Optimisation de l'énergie dans une architecture mémoire multi-bancs pour des applications multi-tâches temps réel

Hanene Ben Fradj

UNIVERSITE DE NICE-SOPHIA ANTIPOLIS

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

T H E S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention : Automatique, traitement du signal et des images

présentée et soutenue par

Hanene BEN FRADJ

Optimisation de l'énergie dans une architecture mémoire multi-bancs pour des applications multi-tâches temps réel

Thèse dirigée par : *Cécile BELLEUDY et Michel AUGUIN*

soutenue le : *13 décembre 2006*

Jury :

M. Olivier SENTIEYS	Professeur IRISA, Rennes	Rapporteur
M. Christian PIGUET	Professeur CSEM Neuchâtel	Rapporteur
M. Marc RENAUDIN	Professeur INPG Grenoble	Examinateur
M. Edmond CAMBIAGGIO	Professeur UNSA Nice	Examinateur
Mme. Cécile BELLEUDY	Maître de Conférences, UNSA	Encadreur
M. Michel AUGUIN	Directeur de recherche CNRS, I3S	Directeur de thèse

Table des matières

Introduction

«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 à quoi ils sont destinés. Omniprésents, ils le sont déjà et le seront de plus en plus»
Patrice Kadionik de l'ENSEIRB

L'originalité de cette citation insiste sur le déploiement considérable des systèmes embarqués dans notre vie quotidienne. Il est intéressant de constater que nous nous servons de plus d'une dizaine de systèmes embarqués chaque jour souvent sans s'en rendre compte. On les trouve dans les applications domestiques comme la cafetière, la télévision, le lecteur DVD,… mais aussi dans les applications personnelles comme le téléphone mobile, les PDA, le lecteur MP3, la console de jeux. On les trouve également dans les domaines industriels comme le transport, le médical, les télécommunications, le militaire… Dans les nouvelles générations de voitures, ils contrôlent une multitude de fonctions pour améliorer les conditions de conduite telles que l'antiblocage des roues (ABS), l'authentification du conducteur, la vérification du respect de la distance de sécurité, l'aide au stationnement…

Cette tendance s'est accentuée suite à l'évolution technologique visant la miniaturisation des transistors qui a permis d'augmenter considérablement la capacité d'intégration dans les systèmes embarqués. En effet, une capacité d'intégration de 7,2 10^9 transistors par cm^2 est envisagée pour 2020 avec une technologie de 14nm selon ITRS-SIA 2005 (ITRS, 2005). Les applications embarquées ont elles aussi beaucoup évolué et sont devenues de plus en plus complexes en offrant un nombre important et varié de fonctionnalités. Par exemple, la norme de communication mobile UMTS intègre des standards actuels de communication (GSM, GPRS, EDGE…) et des possibilités de télécommunication avec des débits plus élevés permettant l'association de capacités multimédia (Visiophonie, MMS Vidéo, Vidéo à la demande, Télévision numérique TNT). Toutes ces fonctionnalités vont être intégrées dans nos téléphones portables de taille toujours réduite et alimentés par batterie.

La conception de tels systèmes intégrés, embarqués (*System on Chip* : SoC) et autonomes est alors un « challenge » que doivent relever les concepteurs afin de répondre à une série de contraintes : robustesse, fiabilité, sûreté, tolérances aux fautes, *packaging*, surface, consommation énergétique, coût et temps de développement. La criticité associée à chaque contrainte est variable en fonction des objectifs fonctionnels du système embarqué et des évolutions technologiques et applicatives. A cause des limites

technologiques des batteries (amélioration seulement de quelques pourcents par an par rapport à un doublement du nombre de transistors dans une puce tous les 18 mois selon la loi de Moore), la criticité associée à la consommation énergétique s'est particulièrement élevée ces dix dernières années. La réduction de la consommation énergétique dans les systèmes embarqués est ainsi devenue une nécessité pour autoriser le déploiement de nouveaux produits. De nombreux travaux anticipent, par exemple, des systèmes basés sur une informatique pervasive ou des réseaux ad-hoc qui induisent nécessairement de supporter, dans un équipement mobile, des fonctionnalités encore plus nombreuses et complexes, sources de consommation d'énergie.

Au début, ce sont les unités de traitement (processeur, DSP, ASIC, ASIP, …) qui ont été identifiées comme source majeure de consommation dans un SoC et ont bénéficié de la majorité des travaux de recherche développés en vue d'une réduction de leur consommation dynamique d'énergie. Des techniques ont été développées à différents niveaux du flot de conception d'un SoC. On cite comme exemples, la technique d'ajustement conjoint de la tension et de la fréquence (*Dynamic Voltage Scaling* : DVS) et la technique de gestion des modes repos du processeur (*Dynamic Power Management* : DPM) utilisées au niveau système, la modification de code au niveau algorithmique, le *clock gating* au niveau circuit, le redimensionnement de la taille des transistors au niveau physique.

Les deux évolutions, architecturale et applicative, mentionnées ci-dessus sont à l'origine de nouveaux types et sources de consommation énergétique dans les SoC. Une dissipation statique des transistors, consommée hors commutations logiques qui était auparavant négligeable, devient de plus en plus significative avec l'évolution technologique. En effet, la diminution de la tension d'alimentation V_{dd} avec la taille du transistor entraîne aussi une diminution de la tension seuil V_{th} afin de garder une vitesse de commutation raisonnable. La diminution de V_{th} est à l'origine d'une augmentation exponentielle du courant statique. Pour une technologie de 65nm des parts égales de consommation dynamique et statique devraient être atteintes.

Une autre source de consommation, qui est souvent du même ordre de grandeur que celle due aux unités de traitement, est consommée par les unités de mémorisation dans les SoC. En effet, la surface dédiée pour mémoriser les données et les codes des applications actuelles de télécommunication ou de multimédia, est en constante augmentation dans les systèmes embarqués. Selon les prévisions de l'ITRS 2005 (ITRS, 2005), la surface totale d'un système embarqué est largement dominée par la surface dédiée aux unités de mémorisation. La société Samsung annonce qu'un téléphone mobile qui n'avait pas de mémoire DRAM en 2002, aura une mémoire dépassant les 20 MB en 2006. La figure 1 souligne cette évolution de la taille mémoire DRAM dans les téléphones 3G suite à l'intégration de composants nouveaux tels que caméra numérique, lecteur MP3 et navigateur internet.

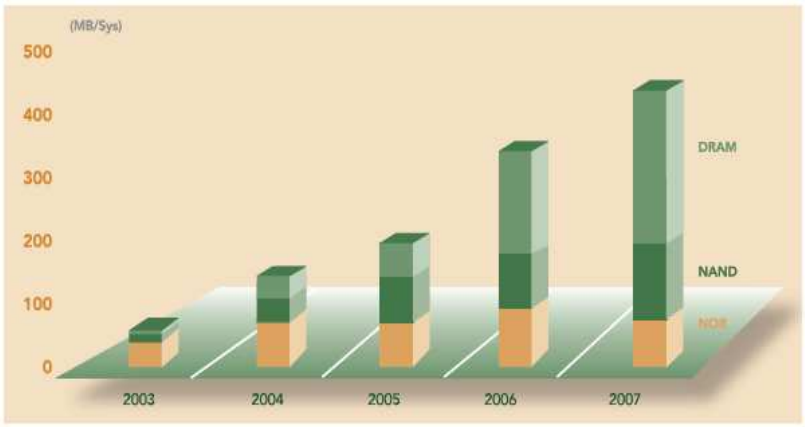


Figure 1. Evolution de la taille mémoire DRAM dans les téléphones 3G (www.samsung.com)

Cette tendance fait que les mémoires deviennent la source dominante de la consommation énergétique d'un système embarqué. En effet, les prévisions de l'ITRS 2005 montrent que la consommation d'un SoC est dominée par la consommation dynamique et statique des mémoires (figure 2). Ainsi, en 2020, la consommation mémoire représenterait environ 75,4% de la consommation totale d'un SoC (ITRS, 2005).

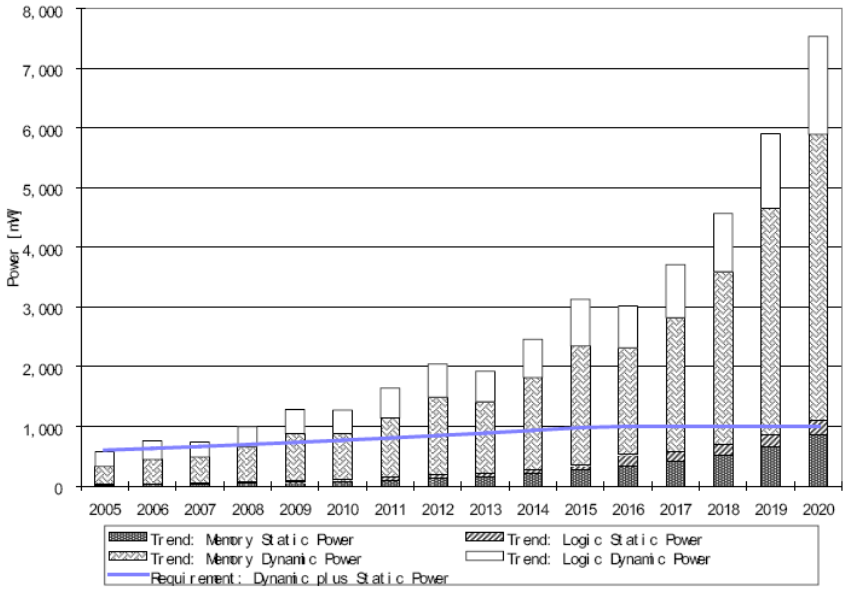


Figure 2. Evolution des consommations de puissance dans un SoC

Une étude de consommation présentée dans (Viredaz *et al.*, 2001), d'un PDA ITSY illustre que les mémoires DRAM sont la source de consommation principale (Tableau 1).

Tableau 1. Puissance moyenne consommée dans un PDA ITSY de COMPAQ, V_{dd} = 3,75V ; P_{mon}[1], $P_{main\ sup}$[2], P_{spkr}[3], P_{codec}[4], P_{DRAM}[5], P_{LCD}[6], P_{main}[7], $P_{core\ sup}$[8], P_{core}[9]

Benchmark	Freq. [MHz]	V_{core} [V]	Power [mW]	P_{mon}	$P_{main\ sup}$	P_{spkr}	P_{codec}	P_{DRAM}	P_{LCD}	P_{main}	$P_{core\ sup}$	P_{core}
Deep sleep	—	—	4.40	8 %	21 %	—	—	—	—	66 %	4.7 %	—
Sleep	—	—	7.18	6 %	14 %	—	—	34 %	—	43 %	2.9 %	—
Sleep, LCD	—	—	25.8	2 %	7 %	—	—	9.5 %	12 %	69 %	0.8 %	—
Idle	59	1.25	55.5	1 %	5 %	—	—	36 %	6 %	21.7 %	7.2 %	23.5 %
Idle	88	1.25	56.7	1 %	4 %	—	—	30 %	6 %	21.4 %	9 %	29.4 %
Idle	59	1.5	68.8	1 %	4 %	—	—	29 %	5 %	24.5 %	7 %	29.4 %
Idle	88	1.5	72.7	1 %	3 %	—	—	23.2 %	5 %	23.4 %	9 %	36 %
Idle	133	1.5	81.9	1 %	3 %	—	—	18.8 %	4.2 %	21.0 %	10 %	42 %
Idle	206	1.5	99.3	1 %	2 %	—	—	15.3 %	3.4 %	17.6 %	11 %	49 %
WAV	59	1.25	260	1 %	12 %	38.3 %	2.2 %	8.7 %	1.3 %	26.9 %	2.1 %	7.3 %
WAV	88	1.25	262	1 %	12 %	38.2 %	2.2 %	7.3 %	1.3 %	26.8 %	2.4 %	8.6 %
WAV	59	1.5	277	1 %	12 %	35.9 %	2.1 %	8.2 %	1.2 %	27.0 %	2.6 %	10.6 %
WAV	88	1.5	281	1 %	11 %	35.6 %	2.1 %	6.8 %	1.2 %	26.8 %	2.9 %	12.4 %
WAV	133	1.5	290	1 %	11 %	34.5 %	2.0 %	6.0 %	1.2 %	26.0 %	3.4 %	15.0 %
WAV	206	1.5	308	1 %	11 %	32.6 %	1.9 %	5.5 %	1.1 %	24.6 %	4.2 %	18.7 %
DECtalk	74	1.25	350	1 %	5 %	6.0 %	1.5 %	37 %	0.9 %	25.8 %	4.7 %	18.9 %
DECtalk	88	1.25	381	1 %	5 %	6.0 %	1.5 %	36 %	0.9 %	26.1 %	4.6 %	19.4 %
DECtalk	74	1.5	393	1 %	4 %	5.4 %	1.4 %	32 %	0.9 %	24.1 %	5.1 %	25.7 %
DECtalk	88	1.5	429	1 %	4 %	5.4 %	1.4 %	32 %	0.8 %	24.2 %	5.1 %	26.4 %
DECtalk	133	1.5	414	1 %	4 %	5.6 %	1.4 %	27 %	0.8 %	24.8 %	6 %	29 %
DECtalk	206	1.5	394	1 %	4 %	5.9 %	1.5 %	23 %	0.8 %	24.7 %	7 %	33 %
MPEG-1	206	1.5	808	1 %	6 %	12.1 %	0.7 %	28.4 %	0.5 %	16.5 %	5.8 %	28.8 %

Cet exemple ainsi que la figure 2 montrent que les efforts entrepris pour réduire ou maîtriser la consommation des processeurs n'ont sans doute pas été les mêmes vis-à-vis des mémoires. Des efforts supplémentaires au niveau des unités de mémorisation sont alors indispensables et nécessaires pour réduire leur consommation et ramener la dissipation énergétique du SoC au niveau autorisé comme indiqué par la courbe bleue de la figure 2.

De plus, les évolutions technologiques et applicatives remettent en cause l'efficacité des techniques initialement proposées pour diminuer la consommation dynamique de la seule unité de traitement. En effet, ces techniques classiques utilisées pour des systèmes dominés par la mémoire et pour des technologies fines, ne montrent plus les mêmes gains car il n'est plus possible de négliger la dissipation statique et celle des mémoires. Dans certains cas, elles sont même responsables d'une augmentation de la consommation au lieu de la réduction attendue.

---

[1] P_{mon} : puissance dissipée par le moniteur de batterie

[2] $P_{main\ sup}$: puissance dissipée par les alimentations numériques et analogiques.

[3] P_{spkr} : puissance dissipée par le locuteur.

[4] P_{codec} : puissance dissipée par la partie analogique du codec. Ce domaine inclut aussi le microphone et l'écran tactile.

[5] P_{DRAM}: puissance dissipée par la DRAM

[6] P_{LCD}: puissance dissipée par l'ecran LCD.

[7] P_{main}: puissance dissipée par les I/Os du processeur StrongARM SA-1100 et le reste de la logique numérique : mémoire flash, mémoire des cartes secondaires, interfaces séries, la partie numérique du codec, accéléromètre 2-axes, et les bouttons.

[8] $P_{core\ sup}$: puissance dissipée par l'alimentation du cœur du processeur.

[9] P_{core}: puissance dissipée par le cœur du processeur StrongARM SA-1100.

Dans nos travaux, nous nous intéressons à la technique de DVS qui contribue par sa grande efficacité à réduire la consommation processeur. Nous nous sommes posés la question de son efficacité réelle en considérant l'augmentation de la surface occupée par les mémoires dans les systèmes embarqués. Dans notre étude, nous montrons que cette technique contribue en réalité à accroître la consommation de la mémoire principale DRAM ce qui peut même conduire à inhiber tout l'intérêt d'utiliser la gestion de DVS sur le processeur. Pour obtenir globalement une réduction efficace d'énergie, nous proposons une architecture mémoire multi-bancs, avec des modes faibles consommation, dont il s'agit de déterminer une configuration en adéquation avec les traitements et les données des applications.

Dans ce sens, l'état de l'art présenté dans **le premier chapitre** est divisé en deux parties. La première partie présente les nouvelles techniques de DVS récemment développées dans la littérature pour faire face aux problèmes posés par les nouvelles avancées technologiques et applicatives. La deuxième partie présente l'historique de l'utilisation du partitionnement mémoire en plusieurs bancs pour, dans un premier temps, augmenter les performances et ensuite pour réduire la consommation aussi bien dans les mémoires statiques internes SRAM que dans les mémoires principales dynamiques externes DRAM.

La configuration de la structure mémoire multi-bancs (nombre et taille des bancs) et l'allocation des différentes tâches de l'application à ces bancs est abordée au niveau système afin de réduire la consommation mémoire mais aussi la consommation totale (processeur et mémoire). Des modèles permettant d'évaluer la consommation énergétique d'une mémoire multi-bancs pour des applications multi-tâches sont développés dans le **deuxième chapitre.**

Deux approches de résolution du problème de configuration et d'allocation des tâches aux bancs mémoire sont présentées dans le **troisième chapitre**. Une première exploration basée sur une méthode exhaustive est développée afin d'obtenir la configuration optimale. Dans un second temps, une heuristique est proposée afin de pallier le problème du temps d'exécution exponentiel de la première méthode.

Des expérimentations de l'approche proposée pour réduire la consommation mémoire et du couple processeur, mémoire sont effectuées dans le **quatrième chapitre** sur des applications de traitement de signal et sur une application de type multimédia (GSM, décodeur MPEG-2). Nous discutons dans la conclusion des résultats obtenus avec l'approche développée et différentes perspectives ouvertes par ce travail sont aussi proposées.

Chapitre 1. Etat de l'art

Dans ce chapitre portant sur l'état de l'art, nous mettons l'accent dans une première partie, sur les travaux relatifs à l'optimisation de la consommation processeur par ajustement de la tension d'alimentation et de la fréquence (DVS) puis dans une deuxième partie sur la consommation mémoire par utilisation d'une structure multi-bancs. En effet, nous illustrons que l'optimisation processeur par DVS, reconnue comme l'une des plus efficaces, peut induire des augmentations significatives de la consommation mémoire. Nous sommes ainsi amenés à exposer des solutions relatives au système mémoire pour obtenir globalement des approches plus efficaces.

Partie 1 : Les techniques de DVS

Introduction

Ces dernières années, la consommation énergétique dans les systèmes embarqués ou enfouis, s'est imposée comme un critère critique à prendre en compte lors de la conception de systèmes sur puces, à part égale avec le respect des contraintes de temps d'exécution et la surface de silicium. Cette prise de conscience a amené les concepteurs à effectuer des premières expérimentations pour identifier les sources de consommation ainsi que la nature de l'énergie dissipée afin de pouvoir la minimiser ensuite. Deux constations majeures ont été relevées dans un premier temps. La première, souligne que la consommation dans un SoC est essentiellement dynamique, i.e. due à l'activité du circuit, et que la consommation statique due à des courants de fuite est négligeable. La deuxième a identifié le processeur comme la source majeure de consommation parmi les composants présents dans le SoC. Par conséquent, les premiers travaux de recherche dans ce domaine ont ciblé l'optimisation de la seule consommation dynamique et uniquement des processeurs embarqués. De nombreuses techniques employées à plusieurs niveaux ont été alors développées. On cite par exemple le *clock gating* (Benini *et al*., 1998) ou le *Dynamic Power Management* (DPM) (Weng *et al*., 2003). Au niveau système, la technique d'ajustement conjoint en tension et en fréquence (*Dynamic Voltage Scaling* : DVS) s'est particulièrement distinguée par sa grande efficacité à réduire la consommation processeur. Elle permet d'exécuter différentes tâches d'une application à des couples tension/fréquence différents en fonction de la charge de travail du processeur. Notons que diminuer la tension d'alimentation V_{dd} impose une fréquence *f* de fonctionnement plus faible. Comme la puissance dynamique consommée est proportionnelle à V_{dd}^2 et à *f*, des gains significatifs sont très vite obtenus. Rapidement, cette technique a été adoptée par les industriels et de nombreux processeurs exploitant cette opportunité ont été conçus. Le tableau 1.1 présente les intervalles de variation de tensions et de fréquences des processeurs les plus utilisés dans les systèmes embarqués.

Tableau 1.1. Variation en tension et fréquence de quelques processeurs **(**Zhai *et al*., 2004)

Processeur	***Tension (V)***	***Fréquence (MHz)***
IBM PowerPC 405LP	1,0 - 1,8	153 - 333
Transmeta Crusoe TM5800	0,8 - 1.3	300 - 1000
Intel XScale 80200	0,95 - 1,55	333 - 733
StrongArm 1100	0,9 - 1,5	59 - 206
lpARM	1,1 - 3,3	10 - 100
PXA255	0,85 - 1,3	100 - 400

Il faut signaler aussi que les changements en ligne du couple (tension, fréquence) sont à l'origine de pénalités temporelles et énergétiques. Comme exemples nous citons, 20µs pour le processeur Xscale (Jejurikar *et al*., Août 2004), 500µs pour le processeur PXA255 d'Intel (Intel, 2003), une pénalité inférieure à 150µs est mesurée pour le StrongARM SA-1100 (Pouwelse *et al*., 2001), 5ms de pénalité pour le Transmeta Crusoe enfin le lpARM met 25µs pour passer de 10 à 100 Mhz (Pering *et al*., 2000). Par contre, le processeur PowerPC 405LP supporte un changement de tension et de fréquence sans interrompre l'exécution du programme (Jejurikar *et al*., Août 2004).

1. Techniques de DVS classiques

La disponibilité de processeurs muni de DVS a conduit à des expérimentations afin d'évaluer l'efficacité des techniques proposées.

1.1. Nature de la technique de DVS

Des hypothèses de plus en plus réalistes ont été considérées et des gains intéressants ont été montrés. Ces techniques peuvent être :

- ciblées pour des architectures monoprocesseur ou multiprocesseur de type homogène ou distribué.

- appliquées à des systèmes temps réel à contraintes souples ou strictes : les techniques de DVS temps réel souple sont généralement basées sur la prédiction des temps d'exécution à partir des exécutions précédentes pour adapter la fréquence et la tension de fonctionnement du processeur. L'efficacité de ces techniques en terme de réduction de la consommation est liée au modèle de prédiction utilisé. Le non respect de quelques échéances est dans ce cas sans conséquences graves sur le fonctionnement du système. Pour les techniques de DVS à temps réel strict, le respect de toutes les échéances des tâches est obligatoire pour assurer le bon fonctionnement du système. Des techniques de ce type sont présentées ci-dessous.

- Intra-DVS ou Inter-DVS : les techniques Intra-DVS utilisent le temps prévu mais non consommé (ou *slack*) de la tâche courante pour réduire la vitesse d'exécution de cette même tâche alors que la technique Inter-DVS utilise le *slack* de la tâche courante pour réduire la vitesse d'exécution des tâches suivantes.

- en ligne ou hors ligne : les prises de décisions d'ajustement de la tension et de la fréquence de fonctionnement du processeur sont faites soit hors ligne, lors de la phase de conception i.e. avant exécution, soit en ligne c'est-à-dire en cours d'exécution.

- dynamique ou statique : les techniques statiques de DVS déterminent une vitesse fixe de fonctionnement qui reste inchangée tout au long de l'exécution alors que les techniques dynamiques adaptent la vitesse du processeur en fonction de plusieurs paramètres comme les temps d'exécution des tâches, le taux d'utilisation du processeur.

- tâches dépendantes ou indépendantes : certaines techniques gèrent les dépendances entres les tâches (dépendances de précédence, dépendances de données, …) et d'autres ne les prennent pas en compte.

1.2. Les différentes approches de DVS

Plusieurs approches d'ajustement conjoint en tension et en fréquence (DVS) visant tout d'abord des systèmes monoprocesseur puis multiprocesseur ont été développées ces dernières années. Dans ce paragraphe nous présentons quelques techniques remarquables.

1.2.1. Vitesse statique minimale

Cette approche consiste à trouver la fréquence la plus faible qui permet de respecter les échéances pour un ensemble de tâches donné. Cette fréquence est établie statiquement en utilisant le taux d'utilisation du processeur dans le pire cas :

$$U = \sum_{i=1}^{N} \frac{wcet_i}{P_i}$$ avec N est le nombre de tâches et P_i est la période de la tâche T_i

où toutes les tâches s'exécutent suivant leur temps au pire cas prévu lors de la phase de conception (*Worst Case Execution Time* WCET).

Par exemple, pour un ordonnancement EDF (*Earliest Deadline First*), si le taux d'utilisation du processeur dans le pire cas est inférieur à 1 pour un ensemble de tâches s'exécutant à une fréquence maximale $f_{nominal}$, ces tâches peuvent être ordonnancées avec une fréquence plus faible $f' = U \times f_{nominal}$ tout en garantissant les échéances des tâches.

1.2.2. Approche par chemin

Dans cette approche la tension et la fréquence sont déterminées à partir d'un chemin d'exécution de référence (WCEP). Si le système dévie de ce chemin de référence, le couple (V, f) est ajusté. Si le nouveau chemin d'exécution est plus lent que le chemin de référence, la fréquence est augmentée afin de respecter les échéances. Contrairement, si le nouveau chemin est plus rapide que le WCEP, la fréquence est baissée pour gagner de l'énergie. L'analyse des différents chemins de contrôle d'une tâche est effectuée par *profiling* par exemple.

1.2.3. Approche stochastique

A l'inverse des approches de DVS qui débutent l'exécution d'une application à une fréquence maximale puis réduisent progressivement la vitesse processeur quand les contraintes de temps le permettent, cette approche emploie une faible vitesse au début de l'exécution et l'augmente progressivement quand c'est nécessaire. En commençant à faible vitesse, et si les temps d'exécutions réels AET (*Actual Execution Time*) des tâches sont inférieurs aux WCET, les fréquences élevées peuvent ne jamais être atteintes. Cette approche se base sur des modèles stochastiques des temps d'exécution des tâches pour déterminer les fréquences de fonctionnement du processeur.

1.2.4 Extension du temps d'exécution jusqu'au NTA (*Arrival time of the Next Task*)

Bien qu'un ensemble de tâches soit ordonnancé avec une vitesse statique minimale, les temps réel d'exécution des tâches (AET) sont toujours inférieurs ou égaux à leurs temps WCET prévus. Pour certaines applications, le rapport entre WCET et AET est supérieur à 10 (Ernst *et al*., 1997). Plusieurs temps d'inactivité sont alors présents dans l'ordonnancement. Un temps d'inactivité existe si le NTA de la tâche suivante se situe après le WCET de la tâche active. Dans ces conditions, la fréquence est ajustée de telle façon que la tâche active termine son exécution juste avant le prochain NTA.

1.2.5. Distribution de *slack* basée sur la priorité

Cette approche dérive des techniques d'ordonnancement à priorité comme le RM et le EDF. Quand une tâche termine son exécution plus tôt que son WCET, les tâches de priorités moins élevées peuvent utiliser l'intervalle de temps (WCET – AET) des tâches plus prioritaires afin de réduire leur vitesse d'exécution.

1.2.6. Mise à jour du taux d'utilisation du processeur

Le taux d'utilisation effectif du processeur est généralement inférieur à celui calculé dans le pire cas. L'approche de mise à jour du taux d'utilisation du processeur consiste à recalculer en un point de l'ordonnancement (généralement au début et la fin d'exécution d'une tâche) le taux d'utilisation en utilisant le temps d'exécution AET des tâches qui ont terminé leurs exécutions. Ainsi, une fois le taux d'utilisation mis à jour, la vitesse du processeur est ajustée. Le principal avantage de cette technique est la simplicité de son implémentation.

Les tableaux 1.2 et 1.3 résument les différentes techniques de DVS trouvées dans la littérature.

Tableau 1.2. Caractéristiques des approches de DVS monoprocesseur (Kim *et al*., 2003)

	Approche de DVS	Décision d'ajustement
IntraDVS	Approche par chemin (1)	Hors ligne
	Approche stochastique (2)	
InterDVS	Vitesse statique minimale (3)	
	Extension du temps d'exécution jusqu'au NTA (4)	En ligne
	Distribution de slack basée sur la priorité (5)	
	Mise à jour du taux d'utilisation processeur (6)	

Tableau 1.3. Classification des algorithmes de DVS suivant les approches présentées

Catégorie	Ordonnancement	Technique de DVS	Approches de DVS employées
InterDVS	EDF	lppsEDF (Shin *et al*., 2000)	(3)+(4)
		ccEDF (Pillai *et al*., 2001)	(6)
		laEDF (Pillai *et al*., 2001)	(6)*
		DRA(Aydin *et al*., 2001)	(3+(4)+(5)
		AGR (Aydin *et al*., 2001)	(4)*+(5)
		lpSHE (Kim *et al*., 2002)	(3)+(4)+(5)*
	RM	lppsRM (Shin *et al*., 2000)	(3)+(4)
		ccRM (Pillai *et al*., 2001)	(3)+(4)*
		lpWDA (Kim *et al*., 2003)	(4)*
IntraDVS	Approche par chemin	(Shin *et al*., 2001)	(1)
	Approche stochastique	(Gruian, 2001)	(2)

* signifie une version modifiée

Une comparaison de performance entre ces différentes politiques de DVS et leur capacité à réduire l'énergie consommée par le système a été réalisée grâce à un simulateur de performance et d'énergie SIMDVS (Shin *et al*., 2002). Des gains très variables sont obtenus en fonction de différents paramètres comme la charge du processeur U au pire cas, le taux d'utilisation du WCET, le nombre de tâches, le nombre de couples (tension, fréquence) disponibles.

2. Techniques de DVS généralisées

L'évolution technologique rapide ainsi que l'apparition de nouvelles applications de traitement de signal ou de multimédia toujours plus volumineuses en nombre de données et en traitement ont remis en cause les techniques de DVS classiques qui se concentre sur la réduction de la dissipation dynamique de la seule unité de traitement (le processeur) parmi celles disponibles dans le SoC (mémoire, bus, périphériques,…). En effet les progrès dans la miniaturisation des composants ont permis d'augmenter le potentiel d'intégration sur une même puce et diminuer la tension d'alimentation. En contre partie, avec la diminution relative de la tension de seuil induite par la diminution de la tension d'alimentation afin de garder des performances élevées, la part de la consommation statique a considérablement augmenté. Les techniques de DVS, ciblant auparavant des technologies moins fines et par conséquent négligeant la consommation statique, ne montrent plus la même efficacité dans la réduction de la consommation pour les technologies actuelles et futures. Par ailleurs, la surface de silicium dédiée aux unités de mémorisation ne cesse de croître dans un SoC, afin de mémoriser une quantité de plus en plus importante de données requises par les applications actuelles. Cette tendance rend la consommation des systèmes embarqués partagée entre les unités de traitement (processeur, DSP, FPGA, ASIC,…) et les unités de mémorisation (cache, mémoire dédiée, mémoire principale DRAM…). Aussi, une évolution des techniques initialement focalisées sur le DVS a été constatée ces deux dernières années pour prendre en compte l'évolution technologique architecturale et applicative. Dans le paragraphe 2.1 nous détaillons les techniques de DVS modifiées pour prendre en compte l'augmentation de la consommation statique. Dans le paragraphe 2.2 nous détaillons les techniques de DVS

modifiées pour prendre en compte la consommation dynamique et statique de la mémoire lors de l'ajustement conjoint de la tension et de la fréquence.

2.1. Techniques de DVS tenant compte de la consommation statique

Comme l'illustre la figure 1.1, à partir de la technologie 70nm, les concepteurs doivent faire face, en plus de la consommation dynamique due à l'activité des circuits logiques, à celle statique dissipée hors activité. Pour ces technologies, les techniques de DVS classiques ne sont plus assez efficaces car elles ne considèrent que la composante dynamique de la consommation.

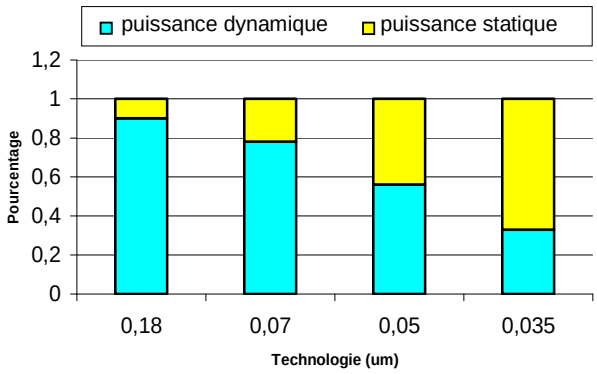


Figure 1.1. Evolution de la puissance statique avec la technologie (Yan *et al*., 2003)

2.1.1. Techniques de DVS combinées à la technique *Adaptive Body Biasing*

La technique *Adaptive Body Biasing* (ABB) permet d'ajuster dynamiquement la tension de seuil V_{th} en vue d'une réduction de la dissipation statique. Elle a l'avantage de réduire exponentiellement le courant statique alors que la technique de DVS le réduit linéairement, d'où l'importance du gain obtenu en ajustant la tension de seuil V_{th} des transistors.

Vu les proportions équivalentes de la contribution des deux consommations dynamique et statique dans les technologies du futur proche, des travaux récents ont ciblé la réduction des deux composantes afin d'obtenir de meilleurs gains et une durée de vie de la batterie plus longue. Si la puissance dynamique est réduite par la diminution de la tension de fonctionnement (V_{dd}) et de la fréquence, la composante statique peut être réduite par l'augmentation de la tension de seuil du transistor (V_{th}). Cette augmentation de V_{th} est obtenue par une élévation de la tension V_{bs} (*reverse body Bias Voltage*). Cette élévation a pour effet de ralentir la vitesse de fonctionnement du système, au même titre qu'une diminution de V_{dd}. La difficulté de l'utilisation simultanée du DVS et de l'ABB, concerne la détermination d'un compromis entre les tensions V_{dd} et V_{bs} afin de diminuer l'énergie totale du système pour une fréquence de fonctionnement donnée.

Martin et al. (Martin *et al*., 2002), ont développé des modèles analytiques d'énergie (dynamique et statique) et de performance en fonction de trois paramètres : la fréquence du processeur *f*, la tension V_{dd} et la tension V_{bs} pour minimiser la consommation totale du processeur. Les valeurs de ces trois paramètres sont déterminées en dérivant la

consommation totale du système par rapport à V_{bs} pour trouver sa valeur optimale. Ensuite, en définissant deux relations linéaires, ils déterminent les tensions optimales V_{dd} et V_{th}. Des simulations avec l'outil SPICE ont été réalisées sur un processeur Transmeta Crusoe 5600 pour des technologies 0,18µm et 0,07µm. Les résultats montrent que l'utilisation simultanée des techniques DVS et ABB permet une réduction moyenne de l'énergie de 48% par rapport à l'utilisation unique de la technique de DVS.

Si les travaux dans (Martin *et al*., 2002), ciblent des systèmes monoprocesseur et mono-tâche pour réduire la consommation totale du système, ceux développés dans (Yan *et al*., 2003) se sont intéressés à des systèmes multi-unités et multi-tâches. Ils présentent une approche qui combine les techniques ABB et DVS et opère en deux phases. La première phase permet de déterminer un compromis optimal entre la tension d'alimentation et la tension seuil pour chaque tâche et pour chaque fréquence de fonctionnement. La deuxième phase consiste à trouver un compromis entre l'énergie totale et le temps d'exécution d'une tâche. Ils définissent alors un critère *energy_derivative* qui indique le taux de réduction d'énergie si le temps d'exécution de la tâche T_i à une fréquence f_i est prolongé par un intervalle de temps (dt). Les tâches possédant la plus grande valeur *energy_derivative* sont les premières à être ordonnancées tout en respectant les échéances. Un gain moyen de 35% par rapport à l'utilisation d'une technique de DVS est obtenu pour une technologie 70nm.

Cette technique qui semble prometteuse manque encore d'expérimentations car les processeurs actuels ne possèdent pas de tension de seuil variable en fonction de la fréquence de fonctionnement.

2.1.2. Gestion des modes faible consommation

Une deuxième technique d'optimisation de la consommation statique processeur consiste à exploiter leurs différents modes faible consommation. Cette technique est connue en littérature sous le nom de DPM (*Dynamic Power Management*). Chaque mode est caractérisé par un nombre de ressources internes au processeur non alimentées, réduisant ainsi la consommation à la fois dynamique et statique. Plus le nombre de ressources non alimentées est important, plus le gain énergétique est significatif. En contre partie, plus le nombre de fonctionnalités opérationnelles est faible, plus la pénalité de retour en mode normal de fonctionnement est importante. Le tableau 1.4 résume les modes repos de quelques processeurs embarqués. L'objectif de cette technique est alors de choisir le mode faible consommation du processeur le plus intéressant, lorsqu'il existe des intervalles d'inactivité dans l'ordonnancement ou quand les tâches ne consomment pas la totalité de leur temps d'exécution au pire cas, pour à la fois réduire la consommation et respecter les échéances.

Tableau 1.4. Modes repos de quelques processeurs

Processeur	Modes Repos (Pénalités)
StrongARM SA110	Idle : 160mW (90 µs) Sleep : 50mW (160 ms)
LPARM (base : ARM8)	Idle : 500µW (1 cycle)
ARM9	Idle, slowclock, standby
ARM1176 (130µm)	Wait, Doze, State retention, Deep Sleep, Hibernate
Transmetta	Autohalt, Quick Start, DeepSleep : 100 mW
Intel PXA255	Idle: 555 mW Sleep : 180µW (300 µs , 600 µJ)
PowerPC603	Doze, Nap, Sleep : 5% active (10 cycles)

Dans (Lee *et al*., Juillet 2003) les auteurs essaient de regrouper les différentes exécutions des tâches afin de bénéficier, le plus longtemps possible, des modes repos les plus intéressants en terme d'énergie. Dans ces travaux, deux techniques d'ordonnancement pour des systèmes temps réel strict minimisant la consommation statique ont été développées. La première technique vise l'ordonnancement à priorité dynamique (EDF) appelée *Leakage-Control EDF* (LC-EDF). La deuxième vise les ordonnancements à priorité fixe (RM) appelée *Leakage-Control* Dual-Priority LC-DP. Ces deux techniques retardent les tâches prêtes pour exécution quand le processeur est inactif dans le but d'allonger sa période d'inactivité et regroupent les intervalles d'inactivité du processeur entre les différentes exécutions des tâches afin de diminuer le nombre de transitions entre les modes basse consommation. Pour obtenir l'algorithme *Leakage-Control EDF* (LC-EDF), plusieurs modifications ont été introduites à l'ordonnancement EDF. Si le processeur est inactif et lorsqu'une tâche τ_k ayant l'échéance la plus proche est prête pour exécution, cette dernière peut être retardée d'un intervalle Δ_k calculé à l'aide du test de faisabilité de l'EDF. Pour la technique *Leakage-Control* Dual-Priority *LC-DB* les auteurs ont modifié une technique d'ordonnancement appelée *Dual-Priority* (Davis *et al*., 1995). Des simulations utilisant le processeur Intel PXA250 ont montré une réduction importante de la consommation liée aux périodes d'inactivité.

2.1.3. Techniques de DVS avec gestion des modes faible consommation

L'article (Jejurikar *et al*., Juin 2004) propose une approche appelée CS-DVSP (*Critical Speed DVS with Procrastination*) qui vise à réduire la consommation dynamique et statique d'un ensemble de tâches périodiques ordonnancé avec la technique EDF. La composante dynamique est réduite par ajustement de la tension et de la fréquence des tâches quand le processeur est actif. La composante statique est réduite par prolongation

du temps d'inactivité du processeur par un intervalle de longueur Z_i quand il est déjà inactif. Une vitesse processeur critique η_{crit} minimisant la consommation totale est d'abord déterminée en se basant sur des modèles analytiques pour une technologie 70 nm. Ensuite par l'utilisation du test de faisabilité de l'ordonnancement EDF, un facteur de ralentissement statique η_i est calculé.

$$U_\eta = \sum_i \frac{1}{\eta_i} \frac{C_i}{T_i} \leq 1$$

Puisqu'une vitesse de fonctionnement η_i inférieure à la vitesse critique η_{crit} consomme plus d'énergie, la vitesse critique η_{crit} est imposée comme vitesse minimale de fonctionnement du processeur. Dans le cas contraire ($\eta_i > \eta_{crit}$) la vitesse de fonctionnement du processeur est η_i. Pour les modes basse consommation, un intervalle maximum de décalage (Z_i) est déterminé pour chaque tâche tout en garantissant les échéances de toutes les tâches. Ainsi si le processeur est déjà inactif, une tâche T_i prioritaire devenant active sera décalée d'un intervalle Z_i calculé selon l'équation ci-dessous.

$$\forall_{i=1..n}, \frac{Z_i}{T_i} + \sum_{k=1}^{i} \frac{1}{\eta_i} \frac{C_k}{T_k} \leq 1 \quad \forall_{k<i}, Z_k \leq Z_i$$

Les auteurs prouvent théoriquement que leur méthode est plus efficace dans la réduction d'énergie que la technique LC-EDF où toutes les tâches s'exécutent à une vitesse maximale.

Une autre technique, similaire à CS-DVSP, qui combine la méthode de DVS avec les modes faibles consommation pour minimiser la consommation totale du processeur est développée dans (Niu *et al.*, 2004). Cette technique appelée DVSLK vise le problème de la diminution simultanée des puissances dynamique et statique pour des systèmes à temps réel strict ordonnancés avec la politique EDF. Quand le processeur est actif, sa vitesse est choisie de façon à ce que les deux composantes dynamique et statique soient optimisées. Quand il est inactif, les tâches prêtes pour exécution sont retardées afin de regrouper les périodes d'inactivité entre les tâches et mettre le processeur en mode faible consommation pendant une période plus longue. Ainsi, le processeur peut bénéficier des modes faible consommation les plus profonds et les pénalités temporelles et énergétiques dues aux changements de modes sont réduites. La vitesse processeur optimale (s_{th}) qui minimise la consommation dynamique et statique totale du système est calculée par la dérivation de l'énergie totale par rapport à la vitesse processeur. L'augmentation ou la diminution de la vitesse processeur de la valeur optimale s_{th}, augmentera soit la composante dynamique soit la composante statique. Une vitesse processeur s_{DVS} est déterminée selon une technique de DVS nommée LPEDF (Yao *et al.*, 1995). Quand le processeur est actif, il exécute les différentes tâches soit avec la vitesse s_{th}, si $s_{DVS} < s_{th}$, soit avec la vitesse s_{DVS}, si $s_{DVS} \geq s_{th}$. Cependant, une vitesse supérieure à celle nécessaire cause de petits intervalles d'inactivité du processeur. Les auteurs proposent alors une technique pour regrouper ces intervalles afin de réduire les pénalités de changement de mode et d'allonger le temps d'inactivité du processeur en retardant le plus tard possible les tâches prêtes pour exécution tout en respectant les contraintes temporelles.

Une réduction de la consommation statique pouvant atteindre 80% par rapport à la technique CS-DVSP est obtenue. Ce gain important est essentiellement dû à l'efficacité de la stratégie d'estimation des intervalles de prolongation du temps d'inactivité du processeur. En effet, la technique DVSLK propose une approche de prédiction basée sur l'ensemble de tâches, tandis que CS-DVSP l'estime indépendamment pour chaque tâche.

2.2. Techniques de DVS tenant compte de la consommation mémoire

L'évolution technologique a permis de développer des architectures plus puissantes, permettant ainsi d'intégrer des fonctionnalités de plus en plus variées et nombreuses (vidéo, audio, téléphonie …) dans un même circuit. Comme conséquence la surface dédiée à la mémoire ne cesse d'augmenter. Selon les prévisions de ITRS 2005, la mémoire occupera une place de plus en plus large de la surface de silicium d'un système sur puce comme le montre la figure 1.2 (ITRS, 2005).

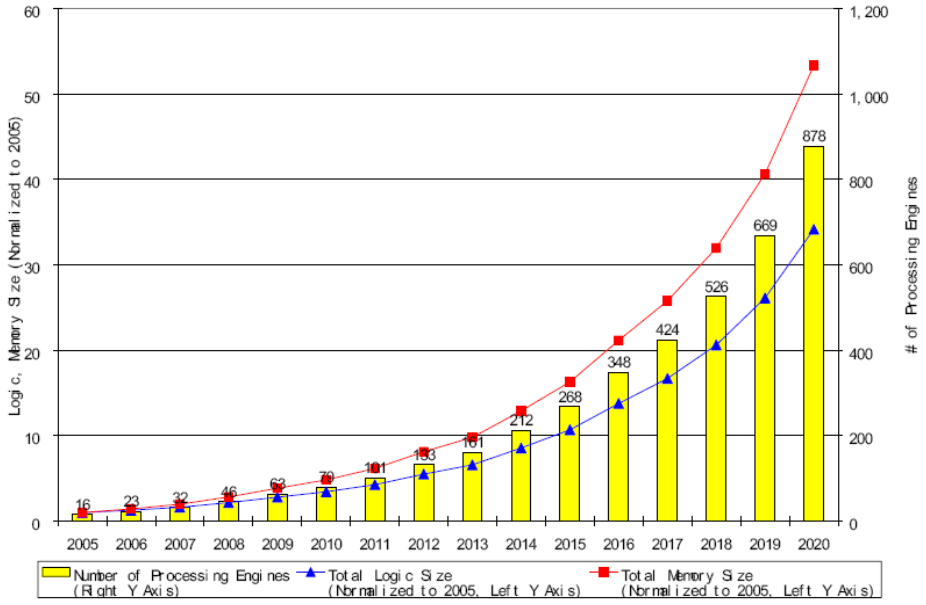


Figure 1.2. Evolution des contributions dans la surface des SoC

Dans (Fan *et al.*, 2003), les auteurs montrent que pour une application MPEG l'énergie consommée par le processeur et sa mémoire est de 10 mJ à 1GHz et elle est de 39 mJ à 50 MHZ. Ce résultat étonnant en premier abord puisqu'il contredit le principe de la technique du DVS, s'explique par deux constations. La première est une co-activation plus longue de la mémoire avec le processeur lors de la diminution de la fréquence processeur. La deuxième est l'augmentation du nombre de préemptions du fait de l'allongement des temps d'exécution des tâches par le DVS, entrainant ainsi plus de changements de contextes et d'accès à la mémoire pour la sauvegarde et la restitution des contextes (Kim *et al.*, 2004). Dans les paragraphes suivants nous montrons comment cet effet négatif du DVS sur la mémoire peut être surmonté. Dans le paragraphe 2.2.1 des techniques développées pour diminuer le nombre de préemptions lors de l'utilisation du DVS sont présentées. Dans le paragraphe 2.2.2, des techniques de DVS tenant compte de la mémoire externe associée à l'unité de calcul sont présentées.

