en trois catégories [17] :

- ✓ Mémoire cache gérée de façon matérielle.
- ✓ Mémoire cache gérée de façon logicielle.
- ✓ Hiérarchie mémoire dédiée.

II.3.1.1. Mémoire cache gérée de façon matérielle

Ce type de cache repose sur l'utilisation de traces d'exécution des programmes et produit une hiérarchie mémoire dédiée à une application. Il représente le modèle le plus couramment utilisé dans les architectures modernes.

Une mémoire cache étant plus petite que la mémoire de niveau supérieur, donc elle ne peut contenir tout le contenu de la mémoire principale, il faut alors définir une méthode indiquant à quelle adresse de la mémoire cache doit être écrit un bloc de la mémoire principale, son fonctionnement est par conséquent contraint par la gestion des blocs du cache par rapport à l'espace d'adressage réel disponible dans la mémoire.

On distingue trois types de gestion des accès à la mémoire cache. Ils se définissent suivant leur degré d'associativité du cache.

- **Le cache à adressage direct** (*Direct Mapped Cache*) :

Chaque bloc de la mémoire principale ne peut être enregistré qu'à une seule adresse de la mémoire cache. Ceci crée de nombreux défauts de cache conflictuels si le programme accède à des données qui sont mappées sur les mêmes adresses de la mémoire cache. La sélection du bloc où la donnée sera enregistrée est habituellement obtenue par [19]:

bloc = (Adresse mémoire) mod (Nombre de blocs).

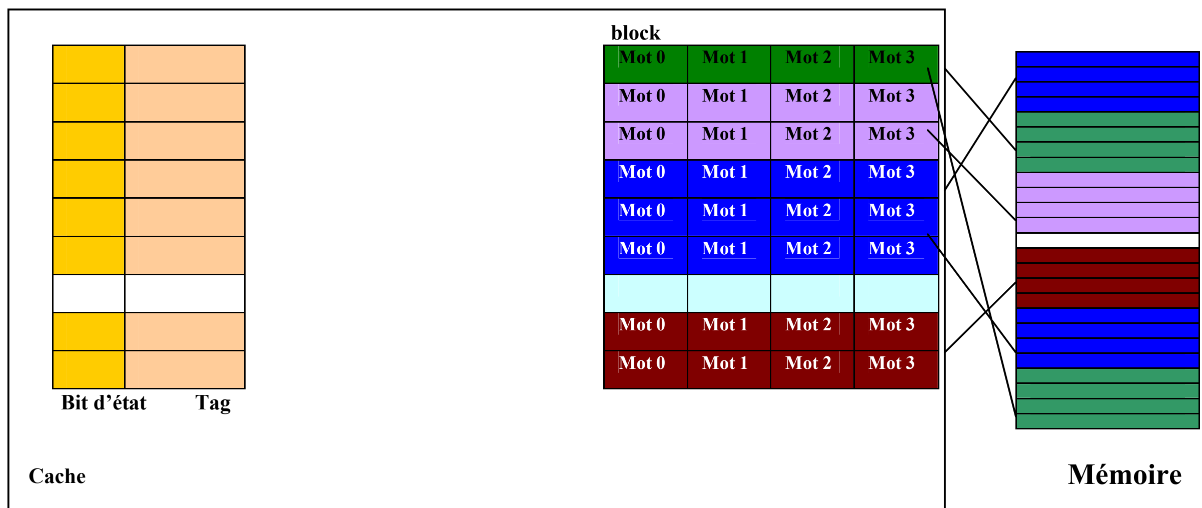


Figure II.2 : Cache à adressage direct

Un bloc du cache est partagé par de nombreuses adresses de la mémoire de niveau supérieur. Il nous faut donc un moyen de savoir quelle donnée est actuellement dans le cache. Cette information est

donnée par le tag, qui est une information supplémentaire stockée dans le cache. L'index correspond au bloc où est enregistrée la donnée. Un bit additionnel (appelé bit de validité ou d'état) est chargé de savoir si une adresse donnée contient une donnée ou non.

L'adressage direct est une stratégie simple mais peu efficace car elle crée de nombreux défauts de cache conflictuels. Une solution est de permettre à une adresse de la mémoire principale d'être enregistrée à un nombre limité d'adresses de la mémoire cache. Cette solution est présentée par le cache associatif par ensembles.

-Le cache associatif par ensembles (Set associative Cache) : Dans un tel cache un bloc mémoire peut être copié dans un ensemble de blocs du cache. Les caches associatifs par ensemble ont généralement des ensembles de 2, 4 ou 8 blocs.

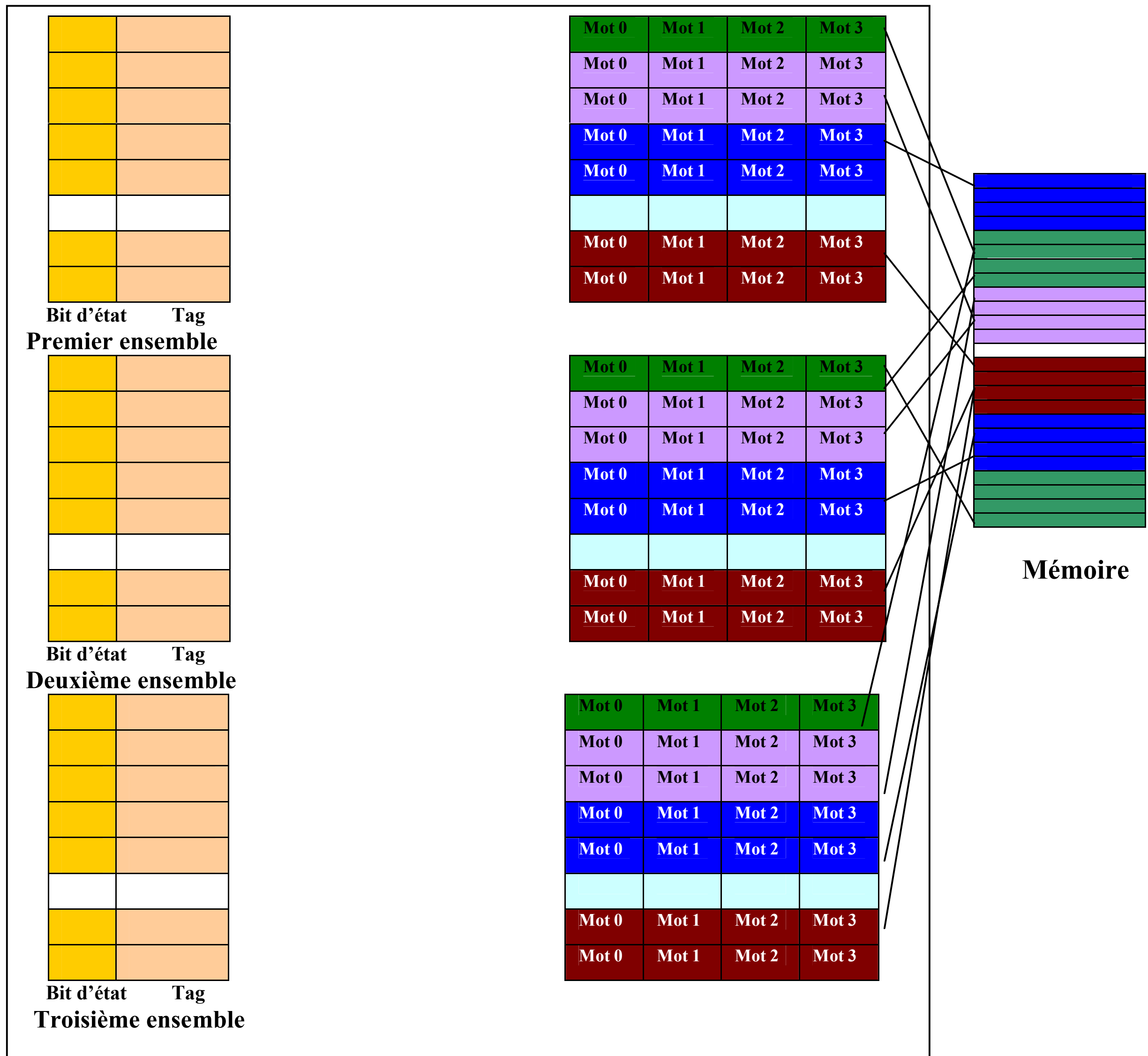


Figure II.3 : Cache associatif par ensembles

Il s'agit d'un compromis entre l'adressage direct et l'adressage complètement associatif, qui sera décrit par la suite, essayant d'allier la simplicité de l'un et l'efficacité de l'autre.

La mémoire cache est divisée en ensembles (sets) de N blocs du cache. Un bloc de la mémoire de niveau supérieur est affecté à un ensemble, il peut par conséquent être écrit dans n'importe quel ensemble. Ceci permet d'éviter de nombreux défauts de cache conflictuels. À l'intérieur d'un ensemble, l'adressage est complètement associatif (ceci explique le nom de cette technique). En général, la sélection de l'ensemble est effectuée par [19]:

Ensemble = (Adresse mémoire) mod (Nombre d'ensembles).

- **Le cache complètement associatif** (Fully Associative Cache) : Dans ce type de cache, chaque bloc mémoire peut être copié dans n'importe quel bloc du cache.

La complexité de gestion du cache augmente avec son degré d'associativité [17]. En effet, plus le degré d'associativité augmente, plus il faut effectuer une recherche complexe pour savoir si une donnée est présente dans le cache. De même, le coût matériel et la consommation en puissance des caches augmentent avec l'associativité.

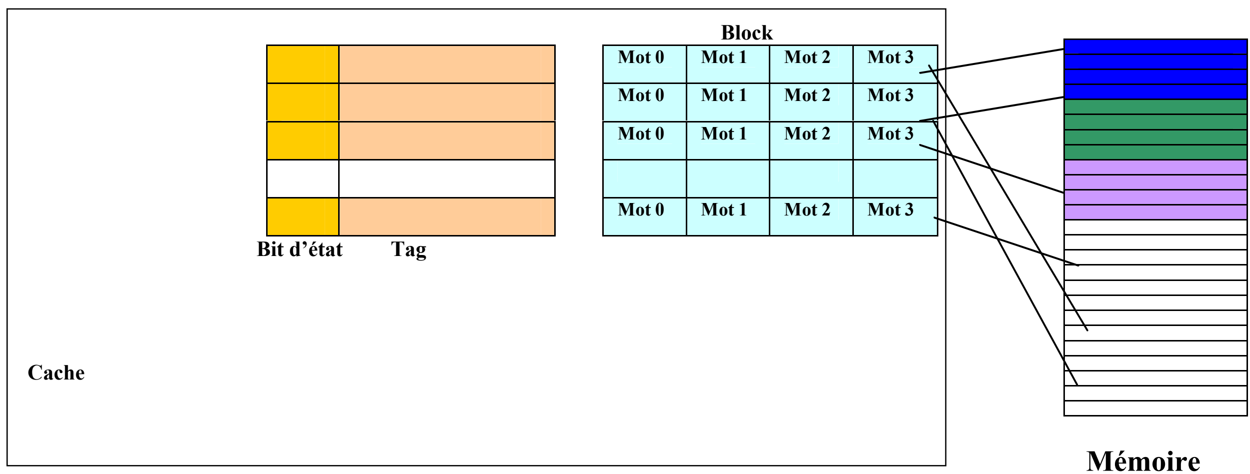


Figure II.4 : Cache complètement associatif

Ce type de cache requiert beaucoup de logique car il donne accès à de nombreuses possibilités. Ceci explique pourquoi l'associativité complète n'est utilisée que dans les mémoires cache de petite taille (typiquement de l'ordre de quelques kilos octets).

Lorsque l'unité de traitement demande une donnée qui n'est pas présente dans le cache, il y a un défaut de cache. Les défauts de cache peuvent être de trois types : *les défauts obligatoires de chargement*, *les défauts de capacité* et *les défauts de conflit* [17].

- **Défauts obligatoires de chargement** (*Compulsory miss ou Cold miss*): ce sont des défauts qui ne peuvent être évités. Le programme accède à des données qui n'ont pas encore été accédées, il faut alors les charger dans le cache pour leur première utilisation.

- **Défauts de capacité** (*Capacity miss*): le cache ne peut pas contenir toutes les données nécessaires à l'exécution d'un programme. Il faut enlever des données du cache pour les remplacer par de nouvelles données ce qui génère les défauts de capacité.

- **Défauts conflictuel** (*Conflictual miss*): ce type de défauts apparaît, pour les caches à adressage direct ou associatif par ensembles, lorsque l'unité de traitement requiert un nombre de blocs qui doivent être mappés dans un même ensemble et dont la taille est supérieure à cet ensemble. En général, le nombre de défauts conflictuels est prépondérant par rapport au nombre de défauts de capacités sauf pour les caches complètement associatifs où ils n'existent pas.

Un cache complètement associatif permet d'améliorer les performances en termes d'accès aux données puisqu'il supprime les défauts conflictuels. Cependant l'amélioration de ces performances doit être mise en balance avec le surcoût en matériel et en consommation introduit par la logique de contrôle et de recherche de blocs présents dans le cache [3].

Pour les caches associatifs par ensemble ou complètement associatifs les optimisations obtenues par la présence d'un cache ne sont fiables que si les techniques de remplacement des blocs lors d'un défaut de cache sont efficaces. Le choix des blocs à remplacer dans un cache peut être géré par différents algorithmes de remplacement.

- **Le remplacement de l'élément le plus ancien** (*Last Recently used – LRU*) : les dates d'accès aux blocs du cache sont maintenues dans une table dans le cache. Lorsqu'un défaut apparaît, le bloc ayant la date d'utilisation la plus ancienne est remplacée.

- **Le remplacement de l'élément le moins utilisé** (*Least Frequently Used – LFU*) : le taux d'accès aux blocs est conservé dans le cache et le bloc ayant le taux le plus faible est remplacé lors d'un défaut.

- **Le remplacement suivant l'ordre d'ancienneté** (*First In First Out – FIFO*) : le remplacement des blocs s'effectue suivant l'ordre dans lequel ils sont entrés dans le cache.

- **Le remplacement aléatoire** : les blocs candidats au remplacement sont choisis de manière aléatoire de façon à obtenir une répartition uniforme.

Pour gérer les écritures, le cache doit permettre d'assurer la cohérence des données entre les données contenues dans le cache et celles contenues dans les niveaux supérieurs de la hiérarchie.

Deux cas sont à distinguer. Pour le premier, l'écriture a pour cible un emplacement mémoire dont une copie est disponible en mémoire cache. Pour le second, il n'existe pas de copie et l'écriture provoque un défaut. Lorsqu'une copie existe, les caches peuvent avoir les comportements suivants :

- **Ecriture simultanée** (Hit Write Through) : les écritures sont effectuées à la fois en cache et en mémoire, ce qui garantit la cohérence des données entre le cache et la mémoire principale. Ainsi la mémoire centrale contient toujours les mêmes données que le cache, ceci est très important dans le cas ou il y'a d'autres dispositifs qui accèdent à la mémoire centrale [20].

- **Ecriture différée** (Hit Write Back) : Les écritures ne sont effectuées que dans le cache. Il n'y a réécriture du bloc modifié en mémoire principale que si celui ci doit être remplacé dans le cache afin de garantir la cohérence. Au même moment, un bit flag correspondant au bloc du cache est mis à jour pour indiquer que le bloc est modifié [20].

Dans le deuxième cas ou la donnée correspondante est non disponible dans le cache, deux politiques de gestion des écritures peuvent être mises en place.

- ***Miss fetch on Write*** : méthode aussi connue sous le nom de *write allocat* , le bloc dans lequel doit se faire l'écriture est d'abord ramené en mémoire cache avant d'être modifié. Ce mécanisme entraîne une pénalité de défaut d'écriture qui est la même que celle d'un défaut de lecture puisqu'il faut lire un bloc complet depuis le niveau hiérarchique supérieur [20].

- ***Miss write around*** : les données sont directement écrites en mémoire principale sans passer par le cache. Cette technique peut être rapide si le bloc considéré n'est pas réutilisé plus tard.

Les paramètres de fabrication d'une mémoire cache gérée de façon matérielle ont une influence importante dans la conception d'une architecture pour un système embarqué. Il convient de construire une architecture avec soin pour limiter les défauts de cache et réduire le nombre de transferts entre les différents niveaux [17].

II.3.1.2. Mémoire cache gérée de façon logicielle

Les caches matériels restent utilisés dans la majorité des cas car ils sont les plus faciles à mettre en place. Cependant, pour des applications dont le code est tout ou partie analysable statiquement, les caches matériels ne sont pas forcément les plus efficaces et sûrement pas les moins consommateurs en termes de puissance. Ce constat a entraîné le développement de caches gérés partiellement de façon logicielle. Les caches logiciels ont la même structure de blocs que les caches matériels mais la gestion des chargements et des remplacements est à la charge du programme. La gestion logicielle du cache est réalisée par le compilateur ou à défaut d'outils par le programmeur [17].

La complexité matérielle nécessaire à la mise en œuvre des caches logiciels est bien moins importante que pour les caches matériels car on élimine toute la circuiterie nécessaire au contrôle ce qui les rends moins consommateurs en énergie.

De plus, les caches matériels gèrent les transferts selon l'ordre d'exécution et des mesures statistiques fixes. En comparaison, les caches logiciels utilisent les directives provenant du compilateur ou du programmeur pour assurer une bonne gestion dynamique dédiée à l'application.

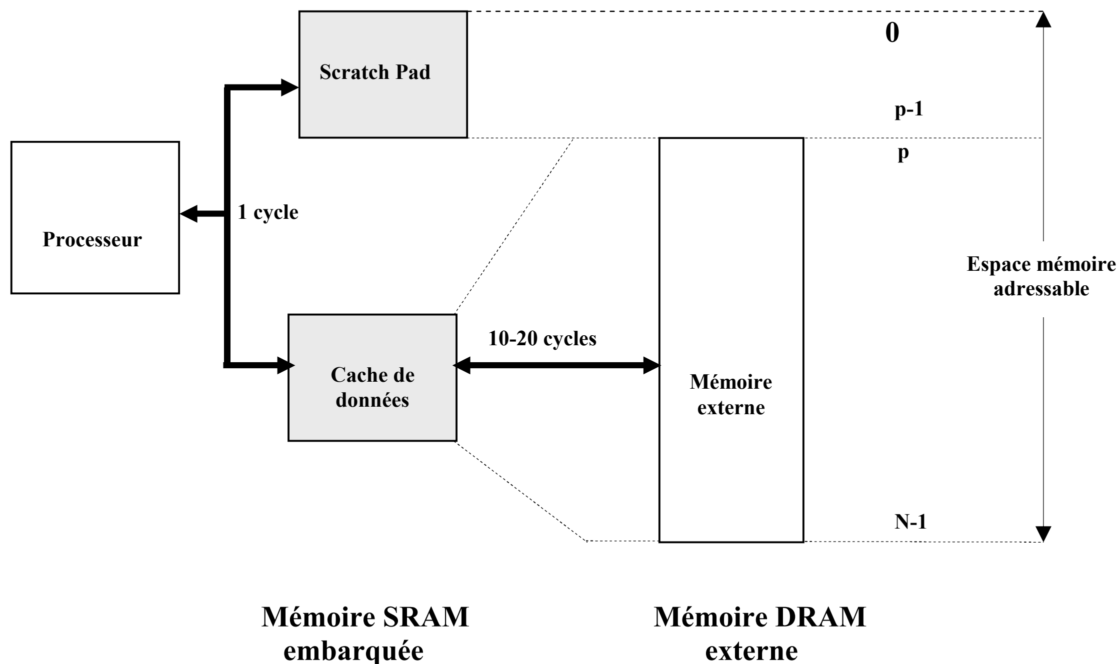
La plus grande différence entre le cache matériel et logiciel réside dans la politique de mise à jour des données, c'est à dire la façon de gérer la cohérence des données. Au niveau matériel, une écriture s'effectue au niveau de hiérarchie mémoire supérieur à chaque apparition d'une écriture ou lorsqu'un bloc est évincé du cache. En logiciel, c'est le compilateur qui décide à quel moment et si oui ou non une écriture doit être effectuée à un niveau hiérarchique supérieur. Cette gestion permet de réduire le nombre de transferts entre les différents niveaux de la hiérarchie mémoire ce qui entraîne une réduction de la bande passante nécessaire et de la puissance dissipée due aux transferts.

La présence d'un cache géré par logiciel présente de nombreux avantages, mais cette utilisation nécessite une analyse importante du programme et des outils de compilation adaptés [17].

Les mémoires scratch-pads

Les mémoires scratch-pads (ou bloc notes) font référence aux mémoires de données embarquées, elles se situent a mi chemin entre les mémoires caches matérielles et les mémoires caches logicielles formant ainsi une hiérarchie qui combine les solutions proposées précédemment.

Elles ont été introduites par Panda [21]. La mémoire embarquée est constituée d'une mémoire cache standard et d'une mémoire SRAM appelée Scratch-pad. Les espaces d'adressage de la scratch-pad et celui de la mémoire externe sont disjoints mais intégrés dans l'espace d'adressage total du système. Cependant, la mémoire scratch-pad est connectée au même bus d'adresses et au même bus de données, la mémoire cache et la scratch-pad autorisent des accès rapides aux données, mais la mémoire scratch-pad garantit un temps d'accès d'un seul cycle alors que la mémoire cache est sujette à défauts de cache. Ce type de mémoire permet de stoker dans une mémoire ayant les performances d'une mémoire cache les données dont l'utilisation provoquerait un grand nombre de défauts de cache.



Mémoire SRAM embarquée **Mémoire DRAM externe**

Figure II.5 : Mémoire Scratch pad [21]

Il ne s'agit pas ici d'un cache logiciel mais d'une mémoire embarquée dédiée à des données utilisées de manière intensive. Cette mémoire augmente la surface des SOC mais permet d'avoir un gain très important en performance et en consommation.

Le contenu d'une mémoire scratch-pad est décidé au moment de la compilation et de ce fait est totalement contrôlé par l'utilisateur. A l'inverse, la gestion des transferts de données dans un cache se fait lors de l'exécution. Donc pour des applications dont une analyse complète ne peut être effectuée au moment de la compilation une solution intégrant une mémoire scratch-pad est plus complexe. Pour ces applications, il est préférable d'utiliser un cache "blocable" qui combine les bénéfices d'une mémoire SRAM de type scratch-pad et d'un cache géré matériellement. La partie ne pouvant être analysée de manière statique est gérée par le contrôleur matériel du cache, et la partie pouvant être analysée statiquement est bloquée dans la scratch-pad par le compilateur.

II.3.1.3. Création d'une hiérarchie mémoire dédiée

La mise en place d'une hiérarchie mémoire dédiée est un processus complexe et dépendant de manière étroite du type d'application traitée. Le choix des paramètres architecturaux relève de nombreux compromis effectués entre performance, consommation et souplesse d'adaptation.

L'hiérarchie mémoires dédiée la plus populaire correspondant à la séparation des transferts des données et des instructions est le cache de données et le cache d'instructions.

- **Mise en place d'un cache de données :** C'est un buffer rapide qui contient les données de l'application. Avant que le processeur n'utilise les données, celles-ci doivent être chargées dans le cache. La sélection des données du cache se base sur la localité temporelle et la localité spatiale des données. Les techniques proposées à ce niveau se basent sur la réorganisation et la transformation du code source pour renforcer la localité des données.
- **Mise en place d'un cache d'instructions :** Un processeur doit pouvoir disposer à chaque cycle d'horloge d'une nouvelle instruction, les caches pour les instructions ne sont en général séparés des caches de données que pour le premier niveau, celui le plus proche du processeur. Cette séparation permet d'utiliser toute la bande passante de chacun des deux caches et ainsi permettre l'alimentation soutenue des processeurs en instruction et donnée.

Les caches d'instructions sont gérés de façon matérielle et ont des structures adaptées aux mécanismes de contrôle des programmes. On trouve par exemple deux types de caches utilisés couramment dans les systèmes [17] :

- ✓ Les caches d'instructions prédécodées permettent de conserver les instructions les plus exécutées dans un format permettant de les utiliser sans avoir recours aux étapes de chargement et de décodage des instructions.
- ✓ Les caches de boucles permettent de garder en mémoire les boucles des programmes. De tels caches sont moins efficaces que les précédents pour les programmes ayant des propriétés de localité très élevées sur certaines instructions, mais sont plus souples.

II.3.2. Mesures de performances du cache

Les performances du cache peuvent être mesurées par plusieurs façons, on présente dans ce qui suit les mesures de performance utilisées dans le domaine des systèmes embarqués :

a. **Taux des défauts de cache (Miss rate)** : Le taux des défauts de cache est le pourcentage des accès aux données qui ne sont pas disponibles dans le cache, c.à.d. les accès provoquant un accès à un niveau supérieur pour le chargement des données. La réduction du taux des défauts du cache engendre une amélioration du temps d'accès et de l'énergie.

b. **Temps d'accès au cache** : Le temps d'accès au cache est le nombre moyen de cycles d'horloge nécessaires pour accéder avec succès à une adresse référencée. Ce paramètre est utile lors de l'évaluation des performances du cache car quoiqu'une conception de cache particulière puisse présenter un faible taux de cache, celle-ci peut être réalisée au détriment du temps d'accès. C'est le cas d'un cache ayant une forte associativité et qui peut avoir un taux de défauts plus faible qu'un cache à adressage direct mais le cache associatif aura un temps d'accès plus lent que celui du cache à adressage direct.

c. **Analyse d'espace** : Les concepteurs des systèmes embarqués s'intéressent non seulement aux performances mais aussi à une meilleure utilisation de l'espace du circuit. Ce paramètre croit avec la croissance des deux paramètres : la taille du cache et l'associativité.

d. **Consommation d'énergie** : Ce paramètre donne la quantité d'énergie consommée par le cache lors de l'exécution d'une application, c'est un paramètre critique dans les systèmes embarqués surtout pour les systèmes alimentés par batterie. Ce paramètre croit avec la croissance des défauts de cache engendrant des accès à la mémoire externe qui consomment plus d'énergie, d'un autre côté une haute associativité accroit l'énergie consommée lors d'un accès au cache. [1] montre qu'une réduction de l'énergie de consommation est observée lors d'une diminution des défauts du cache.

II.3.3. Interaction et influence des différents paramètres

- **Interaction entre l'espace, la performance et la consommation d'énergie d'un cache**

Il existe une forte interaction entre l'espace, la performance et la consommation d'énergie d'un cache, un cache de petite taille consomme moins d'énergie qu'un cache de grande taille, de même un cache rapide réduit potentiellement la consommation d'énergie globale car il permet d'effectuer l'exécution en un temps réduit [22]. La Figure II.6 prise de [22] montre la relation entre la consommation et la capacité du cache.

