Stratégie de réduction des cycles thermiques pour systèmes temps-réel multiprocesseurs sur puce

Khaled Baati

UNIVERSITE DE NICE-SOPHIA ANTIPOLIS

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

T H E S E

pour l'obtention du grade de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention : Informatique

présentée et soutenue par

Khaled BAATI

Stratégie de réduction des cycles thermiques pour systèmes temps-réel multiprocesseurs sur puce

---

Thèse dirigée par ***Michel AUGUIN***

soutenue le *19 Décembre 2013*

Jury

M.Daniel CHILLET	Maître de conférence, Université de Rennes, France	Rapporteur
M.Jean-Philippe DIGUET	Directeur de recherche, CNRS UMR 6285, France	Rapporteur
M.Yvon TRINQUET	Professeur, Université de Nantes, France	Examinateur
M^{me} Cécile BELLEUDY	Maître de conférence, Université de Nice,France	Examinateur
M.Michel AUGUIN	Directeur de recherche, CNRS UMR 7248, France	Directeur de thèse

A mes parents,

A mes frères,

A ma sœur,

A ma femme

« Il est bon d'avoir à soi quelque chose pour le donner. »
Paul Claudel

Résumé

L'augmentation de la densité des transistors dans les circuits électroniques conduit à une augmentation de la consommation d'énergie qui se traduit par des phénomènes thermiques plus complexes à maitriser. Dans le cas de systèmes embarqués en environnement où la température ambiante peut varier dans des proportions importantes (automobile par exemple), ces phénomènes thermiques peuvent conduire à des problèmes de fiabilité. Parmi les mécanismes de défaillance observés, on peut citer les cycles thermiques (CT) qui induisent des déformations dans les matériaux et jouent un rôle majeur dans la fissuration des couches métalliques dans la puce.

L'objectif de la thèse est de proposer pour des architectures de type multiprocesseur sur puce une technique de réduction des CT subis par les cœurs du processeur, tout en respectant les contraintes temps réel des applications. L'exemple du circuit MPC5517 de Freescale a été considéré. Dans un premier temps un modèle thermique de ce circuit a été élaboré à partir de mesures réalisées par une caméra thermique sur ce circuit décapsulé. Un environnement de simulation a été mis en œuvre pour permettre d'effectuer simultanément des analyses thermiques et d'ordonnancement de tâches et mettre en évidence l'influence de la température sur la puissance dissipée.

Une heuristique globale pour réduire à la fois les CT et la température maximale des processeurs a été étudiée. Elle tient compte des variations de la température ambiante et se base sur les techniques DVFS et DPM. Différents résultats de simulation avec les algorithmes d'ordonnancement globaux RM, EDF et EDZL et avec différentes charges processeur (sur un circuit type MPC5517 et un UltraSparc T1 de Sun Microsystems) illustrent l'efficacité de la technique proposée.

Mots clés : Modélisation thermique pour la conception système, système sur puce, ordonnancement temps réel, basse consommation, fiabilité, cycles thermiques, co-simulation.

Abstract

Increasing the density of transistors in electronic circuits leads to an increase in energy consumption resulting in more complex thermal phenomena to master. For systems embedded in environments where the ambient temperature can vary in large range (e.g. automotive), these thermal effects can induce reliability problems. Among classical failure mechanisms thermal cycles (CTs) produce deformations in materials and play a major role in the cracking of the metal layers in the chip.

The aim of the thesis is to propose a reduction technique of CTs suffered by the processor cores in a multiprocessor on chip architecture such that real-time application constraints are met. The example of the Freescale MPC5517 circuit has been considered. In a first step a thermal model of this circuit was developed. This was achieved from measurements taken by a thermal camera on a decapsulated circuit. Next, a simulation environment has been implemented allowing both the analysis of thermal behavior and the scheduling of tasks so as to highlight the influence of temperature on the dissipated power.