2.2.1. Diminution du nombre de préemptions lors du DVS

Des expérimentations rapportées dans (Kim *et al*., Novembre 2003) montrent que le nombre de préemptions des tâches exécutées sur un processeur peut croître jusqu'à 500% avec une technique de DVS par rapport à une exécution sans DVS. Cette augmentation du nombre de préemptions accroît la consommation énergétique en particulier suite aux changements de contexte induits. Deux techniques de réduction des préemptions lors du DVS sont présentées dans (Kim *et al*., 2004). La première technique appelée *Accelerated-completion based preemption control technique*, permet d'accélérer l'exécution d'une tâche afin de finir son exécution avant l'arrivée d'une tâche plus prioritaire et d'éviter ainsi la préemption. Cette technique diminue le nombre de préemptions mais elle est responsable de l'augmentation de la fréquence de certaines tâches et par conséquent l'augmentation de la consommation. La deuxième technique proposée par les mêmes auteurs est appelée *Delayed-completion based preemption control technique*. La tâche la plus prioritaire est reportée pour permettre à la tâche de plus faible priorité de terminer son exécution sans être préemptée, tout en garantissant la faisabilité de l'ordonnancement. Cette tâche prioritaire est décalée d'un intervalle égal à la différence entre son échéance et son temps d'exécution. Ces deux techniques ont été intégrées dans une politique de DVS développée précédemment dans (Kim *et al*., Août 2003). Des expérimentations réalisées avec l'outil SimDVS (Shin *et al*., 2002) souligne l'efficacité de ces deux techniques à réduire le nombre de préemptions lorsqu'une technique de DVS est appliquée au processeur.

Suivant les mêmes objectifs, les auteurs de (Jejurikar *et al*., Août 2004) combinent l'algorithme *Preemptive Threshold Scheduling (PTS)*, qui diminue le nombre de préemptions dans les ordonnancements à priorité fixe (RM) ou dynamique (EDF), avec une technique de DVS ayant des facteurs d'étalement statiques. Les résultats montrent une réduction du nombre de changements de contexte de 90% par rapport à la technique de DVS.

Les auteurs de (Zhang *et al*., 2004) développent une approche de DVS pour des ordonnancements possédant des sections non préemptibles. Deux vitesses de fonctionnement sont calculées. Une faible vitesse est utilisée par le processeur quand il n'y a pas de section non préemptible. Le processeur bascule vers une vitesse élevée dès qu'une section non préemptible se présente afin d'accélérer l'exécution et garantir les échéances des tâches les plus prioritaires bloquées. Cette approche permet de réduire la consommation de plus de 60% dans le cadre de leurs exemples.

2.2.2. Techniques de DVS tenant compte des accès mémoires

Les travaux précédents ont diminué l'impact négatif de la technique de DVS sur la mémoire par la réduction du nombre de préemptions et de changements de contexte, facteurs d'accès plus fréquents à la mémoire principale. Cette approche a visé essentiellement les ordonnancements à priorité fixe et temps réel strict où le nombre de préemptions est facilement quantifiable. Les travaux cités dans ce paragraphe ont adopté une approche différente pour prendre en compte la consommation mémoire. Ils considèrent une décomposition de la charge de travail du processeur en exécution interne au processeur et en exécution externe due aux accès à la mémoire principale.

Dans (Choi *et al*., Février 2004), la fréquence processeur effective est déterminée en fonction de la fréquence processeur nominale, de la fréquence mémoire et d'un facteur de dégradation de performance. En tenant compte de la charge de travail du processeur

(exécution interne et accès mémoire), les auteurs déterminent un couple fréquence et tension de fonctionnement du processeur. Comme résultats, ils obtiennent, pour une dégradation de performance de 20%, un gain de 80% en énergie pour des applications à fort taux d'accès mémoire et un gain de 20% pour des applications à faible taux d'accès mémoire. Cette approche considère uniquement des systèmes mono-tâche à temps réel souple.

Dans (Marculescu, 2000), les auteurs s'intéressent, au niveau micro-architectural, aux pénalités temporelles causées par les défauts de cache lors de l'ajustement de la tension et de la fréquence pour réduire l'énergie. Les intervalles d'inactivité du processeur causés par les défauts de cache sont exploités pour exécuter les instructions restantes dans le pipeline à une fréquence plus faible, jusqu'à la résolution de ce défaut de cache. Comme perspective, les auteurs proposent l'utilisation d'une technique de prédiction des défauts de cache afin de mieux exploiter les intervalles d'inactivité pour réduire la fréquence avant même la production du défaut de cache. Des expérimentations montrent un gain d'énergie total pouvant atteindre 22% pour une dégradation de performance de 6%.

Une autre technique de DVS a été proposée dans (Choi *et al*., Novembre 2004). Elle tient compte, non seulement de la consommation active (processeur actif, accès mémoire) comme les techniques classiques de DVS, mais aussi de la consommation fixe (convertisseur DC-DC, PLL, consommation statique) et de la consommation d'inactivité notée *Standby* (processeur inactif, pas d'accès à la mémoire). La composante active est estimée en ligne par la décomposition de la charge du processeur en exécution d'instructions *on-chip* et des accès mémoire *off-chip*. L'estimation est basée sur des statistiques rapportées par l'unité PMU (*Performance Monitoring Unit*) disponible dans des processeurs comme le PXA255 et le XScale 80200. La composante fixe et celle d'inactivité du système sont mesurées en mettant le processeur et les différents modules en mode repos. Une fréquence processeur optimale est déterminée en prenant en compte, à la fois de la consommation active et de *Standby* du processeur, et à la fois de la consommation des différents modules du système (mémoire, convertisseur DC-DC, PLL…). Cette fréquence peut être supérieure à celle déterminée en considérant uniquement la consommation processeur. Toutefois, elle produit un gain d'énergie sur le système total plus important. Cette technique de DVS a été implémentée sur une plateforme BitsyX possédant un processeur RISC PXA255 et une mémoire principale Micron SDRAM de 64MB. Un gain de 18% sur l'énergie totale est obtenu par rapport aux techniques classiques de DVS.

Partie 2 : Mémoires Multi-Bancs

Introduction

Le partitionnement de l'espace dédié à la mémorisation des données et des instructions en plusieurs modules/bancs/partitions est une solution adoptée initialement par les concepteurs des systèmes embarqués pour augmenter les performances et réduire l'écart de performance entre le processeur et ses unités de mémorisations aussi bien sur les mémoires statiques internes SRAM (cache ou *scartchpad*) que sur les mémoires dynamiques externes DRAM. Ce partitionnement permet d'accélérer l'accès à la mémoire quand le processeur a besoin d'écrire ou de lire des données en cours d'exécution. Récemment, avec l'augmentation de la dissipation énergétique des mémoires dans les SoC, les concepteurs ont eu recourt à la même technique de partitionnement de l'espace de mémorisation pour réduire la consommation dynamique due aux accès puis à réduire également la consommation statique. Cette optimisation des deux composantes énergétiques est devenue possible en rendant l'alimentation de chaque module indépendante et en mettant hors tension ou en mode faible consommation un module dès que celui-ci ne dessert aucune demande d'accès en lecture ou en écriture. Bien que les mémoires multi-bancs permettent d'augmenter les performances et d'économiser de l'énergie, elles ont tendance à augmenter la surface de silicium, surtout s'il s'agit de les embarquer avec le processeur sur la même puce.

Dans cette partie, nous détaillons dans un premier temps les travaux qui visent à augmenter les performances dans les mémoires SRAM, puis dans les mémoires DRAM. Dans un second temps, on cite les recherches effectuées pour réduire la consommation mémoire par division de l'espace de mémorisation dans les mémoires principales DRAM, puis dans les mémoires SRAM. Nous finissons cette partie par la présentation de solutions relatives au problème de l'augmentation de la surface dédiée à la mémoire par son partitionnement en plusieurs bancs.

1. Mémoires multi-bancs pour améliorer la performance

Ce paragraphe présente quelques travaux qui exploitent le partitionnement mémoire en plusieurs bancs pour augmenter les performances des mémoires internes SRAM (scratchpad puis les caches) et des mémoires principales DRAM.

1.1. Mémoires scratchpad multi-bancs pour améliorer la performance

Les processeurs de traitement de signal (DSP) sont parmi les premiers à avoir adopté une structure mémoire multi-bancs. Par exemple, les NEC uPD77016, Analog Device ADSP2100 et DSP56000 de Motorola possèdent 3 bancs mémoire, un banc pour le programme et deux bancs pour les données, chacun avec un espace d'adressage indépendant. Chaque banc mémoire (donnée/instruction) possède son propre bus de données ou d'instructions. Cette architecture permet d'augmenter la bande passante de la mémoire pour accélérer les algorithmes classiques en traitement de signal comme le filtre FIR ou la FFT.

L'exploitation d'une mémoire à deux bancs dans les DSP nécessite d'allouer les données soit manuellement en utilisant des directives assembleur (Powell *et al*., 1992) soit au moment de la compilation (Procaskey, 1995).

En synthèse comportementale, Panda (Panda, 1999) traite le problème d'assignation des données aux bancs mémoire ainsi que la configuration mémoire pour une application spécifique. Dans l'approche présentée par (Leupers *et al*., 2001), des graphes décrivant les dépendances entre les données sont construits pour exploiter le parallélisme. Une approche ILP (*Integer linear Programming*) est ensuite appliquée pour partitionner le graphe de dépendance. (Cho *et al*., 2004) décrit l'implémentation d'un algorithme de génération de code permettant au compilateur une assignation efficace des données aux différents bancs mémoire pour des DSPs à virgule fixe avec des registres hétérogènes. Une heuristique basée sur une méthode de recuit simulé est développée.

1.2. Cache multi-bancs pour améliorer la performance

Le partitionnement d'un cache en plusieurs bancs nécessite l'entrelacement des lignes du cache, de telle sorte qu'une ligne de cache réside complètement dans un banc et que des lignes consécutives résident dans des bancs successifs. L'entrelacement des lignes de cache permet de réduire les conflits entre les différents bancs. Le MIPS R10000 est un exemple de processeur intégrant un cache multi-bancs. Il intègre un cache primaire de données de 32 KB avec une taille de ligne de 32 bytes (Yeager, 1996), partitionné en deux bancs entrelacés. Chaque banc est associatif par ensemble à deux voies. L'entrelacement des deux bancs permet à la fois, la mise à jour du cache interne par une donnée en défaut et l'accès par le microprocesseur.

Les caches multi-bancs sont efficaces dans la réduction du nombre de défauts de cache mais couteux puisqu'ils nécessitent la duplication des cellules mémoire des étiquettes (*tag*) dans chaque banc et l'utilisation de plusieurs ports pour permettre des accès simultanés à tous les bancs. Les auteurs de (Johguchi *et al*., 2005) proposent une architecture mémoire cache unifiée de données et d'instructions multi-bancs et multi-ports, afin de réduire le nombre de défauts de cache et augmenter les performances. Ils

présentent, pour un processeur superscalaire, une architecture de cache associative constituée de 64 bancs de 2 Kbit chacun avec 1 port d'instructions et 2 ports de données.

Pour réduire la pénalité temporelle causée par un défaut de cache L1, un cache de niveau L2 est souvent considéré. La taille de ce cache devient de plus en plus grande dans les processeurs à hautes performances. Elle peut varier de 128 KB à plusieurs MB (tableau 1.5). Cette tendance à l'augmentation de la taille du cache L2, entraîne l'augmentation également de sa latence. Une technique pour diminuer le temps d'accès du cache L2 consiste à diviser l'espace d'adressage en plusieurs bancs. L'intégration d'un cache L2 est de plus en plus fréquente dans les SoC.

Tableau 1.5. Evolution du cache L2 dans des processeurs hautes performances

Processeur	**Taille Max du Cache L2**	**Localisation**
Intel 486	512 KB	Externe
Intel Pentium	512 KB	Externe
Intel Pentium Pro	1024 KB	On chip
Intel Pentium 2	512 KB	Externe
Intel Celeron	0 KB	NA
Intel Celeron Pro	128 KB	On chip
Intel Xeon	2048 KB	Externe
AMD K6-2	2048 KB	Externe
AMD K6-3	256 KB	On chip

1.3. Mémoires externes DRAM multi-bancs pour améliorer la performance

Les premières mémoires DRAM multi-bancs *Multibank DRAM* (MDRAM) ont été développées par la société MoSys. Chaque banc possède son propre port d'entrée/sortie alimentant un bus interne commun. Cette architecture permet des lectures ou écritures simultanées de plusieurs données ce qui les rend beaucoup plus rapides que les DRAM monolithiques. Chaque banc est individuellement accédé. Les accès aux bancs sont entrelacés et ainsi des accès parallèles aux différents bancs seront possibles. Cette technologie mémoire permet d'ajuster également la taille mémoire aux besoins de l'application. Les mémoires MDRAM intègrent un bus interne connecté aux différents bancs et un large bus vers le processeur, permettant une bande passante importante (figure 1.3).

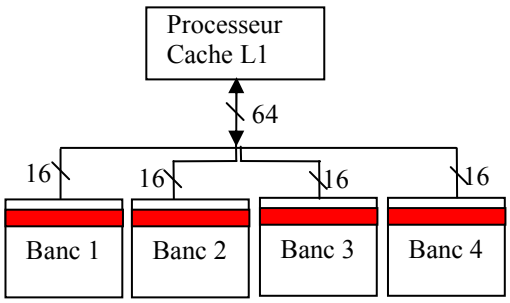


Figure 1.3. Accès mémoire avec entrelacement (*Interleaving*)

Ces mémoires nécessitent cependant une activation simultanée de tous les bancs pour effectuer un accès parallèle ce qui entraîne une augmentation de l'énergie consommée suite à l'activation simultanée de tous les bancs mémoires.

2. Mémoires multi-bancs pour réduire la consommation

La division de l'espace d'adressage en plusieurs modules ou bancs (figure 1.4) a été récemment réutilisée pour réduire l'énergie consommée par les cellules de mémorisation. Cette division en plusieurs bancs, réduit soit uniquement l'énergie dynamique due aux accès (bancs de taille plus faible) soit les deux énergies dynamique et statique en offrant la possibilité de mettre indépendamment chaque banc dans un mode repos, s'il ne dessert aucune demande d'accès.

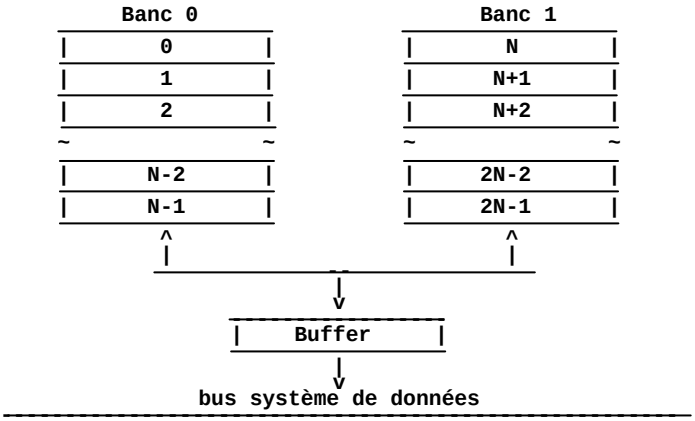


Figure 1.4. Organisation mémoire à accès non entrelacés

2.1. Mémoires externes DRAM multi-bancs pour réduire la consommation

2.1.1. Exemples de mémoires DRAM multi-bancs avec mode repos

Parmi ce type de mémoire, nous citons les mémoires RDRAM de RAMBUS *Technology* (Rambus, 1999), les mémoires Mobile RAM d'Infineon (Infineon, 2004) et les mémoires DDR SDRAM de Micron (Micron, 2006).

- **RDRAM de RAMBUS**

Contrairement aux mémoires DRAM multi-bancs à accès entrelacés, un banc unique fournit la bande passante sur un bus de 16 bits. Un nombre important de bancs est possible dans un composant RDRAM (figure 1.6). On trouve ce type de mémoire dans les mémoires vidéo des consoles de jeux comme Nintendo, PlayStation2 et PlayStation 3 de Sony.

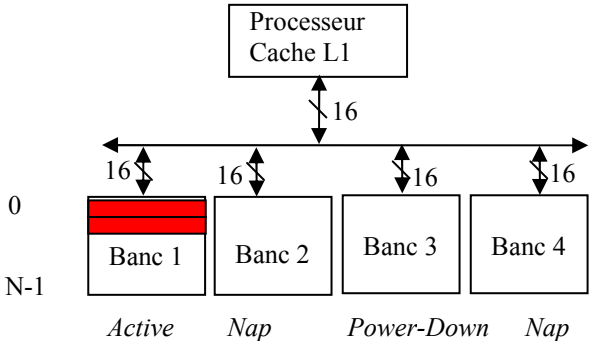


Figure 1.5. Architecture mémoire multi-bancs

Pour servir une demande d'accès en lecture ou écriture, le banc mémoire doit être en mode actif, le mode le plus gourmand en énergie, sinon il peut être basculé vers l'un des modes faible consommation : *Standby, Nap, Power-Down* (figure 1.5).

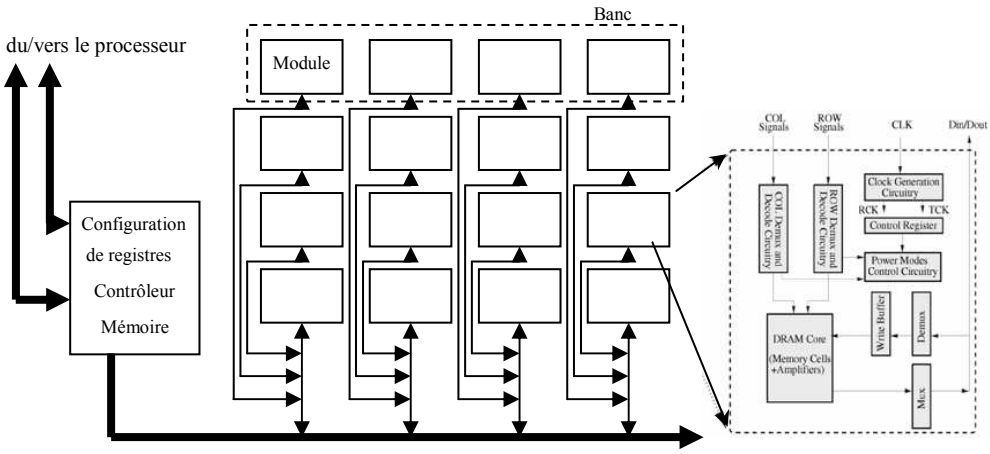


Figure 1.6. Architecture mémoire RDRAM

Chaque mode est caractérisé par une puissance dissipée et un temps de retour au mode actif (temps de resynchronisation). Plus le niveau d'énergie du mode basse consommation est faible (moins il y a de ressources internes alimentées), plus le temps de resynchronisation est important (figure 1.7).

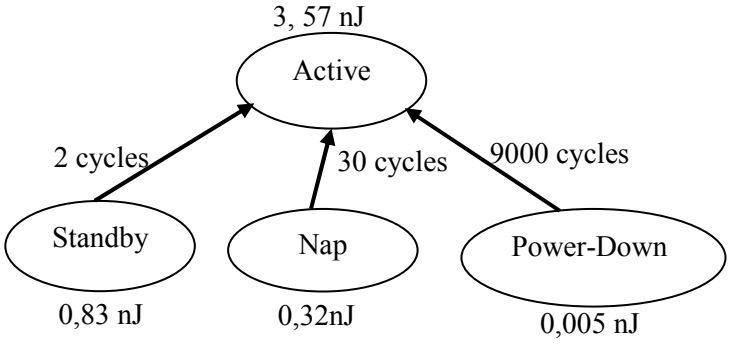


Figure 1.7. Modes de fonctionnement d'une mémoire RDRAM

Le tableau 1.6 décrit les ressources mises hors tension pour réduire la consommation dans chaque mode repos.

Tableau 1.6. Ressources non alimentées pour chaque mode repos d'une mémoire RDRAM

Modes repos	**Ressources non alimentées**
Standby	Décodeurs de colonnes non alimentés
Nap	Décodeurs de colonnes et de lignes non alimentés
Power-Down	Décodeurs de colonnes et de lignes non alimentés ainsi que l'horloge de synchronisation, sauf le *self-refresh*

- **Mobile-RAM d'Infineon**

Une mémoire Mobile DRAM possède généralement 4 bancs mémoire. Ces mémoires utilisent l'entrelacement des accès pour augmenter les performances. C'est une mémoire SDRAM faible consommation, avec une densité moyenne, optimisée pour des applications embarquées alimentées par batterie. Ses caractéristiques de faible consommation (tableau 1.7) permettent de réduire la puissance consommée jusqu'à 80% par rapport à une mémoire SDRAM standard, ayant une densité comparable.

Tableau 1.7. Mobile-RAM d'Infineon (Fan *et al*., 2003)

Modes de fonctionnement	**Puissance (mW)**	**temps (ns)**
Active	275	$T_{accès}$= 90
Standby	75	-
Powerdown	1.75	-
Transitions		
Standby →Active	-	+0
Powerdown →Active	138	+7.5

- **DDR SDRAM de Micron**

Ce type de mémoire permet de contrôler les modes repos par rangée ou *rank* de plusieurs bancs alors que dans les RDRAM le contrôle s'effectue au niveau de chaque banc. Les caractéristiques de cette mémoire (figure 1.8) la rendent intéressante dans la réduction de la consommation dans les systèmes embarqués et les ordinateurs personnels.

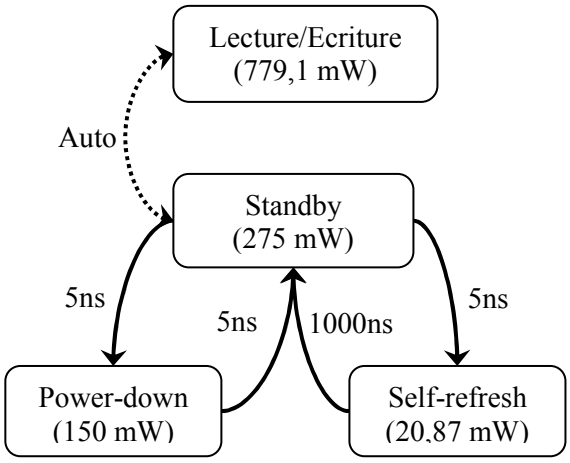


Figure 1.8. 512MB DDR module avec 8 bancs par rangée (Huang *et al*., 2004)

2.1.2. Techniques de contrôle des modes dans les mémoires DRAM multi-bancs

Les mémoires dynamiques DRAM multi-bancs munies de plusieurs modes de fonctionnement nécessitent un contrôleur qui décide quand (l'instant) et vers quels modes repos un banc doit transiter. Ces techniques de gestion des modes repos sont soit matérielles, soit logicielles (compilateur), soit basées sur le système d'exploitation ou encore hybrides matérielles et logicielles.

2.1.2.1 Techniques matérielles de gestion des modes faible consommation

Le contrôleur matériel intégré dans l'architecture mémoire gère les modes de chaque banc. Ce module physique contient généralement des compteurs ainsi que des registres. La technique consiste à basculer un banc vers un mode de repos plus profond après une période de temps de non accès. L'hypothèse considérée est que si un module n'a pas été accédé pendant une certaine période de temps, il est susceptible de ne pas être accédé dans un avenir proche. Un seuil de basculement est employé pour déterminer la période d'inactivité d'un banc à partir de laquelle il sera basculé vers un mode de repos plus profond.

- ***Adaptive Threshold Predictor*** **(ATP)**

Cette approche adapte dynamiquement la valeur du seuil de basculement. L'emploi de différents seuils à des instants différents de l'exécution d'une application permet de mieux prévoir les périodes d'inactivité. Ce mécanisme commence avec un seuil initial, et bascule vers un mode de repos plus profond si le module n'est pas accédé durant la

période correspondante. Si le prochain accès est survenu peu après cette période (la consommation d'énergie de resynchronisation est dans ce cas plus importante que le gain obtenu par le basculement vers le mode repos), la valeur du seuil de basculement est alors doublée. Inversement, si le prochain accès survient tardivement, et que la valeur du seuil était conservatrice, le seuil est remis à sa valeur initiale.

- ***Constant Threshold Predictor* (CTP)**

Cette technique est similaire à la précédente à l'exception de la valeur du seuil qui n'est jamais doublée à cause de l'important surcoût matériel de l'implémentation de l'approche à seuil adaptatif (ATP). Le banc est graduellement basculé vers des modes repos de plus en plus profond en fonction du temps de non accès au banc (figure 1.9).

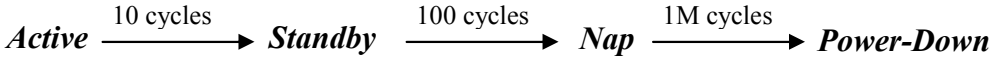


Figure 1.9. Transition progressive d'un mode repos vers un autre plus profond

Deux principaux inconvénients ont été soulevés dans les deux techniques ATP et CTP. Premièrement, le basculement d'un banc vers un mode repos plus profond s'effectue progressivement, d'un mode vers un autre (c.-à-d., pour entrer dans le mode *Power-Down*, le banc doit passer par les modes *Standby* puis *Nap*). Le banc pourrait avoir une transition directe vers le mode repos souhaité si une prédiction efficace était employée. Deuxièmement, une pénalité de resynchronisation est inévitable lors d'un accès mémoire si le banc est déjà en mode faible consommation.

- **Prédiction basé sur l'historique (HBP)**

Dans l'approche HBP, les temps entre les accès aux différents bancs sont estimés. Le principe utilisé est que les temps entre les accès précédents sont indicatifs des temps entre les accès futurs. La transition s'effectue alors directement vers le meilleur mode repos et le banc est pré-activé avant le prochain accès estimé pour éviter la pénalité de resynchronisation. Le mécanisme d'estimation du temps entre les accès (IAT) se base sur une table qui contient les valeurs maximales et minimales des temps entre les accès estimés pour lesquels un mode repos particulier est optimum. La table est préconstruite à base des valeurs d'énergies et des temps de resynchronisation des différents modes, et doit contenir autant d'entrées que de modes repos. Une fois le mode repos déterminé, le temps correspondant de resynchronisation est soustrait au temps IAT estimé, afin de déterminer la période exacte de mise en repos du banc.

Des expérimentations reportées dans (Delaluz *et al*., Janvier 2001) montrent que la technique CTP consomme en moyenne 27% d'énergie en plus que l'optimal. La technique HBP se distingue par sa grande capacité à réduire la consommation par rapport aux deux autres techniques ; en moyenne 9% d'énergie est consommée en plus de l'optimal. Cette efficacité des techniques matérielles est obtenue grâce à la grande réactivité (de l'ordre de 10ns à 100µs) de l'approche matérielle.

Certes ces techniques matérielles sont efficaces dans la réduction de l'énergie mémoire, elles introduisent un coût supplémentaire suite au matériel ajouté qui consomme lui-même une part d'énergie. Ces techniques ne sont pas aussi flexibles et nécessitent la modification des seuils de basculement à chaque changement de

l'application. Dans un contexte multi-tâches, ces approches matérielles sont difficilement applicables.

2.1.2.2. Techniques logicielles de gestion des modes faibles consommation

La gestion des modes repos des mémoires multi-bancs par le compilateur peut s'effectuer de deux manières différentes. La première consiste à basculer les bancs vers un mode repos lorsqu'une période de non accès est identifiée sans aucune modification du code ni des données à l'exception de l'insertion explicite d'instructions d'activation ou de mise en repos. La deuxième consiste à gérer les modes repos tout en effectuant des transformations sur les structures de données (exemple le *clustering*) ou des optimisations de code (*loop splitting*) figure 1.10.

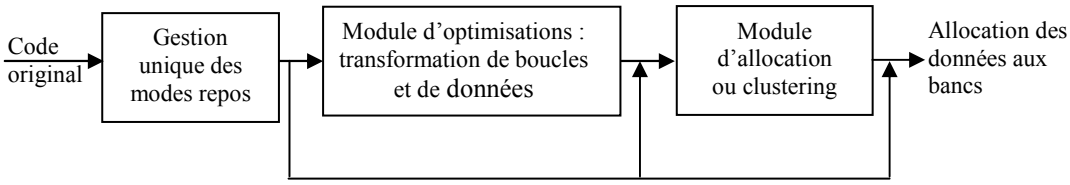


Figure 1.10. Techniques de gestions de modes repos mémoire au niveau compilateur

- **Gestion des modes de fonctionnement par insertion d'instructions**

Le compilateur doit identifier les périodes d'inactivités des bancs mémoires et effectuer une pré-allocation des données aux bancs. La gestion des modes repos s'effectue par l'insertion explicite d'instructions d'activation et de mise en repos. Cependant, ces instructions peuvent introduire une pénalité sur le temps d'exécution processeur. Cette pénalité peut être maitrisée s'il s'agit d'opérer sur des sections de code importantes. De même, pour réduire les pénalités sur les performances, un banc est pré-activé avant la date de son prochain accès. Le temps de pré-activation dépend du temps de resynchronisation du mode repos. Le compilateur sélectionne le mode repos adéquat selon le temps d'inactivité T. En effet, si le temps entre deux accès successifs est T, et le temps de resynchronisation est T_p, le compilateur mettra le banc dans un mode faible consommation durant la période $T-T_p$. Il active le banc à la fin de cette période pour le remettre en mode de fonctionnement normal ce qui évite les temps de resynchronisation avant le prochain accès.

Les résultats dans (Delaluz *et al*., Janvier 2001) montrent que la technique de contrôle des modes repos par le compilateur est intéressante mais conduit en moyenne à 48% d'énergie en plus par rapport à l'optimal, montrant ainsi une efficacité moindre que celle obtenue par l'approche matérielle. Des résultats montrent également que l'utilisation unique du mode *Nap* comme mode repos permet d'obtenir un gain meilleur ou égal à celui obtenu par l'utilisation du mode *Standby*, du mode *Power-Down* ou même l'utilisation simultanée des trois modes.

- **Regroupement (*clustering*) et allocation des données**

Classiquement le compilateur alloue les données aux bancs dans l'ordre de leurs déclarations dans le programme. Cette façon de placer les données dans les bancs ne

permet pas dans la plupart des cas de profiter efficacement des modes repos les plus profonds et d'obtenir de meilleurs gains. Plusieurs techniques de placement de données aux bancs ont été proposées dans la littérature. Elles exploitent la localité temporelle des données pour les placer dans un nombre minimum de bancs afin de minimiser la consommation mémoire. Le problème du placement de données dans une mémoire multi-bancs est NP-complet (Farrahi *et al*., 1998) (Mace, 1987). Des heuristiques de groupement de données ont été d'abord proposées en considérant un placement statique des données. Avec l'apparition d'applications à profil d'accès variables pendant l'exécution, les placements statiques ne permettent plus d'atteindre des gains d'énergie intéressants. Ainsi, des heuristiques de placement dynamique des données ont été proposées. Ces heuristiques consistent à placer les données ayant un modèle d'accès similaire en cours d'exécution dans un même banc.

- ***Placement statique des données***

Les heuristiques de placement statiques des données ont pour but de collecter des informations sur la localité temporelle des données. Ces informations sont obtenues soit par analyse du code soit par *profiling*.

Par analyse du programme, les auteurs de (Delaluz *et al*., Janvier 2001) identifient les données avec des profils d'accès similaires. Ils utilisent cette information pour modifier l'ordre de déclaration de ces données. Ceux à profils d'accès similaires sont déclarés consécutivement dans le programme nécessitant ainsi une réécriture du code. L'heuristique proposée consiste à diviser le programme en plusieurs phases et à détecter les variables utilisées dans chaque phase. Quand une phase finie son exécution, le compilateur peut mettre en mode repos les bancs qui ne seront pas utilisés dans la prochaine phase. Les bancs qui seront utilisés dans la phase p, sont mis en mode actif à la fin de la phase p-1. Des expérimentations sur douze applications test montrent que lorsque le compilateur contrôle les modes repos sans le *clustering*, des gains de 75% peuvent être obtenus par rapport à l'énergie consommée en l'absence de gestion des modes repos. Si en plus la technique de *clustering* est employée des gains allant jusqu'à 85% sont obtenus.

Les auteurs de (Delaluz *et al*., Janvier 2000) développent une heuristique qui, par analyse statique du code, construit un graphe ARG (*Array Relation Graph*) représentant les localités temporelles entre les différentes données d'une application. Les nœuds représentent les données et les arcs entre deux nœuds représentent le nombre de fois que ces deux données sont accédées dans une itération de boucle. L'algorithme place alors les données voisines dans des emplacements mémoire adjacents. Des expérimentations sur des programmes test montrent une amélioration de 6% en moyenne par rapport au gain obtenu avec uniquement la gestion des modes repos par insertion d'instructions d'activation/désactivation des bancs.

Une hypothèse adoptée par les travaux précédents consiste à considérer des bancs mémoire avec une même taille. Bien que cette hypothèse rende la formulation du problème plus simple, elle réduit l'espace d'exploration et réduit l'opportunité de baisser la consommation en autorisant des espaces mémoire qui ne seront jamais utilisés. Dans (Ozturk *et al*., Mars 2005), les auteurs adoptent une architecture mémoire avec des bancs de tailles différentes (non-uniformes). Une formulation avec l'approche ILP permet de fournir l'allocation des données aux bancs ainsi que leurs tailles non uniformes.

Pour maximiser l'utilisation des modes faible consommation des bancs mémoire, les auteurs de (Lyuh *et al*., 2004) présentent une heuristique itérative pour résoudre simultanément : l'assignation des variables aux bancs mémoires ainsi que le ré-ordonnancement des accès mémoire représentés par un graphe DFG (*Data Flow Graph*). Ils montrent que l'approche proposée permet de réduire la consommation de 15% en moyenne par rapport à une technique qui alloue les données dans l'ordre de leur apparition dans le code de l'application.

- ***Placement dynamique des données***

Dans les applications embarquées à fort taux d'accès mémoire, le profil d'accès des différents variables varie généralement en cours d'exécution. Contrairement aux approches précédentes, les auteurs de (Delaluz *et al*., 2002) proposent une technique de placement dynamique des données qui permet de modifier l'emplacement initial des données par migration d'un banc vers un autre. L'idée est de regrouper des données initialement allouées à des bancs différents mais qui sont récemment simultanément accédées. L'implémentation de la migration nécessite la détection en ligne de la localité temporelle des variables ainsi que le moment opportun de la migration des données. La détection des localités temporelles des données se fait régulièrement à chaque intervalle de temps ST (intervalle d'échantillonnage). Ces informations sont stockées dans une table ayant comme entrée des triplets de type (A, B, count) qui sauvegarde le nombre de fois (count) que les données A et B sont accédées ensemble. La migration s'effectue à chaque intervalle MT appelé seuil de migration. Afin de déterminer les données à migrer, deux approches ont été présentées. La première approche nommée mFT migre les m triplets possédant les plus grandes valeurs du paramètre count. La deuxième approche nommée nLM, considère pour la migration uniquement les triplets ayant des valeurs de compteurs supérieures à une valeur seuil n. Des expérimentations montrent un gain moyen supplémentaire de 19% obtenu en utilisant la technique de migration. Bien que cette technique puisse mieux détecter les localités temporelles des données et tenir compte de l'aspect variable des profils d'accès, elle peut être pénalisante vis à vis des performances du fait de la migration. La taille de la table devient importante avec l'augmentation du nombre de données de l'application. Aussi, si l'intervalle d'échantillonnage est faible, une pénalité énorme sur le temps d'exécution est introduite. Un faible seuil de migration entraîne une perte de performance et au contraire, un seuil élevé peut écarter des opportunités intéressantes de migration de données.

Dans (Ozturk *et al*., Avril 2005) les auteurs présentent aussi une technique de migration de données en cours d'exécution (en ligne) afin de mieux exploiter les modes repos disponibles. La migration place les blocs de données ayant un profil d'accès similaire dans le même ensemble de bancs. Cette technique de gestion des modes repos des bancs est formulée sous un problème ILP. Les informations sur les profils d'accès des données sont extraites de la compilation. La migration optimale est déterminée par la résolution du problème ILP et ensuite communiquée au compilateur afin d'insérer dans le programme des instructions de migration. Dans cette formulation, le nombre de bancs est constant et un seul mode repos est considéré.

Les auteurs de (Ozturk *et al*., 2006) proposent une approche qui s'appuie sur la duplication de données sur plus d'un banc mémoire afin d'éviter une réactivation d'un banc déjà en mode faible consommation. La duplication de blocs de données se base sur la localité temporelle des blocs. Deux approches ont été présentées, l'une basée sur une formulation ILP et l'autre sur une heuristique, pour déterminer le bloc de données à dupliquer ainsi que le placement du bloc dupliqué dans les bancs. Cette approche tend à

ne pas dépasser une surface de mémoire supplémentaire lors de la duplication de données en lecture seulement.

Les techniques de gestion des modes repos par le compilateur ainsi que les heuristiques de regroupement de données peuvent être employées seules ou précédées de techniques d'optimisation de code qui sont introduites dans le paragraphe suivant.

- **Transformation de code : optimisation du code et/ou des données**

Des techniques de transformation de code et/ou de données ajoutées aux techniques de contrôle de modes repos et aux heuristiques d'allocation améliorent le gain d'énergie obtenu. Ces transformations de code et/ou de données effectuées à la compilation, visent l'augmentation de la localité spatiale et temporelle des données dans les boucles afin de réduire les défauts de cache générateurs de baisse de performances et d'augmentation de l'énergie consommée par les accès mémoire. Nous présentons dans la suite quelques techniques classiques.

- ***Loop fission***

La technique de distribution de boucles (*Loop fission* ou *distribution*) consiste à diviser une boucle ayant un ensemble d'opérations en plusieurs boucles, chacune contenant un sous-ensemble de ces opérations. Cette technique est utilisée pour augmenter la performance par l'amélioration de la localité des données dans le cache. Son utilisation dans le cadre de la diminution de la consommation d'énergie des mémoires multi-bancs, est illustrée dans l'exemple de la figure 1.11.

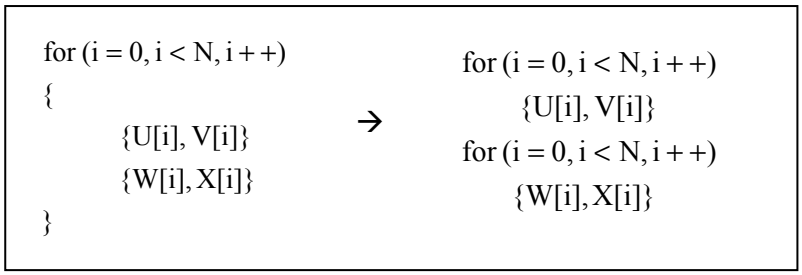


Figure 1.11. Exemple de distribution de boucle (*fission*)

Si nous considérons que les tableaux U et V sont placés dans un banc et les tableaux W et X sont placés dans un deuxième banc, la boucle originale exige que les deux bancs soient actifs en même temps lors de l'exécution de la boucle. Après distribution de la boucle, un seul banc est en mode *Active* à la fois, durant l'exécution de chaque boucle.

- ***Loop splitting***

Cette technique consiste à diviser l'ensemble des valeurs prises par l'indice d'une boucle en deux ou plusieurs parties disjointes. Supposons que U et V de la figure 1.12 occupent chacun deux bancs mémoires, durant l'exécution de la boucle initiale tous les quatre bancs doivent être en mode *Active*. La technique du *loop splitting*, permet de maintenir uniquement deux bancs en mode *Active* durant l'exécution de chaque boucle, les deux autres bancs seront en mode repos consommant ainsi moins d'énergie.

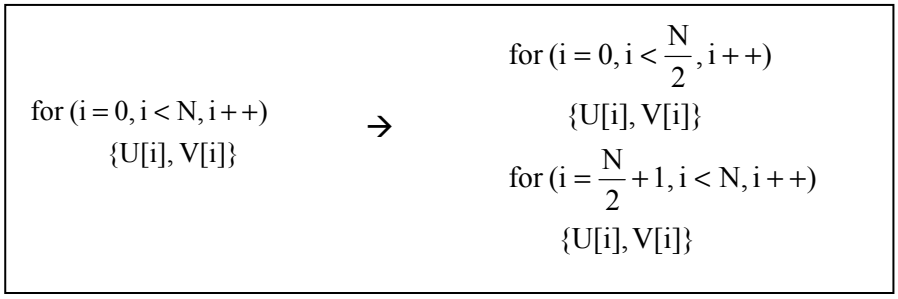


Figure 1.12. Technique de *splitting*

D'autres techniques de transformation de boucles comme la fusion de boucle, les transformations linéaires de boucle, et celle de *loop tiling* sont employées dans (Kandemir *et al*., 2002) pour minimiser la consommation des mémoires multi-bancs.

La stratégie présentée dans (Kandemir *et al*., 2001), modifie l'ordre d'exécution original des itérations des boucles dans les applications à grand nombre d'accès mémoire afin d'augmenter la période d'inactivité des bancs et afin de mettre plus de bancs en mode faible consommation et de bénéficier des modes repos plus profonds. Dans un premier temps, les itérations de boucles sont classées selon leurs profils d'accès. Dans un second temps, l'outil *Omega library*, place les itérations avec des profils d'accès similaires ensemble. Cette technique permet d'améliorer la consommation mémoire de 34%.

Les transformations précédentes opèrent sur le code de l'application et plus particulièrement sur les boucles. Dans le paragraphe suivant, quelques techniques de transformations effectuées sur les données sont présentées.

- ***Array renaming***

La technique *array renaming* des tableaux est une optimisation qui exploite la possibilité de réutiliser un même emplacement mémoire pour ranger différents tableaux, ayant des durées de vie disjointes. Dans l'exemple de la figure 1.13, les tableaux U et W ont des durées de vie disjointes. On suppose que U n'est pas sollicité après l'exécution de la première boucle et que U possède une taille supérieure ou égale à W. Si l'allocateur place U et V dans un banc et W dans un deuxième banc, l'exécution du code de gauche nécessite un seul banc actif pour la première boucle et deux bancs actifs pour la deuxième boucle. On peut utiliser l'espace mémoire réservé à U pour stocker W. En appliquant ce renommage, un seul banc actif à la fois est nécessaire (code de droite).

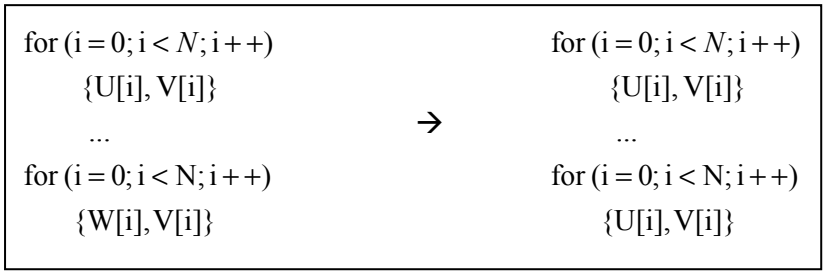


Figure 1.13. Technique de renommage de tableau

- ***Compression de données***

La compression de données permet de réduire la taille des données utilisées par une application. Le nombre de bancs occupés par les données est aussi réduit, ce qui tend à augmenter le nombre de bancs qui peuvent être mis en mode faible consommation. Cette technique a été combinée avec une technique de gestion des modes repos et formulée sous un problème ILP dans (Ozturk *et al*., Avril 2005). Des instructions explicites de compression/décompression sont ajoutées au code par le compilateur.

Dans (Delaluz *et al.*, 2000), des expérimentations sur plusieurs programmes test, ont permis d'évaluer les trois techniques d'optimisations : *loop fission*, *loop splitting* et *data renaming*. La technique *loop fission* permet une amélioration de l'énergie en moyenne de 55% par rapport à la technique de regroupement et d'allocation de données présentée ci-dessus. La technique *Loop splitting* appliquée à la boucle la plus consommatrice, permet une amélioration moyenne du gain d'énergie de 42% par rapport à la technique de regroupement et d'allocation de données présentée ci-dessus. Avec la technique *array renaming*, uniquement deux programmes profitent de cette optimisation. Les combinaisons de ces trois techniques améliorent encore le gain d'énergie. Par exemple, sur un programme test, un gain supplémentaire de 50% est obtenu par l'utilisation des deux techniques de *loop fission* et *loop spliting*.

Les techniques d'optimisation de la consommation mémoire au niveau des compilateurs ont l'avantage d'introduire une pénalité négligeable sur la performance. Cependant, toutes les informations liées aux accès ne sont pas disponibles lors de la phase de compilation réduisant ainsi les potentialités d'optimisation. Pour des SoC on peut penser qu'on dispose des codes sources, sinon on réécrit le code si nécessaire. Par ailleurs, les techniques d'optimisation appliquées à la compilation considèrent des adresses physiques en mémoire et sont donc peu efficace si un adressage virtuel est utilisé à l'exécution. De plus, elles sont généralement mono-tâche.

2.1.2.3. Techniques supportées par le système d'exploitation

Contrairement au cas du compilateur qui peut uniquement traiter une seule tâche à la fois et nécessite l'aide de fonctions d'analyse sophistiquées, le système d'exploitation (OS) possède l'avantage d'avoir une vue globale du système et de permettre une prédiction correcte des périodes d'inactivité des bancs afin de déterminer le mode faible consommation adéquat.

Delaluz *et al.*, (Delaluz *et al*., Juin 2002) proposent une technique où l'ordonnanceur contrôle les transitions vers les modes faible consommation en gardant une trace des accès à chaque banc et pour toutes les tâches du système. Ces traces d'accès sont contenues dans une table d'usage *Bank Usage Table* (BUT) reflétant l'utilisation des bancs par chaque tâche à chaque quantum de temps. Lorsqu'une tâche T_i est en exécution et utilise le banc j, ce dernier est gardé actif et les autres bancs peuvent être mis dans un mode repos. Le banc qui n'a pas été accédé ne sera pas accédé durant les prochains quanta de la tâche en exécution. L'OS permet de déterminer des intervalles de temps durant l'exécution d'une tâche où des bancs restent potentiellement inactifs (non accès), et ainsi peuvent être basculés vers un mode faible consommation. Cette table nécessite une mise à jour à chaque changement de contexte. La plateforme d'évaluation utilisée est une UltraSparc 5 avec Linux RH 6.2. La mémoire physique disponible est composée de 16 bancs. Le quantum du système d'exploitation est de 200ms, les lignes de la table BUT

sont implémentées comme une partie des structures des tâches dans le noyau. Cette implémentation montre que la technique proposée est également robuste quand différents paramètres du système et de charge de travail sont modifiés. Elle fournit les premiers résultats expérimentaux pour l'optimisation de l'énergie mémoire des systèmes multi-tâches sur une plateforme réelle. La technique proposée est applicable aussi bien sur des systèmes embarqués que sur des plateformes de calcul à hautes performances.

Dans (Park *et al*., 2003), une approche basée sur le système d'exploitation pour gérer les modes repos des mémoires DRAM d'un système multi-tâches est développée. Cette approche opère uniquement lors des changements de contexte sans ajouter de pénalité sur la performance du système. Un modèle analytique exprimant la condition de mise en repos des bancs et un algorithme de sélection du mode repos le plus adéquat sont présentés. Cette étude manque d'expérimentations sur une plateforme réelle et des applications multi-tâches réelles ne sont pas considérées.

Cette vue globale combinée avec la flexibilité d'une approche logicielle permet d'obtenir de réelles économies d'énergie sans introduire de coût matériel supplémentaire. Le système d'exploitation à l'avantage de pouvoir anticiper la prochaine date de reprise d'exécution d'une tâche et ainsi de prédire le temps d'inactivité de chaque tâche. La granularité de l'ordre de la milliseconde permet de ne pas introduire de pénalité sur la performance mais en contre partie ne permet pas d'exploiter des périodes d'inactivité plus faibles entre différents changements de contexte.