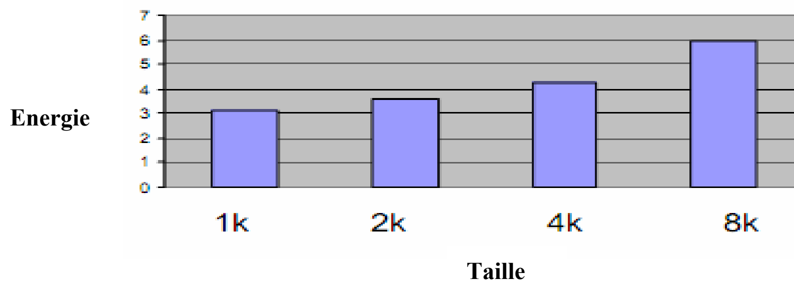


Figure II.6: Croissance de la consommation avec la capacité

Les résultats de l'étude [1] montrent qu'une réduction de la consommation ainsi qu'une amélioration des performances du système sont observées lors de la réduction du taux de défauts du cache.

D'une autre part le taux de défauts du cache diminue lorsqu'on augmente la taille du cache, ceci est vrai soit pour le cache des données ou le cache d'instructions [22].

Optimiser en même temps les trois paramètres espace, énergie et performance est une opération très difficile car certains paramètres sont conflictuels, pour la majorité des applications réduire la taille du cache engendre une croissance des défauts du cache menant à des accès excessifs à la mémoire

centrale donnant lieu à une croissance du temps d'accès et de consommation d'énergie. En plus une réduction du temps d'exécution influe proportionnellement sur l'énergie consommée.

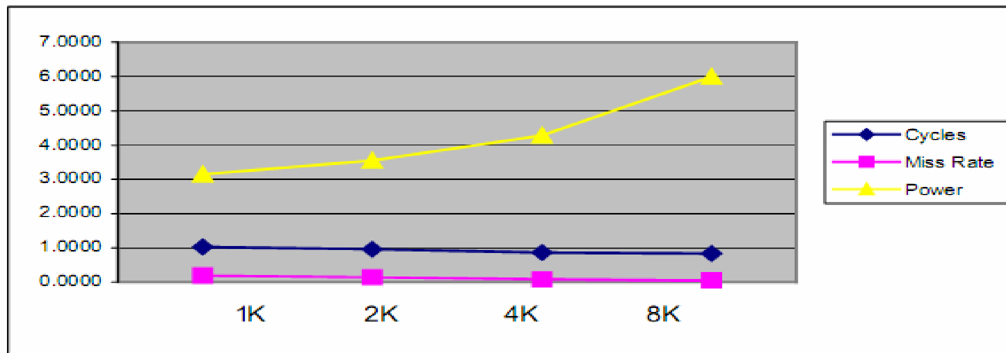


Figure II.7 : Variation de la consommation d'énergie, cycle moyen d'exécution et le taux des défauts avec l'augmentation de la capacité du cache [22].

- **Influence de l'associativité**

Un cache associatif consomme plus d'énergie et plus d'espace. Malgré qu'ils minimisent le taux des défauts, ils requièrent dans la majorité des cas des recherches dans les ensembles consommant plus d'énergie. L'étude menée dans [1] et [22] montre qu'un cache ayant une haute associativité minimise le nombre de défauts du cache mais en contre partie il augmente la complexité de l'architecture du cache, le temps d'accès et l'espace du cache.

II.4. Techniques d'optimisation du cache dans les systèmes embarqués

Pour améliorer les performances des mémoires les concepteurs ont tenté de nombreuses techniques et méthodes, la littérature fournit toutes une gamme de méthodes et à tous les niveaux d'abstraction du flot de conception. Pour masquer les basses performances des mémoires les chercheurs ont constatés qu'ils peuvent intervenir soit au niveau programme, soit au niveau architecture donnant ainsi naissance à deux types de méthodes, celles qui interviennent au niveau code en effectuant des transformations sur le code, généralement cette intervention s'effectue lors de la compilation et garantie une amélioration des caractéristiques de la mémoire et celles qui interviennent au niveau architecture. Ce dernier type est désigné spécialement pour tenir compte de l'architecture mémoire à différents niveaux de granularité, en commençant des registres et les files de registres aux SRAM, Cache, et DRAM.

L'interface CPU-cache est une architecture connue, où les concepteurs du système peuvent bénéficier des décennies de progrès de recherche en compilation. Récemment les chercheurs ont adressés différents problèmes reliés à l'interface du cache et CPU dans les systèmes embarqués vu que ce domaine présente de nouvelles opportunités d'optimisations à cause de [23]:

1. *La flexibilité de l'architecture* : Plusieurs paramètres de conception peuvent être personnalisés pour s'adapter aux besoins de l'application exemple la taille du cache.
2. *Long temps de compilation disponible* : Plusieurs optimisations agressives peuvent être réalisées car plus de temps de compilation est maintenant disponible.
3. *Connaissance complète de l'application* : La supposition que le compilateur a accès à l'application entière nous permet de réaliser plusieurs optimisations globales négligées par les compilateurs traditionnels exemple changement de la disposition des données.

Malgré ses opportunités, les défis de conception des systèmes embarqués sont plus rigoureux que ceux des systèmes à usage général. Généralement les systèmes à usage général offre une plus grande flexibilité lors de la conception des caches en terme de : taille du cache, taille du bloc du cache, associativité et les caches multi niveaux, alors que les systèmes embarqués sont limités par les contraintes d'espace, énergie, temps réel …etc, ce qui rend l'exploitation des caches dans ces systèmes très délicate. Il est donc très important de prendre en compte les spécificités des systèmes embarqués lors du choix d'une hiérarchie mémoire. Des études ont montrés que 50% de l'énergie totale du processeur est consommée par le cache [22], de ce fait la réduction de la consommation d'énergie du cache peut réduire significativement la consommation d'énergie totale du circuit surtout pour les systèmes alimentés par batterie.

Généralement les concepteurs de cache pour système à usage général sélectionnent des paramètres fixes de cache tels que : la capacité du cache, taille du bloc du cache, associativité, visant une performance moyenne du système à cause des variétés d'applications pouvant s'exécuter sur la machine. Cette philosophie de « one size fits all » est inefficace pour les systèmes embarqués et chaque application doit avoir son propre cache disposant de paramètres dédiés et spécifiés selon les caractéristiques de l'application ce qui permet d'atteindre un maximum de performance tout en respectant les contraintes du système.

On présente dans ce qui suit, les principales méthodes d'optimisation du cache et leur impact sur les systèmes embarqués. A noter qu'on s'intéresse seulement aux méthodes visant le cache de données

II.4.1. Augmentation de la capacité du cache et la taille du bloc du cache

La méthode la plus simple pour réduire les défauts du cache est d'utiliser un bloc de taille suffisamment large, ceci permet de réduire les défauts de chargement et d'améliorer le préchargement des données [24]. A signaler que l'augmentation de la taille du bloc du cache sans l'augmentation de la capacité du cache conduit à une croissance des défauts de conflits et de capacités car l'augmentation de la taille du bloc du cache sans affecter celle du cache réduit le nombre de blocs du cache.

Cette technique ne donne pas toujours des résultats satisfaisants pour les systèmes embarqués vu qu'ils sont contraints à un espace restreint et une consommation limitée donc ils ne peuvent contenir des caches de grandes tailles et non plus des caches ayant des tailles de blocs larges [20].

II.4.2. Augmentation de l'associativité

La technique de l'augmentation de l'associativité est une autre technique conçue spécialement pour réduire le taux de défauts du cache [20]. Les caches associatifs par ensemble de 4 ou 8 blocs sont devenus très familiers dans les systèmes actuels. Le cache à adressage direct est simple, facile à concevoir et requiert moins d'espace que les caches associatif par ensembles. L'inconvénient majeur des caches à adressage direct est le taux de défauts de conflits élevé, ce taux représente 40% des défauts du cache [22]. Par contre les caches ayant une forte associativité sont caractérisés par un faible taux de défauts, mais en contre partie ils présentent un temps d'accès au cache élevé, une croissance de complexité de l'architecture du cache ainsi qu'une croissance de l'espace réservé au cache [1]. Généralement l'associativité est nécessaire seulement pour les blocs conflictuels.

Les caches ayant une forte associativité restent non populaires pour les systèmes embarqués car une forte associativité induit une grande consommation d'énergie. Ceci justifie le grand effort de recherche réalisé sur les caches à adressage direct dans les systèmes embarqués.

II.4.3. Le cache victime (victim cache)

Le cache victime est un cache complètement associatif de 4 à 16 blocs qui réside entre le premier niveau du cache caractérisé par un adressage direct et le second niveau de l'hiérarchie mémoire. Quand un défaut de cache survient dans le cache primaire, le cache victime est consulté en premier lieu, si les données recherchées existent dans le cache victime alors elles seront envoyées au processeur et en même temps elles seront placées dans le cache primaire remplaçant ainsi des données existantes qui seront swappées vers le cache victime. Si par contre un défaut survient dans le cache victime, un niveau supérieur est consulté et les données seront transférées vers le cache primaire pour remplacer des données qui vont être à leur tour placées dans le cache victime.

Le cache victime réduit les défauts conflictuels des caches à adressage direct sans affecter leur temps d'accès rapide. Puisque les caches victimes sont complètement associatifs et bien qu'ils aient une faible taille, ils peuvent conserver plusieurs blocs simultanément, ces derniers peuvent être source de défauts conflictuels pour le cache à adressage direct. Si tous les blocs conflictuels peuvent être placés dans le cache victime le taux des défauts et le temps moyen d'accès sont améliorés d'une manière spectaculaire [22].

Le cache victime présente certaines lacunes qui pénalisent son utilisation pour les systèmes embarqués en effet il requière une consommation d'énergie supplémentaire et un espace additionnel.

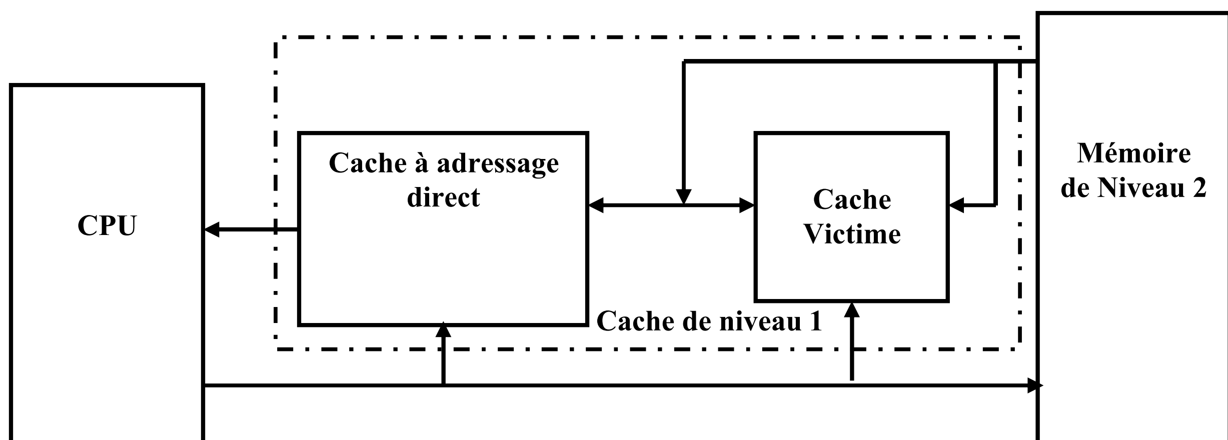


Figure II.8 : Hiérarchie mémoire avec inclusion du cache victime [1]

II.4.4. Cache partitionné

Le partitionnement du cache n'est pas une nouvelle idée, les processeurs modernes reposent sur les architectures du cache partitionné au moins au premier niveau du cache en séparant le cache d'instruction du cache des données. Le cache conventionnel n'implique aucune séparation basée sur la nature des localités des données et il gère d'une manière uniforme toutes les références mémoires, cette gestion consiste à charger un bloc mémoire à chaque défaut de cache pour remplacer un autre bloc déjà existant dans le cache, mais les variables dans les applications réelles présentent une variété de modèle d'accès et de types de localités. Par exemple les scalaires comme les indexes généralement présentent une haute localité temporelle et une localité spatiale moyenne alors que les vecteurs ayant un petit pas représentent une haute localité spatiale, par contre les vecteurs ayant un grand pas présentent une faible localité spatiale et peuvent ou non présenter une localité temporelle. Par conséquent le traitement traditionnel de toutes les références dans un cache de manière uniforme rend le cache de données inefficace.

Plusieurs chercheurs ont proposé de partitionner le cache en cache spatial et cache temporel pour stocker les données ayant respectivement une haute localité spatiale et temporelle. Parmi ces approches celle décrite dans [25] basée sur une prédiction dynamique pour router les données soit vers le cache matériel soit vers le cache logiciel basé sur un buffer d'historique et celle décrite dans [26] qui utilise un cache divisé en cache temporel et un cache spatial mais alloue les variables statiquement aux différents modules mémoires locales en minimisant l'espace et l'énergie générés par le mécanisme de prédiction dynamique.

Ce type de méthode sera détaillé dans le chapitre III.

II.4.5. La mémoire scratch pad

Un concepteur de système embarqué n'est pas limité à l'utilisation d'une mémoire conventionnelle, vu que le système embarqué est désigné à exécuter seulement une application d'une manière répétitive, on peut alors utiliser une variation d'architecture non conventionnelle qui convienne à une application spécifique sous certaine considération. Parmi ces alternatives de conception on a le scratch pad proposé par Panda et al [21].

Le concept de la mémoire scratch pad est d'une considération importante dans les systèmes embarqués modernes.

Voici un exemple [25] qui éclaircisse le rôle que peut jouer le scratch pad dans l'amélioration des performances d'une hiérarchie mémoire.

Exemple

Soit une matrice 4,4 de coefficients, *mask*, qui se glisse sur une image en entrée, *source*, couvrant à chaque itération une région 4,4 différente comme sur la figure II.9. A chaque itération les coefficients du masque sont combinés avec la région de l'image couverte couramment pour obtenir une moyenne pondérée, et le résultat, *acc*, est affecté au pixel du tableau en sortie, *dest*, dans le centre de la région couverte. Si les deux tableaux *source* et *mask* vont être accédés via le cache de données alors les performances vont être affectées par les conflits du cache. Ce problème peut être résolu en stockant le petit tableau *mask* dans le scratch pas. Cette affectation élimine tous les conflits du cache de données, maintenant le cache est utilisé pour les accès au tableau *source* qui sont très réguliers. En stockant le tableau mask dans le scratch pad on assure que les données accéder fréquemment ne sont jamais éjectées hors le circuit off chip. Ainsi on améliore significativement les performances de la mémoire et la consommation d'énergie.

```
# Define N 128
# Define M 4
# Define NORM 16
Int source [N] [N], dest [N] [N] ;
Int mask [M] [M];
Int acc, I , j ,x, y;
.
.
For (x=0 ; x< N-M ; x++)
        For (y=0 ; y< N-M ; y++) {
                Acc=0;
                For (i=0 ; i< M ; i++)
                        For (j=0 ; j<M ; j++)
                                Acc= acc + source [x+i][y+j] * mask [i][j];
                                Dest[x+m/2]= acc/NORM;
}
```

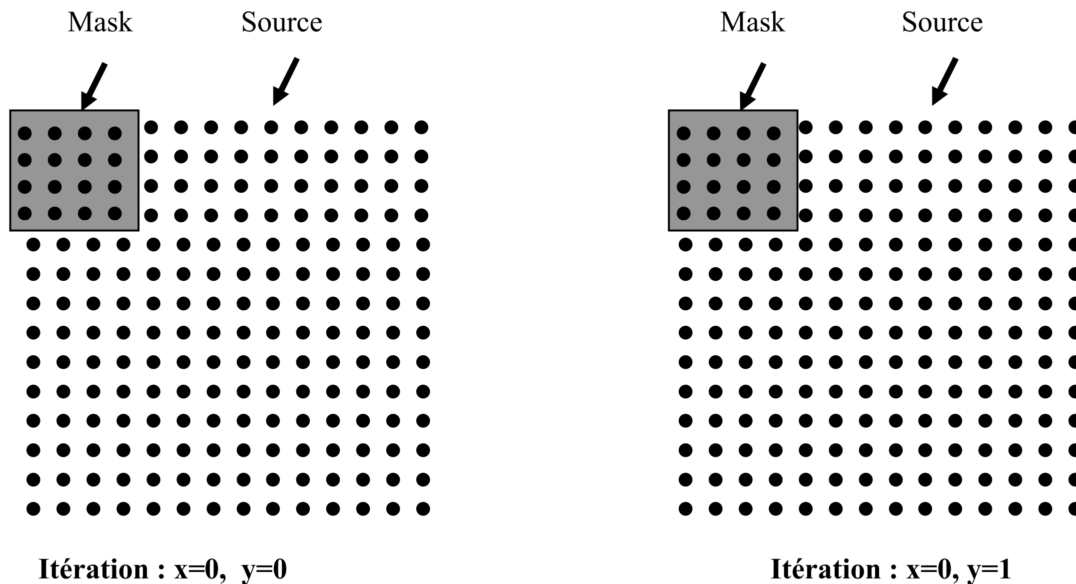


Figure II.9 : Exemple de cas d'utilisation du scratchpad

La technique d'affectation de la mémoire proposée par Panda et al [21] détermine au début le facteur de conflit total appelé TCF (Total Factor Conflit) pour chaque tableau, basé sur la fréquence d'accès et les possibilités de conflit avec les autres tableaux, puis calcule le TCF/ (la taille du tableau) pour déterminer quel tableau doit être affecté au scratch pad, leur choix sera basé en donnant la priorité au Haut conflit/petite taille du tableau.

II.4.6. Le préchargement

Le préchargement est une technique utilisée pour améliorer l'efficacité du cache, généralement l'augmentation de la taille du bloc du cache représente le moyen le plus simple pour réaliser le préchargement[20]. Ceci peut être réalisé soit par hardware ou par software, le préchargement basé sur hardware nécessite un hardware supplémentaire connecté au cache alors que le préchargement software repose sur la technologie des compilateurs pour insérer explicitement des instructions de préchargement. Pour les données exposant une localité spatiale le préchargement est très bénéfique.

Le préchargement doit être réalisé délicatement lors de la conception des caches pour systèmes embarqués car un préchargement non nécessaire engendre une perte d'énergie et peut être source du phénomène de pollution du cache (suppression du cache de données utiles pour l'exécution du programme).

II.4.7. Cache multi niveaux

L'ajout d'un cache supplémentaire dans l'hiérarchie mémoire est une technique utilisée pour améliorer les performances des systèmes. Généralement il est inséré entre le cache primaire et la mémoire principale, des systèmes peuvent avoir plusieurs niveaux de caches. Le cache du premier niveau L1 a généralement une petite taille lui permettant de s'harmoniser avec le cycle du processeur alors que le niveau L2 peut être large suffisamment pour contenir des données et réduire ainsi le temps d'accès.

L'ajout d'un niveau supplémentaire à la hiérarchie dans un système embarqué n'est pas favorisé car ceci influe sur l'espace du circuit.

II.5. Optimisation de la mémoire et placement dans le flot de conception

Au début, la conception des systèmes embarqués à viser l'optimisation des processeurs pour améliorer les performances d'un système tout en considérant la mémoire comme une boite noire dont l'amélioration du fonctionnement est basée sur les développements technologiques (exemple DRAM, SRAM…etc) ou une simple hiérarchie d'un ou de plusieurs niveaux, alors que la mémoire présente d'énormes opportunités pour l'amélioration des performances et de l'énergie du système et ceci soit au niveau architectural (architecture de la mémoire) ou software (compilateurs et applications) [23].

Mais ces dernières années une large gamme de travaux a été réalisée pour l'optimisation des mémoires, ces travaux ont été réalisés dans quatre domaines différents : Synthèse de haut niveau, optimisations des localités pour les caches, architecture des ordinateurs et systèmes de fichiers et des bases de données [23].

Le problème de gestion des données et de la mémoire peut être traité au cours des différentes étapes du flot de conception. Si des optimisations sont possibles à toutes les étapes, les opportunités d'optimisation sont plus importantes à haut niveau [3].

L'optimisation de la mémoire en synthèse de haut niveau a progressé de l'allocation des registres à l'optimisation de la mémoire on chip et off chip.

II.6. Stratégie globale de la synthèse de la mémoire

La synthèse comportementale est une démarche descendante qui varie selon la classe de l'application, généralement les architectures synthétisées sont décomposées en quatre unités fonctionnelles :

- Unité de traitement dont le but est de réaliser l'ensemble des calculs de l'algorithme.
- Unité de mémorisation dont le rôle est de stocker l'ensemble des données manipulées par l'application.
- Unité de communication qui gère les transferts des données.
- Unité de contrôle qui pilote les autres unités.

Chacune de ces unités fonctionnelles peut être conçue indépendamment des autres, en effet des modèles génériques peuvent être formalisés pour chaque unité fonctionnelle. Cependant une approche globale du problème est nécessaire afin d'optimiser la conception architecturale [27] mais dans ce cas le nombre de contraintes fonctionnelles qu'il s'agit de satisfaire devient très grand à cause de l'interdépendance des unités l'une vis-à-vis de l'autre, cette augmentation rend le problème d'optimisation qui est de classe NP-complet insoluble. Un choix quant à l'ordre de conception des unités doit être fait, celui-ci définit la stratégie de conception. La synthèse de mémoire consiste à partir d'une description comportementale écrite dans un langage de description du matériel tel que VHDL ou le SystemC de fournir une architecture mémoire satisfaisant des contraintes de temps, d'espace et de consommation.

Lorsqu'on aborde le problème de la synthèse de la mémoire un certain nombre de phases sont incontournables [27]:

- ✓ L'augmentation de la complexité des unités de mémorisation nécessite la mise en place d'estimateurs rapides et puissants afin de limiter l'espace de recherche lors de la phase de conception. La plupart des études qui ont abordé ce point traite l'estimation de la mémoire nécessaire pour stocker l'algorithme compilé. Dans ce domaine des efforts importants ont été réalisés pour obtenir des informations les plus précises possibles avant la synthèse et cela en vue de limiter l'espace de recherche.

- ✓ L'estimation de complexité n'est vraiment utile que si le concepteur est capable d'appliquer des transformations à son problème. Il s'agit dans ce cas des transformations de l'algorithme permettant de réduire une fonction de coût mémoire. Ceci est l'objectif des travaux qui visent à prendre en compte la mémorisation au plus haut niveau. Des transformations s'appliquent en particulier aux boucles comme ceux de l'algorithme de Cathoor et al [28] de IMEC, Leuven Belgique qui ont proposé d'effectuer des transformations de code limitant soit le nombre de points mémoires à mettre en œuvre, soit le nombre d'accès à réaliser entre unité de traitement et la mémoire.

Exemple

```
For i := 1 To N Do
        B[i] := f(A[i]);
For i := 1 To N Do
        C[i] := g(B[i]);
```

Code avant transformation

```
For i := 1 To N Do  Begin
        B[i] := f(A[i]);
        C[i] := g(B[i]);
End ;
```

Code après transformation

Dans le code avant transformation, il est nécessaire de prévoir la mémorisation de trois vecteurs (A, B, C). De plus cette séquence de code provoque 4* N transferts mémoires (N écritures pour B, N lectures pour A, N lectures pour B, N écritures pour C). La transformation consiste à regrouper les deux boucles de façon à limiter le nombre d'accès à la mémoire et le nombre de points mémoires nécessaires. En effet après transformations la mémorisation du B n'est plus nécessaire et les transferts du vecteur B peuvent être supprimés. Dans ce cas on peut se satisfaire d'un seul registre dans l'unité de calcul. Cette séquence provoque alors 2*N transferts mémoires (N lectures pour A, N écritures pour C). Ces transformations permettent des gains importants sur : le nombre de points mémoire et d'accès des mémoires, la complexité de l'adressage et la consommation du circuit.

Pour être réaliste, la synthèse des unités de mémorisation doit s'appuyer sur une bibliothèque de composants existants. Vu la croissance de la complexité de conception des systèmes et la contrainte de délai de mise en marché (time to market)

le besoin de réutiliser des composants c'est avéré très utile, cette réutilisation est possible grâce aux bibliothèques des composants. Des mémoires sous forme de package standard ou des modules mémoires sont largement utilisés dans l'industrie actuellement sous forme de cache, scratch pad et même des DRAM embarquées. Il est alors nécessaire de développer des algorithmes capables de sélectionner parmi un ensemble de mémoires candidates le sous ensemble répondant au mieux au problème posé.

- ✓ La fonction d'adressage doit elle aussi être synthétisée, la matérialisation de cette fonction doit être minimisée selon un critère de synthèse souhaitée (surface, consommation, temps…etc).
- ✓ La quantité de données à mémoriser ainsi que la fréquence d'accès ne cessent de croître, la synthèse de la mémoire doit de plus en plus s'attacher à proposer des solutions de stockage hiérarchique.Il s'agit alors de sélectionner la hiérarchie la mieux adaptée au problème.

La stratégie globale visant l'optimisation de l'unité de mémorisation lors de la synthèse de haut niveau peut être envisagée en plusieurs phases :

- ✓ Choix d'une hiérarchie mémoire.
- ✓ La distribution des structures de données.
- ✓ Le placement des structures de données.
- ✓ L'ordonnancement des opérations de l'unité de traitement.
- ✓ Génération des adresses.

II.6.1. Choix d'une hiérarchie mémoire

Cette étape comporte des méthodes indépendantes de l'architecture cible. Elle commence par l'analyse du code source pour effectuer des transformations de code permettent éventuellement d'optimiser le nombre de transfert et le nombre de transition sur les bus des adresses. Une hiérarchie mémoire est définie en nombre de niveaux de hiérarchie, en taille et largeur de chaque niveau et en nombre de port : Les choix s'effectuent par rapport à des fonctions de coûts s'appuyant sur des

modèles caractérisant les mémoires en fonction de leurs paramètres architecturaux, algorithmiques et technologiques (bibliothèque mémoire).

II.6.2. Distribution des structures de données

Une fois la hiérarchie définie, on peut procéder à la distribution des structures de l'application sur les différents niveaux mémoires en se basant sur leurs localités temporelles et spatiales. Le principe consiste à favoriser les structures les plus utilisées et les stocker dans des niveaux proches de l'unité de traitement pour améliorer les performances. L'information finale de cette distribution sur les niveaux de l'hiérarchie est appelée le mapping mémoire.

II.6.3. Placement des structures de données

Le placement des structures de données en mémoire consiste à définir pour chaque structure de données où la placer et pour combien de temps, ceci revient à attribuer une adresse à chaque donnée distribuée en mémoire. L'attribution d'une même adresse pour plusieurs données permet de limiter la taille globale de la mémoire. Deux structures de données sont utilisées dans ce contexte: le graphe de dépendance d'expressions (GDE) pour la représentation des dépendances temporelles et le graphe des dépendances de structures (GDS) pour la représentation des dépendances structurelles ou spatiales.

II.6.4. Ordonnancement sous contraintes mémoires

Dans de nombreux cas, une architecture sur mesure fournit une bande passante et des caractéristiques de consommation de mémoire plus intéressantes qu'une architecture traditionnelle incluant un cache des données [27]. Concevoir une architecture mémoire spécifique signifie décider combien de mémoires utiliser et quels types de mémoires utiliser. De plus, les accès mémoires doivent être ordonnés dans le temps de façon à prendre en compte les contraintes temps réel.

Un des facteurs importants qui affecte le coût d'une architecture mémoire est l'ordre relatif des accès mémoire contenus dans la spécification. La plupart des méthodes d'ordonnancement prenant en compte la partie mémoire essaient de réduire le coût mémoire en estimant les besoins de mémorisation pour un ordonnancement donné.

II.6.5. Génération des adresses

Parallèlement à la fonction de mémorisation proprement dite, il faut mettre en place la fonction d'adressage des données. Il s'agit d'assurer à chaque pas de transferts, la production de toutes les adresses de stockage des données. Pour le faire une mise en œuvre de matériel fournissant les adresses des données à accéder est indispensable. Il existe plusieurs méthodes pour générer les adresses des données auxquelles on désire accéder.

La fonction de génération des adresses est, généralement, réalisée de trois façons différentes :

- ✓ par une unité arithmétique associée à un ensemble de registres : C'est la solution retenue dans les architectures d'usage général.
- ✓ par une unité spécifique : Il s'agit de proposer un schéma qui permet d'effectuer des adressages spécifiques.
- ✓ par une machine d'états : Ce modèle ne peut être utilisé que pour des adressages simples, c'est à dire des adressages dont on connait toutes les adresses avant exécution.

Ces deux derniers modèles conviennent particulièrement bien aux architectures des coprocesseurs dédiés pour lesquels, une fois la requête effectuée, le déroulement du programme est parfaitement connu.

Vu qu'on s'intéresse seulement aux optimisations de la mémoire au niveau cache, la suite de cette étude sera concentrée à la proposition d'une hiérarchie mémoire basée sur le partitionnement du cache de données.

II.7. Applications cibles

Les applications multimédia embarquées constituent le domaine cible de cette étude. Du fait que ces applications se caractérisent par une utilisation intensive d'un grand volume de données comme l'audio, les graphiques 2D et 3D, animation, images et vidéo et qui sont soumis à des contraintes de débit, de latence et de réactivité différentes, elles ont la particularité de nécessiter de grandes quantités de mémoire et d'en faire un usage intensif. Ces applications dans leur forme embarquée impliquent un parcours régulier de la mémoire associé à des contraintes temporelles et ceci n'est réalisable que sur une architecture utilisant une mémoire hiérarchique à plusieurs niveaux. Leur développement pose des difficultés particulières : Une fois l'application spécifiée et validée, le passage à une réalisation

implique le choix d'une architecture appropriée sur laquelle seront implantées les fonctionnalités de l'application. Le concepteur est alors amené à prendre plusieurs décisions pour préciser les détails de cette architecture et qui permet de conduire aux meilleurs compromis performances, surface, consommation, flexibilité, réutilisabilité et le temps de mise sur le marché (time-to-market) [29].

Les applications multimédias présentent un nombre important de caractéristiques communes qui peuvent être exploitées pour la personnalisation de la hiérarchie mémoire, ces caractéristiques peuvent être résumées en six points [29] :

- ✓ **Fort volume de données** : La fonctionnalité des systèmes est basée sur le traitement intensif de flux de données.
- ✓ **Flux de données multidimensionnelles**: Le traitement de l'information est typiquement réalisé sur des flux de données structurées et représentées sous la forme de matrices mono ou multidimensionnelles.
- ✓ **Motifs de calcul répétitifs** : Les procédures implantées dans les applications de traitement du signal et de l'image ont une structure régulière. La description des opérations réalisées sur des tableaux multidimensionnels est spécifiée à l'aide de boucles imbriquées permettant la répétition d'un motif de calcul.
- ✓ **Structure modulaire** : Les flux de données dans les applications multimédias sont soumis à un grand nombre de transformations consécutives (codage source, cryptage, modulation) qui peuvent elles-mêmes être constituées de sous-étapes.
- ✓ **Fonctions standards** : La structure modulaire est, dans la plupart des cas, basée sur des fonctions algorithmiques standards : les filtres, les transformations, les contrôles d'erreurs …etc.
- ✓ **Contraintes de conception** : Les applications multimédias sont fortement utilisées dans des produits grand public qui requièrent un délai de mise sur le marché extrêmement court. A cela s'ajoutent des contraintes drastiques de consommation d'énergie et de surface lorsque les systèmes sont embarqués dans des appareils électroniques portables.

Les versions embarquées en particulier des applications multimédias ont des exigences de plus en plus sévères, elles doivent répondre à des contraintes de miniaturisation, de faible consommation d'énergie, de traitement temps réel et de coût. Ces contraintes sont d'autant plus difficiles à satisfaire que la complexité des applications ne cesse d'évoluer. Mais l'exploitation des caractéristiques connues à l'avance de l'application soit au niveau traitement ou données offre aux concepteurs un moyen efficace pour optimiser l'architecture du système tout en répondant aux contraintes désirées.

La mémoire qui constitue un goulet d'étranglement dans les systèmes vu qu'elle constitue l'élément le plus consommateur d'énergie, occupant plus d'espace d'un chip et dont les accès sont très coûteux est un champ d'optimisation privilégié aux concepteurs car les résultats de l'optimisation se répercutent sur la performance, l'espace et l'énergie consommée tant désiré par les concepteurs des systèmes embarqués.

Chapitre III

Approches du partitionnement
Du
data Cache

Introduction

Différents groupes de recherches ont proposés le partitionnement du cache horizontalement pour améliorer les performances du cache. Un partitionnement horizontal par opposition au partitionnement vertical (qui veut dire les niveaux) consiste à partitionner le cache en deux ou plusieurs sous caches qui résident au même niveau dans l'hiérarchie mémoire, l'exemple le plus populaire du partitionnement horizontal est la séparation du cache d'instruction et le cache de données. Logiquement on peut penser au partitionnement du cache de données en sous caches pour exploiter les différentes localités présentées par les données.

Les organisations existantes du cache souffrent de l'incapacité d'exploiter les caractéristiques des différents types de localités des données vu qu'un seul cache de données est désigné généralement pour contenir tous les types de données : scalaires, tableaux, pointeurs …etc. Cette situation engendre dans la majorité des cas des mouvements et des déplacements de données non nécessaires à travers les différents niveaux de l'hiérarchie mémoire ce qui augmente le taux des défauts du cache, la consommation globale du circuit et génère dans la majorité des cas le phénomène de pollution du cache. Généralement le cache exploite la localité temporelle en retenant les références de données récemment utilisées pour un long temps, et la localité spatiale en chargeant de multiples mots voisins. Si une donnée n'expose pas de localité temporelle, son chargement dans le cache est inutile, de même si une donnée ne dispose pas de localité spatiale, le chargement d'un bloc entier entraîne un gaspillage de temps et d'énergie [30].

Vu que les données n'exposent pas les mêmes localités temporelles ou spatiales, utilisé un cache de données unifié pour contenir les deux types de données est inefficace, il semble alors avantageux d'exploiter des caches de données séparés : un cache pour les tableaux et un autre pour les scalaires spécialement pour les applications utilisant les données de manière intensive comme les applications multimédia, ceci permet de tirer avantage des caractéristiques de chaque type de données vu que les données scalaires exposent généralement une localité temporelle alors que les tableaux exposent une localité spatiale.

III.1. Architectures du cache partitionné

Différentes architectures ont été proposées par les chercheurs pour les caches partitionnés, on présente dans ce qui suit une description des différentes architectures d'un cache partitionné, tout en se limitant au partitionnement basé sur les localités spatiale et temporelle.

III.1.1. Partitionnement du cache selon la localité spatiale

III.1.1.1. Split Temporal/Spatial Cache (STS Cache)

Le partitionnement du cache selon les localités spatiale/temporelle a été introduit par un groupe de chercheurs incluant des membres de l'université de Belgrade, Yougoslavie, l'université de Monténégro de Podgorica, Yougoslavie et Sun Microsystems de Mountain de l'USA [31]. Dans cette conception, le cache est partitionné en un sous cache spatial et un sous cache temporel, le bloc du cache temporel a une taille d'un seul mot de 4 octets alors que le bloc du cache spatial est de 4 mots dont chaque mot est de 4 octets. L'étude réalisée compare le rapport entre la capacité du cache spatial et celle du cache temporel, trois cas ont été étudiés : 1 :1, 1 :2 et 1 :4. La décision quel cache utilisé est réalisée lors de l'exécution, pour le faire une heuristique basée sur les compteurs est utilisée. Le principe de l'heuristique consiste à tenir deux compteurs X et Y pour chaque bloc du cache spatial, le compteur Y est incrémenté à chaque accès au bloc du cache par contre le compteur X est incrémenté lorsqu'on accède à la partie haute du cache et décrémenté si on accède à la partie basse du cache. Quand le compteur Y est saturé, si la valeur absolue de X atteint un seuil spécifié le bloc du cache est considéré temporel et il est supprimé du sous cache spatial. Un préchargement séquentiel est ensuite effectué mais seulement dans le cache spatial.

III.1.1.2. Dual Data Cache

Le dual Data Cache a été introduit par un groupe de chercheurs à l'université polytechnique de Catalonie, Barcelone, Espagne [32]. Dans cette conception, le cache de données est partitionné en sous cache temporel composé de 33% de la capacité totale du cache et un sous cache spatial composé de 66% de la capacité totale du cache. Le sous cache temporel à un bloc de 8 octets, alors que le cache spatial a un bloc plus large de 16 ou 32 octets. La décision quel cache utilisé est réalisée lors de l'exécution, pour le faire on utilise un prédicateur de préchargement dirigé par pas, ce prédicateur et à chaque instruction load/store d'accès à une adresse mémoire, vérifie si cette adresse diffère par pas

constant des adresses accédées précédemment. Cette information sera utilisée à chaque défaut du cache par le contrôleur du cache pour placer les données soit dans le sous-cache temporel, sous cache spatial ou ne pas les placer du tout dans le cache [33]. Un préchargement séquentiel est aussi utilisé dans cette conception mais seulement dans le cache spatial.

III.1.1.3. Selective and dual data cache

Un autre groupe de la même université a décrit une autre version de la conception du dual data cache [34]. Dans cette nouvelle conception, le sous-cache temporel est seulement un petit buffer complètement associatif de plus de 16 mots, alors que le cache spatial à un bloc de 32 octets. La décision quel cache utilisé est réalisée lors de la compilation en analysant la localité des données. Dans cette conception, il est possible que les données soit placées dans le sous cache temporel, le sous cache spatial ou ne pas être placées du tout dans le cache [33].

III.1.1.4. Array Cache

Ce type de cache a été introduit par un groupe de chercheurs à l'université du Colorado, Fort Collins USA [35]. Dans cette conception, le cache de données primaire est partitionné en un sous cache pour les scalaires appelé Scalar cache et un sous cache pour les tableaux appelé Array cache, Scalar cache a une taille de 25% de la taille globale du cache primaire et un bloc de cache de 32 octets alors que Array cache dispose de 75% de la taille globale du cache primaire et une taille de bloc de cache variant expérimentalement de 64 à 512 octets. La décision quel cache utilisé est réalisée lors de la compilation, en dirigeant les accès vers les variables scalaires au Scalar cache alors que les accès vers les tableaux au Array cache [33].

III.1.2. Partitionnement du cache selon la localité temporelle

III.1.2.1. Assist cache

Proposé par Chen, cette conception a été incorporée dans le CPU HP7200 [36]. Le partitionnement du cache consiste dans ce cas d'adopter deux sous caches, le premier sous cache appelé Main cache est un cache d'une large capacité, externe et à adressage direct par contre le deuxième sous cache est appelé Assist Cache est un cache on chip et complètement associatif. Les deux sous caches ont une taille de bloc du cache de 32 Octets. Assist cache sert à réduire le nombre des conflits dans la Main cache comme il sert en tant qu'un sous cache non temporel, alors que le

Main cache sert comme un cache temporel. Les quatre approches précédentes partitionnent le cache selon la localité spatiale, Assist cache le partitionne selon la localité temporelle. Lors de la compilation certaines instructions d'accès à la mémoire sont marquées comme ayant une localité spatiale seulement et les données accédées par ces instructions ne sont pas placées dans le main cache mais seulement dans Assist Cache. Les autres données sont considérées comme ayant une localité spatiale et temporelle et peuvent être placées dans le Main cache [33].

III.1.2.2. Non Temporal Streaming Cache (NTS cache)

Cette approche de partitionnement du cache a été proposée par une équipe de chercheurs de l'université de Michigan, Ann Arbor, USA [37], elle est similaire à l'approche de l'Assist cache et consiste à partitionner le cache en un sous cache temporel de grande capacité et à adressage direct et un autre sous cache non temporel de petite capacité et complètement associatif, les deux sous caches ont une taille de bloc de 32 Octets. Le partitionnement se fait lors de l'exécution et selon la localité temporelle des données, son principe consiste à associer un vecteur de bits à chaque entrée du cache, si aucune réutilisation n'est détectée à un bloc particulier dans le sous cache temporel il sera marqué comme non temporel et il sera expulsé du cache temporel, les accès futurs à cette référence mémoire causeront une recherche dans le sous cache non temporel [33].

III.2. Partitionnement du Data cache pour les applications multimédia embarquées

Vu que les performances du cache dépendent des caractéristiques de localité exposée par les données traitées par le programme ainsi que de l'architecture sous jacente [38], et en plus les applications multimédia sont des applications orientées données d'une manière intensive, on propose une architecture de cache qui groupe les données selon leur localité inhérente soit temporelle ou spatiale et mappe chaque groupe à un cache dédié.

Le cache des tableaux (array cache) est un cache à adressage direct avec une large taille de bloc pour exploiter la localité spatiale exposée par ce type de données et ceci pour permettre de charger des blocs voisins au bloc recherché réduisant ainsi les défauts obligatoires de chargement (compulsory miss). La

localité spatiale est exploitée en chargeant un bloc entier en cache (Un bloc contient plusieurs mots) [39].

Le cache des scalaires (scalar cache) est un cache à adressage direct avec une faible taille de bloc pour exploiter la localité temporelle exposée par ce type de données et réduisant les défauts de conflits (conflictual miss). La localité temporelle est exploitée en utilisant de petit bloc et une hiérarchie mémoire [39].

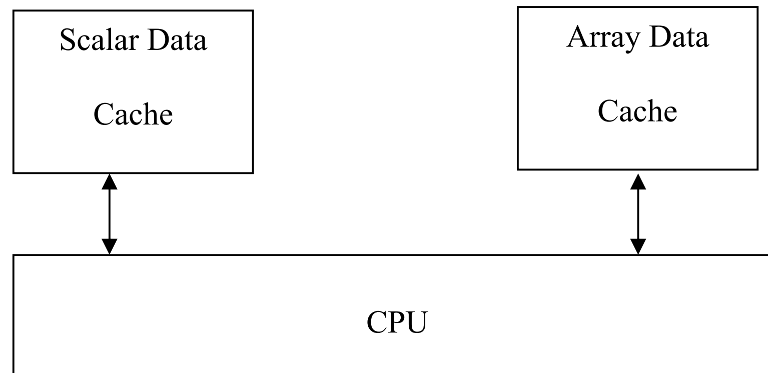


Figure III.1. Architecture d'un data cache partitionné en Scalar cache et Array cache

III.3. Méthodes d'analyse du cache:

Vu que le gap entre le processeur et la mémoire ne cesse d'accroître au fil des années, les méthodes d'évaluation des mémoires deviennent de plus en plus importantes. Certaines méthodes sont utilisées après l'implémentation hardware et sont coûteuses alors que d'autres sont utilisées avant l'implémentation et sont plus utilisées pour les études. Cette section présente les techniques utilisées pour mesurer les performances du cache. D'une manière générale pour caractériser le comportement de n'importe quel composant architectural et quantifier ses performances, trois approches ont été utilisées: le hardware monitoring, la modélisation analytique et la simulation logicielle.

III.3.1. Hardware Monitoring: la méthode la plus précise pour mesurer les performances du cache est de mesurer directement les performances du cache en vraie grandeur en le faisant fonctionner. Ceci implique dans la majorité des cas un hardware conçu spécialement pour un système donné. Par exemple Wood décrit embedded SPUR monitoring hardware qui mesure plusieurs métriques de performance du cache mais il constitue une partie interne du SPUR cache [40]. Le hardware Monitoring donne des résultats de performance du cache très précis et réalistes, cependant cette méthode est coûteuse et peu flexible car la plupart des paramètres du cache sont figés. Ainsi on ne peut mesurer que des configurations de caches existantes et n'est donc pas adapté quand il y a un choix à faire entre plusieurs configurations. Vu que les coûts de fabrication étant élevés, les études sont souvent limités à des composants existants ce qui limite l'intérêt de cette approche pour l'exploration de nouvelles tendances architecturales. Pour ces raisons, de nos jours, cette méthode n'est plus utilisée [41].

III.3.2. La modélisation analytique: Dans ce cas le comportement du cache est représenté par une fonction mathématique (modèle). Le modèle permet d'évaluer rapidement les performances du cache mais en contrepartie, la grande complexité du cache fait que le modèle utilisé est souvent trop simplifié et donne une estimation peu précise des performances [41]. A titre d'exemple un modèle analytique de mémoire cache est présenté dans [42]. La modélisation analytique est moins coûteuse que le hardware monitoring et elle est suffisamment flexible pour estimer les performances d'une gamme de configurations. Le problème avec les modèles analytiques est que ses résultats sont théoriques. Les modèles analytiques sont plus bénéfiques pour la compréhension d'une intuition, par

exemple « le taux des défauts du cache directement adressable de taille X est le même qu'un cache associatif par 2 ensembles ayant une taille X/2 » ou encore « Doubler la taille du cache diminue le taux des défauts de cache par 25% ». Ces modèles ne sont pas appropriés dans le cas ou la précision et l'exactitude sont demandées mais ils sont utiles quand le temps de simulation est trop lent [40].

Malgré son manque de précision, cette approche peut tout de même être utilisée pour dégrossir le champ d'investigation et orienter la recherche avant d'utiliser la simulation logicielle en guise de validation ou pour affiner les résultats.

III.3. 3. Simulation:

Dans ce cas le cache est modélisé par un programme : le simulateur. Celui-ci permet la représentation précise des mécanismes matériels complexes et génère en même temps l'information nécessaire à l'étude. La simulation logicielle est rapide à mettre en œuvre et financièrement intéressante comparé au monitoring hardware. Cette approche est aussi très flexible : les paramètres du cache simulé peuvent être aisément changés et plusieurs alternatives peuvent être comparées. Cependant, la flexibilité à un coût qui est la lenteur de la simulation, et ne permet pas, par conséquent, de simuler des programmes de taille réaliste.

Néanmoins, la simulation logicielle remporte un grand succès : c'est désormais une approche systématiquement utilisée pour la conception des caches car elle allège les limites du hardware monitoring et des modèles analytiques. Les simulateurs peuvent être fonctionnels ou temporels. Ils peuvent être dirigés par trace ou par exécution comme ils peuvent simuler le système tout entier ou un composant du système. Les simulateurs fonctionnels simulent la fonctionnalité du processeur et ils produisent des informations de performance telle que les cycles d'exécutions, le taux des hits ...etc.

III.3. 3.1. La simulation dirigée par trace:

La simulation dirigée par trace a attiré l'attention des chercheurs durant les années passées et a subi ainsi un développement rapide. Elle consiste à écrire un programme qui simule le comportement d'une conception mémoire donnée, puis à appliquer une séquence de références mémoires au modèle de simulation pour reproduire la manière dont un processeur réel doit exercer la conception. La séquence des références mémoires est appelée la *trace d'adresses*, et la méthode est appelée la simulation dirigée par trace. Elle consiste en un modèle de simulation ayant des entrées modélisées sous forme de trace

ou de séquence d'information représentant les instructions exécutées ou les adresses mémoires référencées. Un simple simulateur dirigé par trace a besoin de trace représentant les valeurs d'adresses référencées lors de l'exécution d'un programme , ces adresses peuvent représentées celle de données ou d'instructions dépendant de ce que le simulateur modélise un cache unifié d'instruction et de données , cache de données ou d'instructions. Contrairement au hardware monitoring un simulateur peut vérifier plusieurs configurations en plus il peut produire des résultats précis et exactes au moins à la limite des traces d'entrées.

La figure suivante schématise le principe de fonctionnement d'un simulateur dirigé par trace.

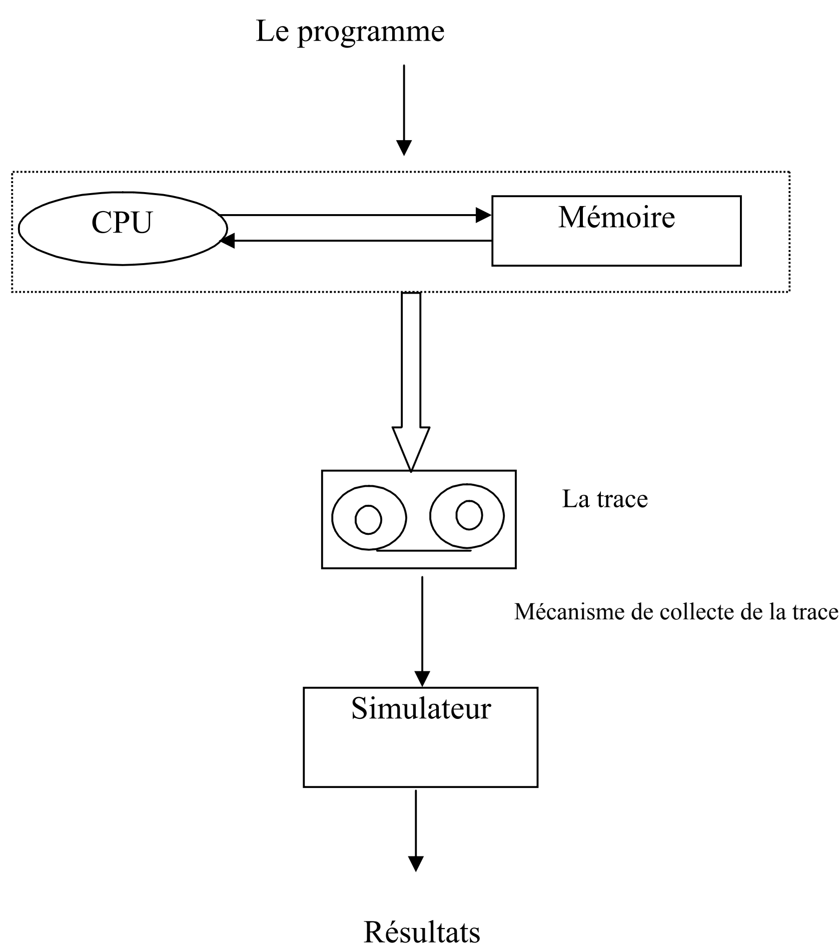


Figure III.2. Principe de fonctionnement d’un simulateur dirigé par trace [40]

Le principe de fonctionnement d'un cache dirigé par trace consiste a :

- ✓ **Collecte de la trace des adresses** : Cette étape est aussi importante que la simulation du cache, elle consiste à collecter les adresses mémoires référencées lors de l'exécution du programme et de les sauvegarder pour une ultérieure utilisation. Ces traces peuvent être aussi volumineuses quelles nécessites une compression avant la sauvegarde. Notons que la trace est générée une seule fois mais peut être utilisée autant de fois.
- ✓ **Analyse du cache et génération des résultats** : Lors de cette phase le simulateur utilise le fichier trace pour simuler le cache selon une configuration donnée (capacité du cache, taille du bloc, politique de remplacement, technique d'écriture...etc.), pour le faire il extrait chaque référence de la trace et la soumit au cache simulé de la même manière qu'un cache réel. En variant la configuration du cache on peut simuler plusieurs alternatives du cache.

Conceptuellement elle apparaît simple mais en pratique un certain nombre de facteurs rend la simulation dirigée par trace difficile. La collection d'une trace d'adresses complète et détaillée peut être dure, spécialement si elle est utilisée pour représenter une application complexe composée de plusieurs processus, système d'exploitation et éditeur de lien dynamique ou un code compilé dynamiquement. Un autre problème pratique et que les traces d'adresses sont typiquement très larges, consommant potentiellement des giga bytes d'espace de stockage. Finalement, le traitement d'une trace pour simuler les performances d'une mémoire hypothétique est une tâche consommant le temps.

Pour cette raison, une étape supplémentaire de compression de trace peut être utilisée avant son stockage sur disque. D'autres méthodes permettent d'obtenir directement une trace réduite composée d’évènements significatifs au lieu des adresses. Dans ce cas, une étape supplémentaire en amont du simulateur est nécessaire pour reconstruire une trace utilisable.

Dans tous les cas, la simulation dirigée par trace nécessite de trouver un compromis entre la taille des données stockées sur disque et le temps qu'il faut pour compresser /décompresser ou reconstruire la trace. Durant les dernières dix ans, les chercheurs travaillant sur ces problèmes ont fait d'importants développements dans: la collection de la trace, la réduction de la trace et le traitement de la trace d'adresses.

Exemple d'outils de simulation dirigée par trace:

DineroIV: est un simulateur écrit par Mark Hill et disponible gratuitement pour une utilisation non commerciale au site de l'université Wisconsin de Madison [43].

Ce simulateur est dirigé par trace, non temporel en d'autre terme il n'a aucune notion de temps et non fonctionnel c'est-à-dire que les données et les instructions ne se sont pas sauvegarder dans le cache, il a besoin seulement d'information reliée à l'adresse de la valeur ayant été placée dans le cache. Cette information est suffisante pour déterminer un futur hit ou miss.

Dinero est destiné pour les systèmes uniprocesseur, il permet d'évaluer un seul cache à la fois mais il produit plusieurs informations de performance et permet la variation de plusieurs options du cache (Write back vs Write through, LRU vs remplacement aléatoire...). Dinero est plus utile pour l'étude de quelques alternatives du cache de l'espace de conception.

Tycho: Est un simulateur de cache écrit par Mark Hill de l'université Wisconsin, il fait partie des outils regroupés sous Warts, Tycho permet d'évaluer plusieurs alternative de cache pour un système uniprocesseur, mais il stricte sévèrement les options de conception qui doivent être variées. Ainsi en une seule passe et en utilisant une trace générée par un générateur de trace tel que QPT par exemple, Tycho génère une table des taux de défauts des caches de plusieurs tailles et différentes associativités tout en supposant que ces caches ont la même taille de bloc et utilisant LRU comme algorithme de remplacement. Tycho est utile dans le cas ou on désire réduire la capacité du cache.