2.1.2.4. Techniques hybrides logicielles et matérielles

Bien que les techniques logicielles aient généralement un plus faible impact sur les performances, les techniques matérielles permettent un gain énergétique plus important en remontant des informations en ligne à grain fin, qui sont généralement indisponibles au niveau système. La combinaison de ces deux techniques (figure 1.14) montre une meilleure efficacité dans la réduction de la consommation des mémoires multi-bancs disposant de plusieurs modes repos.

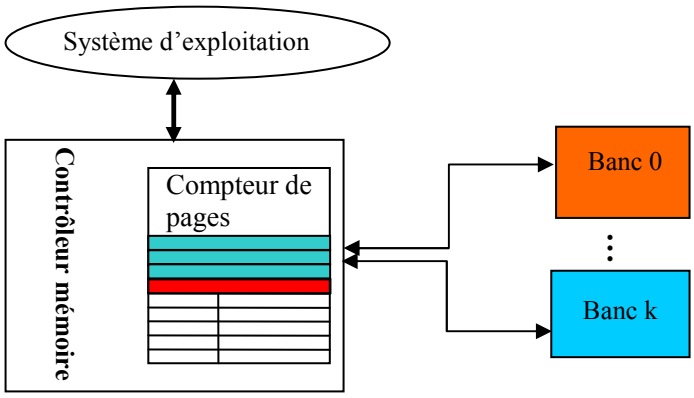


Figure 1.14. Collaboration du contrôleur mémoire matériel avec le système d'exploitation

Afin de réduire la consommation mémoire, Lebeck et al., combinent (Lebeck *et al*., 2000) des techniques de placement de pages : aléatoire, séquentielle ou basée sur fréquences d'accès dans les bancs mémoire avec des politiques matérielles statiques et

dynamiques de gestion des modes repos. L'allocation aléatoire place aléatoirement les pages dans les bancs mémoire. L'allocation séquentielle alloue les pages dans les bancs dans l'ordre de leur apparition dans le programme, remplissant ainsi complètement un banc avant de passer au suivant. Elle permet de rassembler les pages dans un nombre réduit de bancs. L'allocation basée sur la fréquence d'accès commence par placer la page la plus fréquemment accédée et continue en séquence jusqu'à la page la moins souvent accédée. Elle rassemble les données ayant des fréquences d'accès similaires dans un même banc. La politique matérielle de gestion des modes repos est soit statique soit dynamique. La politique statique manipule deux modes de fonctionnement, un mode actif et un mode repos qui peut être l'un des modes *Standby*, *Nap* ou *Power-Down*. La politique dynamique considère les quatre modes de fonctionnement, le mode actif et les trois modes repos (*Standby*, *Nap*, *Power-Down*). Des simulations basées sur des traces ou sur des exécutions ont été utilisées dans ces travaux.

Les résultats montrent que pour une allocation aléatoire avec la technique matérielle statique, le mode de repos *Nap* donne les meilleurs résultats et permet de réduire le produit *Energie*× *Latence* de 60 à 85% par rapport au cas où tous les bancs sont maintenus actifs. Ce type d'allocation employé avec une technique matérielle dynamique n'est pas très efficace. En effet, les seuils de basculement sont difficilement prédictibles et nécessitent l'ajout d'un matériel complexe. Pour une allocation de type séquentiel, la technique statique augmente de 10 à 30% le gain obtenu avec l'allocation précédente. Cependant, elle nécessite l'aide du système d'exploitation. La technique matérielle dynamique avec quatre modes repos donne un gain allant de 80 à 99% par rapport au cas où tous les bancs sont actifs et un gain de 6 à 55% par rapport au cas où un seul mode repos est utilisé. Cette technique nécessite la coopération entre le matériel et le logiciel. Pour une allocation de pages basée sur la fréquence d'accès une amélioration de 40 à 60% est obtenue par une allocation de pages aléatoire avec le mode repos *Nap*. Cependant cette approche ne considère que des applications mono-tâche.

Dans (Delaluz *et al.*, Novembre 2001), les auteurs combinent une approche matérielle et une approche logicielle pour gérer les modes repos et diminuer la consommation de la mémoire DRAM. L'approche matérielle emploie les trois techniques CTP, ATP et HBP introduites dans le paragraphe 2.1.2.1. Dans l'approche logicielle, le compilateur gère les modes repos ainsi que le regroupement et le placement de données dans les bancs. Des expérimentations montrent que la combinaison des deux approches est marginalement meilleure et montre parfois de mauvais résultats.

L'approche proposée dans (Fan *et al*., 2001) étudie des politiques de gestion des modes repos des mémoires multi-bancs dans des systèmes avec des mémoires caches. Les auteurs développent un modèle analytique qui approxime les périodes d'inactivités des bancs mémoires DRAM. Le modèle est validé par des simulations basées sur des traces d'exécutions pour deux techniques d'allocation de pages : placement aléatoire et placement basé sur la fréquence d'accès des pages. Les résultats prouvent qu'une simple politique de basculement immédiat d'un banc dans un mode faible consommation dès qu'il n'est plus accédé, est plus efficace que des politiques plus complexes qui prédisent les temps d'inactivité des bancs. Plus tard, ces mêmes auteurs emploient des réseaux de Petri stochastiques pour modéliser et évaluer des politiques complexes de gestions de modes repos (Fan *et al*., 2002). Ils modélisent des politiques de contrôle à deux modes puis à quatre modes de fonctionnement. Les résultats montrent que pour des politiques à deux modes de fonctionnement, celle utilisant le mode *Nap* présente des meilleurs gains d'énergie quand l'intervalle d'inactivité dépasse un certain seuil. Pour des politiques gérant quatre modes de fonctionnement, les bancs mémoires DRAM devraient

immédiatement basculer vers le mode *Standby*, puis vers le mode *Nap* ou *Power-Down* selon la valeur de la période d'inactivité, et ne jamais basculer vers le mode repos le plus profond *Power-Down* quand les intervalles entre des accès successifs suivent une distribution exponentielle.

2.1.3. Gestion des modes repos DRAM dans des systèmes multi-tâches

Dans un contexte multi-tâches, chaque tâche possède son propre profil d'accès mémoire différent des autres tâches, obligeant le contrôleur mémoire à mettre à jour à chaque changement de contexte, les seuils de basculement des modes repos. Ce phénomène entraîne une dégradation de performance et diminue le gain d'énergie effectif pouvant être obtenu. Ceci devient d'autant plus critique que le nombre de tâches est important et que la période de changement de contexte est courte. Une technique de gestion des modes repos relative à ce problème qui opère par coopération entre le logiciel et le matériel est présentée dans (Huang *et al*., 2004). Le contrôleur mémoire matériel reçoit des informations du système d'exploitation à chaque changement de contexte. Dans ce cas, le contrôleur mémoire sauvegarde et restore des registres internes qui sont utilisés pour garder une trace des profils d'accès précédents. Le contrôleur mémoire peut ainsi gérer les modes repos de la tâche en cours d'exécution en se basant sur son profil d'accès sauvegardé à l'issue des précédentes exécutions. Les mémoires DRAM visées sont de type DDR (Double Data Rate) constitué de modules ou DIMM. Chaque module contient 1, 2 ou 4 rangées (*rank)*. Chaque rangée peut être indépendamment mise dans un mode repos. La figure 1.8 présente les modes repos disponibles dans ce type mémoire. Cette technique réduit la consommation de 14 à 17% par rapport à une technique matérielle et de 16 à 26% par rapport à une technique logicielle. Un gain de plus de 70% est obtenu comparé à la consommation en l'absence de gestion de mode repos.

Exploitant le placement de pages mémoire basé sur la fréquence d'accès proposé initialement dans (Lebeck *et al*., 2000), Huang et al, (Huang *et al*., 2005) visent des systèmes multi-tâches, et présentent une technique qui modifie dynamiquement le trafic mémoire pour produire des périodes d'inactivité plus longues entre les accès afin de profiter des modes repos les plus profonds. Cette technique place les pages les plus fréquemment accédées dans une rangée appelée *hot rank* et celles les moins fréquemment utilisées dans une deuxième rangée appelée *cold rank*. Une collaboration entre le contrôleur matériel et le système d'exploitation est mise en place. Le contrôleur matériel comptabilise le nombre d'accès à chaque page qui est sauvegardé dans une table. Cette table, accessible par le système d'exploitation, est utilisée pour migrer les pages. L'allocation des pages dans le *hot rank* est effectuée par migration en ligne des pages les plus souvent accédées du *cold rank* vers le *hot rank*. Les expérimentations montrent que le changement actif du trafic mémoire permet d'améliorer de 35% le gain d'énergie par rapport à un placement statique. De manière évidente, pour les applications à faible taux d'accès mémoire, l'impact sur les performances est faible, alors qu'il est significatif pour les applications à fort accès mémoire.

Alors que tous les travaux cités précédemment ont considéré une seule technologie de réalisation de la mémoire principale, l'étude développée dans (Lee *et al*., Août 2003) considère une architecture mémoire hétérogène, constituée de mémoires SDRAM, NAND Flash et NOR Flash afin de bénéficier des avantages de chaque technologie mémoire. Les systèmes autonomes nécessitent généralement une mémoire non volatile à accès aléatoire pour le démarrage. La mémoire NOR Flash est ainsi utilisée pour garder une trace du code et copier son contenu dans la mémoire SDRAM lors du démarrage. Cette technologie de mémoire est avantageuse en temps d'accès et en consommation en

mode lecture, (par exemple, 33,6nJ/4words, 210ns/4words), contrairement le mode écriture est très couteux en temps et en énergie (par exemple, 464µs/16words, 107µJ/16words). La mémoire NAND Flash est généralement utilisée comme un second moyen de stockage qui coopère avec le premier niveau (SDRAM) pour les applications volumineuses en instructions et en données. Les mémoires NAND Flash sont très couteuses en consommation et en temps, (de l'ordre de 1,18uJ/512bytes, 35,8µs/512bytes) pour une lecture et (9,51µJ/512bytes, 226µs/512bytes) pour une écriture. Cependant, elles ont une grande capacité d'intégration et donc un coût par bit très faible par rapport aux mémoires synchrones SDRAM. Cette technologie mémoire nécessite aussi un code détecteur et correcteur d'erreurs. Les auteurs développent une heuristique d'allocation de tâches (instructions et données) aux différents types de mémoire afin de réduire l'énergie consommée dans la structure mémoire. Les instructions et les données d'une tâche peuvent être réunies dans une seule technologie de mémoire ou stockées séparément dans des technologies différentes. Les tâches sont caractérisées par un temps d'exécution, une taille d'instructions, une taille de données de l'utilisateur ainsi qu'un nombre de transactions mémoire, déterminé à partir d'une trace d'exécution. Une allocation statique modélisée avec l'heuristique du sac à dos est d'abord réalisée en considérant une taille fixe de chaque type de mémoire. Une allocation dynamique permet de changer dynamiquement l'allocation initiale par migration de tâches, lors d'un changement du profil d'utilisateur (temps d'exécution, taille des données). Le coût de migration est pris en compte. Des résultats obtenus par simulation montrent un gain en consommation pouvant atteindre 26% par rapport à une allocation classique.

Pour des applications multimédia multi-tâches à comportement dynamique, des études menées par plusieurs partenaires (IMEC, Université de Bologne d'Italie, Université d'Espagne) (Gomez *et al*., 2002), (Marchal *et al*., 2004) et (Marchal *et al*., 2005) ont proposé une solution au problème d'ordonnancement de tâches, couplé à l'assignation des structures de données dans une mémoire SDRAM multi-bancs avec pour but de réduire la consommation. La minimisation de la consommation dynamique des mémoires SDRAM est obtenue par réduction du nombre de défauts de pages dus à l'activation simultanée de plusieurs tâches dans une architecture multiprocesseur. Une heuristique divisée en deux phases, une phase hors ligne et une phase en ligne, est proposée. Pendant la phase hors ligne, un critère appelé *Selfishness* (Sds) est calculé pour chaque structure de données. Ce paramètre exprime l'intérêt énergétique de placer une structure de données seule dans un banc. Les structures ayant les plus grandes valeurs de Sds sont distribuées dans des bancs différents. Durant cette phase, différents scénarios d'exécution sont aussi identifiés par *profiling*. Utilisant le critère de *Selfishness*, un ensemble d'ordonnancements et d'assignations mémoire associées est déterminé pour chaque scénario avec pour objectif de réduire la consommation. Ces solutions forment un ensemble Pareto montrant un compromis entre le temps, l'énergie ainsi que le nombre de bancs. Ces solutions sont ensuite intégrées dans le système d'exploitation temps réel. En ligne, après identification du scénario en exécution, les informations stockées dans le système d'exploitation sont utilisées pour sélectionner un point de fonctionnement de la courbe de Pareto qui minimise l'énergie et respecte les échéances.

2.2. Mémoires caches multi-bancs pour réduire la consommation

Les mémoires internes, généralement de technologie SRAM, offrent un temps d'accès de l'ordre de 1 à 2 cycles processeur. Ces mémoires sont soit des mémoires caches gérées matériellement soit des mémoires dédiées ou *scratchpad* gérées logiciellement par le concepteur où a priori les données les plus fréquemment utilisées

sont stockées. Ces mémoires SRAM internes consomment une part non négligeable de l'énergie consommée par le circuit. Dans cette partie, nous présentons des approches de partitionnement en plusieurs modules (ou bancs) employées dans les mémoires cache SRAM pour diminuer leur consommation énergétique.

2.2.1. Réduction de la consommation dynamique

Une source majeure de consommation dans une mémoire cache à organisation conventionnelle est attribuée aux transitions d'états logiques dans les lignes correspondant à l'étiquette et aux données activées. La dissipation se produit lors de la charge ou la décharge des lignes de cache. Des expérimentations reportées dans (Ghose *et al*., 1999) montrent que la consommation des caches augmente avec la taille des lignes du fait de l'activation d'un nombre important d'amplificateurs de sortie lors d'une lecture ou d'une écriture. Les auteurs de (Villa *et al*., 2000) annoncent que les lignes de cache (figure 1.15) et les amplificateurs de sortie consomment en écriture environ 70% de la consommation totale du cache. Une solution architecturale a été proposée pour diminuer cette part de la consommation des caches. Elle consiste à diviser les cellules de mémorisation en plusieurs partitions pour réduire la consommation dynamique dissipée lors des accès. Ce partitionnent est soit horizontal, soit vertical soit à la fois horizontal et vertical.

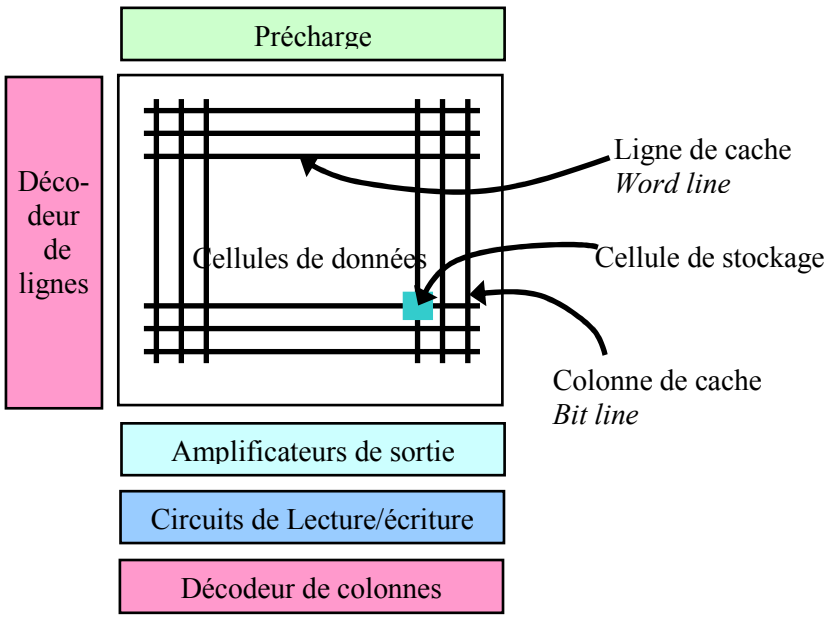


Figure 1.15. Organisation conventionnelle d'une mémoire cache

2.2.1.1 Partitionnement horizontal

Il s'agit de partitionner horizontalement les cellules mémoire en plusieurs bancs de façon que seul le banc en accès soit actif. La taille de la ligne de cache (*word line*) est réduite de même que le nombre de cellules mémoire actives (figure 1.16). Cette architecture permet d'améliorer la performance et de réduire la consommation. Dans

certains cas, les pénalités temporelles et de consommation associées à la circuiterie de décodage des bancs deviennent importantes.

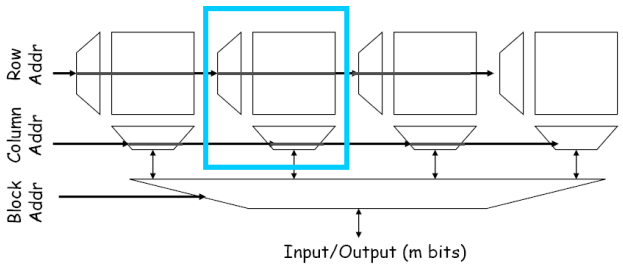


Figure 1.16. Structure de la mémoire cache partitionnée.

- **Cache *subbanking***

Pour réduire l'énergie dissipée dans la ligne de cache, les cellules mémoires réservées aux données sont divisées en plusieurs bancs de façon que seul le banc contenant la donnée demandée soit lu (Su *et al.*, 1995). Cette technique auparavant connue par les concepteurs de mémoires RAM comme le multiplexage de colonnes *column multiplexing* est utilisée pour réduire le nombre d'amplificateurs de sorties. Un banc consiste en un nombre consécutif de colonnes de cellules de données agencées en sous-bancs (figure 1.17). Ainsi une ligne de données est répartie sur plusieurs bancs. Puisqu'une donnée est lue à partir d'un seul banc, un ensemble commun d'amplificateurs de sortie est partagé par les sous-bancs. La taille d'un banc correspond à la largeur physique d'un mot. Chaque banc peut être activé indépendamment. En employant des bits d'étiquette pour indiquer la présence/absence des bancs dans le buffer de ligne, cette étape d'accès d'une donnée peut déterminer si un banc doit être lu pour la demande d'accès courante. Cette technique n'affecte pas le temps d'accès ni le pipeline du cache.

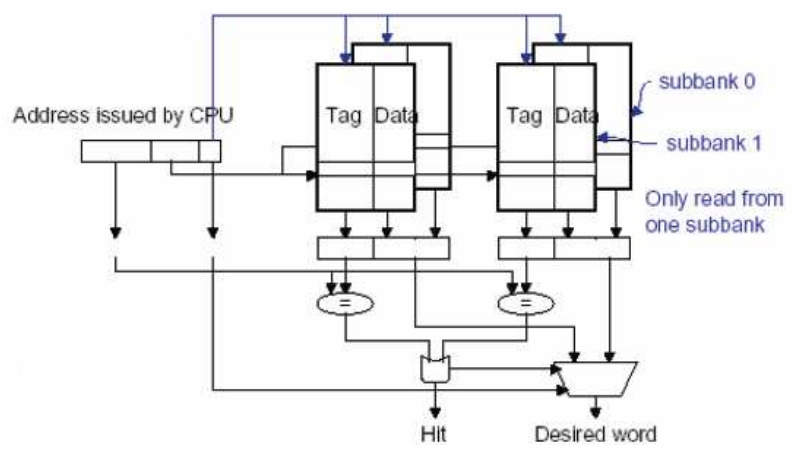


Figure 1.17. Organisation en plusieurs bancs d'une mémoire cache

Des expérimentations reportées dans (Ghose *et al.*, 1999) montrent que le gain d'énergie est d'autant plus important pour des lignes de cache de grande largeur. Par

exemple, pour une ligne de cache de 64 bytes, avec largeur de banc de 4 bytes, un gain de plus de 80% est obtenu pour un cache L1 d'instructions.

- **Mémoire cache de plusieurs régions**

La technique *Region-Based Caching* proposée par Lee et al. (Lee *et al*., 2000), est une autre implémentation du partitionnement horizontal. Cette technique exploite les différents types de données pour les allouer à trois modules différents du cache : un petit module pour les données de pile (*stack-cache*), un petit module pour les données globales (*global-cache*) et un large module pour les données restantes. Par exemple, un *stack-cache* de 4 KB à correspondance directe, un *global-cache* de 4 KB à correspondance directe et un cache principal de 32 KB à correspondance directe sont proposés dans l'article. Le module qui sera accédé est déterminé grâce à l'adresse délivrée par le processeur. Quand le *stack cache* ou le *global-cache* est activé, l'énergie d'accès dissipée est réduite suite à la faible taille de ces modules. Comparé à une organisation conventionnelle de cache, les auteurs reportent que 70% des références mémoires sont des succès dans ces modules.

2.2.1.2. Partitionnement vertical

Alors que le partitionnement horizontal répartit une ligne de cache sur plusieurs bancs en gardant une taille fixe de colonnes, le partitionnement vertical ou *Bit-line partitioning* répartit la colonne du cache en plusieurs modules construisant une hiérarchie mémoire. Cette technique est largement exploitée pour améliorer les performances et réduire la consommation. Les auteurs de (Ghose *et al*., 1999) évaluent l'effet du partitionnement vertical sur l'énergie consommée par le cache. L'emploi d'un cache L1 réduit l'énergie d'accès à la mémoire du fait d'une plus faible surface mémoire active. Le nombre d'accès à la mémoire principale est réduit par l'intégration d'un cache L1. De la même façon, l'ajout d'un petit cache de niveau 0 entre le processeur et le cache L1 peut réduire d'une manière significative l'énergie d'accès.

Les auteurs de (Kin *et al*., 1997) proposent un cache de faible taille de niveau L0 nommé *filter cache*. L'accès au niveau L1 du cache se produit uniquement lors d'un défaut de cache dans le *filter cache*. Les auteurs montrent que le *filter cache* réduit de 51% le produit énergie × temps d'accès sur un ensemble d'applications multimédia, comparé à une architecture conventionnelle de cache. Une autre étude suivant cette même approche est proposée dans (Bajwa *et al*., 1997), pour réduire l'énergie consommée dans un cache d'instructions.

L'efficacité du partitionnement vertical dépend largement du nombre de références mémoire pouvant être concentrées dans le niveau 0 de cache. Les auteurs de (Bellas *et al*., 1999) proposent une gestion dynamique du cache pour allouer les blocs d'instructions les plus utilisés dans le cache de niveau 0. Le module de prédiction de branchement intégré dans le cœur du processeur est exploité pour identifier ces blocs.

2.2.1.3. Partitionnement horizontal et vertical

Dans (Ko *et al*., 1998), les auteurs proposent une architecture cache de type MDM (*Multi-Divided Module*). Le cache est divisé horizontalement et verticalement en plusieurs modules. Chaque module de faible taille possède ses propres circuits de périphériques, de manière à ce qu'il fonctionne comme un cache indépendant et

autonome. Un seul module, désigné par l'adresse de référence mémoire est actif à chaque accès. Si le cache MDM possède M modules indépendants, la moyenne de capacité de charges devient de l'ordre 1/M, comparée à une organisation conventionnelle de cache.

Une autre technique similaire appelée sous-cache (*subcache*) a été proposée dans (Kim *et al*., 2001). Les auteurs réduisent l'énergie consommée par une division architecturale du cache en plusieurs sous-caches. Chaque sous-cache fonctionne comme un cache autonome, défini par une taille, une taille de ligne de cache et un degré d'associativité. Ces sous-caches peuvent être, soit identiques et le cache est dit homogène, soit différents (taille, associativité, ligne de cache) et le cache est dit hétérogène. Afin d'éviter le coût introduit par l'accès aux sous-caches non utilisés, seul le sous-cache contenant une donnée demandée est accédé. Le sous-cache le plus récemment utilisé (MRU) est prédit afin de servir la prochaine demande d'accès. Cette prédiction est renforcée par l'exploitation de la localité temporelle et spatiale.

Dans (Kim *et al*., Mai 2003), les auteurs s'intéressent à la mémoire cache d'instructions. Leurs expérimentations montrent que les accès à ce cache consomment en moyenne 70% de la consommation du système mémoire entier. Un accès à la mémoire cache d'instructions est effectué à chaque cycle processeur. Cette approche se caractérise par son efficacité à identifier le sous-cache contenant l'instruction demandée par le processeur. En effet, quand la prédiction échoue, un module sauvegardant l'identité du sous-cache contenant l'instruction permet d'éviter l'accès aux cellules de comparaison des étiquettes. Cette approche se différencie de celle proposée dans (Powell *et al*., 2001) qui accède à toutes les cellules du *tag* pour identifier le sous-cache, consommant ainsi une énergie supplémentaire. Un placement des pages dans les sous-caches ainsi qu'une réallocation dynamique de pages est aussi proposée. La page qui a causé le dernier défaut dans le sous-cache le plus sollicité est placée dans le sous-cache le moins sollicité. Plusieurs modules matériels sont nécessaires pour gérer l'ensemble des sous-caches (figure 1.18) ce qui rend cette solution difficile à implémenter.

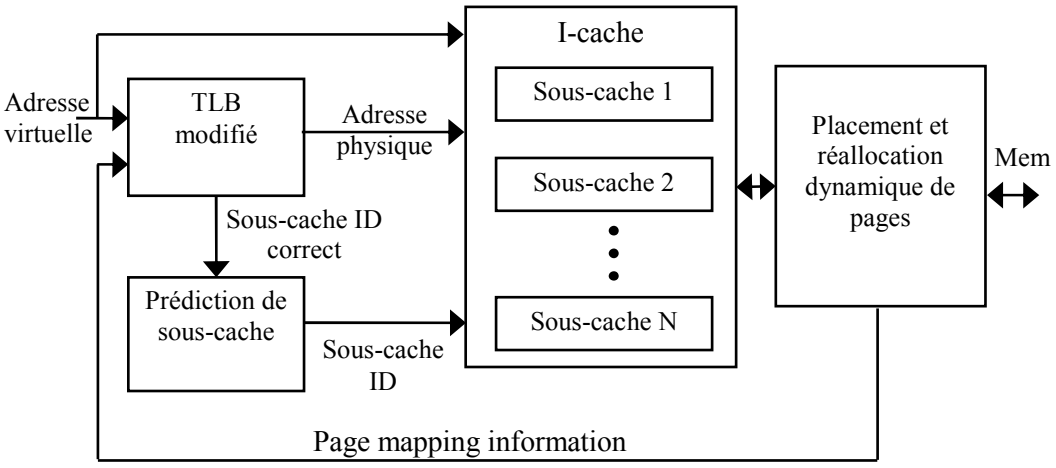


Figure 1.18. Architecture d'un cache d'instructions organisée en sous-caches

2.2.1.4. Activation sélective de voies dans un cache associatif

Dans un cache associatif par ensemble, toutes les voies sont activées à chaque accès. Pourtant uniquement une voie contient la donnée recherchée en cas de succès. Pour réduire la consommation gaspillée dans les voies non concernées, deux techniques ont été proposées. La première emploie un accès séquentiel (ou retardé) et la deuxième prédit a priori la voie à utiliser pour le prochain accès.

- **Accès aux données retardé**

Le cache L2 de processeur Alpha 21164's (Chandrakasan *et al*., 2001) ainsi que le cache par phase (*phased cache*) du processeur Hitachi SH3 (Hasegawa *et al*., 1995) emploient une technique qui accède uniquement aux bancs mémoire sélectionnés pour réduire la consommation. Au lieu d'accéder en parallèle aux cellules de comparaison des étiquettes et à celles des données (figure 1.19.a), l'accès aux cellules de données est retardé en permettant d'abord aux cellules de comparaison (*tag*) de sélectionner la voie appropriée. Si la ligne de l'étiquette indique un succès, elle est ensuite utilisée pour activer uniquement la voie qui contient la donnée référencée (figure 1.19.b). Contrairement, si la comparaison indique un défaut, un remplacement d'une ligne de cache sera alors effectué sans réaliser d'accès en lecture aux voies.

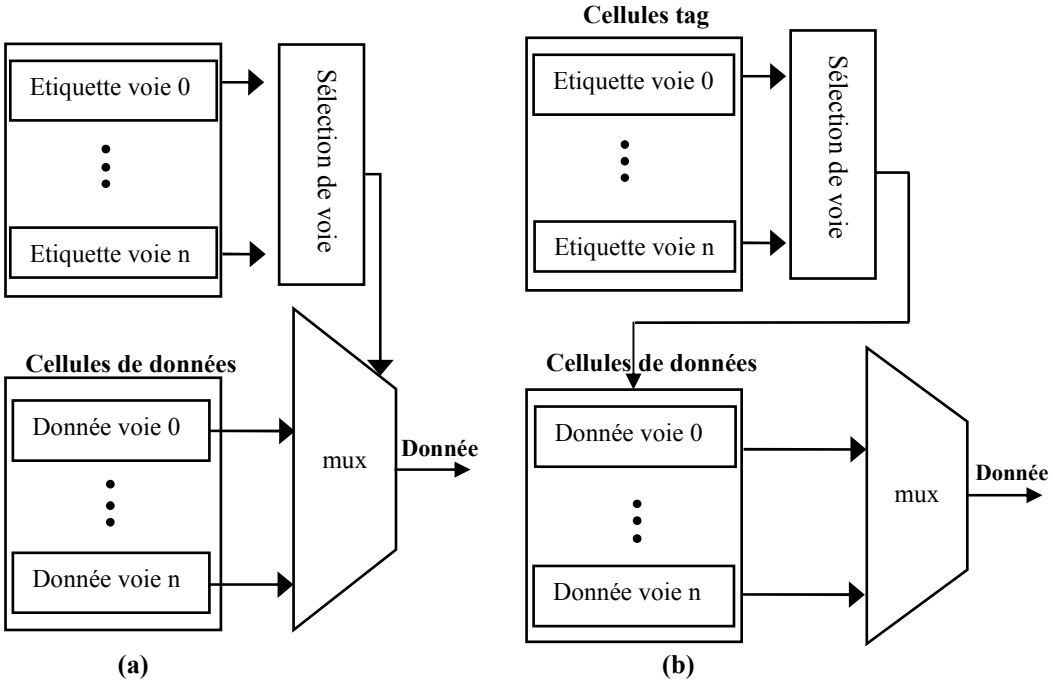


Figure 1.19. Deux types d'accès à une mémoire cache associative, (a) Accès parallèle conventionnel, (b) Accès séquentiel

Bien que l'accès séquentiel réduise la consommation dynamique des mémoires cache en permettant l'accès uniquement à la voie sélectionnée par l'étiquette, il induit une augmentation du temps d'accès mémoire et une dégradation de performance. Si la voie contenant la donnée référencée est connue avant de commencer l'accès au cache (par

prédiction par exemple), on peut anticiper l'accès à la voie et éliminer la pénalité sur le temps d'accès.

- **Prédiction de la voie**

Dans cette approche, la voie susceptible de contenir la donnée demandée est prédite. Plutôt que d'alimenter les cellules de comparaison des étiquettes comme dans le processeur Hitachi SH3, les auteurs de (Inoue *et al*., 1999) évaluent une approche appelée *way prediction* qui tente d'éviter l'activation des voies non utilisées dans un cache associatif. Cependant, cette technique introduit une pénalité sur la performance et nécessite l'accès aux autres voies quand la prédiction échoue. D'autres types d'accès séquentiels à la mémoire cache associative ont été proposés (Calder *et al*., 1996) (Huang *et al*., 2000) et peuvent être utilisés dans le cadre de l'optimisation de la consommation d'énergie.

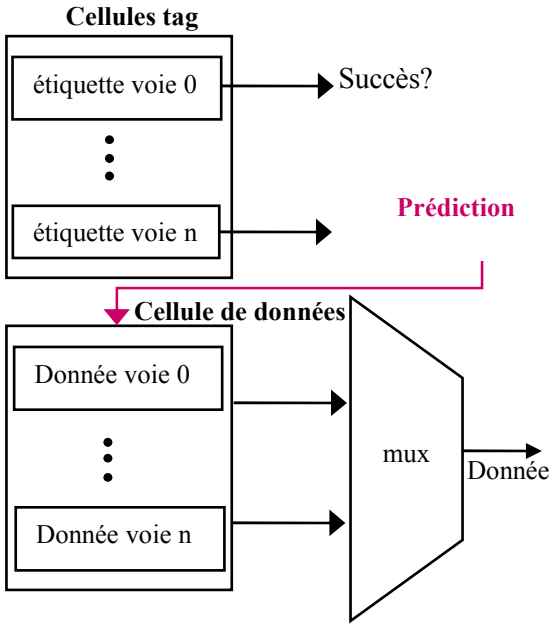


Figure 1.20. Accès basé sur une prédiction

Visant aussi la réduction de la consommation dynamique des caches associatifs, (Albonesi, 1999) développe une technique appelée *selective cache way*. Selon le comportement des applications, le nombre de voies activées dans une mémoire cache est ajusté en autorisant une dégradation de performance. Chaque voie est activée/désactivée par mise à jour d'un registre *(CWSR Cache Way Select Register)*.

Les deux approches *way-prediction* et *selective cache way* sont uniquement efficaces dans le cas de caches à forte associativité. Elles sont rapides mais ne sont pas assez précises. La technique *way-prediction* a été améliorée par Powell et al. (Powell *et al*., 200). Pour augmenter la précision de la prédiction, une politique *selective direct-mapping* est proposée. Les blocs qui ne sont pas en conflit sont placés dans un module à correspondance directe et les blocs en conflit utilisent un module associatif par ensemble. Ainsi, si un bloc est sans conflit, le module à correspondance directe est utilisé pour

identifier précisément l'emplacement des données. Cette politique montre une meilleure prédiction que la méthode *way-prediction*, mais elle nécessite un matériel plus complexe.

2.2.2. Réduction de la consommation statique

Le partitionnement du cache en plusieurs modules est largement exploité pour réduire la consommation dynamique. La consommation statique dans les caches devient de plus en plus importante avec l'évolution de la technologie. La limitation et la concentration de l'activité mémoire sur une partition à la fois, permettent d'appliquer plus facilement des techniques de réduction de la consommation statique. Dans ce paragraphe, nous détaillons quelques principes de partitionnement du cache en plusieurs modules pour réduire la partie statique de la consommation du cache.

2.2.2.1. DRI i-cache

La technique *DRI i-cache* introduite dans (Yang *et al*., 2000) est la première approche de réduction de la consommation statique au niveau architectural dans un cache d'instructions de niveau L1. Elle est basée sur des observations montrant que l'utilisation du cache L1 d'instructions varie largement avec les applications. Quand la taille du code nécessaire à une application change, *DRI i-cache* redimensionne dynamiquement la taille logique du cache et coupe l'alimentation des portions non utilisées du cache afin d'éliminer les courants statiques. Cependant cette technique est difficilement applicable au cache de données car généralement la performance du cache de données est très sensible à la variation de sa taille.

2.2.2.2. *Cache decay*

Une approche plus générale appelée *cache decay* pour réduire la consommation statique dans les caches est introduite dans (Kaxiras *et al*., 2001). Cette technique consiste à mettre hors tension (gated-Vdd) une ligne de cache non accédée pendant une période de temps. Les cellules mises hors tension perdent alors leurs contenus (*state-destroying*). Un accès à ces cellules nécessite une recharge des données du niveau hiérarchique supérieur donc une consommation dynamique supplémentaire. Alors que l'approche *i-cache* possède une granularité de plusieurs blocs contigus, la technique *cache decay* considère individuellement les blocs du cache. Cependant, la technique *cache decay* nécessite plusieurs modifications de la structure de cache pour être implémentée.

2.2.2.3. *Drowsy cache*

La technique *Drowsy cache* (Flautner *et al*., 2002) utilise l'ajustement en tension et fréquence DVS et consiste à mettre une ligne de cache en mode repos sous une tension plus faible. Les données des cellules mises sous une faible tension sont préservées (*state-preserving*), mais elles ne sont pas accessibles. Ces cellules ont besoin de plusieurs cycles pour être réveillées et retourner à la tension normale de fonctionnement. La réduction de la consommation statique du cache par cette technique n'est pas aussi agressive que d'autres techniques.

La combinaison du partitionnement de la mémoire cache et des techniques de réduction de la consommation statique est peu développée dans la littérature. De notre point de vue, cette combinaison constitue un axe de recherche important et prometteur dans la réduction de la consommation dynamique et statique des mémoires caches. Nous

présentons ici deux travaux qui ont combiné le partitionnement vertical et les techniques de réduction de consommation statique décrites dans ce même paragraphe.

Selon (Li *et al*., 2002), le cache L1 doit être de type *state-destroying* (importante réduction de consommation) parce que les pénalités des rechargements des données lors d'un défaut de cache du cache L2 sont faibles. Inversement, le cache L2 doit être plutôt en *state-preserving* parce qu'en cas de défaut, l'accès à la mémoire externe est plus coûteux en énergie dynamique et latence. Des lignes de cache LRU (*Least Recently Used*) sont mises hors tension, nécessitant ainsi l'ajout de compteurs dans l'architecture qui sont responsables d'une consommation dynamique et statique supplémentaire. Cette technique est employée généralement dans le cache L1. Toutes les lignes des cellules de stockage L2 sont en mode *drowsy* (tension plus faible).

Dans (Abella *et al*., 2003) une mémoire cache de données avec plusieurs bancs de caractéristiques différentes est étudiée. Les auteurs comparent plusieurs configurations de cache en consommation et en performance. Un modèle de base est constitué d'un cache L1 de données monolithique possédant 1 cycle de latence. Une deuxième configuration est constituée d'un cache L1 avec deux bancs implémentés avec des caractéristiques différentes. Le premier, nommé *Fast Cache* possède une latence de 1 cycle et est implémenté avec la même technologie que le modèle de base L1. Le deuxième banc possède une latence de 2 cycles, une tension d'alimentation V_{dd} plus basse et une tension de seuil V_{th} plus importante que celles du modèle de cache de base afin de diminuer la consommation dynamique et statique. En contre partie, le temps d'accès de ce banc *Slow cache* est plus élevé. Des expérimentations montrent que cette organisation de cache de données en plusieurs bancs mémoire, permet d'obtenir une performance similaire à un cache monolithique, en réduisant la consommation dynamique et statique.

2.3. Mémoires scratchpad multi-bancs pour réduire la consommation

Les mémoires internes dédiées multi-bancs (*scratchpad*) peuvent être employées pour augmenter les performances en autorisant des accès parallèles comme décrit dans le paragraphe 1.1 de cette deuxième partie. Ces mêmes mémoires peuvent aussi être employées pour réduire la consommation. Benini et al. (Benini *et al*., 2000), s'intéressent à la réduction de la consommation dans les mémoires dédiées internes SRAM. Ils proposent une méthode de partitionnement automatique en plusieurs bancs pour réduire la consommation dynamique. Chaque banc peut être accédé de manière indépendante. Selon le profil dynamique d'exécution d'une application s'exécutant sur un processeur embarqué, ils synthétisent une architecture mémoire multi-bancs optimale, adaptée au profil d'exécution. Une contrainte sur un nombre de bancs maximum à ne pas dépasser est spécifiée. La consommation dynamique est minimisée suite à la réduction de la taille des bancs, cependant les auteurs n'envisagent pas l'exploitation des modes repos des bancs pour diminuer l'énergie statique. Cette technique est conçue pour une seule application et non applicable quand il s'agit de système multi-tâches.

Dans (Wehmeyer *et al*., 2004), les auteurs s'intéressent aussi à une architecture mémoire *scratchpad* avec plusieurs partitions de tailles différentes pour réduire l'énergie mémoire. Ce type de mémoire avec plusieurs partitions de tailles différentes existe dans l'industrie. Par exemple, le processeur de la famille ARM9E possède deux mémoires configurables couplées appelées TCM (*Tightly Coupled Memories*) qui peuvent être employées soit comme *Smartcaches* soit comme *plain memory region* (ARM, 2003). Dans le dernier cas, les TCMs sont utilisées comme mémoires *scratchpad*. Le modèle de

l'architecture ARM implique qu'une mémoire des TCM est utilisée pour stocker uniquement les données et l'autre pour les instructions.

Les auteurs modélisent l'allocation des données dans les partitions, chacune avec une taille fixe est modélisée, sous la forme d'un problème ILP. L'algorithme d'allocation est intégré dans un compilateur qui utilise des informations obtenues par *profiling*. Un gain en énergie moyen de 22% est obtenu pour un ensemble de programmes test par rapport à une mémoire *scratchpad* monolithique.

3. Impact des mémoires multi-bancs sur la surface de silicium

Les architectures mémoire multi-bancs sont malheureusement responsables d'une augmentation de la surface de silicium surtout s'il s'agit de mémoires embarquées sur la même puce avec le processeur. Les auteurs de (Yamauchi *et al*., 1997) constatent qu'à capacité identique, une mémoire DRAM embarquée de 16 bancs occupe 1,4 fois plus de surface qu'une mémoire à 4 bancs. Une mémoire DRAM embarquée de 32 bancs est 1,8 fois plus coûteuse en surface que celle à 4 bancs pour une même taille totale. Cette pénalité de surface réduit considérablement la quantité de DRAM pouvant être embarquée avec le processeur. Afin d'obtenir une grande performance avec une pénalité de surface minimale, une architecture DRAM multi-bancs hiérarchique est proposée dans (Yamauchi *et al*., 1997). Dans cette architecture, les bancs mémoire sont divisés en bancs principaux et sous-bancs. Les bancs principaux sont identiques à des bancs conventionnels indépendants. Les sous-bancs sont implémentés dans un banc principal. Tous les sous-bancs appartenant à un banc principal partagent les circuits d'entrée/sortie, les décodeurs et les amplificateurs de sortie.

Exploitant le même intérêt des mémoires DRAM embarquées, les auteurs de (Fromm *et al*., 1997) montrent l'efficacité énergétique des architectures IRAM (Intelligent RAM), qui intègrent sur la même puce le processeur et la mémoire DRAM. Par rapport à une hiérarchie mémoire conventionnelle (avec cache L1 et L2), une réduction de la consommation 22% est obtenue par l'approche IRAM.

Dans (Shiue *et al*., 2004), les auteurs présentent une approche ILP et une heuristique qui déterminent la configuration mémoire multi-bancs et l'assignation des données qui minimisent : la surface sous une contrainte d'énergie, l'énergie sous une contrainte de surface ainsi que l'allocation des données qui minimise la consommation pour une configuration mémoire donnée (nombre de bancs, taille et nombre de ports par banc). Des expérimentations montrent une faible différence des résultats de l'heuristique par rapport à ceux obtenus par l'approche ILP.

Conclusion

Dans ce chapitre, une première partie a souligné les effets négatifs, liés à l'évolution de la technologie et des applications, sur les premières techniques de DVS. Ces techniques ont eu comme objectif de minimiser uniquement la consommation dynamique du processeur en négligeant sa consommation statique ainsi que la consommation mémoire (dynamique et statique). De nouvelles techniques de DVS tenant compte soit de la consommation statique du processeur soit du système mémoire ont été présentées.

Dans nos travaux, nous nous sommes particulièrement intéressés à l'augmentation de la consommation mémoire qui résulte en particulier de l'application du DVS sur le processeur. Le partitionnement en plusieurs bancs nous a paru intéressant comme solution pour réduire cet effet négatif surtout que technologiquement l'approche multi-bancs est de plus en plus utilisée. Dans une deuxième partie, des travaux adoptant le partitionnement en plusieurs bancs comme solution pour augmenter les performances des mémoires et permettre des accès parallèles (dans les SRAM et les DRAM) ont été présentés. Ensuite, des travaux basés sur cette même solution pour réduire la part de la consommation mémoire ont eux aussi été présentés.

Dans le chapitre suivant, nous présentons les modèles énergétiques développés au niveau système pour estimer la consommation du processeur et de sa mémoire principale multi-bancs. Cette estimation est un préalable nécessaire afin d'appliquer ensuite des techniques d'optimisation de l'énergie.

Chapitre 2. Modèles de consommation d'un processeur et de sa mémoire multi-bancs

Introduction

Dans ce chapitre, l'architecture cible ainsi que les applications visées sont décrites respectivement dans les paragraphes 1 et 2. Des modèles de consommation processeur, au niveau système, ainsi que les principes de la technique de DVS sont présentés dans le paragraphe 3. Dans le paragraphe 4, des modèles de consommation mémoire multi-bancs disposant de modes faible consommation sont développés.

Ces modèles de consommation processeur et mémoire permettront d'évaluer le bilan énergétique du système total muni de la technique de DVS. Ces modèles seront utilisés dans le chapitre 3 afin de choisir l'allocation des différentes tâches de l'application aux bancs qui minimise la consommation mémoire.

1. Description de l'architecture système

L'architecture ciblée dans notre approche est celle d'un système sur puce *SoC* principalement monoprocesseur. Des extensions au cas multiprocesseur sont proposées.

L'architecture monoprocesseur est composée d'une unité de traitement (processeur, DSP…), d'un cache de niveau 1 (L1) et d'une mémoire principale externe (DRAM, SDRAM, ...) monolithique ou multi-bancs (figure 2.1).

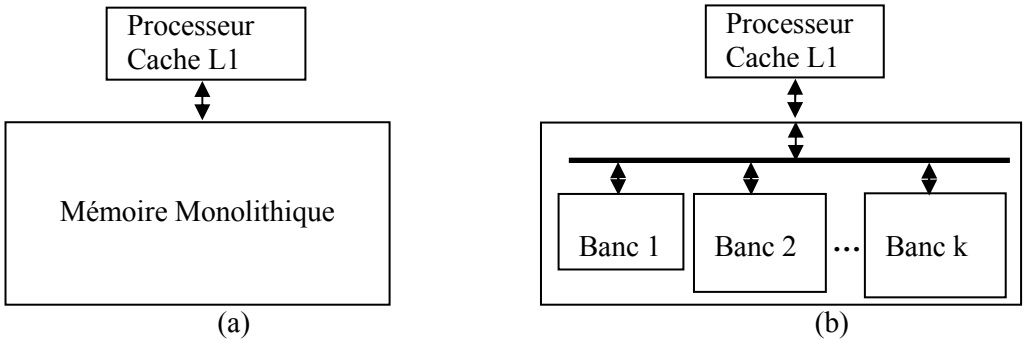


Figure 2.1. Architectures monoprocesseur, (a) architecture avec mémoire principale monolithique, (b) architecture avec mémoire multi-bancs non uniformes

L'architecture multiprocesseur inspirée des plateformes actuelles est composée de deux unités de traitement. Un processeur RISC maître et un processeur de traitement de signal DSP esclave. Chaque processeur possède ses mémoires internes (cache ou locale). La communication entre les deux processeurs s'effectue à l'aide d'une mémoire partagée et de contrôleurs DMA (*Direct Memory Access*). La mémoire principale externe est soit synchrone (DRAM, SDRAM, ...) soit asynchrone (Flash, EDA, …) soit encore une combinaison des deux. Dans notre étude, nous considérons des mémoires synchrones et en particulier des mémoires multi-bancs (figure 2.2). Comme exemple de ces architectures on cite la plateforme OMAP de Texas Instruments et la plateforme Nexperia de NXP.

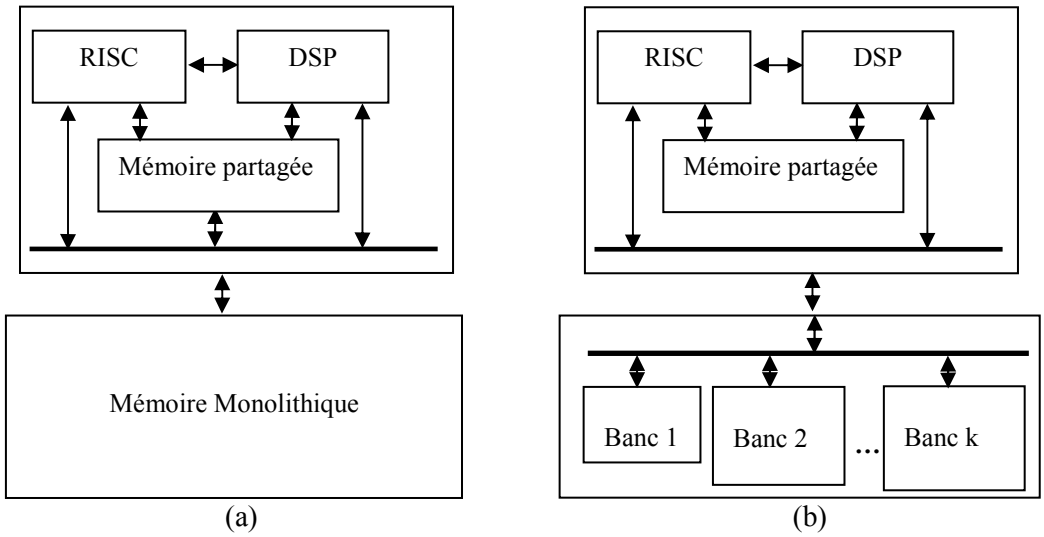


Figure 2.2. Architectures multiprocesseur, (a) architecture avec mémoire principale monolithique, (b) architecture avec mémoire multi-bancs non uniformes

Les bancs de la mémoire principale peuvent avoir des tailles différentes (architecture mémoire multi-bancs non uniformes) adaptées aux besoins. Bien que cette hypothèse complexifie la résolution du problème de configuration mémoire et d'allocation des tâches aux différents bancs, elle permet d'explorer un espace de solutions plus large afin de réduire au maximum la consommation mémoire.

Un exemple de mémoire DRAM multi-bancs avec plusieurs modes repos est la mémoire RDRAM de RAMBUS. Les valeurs d'énergies consommées dans chaque mode pendant un cycle mémoire sont données dans le tableau 2.1. Pour servir une demande d'accès en lecture ou en écriture, le banc mémoire doit être en mode actif, le mode le plus gourmand en énergie, sinon il peut être basculé vers l'un des modes faible consommation disponibles (*Standby, Nap, Power-Down*). Chaque mode faible consommation est caractérisé par une puissance dissipée et un temps de retour au mode actif (temps de resynchronisation). Plus l'énergie du mode basse consommation est faible (moins de ressources internes sont alimentées), plus le temps de resynchronisation est important (tableau 2.1).

Tableau 2.1. Energie consommée (par cycle mémoire) et pénalités de resynchronisation des différents modes de fonctionnement d'une mémoire RDRAM

Mode de fonctionnment	Energie consommée (nJ)	Temps de resynchronisation (cycles)	Energie de resynchronisation (nJ)
Active	3,57	0	0
Standby	0,83	2	1,2
Nap	0,32	30	10
Power-Down	0,005	9000	3800

2. Description des applications cibles

La classe d'applications considérées dans nos travaux regroupe des applications de traitement de signal à temps réel strict, généralement exécutées sur un système monoprocesseur et des applications multimédia à temps réel souple qui nécessitent des architectures multiprocesseur.

Ces applications peuvent souvent être décrites par un ensemble de N tâches périodiques, à échéances sur requête et synchrones. Chaque tâche T_i est caractérisée par des paramètres temporels correspondant à la période P_i, au temps d'exécution c_i et des paramètres non temporels tels que la taille S_{Ti} des instructions et des données et le nombre de défauts de cache L1 noté M_i.

Ces tâches sont ordonnancées suivant une méthode adaptée, en ligne ou hors ligne, avec ou sans préemption. Par exemple, la figure 2.3 présente l'ordonnancement d'un ensemble de tâches indépendantes décrit dans le tableau 2.2 sur une hyperpériode de 12 unités de temps selon la technique *Rate Monotonic* (RM) qui considère des tâches à priorité fixe et préemptives (Liu *et al*., 1973).

On note H l'hyperpériode de l'ensemble des N tâches : H = PPCM (P_1, …, P_N).

Tableau 2.2. Exemple de 3 tâches

Tâche	c_i	P_i
T_1	1	3
T_2	1	4
T_3	2	6

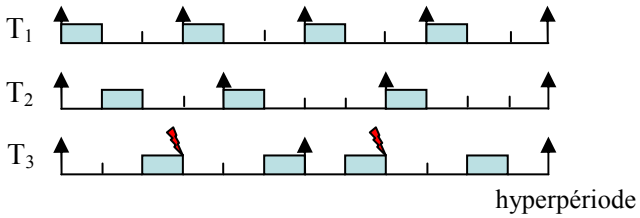


Figure 2.3. Ordonnancement RM de l'exemple à 3 tâches

3. Modèles de consommation processeur

Dans ce paragraphe, on s'intéresse à la contribution du processeur par rapport à l'énergie pour un ensemble de tâches. Les sources de consommation du processeur sont d'abord présentées dans le paragraphe 3.1 afin d'identifier les paramètres qui agissent de façon significative et d'analyser comment ces paramètres sont pris en compte par les techniques d'optimisation considérées. Dans le paragraphe 3.2, une approche statique

d'ajustement conjoint de la tension et la fréquence appliquée globalement à toutes les tâches durant l'hyperpériode est décrite. Dans le paragraphe 3.3, la même technique de DVS est présentée mais appliquée localement à chaque tâche si son échéance n'est pas encore atteinte. Ces techniques d'optimisation fournissent des niveaux d'optimisation différents de la consommation processeur. Quelque soit la technique utilisée, notre approche de configuration du système mémoire peut s'appliquer, ainsi nous n'avons pas considéré de techniques de DVS plus agressives telles que celles présentées dans le chapitre état de l'art.

3.1. Source de consommation du processeur

La consommation d'un cœur de processeur comprend deux composantes : dynamique et statique. La composante dynamique $P_{dynamique}$ est essentiellement due à l'activité de commutation :

$$P_{dynamique} \approx C_{equ} f_{no\min al} V_{dd}^2 \qquad [2.1]$$

où C_{equ} est la somme des capacités dans le circuit chargées et déchargées à chaque cycle, V_{dd} la tension d'alimentation et $f_{nominal}$ la fréquence nominale du processeur.

Pour une puissance consommée constante pendant un temps d'exécution t, l'énergie $E_{dynamique}$ consommée se calcule comme suit :

$$E_{dynamique} = P_{dynamique} \times t$$

L'énergie dynamique consommée est quadratiquement liée à la tension d'alimentation. Baisser le couple fréquence/tension de fonctionnement du processeur permet alors d'avoir un gain en énergie proportionnel au carré de la tension d'alimentation.

La composante statique $P_{statique}$ est due principalement à des courants de fuite :

$$P_{statique} \approx I_{fuite} V_{dd}$$

L'énergie $E_{statique}$ consommée pendant un temps d'exécution t, pour une puissance statique constante, est alors :

$$E_{statique} = P_{statique} \times t$$

L'énergie totale E_{totale} consommée est la somme de l'énergie dynamique et de celle statique :

$$E_{totale} = E_{dynamique} + E_{statique}$$

La consommation statique du processeur n'a pas été prise en compte car cette dernière reste négligeable pour les processeurs actuels. Cependant, des modèles de cette consommation en fonction de la tension et de la fréquence du processeur existent dans la littérature (Martin *et al*., 2002), (Jejurikar *et al*., Juin 2004). La part statique de la consommation processeur obtenue par ces modèles peut être ajoutée pour de futures

générations de processeurs. La consommation statique de la mémoire est elle prise en compte dans nos modèles.

3.2. Ajustement conjoint en tension et en fréquence (DVS) statique global

On prend ici l'exemple d'un ordonnancement de type *Rate Monotonic* (RM). La priorité d'une tâche est attribuée suivant l'ordre inverse des périodes. L'objectif est alors de déterminer la fréquence processeur la plus faible qui permet de respecter les échéances temporelles strictes des tâches. Le test de faisabilité suffisant mais non nécessaire de l'ordonnancement RM, permet de calculer un facteur d'étalement global α_{max} appliqué à toutes les tâches s'exécutant sur le processeur (Aydin *et al*., 2001). Ce facteur d'étalement consiste à exécuter les tâches à une fréquence plus basse que la fréquence nominale $f_{nominal}$ du processeur.

Pour N tâches synchrones et à échéances sur requête, le test de faisabilité de l'ordonnancement RM est :

$$U = \sum_{i=1}^{N} \frac{c_i}{P_i} \leq N(2^{1/N} - 1) \qquad [2.2]$$

Le facteur d'étalement maximal est calculé avec l'équation 2.3:

$$\alpha_{\max} = \frac{N(2^{\frac{1}{N}} - 1)}{U}; \quad \alpha_{\max} > 1 \qquad [2.3]$$

La nouvelle fréquence f_{DVS_G} du processeur est alors :

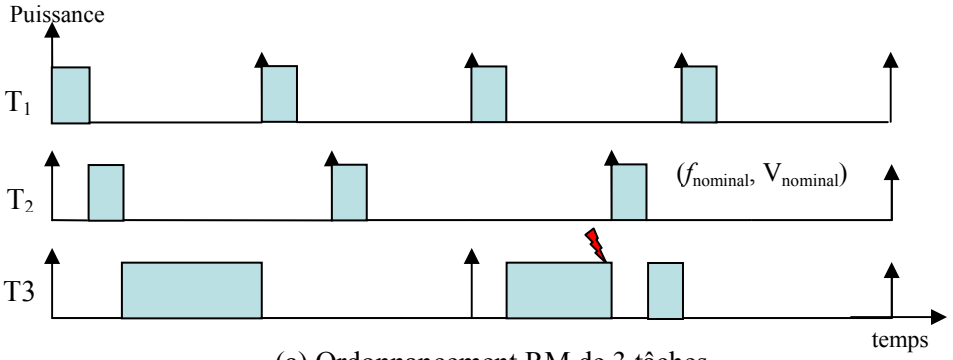
$$f_{DVS\_G} = \frac{f_{nominal}}{\alpha_{\max}}$$

Les temps d'exécution allongés des tâches sont obtenus par :

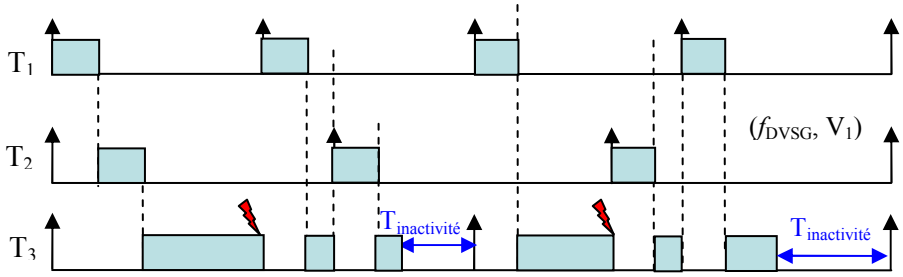
$$c_i^{'} = \alpha_{\max} c_i$$

3.3 Ajustement conjoint en tension et fréquence (DVS) statique local

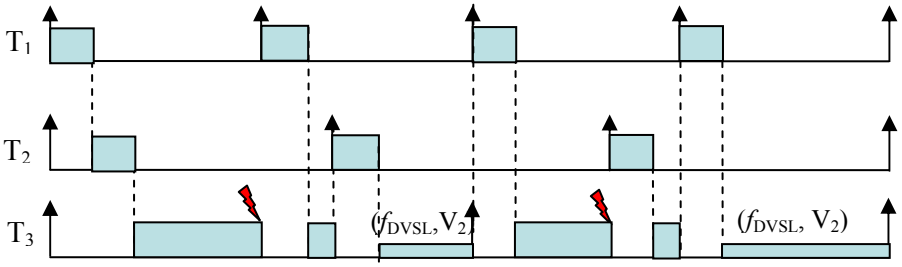
Le facteur d'étalement global α_{max} calculé à partir de l'équation 2.3 ne permet pas, généralement, d'exploiter toutes les périodes d'inactivité du processeur. Pour tirer profit de ces zones d'inactivité et réduire au mieux la fréquence processeur, l'ordonnancement obtenu après étalement est retouché localement au niveau de chaque tâche. Cette technique est connue en littérature sous le nom de *One Task Extension (OTE)* (Shin *et al*., 1999). Si une période d'inactivité ($T_{inactivité}$) du processeur suit l'exécution d'une tâche T_i, cette tâche peut être exécutée avec une nouvelle fréquence f_{DVSL_Ti} plus faible que celle trouvée précédemment (f_{DVS_G}).



(a) Ordonnancement RM de 3 tâches



(b) Ajustement global de l'ordonnancement



(c) Ajustement local de l'ordonnancement

Figure 2.4. Techniques d'optimisation de la consommation processeur : (b) DVS global, (c) DVS local et (d) modes faible consommation

La figure 2.4 illustre les différentes techniques d'optimisation processeur présentées dans les paragraphes précédents. Les caractéristiques des tâches sont données dans le tableau 2.3. L'ordonnancement selon la technique RM est illustré par la figure 2.4.a. La figure 2.4.b décrit l'ajustement global des tâches par un facteur α_{max}= 1,248. Les nouveaux temps d'exécution sont donnés dans la quatrième colonne du tableau 2.3. Une préemption supplémentaire de la tâche T_3 par la tâche T_2 apparaît suite au DVS global. Dans la figure 2.4.c, les tâches T_2 et T_3 sont ajustées localement. Les nouveau temps d'exécution sont donnés dans la cinquième colonne du tableau 2.3.

Tableau 2.3. Les temps d'exécution de 3 tâches après DVS global et local

Tâche	c_i	P_i	c'_i	c''_i
T_1	0,5	3	0,62	0,62
T_2	0,5	4	0,62	1,632
T_3	2	6	2,496	4,128

L'utilisation des modes repos du processeur permettent, lorsque l'ordonnancement le permet, de réduire encore la consommation, aussi bien dans la partie dynamique que statique. Comme indiqué précédemment, l'exploitation de ces modes peut tout à fait être envisagée dans notre approche car nous nous basons sur un résultat d'ordonnancement pour déterminer une configuration mémoire.

4. Modèles de consommation mémoire DRAM

Dans ce paragraphe, on s'intéresse à définir un modèle de consommation des mémoires principales DRAM. Ce modèle est utilisé dans notre approche pour évaluer la consommation des structures de mémoire construites. Dans le paragraphe 4.1, les sources de consommation mémoire sont présentées. Dans le paragraphe 4.2, les différents paramètres influant sur la consommation d'une mémoire multi-bancs sont identifiés ainsi que la variation de cette consommation en fonction de ces paramètres. Dans le paragraphe 4.3, un modèle énergétique de la consommation d'une mémoire multi-banc est développé. Ce modèle permet de quantifier la consommation mémoire pour un nombre de bancs et une allocation de tâches à ces bancs fixés. Le modèle est développé au niveau système où des gains d'énergie important sont attendus. Le niveau de granularité considéré correspond à l'ensemble des instructions et des données de chaque tâche. La tâche est donc l'entité de base de notre approche. On considère de plus qu'une tâche ne peut être stockée que dans un seul banc au plus. Ce modèle de mémoire tient compte aussi bien de la consommation dynamique due aux accès que de la consommation statique dissipée par les bancs en mode faible consommation.

4.1. Sources de consommation des mémoires

La puissance consommée par une mémoire est le produit de sa tension d'alimentation par le courant consommé :

$$P = V_{dd}.I_{dd}$$

Le courant I_{dd} est décrit par la formule suivante (Itoh *et al.*, 1995):

$$I_{dd} = m.I_{act} + m(n-1)I_{ret} + (n+m)\,C_{de}.V_{\text{int}}.f + C_{pt}.V_{\text{int}}.f + I_{dcp}$$

où m est nombre de colonnes, n est le nombre de lignes et V_{dd} est la tension d'alimentation externe. I_{act} est le courant effectif des cellules actives, I_{ret} est le courant de rafraîchissement des cellules inactives, C_{de} est la capacité de sortie de chaque décodeur,

V_{int} est la tension d'alimentation interne, C_{pt} est la capacité totale des périphériques, I_{dcp} est le courant statique consommé par la circuiterie des colonnes et des amplificateurs d'entrées/sorties.

Ce modèle de consommation mémoire ne s'applique pas au niveau de conception choisi mais il permet d'illustrer l'augmentation du courant I_{dd} avec le nombre de colonnes m et de lignes n de la mémoire. Cette constatation est un point très important dans notre étude et qui fait tout l'intérêt d'une architecture mémoire multi-bancs pour la réduction de l'énergie. Dans le paragraphe suivant, on montre que cette augmentation de la consommation mémoire en fonction de sa taille est quantifiée pour une mémoire de technologie RDRAM.

4.2. Paramètres influant sur la consommation mémoire

Définissons tout d'abord une fonction d'allocation φ qui associe à chaque tâche T_i appartenant à un ensemble fini de N tâches un banc b_j appartenant lui aussi à un ensemble fini de k bancs ($1 \leq k \leq N$).

$$\varphi: \{T_1,T_2,\ldots T_N\} \rightarrow \{b_1,b_2,\ldots b_k\}$$

$$\varphi(T_i) = b_j$$

4.2.1. Taille des bancs

La consommation mémoire augmente d'une façon monotone avec la taille de la mémoire. Les modèles analytiques développés dans (Itoh *et al.*, 1995) ainsi que les travaux de (Benini *et al.*, 2000) et de (Wehmeyer *et al.*, 2004) illustrent que la consommation mémoire augmente avec le nombre de lignes et de colonnes de la mémoire. Dans le cas des mémoires principales multi-bancs, plusieurs travaux considèrent que les énergies données dans le tableau 2.1 augmentent de $\tau_1 = 30\%$ quand la taille du banc est doublée (Ozturk *et al.*, 2005). Dans notre approche on considère que la taille d'un banc b_j est la somme des tailles S_{T_i} des tâches allouées à ce banc.

$$S_{b_j} = \sum_{T_i / \varphi(T_i)=b_j} S_{T_i}$$

Une architecture mémoire peut alors comporter des bancs de tailles non identiques (tailles des bancs non uniformes). Ainsi la variation des énergies par cycle mémoire (E_{Active}, $E_{Standby}$, E_{Nap}, $E_{Power\text{-}Down}$, $E_{resynchronisation}$) en fonction de la taille S_{b_j} du banc exprimée en kbyte est calculée comme suit :

$$E_x = E_{0x} \times (1+\tau_1)^{Log_2(\frac{S_{b_j}}{8})} \qquad [2.4]$$

où x prend sa valeur dans {*Active*, *Standby*, *Nap*, *Power-Down*, resynchronisation} et E_{0x} représente les valeurs initiales des énergies par cycle mémoire pour une taille de banc de 8 kB données par le tableau 2.1.

4.2.2. Nombre de bancs

La consommation des mémoires multi-bancs dépend aussi du nombre de bancs mémoire qui la compose. Pour un nombre de tâches fixé, quand un nouveau banc est ajouté à l'architecture mémoire, la taille des bancs décroît (moins de tâches par bancs) donc les énergies par cycle mémoire en mode *Active*, en mode faible consommation {*Standby*, *Nap*, *Power-Down*} et de resynchronisation diminuent. Cependant, la consommation sur le bus et plus précisément sur la logique de contrôle reliant les différents bancs augmente. Dans (Benini *et al*., 2000), les auteurs considèrent que cette augmentation est de $\tau_2 = 20$ % à chaque banc ajouté. Ainsi on établit l'équation 2.5 qui permet de calculer l'énergie de communication en fonction du nombre de bancs k.

$$E_{bus} = E_{0bus} \times (1 + \tau_2)^{k-1} \qquad [2.5]$$

L'énergie E_{0bus} représente la consommation sur le bus pour une architecture avec un seul banc mémoire.

Cette augmentation de consommation pourrait être éventuellement réduite en partitionnant le bus. Cependant, cela impliquerait d'associer un contrôleur à chaque banc mémoire ce qui va à l'encontre des architectures mémoire multi-bancs disponibles actuellement.

Dans notre approche nous considérons que les taux τ_1 et τ_2 sont ajustables selon la technologie et l'architecture mémoire.

4.2.3. Successivité et préemption entre tâches

On appelle successivité entre deux tâches T_i et T_j notée σ_{ij}, le fait que la tâche T_j commence (finit) son exécution juste après (avant) l'exécution de la tâche T_i ou lorsqu'une tâche T_j préempte une tâche T_i moins prioritaire. Les différentes valeurs de successivité entre N tâches sont représentées par une matrice S_N triangulaire supérieure de dimension N×N. La matrice de successivité est triangulaire supérieure du fait des deux relations :

$$\begin{cases} S_{ij} = S_{ji} \\ S_{ii} = 0 \end{cases}$$

L'exploitation de la successivité permet de diminuer le nombre de resynchronisations des bancs mémoire et donc d'allonger leurs périodes de repos. Le nombre d'activations d'un banc b_j est calculé par l'équation 2.6 :

$$N_{resynchronisation\_bj} = \sum_{T_i / \varphi(T_i) = b_j} N_{exeT_i} - \sum_{T_i, T_j / (\varphi(T_i), \varphi(T_j)) = (b_j, b_j)} \sigma_{ij} \qquad [2.6]$$

où N_{exeTi} égal à $\frac{H}{P_i}$ est le nombre d'exécutions de la tâche T_i durant l'hyperpériode H.

Par exemple, à partir de l'ordonnancement de la figure 2.3, les successivités entre les 3 tâches sont : $\sigma_{12} = 3$, $\sigma_{13} = 4$, $\sigma_{23} = 3$ et la matrice de successivité S_3 est :

$$S_3 = \begin{bmatrix} 0 & 3 & 4 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{bmatrix}$$

En considérant l'allocation des tâches de la figure 2.5, le nombre d'activations de chaque banc est :

$N_{resynchronisation\_b1} = N_{exeT1} + N_{exeT3} - \sigma_{13} = 4$

$N_{resynchronisation\_b2} = N_{exeT2} = 3$

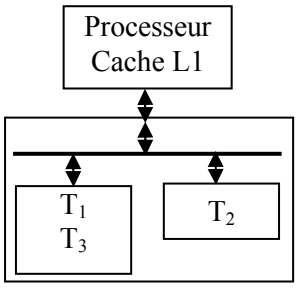


Figure 2.5. Allocation de 3 tâches sur 2 bancs

En conclusion, ajouter un banc à l'architecture mémoire, signifie :

- ✓ Des tailles de bancs plus faibles (moins de tâches par banc) donc des énergies par cycle mémoire (E_{active}, E_{Mode_repos}, $E_{resynchronisation}$) plus faibles (selon l'équation 2.4). La notation E_{Mode_repos} représente l'énergie associée à l'un des modes *Standby*, *Nap* ou *Power-Down* du tableau 2.1.
- ✓ Plus de bancs à mettre en mode basse consommation et plus de périodes de repos donc une énergie de repos plus importante.
- ✓ Une énergie, due à l'interconnexion entre les bancs, plus importante.
- ✓ Plus de bancs mémoire à réveiller, donc une énergie due à la resynchronisation plus importante.

Les paramètres identifiés dans ce paragraphe sont employés afin de développer un modèle de consommation mémoire multi-bancs décrit dans le paragraphe suivant.

4.3. Estimation de la consommation d'une mémoire principale multi-bancs

4.3.1. Modèle mono-tâche

Les auteurs de (Gomez *et al*., 2003) présentent un modèle de consommation d'une mémoire multi-bancs possédant plusieurs modes faible consommation. La consommation

totale est la somme des consommations dynamique et statique dissipées dans tous les bancs mémoire.

$$E_{mémoire} = \sum_{i=1}^{k} \left(E^{i}_{statique} + E^{i}_{dynamique} \right)$$

La consommation dynamique est calculée par l'équation suivante :

$$E^{i}_{dynamique} = N^{i}_{pa} E_{pa} + N^{i}_{rw} E_{rw}$$

Avec E_{pa} est l'énergie d'une précharge/activation, E_{rw} est l'énergie pour une lecture/écriture.
N^{i}_{pa} représente le nombre de précharges/activations dans le banc i et N^{i}_{rw} représente le nombre de lectures /écritures dans le banc i.

La consommation statique est donnée par l'équation suivante :

$$E^{i}_{statique} = P_{cs} t^{i}_{cs} + P_{stby} t^{i}_{pwdn} + P_{pwdn} t^{i}_{pwdn}$$

P_{cs}, P_{stby}, P_{pwdn} sont respectivement les puissances dissipées dans les modes *clock suspend*, *Standby* et *Power-Down* par un banc.

t^{i}_{cs}, t^{i}_{stby}, t^{i}_{pwdn} sont respectivement les temps où le banc i est mis en mode *clock suspend*, *Standby* et *Power-Down*.

Les valeurs de N^{i}_{rw}, N^{i}_{pa}, t^{i}_{cs}, t^{i}_{stbye}, t^{i}_{pwdn} sont obtenues par simulation avec un outil développé par la société Micron. Une mémoire multi-bancs *Mobile* SDRAM de Micron (Micron, 2006) est expérimentée dans ces travaux.
Ce modèle correspond à des applications mono-tâche et à un niveau données/instructions.

4.3.2. Modèle complet pour des applications multi-tâches

Nous avons reconsidéré le modèle ci-dessus afin de représenter l'énergie consommée par une mémoire pour des applications multi-tâches et où la granularité considérée est l'ensemble des données et des instructions d'une tâche.

L'ensemble des paramètres utilisés dans le modèle sont définis dans le tableau 2.4.

Tableau 2.4. Paramètres du modèle

Paramètres du modèle	Définition
Paramètres architecturaux	
$E_{0accès}$	Valeur initiale de l'énergie d'accès par cycle mémoire
$E_{0nonaccès}$	Valeur initiale de l'énergie de non accès par cycle mémoire
E_{0Mode_repos}	Valeur initiale de l'énergie en mode repos par cycle mémoire
$E_{0resynchronisation}$	Valeur initiale de l'énergie de resynchronisation par cycle mémoire
E_{0bus}	Energie consommée sur le bus pour un nombre de bancs égal à un
$E_{changement\_contexte}$	Energie dissipée lors de la sauvegarde du contexte
$t_{accèsM}$	Temps d'un accès à la mémoire principale
$f_{mémoire}$	Fréquence de la mémoire principale
$f_{processeur}$	Fréquence du processeur
Paramètres de l'application	
M_i	Nombre d'accès à la mémoire principale de la tâche T_i
$N_{cycles\_accès\_Ti}$	Nombre d'accès en cycles mémoire de la tâche T_i
$N_{cycles\_nonaccès\_Ti}$	Nombre de non accès en cycles mémoire de la tâche T_i
$N_{cycles\_Mode\_repos\_bj}$	Nombre de cycles mémoire, le banc b_j est en mode faible consommation.
Paramètres de l'ordonnancement	
$N_{resynchronisation\_bj}$	Nombre de resynchronisations du banc b_j (équation 2.6)
S_{ij}	Successivité entre les tâches T_i et T_j
N_{exeTi}	Nombre d'exécutions de la tâche T_i durant l'hyperpériode
$N_{préemptions}$	Nombre de préemptions des tâches durant l'hyperpériode

L'énergie consommée par une architecture mémoire comportant k bancs munie d'une allocation φ de N tâches à ces bancs est décomposée en plusieurs énergies comme décrit dans l'équation 2.7.

$$E_{mémoire} = E_{accès} + E_{nonaccès} + E_{Mode\_repos} + E_{resynchron\ isation} + E_{préemption} + E_{bus} \quad [2.7]$$

La figure 2.6 représente ces différentes énergies consommées par une mémoire à 2 bancs avec l'allocation des tâches décrite sur la figure 2.5.

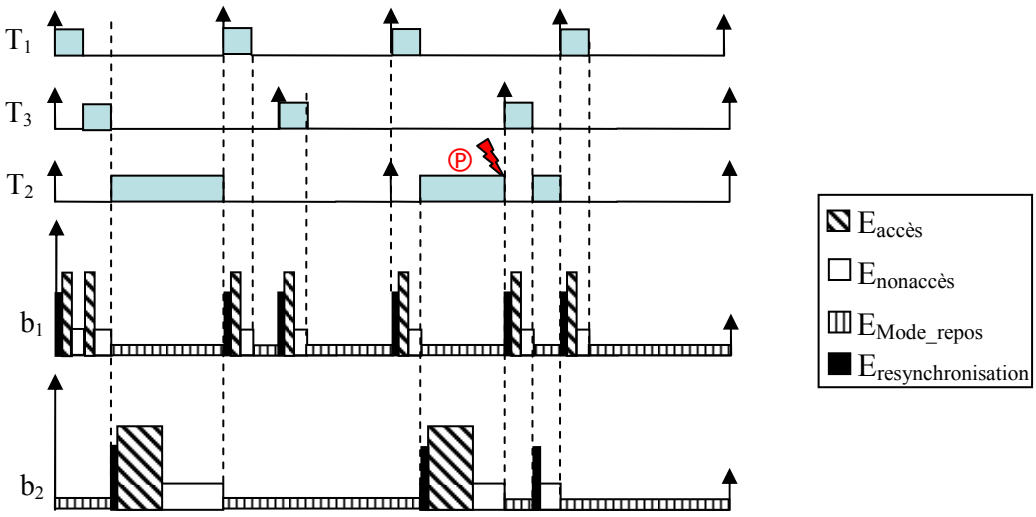


Figure 2.6. Les différentes énergies consommées pour un exemple d'allocation

Au début de l'exécution de la tâche T_1 le banc b_1 passe en mode actif (resynchronisation). Cette tâche effectue des accès en lecture et/ou écriture au banc (partie hachurée). L'exécution de certaines instructions par le processeur ne provoque pas d'accès au banc (cache *hit* par exemple). Cependant la mémoire reste en mode actif mais n'est pas accédé ($E_{nonaccès}$). Pendant ce temps, le banc b_2 est en mode repos. A partir de la séquence d'exécution des tâches, il est ainsi aisé de déterminer la succession des états de chaque banc.

On note que sur la figure 2.6, l'énergie d'accès est représentée au début de chaque tâche par soucis de clarté. Dans la réalité, les accès sont distribués tout au long de l'exécution de la tâche mais d'un point de vue énergétique le résultat est le même. L'énergie mémoire totale est donc la somme des différentes énergies consommées dans les deux bancs mémoire.

L'énergie $E_{accès}$ consommée lors des accès en lecture ou en écriture aux k bancs mémoire est :

$$E_{accès} = \sum_{b_j / j=1}^{k} \left(\sum_{T_i / \varphi(T_i)=b_j}^{N} N_{cycles\_accès\_T_i} \times E_{0accès} \times (1+\tau_1)^{Log_2(\frac{S_{b_j}}{8})} \right)$$

Le nombre d'accès $N_{cycles\_accès\_Ti}$ en cycles mémoire de la tâche T_i au banc mémoire est défini par :

$$N_{cycles\_accès\_Ti} = M_i \times t_{accèsM} \times f_{mémoire}$$

L'énergie $E_{nonaccèss}$ intervient quand les bancs mémoire sont actifs mais sans être accédés en lecture ou en écriture. Cette énergie est due essentiellement à la co-activation des bancs mémoire avec l'exécution des tâches allouées à ces bancs.
Contrairement à (Ozturk *et al.*, 2005), le mode *Active* décrit dans le tableau 2.1 est décomposé en deux modes différents: mode lecture/écriture (accès) et mode actif sans lecture/écriture (non-accès). Ce choix est fait pour mieux tenir compte de l'augmentation de $E_{nonaccès}$, suite à l'allongement des temps d'exécution des tâches lors de l'utilisation du DVS, alors que $E_{accès}$ reste constante.

$$E_{nonaccès} = \sum_{b_j / j=1}^{k} \left(\sum_{T_i / \varphi(T_i)=b_j}^{N} N_{cycles\_nonaccès\_T_i} \times E_{0nonaccès} \times (1+\tau_1)^{Log_2(\frac{S_{b_j}}{8})} \right)$$

Le nombre de cycles mémoire $N_{cycles\_nonaccès\_Ti}$ de la tâche T_i où le banc mémoire est actif mais sans accès est :

$$N_{cycles\_nonaccès\_T_i} = (c_i \times \frac{1}{f_{processeur}} - M_i \times t_{accèsM}) \times f_{mémoire}$$

L'énergie E_{Mode_repos} consommée par les bancs mémoire quand ils sont dans un mode faible consommation est :

$$E_{Mode\_repos} = \sum_{b_j / j=1}^{k} N_{cycles\_Mode\_repos\_b_j} \times E_{0Mode\_repos} \times (1+\tau_1)^{Log_2(\frac{S_{b_j}}{8})}$$

Le nombre de cycles mémoire $N_{cycles\_Mode\_repos\_bj}$ où le banc b_j est en mode faible consommation est déterminé par :

$$\mathrm{N}_{\mathrm{cycle\_Mode\_repos\_bj}} = \left(H - \sum_{Ti / \varphi(Ti)=b_j}^{N} N_{exeTi} \times c_i \right) \times \frac{f_{mémoire}}{f_{processeur}}$$

L'énergie $E_{resynchronisation}$ relative aux transitions du banc mémoire du mode faible consommation au mode actif pour servir un accès mémoire :

$$E_{resynchronisation} = \sum_{b_j / j=1}^{k} N_{resynchronisation\_b_j} \times E_{0resynchronisation} \times (1+\tau_1)^{Log_2(\frac{S_{b_j}}{8})}$$

Le nombre de resynchronisations d'un banc b_j $N_{resynchronisation\_bj}$ est déterminé par l'équation 2.6.

L'énergie $E_{préemption}$ correspond à la sauvegarde de contexte due aux préemptions entre les tâches :

$$E_{préemption} = N_{préemptions} \times E_{changement\_contexte}$$

Le terme $E_{changement\_contexte}$ représente l'énergie dissipée lors de la sauvegarde du contexte et est considérée comme une constante dans notre modèle.

L'énergie E_{bus} consommée dans la connexion des bancs mémoire est :

$$E_{bus} = E_{0bus} \times (1 + \tau_2)^{k-1}$$

Les paramètres $N_{cycles\_accès\_Ti}$, $N_{cycles\_nonaccès\_Ti}$ sont dépendants de l'application car déduits du nombre d'accès M_i des tâches à la mémoire principale.

L'évaluation de l'énergie d'une configuration mémoire implique le calcul des différentes énergies et des différents paramètres présentés ci-dessus. Une complexité en $O(N^2)$ est alors constatée pour cette fonction d'évaluation d'énergie mémoire.

Conclusion

Le modèle multi-bancs proposé montre que de nombreux paramètres interviennent dans l'estimation de la consommation. Ce modèle laisse entrevoir qu'il n'est sans doute pas aisé de définir une structure mémoire (nombre de bancs, taille des bancs) et une allocation efficace d'un point de vue énergétique en liaison avec une gestion d'énergie par DVS. On remarque en effet, que la fréquence du processeur est un des paramètres qui intervient dans le modèle énergétique de la mémoire.

Dans le chapitre suivant, nous nous intéressons à définir une méthode de conception et d'optimisation de la structure mémoire multi-bancs qui utilise ce modèle afin d'estimer la consommation des solutions proposées.

Chapitre 3. Configuration mémoire multi-bancs minimisant l'énergie

Introduction

Dans ce chapitre, on présente dans une première partie la méthode utilisée pour analyser l'interaction entre la technique de DVS employée pour réduire la consommation processeur et la consommation de la mémoire principale monolithique. Les premières observations montrent que la consommation mémoire augmente avec la diminution de la tension et de la fréquence du processeur. En effet, cette réduction de fréquence oblige la mémoire à rester active plus longtemps en parallèle avec le processeur. Cet effet négatif sur la mémoire diminue globalement l'efficacité de la technique de DVS à réduire la consommation dans un SoC. Une solution proposée dans notre étude est de réduire la surface active d'une mémoire totale monolithique à une seule partition contenant les tâches en cours d'exécution par le processeur. L'objectif de ce type d'architecture mémoire multi-bancs est donc qu'un seul banc est maintenu actif à la fois, les autres sont en mode faible consommation.

La question qui se pose en choisissant ce type d'architecture mémoire est : comment allouer les différentes tâches du système aux bancs et quelle doit être la configuration mémoire correspondante (nombre de bancs et tailles de chaque banc) afin d'obtenir une consommation mémoire minimale? L'objectif est que la consommation mémoire couplée à celle du processeur soit globalement optimisée lorsque des techniques de DVS sont employées côté processeur.

La deuxième partie du chapitre aborde ce problème d'allocation de tâches et présente deux méthodes de résolution. La première est une exploration exhaustive de tout l'espace de solutions permettant d'obtenir la solution optimale. La deuxième est une heuristique capable d'explorer un sous-espace potentiellement intéressant et de résoudre le problème en un temps polynomial.

Le modèle d'application adopté est un modèle multi-tâches préemptif. Ce modèle nécessite généralement un système d'exploitation temps réel (RTOS) pour gérer les différentes tâches. Le code ainsi que les données du RTOS résident forcément dans un banc mémoire qui est activé à chaque fois qu'il y a une opération système à effectuer. La troisième partie de ce chapitre s'intéresse ainsi à la modélisation et à l'allocation du RTOS aux bancs mémoire.

1. Etude du comportement énergétique du couple (processeur, mémoire)

Ce paragraphe vise à étudier l'influence de la technique de DVS sur la consommation mémoire. Il faut souligner que l'objectif de cette étude n'est pas de proposer une nouvelle technique de DVS mais uniquement de montrer que les techniques de DVS classiques sont à l'origine de l'augmentation de la consommation mémoire et de justifier nos choix d'adopter une architecture mémoire multi-bancs pour contenir cette augmentation de la consommation mémoire liée au DVS. Pour illustrer le comportement énergétique global (processeur, mémoire) quand la tension et la fréquence sont ajustées sur le processeur, nous considérons une technique simple de DVS appliquée à un ordonnancement *Rate Monotonic* (RM). Un facteur d'étalement maximal α_{max} est appliqué d'une façon statique et globale sur toutes les tâches pendant l'hyperpériode. Ce facteur est calculé en respectant les échéances des tâches à l'aide du test de faisabilité des ordonnancements à priorités fixes (RM). Le passage à un ordonnancement dynamique (EDF par exemple) peut aisément être considéré.

Pour l'ordonnancement RM, le facteur d'ajustement maximum α_{max} réduisant la consommation et respectant les échéances est calculé dans le chapitre 2 selon l'équation 2.3.

Afin d'analyser l'évolution de la consommation du processeur et de la mémoire principale en fonction du DVS, le facteur d'ajustement appliqué au processeur varie de 1 (tension et fréquence nominales) à sa valeur maximale α_{max}. Cette analyse est effectuée pour deux architectures différentes. La première comprend un processeur avec une mémoire principale monolithique. La deuxième comprend un processeur avec une architecture mémoire multi-bancs.

1.1. Ajustement conjoint de la tension et fréquence avec une mémoire monolithique

La mémoire monolithique contient les codes et les données des différentes tâches de l'application. La consommation processeur est évaluée suivant l'équation 2.1 du chapitre 2. La consommation mémoire est évaluée à l'aide des modèles établis dans la section 4.3.2 chapitre 2 en prenant k égal à 1.

1.1.1. Augmentation de la co-activité de la mémoire en fonction du temps d'exécution processeur

La figure 3.1 présente la consommation processeur et mémoire lors de l'exécution d'une tâche T_i à la fréquence et à la tension nominales de fonctionnement. Durant l'exécution de cette tâche T_i par le processeur, la mémoire est en mode *Active*. Ce mode est séparé en deux sous-modes dans notre étude. Un sous-mode *accès* le plus consommateur où la mémoire sert les demandes d'accès en lecture et/ou écriture. Un deuxième sous-mode *nonaccès* où la mémoire est active mais ne sert aucun accès en lecture et/ou écriture. Sur la figure 3.1, les accès sont regroupés au début d'exécution de la tâche pour raison de clarté.

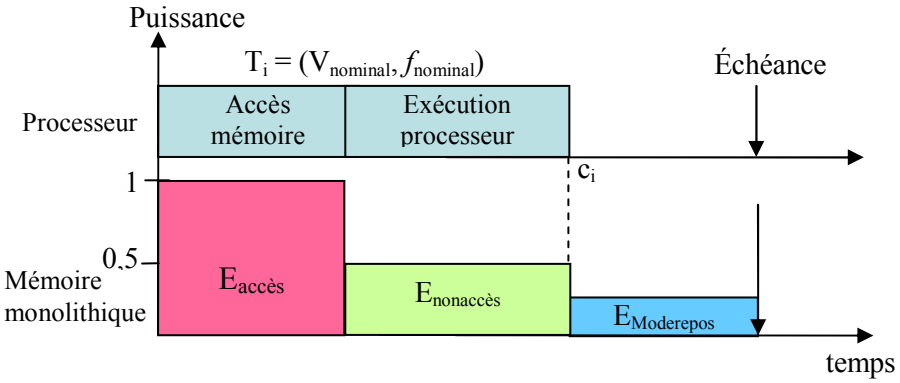


Figure 3.1. La contribution des différentes énergies lors de l'exécution d'une tâche T_i

L'application d'un facteur d'étalement α sur le processeur, ne modifie pas le temps ni l'énergie d'accès de la mémoire. En effet, quelque soit la valeur de α, les tâches génèrent toujours le même nombre d'accès et la mémoire est caractérisée par la même énergie par accès (la taille mémoire est fixe). Cependant, la durée d'activation de la mémoire sans servir de demandes d'accès devient plus large avec l'augmentation du temps d'exécution de la tâche (figure 3.2). Une énergie de non-accès $E_{nonaccès}$ plus importante est alors dissipée par la mémoire monolithique lors de l'utilisation du DVS sur le processeur.

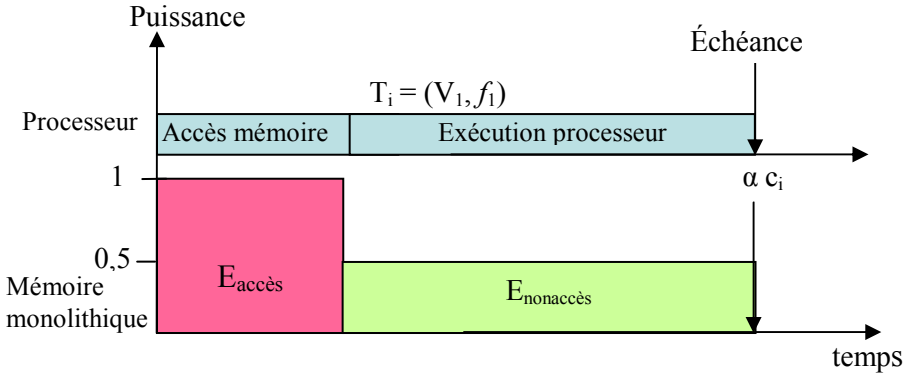


Figure 3.2. La contribution des différentes énergies lors du rallongement du temps d'exécution de la tâche T_i par un facteur α pour une mémoire monolithique

1.1.2. Augmentation du nombre de préemptions

Une deuxième conséquence de l'ajustement conjoint de la tension et de la fréquence est l'augmentation du nombre de préemptions des tâches dans les ordonnancements à priorités. Suite aux temps d'exécution plus longs des tâches lors de l'utilisation du DVS, les tâches les moins prioritaires sont plus souvent préemptées par les tâches plus prioritaires. Ces préemptions causent des changements de contexte plus fréquents. Les changements de contextes nécessitent des accès supplémentaires à la mémoire principale pour la sauvegarde et la restauration des contextes des tâches préemptées. Par conséquent, la consommation de la mémoire principale augmente du fait des accès supplémentaires causés par les préemptions.

La figure 3.3.a présente l'ordonnancement RM d'un ensemble de 3 tâches décrit dans le tableau 3.1.

Tableau 3.1. Caractéristiques d'un ensemble de 3 tâches

Tâche	c_i	P_i	$c'_i = \alpha_{max}\, c_i$
T_1	0,5	3	0,624
T_2	0,5	4	0,624
T_3	2	6	2,496

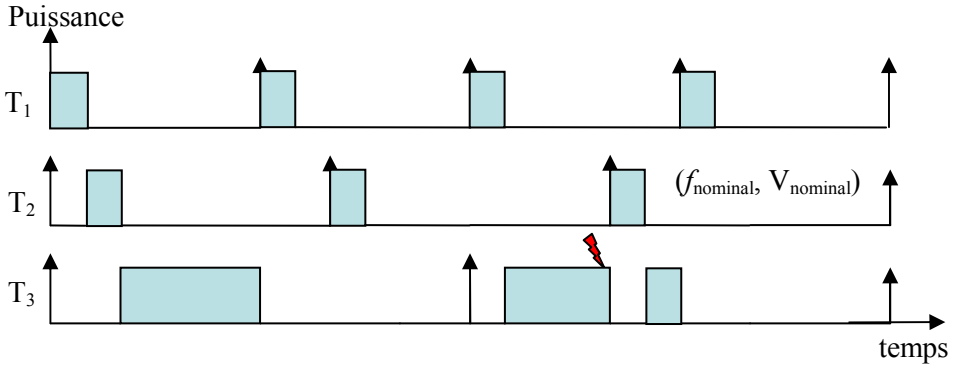


Figure 3.3.a. Ordonnancement RM de 3 tâches

En utilisant le test de faisabilité de l'ordonnancement RM pour ces trois tâches, un facteur d'étalement α_{max} = 1,248 est obtenu. Les nouveaux temps d'exécutions des tâches sont donnés dans la quatrième colonne du tableau 3.1. Trois préemptions supplémentaires de la tâche la moins prioritaire T_3 se produisent (figure 3.3.b). La première se produit à l'instant t=3 à cause de la tâche T_1 la plus prioritaire, la deuxième à l'instant t=4 suite à la tâche T_2 et enfin le troisième à l'instant t=9 suite à la tâche T_1. Cette préemption cause un certain nombre d'accès supplémentaires à la mémoire principale augmentant ainsi sa consommation.