Tycho et Dinero font tous deux parties de l'ensemble d'outils Warts de l'université Wisconsin, Ils sont écrits en C et utilisent le même format ASCII du fichier trace, ils sont utilisés par plusieurs universités et compagnies.

Pin: Pin est outil d'instrumentation des programmes. Il fonctionne sous Linux et Windows pour les architectures IA-32, Intel 64 et IA-64. Pin permet d'insérer du code arbitraire (écrit en C et C++) dans des emplacements arbitraires dans un fichier exécutable, ce code est ajouté dynamiquement lors de l'exécution du programme ce qui lui permet d'être attaché un processus en cours d'exécution.

Pin inclus le code source d'un ensemble d'outils d'instrumentation comme: profiler de blocs, simulateurs de cache, générateur de trace...etc. Il est très facile de dériver de nouvels outils en se servant de ces exemples comme des templates. [44]

III.3. 3.2. La simulation dirigée par exécution:

La technique de simulation vue précédemment repose sur l'hypothèse que la trace générée par un programme de test s'exécutant sur une machine hôte reflète correctement la charge de travail du composant simulé, bien que celui-ci ait des caractéristiques différentes [41]. Par exemple, les références aux données faites par un programme peuvent être utilisées pour simuler des mémoires cache différentes de celles implémentées sur la machine hôte.

Un simulateur dirigé par exécution est imbriqué dans le programme de test et l'information transite entre les deux et dans les deux sens. La simulation dirigée par exécution demande une infrastructure de traçage et une interface plus complexe que la simulation dirigée par trace car le simulateur nécessite une information précise et détaillée [41].

Le simulateur le plus utilisé dans cette catégorie est le SimpleScalar, l'avantage de ce type de simulation est la rapidité.

SimpleScalar: Exemple de simulateur dirigé par exécution:

SimpleScalar est un ensemble d'outils de simulation et d'analyse de performance des programmes, il a été développé à l'université Wisconsin et il regroupe un ensemble d'outils pour le profiling et la simulation du cache. SimpleScalar est un simulateur de processeur au niveau architectural. C'est un programme gratuit et libre ce qui permet de le modifier pour créer d'autres simulateurs. En réalité, SimpleScalar est composé de plusieurs simulateurs présentés ci dessous [45].

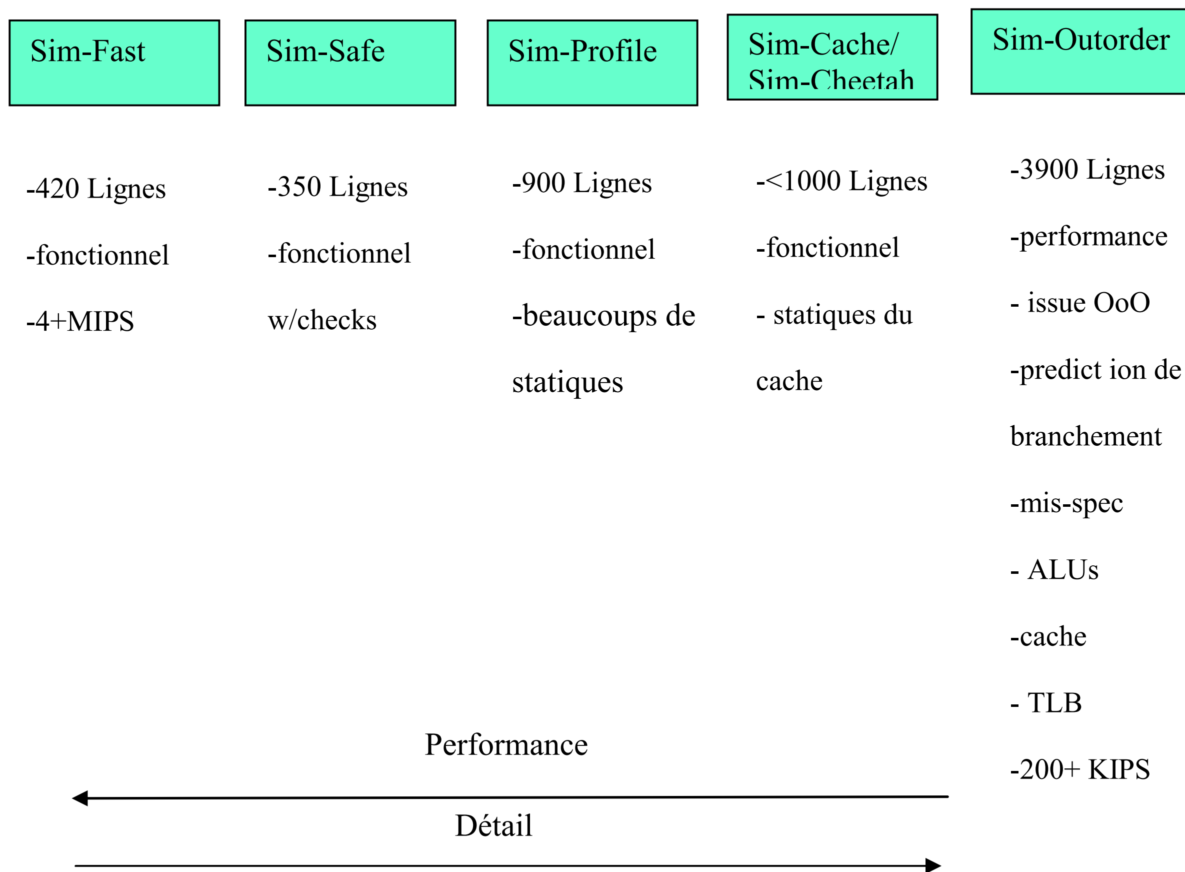


Figure III.3. Les simulateurs de *SimpleScalar* d'après [45]

Les seuls formats binaires connus en entrée par SimpleScalar sont le binaire Alpha et le PISA (Portable Instruction Set Architecture). Pour pouvoir simuler un exécutable avec SimpleScalar, celui-ci doit être compilé avec un compilateur GCC spécial qui permet la création d'un fichier binaire au format PISA.

Chapitre IV

Etude Expérimentale

Introduction

Dans ce chapitre on va décrire l'environnement expérimental utilisé pour notre étude, on commencera par le choix du simulateur, puis on va spécifier les paramètres du cache telle que : la capacité du cache, la taille du bloc du cache, associativité et l'algorithme de remplacement ainsi que les benchmarks utilisés lors de l'évaluation du cache.

IV.1. Choix du simulateur:

La simulation dirigée par exécution présente plusieurs avantages par rapport à la simulation dirigée par trace en outre elle permet l'accès à toutes les données produites et consommées durant l'exécution du programme. Ces valeurs sont critiques pour l'étude des optimisations telle que la compression de mémoire, l'analyse dynamique de l'énergie…etc. On a choisi SimpleScalar qui appartient a cette catégorie de simulateurs pour simuler notre cache et ceci pour plusieurs raisons:

- ✓ Il est gratuit et open source.
- ✓ Il est devenu populaire dans la communauté des chercheurs de l'architecture de l'ordinateur. En 2000, plus d'un tiers des meilleurs articles publiés dans les conférences de l'architecture des ordinateurs utilisent les outils de SimpleScalar pour évaluer leurs conceptions.
- ✓ Il offre cinq simulateurs dirigés par exécution, ces simulateurs appartiennent au simulateur fonctionnel extrêmement rapide ou même simulateur d'un processeur SuperScalair.
- ✓ Portable: pour l'utiliser on a besoin tout simplement de produire les outils GNU offerts avec l'ensemble en plus il supporte de multiples ISA.
- ✓ Il s'exécute sur la majorité des plateformes Unix et NT.
- ✓ Facilement extensible : il contient un jeu d'instruction pour fournir une annotation détaillée, alors on peut facilement écrire de nouvelles fonctions pour étendre l'ensemble des outils.
- ✓ Rapide: plusieurs millions d'instructions par seconde (4+MIPS).
- ✓ Détaillé: des simulateurs avec un détail arbitraire sont disponibles.

Remarque :

Pour mesurer la consommation d'énergie, le temps d'accès et la surface du cache on a utilisé CACTI version 3[46]. Initialement développé par Wilton et Jouppi, CACTI est un modèle analytique intégré estimant les cycles, la surface, l'énergie, la puissance dynamique et les temps d'accès des mémoires caches. En intégrant tous ces modèles ensemble, les différences entre temps, puissance et surface sont toutes fondées sur les mêmes hypothèses et, par conséquent, sont mutuellement cohérentes [47].

IV.2. Paramètres du Cache:

Dans les applications multimédia dominées par les données et qui utilisent des structures de données très larges comme les tableaux multidimensionnels, une grande quantité de mémoire est nécessaire pour la sauvegarde et le traitement des ces données. Le transfert entre les mémoires a un grand effet sur la consommation de l'énergie, espace et la performance du système. Comme il est indiqué dans [48] la mémoire est l'unité qui consomme plus d'énergie dans les applications multimédia. Ceci est du a deux raisons majeures:

- ✓ La nature des applications multimédia qui sont dominées par les données.
- ✓ L'énergie consommée est dû à l'accès à des mémoires externes, qui est plus significative que celle des opérations arithmétiques et logiques.

Le cache joue un rôle important dans l'amélioration des performances de ces systèmes, l'efficacité de ce dernier est liée aux paramètres du cache ainsi qu'au programme en cours d'exécution.

IV.2.1. Capacité du cache

Le paramètre le plus significatif dans la conception du cache est la capacité du cache, elle représente le nombre total de données qu'on peut sauvegarder dans le cache, celle-ci est incrémentée par un facteur de 2 et elle est indépendante de l'organisation du cache. L'augmentation de la capacité du cache améliore les performances du système mais ces performances peuvent ne pas être significatives dans les systèmes embarqués car le cache dans ces systèmes est soumis à des contraintes de coût et d'espace c'est pourquoi la décision quelle est la capacité du cache à implémenter dans le système est critique. Un cache avec une large capacité peut améliorer le taux de hit pour certaines applications aux dépens de plus d'énergie consommée pour la récupération (fetch) et la sauvegarde des données

dans le cache. Les applications orientées performance bénéficient des cache de large capacité [48] par contre pour les applications orientées énergie ce paramètre doit être choisis délicatement.

Pour spécifier la capacité du cache à simuler on présente dans le tableau IV.1 un état des capacités des caches de données utilisées dans les processeurs embarqués.

Processeur	Capacité du cache de données
Sempron2100+/ AMD	64KO
IBM PowerPC 405	16KO
Mobile Celeron M ULV/Intel	64KO
Freescale MC9328MX1 i.MX1 and i.MXL	16KO
Embedded pentium	8KO
Samsung S3C2400/2410	16KO
Samsung S3C44B0X	8KO
NEC VR5432	64 KO
Renesas SH7780	32KO
IDT R3081	4KO ou 8KO
STn8811 Nomadik Mobile Multimedia Application Processor	16KO
Sony Emotion Engine	8KO
Hitachi SH7750	16KO
NEC VR5400	32KO
Sempron2100+/ AMD	64KO
IBM PowerPC 405	16KO

Tableau IV. 1 : Quelques capacités du data cache pour quelques processeurs embarqués

Ce tableau montre les capacités du cache de données utilisé dans les systèmes embarqués, nous avons sélectionné les tailles de 8kO, 16KO, 32 KO et 64KO pour notre étude.

IV.2.2. Taille du bloc:

La taille du bloc du cache est un autre paramètre qui affecte grandement la performance du cache, elle représente le nombre d'octets déplacés de/vers le second niveau mémoire lors d'un défaut du cache. Typiquement la taille varie entre 16, 32 et 64 octets. Généralement si on incrémente la taille du bloc du cache, le taux des défauts diminue car à chaque chargement d'un bloc en cache on extrait plus de données de la mémoire ainsi peu d'accès à la mémoire sont nécessaires. Il est clair que l'augmentation de la taille du bloc réduit le taux des défauts de chargement et améliore le préchargement des données, mais actuellement l'augmentation de la taille du bloc (sans toucher celle du cache) augmentent le taux des autres types de défauts du cache, car l'augmentation de la taille du bloc réduit le nombre de blocs menant à une augmentation des défauts de conflits et de capacité [22]. En plus comme indiqué dans [48] les blocs de grandes tailles tentent de fournir une localité spatiale supérieure mais requièrent plus de données à lire et peut être à écrire dans un write back lors d'un défaut de cache, pour cette raison, un trafic de mémoire minimal est observé avec les caches ayant une petite taille du bloc.

Pour sélectionner les tailles de blocs à utiliser dans notre étude, on a consulté celles utilisées dans les systèmes embarqués (Voir la figure IV.2)

Processeur	Bloc (Octet)	Processeur	Bloc (Octet)
AMD-K6-IIIE	32	Motorola MPC8540	32/64
Alchemy AU1000	32	Motorola MPC7455	32
ARM7	16	NEC VR5500	32
Hitachi SH7750S (SH4)	32	NEC VR4131	16/32
Hitachi SH7727	16	NEC VR4181	16
IBM PPC 750CX	32	NEC VR4181A	32
IBM PPC 7603	32	PMC Sierra RM 7000A	32
IBM 750FX	32	SandCraft sr71000	32
IBM403GCX	16	SuperH	32
IBMPower PC 405CR	32	TriMedia TM32A	64
Motorola MPC8240	32	Xilinx Virtex IIPro	32
Motorola MPC823E	16	Triscend A7	16

Tableau IV.2 : Quelques tailles de bloc utilisées dans les caches pour quelques processeurs embarqués

Dans notre étude, pour le cache unifié de scalaire et de tableau la taille du bloc choisi est respectivement de 16octets, 32 octets et 64 octets alors que pour le cache partitionné on a pris la taille du bloc : 8,16 et 32 octets et ceci pour le Scalar cache, pour Array cache on a choisi 16,32 et 64 octets.

IV.2.3. Associativité:

Les études menées dans [49], [50] ont montré que les caches ayant une haute associativité ou ceux ayant un bloc de large taille ne sont pas convenables pour les systèmes multimédia embarqués. Ceci est dû au fait que l'augmentation de l'associativité mène à une consommation excessive de l'énergie, par exemple un cache à adressage direct consomme seulement 30% d'énergie par accès de celle consommée par un cache associatif par 4 ensembles [48], pour cela la majorité des efforts de conception des caches embarqués se sont concentrés sur l'optimisation du cache à adressage direct. Ces derniers présentent les avantages suivants : simples, faciles à concevoir et requièrent moins d'espace Silicon que les caches ayant une haute associativité [24].

L'inconvénient majeur des caches à adressage direct est le taux élevé des défauts conflictuels, ces derniers selon [48] présentent 40% des défauts à adressage direct.

Les applications orientées performances utilisent la plus haute associativité possible alors que les applications orientées énergie s'orientent vers une associativité telle que l'énergie sauvegardée des ensembles pour améliorer le taux des hits dépasse l'énergie augmentée par accès [48].

Vu qu'on va séparer les scalaires des tableaux dans des caches différents et indépendants et aucun des deux n'influe sur l'autre, ce qui réduit significativement le nombre des défauts de conflits et de capacités, on a pris dans notre étude des cache à adressage direct.

Remarque

Pour les autres paramètres du cache comme l'algorithme de remplacement, latence...etc. on a pris les paramètres par défaut du SimpleScalar.

IV.3. Mesures de performance:

Dans ce qui suit on va décrire les mesures de performances utilisées dans notre étude.

IV.3.1. Taux des défauts du cache (Miss Rate):

L'efficacité du cache est sa capacité de garder des données fréquemment utilisés par le processeur, celle-ci est mesurée par le taux de défauts (miss) ou de hit. Le taux des défauts du cache (miss rate) est le rapport du nombre de références qui sont des miss (défaut du cache) sur le nombre total de références mémoires. Ce taux est affecté par la capacité, l'organisation du cache et le programme en cours d'exécution [49]. Le taux des défauts de cache n'est pas une valeur absolue et pour une configuration donnée il dépend du programme exécuté. On a utilisé SimpleScalar pour mesurer le taux des défauts du cache.

IV.3.2. Temps d'accès au cache:

Le temps d'accès au cache est le nombre moyen de cycles nécessaires pour accéder avec succès à une adresse référencée. On a utilisé CACTI avec une technologie 0.13 µm pour calculer le temps d'accès au cache. Ce paramètre est très utile lors de l'évaluation des performances du cache car même si un cache présente un faible taux de défauts du cache, il peut en revanche consommer un temps d'accès considérable lors d'un hit. Par exemple un cache avec une grande associativité peut avoir un faible taux de défauts qu'un cache à adressage direct mais ce dernier dispose d'un temps d'accès plus faible que celui d'un cache associatif [47].

IV.3.3. Surface du cache:

Vu que les concepteurs des systèmes embarqués ne s'intéressent pas seulement à la performance mais aussi à une meilleure utilisation de l'espace silicon, la surface du cache constitue alors un paramètre très important dans la conception de ces systèmes. On a utilisé CACTI avec une technologie 0.13 µm pour mesurer l'espace silicon occupé par le cache.

VI.3.4. Consommation d'énergie:

La consommation d'énergie constitue elle aussi un paramètre important dans la conception des systèmes embarqués vu que ces derniers sont généralement alimentés par batterie, Les études menées dans [51], [52] et [53] ont conclue que le cache on chip est responsable de 50% de l'énergie dissipée par le processeur embarqué, ainsi la minimisation de cette consommation est l'un des objectifs visés par les concepteurs. On a utilisé CACTI avec une technologie 0.13 µm pour mesurer la consommation dynamique d'accès au cache.

IV.4. Les benchmarks:

Dans notre étude on a sélectionné quelques benchmarks de la suite Mediabench, celle-ci représente les applications multimédia destinées aux systèmes embarqués, ces benchmarks sont des applications écrites en C et couvrent le traitement d'images et de vidéo, audio, traitement de la voix et aussi le cryptage et les graphiques [54]. Mediabench est défini par Lee, Potkonjak, et Mangione-Smithet, cet ensemble regroupe les applications suivantes:

Benchmark	Description
ADPCM	Codeur et décodeur audio simple par modulation différentiel adaptatif (*rawcaudio / rawdaudio*)
EPIC	Un codeur et décodeur de compression d'images incluant le codage run-length/huffman. (*epic/unepic*)
G.721	Codeur et décodeur de compression de voix basé sur les standard G.711, G.721 et G.723 (*decode/decode)*
GhostScript	Interpréteur pour le language script
GSM	Codeur et décodeur de transcodage de la voix basé sur le standard européean GSM 06.10 (*gsmencode/gsmdecode)*
H.263	Codeur et décodeur de vidéo basé sur la standard H.263 fourni par Telenor R&D(*h263enc/h263dec*)
JPEG	Un codeur et décodeur de compression d'image avec perte pour les image couleur et niveau de gris, basé sur le standard JPEG. *(cjpeg.djpeg)*
Mesa	Une bibliothèque graphique 3-D de OpenGL, inclus trois programmes de démonstration (*mipmap, osdemo, texgen*)
MPEG-2	Un codeur et décodeur de compression de vidéo (motion) pour une transmission vidéo haute qualité, basée sur le standard MPEG-2 (*mpeg2enc mpeg2dec*)
MPEG-4	Un codeur et décodeur de compression de vidéo pour le codage des vidéo en utilisant les modèles objets vidéo, basé sur le standard MPEG-4 et fourni par European ACTS project MoMuSys. (*mpeg4enc mpeg4dec*)
PEGWIT	Une clé publique de cryptage et d'authentification (*pegwitenc/pegwitdec*)
PGP	Une clé publique pour le codage/décodage du cryptage incluant un support pour les signatures. (*pgppenc/pgpdec*)
RASTA	Une application de reconnaissance de la voix qui supporte PLP, RASTA et les nouvelles techniques d'extraction futures de JAH-RASTA (*raste*)

Tableau IV.3 : Description de la suite Mediabench

Pour notre étude on a sélectionné six benchmarks qui sont : *mpeg2encode, mpeg2decode, epic, g721, gsm-toast et unepic.*

Au début on a fixé la capacité du cache et on fait varier la taille du bloc, puis dans la deuxième étape on a fixé la taille du bloc et on a fait varier la capacité du cache. Les combinaisons prises sont:

Capacité du data cache (Koctet)	8	16	32	64

Taille du bloc (octet)	16	32	64

Notre objectif est d'étudier l'effet de la taille du bloc ainsi que la capacité du cache sur le taux de défaut, espace, énergie et temps d'accès au data cache dans le cas des applications multimédia embarqués.

On a obtenu les résultats suivants:

IV.5. Résultats expérimentaux d'un data cache unifié (scalaire et tableau)

IV.5.1. Taux de défauts du data cache

- **Capacité du data cache fixe et taille du bloc variable**

  - ✓ **Capacité du data cache = 8Koctet et taille du bloc varie entre 16, 32 et 64 octets**

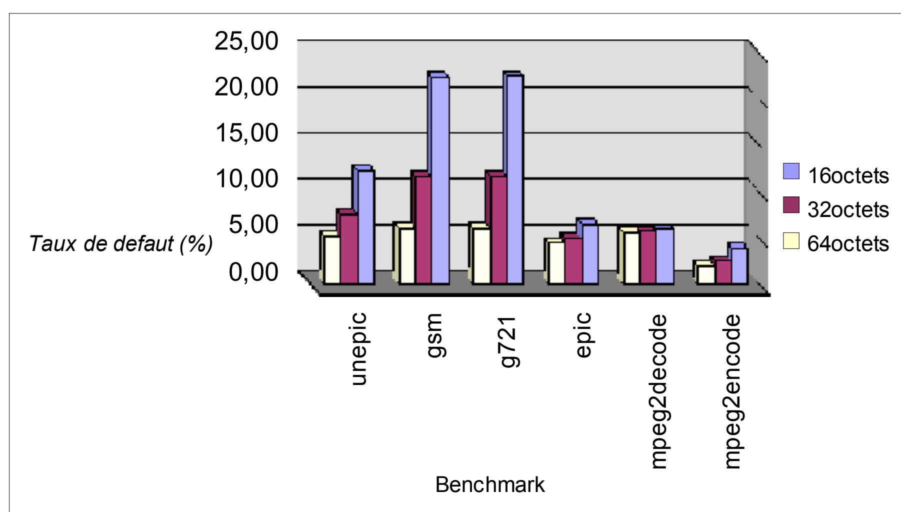


Figure IV.1 : Taux de défauts pour un cache de données de 8 KOctets

✓ **Capacité du data cache = 16 KOctet et taille du bloc varie entre 16, 32 et 64 octets**

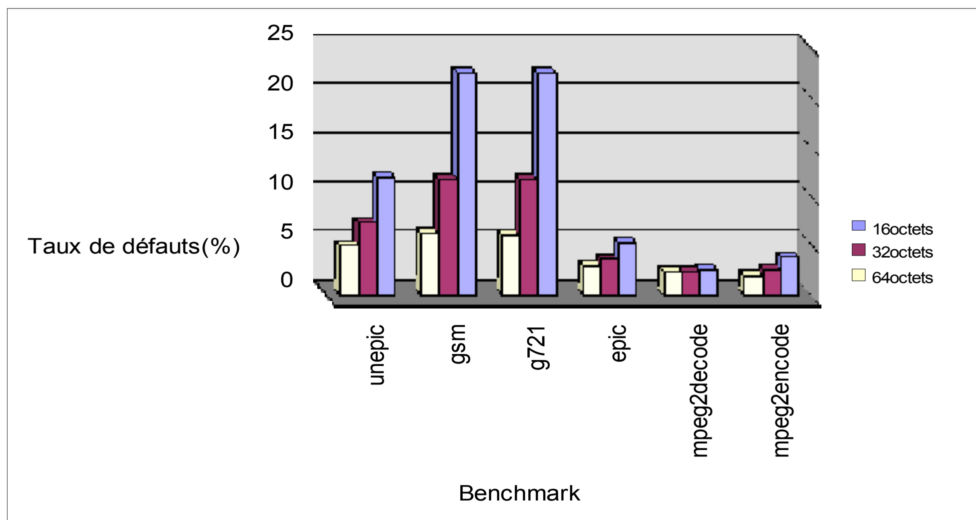


Figure IV.2 : Taux de défauts pour un cache de données de 16 KOctets

✓ **Capacité du data cache = 32 KOctet et taille du bloc varie entre 16, 32 et 64 octets**

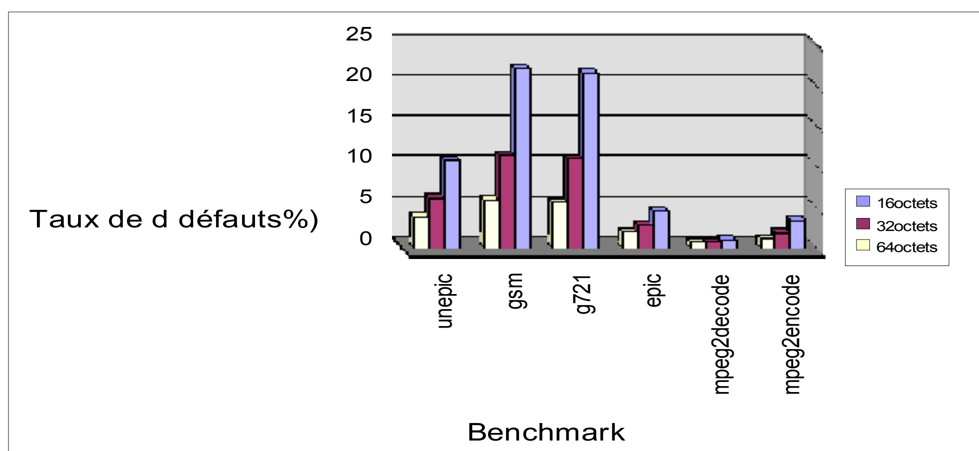


Figure IV.3: Taux de défauts pour un cache de données de 32Koctets

✓ **Capacité du data cache = 64 KOctet et taille du bloc varie entre 16, 32 et 64 octets**

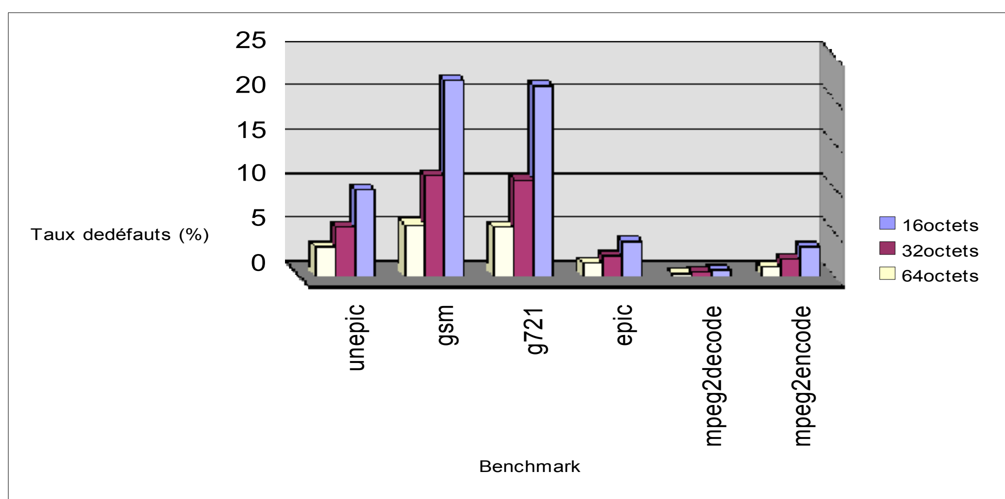


Figure IV.4: Taux de défauts pour un cache de données de 64 KOctets

Discussion

On constate que la taille du bloc influe directement sur le taux des défauts du cache, plus on augmente la taille du bloc pour un même cache plus le taux des défauts du cache diminue.