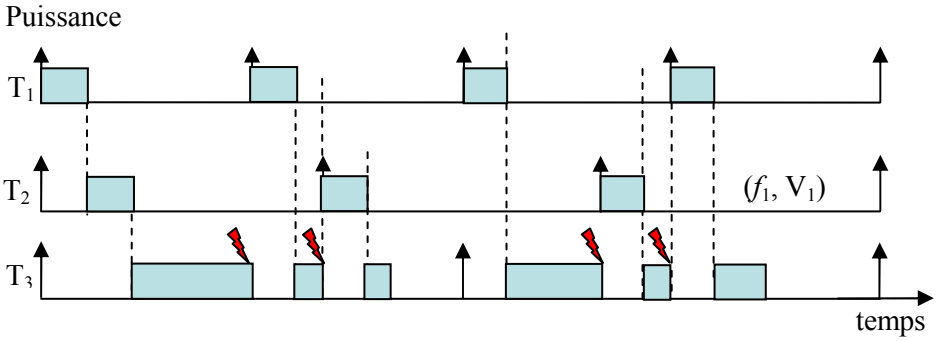


Figure 3.3.b. Ajustement global de l'ordonnancement

Afin d'améliorer le gain énergétique visé par la technique de DVS, il est nécessaire de minimiser les deux effets négatifs causés par le DVS sur la mémoire principale et illustrés ci-dessus.

Pour réduire le nombre de préemptions causé dans les ordonnancements à priorité, des travaux ont été récemment publiés. Quelques un sont déjà présentés dans le paragraphe 2.2.1 du chapitre 1.

Dans nos travaux, on s'est plutôt intéressé à réduire la surface mémoire coactive avec le processeur lors de l'étalement des temps d'exécution des tâches. Dans le paragraphe suivant une étude détaillée explique la réduction de la consommation par la combinaison de la technique DVS avec une mémoire multi-bancs.

1.2. Ajustement conjoint de la tension et de la fréquence avec une mémoire multi-bancs

Par rapport à une mémoire monolithique, une architecture mémoire multi-bancs permet de diminuer la surface mémoire active à uniquement un seul banc contenant la tâche en exécution par le processeur. Les autres bancs dont les tâches ne sont pas en cours d'exécution seront en mode repos.

Par exemple, en adoptant une architecture mémoire à deux bancs décrite dans la figure 3.4 et l'ordonnancement de la figure 3.3.a, le banc b_1 est actif uniquement lors de l'exécution de la tâche T_1 et T_2 par le processeur, ailleurs il est en mode faible consommation. De même pour le banc b_2 qui est actif uniquement lors de l'exécution de la tâche T_3 par le processeur. En appliquant la technique de DVS, un seul banc est alors actif à la fois, réduisant ainsi la surface mémoire active.

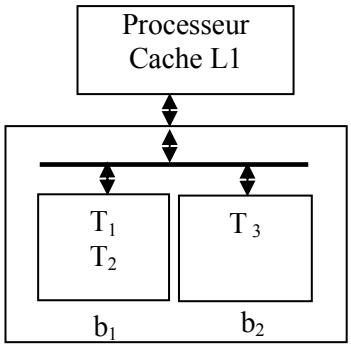


Figure 3.4. Exemple d'architecture mémoire à deux bancs

On considère pour simplifier dans cet exemple que la taille du banc b_1 est la moitié de la taille de la mémoire monolithique considérée dans le paragraphe 1.1. Selon l'équation 2.4 du chapitre 2 qui permet de calculer les énergies consommées par la mémoire en fonction de la taille du banc, les valeurs des énergies $E_{accès}$ et $E_{nonaccès}$ consommées pendant un cycle mémoire par une mémoire monolithique ($E_{accès}$= 1 et $E_{nonaccès}$= 0,5 sur la figure 3.2) sont divisées par un facteur 1,3 pour le banc b_1 soit $E_{accès}$= 0,769 et $E_{nonaccès}$= 0,384 (figure 3.5).

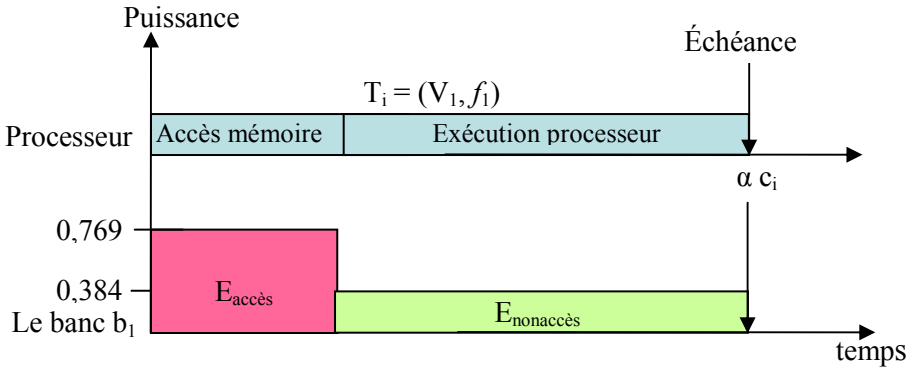


Figure 3.5. La contribution des différentes énergies lors du rallongement du temps d'exécution de la tâche T_i par un facteur α pour une mémoire multi-bancs

Cet exemple illustre qu'un problème couplant l'allocation de tâches aux bancs mémoire et la configuration mémoire (le nombre de bancs, la taille des bancs) apparaît pour minimiser tout d'abord la consommation mémoire et également pour augmenter le gain énergétique du système complet (processeur et mémoire) lorsque la technique de DVS est appliquée au processeur. Dans le paragraphe suivant, on détaille l'approche développée pour trouver cette allocation des tâches et la configuration mémoire correspondante qui minimisent la consommation mémoire.

2. Allocation de tâches et configuration mémoire multi-bancs minimisant la consommation

La modélisation de l'énergie consommée par une mémoire multi-bancs développée dans le chapitre 2, a permis d'identifier un nombre important de variables ou de paramètres dans le modèle ainsi que leurs fortes dépendances. Cette complexité rend le problème difficile à résoudre au premier abord. Notre approche a consisté premièrement à effectuer une exploration exhaustive de tout l'espace de solutions dans le but d'obtenir la solution optimale et d'évaluer l'influence des variations de chaque paramètre sur la consommation totale de la mémoire afin de construire par la suite une heuristique moins complexe et donc plus rapide. La figure 3.6 présente une vue globale de l'approche développée.

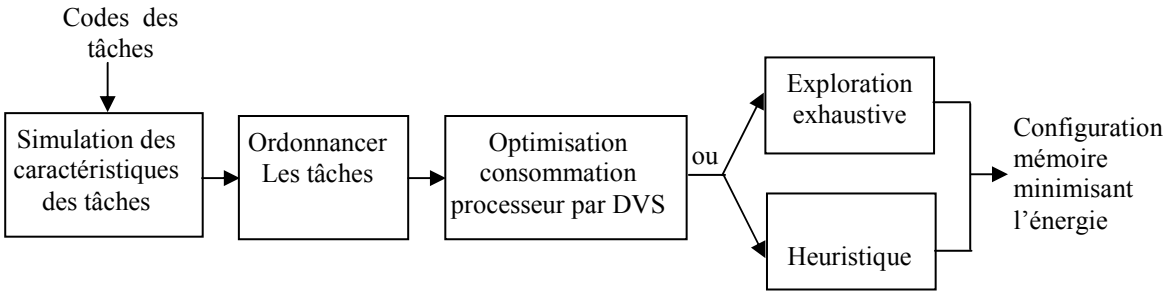


Figure 3.6. Vue globale de notre approche

2.1. Caractérisation du système et hypothèses

Les deux approches (exhaustive et heuristique) se basent sur l'ordonnancement des différentes tâches considérées. Cependant elles sont indépendantes de la politique d'ordonnancement (RM, EDF, List scheduling …) choisie. Il s'agit d'extraire, à partir d'un ordonnancement quelconque, certains paramètres comme les successivités entre les tâches, le nombre d'exécutions de chaque tâche et le nombre de préemptions, paramètres qui seront utiles pour évaluer la consommation mémoire.

- **Caractérisation des tâches**

Afin de spécifier chaque tâche en temps d'exécution c_i, taille S_{Ti} et nombre de défauts de cache M_i, l'outil de simulation architectural SimpleScalar (Burger *et al*., 1997) a été utilisé.

Les applications ciblées dans notre étude sont des systèmes multi-tâches. Dans de tels systèmes, en plus des défauts de cache compulsifs, c'est-à-dire liés à la première exécution de la tâche et les défauts de cache intrinsèques i.e. qui sont inhérents à la tâche, un troisième type extrinsèque dû à l'interaction des différentes tâches de l'application se produit. Ce type de défauts de cache se produit quand deux blocs mémoire appartenant à deux tâches différentes sont en conflit sur le même emplacement en cache (ligne de cache).

Visant des systèmes multi-tâches, deux principaux travaux (Agarwal *et al*., 1989) et (Thiebaut *et al*., 1987) ont modélisé analytiquement les différents types de défauts de cache qui se produisent dans de tels systèmes. Ces modèles sont stochastiques et emploient plusieurs paramètres extraits d'une trace d'exécution. Ces modèles restent complexes (plusieurs paramètres ne sont pas toujours disponibles) et difficilement applicables dans notre cas.

Une technique pratique et plus simple se basant sur des simulations avec l'outil de simulation SimpleScalar a été adoptée. Comme cet outil est principalement conçu pour la simulation d'architectures de processeurs dans le cas d'applications mono-tâche, une modification a été introduite pour simuler des applications multi-tâches. Une tâche est définie comme une fonction. Chaque instance de tâches dans l'ordonnancement considéré est un appel de fonction. L'outil Dlite! de SimpleScalar est utilisé pour insérer des points de tests à chaque changement de contexte du système (début et fin d'exécution, préemptions d'une instance de tâche) et pour collecter le nombre de défauts de cache entre deux points de test consécutifs. Pour chaque tâche, une moyenne de défauts de cache de toutes ses instances est retenue. Par exemple, l'ordonnancement de la figure 3.7 montre une moyenne de défauts de cache M_1 de 262 pour la tâche T_1, qui est bien inférieure aux 859 défauts de cache de la première instance de T_1.

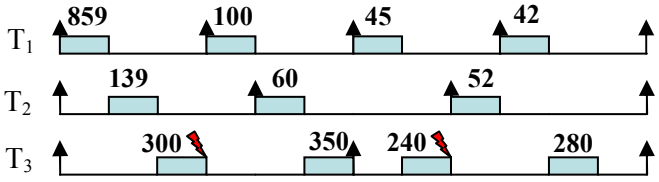


Figure 3.7. Les défauts de cache pour un système multi-tâches

- **Hypothèses**

En adoptant une architecture mémoire multi-bancs, les deux composantes (dynamique et statique) de la consommation sont réduites. La composante dynamique est réduite par la réduction des énergies d'accès, de resynchronisation et de mode repos qui dépendent de la taille des bancs. La composante statique est réduite par la mise en mode faible consommation des bancs inactifs.

Trois hypothèses ont été adoptées pour simplifier les problèmes de l'allocation des tâches aux bancs mémoire et de la définition d'une configuration mémoire en nombre et tailles de bancs afin de minimiser la consommation énergétique totale de la mémoire. La première hypothèse consiste à considérer que le changement d'état d'un banc vers un mode faible consommation s'effectue en un temps nul. Cette simplification permet de comparer les consommations intrinsèques des solutions de configuration mémoire. Des propositions de gestion des modes ont été présentées dans le chapitre deux. La deuxième hypothèse consiste à considérer uniquement deux modes de fonctionnement : un mode normal de fonctionnement *Active* et un seul mode repos. Cette hypothèse a été adoptée en se basant sur plusieurs travaux illustrant les meilleurs résultats obtenus avec une telle hypothèse (Lebeck *et al*., 2000) (Fan *et al*., 2001). La dernière hypothèse considère qu'une tâche est entièrement stockée dans un seul banc mémoire et possède des adresses en mémoire physique continues et successives. Cette hypothèse est conforme aux réalisations de nombreux systèmes embarqués.

2.2. Présentation des différentes étapes de la méthode

La figure 3.8 détaille les principales étapes suivies dans notre approche. L'algorithme utilisé est soit un algorithme exhaustif permettant de trouver la solution optimale soit une heuristique. Cet algorithme possède 3 entrées. La première renferme les caractéristiques des différentes tâches de l'application obtenues avec l'outil de simulation SimpleScalar. La deuxième entrée intègre les caractéristiques des différentes unités de l'architecture : processeur (fréquence, puissance (*f*, V)), cache L1 (fréquence, latence, énergie d'accès en cas de succès E_{hit}, énergie dissipée en cas d'échec E_{miss}) et mémoire principale (fréquence, temps d'accès, $E_{0accès}$, $E_{0nonaccès}$, E_{0mode_repos}, $E_{0resynchronisation}$, E_{0bus}, $E_{changement\_contexte}$, τ_1, τ_2). La troisième entrée correspond aux différents modèles de consommation processeur et mémoire principale développés et présentés dans le chapitre deux afin d'évaluer la consommation de la ou des solutions calculées par l'algorithme.

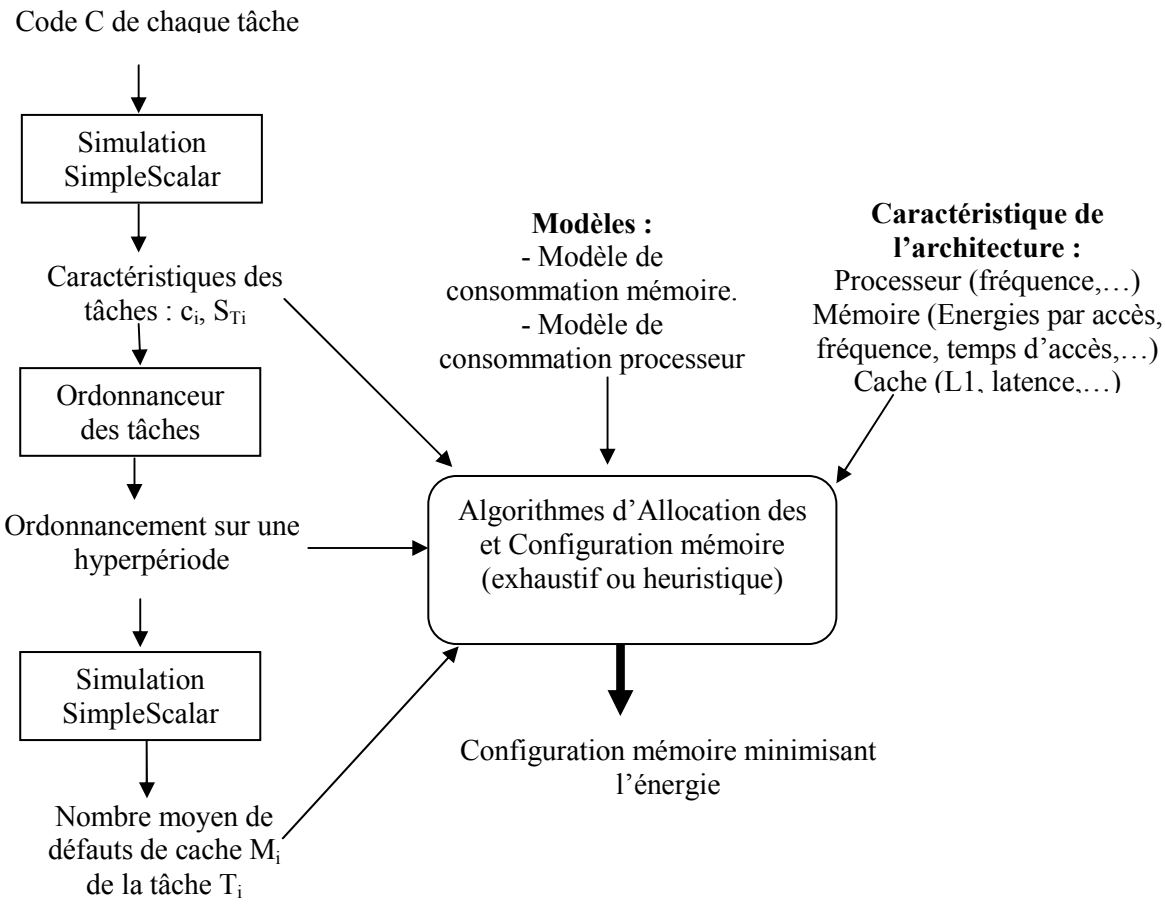


Figure 3.8. Méthodologie d'exploration de la configuration mémoire

Dans les deux paragraphes suivants, on détaille dans un premier temps l'algorithme exhaustif et ensuite l'heuristique proposée.

2.3. Exploration exhaustive

La nature même du problème d'allocation et l'interdépendance des différents paramètres influant sur la consommation mémoire, comme expliqué dans le chapitre 2, rendent ce problème NP-complet (Farrahi *et al*., 1998) (Mace, 1987). Une approche exhaustive qui explore tout l'espace des solutions a été adoptée dans un premier temps. Ce type de résolution a l'avantage de fournir la solution optimale ainsi que de suivre la variation des différentes énergies en fonction de chaque paramètre et d'évaluer l'impact de chaque paramètre sur la consommation totale de la mémoire.

2.3.1. Principe

L'algorithme exhaustif développé permet de trouver toutes les possibilités de ranger les N tâches dans k bancs, avec k variant de 1 à N. En effet, on suppose qu'au minimum un seul banc contient toutes les tâches (cette configuration correspond à l'appellation mémoire monolithique k = 1). Au maximum chaque tâche de l'application est allouée séparément dans un banc, le nombre de bancs mémoire est alors égal au nombre de tâches k = N.

Le nombre total de configurations mémoire pour N tâches correspond au nombre de Bell qui est une somme des nombres de Stirling du second ordre S(N,k) (Bell, 1934).

Le nombre de Stirling S(N,k) représente le nombre de configurations mémoire de N tâches dans k bancs décrit par l'équation :

$$S(N,k) = \frac{1}{k!}\sum_{j=1}^{k}(-1)^{k-j}\binom{k}{j}j^{N}$$

Le nombre total de solutions est donc la somme des nombres de Stirling pour un nombre de bancs k variant de 1 à N :

$$B_N = \sum_{k=1}^{N} S(N,k)$$

L'algorithme exhaustif développé consiste à construire les B_N configurations possibles. Une fois que toutes ces configurations sont identifiées, la consommation mémoire pour chaque configuration est évaluée à l'aide des modèles de consommation développés dans le chapitre 2.

Il est alors aisé d'identifier la solution optimale : l'allocation optimale $\varphi_{optimale}$ des tâches aux bancs, le nombre de bancs optimal k_{opimal} et les tailles optimales des bancs $S_{bj\ optimal}$, j = 1 à $k_{optimal}$.

2.3.2. Exemple illustratif

Considérons par exemple 4 tâches et déterminons toutes les configurations mémoire permettant de ranger les 4 tâches dans un nombre de bancs variant de 1 à 4.

Pour un nombre de bancs égal à 1, une seule allocation est possible, elle consiste à mettre toutes les tâches dans un même banc.

Pour un nombre de bancs égal à deux, il existe 2 possibilités de rangement. La première est de mettre 3 tâches dans un banc et 1 tâche dans le deuxième banc. La deuxième consiste à mettre deux tâches dans chaque banc (figure 3.9). Maintenant pour chaque possibilité il existe différentes permutations des tâches aux bancs. Par exemple dans le premier cas, l'allocation des tâches T_1, T_2 et T_3 au banc b_1 et de la tâche T_4 au banc b_2 notée $\{(T_1, T_2, T_3) ; T_4\}$ a une consommation différente de l'allocation des tâches T_2, T_3 et T_4 au banc b_1 et de la tâche T_1 au banc b_2 i.e. $\{(T_2, T_3, T_4) ; T_1\}$. Chaque permutation différente de tâches peut impliquer des caractéristiques différentes associées à chaque banc (nombre de réveils, taille des bancs, nombre d'accès à chaque banc…) donc une consommation mémoire totale différente. Dans l'exemple, avec 4 tâches et 2

bancs il existe 4 permutations différentes avec 3 tâches dans un banc et 1 tâche dans le deuxième banc et 3 permutations avec 2 tâches par banc.

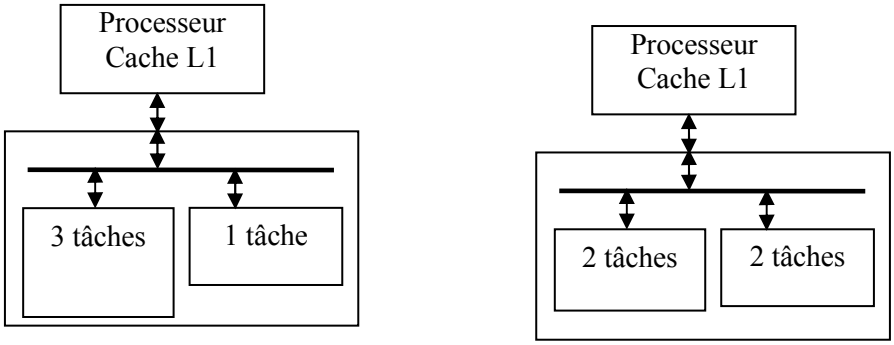


Figure 3.9. Allocations de 4 tâches dans 2 bancs

Pour un nombre de bancs égal à 3, il existe une seule possibilité de ranger 4 tâches dans 3 bancs (2 tâches dans un banc et 1 tâche dans chacun des 2 bancs restants comme illustré sur la figure 3.10). Mais il existe six permutations possibles.

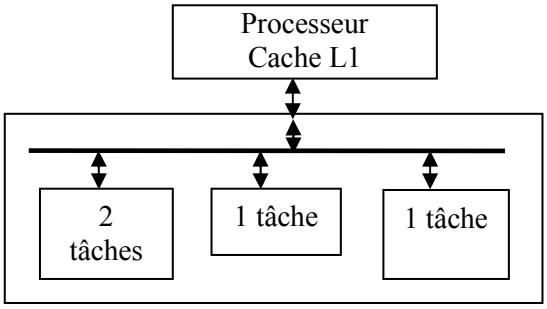


Figure 3.10. Allocations de 4 tâches dans 3 bancs

Pour un nombre de bancs égal à 4, la seule allocation possible est de mettre chaque tâche dans un banc mémoire différent.

Finalement, 15 allocations différentes sont possibles pour ranger 4 tâches dans un nombre de bancs variant de 1 à 4.

Bien que l'approche exhaustive permette de rendre la solution optimale, l'espace d'exploration augmente exponentiellement avec le nombre de tâches. Pour donner une idée de la variation du nombre de configurations mémoire en fonction du nombre de tâches, quelques valeurs des nombres de Bell sont présentées : $B_4 = 15$, $B_5 = 52$, $B_6 = 203$, $B_7 = 877$, $B_8 = 4140$, $B_9 = 21147$, $B_{10} = 115975$, … $B_{20} = 51724158235372$.

Ce type de résolution exhaustive reste intéressant s'il s'agit de l'employer hors ligne, sur des applications à faible nombre de tâches (inférieur à 10 par exemple). Une approche heuristique capable d'explorer un sous-espace potentiellement intéressant et de résoudre le problème en un temps polynomial est nécessaire pour traiter des problèmes de taille

plus importante ou pour envisager une exécution en ligne afin de supporter par exemple la création dynamique de tâches.

2.4. Approche heuristique

Il s'agit de développer une heuristique qui présente un compromis entre vitesse d'exploration et qualité des solutions.

2.4.1. Type de résolutions possibles

Plusieurs heuristiques pour aborder la résolution de problèmes NP-complet sont disponibles et bien étudiées dans la littérature. Les algorithmes du recuit simulé, le *tabu search*, les algorithmes génétiques ou le *branch and bound* sont des exemples d'heuristique capables de résoudre ce type de problème. Ces heuristiques possèdent généralement des temps de résolution relativement importants et cependant l'optimalité de la solution obtenue n'est généralement pas garantie. Elles sont généralement employées hors ligne lors de la phase de conception. Des extensions à ces heuristiques sont peu envisageables afin de réduire leurs temps de résolution et de les adapter à des systèmes mobiles et flexibles (création dynamique de tâches, téléchargement d'applications,…) qui sont de plus en plus présents dans les applications multimédia embarquées. Ces applications nécessitent une réaction rapide en ligne afin d'adapter la configuration du système aux changements et de proposer une nouvelle solution (partitionnement, allocation, …) performante. Des heuristiques plus rapides sont alors nécessaires pour de tels systèmes.

Deux heuristiques nous semblaient pouvoir être utilisées pour résoudre le problème d'allocation de N tâches dans un nombre de bancs k variant de 1 à N de tailles différentes (non uniforme). La première est celle développée pour résoudre le problème de *bin packing* ou de sacs à dos multiples. Ce problème consiste à placer un certain nombre d'objets de tailles différentes dans des sacs de taille fixe. Dans notre cas, les objets peuvent représenter les tâches et les sacs représentent les bancs. Cependant le problème du *bin packing* considère un nombre fixe de sacs, de taille uniforme. Ces deux différences rendent notre problème d'allocation de tâches aux bancs non modélisable sous la forme d'un problème de *bin packing*.

Une deuxième heuristique possible est l'algorithme de Kernighan/Lin ou le *min-cut*. Cet algorithme consiste à regrouper tous les objets dans une partition initiale, puis à déterminer la variation d'une fonction coût lorsque chaque objet est déplacé de la partition initiale vers une deuxième partition. L'objet qui produit la plus grande réduction du coût est déplacé. L'algorithme itère suivant ce principe tant que le coût peut être diminué. Cet algorithme considère uniquement deux partitions. Une amélioration a conduit à l'algorithme *multi-way partitioning* où plusieurs partitions sont considérées. Le déplacement de chaque objet est alors testé dans chaque partition. Pour modéliser notre problème avec cette approche, N itérations successives de cet algorithme, en considérant un nombre de partitions k de 1 à N, sont nécessaires. La solution finale est la moins consommatrice parmi les N solutions rendues à chaque itération de l'algorithme. Cette approche possède une complexité importante et devient inapplicable quand le nombre de tâches N augmente.

2.4.2. Heuristique proposée

L'heuristique développée est divisée en deux étapes. La première génère une première allocation des tâches aux bancs et fixe le nombre de bancs de l'architecture mémoire. Un raffinement de cette solution est effectué dans la deuxième étape tout en gardant le nombre de bancs de la solution obtenue dans la première étape.

- Première étape : *Génération d'une solution initiale*

Dans un premier temps, toutes les tâches sont allouées à un banc initial. Parmi les tâches présentes dans le banc initial, on isole celle qui produit la plus grande réduction d'énergie quand elle est placée dans un banc supplémentaire. Itérativement on répète ce processus pour toutes les tâches restantes dans le banc initial. Le choix de la tâche à isoler dans un banc supplémentaire est basé sur un critère appelé *Isolation(T_i)* calculé pour chaque tâche. Il exprime le gain énergétique local au banc initial lié au fait d'isoler la tâche considérée dans un nouveau banc.

$$Isolation\ (T_i) = \left[\begin{array}{l} \frac{H}{P_i} \times \left(N_{cycles\_accès\_Ti} \times E_{0accès} + N_{cycles\_nonaccès\_Ti} \times E_{0nonaccès} + c_i \times E_{0Mode\_repos} \right) \\ + \left(\frac{H}{P_i} - \sum_{j=1, j \neq i / \varphi(T_j)=b_1}^{N} S_{ij} \right) \times E_{0resynchrni\ sation} \end{array} \right] \times \left(\frac{\sum_{j=1, j \neq i\ /\ \varphi(T_j)=1}^{N} S_{T_j} - S_{T_i}}{S_{T_i}} \right.$$

Ce critère tient compte à la fois des caractéristiques de la tâche, de l'ordonnancement ainsi que de l'architecture mémoire. Les caractéristiques de la tâche sont le nombre d'accès en cycles mémoire $N_{cycles\_accès\_Ti}$, le nombre de cycles sans accès mémoire $N_{cycles\_nonaccès\_Ti}$, le temps d'exécution c_i et la taille S_{Ti}. Les caractéristiques de l'ordonnancement sont exprimées par le nombre d'exécutions de la tâche durant l'hyperpériode ainsi que les successivités de la tâche T_i par rapport aux autres tâches du système. Les énergies $E_{0accès}$, $E_{0nonaccès}$, $E_{0resynchronisation}$, E_{0Mode_repos} reflètent les caractéristiques en consommation de la mémoire. Ce critère favorise l'isolation de la tâche qui effectue de nombreux accès à la mémoire, qui oblige la mémoire à rester longtemps coactive avec le processeur, qui permet au banc initial de rester le plus longtemps en mode faible consommation suite à son isolation, qui réveille souvent le banc initial et en même temps qui possède une faible taille par rapport aux tâches restantes dans le banc initial. Ce dernier facteur a pour objectif d'éviter que l'énergie de repos, d'un banc de grande taille ajouté à la mémoire, devienne prépondérante par rapport au gain en énergie obtenu par création de ce banc.

Une évaluation de la consommation mémoire totale est effectuée après le déplacement de la tâche T_i maximisant la valeur du critère *Isolation(T_i)* (nouvelle solution). Si une réduction de la consommation mémoire totale est constatée dans la nouvelle solution, on garde cette configuration mémoire, on met à jour la valeur du critère pour les tâches restantes (la contribution liée à la successivité a changé après déplacement de la tâche) et on réitère pour isoler une nouvelle tâche issue du banc initial. Sinon, la solution actuelle est rejetée et la configuration mémoire de la solution précédente est retenue pour un raffinement suivant l'étape 2. La figue 3.11 présente l'algorithme de cette première étape.

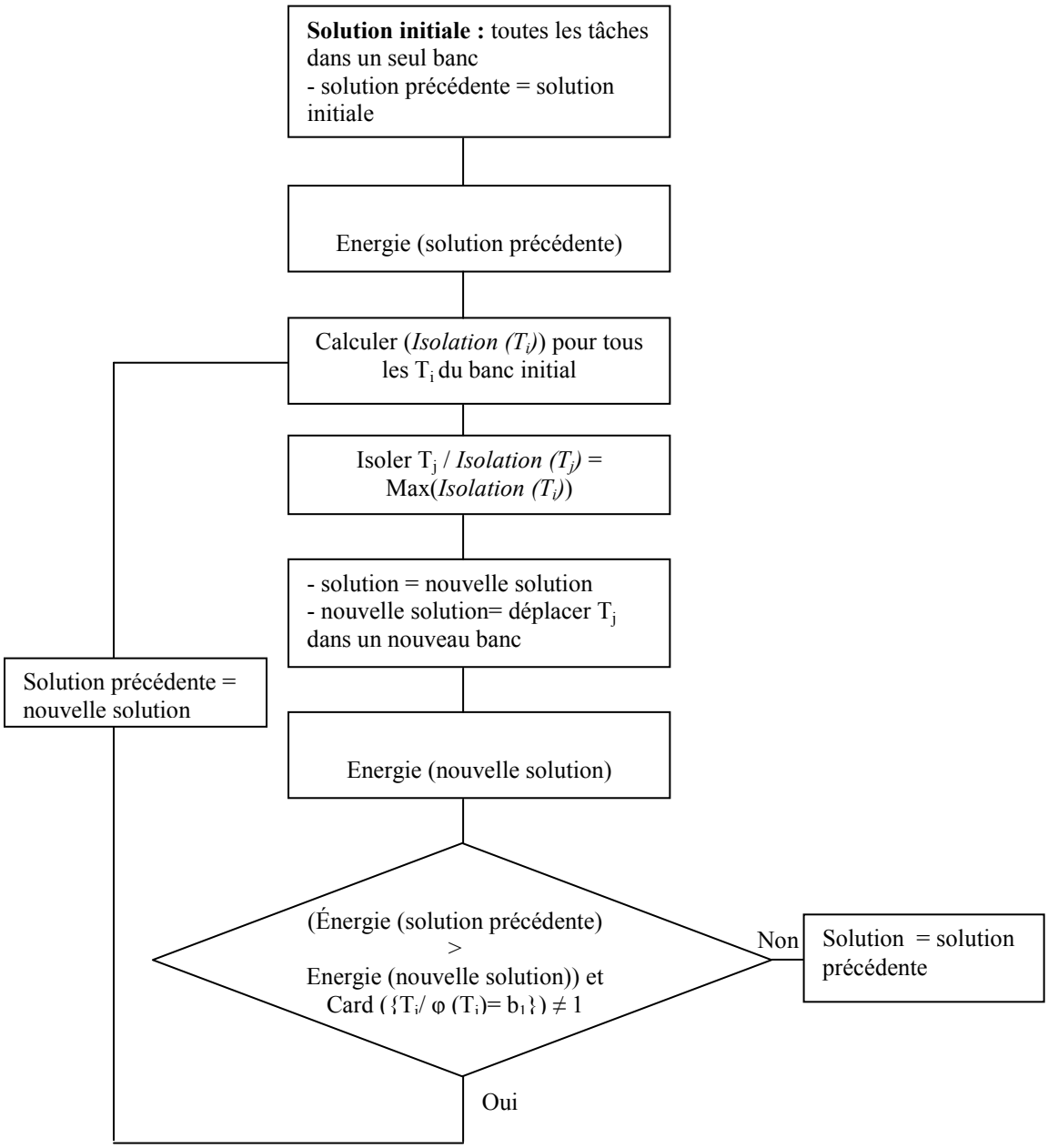


Figure 3.11. Algorithme de la première étape de l'heuristique : génération d'une première solution

Dans cette première étape nous avons choisi d'utiliser le critère *Isolation(T_i)* pour sélectionner la tâche à déplacer du banc initial. On aurait pu considérer le calcul du gain énergétique global sur l'ensemble des bancs mémoire et des tâches qu'ils contiennent, pour identifier la tâche à déplacer depuis le banc initial. Or, comme illustré dans le chapitre deux, l'évaluation du gain énergétique global a une complexité en $O(N^2)$ ce qui ne milite pas pour retenir ce gain global en vue d'obtenir un compromis entre qualité de

solution et rapidité d'exécution. C'est la raison pour laquelle nous avons choisi le critère *Isolation(T_i)* qui possède une complexité en O(N).

A l'issue de cette première étape, nous obtenons un ensemble de bancs qui, hormis le banc initial, contiennent chacun une tâche. Les autres tâches (au moins une) étant regroupées dans le banc initial. Un exemple de configuration mémoire et d'allocation pour un ensemble de 8 tâches obtenu à l'issue de cette première étape est donné figure 3.12.

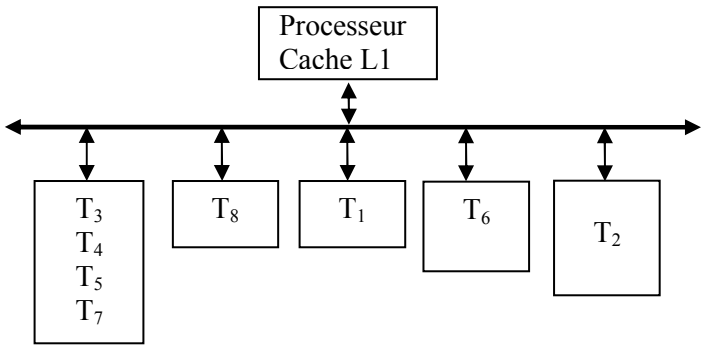


Figure 3.12. Exemple d'architecture mémoire et d'allocation rendue par la première étape de l'heuristique

Le nombre de bancs de l'architecture mémoire est fixé dans cette première étape et reste inchangé durant la deuxième étape. Cette dernière consiste à migrer aux mieux les tâches du banc initial vers les autres bancs afin de réduire encore la consommation d'énergie et d'améliorer la solution obtenue par la première étape.

- Deuxième étape : *Raffinement de la solution rendue par la première étape*

Les bancs sont tout d'abord classés selon l'ordre croissant de leur consommation. La tâche T_i du banc initial ayant la plus grande valeur du critère *Isolation(T_i)* est allouée au banc b_j le moins consommateur. Une évaluation de la consommation globale est faite pour cette allocation potentielle. Si une réduction de la consommation mémoire est constatée, on garde cette nouvelle solution. On met à jour la valeur du critère relatif aux tâches du banc initial et la consommation des bancs concernés par le déplacement (le banc initial et le banc b_j). Ce processus est réitéré tant que des tâches peuvent être déplacées du banc initial vers le banc le moins consommateur qui doit être différents du banc initial. Sinon on garde la configuration mémoire précédente. La figue 3.13 présente l'algorithme de cette deuxième étape de l'heuristique.

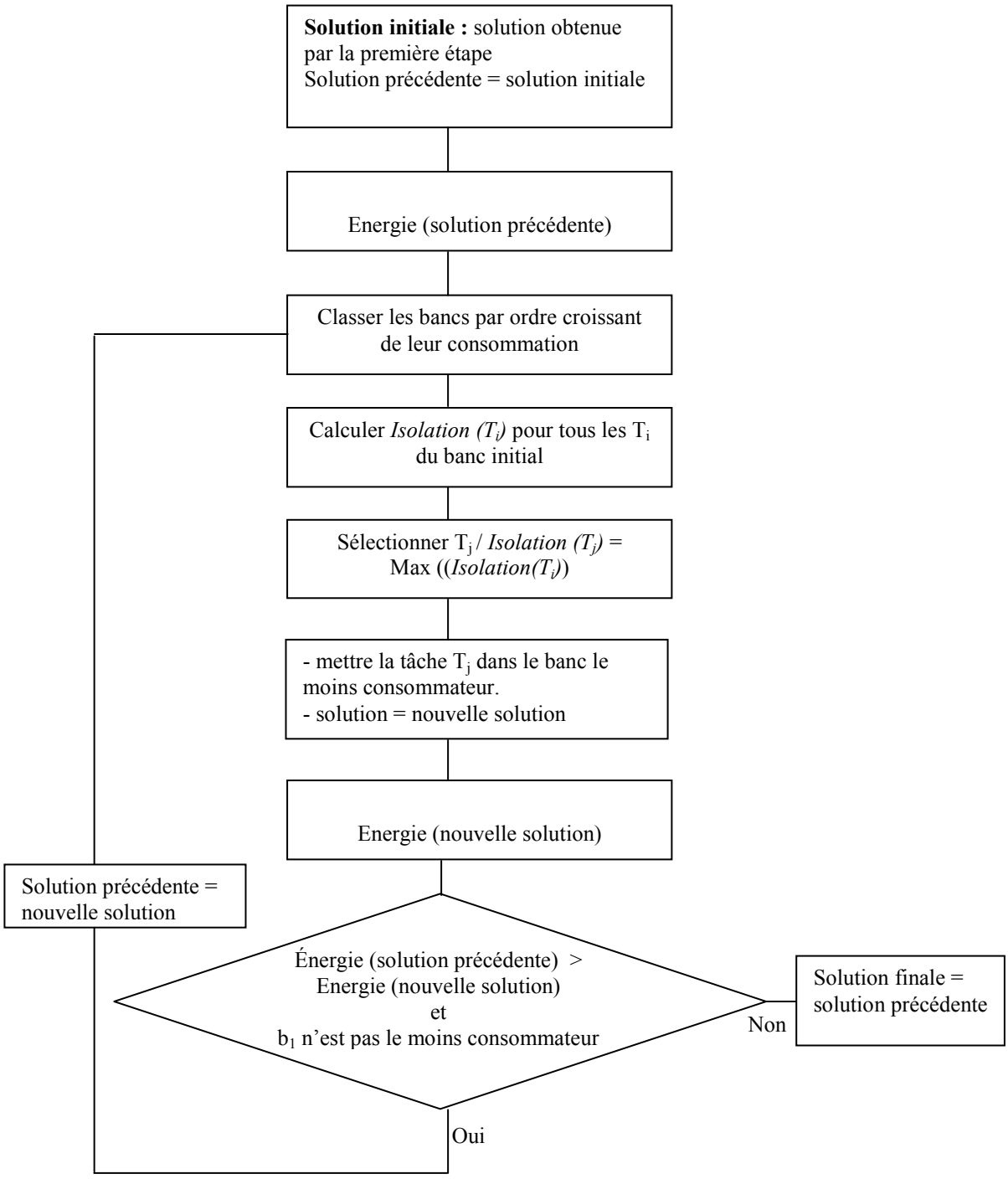


Figure 3.13. Algorithme de la deuxième étape de l'heuristique : raffinement de la solution rendue par la première étape

À l'issue des deux étapes de l'heuristique et pour le même exemple de tâches considéré dans la figure 3.12, un exemple de configuration mémoire et d'allocation de tâches est décrit dans la figure 3.14.

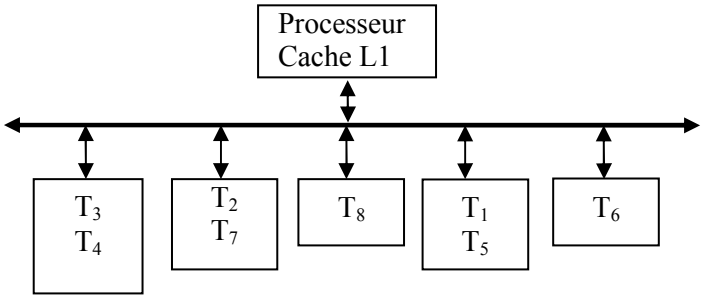


Figure 3.14. Configuration et allocation rendue par la deuxième étape de l'heuristique

Afin de tester l'efficacité de l'heuristique, la solution rendue sera comparée à la solution optimale obtenue par l'algorithme exhaustif dans les résultats présentés dans le chapitre 4.

Une complexité en $O(N^3)$ de l'heuristique est obtenue qui est bien inférieure à celle de l'approche exhaustive. Cette complexité est dominée par celle en $O(N^2)$ de la fonction d'évaluation de l'énergie. La complexité de cette heuristique permet de l'employer en ligne sur des applications avec un nombre important de tâches (plusieurs dizaines) comparé à la méthode exhaustive.

Pour des systèmes mobiles (création dynamique de tâches, téléchargement d'applications,…) où le nombre de tâches évolue en cours d'exécution, une migration de tâches d'un banc vers un autre peut être alors imaginée pour minimiser la consommation d'énergie et exploiter efficacement le DVS du processeur.

3. Allocation du système d'exploitation temps réel (RTOS)

Les systèmes d'exploitation temps réel sont de plus en plus présents dans les systèmes embarqués afin de gérer la complexité croissante des applications (Communication, Vidéo, Visiophonie, Internet, Jeux …) ainsi que de l'architecture (unités de calcul hétérogènes, mémoire virtuelle, séquencement complexe d'interruptions, accélérateurs graphique, …). Comme exemples de systèmes d'exploitation dédiés aux systèmes embarqués, on cite le PalmOS pour les ordinateurs de poche (les Palms), le système d'exploitation SymbianOS présent sur les téléphones portables. Plusieurs versions de ce dernier sont disponibles sur le marché : Symbian 6.1 pour les téléphones 2G et 2.5G (GSM & GPRS), Symbian 7.0 pour les téléphones 2.5G (GPRS), Symbian 8.0 pour les téléphones 3G (GPRS & UMTS), Symbian 9.0 pour les téléphones 3G (EDGE & UMTS). On cite aussi le système d'exploitation VxWorks qui a été employé par la NASA pour des missions spatiales comme la sonde martienne *Mars Reconnaissance*

Orbiter. Les systèmes d'exploitation µcosII et Nucleus sont aussi très utilisés dans les systèmes embarqués temps réel.

La taille de ces systèmes d'exploitation (code, données et pile des tâches) peut ne pas être négligeable surtout avec les fonctionnalités de plus en plus variées existantes dans les dernières versions de ces RTOS. La taille peut varier de quelques kilo-octets pour un noyau basique jusqu'à 1MB pour les plus gros avec un nombre de fonctionnalités important. Ce code est forcément résident dans un des bancs mémoire et, chaque fois qu'il y a une opération système (réveil d'une tâche, fin d'exécution d'une tâche, préemption, gestion mémoire, …) il est nécessaire d'activer le banc où le RTOS réside. Cependant dans les deux approches décrites précédemment (exhaustive et heuristique), l'emplacement du code et des données du système d'exploitation temps réel dans la mémoire principale a été négligé. Dans ce paragraphe on détaille comment l'allocation du système d'exploitation peut être prise en compte dans notre approche : sa modélisation ainsi que sa contribution énergétique dans une mémoire multi-bancs.

Définissons une fonction d'allocation φ' qui associe à chaque tâche T_i appartenant à un ensemble fini de N+1 tâches (N tâches applicatives + RTOS) un banc b_j appartenant lui aussi à un ensemble fini de k bancs ($1 \leq k \leq N+1$).

$$\varphi' : \{T_1, T_2, \ldots T_N, \text{RTOS}\} \rightarrow \{b_1, b_2, \ldots b_k\}$$

$$\varphi'(T_i) = b_j$$

Le système d'exploitation peut ainsi être seul dans un banc (figure 3.15.b) ou résider avec une ou plusieurs tâches dans un même banc (figure 3.15.a).

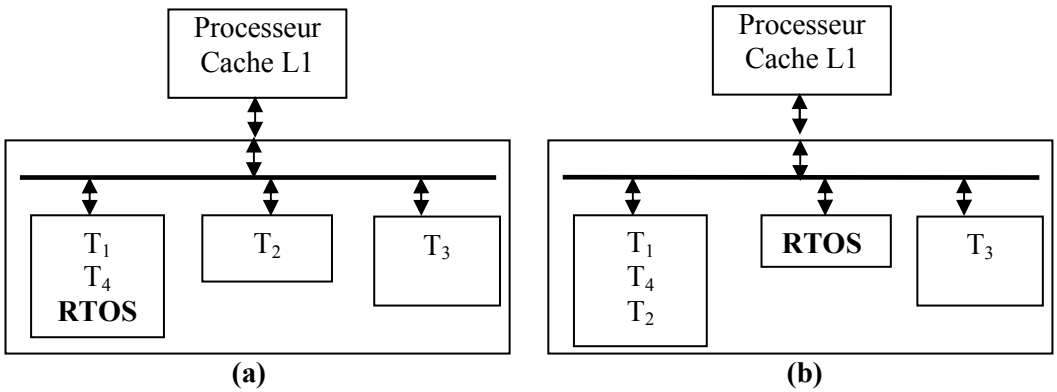


Figure 3.15 : Deux allocations possibles du RTOS : (a) avec d'autres tâches dans un banc, (b) seul dans un banc

3.1. Modélisation du système d'exploitation temps réel

Le système d'exploitation est modélisé comme une tâche ayant une taille S_{RTOS} (code, données et pile des différentes tâches), un temps d'exécution processeur c_{OS} et un nombre d'accès à la mémoire principale M_{RTOS}. Chaque tâche est caractérisée par un paramètre supplémentaire R_{Ti} qui exprime le nombre d'appels aux services du RTOS effectués par la tâche T_i. La valeur de R_{Ti} considérée est la moyenne des appels aux

services du RTOS par toutes les instances de la tâche T_i pendant l'hyperpériode. Le paramètre R_{Ti} correspond à la notion de successivité définie au niveau des tâches.

Le tableau 3.2 décrit les caractéristiques d'un exemple d'application de 3 tâches intégrant un RTOS.

Tableau 3.2. Paramètres d'un modèle d'application avec un système d'exploitation

Tâche	P_i(cycles)	c_i (cycles)	S_{Ti} (kB)	M_i	R_{Ti}
T_1	3000	1000	128	167	38
T_2	4000	1000	64	135	45
T_3	6000	2000	512	265	62
RTOS	-	400	32	290	-

En disposant du code source d'un RTOS, la méthode de simulation des caractéristiques des tâches (c_i, S_{Ti}, M_i) avec SimpleScalar décrite dans le paragraphe 2.1 pouvait s'appliquer pour déterminer les caractéristique du RTOS en temps d'exécution c_{RTOS} et nombre d'accès mémoire M_{RTOS} effectués par l'ensemble des services. L'outil Dlite! de SimpleScalar peut être utilisé pour insérer des points de tests au début et à la fin de chaque service du RTOS d'une tâche T_i, obtenant ainsi les temps d'exécution et le nombre d'accès mémoire relatif à l'ensemble des services RTOS de la tâche T_i. Des valeurs moyennes peuvent être obtenues pour les différentes instances de la tâche T_i sur l''hyperpériode. Une sommation de ces deux nombres pour toutes les tâches de l'application donne les nombres c_{RTOS} et M_{RTOS} (figure 3.16).

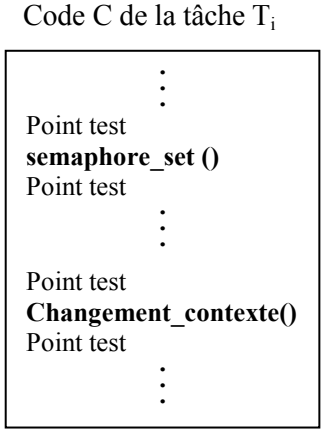


Figure 3.16. Simulation des caractéristiques du RTOS (c_{RTOS} et M_{RTOS})

Cependant la mise en œuvre de cette technique est complexe. Premièrement, il faut disposer du code source du RTOS. Aussi, une modification des codes sources des tâches par l'introduction des appels système est nécessaire. Pour des raisons de simplicité, une approximation des paramètres du RTOS par analyse des différentes exécutions des tâches sur l'hyperpériode est réalisée.

Le temps d'occupation processeur du système d'exploitation c_{RTOS} est calculé à partir de l'ordonnancement des tâches construit sur l'hyperpériode. L'ensemble des appels

systèmes liés aux préemptions, aux changements d'états des tâches, gestion des ressources critiques sont ainsi comptabilisés et traduits en temps d'exécution et en taille des piles allouées à chaque tâche.

Le nombre d'accès M_{RTOS} du système d'exploitation à la mémoire principale est calculé avec l'équation 3.1

$$M_{RTOS} = l \times R_{total} \quad \text{avec} \quad l \geq 1 \qquad [3.1]$$

Où l est le taux moyen d'accès mémoire sur l'ensemble des services du RTOS.
Le nombre R_{total} est le nombre total d'appels aux services du RTOS par toutes les tâches est calculé :

$$R_{total} = \sum_{T_i / i=1}^{N} R_{Ti}$$

L'équation 3.1 suppose que chaque service du système d'exploitation effectue *l* accès à la mémoire. En réalité, ce facteur dépend de la nature du service. Il a une valeur importante pour les services relativement complexes (préemption, changement de contexte…) et faible pour les services plus simple (synchronisation, …). Il est considéré constant pour tous les types de services dans cette étude. Différencier des valeurs pour chaque service ne poserait pas de difficultés particulières.

Les modèles de consommation établis dans le paragraphe 4.3.2 du chapitre 2, restent valables tout en considérant le système d'exploitation comme une tâche.

$$E_{mémoire} = E_{accès} + E_{nonaccès} + E_{Mode\_repos} + E_{resynchron\ isation} + E_{préemption} + E_{bus}$$

Par exemple, supposons que le code et les données du RTOS résident avec la tâche T_3 dans le banc b_2 (figure 3.17.a). Durant l'exécution des tâches T_1 et T_2 le banc b_2 est sollicité à chaque fois que ces deux tâches font appel à un service du RTOS. Ces appels de services du RTOS sont à l'origine de dissipation énergétique liée aux réveils du banc b_2 et aux accès à la mémoire. Sur la figure 3.17.b, l'exécution des services du RTOS pendant un temps c_{RTOS} est regroupée à la fin de l'hyperpériode par commodité. En réalité, elle est distribuée ainsi que les énergies ($E_{accès}$, $E_{nonaccès}$, $E_{resynchronisation}$) consommées par le RTOS sur l'hyperpériode pendant les exécutions des 3 tâches.

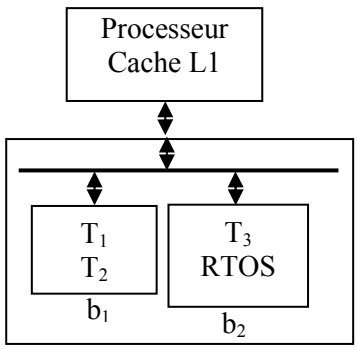


Figure 3.17.a. Allocation de trois tâches et un RTOS dans deux bancs mémoires.

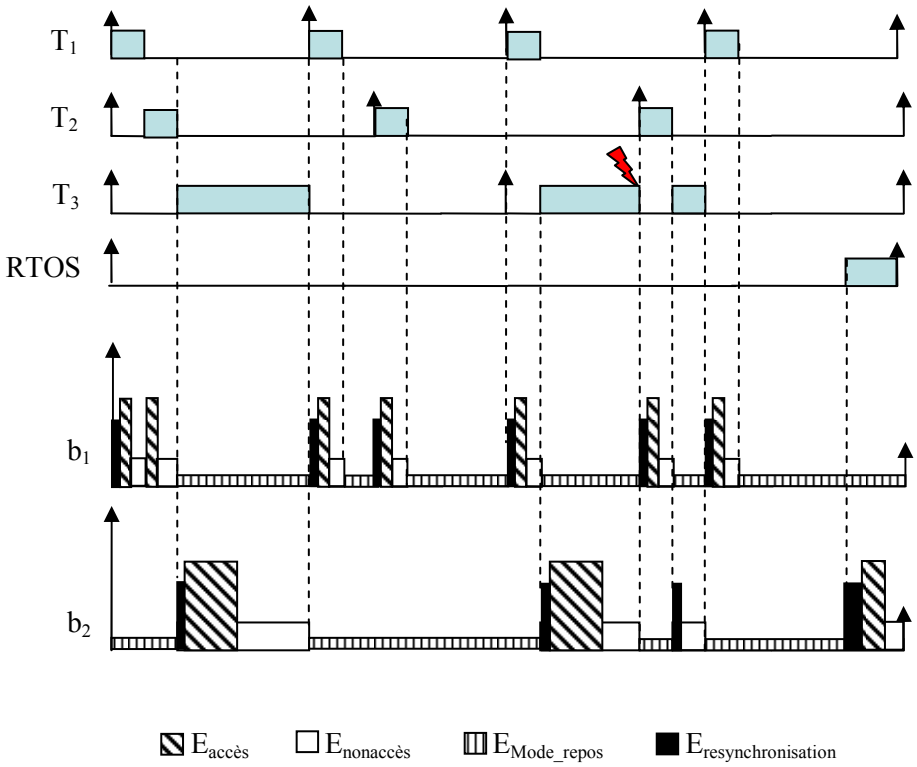


Figure 3.17. b. Les différentes énergies consommées pour l'allocation de la figure 3.17.a.

On détaille maintenant l'adaptation du modèle d'évaluation de la consommation pour tenir compte de la contribution due au RTOS.
La taille du banc b_j renfermant le RTOS a sa taille S_{bj} augmentée par la taille S_{RTOS} du RTOS. Le calcul des différents paramètres relatif au RTOS s'effectue de la même façon que les tâches de l'application.

Le nombre d'accès $N_{cycles\_accès\_Ti}$ en cycles mémoire du RTOS au banc mémoire est défini par :

$$N_{cycles\_accès\_RTOS} = M_{RTOS} \times t_{accèsM} \times f_{mémoire}$$

Le nombre de cycles mémoire $N_{cycles\_nonaccès\_Ti}$ du RTOS où le banc mémoire est actif mais sans accès est :

$$N_{cycles\_nonaccès\_RTOS} = (c_{RTOS} \times \frac{1}{f_{processur}} - M_{RTOS} \times t_{accèsM}) \times f_{mémoire}$$

Le nombre de cycles mémoire $N_{cycles\_Mode\_repos\_bj}$ où le banc b_j est en mode faible consommation est déterminé par :

$$N_{cycles\_nonaccès\_RTOS} = (c_{RTOS} \times \frac{1}{f_{processur}} - M_{RTOS} \times t_{accèsM}) \times f_{mémoire}$$

Le nombre de resynchronisations du banc contenant le RTOS est modifié du fait des appels de services par les tâches. Selon l'emplacement du RTOS (seul ou avec d'autres tâches dans un banc), deux équations différentes sont établies.

- Le RTOS réside avec d'autres tâches dans un banc

Le nombre de resynchronisations de ce banc est calculé selon l'équation suivante.

$$N_{resynchronisation\_bj} = \sum_{T_i / \varphi'(T_i)=b_j} N_{exeT_i} - \sum_{T_i,T_j / (\varphi'(T_i),\varphi'(T_j))=(b_j,b_j)} \sigma_{ij} + R_{total} - \sum_{T_i / \varphi'(T_i)=b_j} R_{Ti}$$

Le terme $R_{total} - \sum_{T_i / \varphi'(T_i)=b_j} R_{Ti}$ exprime le nombre de services du RTOS invoqués par les tâches ne résidant pas avec le RTOS dans le même banc.

- Le RTOS réside seul dans un banc

Deux cas sont possibles. Le premier consiste à réveiller le banc à chaque fois qu'il y a un appel système. Il est en mode repos entre deux appels. Le nombre de resynchronisations de ce banc est alors calculé selon l'équation suivante.

$$N_{resynchronisation\_bj} = R_{total}$$

Le deuxième cas consiste à maintenir ce banc en mode *Active* durant l'hyperpériode et interdire une mise en mode repos. Une comparaison d'énergie et de performance de ces deux cas sera présentée dans le chapitre suivant.

3.2. Modification de l'algorithme exhaustif

Pour une application à N tâches avec un système d'exploitation temps réel, seul le nombre de configurations augmente et devient égal à B_{N+1} au lieu de B_N puisque le RTOS est considéré comme une tâche supplémentaire. En considérant le RTOS, l'algorithme exhaustif doit donc explorer un espace de solutions plus large. Le nombre de bancs minimum reste égal à 1 lorsque toutes les tâches ainsi que le RTOS y résident. Le nombre de bancs maximum passe à (N+1) bancs en considérant le système d'exploitation temps réel et les tâches alloués chacun à un banc.

3.3. Modification de l'heuristique

La structure en deux étapes de l'heuristique reste inchangée. La seule modification est introduite dans la définition du critère *Isolation (T_i)*. Ce critère doit tenir compte, en plus des différentes tâches de l'application, du système d'exploitation temps réel RTOS. A partir de la solution initiale et suivant la valeur du critère, il peut alors être décidé d'isoler soit une tâche T_i soit le RTOS dans un banc afin de minimiser la consommation. Ce critère est calculé pour chaque tâche T_i ainsi que pour le RTOS :

$$Isolation\ (T_i) = \left[\begin{array}{l} \frac{H}{P_i} \times \left(N_{cycles\_accès\_Ti} \times E_{0accès} + N_{cycles\_nonaccès\_Ti} \times E_{0nonaccès} + c_i \times E_{0Mode\_repos} \right) \\ + \left(\frac{H}{P_i} - \sum_{T_i, T_j / (\varphi(T_i), \varphi(T_j)) = (b_j, b_j)} \sigma_{ij} + R_{total} - R_{Ti} \right) \times E_{0resynchrnisation} \end{array} \right] \times \left(\frac{\sum_{j=1, j \neq i \ / \ \varphi(T_j)=1}^{N} S_{T_j} - S_{T_i}}{S_{T_i}} \right)$$

Le terme ($R_{total} - R_{Ti}$), ajouté dans la dernière somme du critère, permet de calculer le nombre de fois que la tâche T_i réveille le banc initial en tenant compte cette fois des appels aux services du RTOS. Ainsi il favorise l'isolation, du banc initial, la tâche T_i ayant le plus faible nombre d'appels aux services du RTOS.

Conclusion

Le partitionnement mémoire en plusieurs bancs accessibles et contrôlés séparément est adopté comme solution pour contenir l'augmentation de la consommation mémoire lors de l'emploi de la technique de DVS sur le processeur. Une deuxième optimisation de l'énergie est effectuée au sein même de la mémoire multi-bancs. Elle consiste à allouer les tâches aux bancs et à configurer la mémoire (nombre de bancs et tailles des bancs) de façon à minimiser la consommation de la mémoire. Deux approches ont été développées. La première est une exploration exhaustive permettant d'obtenir la solution optimale. Cette approche reste intéressante s'il s'agit de l'employer hors ligne, en phase de conception pour des applications à nombre de tâches moyen. La deuxième approche est une heuristique à complexité réduite. Elle peut être utilisée lorsque le nombre de tâches de l'application rend l'approche exhaustive impraticable ou dans un contexte en ligne pour tenir compte du caractère dynamique des systèmes mobiles comme le téléchargement de nouvelles applications, allocation de tâches crées dynamiquement (jeux, photos…). Des expérimentations, sur des applications de traitement de signal puis sur une application multimédia, afin de tester l'efficacité de notre approche sont présentées dans le chapitre suivant.

Chapitre 4. Expérimentations et Résultats

Introduction

Dans ce chapitre, des résultats permettant la validation de notre méthodologie sont exposés sur deux principales applications test. La première est une combinaison de tâches de traitement de signal temps réel qui s'exécute sur une architecture monoprocesseur. La deuxième est une application multimédia (GSM et décodeur MPEG-2) qui s'exécute sur une architecture multiprocesseur de type maître-esclave comme le système OMAP de Texas Instruments. Une partie est consacrée à la validation de la solution obtenue avec l'heuristique développée. Dans la dernière partie du chapitre on discute de l'allocation du système d'exploitation puis de la variation de la configuration mémoire en fonction de l'ordonnancement.

1. Expérimentations sur des applications de traitement de signal

Dans cette première étude, l'architecture cible est décrite dans le paragraphe 1.1 suivie par la description de l'application dans le paragraphe 1.2. Des expérimentations présentant l'influence de la technique de DVS sur la consommation mémoire ainsi que la justification de l'utilisation d'une mémoire multi-bancs pour augmenter le gain énergétique du système global sont présentées dans le paragraphe 1.3. Dans la dernière partie, la variation de la consommation d'une mémoire multi-bancs est étudiée d'abord en fonction du nombre de bancs (paragraphe 1.4.1), puis en fonction du mode faible consommation employé (paragraphe 1.4.2).

1.1. Description de l'architecture cible

Un processeur PXA270 d'Intel (Intel, 2005) et une mémoire principale RDRAM sont considérés dans les expérimentations. Cette mémoire communique avec le processeur à travers un cache L1 de 32 KB. Par défaut, le mode *Nap* est choisi dans les expérimentations comme mode faible consommation de la mémoire RDRAM.

Pour estimer l'énergie par accès du cache L1, une version améliorée de l'outil CACTI (Shivakumar *et al.*, 2001), appelé eCACTI (*enhancedCACTI*) (Mamidipaka *et al.*, 2004) est utilisée. Cette nouvelle version permet en effet d'estimer en plus de la dissipation dynamique, la consommation statique pour les nouvelles technologies. Le tableau 4.1 récapitule les différentes caractéristiques du système.

Tableau 4.1 Paramètres système utilisés dans nos simulations

Composant	Paramètres
Processeur	PXA270 d'Intel, $f_{nominal}$ = 624 Mhz, **Consommation :** 925 mW à (624 MHZ, 1,55 V) 747 mW à (520 MHz, 1,45 V) 570 mW à (416 MHZ, 1,35V) 390 mW à (312 MHZ, 1,25V) 279 mW à (208 MHZ, 1,15V)
Cache	32 KB cache de données, 32KB cache d'instructions, taille de ligne = 32 Byte, associativité = 32, latence = 2cycles. E_{hit} = 0,5 nJ ; E_{miss}= 0,75nJ
Mémoire principale	Rambus RDRAM, latence = 50 ns, V_{DRAM}= 3,3 V **Consommation :** $E_{0accès}$= 4,6 nJ $E_{0nonaccès}$= 2,5 nJ E_{0Mode_repos} et $E_{0resynchronisation}$ (tableau 2.1 du chapitre 2) $E_{0préemption}$ = 0,2 mJ (Jejurikar *et al*., 2004) E_{0bus} = 1500 nJ

1.2. Description de l'application

Pour expérimenter notre approche d'optimisation de la consommation mémoire, des applications test temps réel provenant de l'Université de Séoul *SNU real-time benchmarks* (http://archi.snu.ac.kr/realtime/benchmark/) sont utilisées. Le tableau 4.2 décrit ces applications test.

Tableau 4.2 Applications test

Tâches	**Description**
CRC	Algorithme de contrôle de redondance cyclique
FIR	Filtre FIR
FFT	Transformée de Fourier utilisant l'algorithme de *Cooly-Turkey*
LMS	Algorithme *adaptive signal enhancement*
LUD	Algorithme de décomposition LU de matrices
Matmul	Multiplication de Matrices
ADPCM	CCITT G.722 Algorithme *Adaptive Differential Pulse Code Modulation*
Fibcall	Sommation de séries de *Fibonacci*
Qsort	Algorithme non récursif rapide de tri
IDCT	Transformée inverse en cosinus discrète

Comme présenté dans le chapitre 3, la spécification de chaque tâche en temps d'exécution c_i, en taille mémoire S_{Ti} (instructions et données) est réalisée avec l'outil de simulation architectural SimpleScalar (Burger *et al*., 1997). Le nombre de défauts de cache M_i est déterminé avec le même outil mais en considérant l'exécution des différentes tâches de l'application selon un ordonnancement de type *Rate Monotonic*.

Dans la suite, plusieurs applications multi-tâches sont construites à partir de différents sous-ensembles de ces applications test.

1.3. Influence de la technique de DVS sur la consommation mémoire

Une application constituée de 7 tâches, décrite dans le tableau 4.3 est considérée pour étudier la consommation mémoire lors de l'utilisation de la technique de DVS sur le processeur. Les résultats en terme de consommation processeur et mémoire sont présentés dans le paragraphe 1.3.1. L'allocation des tâches et la configuration mémoire qui optimisent la consommation mémoire sont présentées dans le paragraphe 1.3.2. La technique de DVS est ensuite appliquée sur le processeur avec l'architecture mémoire multi-bancs optimale trouvée dans le paragraphe 1.3.2.

Tableau 4.3 Caractéristiques de l'application considérée pour la technique de DVS

Tâches	P_i (cycles)	c_i (cycles)	S_{Ti} (kB)	M_i
FIR	5000000	33983	152	47
CRC	10000000	42907	31	99
FFT	5000000	515771	97	493
LMS	5000000	365893	32	123
LUD	2500000	255998	38	102
Matmul	5000000	13985	29	26
ADPCM	20000000	2486633	133	3387

1.3.1 Technique de DVS avec une mémoire monolithique

Un facteur d'étalement maximal de α_{max}=1,7 est calculé avec le test de faisabilité (équation 2.3 du chapitre 2). Sur la figure 4.1, la variation de la consommation du processeur et de la mémoire ainsi que la consommation totale (processeur + mémoire) sont tracées en fonction du facteur d'étalement variant de 1 à sa valeur maximale.

Cette technique reste efficace dans la réduction de la consommation processeur. En effet, un gain de 42% sur l'énergie processeur est obtenu pour α_{max}. En contre partie, cette technique de DVS a augmenté la consommation mémoire de 35% pour α_{max}. La diminution du gain en énergie totale passe ainsi de 42% (dans le cas idéal sans la considération mémoire) à seulement 13% quand l'augmentation de la consommation mémoire est comptabilisée. Dans certains cas, une augmentation de la consommation totale peut être obtenue lorsque l'augmentation de la consommation mémoire est supérieure au gain en consommation réalisé par DVS sur le processeur.

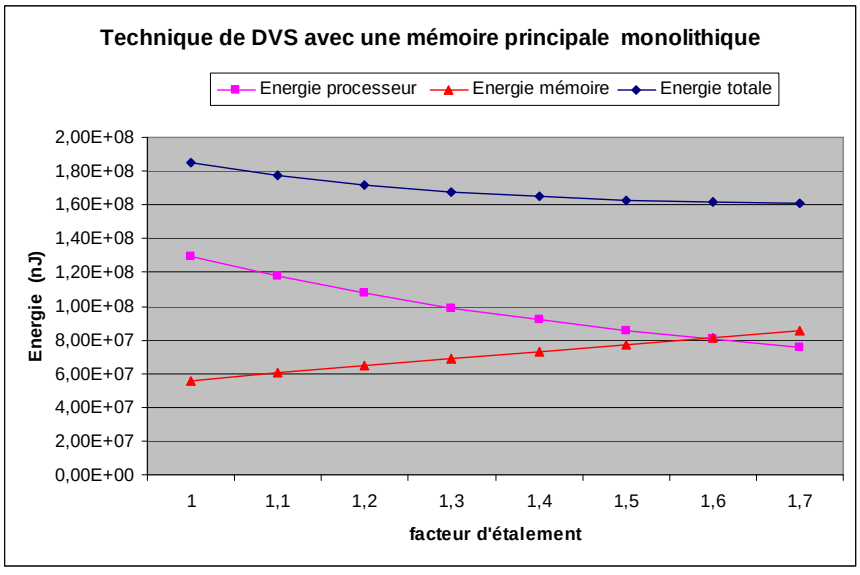


Figure 4.1. Consommations énergétiques lors du DVS pour une architecture avec mémoire monolithique

Afin de cerner les sources de l'augmentation de la consommation mémoire monolithique liée au DVS sur le processeur, la figure 4.2 détaille les contributions des différentes énergies, décrites dans l'équation 2.7 du chapitre 2, pour chaque valeur du facteur d'étalement.

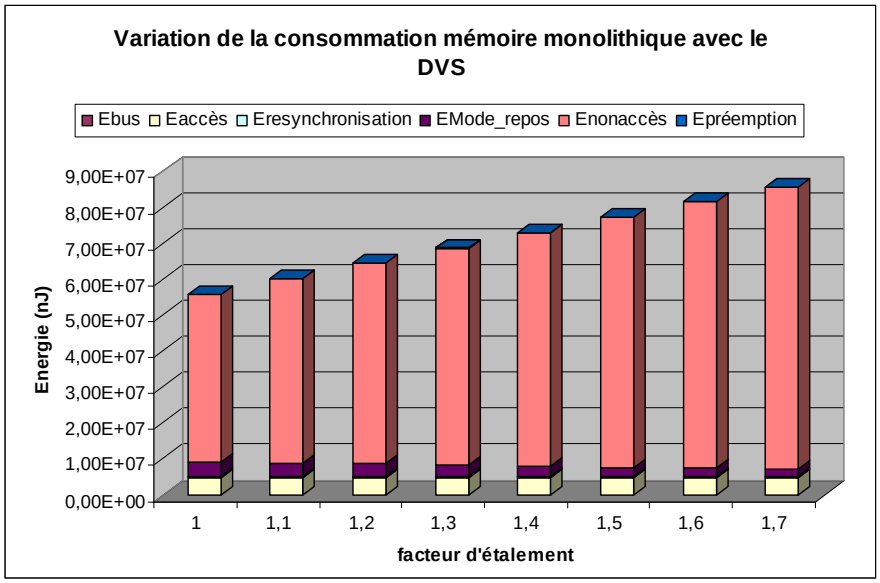


Figure 4.2. Variation des différentes énergies de la consommation d'une mémoire monolithique en fonction du facteur d'étalement

L'augmentation de l'énergie de non accès $E_{nonaccès}$ avec le facteur d'étalement est très remarquable sur la figure 4.2 par rapport à la variation très faible de l'énergie dissipée en mode repos E_{Mode_repos} et la valeur constante de l'énergie d'accès $E_{accès}$.

Pour l'application considérée, des préemptions supplémentaires se produisent pour les valeurs du facteur d'étalement 1,3 et 1,5. La tâche la moins prioritaire ADPCM est préemptée par des tâches plus prioritaires, mais la contribution de l'énergie de préemption à la consommation totale reste faible.

Dans cet exemple, l'augmentation importante de la consommation totale de la mémoire est due, non pas à l'augmentation de l'énergie de préemption, mais essentiellement à une co-activation plus longue de la mémoire avec le processeur lors de la diminution de la fréquence du processeur et par conséquent avec l'allongement des temps d'exécution des tâches.

Si on considère une mémoire multi-bancs avec la possibilité de mettre indépendamment des bancs en mode faible consommation, il est possible de réduire la quantité de mémoire co-active, avec le processeur. D'une taille corresponde à la totalité de la mémoire (mémoire monolithique) il est possible de passer à uniquement un seul banc et ainsi espérer des gains en énergie plus intéressants lorsque la fréquence et la tension d'alimentation du processeur sont réduites.

1.3.2. Consommation d'une mémoire principale multi-bancs

Étant donné que l'application considérée contient 7 tâches, B_7 = 877 configurations mémoire sont possibles. Ce nombre de configurations étant important, seules les configurations (allocation, nombre et taille de bancs) donnant une énergie minimale pour chaque nombre de bancs, qui constituent l'architecture mémoire, sont présentées sur la figure 4.3.

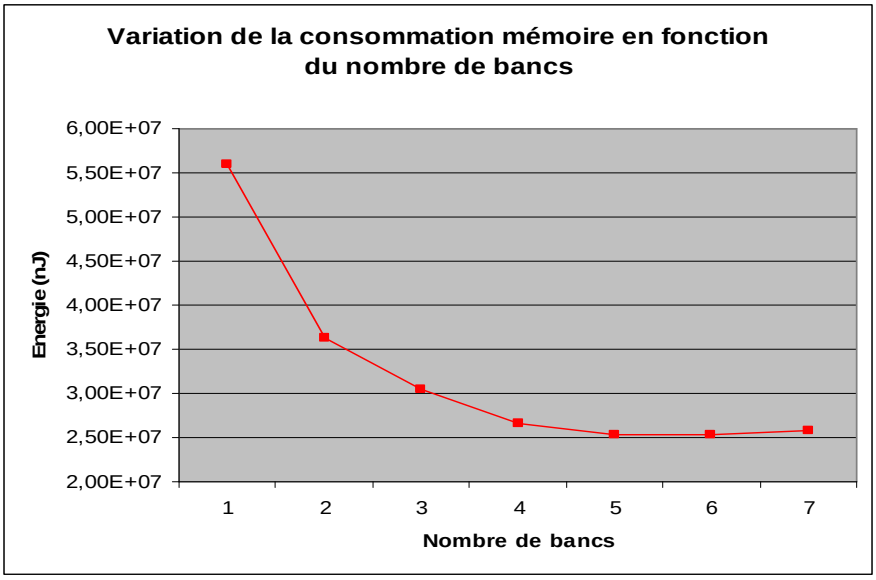


Figure 4.3. Variation de la consommation mémoire en fonction du nombre de bancs de la structure mémoire

L'efficacité de l'architecture multi-bancs dans la réduction de la consommation mémoire par rapport à l'architecture mémoire monolithique (correspondant au cas d'un seul banc sur la figure 4.3) est illustrée. L'énergie mémoire optimale est obtenue avec une architecture à 5 bancs. Le gain en énergie est de 54% par rapport à la solution monolithique. L'allocation des tâches optimale rendue par l'algorithme exhaustif est présentée sur la figure 4.4.

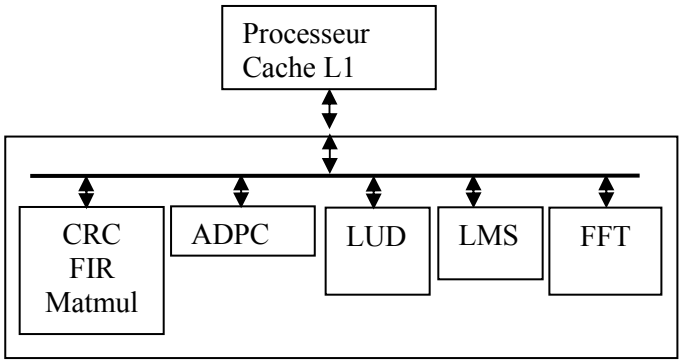


Figure 4.4. Allocation optimale rendue par l'algorithme exhaustif

Cette allocation de tâches et sa configuration mémoire correspondante sont considérées dans le paragraphe suivant lors de l'ajustement de la tension et de la fréquence du processeur.

1.3.3. Technique de DVS avec une mémoire principale multi-bancs

Une fois que l'allocation des tâches et la configuration mémoire optimales sont obtenues (figure 4.4), l'ajustement conjoint de la tension et de la fréquence du processeur est effectué. La consommation du processeur, de la mémoire et la consommation totale (processeur + mémoire) sont présentées dans la figure 4.5 en fonction du facteur d'étalement $\alpha = 1$ à α_{max}.

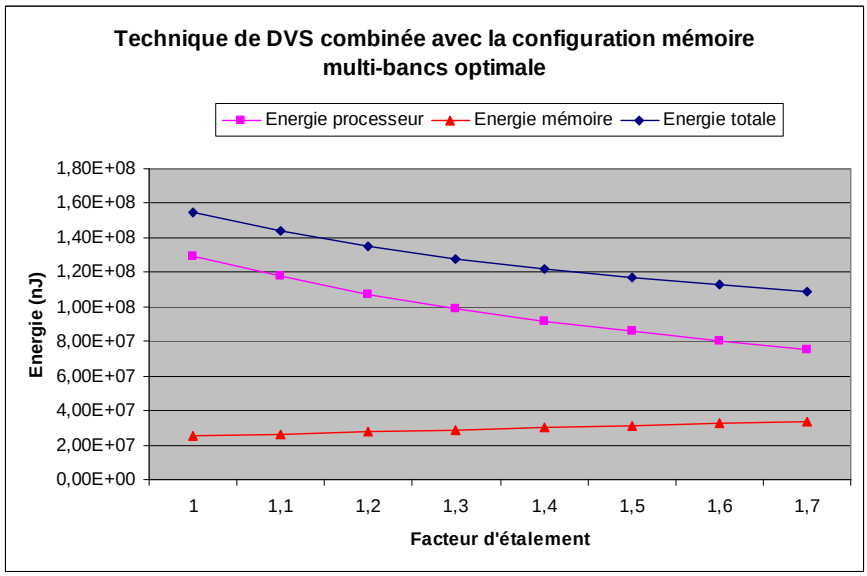
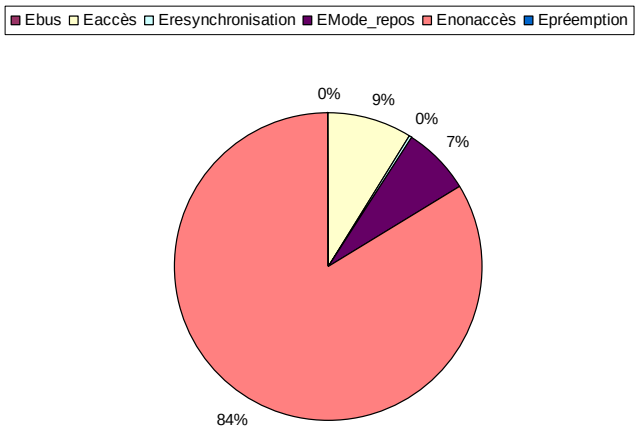


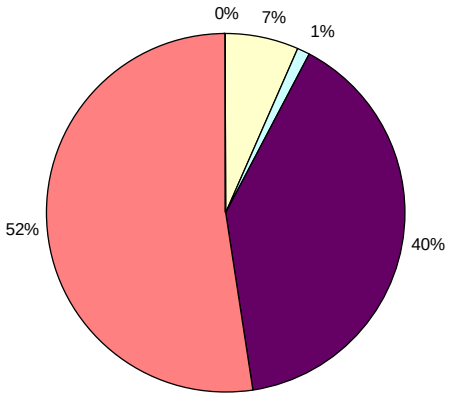
Figure 4.5. Ajustement conjoint en tension et fréquence avec l'architecture mémoire optimale

La figure 4.5 montre comme cela était attendu que, lorsque la technique de DVS est combinée avec une architecture mémoire multi-bancs, un gain plus important en consommation totale du système est obtenu par rapport à la technique de DVS employée avec une mémoire monolithique. En effet, un gain de 29,5% sur la consommation totale est obtenu comparé au gain de 13% obtenu avec une mémoire monolithique.

Cette augmentation du gain en énergie est due, en grande partie, à la diminution de la consommation $E_{nonaccès}$ mémoire en ne gardant actif que le banc concerné par les tâches en cours d'exécution par le processeur. En effet, la figure 4.6 illustre que la contribution de $E_{nonaccès}$ est passée de 84% pour l'architecture monolithique à 52% dans l'architecture mémoire multi-bancs optimale. Ces deux contributions sont calculées pour une valeur de α égale à 1.



(a)



(b)

Figure 4.6. Contribution de chaque énergie dans la consommation mémoire (a) dans une architecture monolithique (b) dans la mémoire multi-bancs optimale

Le gain énergétique total, obtenu avec la technique de DVS combinée d'une part à une mémoire monolithique et d'autre part à une mémoire multi-bancs, est tracé sur la figure 4.7. Ce gain varie de 15% pour un facteur d'étalement de 1 jusqu'à 35% pour un facteur d'étalement de 1,7 pour l'application considérée. L'intérêt de partitionner l'espace d'adressage en plusieurs bancs croît ainsi avec l'augmentation du facteur d'étalement.

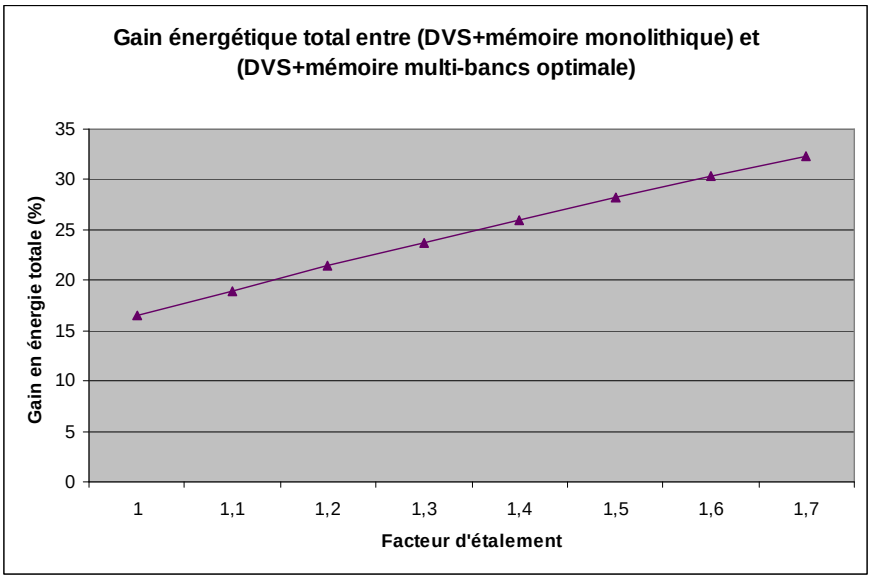


Figure 4.7. Gain en énergie totale entre (DVS + Mémoire Monolithique) et (DVS + mémoire multi-bancs optimale) en fonction du facteur d'étalement

La justification de notre choix d'adoption d'une architecture mémoire multi-bancs au lieu d'une classique mémoire monolithique pour minimiser la consommation totale lorsque la tension et la fréquence sont diminuées sur le processeur, est illustrée avec cet exemple. On s'intéresse dans les paragraphes suivants à étudier de façon plus détaillée la consommation de la mémoire multi-bancs ainsi que sa variation avec les paramètres identifiés dans le paragraphe 4.2 du chapitre 2.

1.4. Expérimentations sur une mémoire multi-bancs

Les expérimentations effectuées dans le paragraphe précédent ont montré que les mémoires multi-bancs sont intéressantes pour minimiser l'impact négatif de la technique de DVS appliquée au processeur sur la consommation mémoire. Dans cette partie, on se concentre sur l'étude de la consommation de la mémoire multi-bancs. La première expérimentation étudie la variation de la consommation mémoire avec le nombre de bancs. La deuxième consiste à tester les différents modes repos de la mémoire afin de choisir celui qui atteint les meilleurs gains d'énergie.

1.4.1. Comportement de la consommation mémoire avec le nombre de bancs

Deux comportements différents de la consommation mémoire en fonction du nombre de bancs sont constatés dans cette étude. Le premier comportement se traduit par une diminution de la consommation mémoire avec l'accroissement du nombre de bancs, et ce jusqu'à un nombre optimal, puis elle augmente à nouveau, à moins que ce nombre corresponde au nombre maximum de bancs possible (le nombre de tâches de l'application).

Le deuxième comportement se traduit par une augmentation de la consommation mémoire dès qu'un banc est ajouté à une architecture mémoire monolithique. Dans ce

cas, la mémoire monolithique est la moins consommatrice et une structure mémoire multi-bancs est inefficace pour réduire l'énergie. L'application, décrite par le tableau 3.4, est utilisée pour étudier ces deux comportements.

Par rapport au tableau 4.3, les périodes d'activation de certaines tâches ont été modifiées ce qui conduit à un ordonnancement RM différent. Il s'en suit que les valeurs des nombres d'accès à la mémoire M_i sont également différents sur l'hyperpériode.

Tableau 4.4. Application constituée de 6 tâches

Tâche	P_i (cycles)	c_i (cycles)	S_{Ti} (kbytes)	M_i
IDCT	250000	16131	33	193
FIR	1000000	33983	152	133
Fibcall	1000000	9536	27	114
Qsort	1000000	13309	31	97
FFT	5000000	515771	97	38404
ADPCM	10000000	2486633	133	4053

Par exploration exhaustive, la variation de la consommation mémoire en fonction du nombre de bancs est donnée sur la figure 4.8. Cette figure montre que l'ajout d'un banc à une architecture mémoire diminue la consommation jusqu'à atteindre une architecture à 4 bancs. Un gain d'énergie de 24% par rapport à une architecture monolithique est obtenu. Lorsque ce nombre de bancs est dépassé, la consommation mémoire augmente à nouveau.

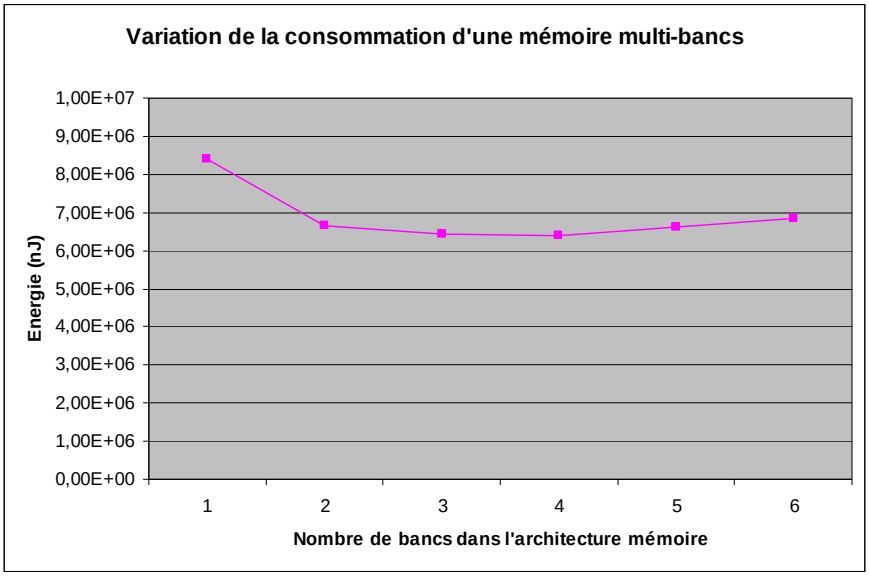


Figure 4.8. Variation de la consommation mémoire avec le nombre de bancs

L'analyse de ce comportement est détaillée sur la figure 4.9. Elle représente pour chaque nombre de bancs de l'architecture mémoire, la contribution des différentes énergies ($E_{accès}$, $E_{nonaccès}$, E_{Mode_repos}, $E_{resynchronisation}$, E_{bus}, $E_{préemption}$) dans la consommation mémoire totale.

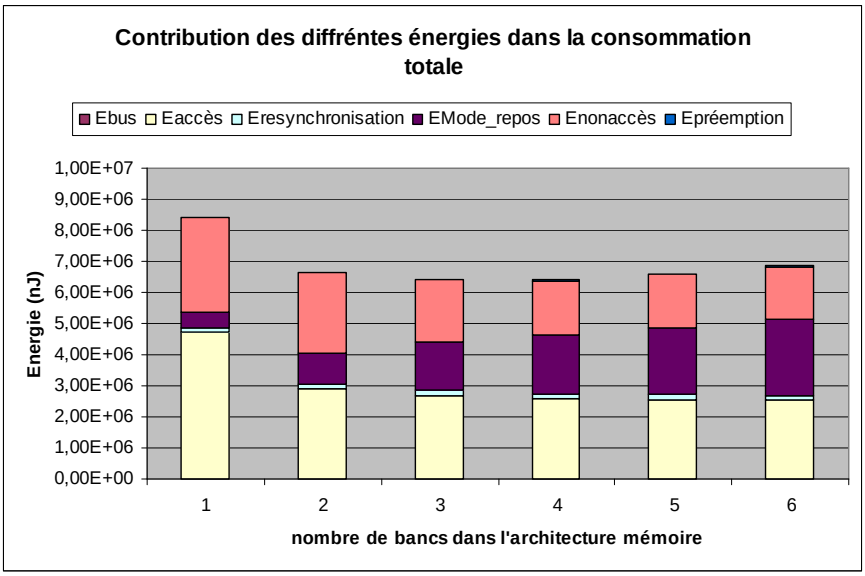


Figure 4.9. Contribution des différentes énergies dans la consommation mémoire totale

La figure 4.9 montre essentiellement qu'il y a des énergies croissantes avec le nombre de bancs et d'autres décroissantes. Les énergies décroissantes sont $E_{accès}$ et $E_{nonaccès}$. Inversement les énergies croissantes avec le nombre de bancs sont E_{Mode_repos}, E_{bus} et $E_{resynchronisation}$.

On définit $E_{décroissante}$ et $E_{croissante}$ comme suit :

$$E_{décroissante} = E_{accès} + E_{nonaccès}$$

$$E_{croissante} = E_{Mode\_repos} + E_{resynchronisation} + E_{bus}$$

A priori, les applications ayant une énergie $E_{décroissante}$ importante, c'est-à-dire les applications à fort taux d'accès mémoire ($E_{accès}$) ou les applications avec des temps d'exécution longs et des taux d'accès à la mémoire faibles ($E_{nonaccès}$), bénéficient le plus de structures mémoires multi-bancs pour réduire la consommation.

Pour identifier le deuxième comportement de la mémoire, des caractéristiques modifiées de cette application sont considérées et décrites dans le tableau 4.5. Les modifications du nombre d'accès mémoire par les tâches ainsi que des temps d'exécution plus faibles sont ici choisis arbitrairement pour révéler le comportement désiré.

Tableau 4.5. Application de 6 tâches modifiées

Tâche	P_i (cycles)	c_i (cycles)	S_{Ti} (kB)	M_i
IDCT	250000	6131	33	63
FIR	1000000	3983	152	33
Fibcall	1000000	2536	27	14
Qsort	1000000	5309	31	27
FFT	5000000	35771	97	2404
ADPCM	10000000	1286633	133	253

La contribution de chaque énergie dans la consommation mémoire totale, pour chaque nombre de bancs, est donnée par la figure 4.10.

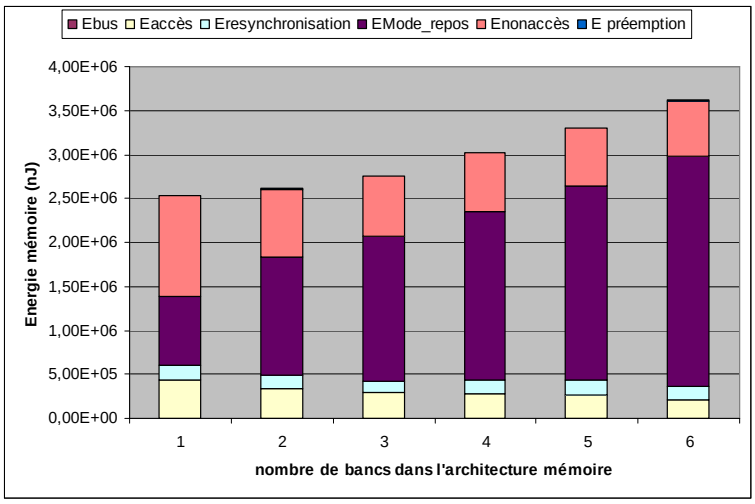


Figure 4.10. Contribution des différentes énergies dans la consommation mémoire totale

L'ajout d'un banc à une mémoire monolithique augmente les énergies croissantes E_{Mode_repos} et $E_{resynchronisation}$ et E_{bus} plus qu'il ne baisse les énergies croissantes $E_{accès}$ et $E_{nonaccès}$. Une consommation totale plus importante est alors obtenue. Dans ce cas, une mémoire monolithique est la moins consommatrice en énergie. Ce sont les applications à faible taux d'accès mémoire et à faible temps d'exécution (c'est-à-dire $E_{décroissante}$ faible) qui ne nécessitent pas a priori le partitionnement en plusieurs bancs de la mémoire.

Ces expérimentations justifient le choix du critère *Isolation(T_i)* de l'heuristique développée dans le chapitre 3. En effet, ce critère permet de réduire de façon efficace l'énergie décroissante en favorisant l'isolation de la tâche qui effectue de nombreux accès à la mémoire (lié à l'énergie $E_{accès}$) et qui oblige la mémoire à rester plus longtemps coactive avec le processeur (lié à l'énergie $E_{nonaccès}$). De la même manière, le critère permet aussi de contrôler l'augmentation des énergies croissantes en favorisant l'isolation de la tâche qui permet au banc initial de rester le plus longtemps en mode repos (lié à E_{Mode_repos}) et qui réduit le nombre de réveils de ce même banc initial (lié à $E_{resynchronsation}$)

1.4.2. Influence du mode faible consommation sur la consommation mémoire et les performances

Dans ce paragraphe, l'influence du mode repos mémoire utilisé sur la réduction de la consommation mémoire et sur les performances est étudiée afin de choisir parmi ceux disponibles celui qui permet d'obtenir les meilleurs gains en énergie sans dégradation significative des performances. A titre d'exemple, considérons les 4 tâches décrites dans le tableau 4.6.

Tableau 4.6. Application de 4 tâches

Tâche	P_i (cycles)	c_i (cycles)	S_{Ti} (kB)	M_i
Matmul	600000	13985	29	97
FIR	600000	33983	152	58
FFT	2000000	515771	97	315
ADPCM	6000000	2486633	133	1801

Il s'agit de trouver la configuration mémoire optimale et les gains d'énergie en considérant à chaque fois un mode faible consommation de la mémoire différent. On commence par le mode repos le moins profond (*Standby*), ensuite le mode *Nap* et finalement le mode le plus profond (*Power-Down*). Dans chaque cas, la consommation des différentes configurations est évaluée ainsi que la charge processeur U afin de tester l'impact du mode repos sur les performances. Le test de faisabilité de l'ordonnancement RM est calculé en considérant les pénalités temporelles de réveil des bancs.