Par exemple le taux des défauts pour un cache ayant un bloc de 16 octets et une capacité du cache de 64 koctets dans le cas du benchmark epic est de 3,94% alors que ce taux devient 1,57% pour un cache ayant la même capacité du cache et une taille du bloc de 64 octets.

La plus grande amélioration du taux de défauts de cache est observée pour le benchmark gsm-toast dans le cas d'un cache de 64 koctets, ce benchmark avait un taux de défauts de 22.2 % pour un bloc de 16 octets, 11.44% pour un bloc de 32 octets et 5.92 % pour un bloc de 64 octets représentant ainsi une amélioration du taux entre un bloc de 16 octets et celle de 64 octets de 73,33%.

L'amélioration la plus faible est observée pour le benchmark mpeg-decode dans le cas d'un cache de 16 Koctets, ce benchmark avait un taux de défauts de 2.55% pour un bloc de 16 octets, 2.37% pour un bloc de 32 octets, 2.36 % pour un bloc de 64 octets représentant ainsi une amélioration du taux entre un bloc de 16 octets et celle de 64 octets de 7,45%.

Le meilleur taux de défauts pour les applications vidéo représentées par les benchmarks MPEG est celui de mpeg2dec dans le cas d'un cache de 64 Koctets ayant un bloc de 64 octets avec un taux de 0.37%, alors que les applications audio représentées par les applications GSM et G721, le meilleur

taux de défauts est celui de g721-encode dans le cas d'un cache de 64 Koctets ayant un bloc de 64 octets avec un taux de 5.67%. Pour les applications de traitement d'images représentées dans notre cas par le benchmark epic le meilleur taux de défauts est celui de l'epic dans le cas d'un cache de 64 Koctets ayant un bloc de 64 octets avec un taux de 1.57%. (Voir les figure IV.1, figure IV.2, figure IV.3, figure IV.4).

- **Capacité du data cache variable et taille du bloc fixe**
  - ✓ **Taille du bloc = 16 Octets et capacité du data cache varie entre 8Koctets, 16 Koctets, 32 Koctets et 64 Koctets**

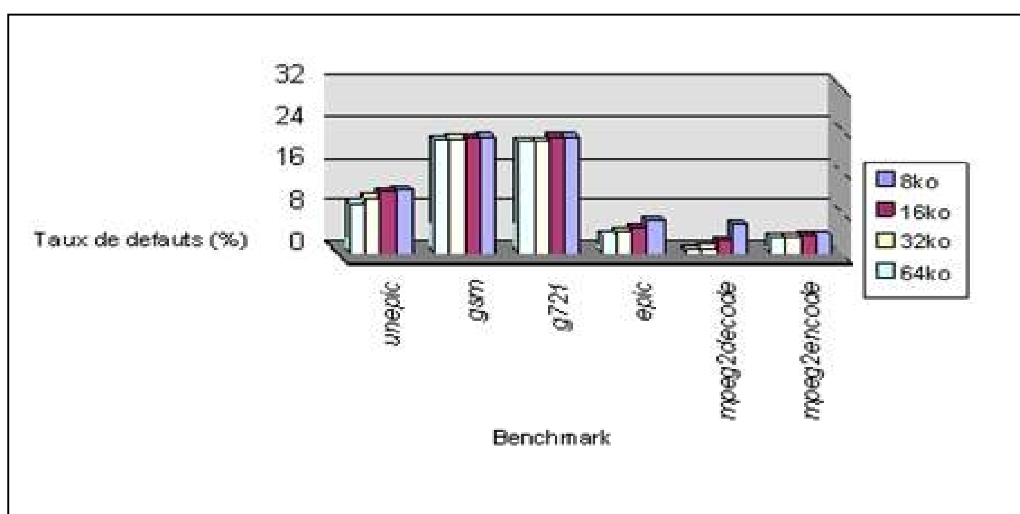


Figure IV.5: Taux de défauts pour une taille du bloc de 16 Octets

✓ **Taille du bloc = 32 Octets et capacité du data cache varie entre 8Koctets, 16Koctets, 32 Koctets et 64 Koctets**

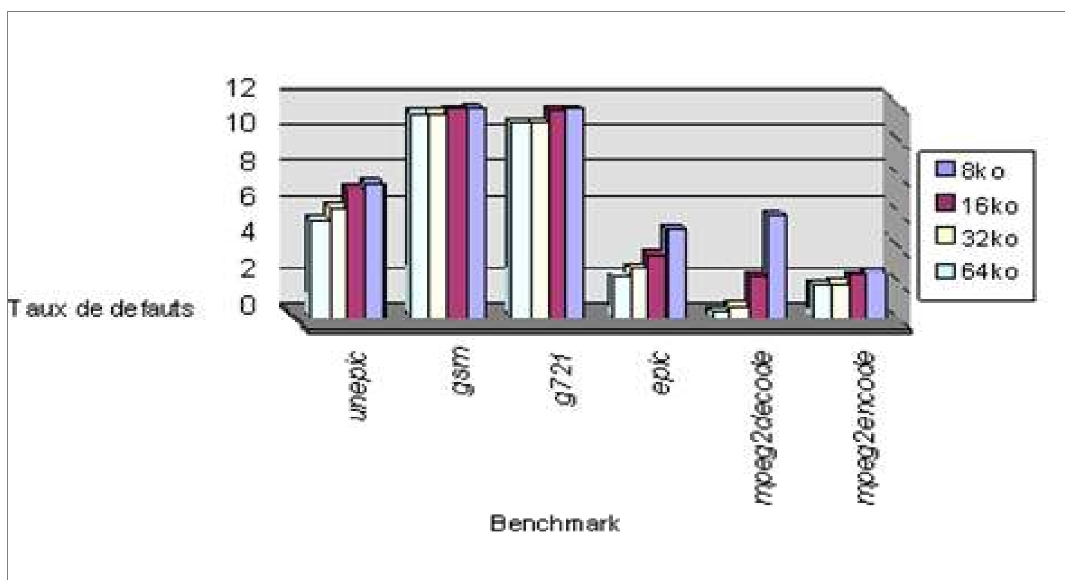


Figure IV.6: Taux de défauts pour une taille du bloc de 32 Octets

✓ **Taille du bloc = 64 Octets et capacité du data cache varie entre 8Koctets, 16 Koctets, 32 Koctets et 64 Koctets**

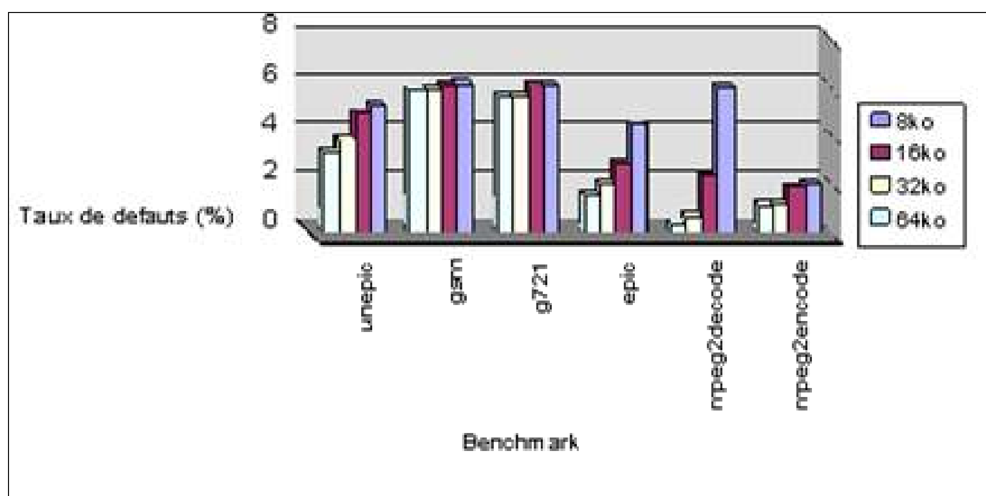


Figure IV.7: Taux de défauts pour une taille du bloc de 64 Octets

Discussion

On constate que la capacité du cache influe directement sur le taux des défauts du cache, plus on augmente la capacité du cache plus le taux des défauts du cache diminue. Mais le choix de la capacité du cache pour les applications embarqués est délicat car un cache de grande taille implique une consommation excessive de l'énergie et de l'espace.

Par exemple le taux des défauts pour un cache de 8 Koctet ayant une taille du bloc de 32 octets dans le cas du benchmark mpeg2decode est de 5,82% alors que ce taux devient 0,79% pour un cache ayant la même taille du bloc et une capacité du data cache de 64 Koctets.

La plus grande amélioration du taux de défauts du data cache est observée pour le benchmark mpeg2dec, ce benchmark avait un taux de défauts de 6.02% pour un data cache de 8koctets, 2.36% pour 16 Koctets, 0.7 % pour 32 Koctets et 0.37% pour un data cache de 64Koctets et ceci pour un bloc de 64 octets représentant ainsi une amélioration du taux entre un data cache de 8koctets et celui de 64 koctets de 93,85%.

L'amélioration la plus faible est observée pour le benchmark GSM-toast, ce benchmark avait un taux de défauts de 22.63% pour un data cache de 8koctets, 22.57% pour 16 Koctets, 22.25 % pour 32 Koctets et 22.2% pour un data cache de 64Koctets et ceci pour un bloc de 16octets représentant ainsi une amélioration du taux entre un data cache de 8koctets et celui de 64 koctets de 1,90%.

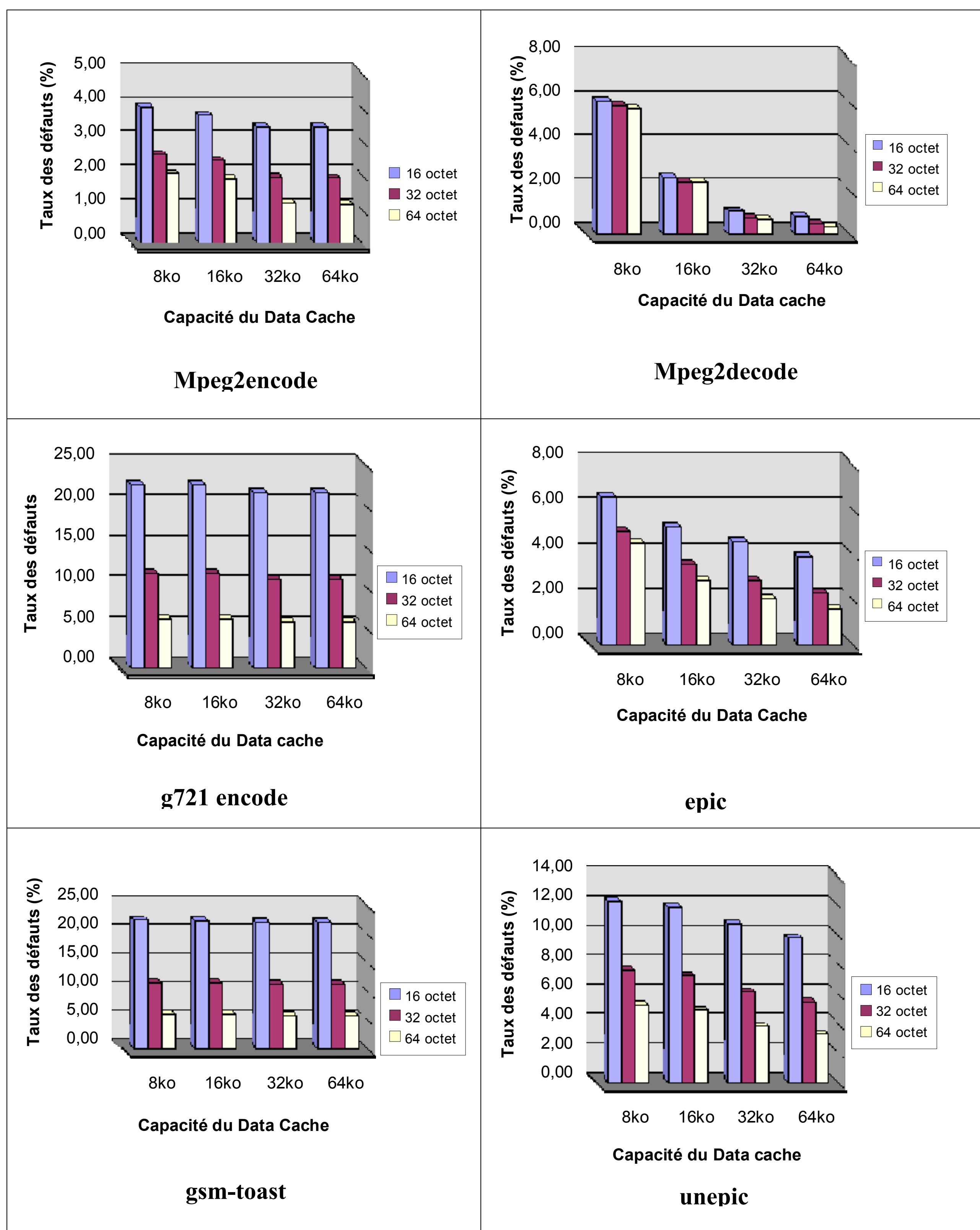


Figure IV.8: Taux de défauts du cache de données en fonction de la taille du bloc et la capacité du cache

IV.5.2. Temps d'accès au cache

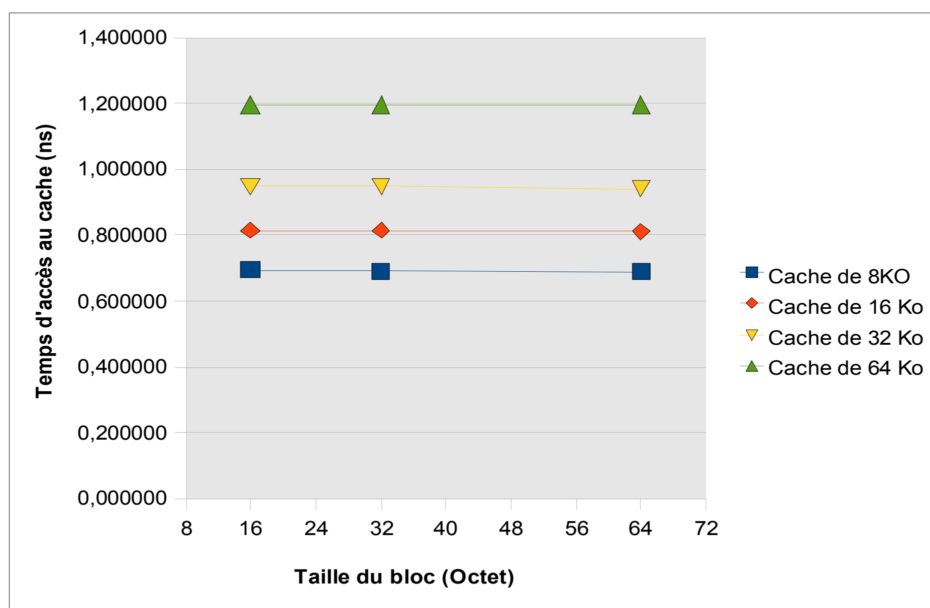


Figure IV.9: Variation du temps d'accès au data cache en fonction de la taille du bloc

Discussion

Comme on le remarque sur la figure IV.9 la taille du bloc a un léger effet sur le temps d'accès, cette effet est de 0,81% au maximum pour un data cache de 16Koctets avec un bloc de 64 octets par rapport à un data cache ayant la même capacité mais avec un bloc de 16 octets. Par contre la capacité du data cache influx grandement sur le temps d'accès ainsi plus on augmente la capacité du data cache plus le temps d'accès augmente. Lorsqu'on double la capacité du data cache on remarque une augmentation du temps d'accès d'au moyenne 83.5 % par exemple en passant d'un data cache de 8Koctets à un data cache de 16Koctets on a une augmentation du temps d'accès de 85.21%, ce taux devient 86.06 lorsqu'on passe d'un data cache de 16Koctets à un data cache de 32Koctets.

Calcul du temps d'accès global des benchmarks:

Pour calculer le temps d'accès pour chaque benchmark on a utilisé la formule donnée dans [48]:

$$Tb= H * Tc + M * Tm \qquad (1)$$

Où:

Tb: est le temps d'accès au cache du benchmark

H: Le nombre de hit calculé par SimpleScalar

Tc: est le temps d'accès au cache (temps du hit) calculé par CACTI.

M: est le nombre de misses (défauts du cache) calculé par SimpleScalar.

Tm: est le temps d'accès à la mémoire du second niveau (Dans notre cas on a considéré uniquement le cache de niveau 2).

Vu que le temps d'accès à la mémoire du second niveau dépend du type et de la technologie de mémoire et ce temps est supérieure de 5 à 10 fois par rapport au temps d'accès au cache du premier niveau [9], alors on a pris :

$Tb = H * Th + M * K$ *(2)*

Où $K = 7.5$ dans notre cas.

Les résultats obtenus sont récapitulés sur la figure IV.10

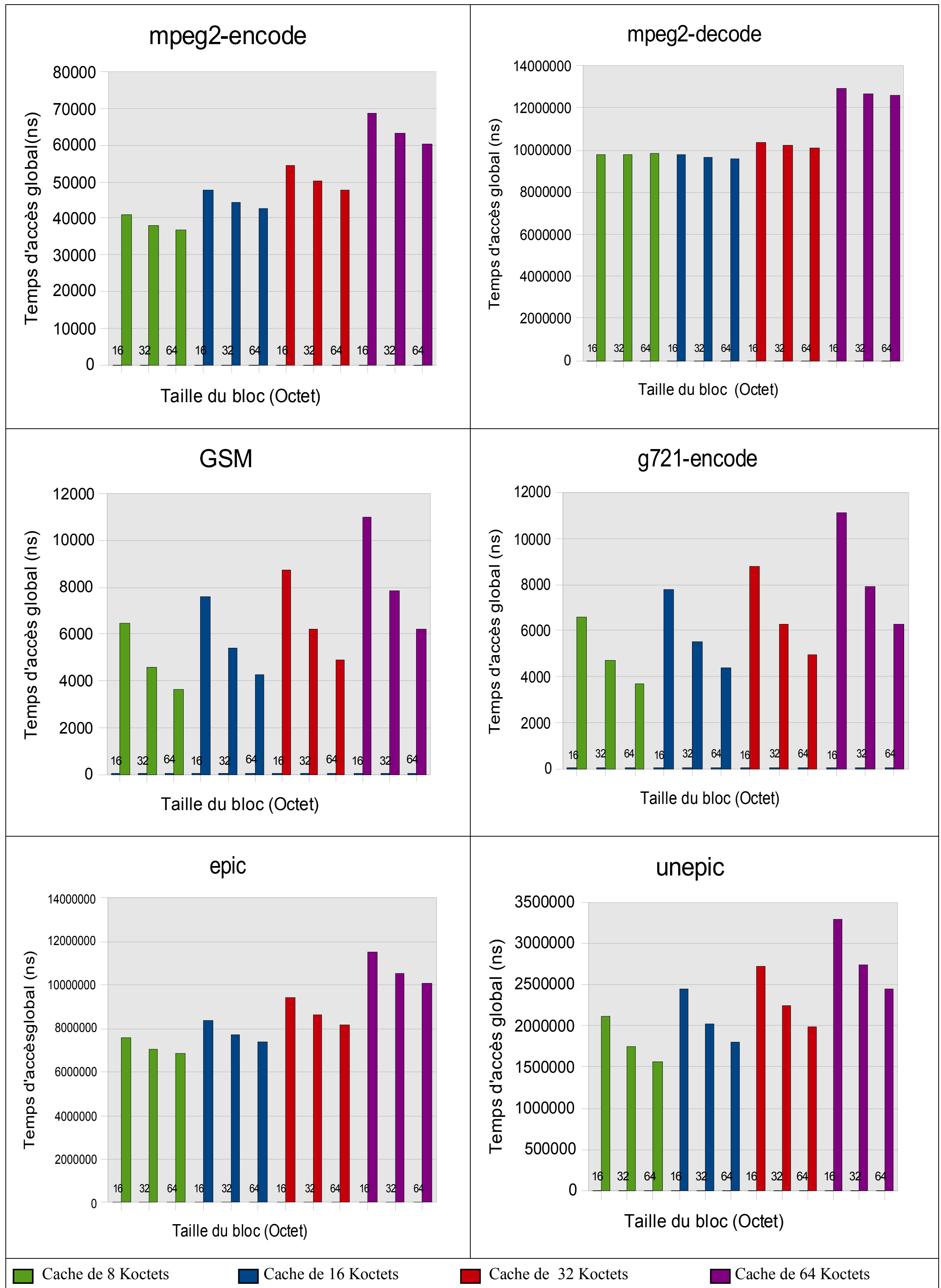


Figure IV.10: Variation du temps d'accès au data cache des benchmarks en fonction de la taille du bloc.

Discussion

Dans la section IV.1 on a déduit que plus on augmente la taille du bloc plus on réduit le taux de défauts du cache ainsi on réduit le nombre d'accès à la mémoire externe**.** La réduction des accès externe signifie une réduction du temps consommé lors des accès au cache ceci est bien observé pour tous les benchmarks sur la figure IV.10. Dans le cas du benchmark GSM par exemple lorsqu'on passe d'un cache de 64 Koctets et d'un bloc de 16 Octets à un cache de 64 Koctets ayant un bloc de 64 Octets on a une réduction du temps d'accès de 56. 65%.

Pour la capacité du cache, elle influx elle aussi directement sur le temps d'accès, plus on augmente la capacité plus le temps d'accès augmente.

On remarque pour la majorité des benchmarks que la meilleure configuration pour le temps d'accès est obtenue lors de l'utilisation d'un cache de 8Koctets avec un bloc de 64octets.

IV.5.3. Surface du cache

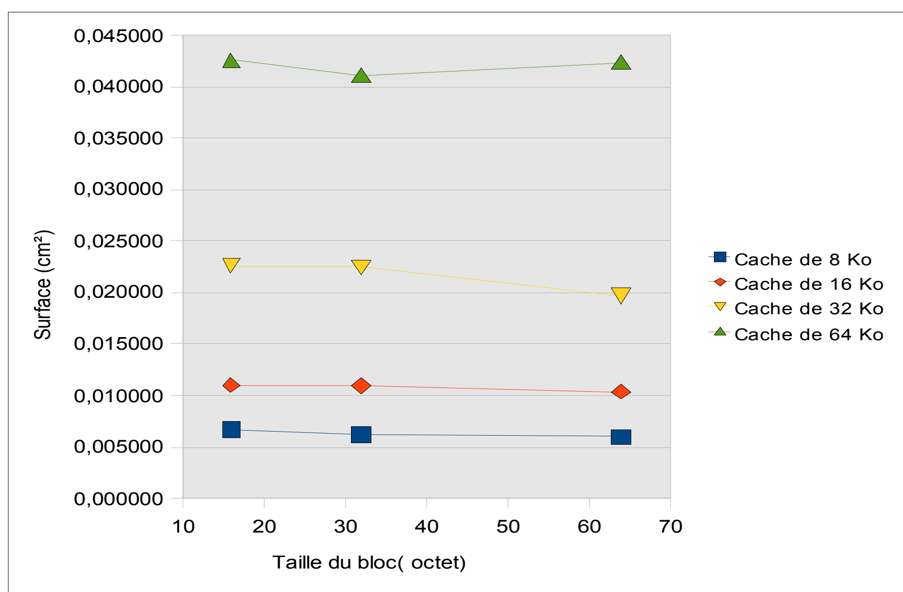


Figure IV.11:Variation de la surface en fonction de la taille du bloc du data cache

Discussion

On remarque dans ce cas que plus on augmente la capacité du cache plus la surface occupée par le cache augmente elle aussi. La taille du bloc n'a pas d'influence sur la surface du cache, par contre lorsqu'on augmente la capacité du cache la surface occupée devient plus importante.

Par exemple lorsqu'on double la capacité on a une augmentation d'au moyenne de 46%, ce taux devient 84.36% lorsqu'on passe d'un cache de 8Koctets à un cache de 64 Koctets.

IV.5.4. Energie consommée par accès

Il existe deux principaux composants constituant la source de la dissipation d'énergie dans les circuits CMOS, cette dernière est composée de l'énergie statique due à la fuite courante et la l'énergie dynamique due au chargement et déchargement des capacités pour la sauvegarde des données.

L'énergie dynamique par accès au cache est égale à l'énergie dynamique de tous les circuits du cache multipliée par le temps d'accès au cache. L'énergie statique est égale à l'énergie statique multipliée par le temps. L'énergie dynamique constitue la partie la plus importante de la dissipation d'énergie dans la technologie CMOS mais l'énergie statique est en augmentation par rapport à l'énergie totale. On considère les deux types dans notre étude.

Au début on a calculé l'énergie consommée par accès au cache en utilisant CACTI, après on a calculé l'énergie consommée par les accès au cache et ceci pour chaque benchmark (Voir la figure IV.12 et figure IV.13).