L'ordonnançabilité du système correspondant à une allocation mémoire avec un nombre de bancs k égal au nombre de tâches N est testée avec l'équation 3.1 où chaque exécution d'une tâche de l'application nécessite un réveil de banc.

$$U = \sum_{i=1}^{N} \frac{c_i + t_{resynchronisation}}{P_i} \leq 4 \times \left(2^{\frac{1}{4}} - 1 \right) = 0{,}7568 \qquad [3.1]$$

L'ordonnançabilité du système est testée, pour une allocation mémoire donnée de k bancs, avec l'équation 3.2.

$$U = \sum_{i=1}^{N} \frac{c_i + \sum_{j=1}^{k} N_{resynchronisation\_bj} \times t_{resynchronisation}}{P_i} \leq 4 \times \left(2^{\frac{1}{4}} - 1 \right) = 0{,}7568 \qquad [3.2]$$

- **Mode *Standby***

Une étude de la consommation mémoire multi-bancs et des performances obtenues lorsque seulement le mode *Standby* est employé est réalisée dans cette section. Dans le modèle de mémoire considéré (tableau 2.1 du chapitre 2), le mode *Standby* est caractérisé par une consommation $E_{0Standby}$ = 0,83 nJ, une énergie par réveil $E_{0resynchronisation}$ = 10 nJ et une pénalité temporelle de réveil $t_{resynchronisation}$ = 2 cycles.

Deux cycles de pénalité de resynchronisation sont ajoutés aux temps d'exécution c_i des tâches à chaque exécution de la tâche. La charge de travail processeur U est maintenant égale à 0,752279. Le système est toujours ordonnançable avec l'ordonnancement RM (U< 0,7568).

Pour la consommation, les énergies des différentes configurations sont présentées sur la figure 4.11. La solution optimale est obtenue avec une mémoire monolithique (1 banc). Une augmentation importante de l'énergie de repos est constatée lorsqu'un deuxième banc est ajouté à l'architecture comme indiquée sur la figure 4.11.

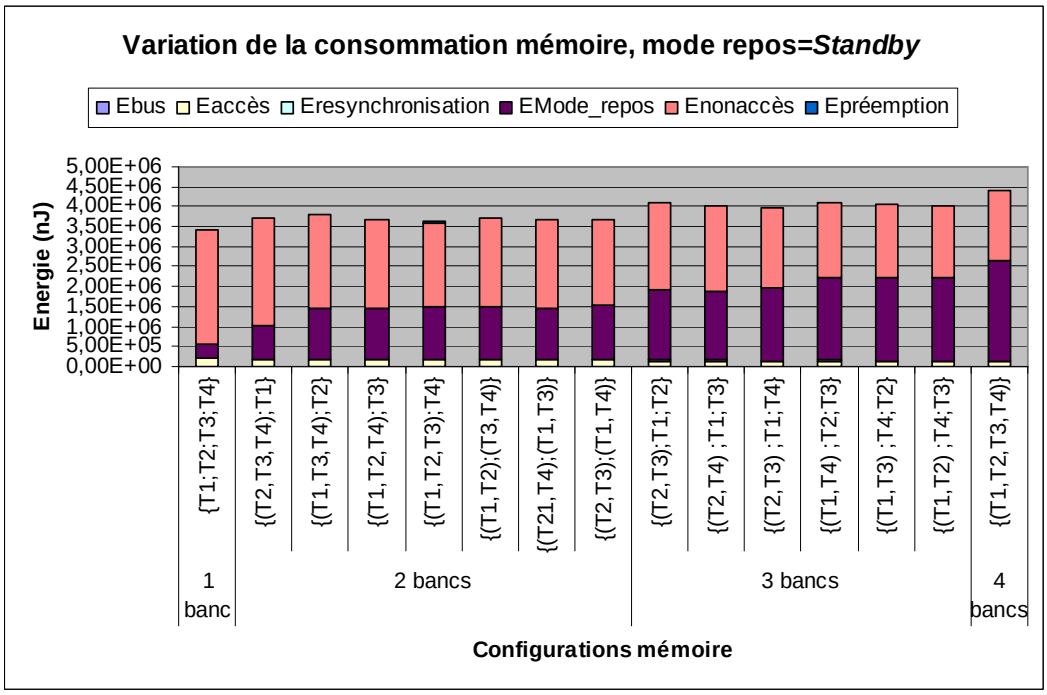


Figure 4.11. Contributions de chaque composante dans la consommation totale des différentes configurations mémoire avec le mode *Standby*; T_1= Matmul; T_2=FIR; T_3= FFT; T_4=ADPCM

La consommation relativement élevée du mode repos *Standby* ($E_{0Standby}$=0,83 nJ), comparée aux valeurs de consommations des autres modes faible consommation (*Nap* et *Power-Down*), ne permet donc pas de baisser la consommation de la mémoire monolithique par ajout de bancs à l'architecture. La contribution de l'énergie de resynchronisation dans la consommation mémoire totale reste très faible par rapport aux autres énergies.

- **Mode *Nap***

La même expérimentation est refaite mais cette fois en choisissant le mode *Nap* comme mode faible consommation qui correspond à un niveau d'énergie plus faible que le précédent. Le mode *Nap* est caractérisé par une consommation E_{0Nap}=0, 32nJ et des pénalités de resynchronisation $E_{0resynchronisation}$=1000nJ et $t_{resynchronisation}$ = 30 cycles (tableau 2.1 du chapitre 2).

L'étude de performance montre que le système est toujours ordonnançable (U = 0,7524 inférieur à 0,7568) quand les 30 cycles de resynchronisation d'un banc en mode *Nap* sont considérés.

Une structure mémoire constituée de 3 bancs avec une allocation de tâches {(Matmul, FIR) ; FFT ; ADPCM} est optimale quand le mode *Nap* est utilisé (figure 4.12). Un gain en énergie de 12,5% est obtenu par rapport à la solution à 1 banc.

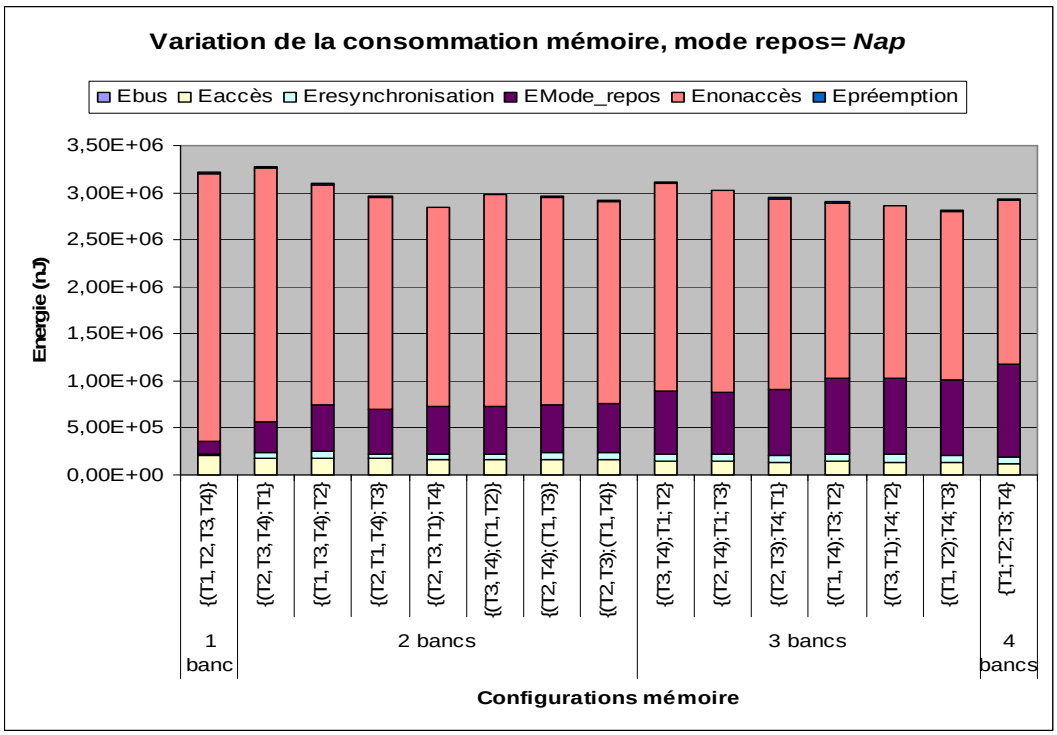


Figure 4.12. Contributions de chaque composante dans la consommation totale des différentes configurations mémoire avec le mode *Nap*; T_1= Matmul; T_2=FIR; T_3= FFT; T_4=ADPCM

De manière attendue, l'augmentation de la consommation du mode repos est plus faible par rapport à celle de la figure 4.11 (correspondante au mode *Standby*), en contre partie, l'énergie de réveil devient plus significative.

- **Mode *Power-Down***

Finalement, le mode *Power-Down* est évalué sur le même exemple. Ce mode est caractérisé par une consommation $E_{0Power\text{-}Down}$=0,005nJ et des pénalités de resynchronisation pour revenir en mode *Active* de $E_{0resynchronisation}$=3800nJ et $t_{resynchronisation}$= 9000 cycles (tableau 2.1 du chapitre 2).

La prise en compte des 9000 cycles processeur de latence de réveil dans les temps d'exécution des tâches, augmente la charge processeur U à 0,7882. Le système peut ne pas être ordonnançable avec la technique RM (U = 0,7882 > 0,7568).

La consommation mémoire des différentes configurations représentées sur la figure 4.13, montre que la configuration monolithique est la moins consommatrice.

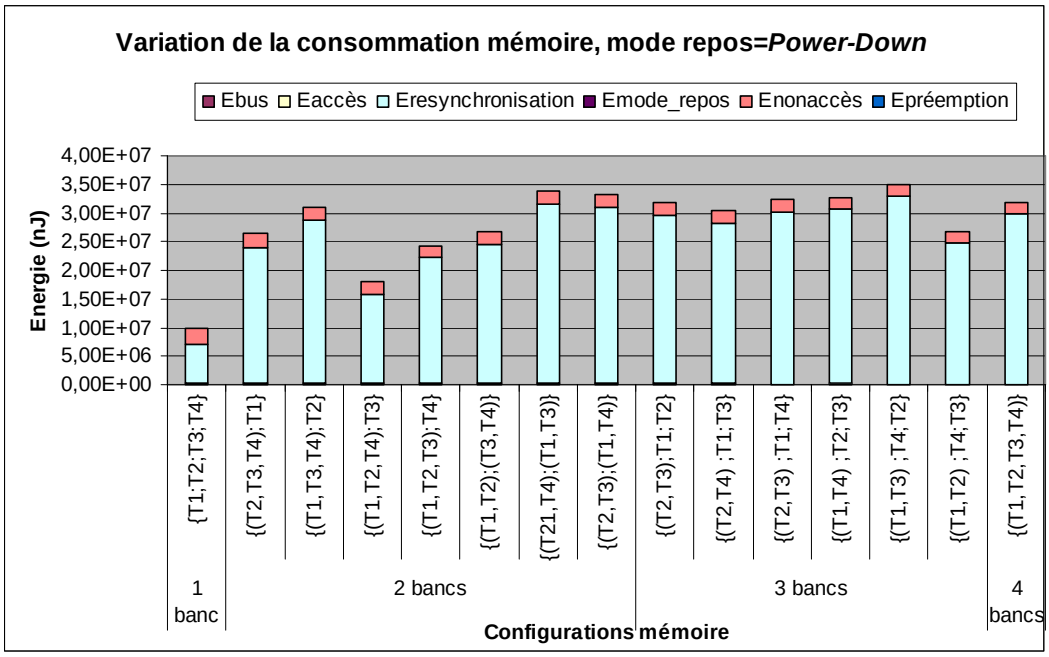


Figure 4.13. Contributions de chaque composante dans la consommation totale des différentes configurations mémoire avec le mode *Power-Down*; T_1= Matmul; T_2=FIR; T_3= FFT; T_4=ADPCM

A cause de l'énergie de réveil élevée (3800nJ) du mode *Power-Down*, la dissipation d'énergie due aux réveils des bancs devient très importante et pénalise l'intérêt d'ajouter des bancs à l'architecture mémoire.

En conclusion de cette étude, on peut déduire que le mode *Standby*, a un faible impact sur la performance, mais à cause de la faible réduction de consommation induite par ce mode comparée au mode actif, des gains d'énergie significatifs ne peuvent pas être obtenus (dans notre cas pas d'optimisation avec ce mode). A l'inverse, le mode *Power-Down* peut avoir un impact important sur les performances (dans notre cas le système n'est à priori plus ordonnançable) mais il possède une énergie en mode faible consommation très faible. Cependant l'énergie de réveil étant importante, elle réduit la possibilité d'éventuelles optimisations (dans notre cas l'énergie de réveil induite par l'ajout d'un banc produit une augmentation de la consommation totale). Le bon compromis entre consommation et pénalités de réveil (temps et énergie) du mode *Nap*, permet dans les exemples traités d'obtenir des meilleurs gains. Ceci confirme l'étude réalisée dans (Lebeck *et al.*, 2000) (Delaluz *et al.*, 2001) sur des applications mono-tâches.

En se basant sur cette étude, le mode *Nap* est donc utilisé par défaut comme mode faible consommation dans toutes nos expérimentations. Cependant, les autres modes pourraient éventuellement apporter des gains en énergie en fonction de la nature des applications considérées.

2. Expérimentations sur une application multimédia

Dans cette partie, la variation de la consommation mémoire multi-bancs est étudiée sur une application plus conséquente associant une fonction de communication sans fil (GSM) et multimédia (MPEG-2). Cette application est construite à partir des deux fonctions GSM et MPEG-2 car elles sont disponibles aisément. Il aurait été plus réaliste de considérer la norme GPRS voire EDGE comme méthode de communication sans fil afin d'être mieux en correspondance avec le débit nécessité par la décompression MPEG-2. Pour des raisons de performances, ce type d'application nécessite généralement plus d'une unité de traitement pour une exécution temps réel. La description de l'application est présentée dans la section 2.1, et celle de la plateforme cible (OMAP) dans la section 2.2. Dans la section 2.3, le partitionnement ainsi que l'ordonnancement de l'application sur l'architecture OMAP sont décrits. Dans la section 2.4, les caractéristiques des différentes tâches sont données. Dans la section 2.5, les résultats d'expérimentations sont présentés.

2.1. Description de l'application multimédia

L'application multimédia considérée est composée d'une application de communication mobile (la norme GSM) décrite dans le paragraphe 2.1.1 et d'une application de décodage vidéo MPEG-2 décrite dans le paragraphe 2.1.2.

2.1.1. Description de l'application GSM

- Description de l'application GSM

Du microphone de l'appareil émetteur au haut-parleur de l'appareil récepteur, la voix du locuteur subit toute une série de traitements permettant sa transmission jusqu'à l'oreille de l'auditeur. Nous décrivons par la suite les différentes étapes de la chaine de traitement du signal utilisées en GSM (Auslander *et al.*, 1994).

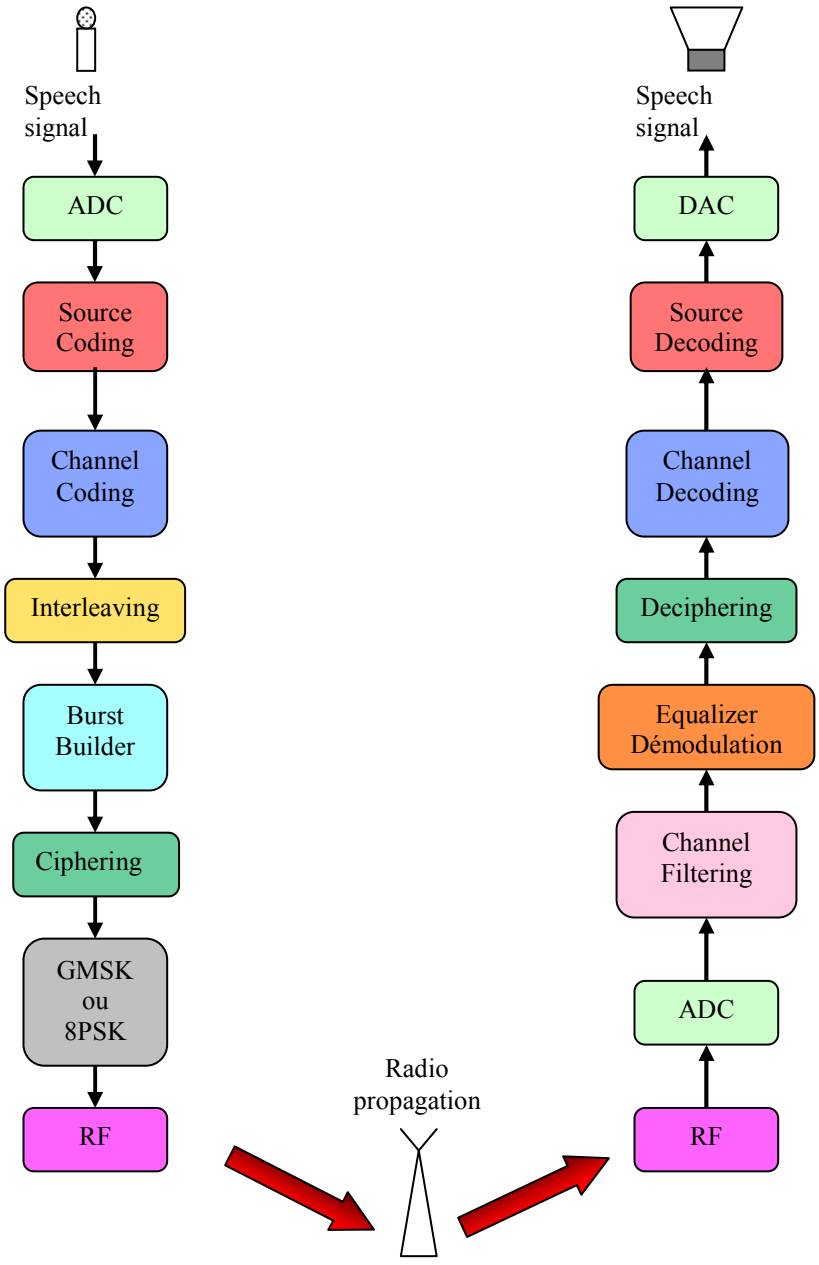


Figure 4.14. Chaîne de traitement du signal du GSM

La figure 4.14 présente les blocs fonctionnels d'un système émetteur-récepteur typique, tel qu'il peut être défini dans les téléphones portables de 2ème et 3ème génération actuellement sur le marché. Cette chaîne se compose d'une partie montante de codage et d'une partie descendante de décodage.

- **Analog to Digital Conversion (ADC)**

La bande de fréquence de la voix étant comprise entre 300Hz et 3400Hz, le signal est échantillonné à 8 kHz ce qui conduit à un débit de 64 kbps. L'ADC délivre ainsi 160 échantillons au codeur de parole toutes les 20ms, période pendant laquelle un signal de parole est considéré comme stationnaire.

- **Speech Coding (SC)**

Afin de réduire le débit résultant de 64 kbps et ainsi augmenter le nombre d'utilisateurs pour un canal radio ou la bande est une ressource rare, le signal de parole est compressé. En mode *full-rate* (FR) par exemple, un débit de 13 kbps (soit 260 bits) est obtenu après compression d'une trame de parole de 20ms. Le codeur de parole tire avantage de la redondance inhérente d'un signal de parole afin de réduire le débit. Le codeur FR compresse le signal selon la norme RPE/LTP (*Rectangular Pulse Excited Linear Predictive Coding with Long Term Prediction*) générant 260 bits par trame de 20ms.

- **Channel Coding (CC)**

Le codage canal ajoute au contraire de la redondance au signal afin de protéger celui-ci des aléas d'une transmission par voie radio. Cette information supplémentaire permettra de détecter et éventuellement corriger les erreurs introduites lors de la transmission. Le codage canal génère 456 bits par trame en FR.

- **Interleaving**

L'entrelacement consiste à repartir les 456 bits sur 8 *bursts* de 57 bits. L'objectif est de rendre plus aléatoire les positions des erreurs de transmission radio. Les symboles codés sont ainsi mélangés avant leur transmission afin d'augmenter les performances de correction en réception des codes correcteurs d'erreurs. L'inconvénient de l'entrelacement est le délai supplémentaire introduit dans la chaine de transmission (et réception).

- **Burst Building (BB)**

Le bloc BB ajoute une *training sequence*, des *tail* bits et éventuellement des bits de gardes aux symboles codés. Le burst ainsi construit (156 bits pour un *burst* normal) est prêt à être modulé et transmis sur le canal radio

- **Ciphering**

Les données entre le mobile et la station de base sont protégées grâce à un chiffrement ou cryptage de type A5. Il permet de garantir la confidentialité des informations personnelles de l'utilisateur.

- **Modulation GMSK**

Un *burst* est modulé pour adapter le signal numérique au canal de transmission. La modulation utilisée en GSM est une modulation de fréquence à enveloppe constante appelée GMSK (*Gaussian Minimum Shift Keying*). Elle fournit un bon compromis entre

efficacité spectrale, robustesse contre les bruits, amplification et contraintes d'implémentation.

Les deux tâches, *Interleaving* et modulation GMSK, n'ont pas été considérés dans les expérimentations à cause de non disponibilité de leurs codes.

2.1.2. Description de l'application MPEG-2

La norme MPEG-2 est capable de coder un flux vidéo standard entre 3-15Mb/s et un flux haute définition entre 15-30 Mb/s. Dans cette étude, uniquement la partie décodage du MPEG-2 est considérée. Dans les algorithmes MPEG-2, les images sont divisées en macroblocs. Chaque macrobloc est organisé en 6 blocs 8×8 et chaque bloc est décodé selon le schéma de la figure 4.15 (Pazos *et al*., 2004).

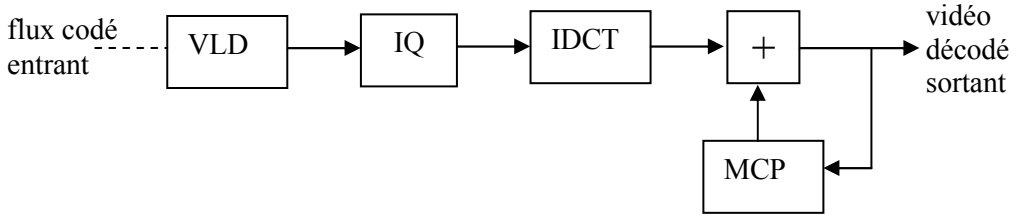


Figure 4.15. Décodeur MPEG-2

Les tâches qui permettent la réalisation les opérations de décodage élémentaires sont : VLD (Décodage en longueur variable), IQ (Quantification inverse), IDCT (Transformée Cosinus Discrète Inverse), ADD (Addition) et la MCP (Prédiction de Compensation de Mouvement).

2.2. Description de la plateforme OMAP

La plateforme OMAP (*Open Multimedia Application Plateform*), présentée sur la figure 4.16, est un exemple d'architecture multiprocesseur de type maître-esclave. Elle intègre un processeur DSP de traitement numérique du signal qui agit sur événements (commandes/contrôles) reçus du processeur maître ARM. Elle a été retenue par de nombreux industriels pour le développement des téléphones de génération 2G, 2.5G et 3G (GSM, GPRS, EDGE) et du multimédia : MMS (messages multimédia), le téléchargement et l'écoute de musique, les échanges de photos, la vidéo conférence, les jeux 3D. Chaque processeur possède ses mémoires SRAM internes (cache et/ou RAM). Une mémoire partagée SRAM on-chip de 192 kB sert de communication et de partage de données entre les processeurs ARM et DSP. Une mémoire externe synchrone (SDRAM, RDRAM, DDR SDRAM) ou asynchrone (FPM *Fast Page mode*, EDO : *Extended Data Out*) permet d'offrir un espace de stockage plus important.

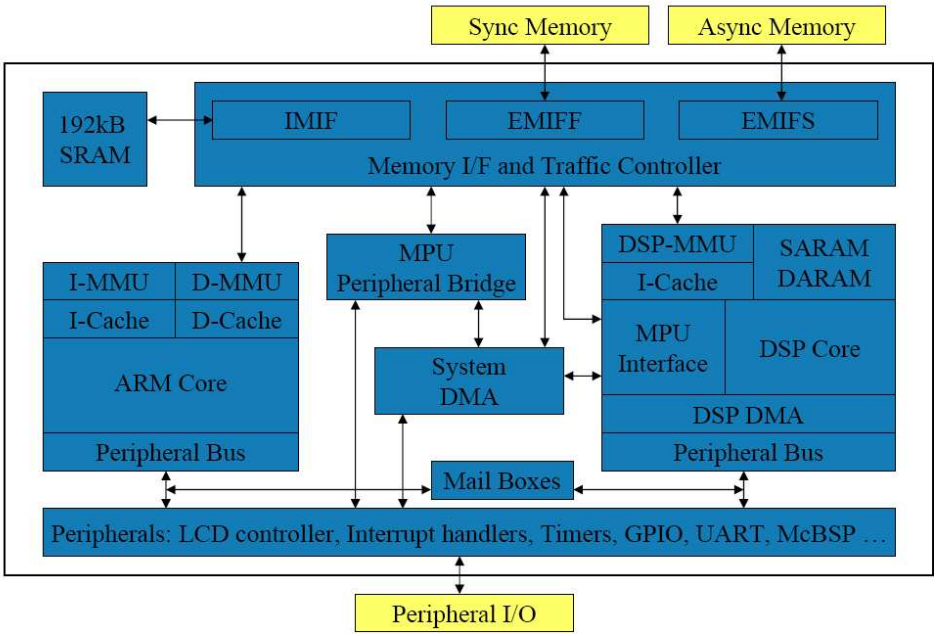


Figure 4.16. La plateforme OMAP de Texas Instruments

Les caractéristiques des deux processeurs DSP TMS 320C55x et ARM9 TI 925T TDMI et de la mémoire principale de la plateforme OPMAP 1510 sont résumées dans le tableau 4.7.

Tableau 4.7. Description de la plateforme OMAP 1510 (www.ti.com)

Composant	**Caractéristiques**
TMS 320C55x DSP	- 192 MHz, Voltage: 1,5V nominal - 64 KB on-chip dual-access RAM (DARAM) - 96 KB on-chip single-access RAM (SARAM) - 16 KB I-cache, 8 KB D-cache.
TI925T ARM9 TDMI	- 168 MHz, Voltage: 1,5V nominal - 16KB I-cache; 8KB D-cache - 192 KB SRAM mémoire interne partagée
Mémoire principale	RDRAM, latence = 40 cycles.

2.3. Partitionnement et ordonnancement de l'application multimédia

Le partitionnement des différentes tâches des deux applications sur le processeur de traitement de signal DSP ou sur le processeur généraliste ARM est déduit de la nature des tâches. Si la tâche ne renferme que du traitement, elle est exécutée a priori sur le processeur DSP. Par contre si la tâche renferme plus de contrôles que de traitements, le processeur ARM est plus adapté à son exécution.

Ainsi les tâches *Speech Coding* (SC), *Coding channel* (CC) de la voie montante du GSM, les tâches *Channel Decoding* (CD), *Speech Decoding* (SD) de la voie descendante du GSM et les tâches IDCT, MC, ADD du décodeur MPEG-2 sont exécutées par le

processeur DSP. Inversement, les tâches BB et EQ du GSM, VLD, IQ du MPEG-2 qui renferment moins de traitements sont allouées au processeur ARM. Bien évidement, d'autres partitions des tâches entre les deux processeurs sont possibles. Elles peuvent conduire à des solutions plus au moins efficaces. La recherche du meilleur partitionnement des tâches entre les processeurs n'est pas ici l'objectif.

Les normes relatives aux deux applications GSM et MPEG-2 définissent les périodes P_i ainsi que les dépendances temporelles entre les différentes tâches qui impliquent un ordonnancement des tâches sur les deux processeurs. L'ordonnancement des tâches est effectué sur une période de 20 ms entre deux codages de parole (SC). Sur cette fenêtre de 20ms, 4 blocks 8×8 pixels sont décodés. L'ordonnancement obtenu est présenté sur la figure 4.17.

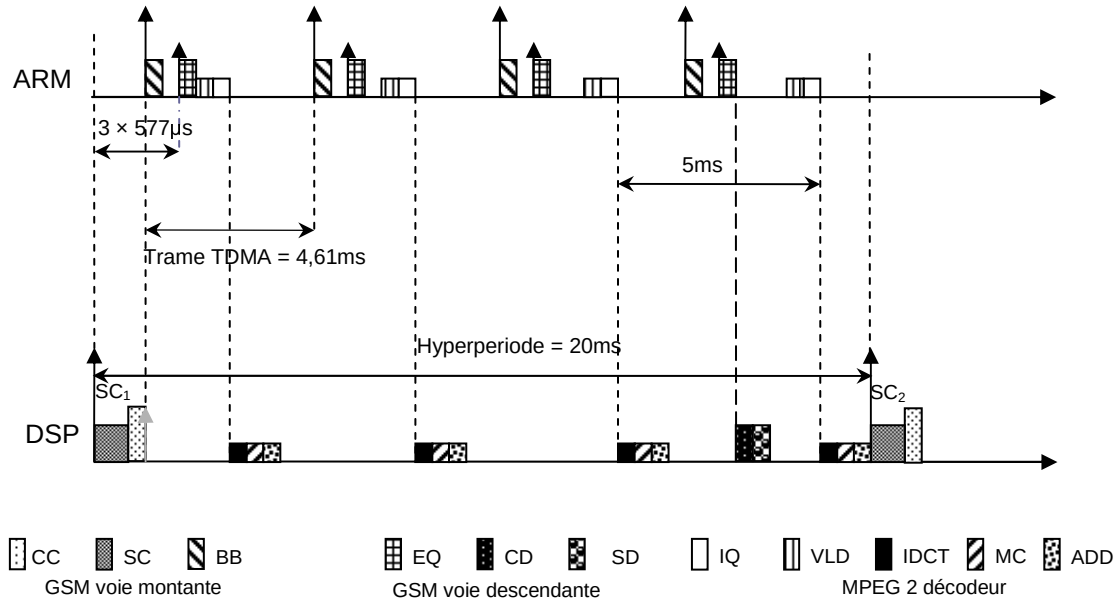


Figure 4.17. Ordonnancement de l'application GSM et décodeur MPEG-2 sur une plateforme de type OMAP pendant une fenêtre de 20 ms.

A partir de l'ordonnancement construit, la matrice de successivité S_{10} et le vecteur du nombre d'exécutions N_{exeTi} des différentes tâches sont déduits. On considère que, T_1 = Burst Builder, T_2 = Equalizer, T_3 = Motion Compensation couplée avec la tâche ADD, T_4 = Variable Length Coding, T_5 = Quantification Inverse, T_6 = Inverse Discret Cosine Transform, T_7 = Speech Coding, T_8 = Speech Decoding, T_9 = Coding Channel et T_{10} = Channel Decoding.

$$S_{10} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ & & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ & & & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ & & & & 0 & 4 & 0 & 0 & 0 & 0 \\ & & & & & 0 & 0 & 0 & 0 & 0 \\ & & & & & & 0 & 0 & 1 & 0 \\ & & & & & & & 0 & 0 & 1 \\ & & & & & & & & 0 & 0 \\ & & & & & & & & & 0 \end{bmatrix} ; \; N_{exeTi} = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \\ 4 \\ 4 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

2.4. Caractéristiques des tâches multimédia

Le code source du décodeur MPEG-2, développé par le groupe *Software Simulation Group*, a été utilisé (http://www.mpeg.org/MPEG/MSSG/) pour obtenir par simulation avec l'outil SimpleScalar les caractéristiques des tâches présentées dans le tableau 4.8.

Tableau 4.8. Caractéristiques des tâches du décodeur MPEG-2

décodeur MPEG-2	P_i (ms)	c_i (cycles)	S_{Ti} (kB)	M_i
MC	5	87836	213	888
VLD	5	58783	281	1465
IQ	5	12922	29	355
IDCT	5	16131	33	193

Les caractéristiques des tâches de l'application GSM sont issues de (Auslander *et al*., 1994) et présentées sur le tableau 4.9.

Tableau 4.9. Caractéristiques des tâches du GSM

GSM	P_i (ms)	c_i (cycles)	S_{Ti} (kB)	M_i
BB	4,615	2000	2,15	456
EQ	4,615	23000	5,27	502
SC	20	36000	1,43	8863
SD	20	10000	1,64	2435
CC	20	6875	1,4	1700
CD	20	15140	1,45	3626

2.5. Expérimentations et résultats

Pour une mémoire monolithique sans mode repos, la contribution de chaque énergie obtenue suivant les modèles développés fait apparaitre que, l'énergie consommée par la mémoire hors accès est prépondérante. Elle représente 97% de la consommation totale alors que l'énergie due aux accès ne représente que 3%. L'utilisation d'une architecture multi-bancs avec gestion des modes repos sera bénéfique pour réduire l'énergie due aux périodes sans accès et transformer une partie de cette dernière en énergie de repos.

Comme l'application est constituée de 10 tâches, une architecture mémoire avec au maximum 10 bancs est explorée. Ceci conduit à B_{10} = 115975 configurations mémoire possibles. Sur la figure 4.18, uniquement la configuration mémoire la moins consommatrice, pour chaque nombre de bancs constituant la mémoire, est représentée. Ajouter un banc à l'architecture mémoire permet de réduire la consommation totale jusqu'à atteindre une architecture avec un nombre de bancs égal à 3. Un gain en énergie de 74% est obtenu par rapport à une architecture mémoire monolithique sans mode repos et un gain d'énergie de 14,7% par rapport à une architecture mémoire monolithique avec mode repos (le mode *Nap* est employé).

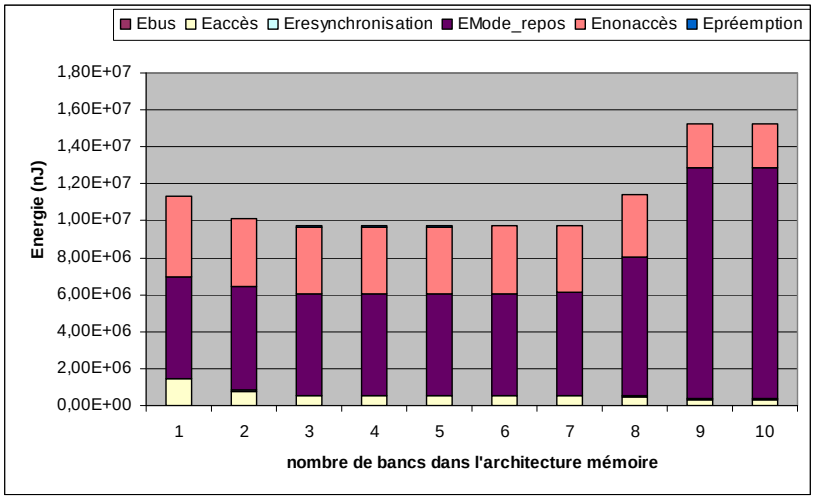


Figure 4.18. Variation et contribution des différentes énergies dans la consommation totale d'une mémoire multi-bancs pour l'application multimédia.

Nous notons que l'énergie de réveil et celle consommée sur le bus mémoire sont très faibles par rapport aux énergies d'accès, de non-accès et de repos.
La configuration mémoire optimale et l'allocation des tâches aux bancs sont décrites sur la figure 4.19.

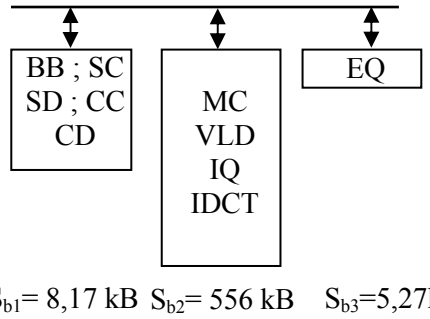


Figure 4.19. Configuration mémoire multi-bancs optimale pour l'application multimédia

De manière évidente, la charge de travail des processeurs ARM et DSP n'est pas très importante. Par conséquent, des réductions de la fréquence et de la tension d'alimentation des processeurs pourraient être appliquées, si ces processeurs supportaient cette possibilité. De même d'autres fonctions pourraient être allouées à ces processeurs tant que leur charge de travail le permet (téléchargement de jeux, décodage audio AC3). Dans ce cas la solution de la figure 4.19 ne serait sans doute plus optimale.

3. Allocation du système d'exploitation temps réel (RTOS)

Les applications ciblées dans nos travaux sont des applications multi-tâches temps réel et préemptives. Ce type d'applications nécessite un système d'exploitation temps réel pour la gestion des différentes tâches (ordonnancement, préemption, changement de contexte, synchronisation …). Les instructions ainsi que les données propres au système d'exploitation résident nécessairement dans l'un des bancs mémoire. A chaque appel par une tâche à un service du RTOS, un ensemble d'accès au banc mémoire où réside ce dernier est effectué. Cependant, dans toutes les expérimentations précédemment réalisées, l'emplacement du système d'exploitation temps réel RTOS dans la mémoire principale multi-bancs a été négligé. Cette partie du chapitre est consacrée aux expérimentations effectuées afin d'allouer le système d'exploitation aux bancs mémoire. Deux ensembles de tâches sont considérés. Le premier contient 6 tâches et le deuxième est constitué de 4 tâches issues des applications test du tableau 4.2.

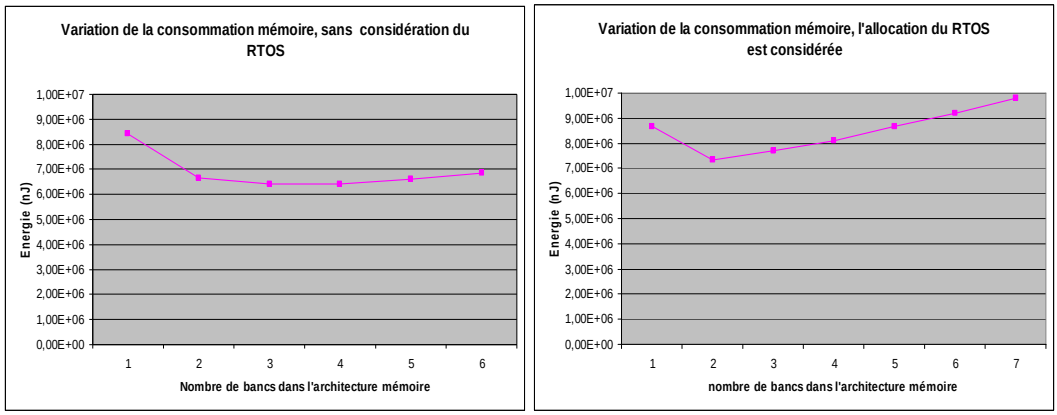
3.1. Expérimentations avec un ensemble de 6 tâches

Une application de 6 tâches déjà utilisée dans le paragraphe 1.4.1 et décrite dans le tableau 4.4 est utilisée ici. Les paramètres liés au système d'exploitation temps réel (taille S_{RTOS}, temps d'exécution c_{OS}, nombre d'accès mémoire M_{RTOS}, nombre de services R_{Ti} appelés par la tâche T_i) sont ajoutés (tableau 4.10).

Tableau 4.10. Caractéristiques de l'application de 6 tâches avec le RTOS

Tâche	P_i (cycles)	c_i (cycles)	S_{Ti} (kB)	M_i	R_{Ti}
IDCT	250000	16131	33	193	80
FIR	1000000	33983	152	133	20
Fibcall	1000000	9536	27	114	20
Qsort	1000000	13309	31	97	20
FFT	5000000	515771	97	38404	18
ADPCM	10000000	2486633	133	4053	44
RTOS	-	30753	32	300	-

Sur la figure 4.20, les variations de l'énergie mémoire en fonction du nombre de bancs sont illustrées. Sur la figure 4.20.a, l'allocation des données et des instructions propres au RTOS est négligée, contrairement au cas de la figure 4.20.b. Dans ce dernier cas, on considère que le banc contenant le RTOS est réveillé à chaque appel de l'un de ses services par les tâches qui ne sont pas allouées à ce même banc.



(a) (b)

Figure 4.20. Variation de l'énergie mémoire avec le nombre de bancs (a) sans la considération du RTOS, (b) l'allocation du RTOS est pris en compte.

La première remarque est que le nombre de bancs maximum est passé de 6 à 7 bancs du fait que le RTOS est considéré comme une tâche dans notre approche.

La deuxième remarque est qu'avec le système d'exploitation, la configuration mémoire optimale du point de vue énergétique est obtenue avec une architecture à 2 bancs (figure 4.21) au lieu d'une configuration avec 4 bancs lorsque l'allocation du RTOS est ignorée. Pour expliquer ce changement de configuration mémoire, on représente sur la figure 4.22 la contribution de chaque énergie dans la consommation mémoire totale pour chaque configuration et dans les deux cas : avec et sans allocation du RTOS.

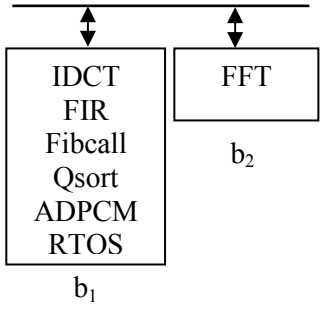
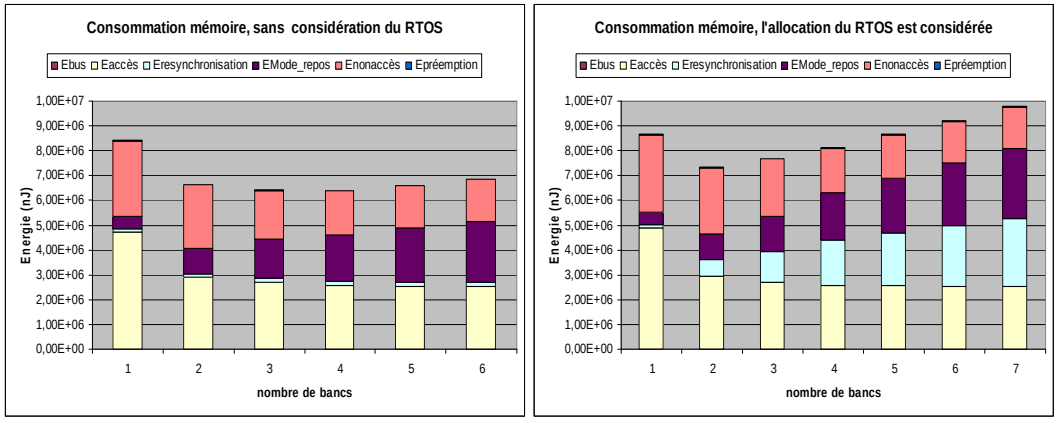


Figure 4.21. Allocation et configuration mémoire optimale en considérant le RTOS



(a) (b)

Figure 4.22. La contribution des énergies mémoire, (a) en négligeant le RTOS, (b) en tenant compte du RTOS

Sur la figure 4.22.b, on constate une augmentation importante de l'énergie de resynchronisation des bancs et plus précisément du banc où réside le RTOS. Une telle variation de l'énergie de réveil inhibe rapidement l'intérêt d'ajouter un banc à l'architecture mémoire ce qui explique la diminution du nombre de bancs de la configuration mémoire optimale (2 bancs contre 4 bancs sans l'allocation du RTOS).

Du point de vue performance, 18 réveils supplémentaires du banc b_1 se produisent lors de l'exécution de la tâche FFT (tableau 4.10) suite aux appels effectués par cette dernière aux services du RTOS.

Avec un temps de réveil de 30 cycles à partir du mode *Nap*, une pénalité de réveil du banc b_1 relativement importante de 540 cycles (18 × 30) liée aux appels des services du RTOS par la tâche FFT, nous a amené à tester le cas où le RTOS est alloué à un banc mémoire spécifique qui demeure actif en permanence.

La configuration mémoire optimale dans ce cas, est alors identique à celle obtenue avec 4 bancs sans considérer le RTOS (4.20.a) et à laquelle un banc réservé au RTOS est ajouté (figure 4.23).

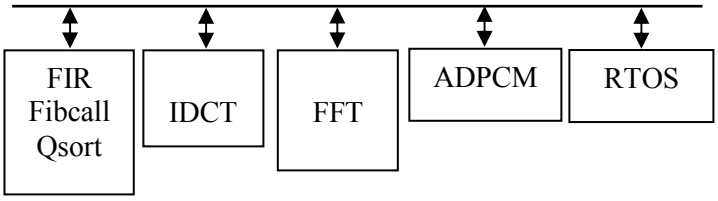


Figure 4.23. Allocation optimale dans le cas où le RTOS est seul dans un banc

Par rapport à la solution à 4 bancs de la figure 4.20.a, la consommation obtenue avec le RTOS est augmentée de 68%, cette augmentation est due au banc additionnel associé au RTOS.

Par ailleurs, une consommation supérieure de 63% est obtenue par rapport à la consommation de la configuration mémoire optimale à 2 bancs de la figure 4.22 ce qui illustre que des écarts de consommation important peuvent être obtenus suivant les applications et les configurations mémoire associées. Du point de vue performance, on peut remarquer que le RTOS ne rajoute aucune pénalité due aux réveils dans la solution de la figure 4.23.

La configuration mémoire décrite par la figure 4.21 reste alors la moins consommatrice même si un nombre plus important de réveils du banc où réside le RTOS est effectué. Un autre exemple illustratif de ces résultats est présenté dans le paragraphe suivant qui aborde la validation de l'heuristique développée au chapitre 3.

3.2. Expérimentation avec un ensemble de 4 tâches

Un deuxième exemple tiré de (Lee *et al.*, 2001) et décrit dans le tableau 4.11, est considéré afin d'étudier la consommation mémoire lors de l'allocation du RTOS dans une structure mémoire multi-bancs.

Tableau 4.11. Caractéristiques des 4 tâches avec considération du RTOS (Lee *et al.*, 2001)

Tâche	P_i (cycles)	c_i (cycles)	S_{Ti} (kB)	M_i	R_{Ti}
FFT	320000	60234	96	280	2100
LUD	1120000	255998	38	364	1100
LMS	1920000	365893	32	474	1050
FIR	6000000	557589	152	405	756
RTOS	-	223971	64	6018	-

La variation de la consommation mémoire en fonction du nombre de bancs de la structure mémoire est illustrée sur la figure 4.24.a, sans prise en compte de l'allocation du RTOS puis avec prise en compte de l'allocation du RTOS sur la figure 4.24.b.

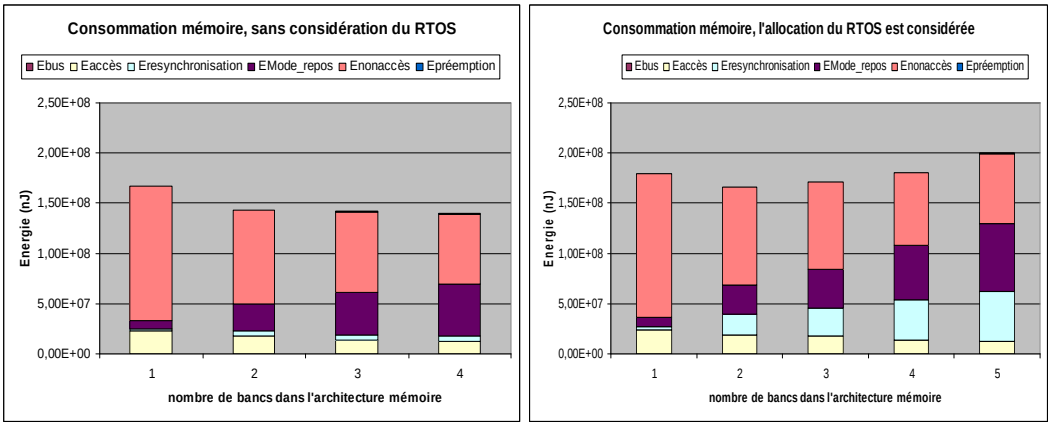


Figure 4.24. Variation de l'énergie mémoire en fonction du nombre de bancs, (a) en négligeant le RTOS, (b) en tenant compte du RTOS

Cet exemple illustre à nouveau que l'activité due au RTOS conduit à une solution optimale (à 2 bancs) avec un nombre de bancs plus faible par rapport à la solution (à 4bancs) qui ignore l'influence du RTOS sur la consommation globale. Là encore, c'est l'énergie de réveil du banc relatif au RTOS (figure 4.25) par les tâches LUD et LMD qui induit cette différence.

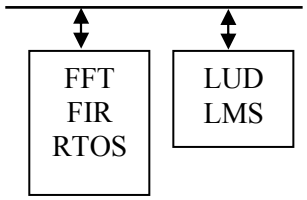


Figure 4.25. Allocation optimale du RTOS dans la mémoire multi-bancs.

Les deux exemples montrent que la structure mémoire optimale peut varier de manière significative en fonction de l'activité du RTOS induite par les tâches de l'application. Ce point illustre ainsi plus globalement d'une part, que la structure mémoire peut avoir un impact important sur la consommation globale et d'autre part, que les résultats obtenus par les techniques d'optimisation sont sensibles à la qualité des modèles considérés.

4. Validation de l'approche heuristique

Afin d'évaluer l'efficacité de l'approche heuristique du chapitre 3, les solutions obtenues par cet algorithme sont comparées en consommation et en temps de résolution aux solutions optimales obtenues avec l'exploration exhaustive, sous les mêmes conditions (caractéristiques de l'application et de l'architecture).

Cette comparaison est effectuée sur toutes les applications considérées dans ce chapitre. Les résultats sont résumés dans le tableau 4.12.

Tableau 4.12. Comparaison des solutions obtenues par l'algorithme exhaustif et l'heuristique

Application	**Allocation optimale**			**Solution rendue par l'heuristique**		**Différence %**
	Configuration mémoire	énergie (10^7nJ)	T (sec)	Configuration mémoire	énergie (10^7nJ)	
7 tâches Tableau 4.3	{(CRC,FIR,Matmul);LMS ; FFT; LUD; ADPCM}	2,525	1	{(CRC,FIR,Matmul); LMS ; FFT; LUD; ADPCM}	2,525	0
	{(CRC,FIR,Matmul,RTOS); LMS ; FFT; LUD; ADPCM}	3,606	11	{(CRC,FIR,Matmul,RTOS); LMS ;FFT;LUD; ADPCM}	3,606	0
6 tâches Tableau 4.4	{(Fir, Fibcall, Qsort} ; IDCT ; FFT ; ADPCM)}	0, 639	0	{(Fir, Fibcall, Qsort} ; IDCT ; FFT ; ADPCM)}	0,6 39	0
	{(IDCT,FIR,Fibcall,Qsort,ADPCM,RTOS) ; FFT}	0,733	1	{(IDCT,FIR,Fibcall,Qsort, ADPCM,RTOS) ; FFT}	0,733	0
4 tâches Tableau 4.5	{(Matmul,FIR); ADPCM ;FFT}	0,2809	0	{(Matmul,FIR); ADPCM ;FFT}	0,2809	0
	{(Matmul,FIR,FFT,RTOS) ; ADPCM}	0,321	0	{(Matmul,Fir,FFT,RTOS) ; ADPCM}	0,321	0
4 tâches Tableau 4.11 article	{FFT ; LUD; LMS ; FIR}	1,40	0	{FFT;FIR;LUD; LMS}	1,40	0
	{(FFT,FIR,RTOS); (LUD,LMS)}	1,66	0	{(FFT,FIR,RTOS) ;LUD ; LMS}	1,71	3,22
Multimédia Tableaux 4.8 et 4.9	{(BB,SC,SD,CC,CD);(MC,VLD,IQ,IDCT); EQ}	0,9686	1549	{MC,VLD,IQ,IDCT,);SC; SD; CD; BB; CC;EQ }	0,9688	0,02

Le tableau 4.12 montre que la consommation des solutions obtenues avec l'heuristique approche celle des solutions optimales dans la plupart des cas.

Nous notons que notre approche s'éloigne de la solution optimale quand il s'agit de déplacer simultanément un groupe de tâches dans un banc alors que l'approche développée dans l'heuristique teste itérativement une seule tâche à la fois. Ce cas se produit dans l'application à 4 tâches avec le RTOS : l'heuristique a isolé la tâche LUD, puis la tâche LMS alors que regrouper les deux tâches dans un même banc permet une meilleure réduction de la consommation. Comme l'heuristique permet de trouver un minimum local qui approche la consommation optimale dans tous les exemples traités, on a décidé de ne pas considérer le déplacement d'un groupe de tâches qui aurait pour conséquence d'entraîner une augmentation de la complexité de l'algorithme.

Au niveau des temps de résolution, pour des ensembles de tâches constitués d'au plus 7 tâches, l'exploration exhaustive reste rapide (0-1sec). A partir de 8 tâches, 11 secondes sont nécessaires pour explorer toutes les configurations mémoires, 106 secondes pour un ensemble de 9 tâches et environ 25 mn pour un ensemble de 10 tâches. Ces temps sont obtenus sur un PC Windows 2,6 GHz avec 1 Giga octets de mémoire. En ce qui

concerne l'heuristique, elle reste avantageuse en temps de résolution dans tous les cas : la solution est fournie en un temps inférieur à la seconde.

5. Validité de la solution mémoire avec le changement de l'ordonnancement

L'ordonnancement des tâches d'une application sur une hyperpériode est une entrée essentielle sur laquelle se base notre approche statique d'allocations et de configuration mémoire. Cependant, en cours d'exécution, des changements dans l'ordonnancement, conçu pendant la phase de conception, sont très probables. Les changements dans l'ordonnancement ont plusieurs origines comme la variation des temps d'exécutions avec pour conséquence des nombres différents de suspensions de tâche pendant l'hyperpériode. Ces changements entraînent une modification de certains paramètres de l'ordonnancement qui sont les successivités entre les tâches et les préemptions et par la suite une consommation différente pour une même allocation mémoire.

Dans ce paragraphe, des expérimentations sont réalisées afin de tester si une solution mémoire statiquement obtenue est efficace en énergie si des changements dans l'ordonnancement se manifestent. Plus généralement, il s'agit de tester jusqu'à quel point notre approche est liée à la séquence d'exécution des tâches.

Afin d'obtenir des changements dans l'ordonnancement RM initial, des modifications sur les périodes de quelques tâches sont effectuées ce qui permet d'obtenir une séquence différente d'exécution des tâches. Par ailleurs, une deuxième technique d'ordonnancement à priorités dynamiques EDF est employée.

L'application considérée est précédemment utilisée dans le paragraphe 4.1 mais avec des périodes de tâches modifiées afin d'obtenir un ordonnancement différent. Un changement de périodes des tâches entraîne une variation dans le nombre d'accès à la mémoire principale. Les caractéristiques modifiées de cette application sont décrites dans le tableau 4.13.

Tableau 4.13. Caractéristiques de l'application à 6 tâches

Tâche	P_i (cycles)	c_i (cycles)	S_{Ti} (kB)	M_i
Fibcall	625000	9536	27	114
FFT	800000	515771	97	4404
Qsort	2500000	13309	31	97
IDCT	5000000	16131	33	193
FIR	10000000	33983	152	133
ADPCM	40000000	2486633	133	4053

Les tâches de cette application sont dans un premier temps ordonnancées avec la technique RM. La matrice de successivités $S_{6,1}$ et le vecteur de préemptions $Pr_{6,1}$ entre les tâches, construits sur l'hyperpériode, sont donnés dans la figure 4.26.

$$S_{6,1} = \begin{bmatrix} 0 & 82 & 4 & 0 & 0 & 8 \\ 0 & 0 & 12 & 0 & 0 & 15 \\ 0 & 0 & 0 & 8 & 0 & 2 \\ 0 & 0 & 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} ; \quad Pr_{6,1} = \begin{bmatrix} 0 \\ 40 \\ 0 \\ 0 \\ 0 \\ 13 \end{bmatrix}$$

Figure 4.26. Paramètres de l'ordonnancement RM des tâches du tableau 4.13

Deux ordonnancements différents de la même application sont obtenus, en effectuant les modifications ci-dessous.

Modification 1

Les périodes des tâches Fibcall et FFT sont permutées ce qui induit un ordonnancement différent avec la technique RM.
Les nouvelles valeurs de successivités et de préemptions entres les tâches sont données dans la figure 4.27.

$$S_{6,2} = \begin{bmatrix} 0 & 42 & 4 & 0 & 0 & 33 \\ 0 & 0 & 12 & 0 & 0 & 21 \\ 0 & 0 & 0 & 8 & 0 & 3 \\ 0 & 0 & 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} ; \quad Pr_{6,2} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix}$$

Figure 4.27. Paramètres de l'ordonnancement issus de la modification 1

Modification 2

Les périodes des tâches IDCT et FIR sont permutées dans le tableau 4.13 ce qui conduit aux nouvelles valeurs de successivités et de préemptions entres les tâches présentées sur la figure 4.28.

$$S_{6,3} = \begin{bmatrix} 0 & 82 & 4 & 0 & 26 & 0 \\ 0 & 0 & 12 & 0 & 64 & 0 \\ 0 & 0 & 0 & 8 & 6 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} ; \quad Pr_{6,3} = \begin{bmatrix} 0 \\ 40 \\ 0 \\ 0 \\ 50 \\ 0 \end{bmatrix}$$

Figure 4.28. Paramètres de l'ordonnancement issus de la modification 2

Enfin, nous considérons un ordonnancement suivant une politique EDF des tâches du tableau 4.13. Les priorités des tâches sont attribuées dynamiquement dans l'ordre inverse des échéances des tâches. La tâche la plus prioritaire est celle qui a l'échéance la plus proche.
La matrice de successivités et le vecteur de préemptions pour cet ordonnancement sont donnés dans la figure 4.29.

$$S_{6,4} = \begin{bmatrix} 0 & 54 & 12 & 0 & 0 & 12 \\ 0 & 0 & 4 & 0 & 0 & 11 \\ 0 & 0 & 0 & 8 & 0 & 2 \\ 0 & 0 & 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} ; Pr_{6,4} = \begin{bmatrix} 0 \\ 12 \\ 0 \\ 0 \\ 0 \\ 13 \end{bmatrix}$$

Figure 4.29. Paramètres de l'ordonnancement relatifs à la technique EDF

Un nombre a priori plus faible de préemptions entre tâches est obtenu avec l'ordonnancement EDF par rapport à l'ordonnancement RM. En effet, dans la politique RM une tâche prête préempte automatiquement une tâche moins prioritaire en fonction des périodes des tâches, alors que dans la politique EDF une tâche ne préempte une autre tâche que si elle devient prête et que son échéance est plus proche.

Les configurations mémoire optimales obtenues pour ces différents ordonnancements sont présentées dans le tableau 4.14. La colonne de droite du tableau correspond à la différence de consommation entre la solution trouvée pour chaque ordonnancement et la solution optimale correspondante à l'ordonnancement initial.

Par exemple, la valeur 1,08% correspond à la différence entre la consommation de la configuration mémoire optimisée pour l'ordonnancement «Modification 2» et la consommation de la configuration mémoire optimale de l'ordonnancement RM initial mais associée cette fois à l'ordonnancement «Modification 2».

Tableau 4.14. Configurations mémoire optimales obtenues pour chaque ordonnancement

Ordonnancement	**Configuration mémoire optimale**	**Différence %**
Ordonnancement RM	{(Qsort, Fibcall, IDCT, FIR, ADPCM); FFT}	-
Modification 1	{(Qsort, Fibcall, IDCT, FIR,ADPCM); FFT}	-
Modification 2	{(Qsort, IDCT, FIR, Fibcall); FFT;ADPCM}	1,08
Ordonnancement EDF	{(Qsort, IDCT, FIR, ADPCM, Fibcall); FFT}	-

L'allocation et la configuration mémoire initialement obtenues avec la technique RM restent inchangées dans 2 cas : modification 1 et ordonnancement EDF. Une configuration mémoire à 3 bancs est optimale pour l'ordonnancement obtenu par la modification 2. Cependant, dans ce cas, la configuration mémoire construite à partir de l'ordonnancement RM initial reste valable puisqu'une faible différence (de l'ordre de 1%) entre les consommations des deux solutions est constatée.

Puisque les variations des paramètres issus des ordonnancements considérés affectent principalement la consommation due aux réveils des bancs et aux préemptions et que la contribution de l'énergie de réveil dans la consommation totale est très faible (ne dépassant pas 1%), ces expérimentations sont reproduites avec des valeurs de $E_{0resynchronisation}$ et $E_{préemption}$ plus importantes (×10). La contribution de $E_{resynchronisation}$ est augmenté à 9 %. Les résultats obtenus sont résumés dans le tableau 4.15.

Tableau 4.15. Configurations mémoire optimales obtenues pour chaque ordonnancement avec des paramètres systèmes différents

Ordonnancement	**Configuration mémoire optimale**	**Différence %**
Ordonnancement RM	{(Qsort, IDCT, FIR, ADPCM); (FFT, Fibcall)}	-
Modification 1	{(Qsort, Fibcall, IDCT, FIR, ADPCM); FFT}	3,93
Modification 2	{(Qsort, IDCT, FIR); (Fibcall, FFT);ADPCM}	0,36
Ordonnancement EDF	{(Qsort, IDCT, FIR, ADPCM, Fibcall); FFT}	0,226

La consommation des configurations mémoire optimales obtenue pour chaque ordonnancement est légèrement différente de la consommation de la configuration obtenue par l'ordonnancement RM initial. Il apparait donc sur cet exemple que la configuration mémoire optimale initiale reste robuste vis-à-vis de variations d'ordonnancement, à nombre et type de tâches constants.

Une autre application test a également été considérée (tableau 4.16) qui confirme cette qualité de robustesse par rapport aux variations d'ordonnancement des tâches (tableau 4.17).

Tableau 4.16. Caractéristiques de l'application à 7 tâches

Tâche	P_i (cycles)	c_i (cycles)	S_{Ti} (kB)	M_i
CRC	750000	42907	31	99
FIR	1000000	33983	152	47
FFT	1000000	515771	97	493
LMS	1500000	365893	32	123
LUD	5000000	255998	38	102
MATMUL	15000000	13985	29	26
ADPCM	30000000	2486633	139	3387

Tableau 4.17. Configurations mémoire optimales obtenues pour chaque ordonnancement

Ordonnancement	**Configuration mémoire optimale**	**Différence %**
$E_{0resynchronisation}$; contribution de $E_{resynchronisation}$ = 2%		
Ordonnancement RM	{(FIR, LUD, Matmul, ADPCM); (CRC, LMS); FFT}	-
Modification 1	{(FIR, LUD, Matmul, ADPCM, CRC); LMS; FFT}	0,845
Ordonnancement EDF	{(FIR, LUD, Matmul, ADPCM); (CRC, LMS); FFT}	-
$E_{0resynchronisation}$ × 10 ; contribution de $E_{resynchronisation}$ = 20%		
Ordonnancement RM	{(FIR, LUD, Matmul, ADPCM); (CRC, FFT); LMS}	-
Modification 1	{(FIR, LUD, Matmul, ADPCM); (CRC, LMS); FFT}	3%
Ordonnancement EDF	{(FIR, LUD, Matmul, ADPCM, CRC) ; LMS; FFT}	9%

La modification 1 du tableau 4.17 correspond à la permutation des périodes des tâches CRC et FIR et celles des tâches FFT et LMS.

Conclusion

Dans ce chapitre, nous avons présenté les résultats obtenus par notre approche pour déterminer une configuration mémoire sur plusieurs applications test. Les principaux résultats montrent que :

1. la technique de DVS employée seule minimise effectivement la consommation processeur mais en contre partie elle est responsable de l'augmentation de la consommation de la mémoire principale. Les sources de cette augmentation sont identifiées : elles concernent l'augmentation du nombre de préemptions et la co-activation plus longue de la mémoire du fait de l'allongement des temps d'exécution des tâches.

2. la solution proposée dans le cadre de nos travaux consiste à partitionner l'espace d'adressage en plusieurs bancs afin de réduire à un seul banc la quantité de surface co-active avec l'exécution des tâches par le processeur. Cette technique permet d'obtenir des gains sur l'énergie totale significatifs (processeur et mémoire).

3. l'heuristique développée approche dans la plupart des cas la configuration mémoire et l'allocation des tâches aux bancs optimales.

4. la prise en compte de l'allocation des instructions et des données propres au RTOS dans les bancs mémoire entraîne une configuration mémoire optimale en énergie avec moins de bancs suite à l'augmentation de l'énergie de réveil du banc où réside le RTOS.

5. l'architecture mémoire obtenue pendant la phase de conception et basée sur une analyse statique de l'ordonnancement, continue à être efficace en énergie même si des changements dans l'ordonnancement surviennent à l'exécution, à condition que l'énergie de réveil ne soit pas prépondérante vis-à-vis de l'énergie totale.

Conclusion

La consommation d'énergie est devenue un des critères majeurs dans la conception de systèmes sur puce. Nous nous sommes intéressés à la consommation d'un processeur muni d'une possibilité d'ajustement en fréquence et en tension couplé à une hiérarchie mémoire. La technologie des SoC conduit à placer la mémoire principale à l'extérieur du composant.

A partir de l'étude de la variation de la consommation mémoire principale lors de l'ajustement conjoint de la tension et de la fréquence du processeur, notre proposition est d'adopter une architecture mémoire multi-bancs. En effet, cette étude a révélé que la consommation d'une mémoire monolithique augmente alors que la consommation du processeur diminue en utilisant la technique de DVS. Cette augmentation de la consommation mémoire est due essentiellement à la dissipation d'énergie produite en maintenant la mémoire plus longtemps co-active avec le processeur, suite à l'allongement des temps d'exécutions des tâches. De plus, ces temps d'exécution des tâches augmentés sont aussi responsables de préemptions plus fréquentes. Les tâches les moins prioritaires sont plus souvent préemptées par des tâches plus prioritaires. Ces préemptions causent des accès supplémentaires à la mémoire principale suite à des défauts de cache, à la sauvegarde et à la récupération des contextes des tâches préemptées entrainant une consommation mémoire accrue. Dans nos travaux, on s'est particulièrement intéressé à contenir l'augmentation de la consommation mémoire induite par la technique de DVS : une architecture mémoire multi-bancs exploitant des modes faible consommation est alors proposée. En effet, une architecture multi-bancs permet de réduire la surface mémoire active à un unique banc, auquel le processeur accède pour lire ou écrire les instructions et/ou les données de la tâche en cours d'exécution. Les autres bancs de la

mémoire qui ne sont pas concernés par l'exécution de la tâche en cours, sont mis dans un mode faible consommation pour réduire l'énergie dissipée.

Pour obtenir une solution globalement efficace, il s'agit alors de trouver la configuration des bancs en nombre et en taille ainsi que l'allocation des tâches à ces bancs pour minimiser l'énergie totale dissipée. Sur quels critères doit-on se baser pour déterminer cette configuration mémoire ? Pour apporter une réponse à ces questions, une identification des paramètres influant sur la consommation mémoire est effectuée et des modèles permettant d'évaluer la consommation mémoire pour des systèmes multi-tâches ont été développés.

Une fois ces modèles de consommation mémoire déterminés, une méthode permettant de trouver la configuration mémoire multi-bancs et l'allocation de tâches correspondante minimisant la consommation est alors possible. Deux approches ont été proposées. La première est basée sur une exploration exhaustive de l'ensemble des solutions, permettant ainsi de trouver la configuration mémoire optimale. La complexité exponentielle du problème (l'espace d'exploration croit exponentiellement avec le nombre de tâches de l'application) limite très vite l'intérêt de cette approche. Pour des applications comportant plus de 10 tâches, une heuristique capable d'explorer un sous-espace potentiellement intéressant et de résoudre le problème en un temps polynomial est développée dans un second temps. La complexité réduite de cette heuristique permet d'envisager de l'employer en ligne sur des applications avec création dynamique de tâches.

Des expérimentations sur des applications de traitement de signal et une application de type multimédia (GSM, décodeur MPEG-2) ont montré des gains notables aussi bien sur la consommation mémoire que sur la consommation du couple (processeur, mémoire). Ces expérimentations ont permis aussi de valider l'approche heuristique proposée et de tester l'efficacité de la configuration mémoire trouvée hors ligne, pendant la phase de conception, quand des changements en ligne dans l'ordonnancement surviennent.

Perspectives

Certaines améliorations peuvent être apportées à ce travail. La première serait de considérer un modèle plus détaillé du nombre d'accès à la mémoire principale, actuellement défini comme une constante dans notre étude. Il pourrait aussi tenir compte de la gestion du cache (degré d'associativité, partitionnement en bancs, politique d'écriture en cas d'échec) et de ses paramètres architecturaux (taille, taille de ligne, nombre de ports,…). La consommation du couple (cache, mémoire principale) permettrait d'adapter également l'architecture du cache en fonction de la consommation de la mémoire principale multi-bancs. Une deuxième amélioration importante serait de considérer la consommation d'autres composants présents dans l'architecture système qui peuvent être concernés par l'utilisation de la technique de DVS appliquée au processeur, on cite comme exemples les réseaux sur puce (*NOC*) et les périphériques.

Une extension qui serait intéressante à considérer dans notre approche serait d'étudier la configuration mémoire multi-bancs pour des systèmes nécessitant une adaptation par rapport à l'environnement où le nombre de tâches est variable en cours d'exécution. Il s'agit par exemple de téléchargements de jeux ou de services en ligne. Ces tâches créées en cours d'exécution nécessitent une allocation dans les bancs mémoire. L'adaptation de l'architecture et de la configuration mémoire multi-bancs initialement trouvée est alors nécessaire pour maintenir les meilleurs gains d'énergie. Plusieurs cas sont possibles, le premier est que les nouvelles tâches sont allouées à un banc déjà existant de la configuration initiale. Une nouvelle configuration avec le même nombre de bancs est employée. Le deuxième cas se produit si un nombre de bancs plus faible suffit pour obtenir de meilleurs gains par cette nouvelle configuration. Dans ce cas, la mise hors tension des bancs non nécessaires est envisageable. Inversement, si un nombre de bancs plus important est nécessaire, des bancs qui sont initialement éteints sont remis en fonctionnement. Dans ces trois cas, si une permutation différente des tâches est nécessaire, une migration utile de tâches d'un banc vers un autre peut être envisagée. Le redimensionnent virtuel des tailles des bancs est nécessaire dans tous les cas pour éviter que la partie du banc non utilisée consomme de l'énergie. Ce point mérite une étude approfondie.

Bien que notre approche de configuration mémoire multi-bancs soit testée sur une architecture biprocesseur de type OMAP, des hypothèses restrictives ont été considérées (fonctionnement de type maître/esclave). Etendre l'étude de la consommation d'une mémoire multi-bancs à une architecture multiprocesseur nous semble prometteuse. Une configuration mémoire avec des bancs réservés à chaque processeur et des bancs partagés accessibles par tous les processeurs peut être imaginée. Le nombre de bancs réservés à chaque processeur dépendrait sûrement du partitionnent des tâches sur chaque processeur. Dans ce cas, localement à chaque processeur notre configuration reste applicable, globalement pour l'ensemble des processeurs notre approche nécessiterait sans doute d'être étendue.

Une autre perspective intéressante à explorer consiste à considérer une architecture mémoire principale hétérogène. Par exemple, constituée de mémoire DRAM multi-bancs, de mémoires *NAND Flash*, *NOR Flash*, mémoire magnétique (MRAM) afin de bénéficier de l'avantage de chaque technologie de mémoire en fonction de la nature et des contraintes des tâches.

Les mémoires *NAND Flash* et *NOR Flash*, largement employée dans les systèmes embarqués comme les appareils photos numériques, les téléphones cellulaires, les assistants personnels (PDA), ou les dispositifs de lecture ou d'enregistrement sonore tels que les baladeurs MP3, possèdent un espace d'adressage monolithique. Le partitionnement de ces mémoires sur plusieurs bancs n'a pas été abordé dans la littérature. Pourtant, ce type de mémoire possède un coût élevé en temps d'accès et en énergie surtout en mode écriture.

Bibliographie

Abella J., Gonzalez A., "Power efficient data cache design" *International Conference on Computer Design ICCD 2003,* 13-15 October 2003, San Jose, CA, p. 8-13.

Albonesi D. H., "Selective Cache Ways: On-Demand Cache Resource Allocation", *International Symposium on Microarchitecture MICRO*, 1999.

ARM, ARM926EJ-S, *Technical Reference Manual,* 2001-2003. http://www.arm.com/pdfs/DDI0198D_926_TRM.pdf

Auslander E., Couvrat M., "Take the Lead in GSM", *Applications of Digital Signal Processing, Proc. of DSP'94*.

Aydin H., Melhem R., Mossé D., Alvarez P. M., "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems", *RTSS'01 (Real-Time Systems Symposium)*, London, England, December 2001.

Bajwa R. S., Hiraki M., Kojima H., Gorney D., Nitta K., Shridhar A., Seki K., and Sasaki K., "Instruction Buffering to Reduce Power in Processors for Signal Processing" *IEEE Transactions on VLSI Systems*, p. 417-424, December 1997.

Bell E.T., "Exponential numbers". *The American Mathematical Monthly*, 41 (7): p. 411-419, August/September 1934.

Bellas N., Hajj I., and Polychronopoulos C., "Using dynamic cache management techniques to reduce energy in a high-performance processor", *in ISLPED*, p. 64-69, August 1999.

Benini L., De Micheli G., "Dynamic Power Management: Design Techniques and CAD Tools", *Kluwer Academic Publishers*, 1998.

Benini L., Macci A., Poncino M. "A recursive algorithm for low-power memory partitioning" *ISLPED*, Rapallo, Italy, 2000.

Burger D. and Austin T.M., "The SimpleScalar Tool Set, Version 2.0," *Univ. of Wisconsin-Madison Computer Sciences Dept. Technical Report #1342*, June 1997.

Calder B., Grunwald D., Emer J., "Predictive sequential associative cache" *In International Symposium on High Performance Computer Architecture*. p. 244–253, 1996

Chandrakasan A., Bowhill W. J., and Fox F, "Design of HighPerformance Microprocessors Circuit". *IEEE Press*, Piscataway, NJ, 2001.

Cho J., Paek Y., Whalley D., "Fast Memory bank Assignment for fixed point digital signal processors", *ACM Transactions on Design Automation of Electronic Systems (TODAES)* Volume 9, Issue 1 (January 2004), p. 52-74.

Choi K., Soma R., Pedram M., "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times " *In Proceedings DATE*, Paris, France, February 2004.

Choi K., Lee W., Soma R., Pedram M., "Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation". *ICCAD 2004*, November 7-11, 2004, San Jose, CA, p. 29-34.

Davis R., Wellings A., "Dual priority scheduling" *In Proceedings of 16th IEEE Real-Time Systems Symposium (RTSS '95)*, p. 100, 1995.

Delaluz V., Kandemir M., Kolcu I., "Automatic Data Migration for Reducing Energy Consumption in Multi-Bank Memory Systems" *DAC,* 2002.

Delaluz V., Sivasubramaniam A., Kandemir M., Vijaykrishnan N. and Irwin M.J., "Scheduler-based DRAM Energy Management", *39th Design Automation Conference DAC,* June, 2002.

Delaluz V., Kandemir M., Vijaykrishnan N., Sivasubramaniam A., M.J Irwin "DRAM Energy Management Using Software and Hardware Directed Power Mode Control". *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01)*, 20-24 January 2001, Mexico, p.159-170.

Delaluz V., Kandemir M., Vijaykrishnan N., Irwin M. J. "Energy-Oriented Compiler Optimizations for Partitioned Memory Architectures" *In Proceedings of CASES* (2000).

Delaluz V., Kandemir M., Vijaykrishnan N., A. Sivasubramaniam, Irwin M. J. "Hardware and software techniques for controlling DRAM Power Modes" *IEEE transaction on computers*, Vol 50, N° 11, November 2001.

Ernst R. and Ye W., "Embedded Program Timing Analysis based on Path Clustering and Architecture Classification", *Computer-Aided Design (ICCAD'97)*, p. 598-604, 1997

Fan X., Ellis C.S., Lebeck A.R., "Memory Controller Policies for DRAM Power Management", *In Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)* August, 2001.

Fan X., Ellis C. S., Lebeck A. R. "Modeling of DRAM Power Control Policies Using Deterministic and Stochastic Petri Nets" (2002)

Fan X., Ellis C. S., Lebeck A. R. "The Synergy Between Power-Aware Memory Systems and Processor Voltage Scaling". *PACS ,* 2003.

Farrahi A. H., Tellez G. E., Sarrafzadeh M., "Exploiting Sleep Mode for Memory Partitioning and Other Applications", *VLSI Design Journal*, Vol 7, No 3, p. 271-287, 1998.

Flautner K., Kim N., Martin S., Blaauw D., Mudge T., "Drowsy Caches: Simple Techniques for Reducing Leakage Power". *International Symposium on Computer Architecture*, 2002, June 2002.

Fromm R., Perissakis S., Carwell N., Kozyrakis C., McGaughy B., Patterson D., Anderson T., Yelick K., "The Energy Efficiency of IRAM Architectures," *ISCA '97: The 24th Annual International Symposium on Computer Architecture ,* Denver, CO, 2-4 June 1997.

Ghose K.and Kamble M. B.,"Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line buffers and Bit-Line Segmentation," *Proc. of the 1999 International Symposium on Low Power Electronics and Design*, p. 70-75, Aug. 1999.

Gomez J.I., Marchal P., Bruni D., Benini L., Prieto M., Catthoor F., Corporaal H., "Scenario-based SDRAM-Energy-Aware Scheduling for Dynamic Multi-media Applications on Multi-Processor Platforms" *Workshop on Application Specific Processors WASP* 2002.

Gruian F., "Hard real-time scheduling using stochastic data and DVFS Processors," *ISLPED*, August 2001.

Hasegawa A., Kawasaki I., Yamada K., Kawasaki S. S., and Biswas P., "SH3: High Code Density, Power," *IEEE Micro*, p. 11–19, December 1995.

Huang M., Renau J., Yoo S.M., Torrellas J., "A framework for dynamic energy efficiency and temperature management". *In International Symposium on Microarchitecture,* 2000.

Huang H., Shin K. G., Lefurgy C., Keller T. "Improving energy efficiency by making DRAM less randomly accessed" *ISLPED* , 2005, p. 393-398.

Huang H., Shin K.G., Lefurgy C., Rajamani K., Keller T., Hensbergen E.V, Rawson F., "Cooperative Software.Hardware Power Management for Main Memory", *Workshop on Power-Aware Computer Systems*, 2004.

Infineon Inc. Mobile RAM *Data Sheet*, 2004.

Inoue K., Ishihara T., Murakami K., "Way-predicting set-associative cache for high performance and low energy consumption". In *International Symposium on Low Power Electronics and Design*. 1999

Intel Inc. "PXA255 and PXA26x Applications Processors Power Consumption During Power-up,Sleep, and Idle". *Data Sheet*, 2003

ftp://download.intel.com/design/pca/applicationsprocessors/applnots/27878601.pdf

Intel Inc. "Intel PXA270 Processor Electrical, Mechanical and Thermal Specification" *Data Sheet*, 2005.

Itoh K., Sasaki K., Nakagome Y., "Trends in Low-Power RAM Circuit Technologies" *Proc. IEEE*, vol. 83, no. 4 (April 1995), p. 524-543.

ITRS, "System Drivers", 2005. http://www.itrs.net/Links/2005ITRS/SysDrivers2005.pdf

Jejurikar R. and Gupta R., "Integrating Preemption Threshold Scheduling and Dynamic Voltage Scaling for Energy Efficient Real-Time Systems", *In Proc. of International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA '04),* August 2004.

Jejurikar R., Pereira C., Gupta R. "Leakage Aware Dynamic Voltage Scheduling in Real-Time Embedded Systems" *In Proc. of DAC* June 2004, CA, USA, p. 275-280.

Johguchi K., Zhu Z., Mattausch H. J., Koide T., Hironaka T. "Unified Data/Instruction Cache with Bank-based Multi-Port architecture" *In Proc. of COE WS* 2005.

Kandemir M., Kolcu I., Kadayif I. "Influence of loop optimizations on energy consumption of multi-bank memory systems" *In Proc. Compiler Construction*, April 2002.

Kaxiras S., Hu Z., Martonosi M. "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power" *Proceedings of the 28th annual international symposium on Computer architecture*, Göteborg, Sweden, 2001 pp: 240 - 251

Kim W., Kim J., Min S., "Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis", *Proceedings of the 2003 international symposium on Low power electronics and design*, Seoul, Korea, August 25-27, 2003

Kim W., Kim J., and Min S. L., "Quantitative Analysis of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems". *In Proceedings of the SoC Design Conference*, Novermber 2003.

W. Kim, J. Kim, S. L. Min, "Preemption-aware dynamic voltage scaling in hard real-time systems" *In Proceedings of the 2004 international symposium on Low power electronics and design*, California, USA 2004, pp: 393 -398.

Kim S., Vijaykrishnan N., Kandemir M., Sivasubramaniam A., Irwin, M. J., "Partitioned Instruction Cache Architecture for energy efficiency" *ACM Transaction on Embedded computing systems*, Vol. 2, No. 2, May 2003, Pages 163-185.

Kim S., Vijaykrishnan N., Kandemir M., Sivasubramaniam A., Irwin, M. J., Geethanjali E. b, "Power-aware partitioned cache architectures", *In International Symposium on Low Power Electronics and Design*. 2001.

Kin J., Gupta M., Mangione-Smith W.H., " The filter cache: an energy efficient memory structure" *30th Annual International Symposium on Microarchitecture (Micro '97)* Los Angeles, CA, USA. 1997, p. 184 – 193.

Ko U., Balsara P., "Energy Optimization of Multilevel Cache Architectures for RISC and CISC Processors" *IEEE Transactions on VLSI Systems*,Vol. 6, No. 2, p. 299-308, June 1998.

Lebeck A. R., F. Xiaobo, Heng Z., Ellis C. S., "Power Aware Page Allocation" *In Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, November 2000.

Lee H. S. and G. S. Tyson, "Region-Based Caching: An Energy-Delay Efficient Memory Architecture for Embedded Processors," *Proc. of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, November 2000, p.120–127.

Lee C., Lee K., Hahn J., Seo Y., Min S. L., Ha R., Hong S., Park C. Y., Lee M., Kim C.S. "Bounding Cache-Related Preemption Delay for Real-Time Systems". *IEEE Transactions on Software Engineering*, V.27 n.9, September 2001, p. 805-826.

Lee Y., Reddy K., Krishna C. "Scheduling techniques for reducing leakage power in hard real-time systems" *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, Porto, Portugal, July 2003.

Lee H. G., N. Chang "Energy-aware Memory Allocation in Heterogeneous non-volatile Memory Systems", *Proc. of ISLPED'03*, August 2003 Seoul, Korea.

Leupers R. and Kotte D. "Variable Partitioning For Dual Memory Bank Dsps" *In IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, p. 1121-1124, 2001.

Li L., Kadayif I., Tsai Y.F., Vijaykrishnan N., Kandemir M. T., Irwin M. J., Sivasubramaniam A., "Leakage Energy Management in Cache Hierarchies". *IEEE PACT* 2002 p. 131-140.

Liu C., Layland J., "Scheduling algorithms for multiprogramming in hard real-time envireonment" *Journal of the ACM 20*, p. 40-61, February 1973,.

Lyuh C., Kim T., "Memory access scheduling and binding considering energy minimization in multi-bank memory systems" *Proceedings of the 41st annual conference on Design automation*, p. 81- 86, CA USA, 2004.

Mace M. E, "Memory storage patterns in parallel processing" *Kluwer Academic Publishers*, Norwell, MA, USA, 1987.

Mamidipaka M., Dutt N. "eCACTI: An Enhanced Power Estimation Model for On-chip Caches" *Center for Embedded Computer Systems (CECS) Technical Report TR-04-28*, Sept. 2004.

Marchal P., Catthoor F., Bruni D., Benini L., Gomez J.I , Pinuel L., "Integrated Task Scheduling and Data Assignment for SDRAMs in Dynamic Applications," *IEEE Design and Test of Computers,* vol. 21, no. 5, p. 378-387, September/October, 2004.

Marchal P., Gomez J. I., Atienza D., Mamagkakis S., Catthoor F., "Power aware data and memory management for dynamic applications" *IEE Proceedings Computers and Digital Techniques*, Mars 2005, Volume: 152, Issue: 2, p. 224- 238.

Marculescu, D. "On the use of microarchitecture-driven dynamic voltage scaling," *Proc. of Workshop on Complexity-Effective Design*, 2000.

Martin S.M., Flautner K., Mudge T., Blaauw D. "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Low Power microprocessors under dynamic workloads" *ICCAD* 2002.

Micron Inc. 512Mb: x4, x8, x16 SDRAM, *Data Sheet*, 2006

http://download.micron.com/pdf/datasheets/dram/sdram/512MbSDRAM.pdf

Niu L., Quan G.,"Reducing Both Dynamic and Leakage Energy Consumption for Hard Real-Time Systems" *Proc. of CASES* September, 2004, DC, USA.

Ozturk O., Kandemir M. "Nonuniform Banking for Reducing memory Energy Consumption" *Proc. of Date 2005*, , p. 814- 819, 7-11 March 2005.

Ozturk O., Kandemir M. "Integer linear programming based energy optimization for banked DRAMs" *GLSVLSI,* 2005, p. 92-95, Chicago, IL, April. 17-19, 2005.

Ozturk O., Kandemir M. "Data replication in banked Drams for reducing energy Consumption" *Proc. of ISQED*, 2006 March 27-29 CA, USA.

Panda, P.R. "Memory bank customization and assignment in behavioral synthesis" *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design,* California, United States p. 477 – 481, 1999

Park J.H., Wu S., B. Izadi, "Coarse- Grained DRAM Power Management", *Workshop of Methodologies in Low Power Design (MLPD'03)*, 2003, pp 248-254.

Pazos N., Maxiaguine A., Ienne P., and Leblebici Y., "Parallel modelling paradigm in multimedia applications: Mapping and scheduling onto a multi-processor system-on-chip platform" *In Proceedings of the International Global Signal Processing Conference*, Santa Clara, Calif., September 2004.

Pering T., Burd T. and Brodersen R. "Voltage Scheduling in the lpARM Microprocessor System" *Proc. of ISLPED'00*, Rappallo, Italy.

Pillai P., Shin K.G. "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems" *In proceedings of 18 th ACM symposium on Operating systems Principles*, p. 89-102, October 2001.

Pouwelse J., Langendoen K., and Sips H. "Dynamic Voltage Scaling on a Low-Power Microprocessor" *7th Int. Conf. on Mobile Computing and Networking (Mobicom)*, Rome, Italy, July 2001.

Powell D.G., Lee E.A., and Newman W.C., "Direct Synthesis of Optimized DSP Assembly Code from Signal Flow Block Diagrams" *In Proceeding of IEEE International conference on Acoustic Speech and Signal Processing (ICASSP)*, vol. 5, San Francisco, CA, , 1992, p. 553-556.

Powell, M., AGARWAL, A., VIJAYKUMAR, T.N., FALSAFI, B., AND ROY,K. Reducing set-associative cache energy via way-prediction and selective direct-mapping. *In International Symposium on Microarchitecture*. 2001.

Procaskey C., "Improving Compiled DSP Code Through Language Extensions" *Proceedings of the International Conference on Signal Processing Applications and Technology*, p. 846-850, DSP Associates, October, 1995.

Rambus Inc. 128/144 MBit Direct RDRAM, *Data Sheet*, 1999.

Seoul National University Real-Time Research groups. SNU real-time benchmarks. http://archi.snu.ac.kr/realtime/benchmark/

Shin Y., Choi, K. "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems" *In proceeding of Design Automation Conference*, p. 134-139 ,1999.

Shin Y., Choi K., Sakurai T., "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors", *In Proceedings of International Conference on Computer-Aided Design (ICCAD'00)*, p. 365-368, November 2000.

Shin D., Kim J.and Lee S., "*Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications*" *IEEE Design & Test of Computers*, vol. 18, no. 2, March-April 2001.

Shin D., Kim W., Jeon J., Kim J., Min S. "SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage Scaling Algorithms" *In*

Proceedings of Workshop on Power-Aware Computer Systems PACS, March 2002, Texas.

Shiue, W. T, Chakrabarti C. "Multi-Module Multi-Port Memory Design for Low Power Embedded Systems" *Design Automation for Embedded Systems*, Volume 9, Number 4, December 2004, p. 235-261(27)

Shivakumar P., Jouppi N., "Cacti 3.0: An Integrted Cache Timing, Power and Area Model", *Technical Repport, Compaq, Western Reaserch Laboratory*, August 2001.

Su C. L. and Despain A. M.,"Cache Design Trade-offs for Power and Performance Optimization:A Case Study," *Proc. of the 1995 International Symposium on Low Power Design*, p. 69–74, Apr. 1995.

Thiebaut D., Stone H.S., "footprints in the caches" *ACM Transactions on Computer systems*, vol. 5, No. 4, November 1987, p. 305- 329.

Villa L.,. Zhang M, and Asanovi'c K., "Dynamic Zero Compression for Cache Energy Reduction" *In Proc of 33rd International Symposium on Microarchitecture*, Monterey, CA, December 2000

Wehmeyer L., Helming U., Marwedel P. "compiler-optimized usage of Partitioned memories", *In Proceedings of the 3rd Workshop on Memory Performance Issues (WMPI2004)*, 2004.

Weng L. C., Wang X., Liu B., "A Survey of Dynamic Power Optimization Techniques," *In Proc of The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC'03)* p. 48, 2003.

Yamauchi T., Hammond L., Olukotun K., "The Hierarchical Multi-Bank DRAM : A High-Performance Architecture for memory integrated with processors" *In the Proceedings of the 17th Conference on Advanced Research in VLSI*, p. 303-319, Ann Arbor, MI, September 1997.

Yan L., Luo J., Jha N.K., "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems". *International Conference on Computer-Aided Design (ICCAD'03)*, November 9-13, 2003, San Jose, CA, USA.

Yang S.H., Powell M.D., Falsafi B., Roy K., and Vijaykumar T. N., "Dynamically resizable instruction cache: An energy-efficient and high-performance deep-submicron instruction cache", *Technical Report ECE-007, School of Electrical and Computer Engineering*, Purdue University, 2000.

Yao F., Demers A., Shenker S., "A scheduling model for reduced cpu energy" *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, p. 374-382, 1995.

Yeager K. C., "The MIPS R10000 superscalar microprocessor" *In Proc. Int. Symp. Microarchitecture*, Apr. 1996, p. 28-40.

Zhai B., Blaauw D., Sylvester D., and Flautner K., "Theoretical and Practical Limits of Dynamic Voltage Scaling" *Proceedings of the 41st Design Automation Conference (DAC).* San Diego, CA, 2004.

Zhang F., Chanson S. T. "Blocking-Aware Processor Voltage Scheduling for Real-Time Tasks" *ACM Transactions on Embedded Computing Systems,* Vol. 3, No. 2, May 2004, p. 307-335.

Publications Personnelles

Revues

Ben Fradj H., El Ouardighi A., Belleudy C., Auguin M., "Energy aware memory architecture configuration" *ACM computer architecture news (CAN)*, Volume 33, Issue 3 (June 2005), pp 3-9.

Ben Fradj H., Icart S., Belleudy C., Auguin M., "Optimisation de la Consommation Mémoire Multi-Bancs pour un Système Multi-Tâches" *Accepté dans le numéro spécial SOC de TSI* qui paraîtra en 2007.

Conférences internationales

Ben Fradj H., Icart S., Belleudy C., Auguin M. "Energy Optimization of a Multi-Bank Main Memory", *SAMOS VI: Embedded Computer Systems: Architectures, MOdeling, and Simulation*, Greece, Samos, July 17-20, 2006, pp 196-205.

Ben Fradj H., Belleudy C., Auguin M. "Multi-Bank Main Memory Architecture with Dynamic Voltage Frequency Scaling for System Energy Optimization", *9th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools,* Croatia 30th Aug 2006 - 1st Sept 2006.

Ben Fradj H., Belleudy C., Auguin M. "System Level Multi-Bank Main Memory Configuration for Energy Reduction" *16th IEEE International Workshop on Power and Timing Modeling Optimization and Simulation (PATMOS)* Montpellier, France, 13-15 September 2006.

Ben Fradj H., Belleudy C., Auguin M. "Main Memory Energy Optimization for Multi-Task Applications" *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SOC)*, Nice, France, October 16-18, 2006.

Ben Fradj H., Belleudy C., Auguin M. "Scheduler-based Multi-Bank Main Memory Configuration for Energy Reduction" *IEEE Symposium on Industrial Embedded Systems (IES)*, Antibe, Nic,e October 18-20, 2006,.

Ben Fradj H., Belleudy C., Auguin M., Pegatoquet A., "Multi-Bank Memory Allocation for Multimedia Application" *In Proc of the 13th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Nice, France, December 10-13, 2006.

Guitton-Ouhamou P., Ben Fradj H., Belleudy C., Auguin M. "Low Power Co-design Tool and Power Optimizations of Schedulings and System Memory", *14th IEEE International Workshop on Power and Timing Modeling Optimization and Simulation (PATMOS)*, Santorini, Greece, September 15-17, 2004, pp. 603 – 612.

Elouardighi A., Ben Fradj H., Belleudy C., Auguin M., "Energy aware memory architecture configuration" *MEDEA international Workshop MEmory performance: DEaling with Applications, systems and architecture*, Antibes, France, September 29-2004, pp 5-11.

Guitton-Ouhamou P., Ben Fradj H., Belleudy C., Auguin M. "Scheduling Refinement and Memory Allocation for Low Power System" *16th IEEE International Conference on Microelectronics ICM ,* Tunis, Tunisia. December 6-8 2004, pp 240-243.

Conférences nationales

Ben Fradj H., Belleudy C., Auguin M. "Energy aware Tasks Allocation to Multi-Bank Memory" *Sophia Antipolis MicroElectronics Forum (SAME)*, October 4-5 2006, Sophia Antipolis, France.