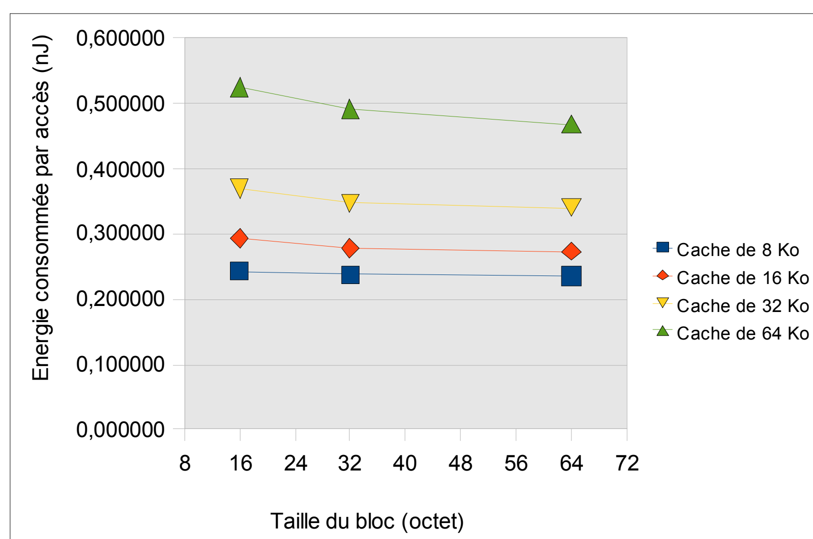


Figure IV.12: Variation de l'énergie consommée par accès en fonction de la taille du bloc du cache

Discussion

La taille du bloc influx positivement mais légèrement sur l'énergie consommée lors d'un accès au cache, comme on peut le remarquer sur la figure IV.12, pour un cache de 8Koctet par exemple ayant une taille du bloc de 64 Octets on a une réduction de 3.27% par rapport à un cache ayant la même taille et un bloc de 16 Octets.

Pour la capacité du data cache on remarque que plus on augmente la capacité du data cache plus l'énergie devient importante. Une augmentation de 17.19 % est observée pour un data cache ayant un bloc de 16 Octets et une capacité de 32 Koctets par rapport à un data cache de 8 Koctets alors que pour un data cache ayant un bloc de 64 Octets et une capacité de 64 Koctets on a une augmentation de 53.6 % par rapport à un data cache ayant une capacité de 8 Koctets et ayant la même taille du bloc ceci est dû au fait que le bloc d'adresse/données devient plus grand nécessitant ainsi plus de capacités à charger par accès.

IV.5.5. Calcul de l'énergie totale consommée par les benchmarks:

Pour calculer l'énergie consommée par tous les accès au data cache par un benchmark on a utilisé l'équation de calcul de l'énergie totale due à l'accès à la mémoire donnée dans [48]:

energy_mem = energy_dynamic + energy_static (3)

ou :

*energy_dynamic = cache_hits * energy_hit + cache_misses * energy_miss* (4)

energy_misses = energy_offchip_access + energy_Up_stall + energy_cache_block_fill (5)

*energy_static = cycles * energy_static_per_cycle* (6)

ou

energy_Up_stall est l'énergie consommée par le processeur lors de l'attente du chargement des données.

Energy_cache_block_fill est l'énergie consommée lors de l'écriture de tout un bloc dans le cache.

energy_offchip_access est l'énergie consommée lors d'un accès à une mémoire off chip.

On calcule *cache_hits* et *cache_misses* en utilisant simplescalar et on calcule energy_hit pour chaque configuration du cache en utilisant CACTI.

Les deux premiers paramètres de calcul de l'energy_misses (*energy_offchip_access, energy_Up_stall)* dépendent hautement du type de mémoire et du processeur utilisé et pour les avoir on doit évaluer un processeur réel ce qui n'est pas possible dans notre cas. L'étude faite par [48] basée sur l'observation des trois paramètres de calcul de *energy_miss* sur des mémoires et des processeurs typiques et commerciaux déduit que *energy_miss* est plus grande de 50 à 200 que *energy_hit* pour la technologie 0.13 micromètre, ainsi il définit *energy_miss* comme :

*energy_miss = k_miss_energy * energy_hit* (7)

ou *k_miss_energy* est égale à 50 et 200. Pour notre étude on a pris *k_miss_energy* égale à (200 + 50)/2 = 125

Finalement cycles est le nombre total de cycles pour qu'un benchmark soit exécuté, ce paramètre est calculé en utilisant simplescalar.

energy_static_per_cycle est l'énergie statique totale consommée par cycle, cette valeur dépend elle aussi hautement du système utilisé comme indiqué dans [48], ce dernier la définit après l'avoir vérifié sur des systèmes typiques et commerciaux par :

*energy_static_per_cycle = k_static * energy_total* *(8)*

Où *k_static* est de 30% et 50 % de l'énergie totale.

Or dans notre cas on a pris k_static = (30%+50%)/2 = 40%

On a obtenu les résultats suivants pour les benchmarks.

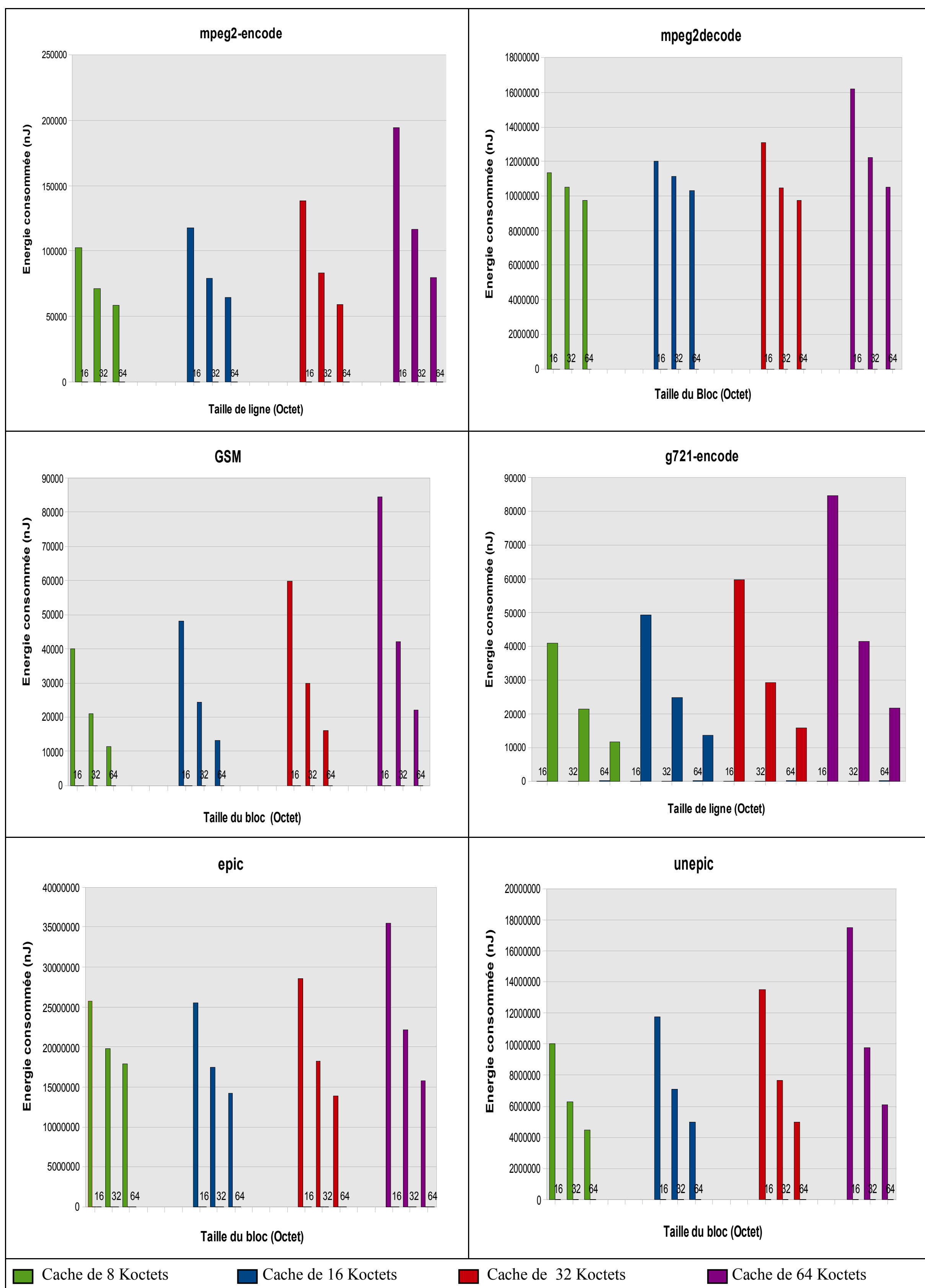


Figure IV.13: Variation de l'énergie consommée lors des accès au data cache par les benchmarks

Discussion

Dans la section IV.1 on a remarqué que plus on augmente la taille du bloc plus on réduit le taux de défauts du cache ainsi on réduit le nombre d'accès à la mémoire externe. La réduction des accès externe signifie une réduction d'énergie ceci est bien observé pour tous les benchmarks sur la figure IV.13. Par exemple pour le benchmark mpeg2-decode et dans le cas d'un cache ayant une capacité de 32 Koctets et une taille de 64 Octets on observe une réduction de 25.53% par rapport à un cache ayant la même capacité et un bloc 16 Octets. Cette réduction est plus considérable pour un cache de 64 Koctets où on observe une amélioration de 35.05% d'un cache ayant un bloc de 64 Octets par rapport à celui de 16 Octets.

Pour tous les autres benchmarks l'augmentation de la capacité du data cache augmente à son tour l'énergie consommée par les benchmarks lors de leurs exécutions.

IV.5.6. Interaction entre les paramètres : énergie, espace et temps d'accès

Il existe une forte interaction entre l'espace, la performance et la consommation d'énergie, plus on réduit la capacité du data cache plus ce cache consomme moins d'énergie.

Les résultats de notre étude montrent qu'une réduction de la consommation ainsi qu'une amélioration des performances des applications multimédia embarquées sont observées lors de la réduction du taux des défauts du data cache.

Optimiser en même temps les trois paramètres espace, énergie et performance est une opération très difficile car certains paramètres sont conflictuels, pour la majorité des benchmarks réduire la capacité du data cache engendre une croissance des défauts du data cache menant à des accès excessifs à la mémoire centrale donnant lieu à une croissance du temps d'accès et de consommation d'énergie.

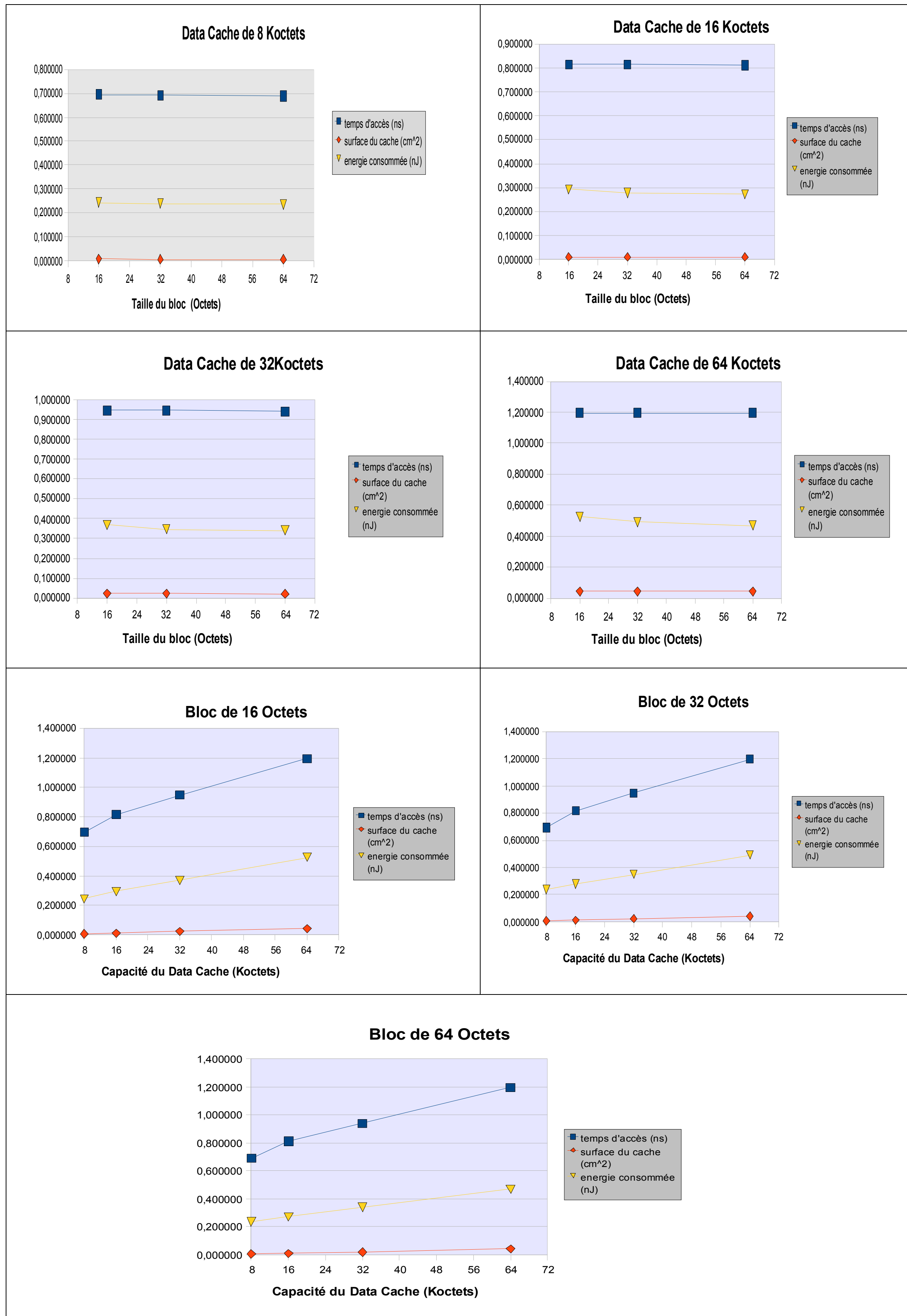


Figure IV.14: Interaction entre temps d'accès, surface du cache et énergie consommée du data cache

IV.6. Résultats expérimentaux d'un cache partitionné (Scalar cache et Array cache)

Dans cette partie on présente notre approche qui consiste à partitionner le cache de données en deux sous caches l'un destiné pour les données présentant une localité spatiale (tableaux) et l'autre pour les données présentant une localité temporelle (scalaires), ce qui nous offre une meilleure exploitation des localités exhibée par chaque type de données.

IV.6.1. Paramètres du cache partitionné

Pour le choix des paramètres de chaque cache on a jugé que les deux caches seront des caches directement associatifs, vu que ce type est plus convenable pour les systèmes embarqués, Array Cache aura un bloc de large taille pour permettre le préchargement des données exploitant ainsi au mieux la localité spatiale exhibée par ce type de donnée. Scalar cache est un cache ayant un bloc de taille relativement petite par rapport au bloc du cache Array Cache et ceci dans le but d'exploiter au maximum la localité temporelle exhibée par ce type de donnée en offrant la possibilité d'avoir plus de blocs dans le cache, ainsi le chargement de plus de données non contiguées en même temps.

Pour la capacité du cache, on a jugé l'utilisation des combinaisons suivantes:

Taille du Scalar Cache est le double de la taille du Array Cache

Le choix des capacités est pris en prenant en compte des capacités offrant moins d'espace, moins de temps d'accès et consommant moins d'énergie et ceci en se basant sur les résultats de la section IV.

Pour Array Cache : taille du bloc 16, 32 et 64 Octets et la taille du cache est de 4 et 8 et 16 Koctets.

Pour Scalar cache : Taille du bloc 8, 16 et 32 Octets et la taille du cache est de 8 et 16 et 32 Koctets.

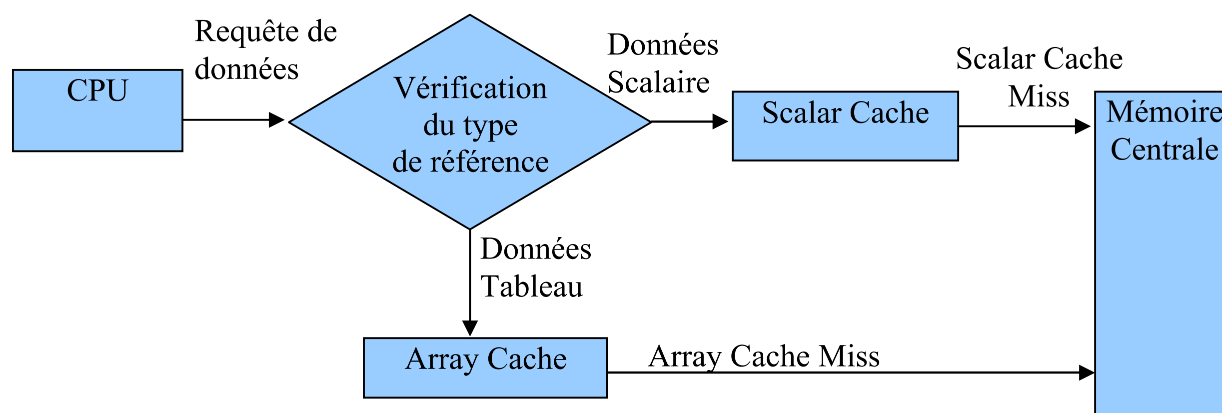


Figure IV.15: Principe de fonctionnement du Cache Partitionné

Dans notre étude on a considéré un seul niveau seulement pour le cache: le niveau interne.

IV.6.2.Mesures de performance

Pour le cache partitionné, on s'est limité seulement à la mesure du taux de défauts du cache partitionné.

Taux de défauts effectif du cache partitionné selon [24] est égale à :

$$EMR = (AMR * RA) + (SMR * RS) \qquad (9)$$

Où

EMR = Taux de défauts effectif (Effectif Miss Rate)

AMR = Taux de défaut de l'Array Cache (Array Miss Rate)

RA = Nombre de Références des tableaux/ Nombre de Références Total

SMR = Taux de défauts du Scalar Cache (Scalar Miss Rate)

RS = Nombre de Références des scalaires / Nombre de Références Total

Pour L'implémentation de notre cache partitionné, on a opté à une annotation des références pour distinguer celles des scalaires par rapport à celles des tableaux puis on a modifié SimpleScalar pour reconnaître les deux types de références et mapper chaque type au cache correspondant.

IV.6.3.Résultats expérimentaux du cache partitionné

IV.6.3.1. Cache des scalaires (scalar cache)

❖ **Taille du Scalar Cache fixe et taille du bloc variable:**

✓ **Capacité du Scalar cache = 8Koctet et taille de bloc varie entre 8, 16 et 32 octets**

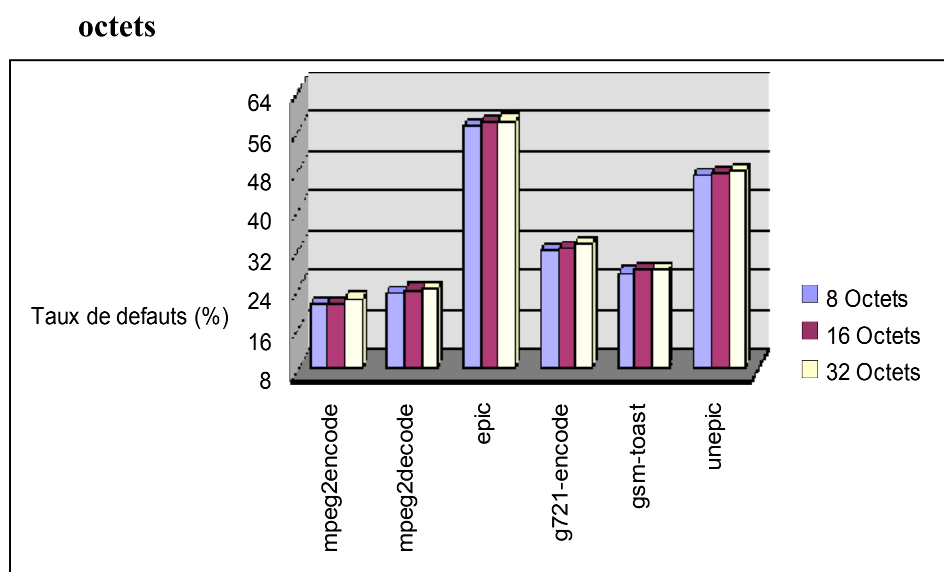


Figure IV.16: Taux de défauts pour un Scalar Cache de données de 8 KOctets

✓ **Capacité du Scalar cache = 16Koctet et taille de bloc varie entre 8, 16 et 32 octets**

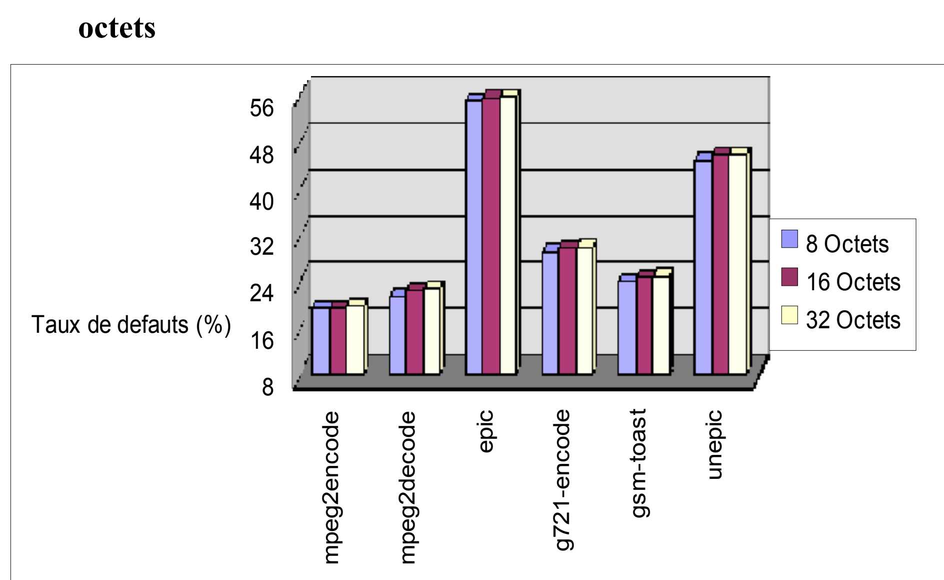


Figure IV.17: Taux de défauts pour un Scalar Cache de données de 16 KOctets

✓ **Capacité du Scalar cache = 32Koctet et taille de bloc varie entre 8, 16 et 32 octets**

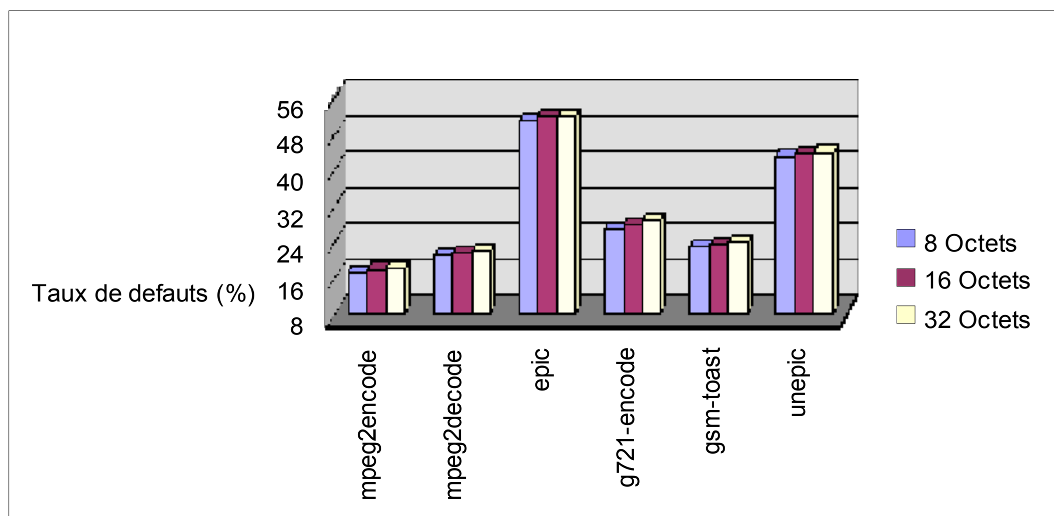


Figure IV.18: Taux de défauts pour un Scalar Cache de données de 32 KOctets

Discussion :

On constate d'après les figures *Figure IV.16, Figure IV.17 et Figure IV.18* que l'augmentation de la taille du bloc dans le cas du Scalar Cache mène à une légère augmentation du taux de défauts du cache, ceci est dû au fait que lorsqu'on augmente la taille du bloc on diminue le nombre de blocs dans le cache, ce qui engendre une augmentation des défauts de capacités ainsi que les défauts conflictuels, en plus le temps de chargement d'un bloc en cache devient plus long. L'augmentation des défauts dans notre cas varie entre 0,51% pour le benchmark epic, 1,45% pour unepic et 2,72% pour le benchmark g721-encode et ceci dans le cas d'un bloc de 16 Octets par rapport à un bloc de 32 Octets. Par contre ce taux est de 1,55% pour le benchmark epic et 4,45% pour mpegencode lorsqu'on passe d'un bloc de 8 Octets à un bloc de 32 Octets.

✓ **Taille du Scalar Cache variable et taille du bloc fixe:**

✓ **Taille du bloc = 8 Octets et Capacité du Scalar Cache varie entre 8, 16 et 32 KOctets**

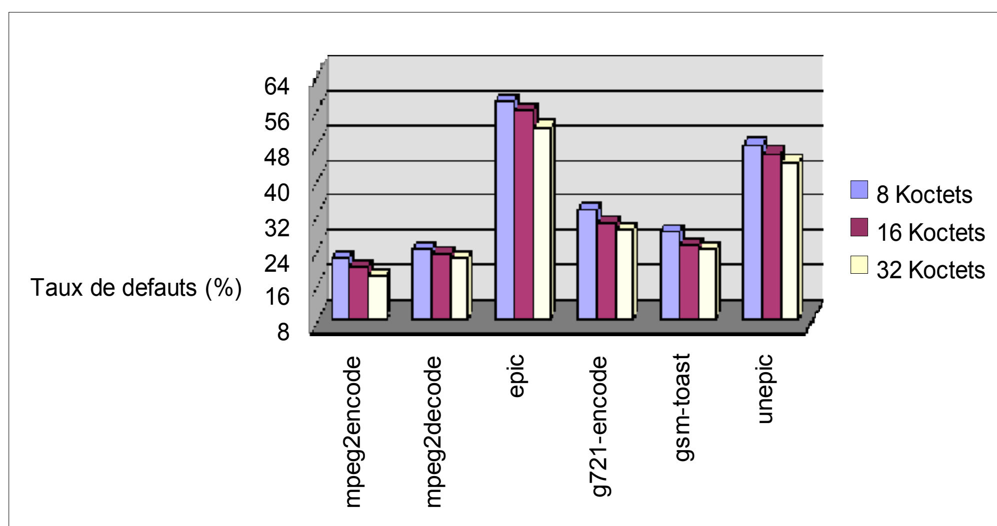


Figure IV.19: Taux de défauts du Scalar Cache pour une taille de bloc de 8 Octets

✓ **Taille du bloc = 16 Octets et Capacité du Scalar Cache varie entre 8, 16 et 32 KOctets**

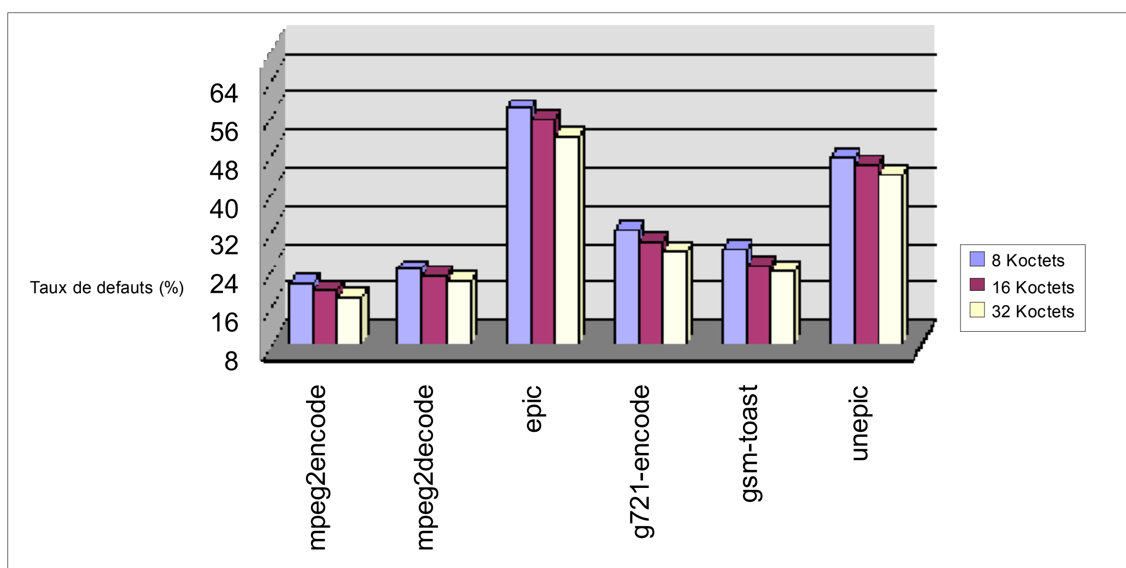


Figure IV.20: Taux de défauts du Scalar Cache pour une taille de bloc de 16 Octets

- ✓ **Taille du bloc = 32 Octets et Capacité du Scalar Cache varie entre 8, 16 et 32 KOctets**

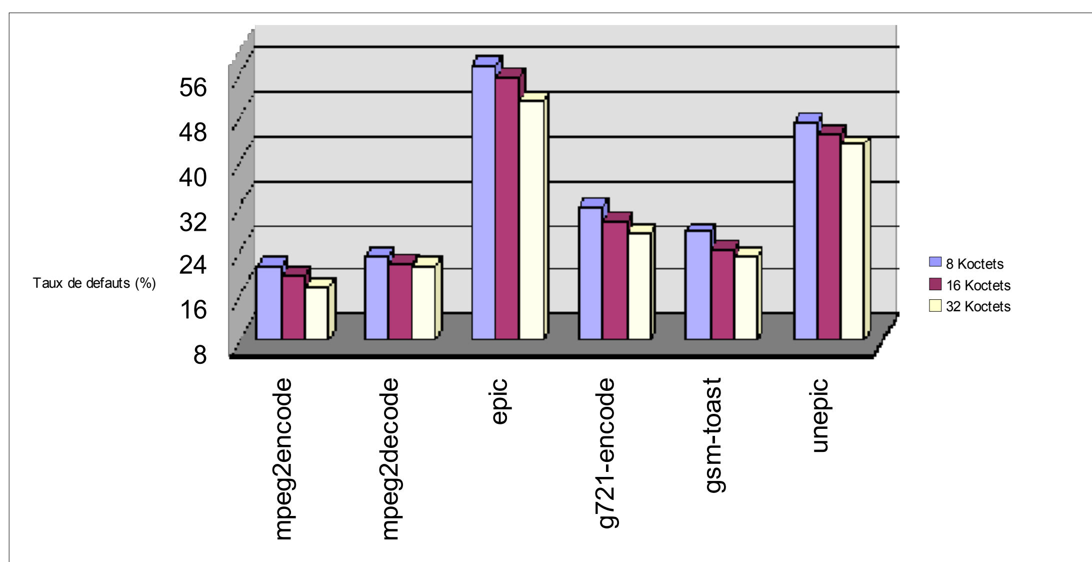


Figure IV.21: Taux de défauts du Scalar Cache pour une taille de bloc de 32 Octets

Discussion

D'après les résultats obtenus, on constate que l'augmentation de la capacité du Scalar Cache améliore considérablement le taux de défauts du Scalar Cache, ceci est dû au fait que plus on a de l'espace plus on peut charger des données en cache permettant ainsi de réduire le nombre des défauts.

D'une autre part comme indiqué dans [55] l'augmentation de la capacité du cache réduit certainement les défauts de capacités, cependant lorsque la capacité du cache augmente, les défauts de capacités deviennent des défauts conflictuels. Ceci est sans négliger l'effet négatif de l'augmentation de la capacité du cache sur l'augmentation du temps d'accès et de l'énergie consommée par le cache.

IV.6.3.2. Cache des Tableaux (Array Cache)

- **Taille du Array Cache fixe et taille du bloc variable:**

  - ✓ **Capacité du Array Cache = 4Koctet et taille de bloc varie entre 16, 32 et 64 Octets**

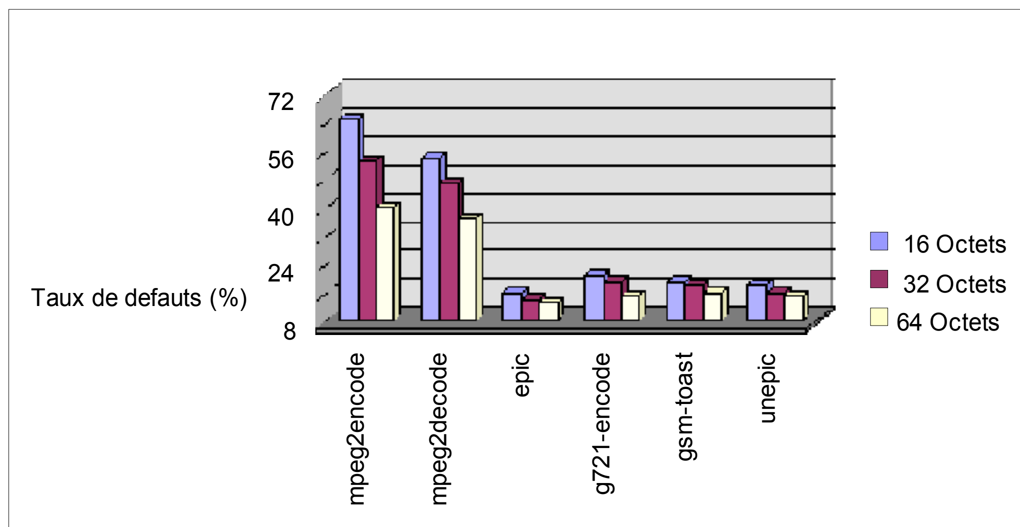


Figure IV.22: Taux de défauts de l'Array Cache pour une capacité de 4 KOctets

  - ✓ **Capacité du Array cache = 8Koctet et taille de bloc varie entre 8, 16 et 32 octets**

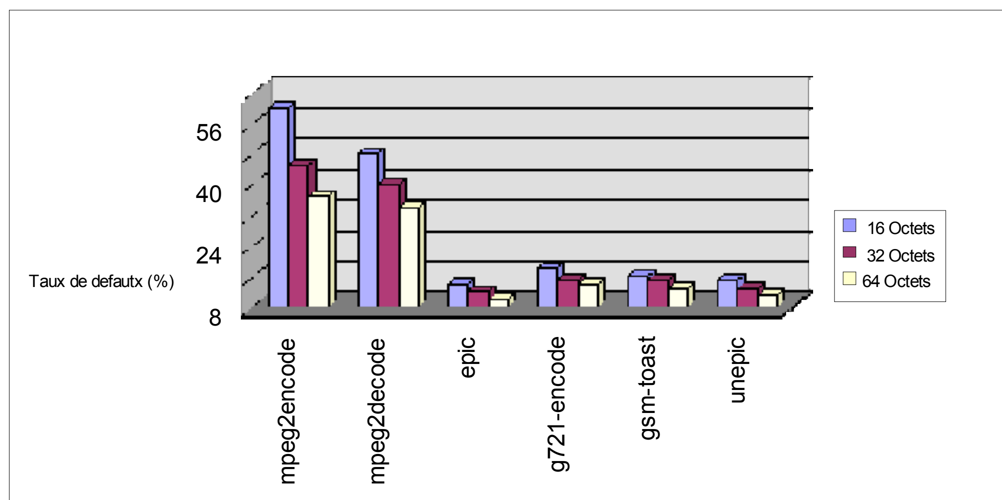


Figure IV.23: Taux de défauts de l'Array Cache pour une capacité de 8 KOctets

✓ **Capacité du Array cache = 16 Koctet et taille de bloc varie entre 8, 16 et 32 octets**

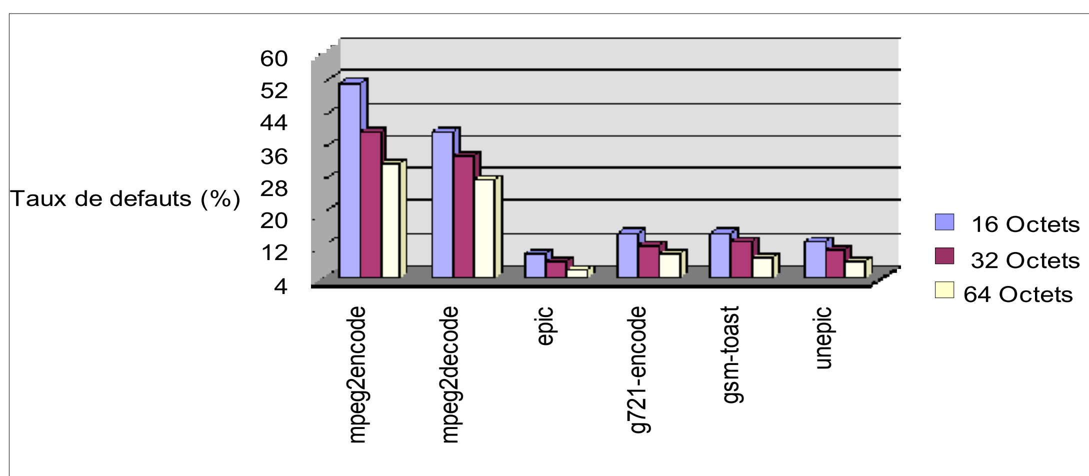


Figure IV.24: Taux de défauts de l'Array Cache pour une capacité de 16 KOctets

Discussion :

Comme il est indiqué par les résultats obtenus dans les figures *Figure IV.24, Figure IV.25 et Figure IV.24* et dans les trois cas étudiés l'augmentation de la taille du bloc démunie considérablement le taux de défauts du cache dans le cas de l'Array Cache, ceci s'explique par le fait de charger un bloc permet le préchargement des éléments suivants puisque les tableaux se caractérisent par la localité spatiale. Ce taux est de 6,25% pour le gsm-toast, 13,33% pour unepic et 16,66% pour mepg-encode lorsqu'on passe d'un bloc de 16 Octets à un bloc de 32 Octets dans le cas de l'Array Cache d'une capacité de 8 Koctets.

❖ Taille du Array Cache variable et taille du bloc fixe:

✓ **Taille du bloc = 16 Octets et Capacité du Array cache varie entre 4, 8, et 16 Koctets**

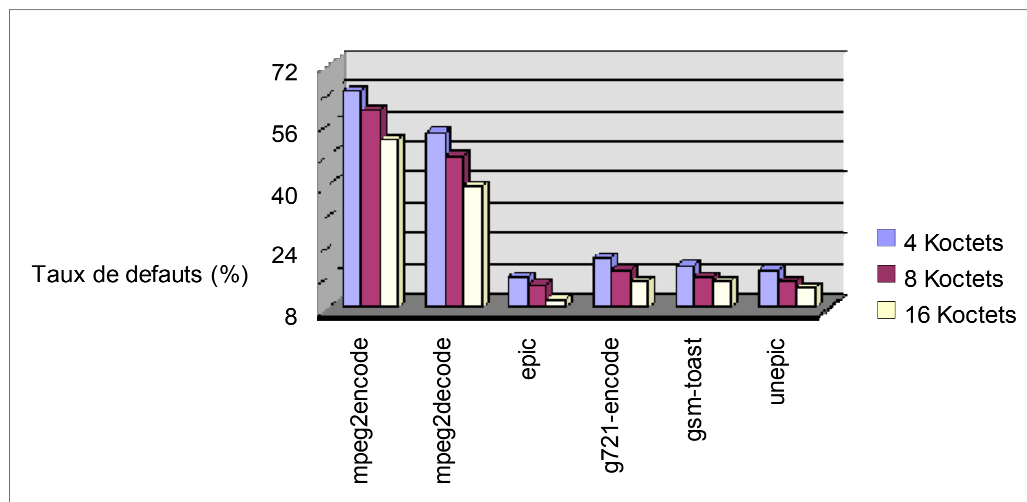


Figure IV.25: Taux de défauts de l'Array Cache pour un bloc de 16 Octets

✓ **Taille du bloc = 32 Octets et Capacité du Array cache varie entre 8, 16 et 32 KOctets**

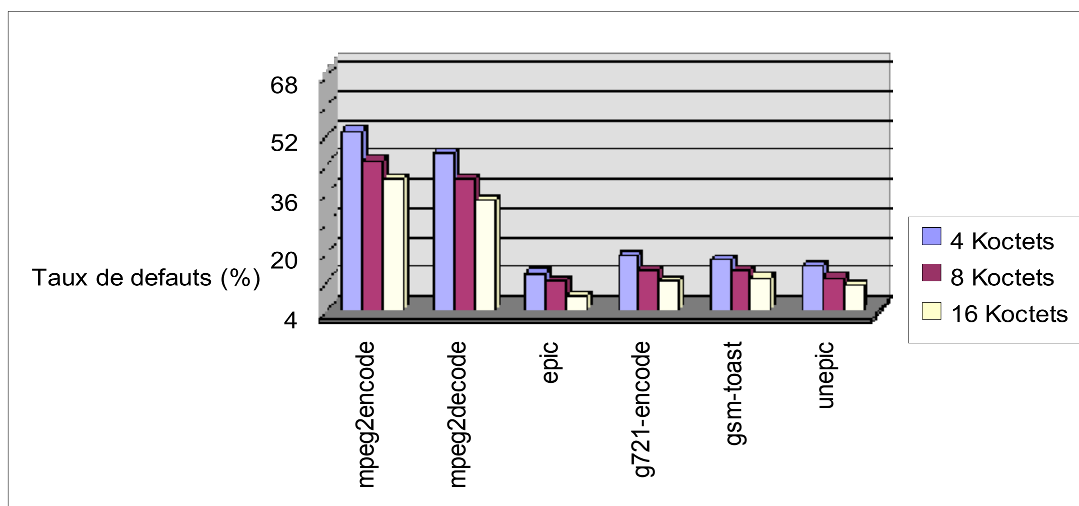


Figure IV.26: Taux de défauts de l'Array Cache pour un bloc de 32 Octets

✓ **Taille du bloc = 64 Octets et Capacité du Array cache varie entre 8, 16 et 32 KOctets**

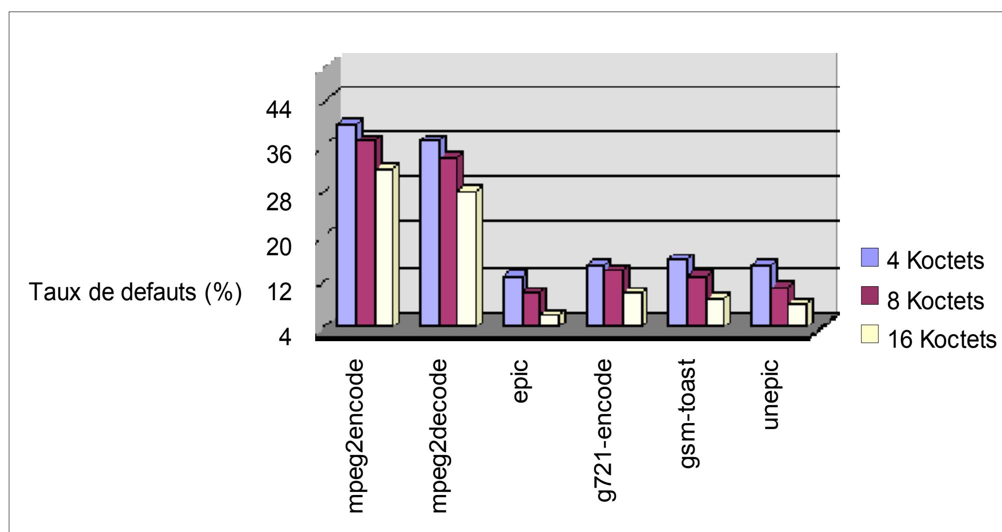


Figure IV.27: Taux de défauts de l'Array Cache pour un bloc de 64 Octets

Discussion

D'après les figures : *Figure IV.25, Figure IV.26 et Figure IV.27*, l'augmentation de la capacité du cache améliore le taux de défauts de l'Array Cache, ceci est dû au fait que plus on a de l'espace plus on peut charger des données en cache permettant ainsi de réduire le nombre des défauts. Ce taux est 36,84% pour g721-encode, 27,65% pour le mpeg-encode et 32% pour epic lorsqu'on passe de l'Array cache d'une capacité de 4Koctets à 16Koctets.

IV.6.3.3. Comparaison entre le cache partitionné (Array Cache & Scalar Cache) et le cache unifié

- ✓ **Comparaison entre un cache unifié de 16 Koctets avec un cache partitionné de 8kOctets pour les scalaires et 4 Koctets pour les tableaux**

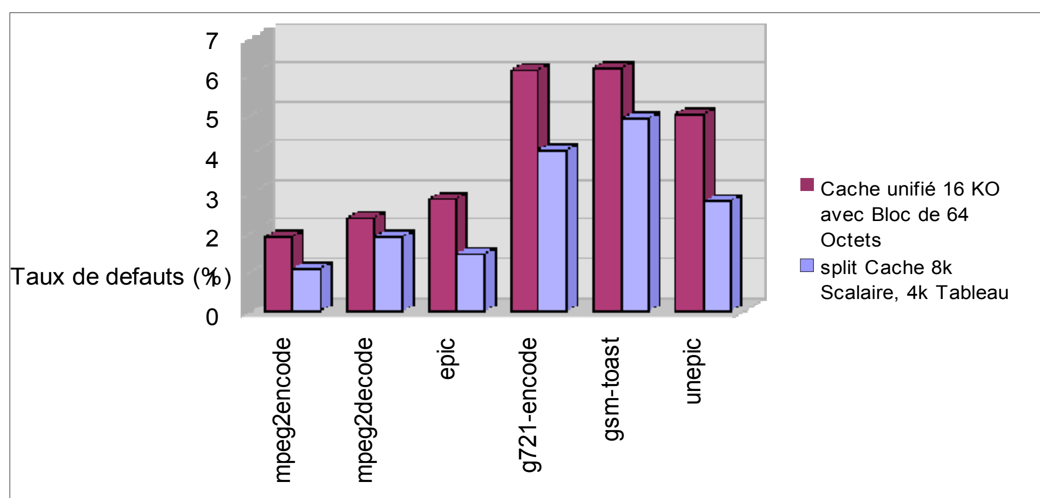


Figure IV.28: Variation du taux de défauts d'un cache de données unifié de 16 Koctets par rapport à un cache partitionné en Scalar Cache et Array Cache(8Koctets et 4 Koctets)

- ✓ **Comparaison entre un cache unifié de 16 Koctets avec un cache partitionné de 8kOctets pour les scalaires et 4 Koctets pour les tableaux**

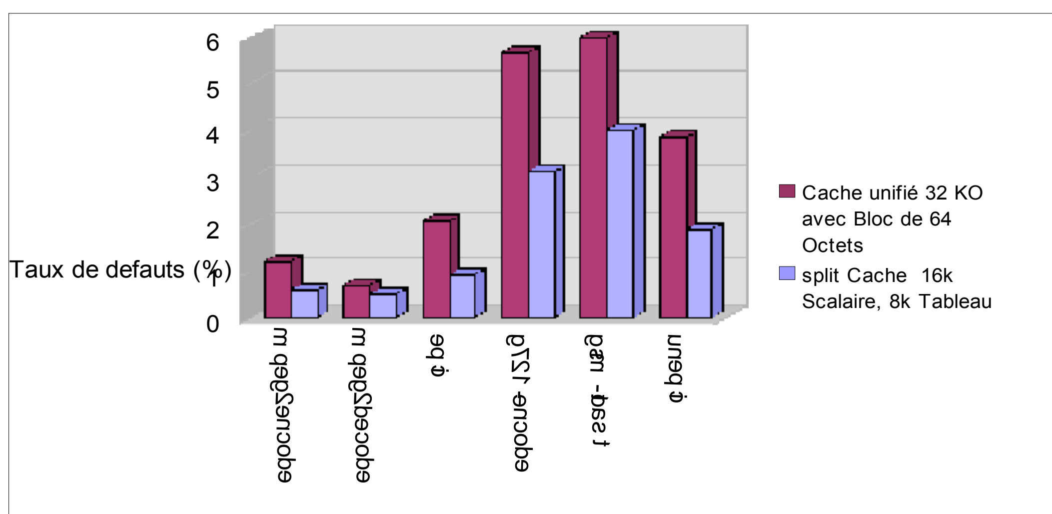


Figure IV.29: Variation du taux de défauts d'un cache de données unifié de 32 Koctets par rapport à un cache partitionné en Scalar Cache et Array Cache(16Koctets et 8 Koctets)

Discussion

Les résultats obtenus en comparant un cache partitionné en Scalar Cache et Array Cache (avec la capacité du Scalar Cache égale le double de la capacité de l'Array Cache) avec un cache de données unifié pour les scalaires et les tableaux montrent que le cache partitionné offre un meilleur taux de défauts pour tous les benchmarks. Par exemple dans le cas d'un cache unifié de 16 Koctets comparé avec un cache partitionné (8Koctets pour les scalaires et 4 Koctets pour les tableaux), le taux d'amélioration des défauts de cache varie entre 20,62% pour gsm-toast et 44,97% pour mpeg-encode. Dans le cas d'un cache unifié de 32Koctets comparé avec un cache partitionné (16Koctets pour les scalaires et 8Koctets pour les tableaux), ce taux d'amélioration varie entre 27,14 % pour le benchmark mpeg-decode et 55,83% pour le benchmark epic. Tout en signalant que la capacité des caches scalaires et tableaux et moins que celle du cache partitionné ce qui se répercute sur l'espace occupé par le cache et l'énergie consommée par le cache d'après les résultats de la section IV.4

Conclusion

CONCLUSION

De nos jours les mémoires caches sont devenus un élément vital dans l'amélioration des performances des systèmes. Dans les systèmes embarqués, ce type de mémoire joue un rôle important dans l'amélioration des performances de ces systèmes à condition que les paramètres soient choisis délicatement.

Dans ce mémoire ont a présenté au début une brève introduction sur les systèmes embarqués puis on a fait un état sur les mémoires utilisées dans les systèmes embarqués en précisant le rôle de l'hiérarchie mémoire dans ces systèmes; ensuite on a fait un survol sur les méthodes d'amélioration des performances du cache et plus précisément les méthodes de partitionnement du cache de données. Après on a proposé un partitionnement du cache de données en un cache pour les scalaires et un cache pour les tableaux; vu que le champ d'application visé par notre étude est les applications multimédia embarquées caractérisée par l'utilisation intensive des données.

Lors de l'étude expérimentale, on a étudié en premier lieu l'effet de la capacité du cache ainsi que la taille du bloc sur les performances du système, on a choisis les paramètres suivants*: le taux de défauts, surface, énergie consommée et le temps d'accès au cache des données* pour mesurer les performances des systèmes multimédia embarqués représentés dans notre cas par la suite des benchmarks mediabench. On a aboutit à une conclusion que ces paramètres sont en forte interaction, le taux de défauts diminue avec l'augmentation de la capacité et de la taille du bloc par contre les autres paramètres tels que : énergie, surface et temps d'accès qui

sont très important dans les systèmes embarqués augmentent, ainsi le choix de ces paramètres lors de la conception des systèmes embarqués doit être fait avec précision car un mauvais choix se répercute automatiquement et d'une manière désastreuse sur les performances du système.

On a remarqué que les données exposent des localités différentes et le cache de données unifié n'exploitent pas les localités exhibées par chaque type de donnée provoquant ainsi des mouvements de données non nécessaire entre processeur et mémoire, pour remédier à cette situation on a présenté une méthode qui consiste à partitionner le cache de données pour les applications multimédia embarqués en un cache pour les scalaires et un cache pour les tableaux, vu que ces deux types de données n'exposent pas le même type de localité, nous avons opté à un remplacement d'un cache unifié pour les données par un cache partitionné composé d'un cache désigné pour les scalaires ayant une capacité relativement grande par rapport à celle des tableaux et possédant une taille de bloc petite pour offrir plus de blocs dans le cache et permettre le chargement de plus de données en évitant évidemment les déchargements des données obligatoires du cache, ceci permet une exploitation maximale de la localité temporelle exposée par ce type de données, et un cache pour les tableaux ayant une capacité relativement petite à celle du cache des scalaires mais possédant à son tour un bloc de taille suffisamment grande pour une meilleure exploitation de la localité spatiale exposée par les tableaux.

Notre travail a démontré que le partitionnement du cache selon la localité inhérente des données : temporelle et spatiale permet d'améliorer significativement les performances du

système et ceci en offrant un taux de défauts et en occupant un espace inférieur à celui offert par le cache unifié.

Ce travail ouvre des horizons pour l'amélioration des performances de ces systèmes et ceci en étudiant en premier lieu l'effet du partitionnement sur l'énergie consommée, espace occupé et le temps d'accès au cache partitionné puis en intégrant de nouvelles architectures comme le scratch pad, cache victime ou autre type de mémoire au cache partitionné pour améliorer les performances des applications multimédia embarquées.

BIBIOGRAPHIE

[1] A. Janapsatya, "*Optimisation of instruction memory for embedded systems*", Thèse de doctorat, University of New South Wales, Sydney Australia, 2005.

[2] L. Wehmeyer, P. Marwedel, *Fast, Efficient and Predictable Memory accesses: Optimization Algorithms for Memory Architecture Aware Compilation*, Springer, Netherlands, 2006.

[3] G. Corre, "*Gestion des unités de mémorisation pour la synthèse d'architecture* ", Thèse de doctorat, Université Bretagne du Sud, 2005.

[4] T. S. Rajesh Kumar, C. P. Ravi Kumar, R. Govin darajan, "*MAX : A multi objective memory architecture exploration framework for embedded systems on chip* ", Proceedings of the International Conference on VLSI Design (VLSI-07), Bangalore, India, Janvier 2007.

[5] M. Portelan, "*Conception d'un système embarqué sûr et sécurisé* ", Thèse de doctorat, Institut National polytechnique de Grenoble, 2006.

[6] H. Benfradj, "*Optimisation de l'énergie dans une architecture mémoire multi-bancs pour des applications multi-tâches temps réel*", Thèse de doctorat, Université Nice-Sophia Antipolis, 2006

[7] E. Simeu, "*Test et surveillance intégrés des systèmes embarqués* ", Thèse de doctorat, Université Joseph Fourier Grenoble, 2005.

[8] Y. Le Mollec, "*Aide à la conception de systèmes sur puce hétérogènes par l'exploration paramétrable des solutions au niveau système* ", Thèse de doctorat, Université Bretagne Sud, Avril 2003

[9] P. Marwedel, *Embedded system design*, Kluwer Academic Publishers, 2003

[10] M. Azizi, "*Co vérification des systèmes intégrés*", Thèse de doctorat, Université Monréal, 2000

[11] P. Adhipathi, "*Model based approach to hardware/software partitioning of soc designs*", Thèse de master, Institut polytechnique de Virginie, 2004.

[12] M. J. Hamon, *"Méthodes et outils de la conception amont pour les systèmes et les microsystèmes "*, Thèse de doctorat, Institut national polytechnique de Toulouse, 2005.

[13] F. Donnet, *"Synthèse de haut niveau contrôlée par l'utilisateur "*, Thèse de doctorat, Université Paris VI, 2004.

[14] T. Bossart, *"Modèles pour l'optimisation de la simulation au cycle près de systèmes synchrones"*, Thèse de doctorat, Université Paris IV, 2006.

[15] *First steps with embedded systems*, Byte Craft Limited, Ontorio Canada, 2002. www.www.bytecraft.com

[16] www.wikipedia.org

[17] A. Fraboulet, *"Optimisation de la mémoire et de la consommation des systèmes multimédia embarqués"*, Thèse de doctorat, Institut National des Sciences Appliquées de Lyon, 2001.

[18] *www.wikelectro.com*

[19] J. L. Hennesy and D. A. Patterson. Computer ***Architecture: A Quantitative Approach***. Morgan aufmann Publishers, San Fransisco, CA, 2003

[20] Smith, A.J., *"Cache memories*", Computing Surveys", v. 14, n. 3, 1982, pp. 473-530.

[21] P. R. Panda, N. D. Dutt, A. Nicaulo, *"Efficient utilization of scratch pad memory in embedded processor applications"*, Proceeding of European Conference on Design and test, Paris, 1997.

[22] A. Naz, *"Split array and scalar data caches: a comprehensive study of data cache organization"*, Thèse de doctorat, University of North Texas , Aout 2007.

[23] P. R. Panda, F. Cathoor, N. D. Dutt, P. G. Kjeldsberg *"Data and memory optimization techniques for embedded systems"*, ACM Transactions on Design Automation of Electronic Systems , vol. 6, N°. 2, Avril 2001.

[24] A. Naz, K.M. Kavi, P.H. Sweany and M. Rezaei, *A study of separate array and scalar caches*, in Proceedings of the 18th International Symposium on High Performance Computing Systems and Applications (HPCS 2004), Winnipeg, Manitoba, Canada, Mai, 2004, pp 157-164.

[25] Prvulovic, M., Marinov D., Milutinovic V., "*A Performance Reevaluation of the Split Temporal/Spatial Cache,*" Workshop Digest of the PAID/ISCA-98, Barcelone, Espagne, Juin 1998

[26] Milutinovic, V., Markovic, B., Tomasevic, M., Tremblay, M., "*The Split Temporal/Spatial Cache: Initial Performance Analysis*," Proceedings of the SCIzzL-5, Santa Clara, California, USA, Mars 1996, pp. 63-69.

[27] Gonzalez, A., Aliagas, C. and Valero, M., *"A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality"*, Proceedings of the International Conference on Supercomputing (ICS '95), Barcelona, Spain, 1995, pp. 338-347.

[28] F.Catthoor, F.Balasa, T.Claes, E.De Greef, M.Eyckmans, F.Franssen, m.Janssen, L.Nachtergaele, M.Miranda, P.Petroni, H.Samsom, and S.Wuytack. *"Optimisation of global communication and memory organisation for decreased size and power in fixed-rate processing systems"*, Internal Report, Juin 1995.

[29] M. Balakrishnan, D. Banerji, A. Majumdar, J. Linders, J. Majithia, *"Allocation of multiport memories in data path synthesis"*, IEEE Transactions Computer Aided Design, Juillet 1990.

[30] D. Chillet, *"Méthodologie de conception architecturale des mémoires pour circuits dédiés au traitement de signal « temp reél »"*, Thèse de doctorat, Université Renne I, 1997.

[31] V. Milutinovic, "*The STS Cache,"* University of Belgrade Technical Report #35/95, Belgrade, Serbia, Yugoslavia, Janvier 1995

[32] F. J. Sanchez, A. Gonzalez, and M. Valero, *"Software Management of Selective and Dual Data Caches"*, IEEE TCCA Newsletters, Mars 97, pp. 3-10.

[33] A. Naz, K.M. Kavi, P.H. Sweany and W. Li, *"Tiny split data caches make big performance impact for embedded applications"*, Special Issue on Embedded Single-Chip Multicore Architectures and related research - from System Design to Application Support of ACM Journal of Embedded Computing, Volume 2 Issue 2, Novembre 2006,pp. 207- 219.

[34] J. H. Lee, J. S. Lee and S. D. Kim, *"A new cache architecture based on temporal and spatial locality"*, Journal of Systems Architecture, Vol. 46 N° 15, Septembre 2000, pp.1451-1467.

[35] M. Tomasko, S. Hadjiyiannis and W. A. Najjar, *"Experimental evaluation of array and scalar caches"*, IEEE Technical Committee on Computer Architecture Newsletter, Mars 1997, pp. 11-17.

[36] Chan, K. K., Hay, C. C., Keller, J. R., Kurpanek, G. P. Schumacher, F. X. Zheng, J., *"Design of the HP PA7200 CPU"*, Hewlett-Packard Journal, Fevrier 1996, pp. 1-12.

[37] Rivers, J. A., Davidson, E. S., *"Reducing Conflicts in Direct-mapped Caches with a Temporality Based Design"*, Proceedings of the International Conference on Parallel Processing, 1996.

[38] M. Prvulovic, D. Marinov , Z. Dimitrijevic, V. Milutinovic, *"The Split Spatial/Non-Spatial Cache: A Performance and Complexity Evaluation"*, IEEE TCCA Newsletters, 1999.

[39] N. JOUPPI, *"Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers"*. In Proceedings of the 17th Annual International Symposium on Computer Architecture (Seattle, WA), IEEE, New York, 1990, pp. 364 –373.

[40] R. E. Kessler, *"Analysis of Multi-Megabyte secondary CPU Cache Memories"*, Thèse de doctorat, Université de Wisconsin Madison, 1991.

[41]T. Lafage, *"Etude, réalisation et application d'une plate-forme de collecte de trace d'execution de programmes"*, thèse de doctorat, Université de Rennes 1, 2000.

[42] Agarwal, M. Horowitz, and J. Hennessy, "*An Analytical Cache Model*," ACM Transactions on Computer Systems, Vol. N° 2, Mai 1989, pp. 184-215.

[43] J. Edler and M. D. Hill, "*Dinero IV Trace-Driven Uniprocessor Cache Simulator*," http://www.cs.wisc.edu/˜markhill/DineroIV/.

[44] V. J. Reddi, A. Settle, et D. A. Connors, *"PIN: A Binary Instrumentation Tool for Computer Architecture Research and Education"*, www. *www.pintool.org*

[45] D. C. Burger and T. M. Austin, "*The SimpleScalar tool-set, Version 2.0*," Technical Report 1342, Department of Computer Science, UW, Juin, 1997, http://www.simplescalar.org

[46] G. Reinman, N. P. jouppi, 1999. *"CACTI2.0: An Integrated Cache Timing and Power Model"*, COMPAQ Western Research Lab, 1999.

[47] M. I. Aouad, O. Zendra, *"Outils de caractérisation du comportement mémoire et d'estimation de la consommation énergétique"*, Rapport de recherche, Février 2008.

[48] C. Zhang, F.Vahid and W.Najjar, *"A highly configurable cache architecture for embedded systems"*, in Proceedings of 30th Annual International Symposium on Computer Architecture, Juin 2003, pp.136 -146.

[49] M. Prvulovic, D. Marinov, *"Performance Evaluation of split Temporal/Spatial caches: Paving the way to new solutions"*, Proceeding of IEEE/ACM ISCA98, Juin, 1998, Barcelone, Espagne.

[50] C. Zhang, F.Vahid and W.Najjar, *"A Highly Configurable Cache for Low Energy Embedded Systems"*,ACM Transactions on Embedded Computing Systems, Vol. 4, No. 2, Mai 2005, pp 363-387.

[51] C. Zhang, F. Vahid and W. Najjar, *"Energy benefits of a configurable line size cache for embedded systems"*, IEEE International Symposium on VLSI Design, Tampa, Florida, Fevrier 2003.

[52] C. Zhang and F. Vahid, *"Using a victim buffer in an application-specific memory hierarchy"*, Design Automation and Test in Europe Conference (DATE), Février 2004, pp. 220-225.

[53] M.B. Kamble and K.Ghosse, *"Analytical energy dissipation models for low power caches"*, in Proceedings of International Symposium on Low Power Electronics and Design, Aout 1997, pp.143 -148.

[54] A. Naz, M. Rezaei, K. Kavi and P. Sweany, *"Improving data cache performance with integrated use of split caches, victim cache and stream buffers"*, in Proceedings of the Workshop on Memory performance dealing with applications, systems and architecture (MEDEA-2004), Juin 2005, Vol. 33 N° 3, pp. 41-48.

[55] C. Zhang, F. Vahid, W. Najjar, *"A Highly Configurable Cache for Low Energy Embedded Systems"*, ACM Transactions on Embedded Computing Systems, Mai 2005.

ANNEXE 1 : EXEMPLE DE SORTIE DU SIMPLESCALAR CAS DU MPEG2DECODE

```
sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).
sim: simulation started @ Fri Jun 19 17:45:09 2009, options follow:
sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.
# -config                         # load configuration from a file
# -dumpconfig                     # dump configuration to a file
# -h                        false # print help message
# -v                        false # verbose operation
# -d                        false # enable debug message
# -i                        false # start in Dlite debugger
-seed                           1 # random number generator seed (0 for timer
seed)
# -q                        false # initialize and terminate immediately
# -chkpt                   <null> # restore EIO trace execution from <fname>
# -redir:sim     /home/hadda/results/mpegdecode32-64.txt # redirect
simulator output to file (non-interactive only)
# -redir:prog              <null> # redirect simulated program output to file
-nice                           0 # simulator scheduling priority
-max:inst                       0 # maximum number of inst's to execute
-fastfwd                        0 # number of insts skipped before timing
starts
# -ptrace                  <null> # generate pipetrace, i.e.,
<fname|stdout|stderr> <range>
-fetch:ifqsize                  4 # instruction fetch queue size (in insts)
-fetch:mplat                    3 # extra branch mis-prediction latency
-fetch:speed                    1 # speed of front-end of machine relative to
execution core
-bpred                      bimod # branch predictor type
{nottaken|taken|perfect|bimod|2lev|comb}
-bpred:bimod     2048 # bimodal predictor config (<table size>)
```

```
-bpred:2lev      1 1024 8 0 # 2-level predictor config (<l1size> <l2size>
<hist_size> <xor>)
-bpred:comb      1024 # combining predictor config (<meta_table_size>)
-bpred:ras                 8 # return address stack size (0 for no return
stack)
-bpred:btb                  512 4 # BTB config (<num_sets>
<associativity>)
# -bpred:spec_update       <null> # speculative predictors update in
{ID|WB} (default non-spec)
-decode:width              4 # instruction decode B/W (insts/cycle)
-issue:width               4 # instruction issue B/W (insts/cycle)
-issue:inorder         false # run pipeline with in-order issue
-issue:wrongpath        true # issue instructions down wrong execution
paths
-commit:width              4 # instruction commit B/W (insts/cycle)
-ruu:size                 16 # register update unit (RUU) size
-lsq:size                  8 # load/store queue (LSQ) size
-cache:dl1       dl1:512:64:1:l # l1 data cache config, i.e.,
{<config>|none}
-cache:dl1lat              1 # l1 data cache hit latency (in cycles)
-cache:dl2       ul2:1024:64:4:l # l2 data cache config, i.e.,
{<config>|none}
-cache:dl2lat              6 # l2 data cache hit latency (in cycles)
-cache:il1       il1:512:32:1:l # l1 inst cache config, i.e.,
{<config>|dl1|dl2|none}
-cache:il1lat              1 # l1 instruction cache hit latency (in
cycles)
-cache:il2               dl2 # l2 instruction cache config, i.e.,
{<config>|dl2|none}
-cache:il2lat              6 # l2 instruction cache hit latency (in
cycles)
-cache:flush           false # flush caches on system calls
-cache:icompress       false # convert 64-bit inst addresses to 32-bit
inst equivalents
-mem:lat         18 2 # memory access latency (<first_chunk> <inter_chunk>)
-mem:width                 8 # memory access bus width (in bytes)
-tlb:itlb        itlb:16:4096:4:l # instruction TLB config, i.e.,
{<config>|none}
-tlb:dtlb        dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
-tlb:lat                  30 # inst/data TLB miss latency (in cycles)
-res:ialu                  4 # total number of integer ALU's available
```

```
-res:imult                 1 # total number of integer multiplier/dividers
available
-res:memport               2 # total number of memory system ports
available (to CPU)
-res:fpalu                 4 # total number of floating point ALU's
available
-res:fpmult                1 # total number of floating point
multiplier/dividers available
# -pcstat             <null> # profile stat(s) against text addr's (mult
uses ok)
-bugcompat             false # operate in backward-compatible bugs mode
(for testing only)
  Pipetrace range arguments are formatted as follows:
    {{@|#}<start>}:{{@|#|+}<end>}
  Both ends of the range are optional, if neither are specified, the entire
execution is traced.  Ranges that start with a `@' designate an address
range to be traced, those that start with an `#' designate a cycle count
  range.  All other range values represent an instruction count range.  The
second argument, if specified with a `+', indicates a value relative   to
the first argument, e.g., 1000:+100 == 1000:1100.  Program symbols may
  be used in all contexts.
    Examples:   -ptrace FOO.trc #0:#1000
                -ptrace BAR.trc @2000:
                -ptrace BLAH.trc :1500
                -ptrace UXXE.trc :
                -ptrace FOOBAR.trc @main:+278
  Branch predictor configuration examples for 2-level predictor:
    Configurations:   N, M, W, X
      N   # entries in first level (# of shift register(s))
      W   width of shift register(s)
      M   # entries in 2nd level (# of counters, or other FSM)
      X   (yes-1/no-0) xor history and address for 2nd level index
    Sample predictors:
      GAg     : 1, W, 2^W, 0
      GAp     : 1, W, M (M > 2^W), 0
      PAg     : N, W, 2^W, 0
      PAp     : N, W, M (M == 2^(N+W)), 0
      gshare  : 1, W, 2^W, 1
  Predictor `comb' combines a bimodal and a 2-level predictor.
  The cache config parameter <config> has the following format:
    <name>:<nsets>:<bsize>:<assoc>:<repl>
```

```
    <name>   - name of the cache being defined
    <nsets>  - number of sets in the cache
    <bsize>  - block size of the cache
    <assoc>  - associativity of the cache
    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random
    Examples:   -cache:dl1 dl1:4096:32:1:l
                -dtlb dtlb:128:4096:32:r
  Cache levels can be unified by pointing a level of the instruction cache
hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache
configuration arguments.  Most sensible combinations are supported, e.g.,
    A unified l2 cache (il2 is pointed at dl2):
      -cache:il1 il1:128:64:1:l -cache:il2 dl2
      -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
    Or, a fully unified cache hierarchy (il1 pointed at dl1):
      -cache:il1 dl1
      -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
sim: ** starting performance simulation **
sim: ** simulation statistics **
sim_num_insn               34329976 # total number of instructions
committed
sim_num_refs               10060312 # total number of loads and stores
committed
sim_num_loads               6406097 # total number of loads committed
sim_num_stores         3654215.0000 # total number of stores committed
sim_num_branches            4116531 # total number of branches committed
sim_elapsed_time                 49 # total simulation time in seconds
sim_inst_rate           700611.7551 # simulation speed (in insts/sec)
sim_total_insn             36893918 # total number of instructions executed
sim_total_refs             11093134 # total number of loads and stores
executed
sim_total_loads             7203347 # total number of loads executed
sim_total_stores       3889787.0000 # total number of stores executed
sim_total_branches          4422588 # total number of branches executed
sim_cycle                  18047337 # total simulation time in cycles
sim_IPC                      1.9022 # instructions per cycle
sim_CPI                      0.5257 # cycles per instruction
sim_exec_BW                  2.0443 # total instructions (mis-spec +
committed) per cycle
sim_IPB                      8.3395 # instruction per branch
IFQ_count                  56092089 # cumulative IFQ occupancy
IFQ_fcount                 12482383 # cumulative IFQ full count
```

```
ifq_occupancy                 3.1081 # avg IFQ occupancy (insn's)
ifq_rate                      2.0443 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                   1.5204 # avg IFQ occupant latency (cycle's)
ifq_full                      0.6916 # fraction of time (cycle's) IFQ was
full
RUU_count                  219890730 # cumulative RUU occupancy
RUU_fcount                   9392368 # cumulative RUU full count
ruu_occupancy                12.1841 # avg RUU occupancy (insn's)
ruu_rate                      2.0443 # avg RUU dispatch rate (insn/cycle)
ruu_latency                   5.9601 # avg RUU occupant latency (cycle's)
ruu_full                      0.5204 # fraction of time (cycle's) RUU was
full
LSQ_count                   65552200 # cumulative LSQ occupancy
LSQ_fcount                   1524927 # cumulative LSQ full count
lsq_occupancy                 3.6322 # avg LSQ occupancy (insn's)
lsq_rate                      2.0443 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                   1.7768 # avg LSQ occupant latency (cycle's)
lsq_full                      0.0845 # fraction of time (cycle's) LSQ was
full
sim_slip                   321532941 # total number of slip cycles
avg_sim_slip                  9.3660 # the average slip between issue and
retirement
bpred_bimod.lookups          4609094 # total number of bpred lookups
bpred_bimod.updates          4116531 # total number of updates
bpred_bimod.addr_hits        3801964 # total number of address-predicted
hits
bpred_bimod.dir_hits         3823915 # total number of direction-predicted
hits (includes addr-hits)
bpred_bimod.misses            292616 # total number of misses
bpred_bimod.jr_hits           423129 # total number of address-predicted
hits for JR's
bpred_bimod.jr_seen           444551 # total number of JR's seen
bpred_bimod.jr_non_ras_hits.PP            0 # total number of address-
predicted hits for non-RAS JR's
bpred_bimod.jr_non_ras_seen.PP            1 # total number of non-RAS JR's
seen
bpred_bimod.bpred_addr_rate    0.9236 # branch address-prediction rate
(i.e., addr-hits/updates)
bpred_bimod.bpred_dir_rate    0.9289 # branch direction-prediction rate
(i.e., all-hits/updates)
```

```
bpred_bimod.bpred_jr_rate    0.9518 # JR address-prediction rate (i.e., JR
addr-hits/JRs seen)
bpred_bimod.bpred_jr_non_ras_rate.PP    0.0000 # non-RAS JR addr-pred rate
(ie, non-RAS JR hits/JRs seen)
bpred_bimod.retstack_pushes       519852 # total number of address pushed
onto ret-addr stack
bpred_bimod.retstack_pops       493755 # total number of address popped off
of ret-addr stack
bpred_bimod.used_ras.PP       444550 # total number of RAS predictions used
bpred_bimod.ras_hits.PP       423129 # total number of RAS hits
bpred_bimod.ras_rate.PP    0.9518 # RAS prediction rate (i.e., RAS
hits/used RAS)
il1.accesses             38272764 # total number of accesses
il1.hits                 37929067 # total number of hits
il1.misses                 343697 # total number of misses
il1.replacements           343196 # total number of replacements
il1.writebacks                  0 # total number of writebacks
il1.invalidations               0 # total number of invalidations
il1.miss_rate              0.0090 # miss rate (i.e., misses/ref)
il1.repl_rate              0.0090 # replacement rate (i.e., repls/ref)
il1.wb_rate                0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
dl1.accesses             10268704 # total number of accesses
dl1.hits                 10197283 # total number of hits
dl1.misses                  71421 # total number of misses
dl1.replacements            70909 # total number of replacements
dl1.writebacks              31134 # total number of writebacks
dl1.invalidations               0 # total number of invalidations
dl1.miss_rate              0.0070 # miss rate (i.e., misses/ref)
dl1.repl_rate              0.0069 # replacement rate (i.e., repls/ref)
dl1.wb_rate                0.0030 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
ul2.accesses               446252 # total number of accesses
ul2.hits                   428999 # total number of hits
ul2.misses                  17253 # total number of misses
ul2.replacements            13157 # total number of replacements
ul2.writebacks               7034 # total number of writebacks
ul2.invalidations               0 # total number of invalidations
ul2.miss_rate              0.0387 # miss rate (i.e., misses/ref)
ul2.repl_rate              0.0295 # replacement rate (i.e., repls/ref)
ul2.wb_rate                0.0158 # writeback rate (i.e., wrbks/ref)
```

```
ul2.inv_rate                  0.0000 # invalidation rate (i.e., invs/ref)
itlb.accesses               38272764 # total number of accesses
itlb.hits                   38272735 # total number of hits
itlb.misses                       29 # total number of misses
itlb.replacements                  0 # total number of replacements
itlb.writebacks                    0 # total number of writebacks
itlb.invalidations                 0 # total number of invalidations
itlb.miss_rate                0.0000 # miss rate (i.e., misses/ref)
itlb.repl_rate                0.0000 # replacement rate (i.e., repls/ref)
itlb.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
itlb.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
dtlb.accesses               10276406 # total number of accesses
dtlb.hits                   10276293 # total number of hits
dtlb.misses                      113 # total number of misses
dtlb.replacements                  0 # total number of replacements
dtlb.writebacks                    0 # total number of writebacks
dtlb.invalidations                 0 # total number of invalidations
dtlb.miss_rate                0.0000 # miss rate (i.e., misses/ref)
dtlb.repl_rate                0.0000 # replacement rate (i.e., repls/ref)
dtlb.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
dtlb.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
sim_invalid_addrs                  0 # total non-speculative bogus addresses
seen (debug var)
ld_text_base              0x00400000 # program text (code) segment base
ld_text_size                  159696 # program text (code) size in bytes
ld_data_base              0x10000000 # program initialized data segment base
ld_data_size                   34320 # program init'ed `.data' and uninit'ed
`.bss' size in bytes
ld_stack_base             0x7fffc000 # program stack segment base (highest
address in stack)
ld_stack_size                  16384 # program initial stack size
ld_prog_entry             0x00400140 # program entry point (initial PC)
ld_environ_base           0x7fff8000 # program environment base address
address
ld_target_big_endian               0 # target executable endian-ness, non-
zero if big endian
mem.page_count                   152 # total number of pages allocated
mem.page_mem                    608k # total size of memory pages allocated
mem.ptab_misses                  327 # total first level page table misses
mem.ptab_accesses          203461929 # total page table accesses
mem.ptab_miss_rate            0.0000 # first level page table miss rate
```

ANNEXE 2 : EXEMPLE DE SORTIE DE CACTI

```
---------- CACTI version 3.2 ----------

Cache Parameters:
  Number of Subbanks: 1
  Total Cache Size: 65536
  Size in bytes of Subbank: 65536
  Number of sets: 2048
  Associativity: direct mapped
  Block Size (bytes): 32
  Read/Write Ports: 1
  Read Ports: 0
  Write Ports: 0
  Technology Size: 0.13um
  Vdd: 1.3V

Access Time (ns): 1.19621
Cycle Time (wave pipelined) (ns):  0.593408
Total Power all Banks (nJ): 0.491459
Total Power Without Routing (nJ): 0.491459
Total Routing Power (nJ): 0
Maximum Bank Power (nJ):  0.491459

Best Ndwl (L1): 1
Best Ndbl (L1): 4
Best Nspd (L1): 1
Best Ntwl (L1): 1
Best Ntbl (L1): 2
Best Ntspd (L1): 8
Nor inputs (data): 3
Nor inputs (tag): 3

Area Components:

Aspect Ratio Total height/width: 2.243660

Data array (cm^2): 0.031702
Data predecode (cm^2): 0.000043
Data colmux predecode (cm^2): 0.000010
Data colmux post decode (cm^2): 0.000002
Data write signal (cm^2): 0.000004

Tag array (cm^2): 0.002540
Tag predecode (cm^2): 0.000043
Tag colmux predecode (cm^2): 0.000020
Tag colmux post decode (cm^2): 0.000006
Tag output driver decode (cm^2): 0.000018
Tag output driver enable signals (cm^2): 0.000004

Percentage of data ramcells alone of total area: 83.308987 %
Percentage of tag ramcells alone of total area: 5.985560 %
Percentage of total control/routing alone of total area: 10.705453 %

Subbank Efficiency : 89.294547
Total Efficiency : 74.630489

Total area One Subbank (cm^2): 0.034393
Total area subbanked (cm^2): 0.041151

Time Components:
```

```
data side (with Output driver) (ns): 1.19621
tag side (with Output driver) (ns): 0.857158
address routing delay (ns): 0
address routing power (nJ): 0
decode_data (ns): 0.593408
            (nJ): 0.0904672
wordline and bitline data (ns): 0.356324
           wordline power (nJ): 0.000673851
            bitline power (nJ): 0.186437
sense_amp_data (ns): 0.134875
               (nJ): 0.104279
decode_tag (ns): 0.277709
           (nJ): 0.0154772
wordline and bitline tag (ns): 0.16244
          wordline power (nJ): 0.000410635
           bitline power (nJ): 0.029423
sense_amp_tag (ns): 0.082875
              (nJ): 0.0490081
compare (ns): 0.248605
        (nJ): 0.000438372
valid signal driver (ns): 0.085528
                    (nJ): 0.000367352
data output driver (ns): 0.111604
                   (nJ): 0.0144773
total_out_driver (ns): 0
                 (nJ): 0
total data path (without output driver) (ns): 1.08461
total tag path is dm (ns): 0.77163
```