A global heuristic to reduce both the CTs and the maximum temperature of processors has been studied. It takes into account variations in the ambient temperature and is based on DVFS and DPM techniques. Simulation results with global scheduling algorithms RM, EDF and EDZL and different processor loads (for a MPC5517 type circuit and a T1 UltraSparc from Sun Microsystems) illustrate the effectiveness of the proposed technique.

Keywords: Temperature aware system-level design, multiprocessor-system on chip, real time scheduling, low-power, reliability, thermal cycles, co-simulation

Glossaire

CT	Cycle Thermique
DM	Deadline Monotonic
DPM	Dynamic Power Management
DVFS	Dynamic Voltage and Frequency Scaling
EDF	Earliest Deadline First
EDZL	Earliest Deadline until Zero First
EM	Elecro-migration
LLF	Least Laxity First
MTTF	Mean Time To Failure
RM	Rate Monotic
RR	Round Robin
TDDB	Time Dependant Dielectric Breakdown
WCET	Worst-case Execution Time

Remerciements

Trois années viennent de s’achever, et durant, j’ai vécu une très belle expérience, que ce soit au niveau scientifique ou au niveau humain. J’aimerai remercier toutes les personnes qui ont contribué de près ou de loin à ce travail et à cette aventure.

Dans ma thèse, j’ai eu la chance d’être encadré par une personne dont son niveau scientifique et son expertise atteignent la perfection, s’ajoute avec, ses qualités humaines exemplaires qu’il a présentées. Je suis ainsi profondément reconnaissant à mon directeur de thèse Michel Auguin, pour sa confiance qu’il m’a accordé, ses orientations et ses conseils pertinent jusqu’au jour de ma soutenance. Je n’oublierai pas nos discussions régulières surtout les matins, le temps qu’il m’a consacré malgré toutes ses responsabilités. Son charisme impressionnant avec sa simplicité et gentillesse que je les admire !

Sincèrement, c’était un plaisir énorme et un privilège pour moi de travailler avec lui et d'apprendre de lui.

Je tiens à remercier M. Daniel Chillet, Maître de conférences à l’Université de Rennes, et M. Jean-Philippe Diguet, Directeur de recherche CNRS à Lorient, pour l’attention qu’ils ont portée à mon travail. Je leur suis très reconnaissant d’avoir accepté de juger mon travail en me faisant l’honneur d’être mes rapporteurs de thèse.

Je tiens aussi à remercier M. Yvon Trinquet, Professeur à l’Université de Nantes, qui a accepté de participer aux membres de jury et qui n’a cessé de soutenir ce travail par ses remarques dans le cadre de nos réunions de projet.

Un grand merci à Mme Cécile Belleudy, Maître de conférences à l’Université de Nice Sophia-antipolis d’être parmi le jury de ma thèse et avec qui j’ai eu la chance de faire mon stage de fin d’étude sur AADL. Merci pour tous les encouragements, les remarques, les conseils et les retours sur mon manuscrit !

Ce travail a été soutenu par l'Agence Nationale française de la Recherche (ANR) sous le nom RESPECTED. Un grand merci à tous les partenaires académiques et industriels de ce projet.

A présent je tiens à remercier tous les membres du laboratoire LEAT, dans lequel j'ai passé mes trois ans de thèse, tous les permanents, les non permanent, les stagiaires... Merci pour tout ce temps passé ainsi que les nombreuses discussions, souvent vives et intéressantes !

Ça sera difficile de citer tous les noms, mais je veux remercier principalement Mehdi Abderrahmen un ami et un frère qui a resté toujours près de moi, en arrière-plan, m'a soutenu et orienté durant mon parcours. Tous mes amis à Sophia Antipolis et en Tunisie, Merci beaucoup!

Je finirai par remercier ma famille de l'autre côté de la Méditerranée, tous mes cousins et cousines, mon oncle Chafik en Allemagne, ma tante à Paris, et leurs familles, je vous aime tous !

Finalement, un énorme et un grand merci à mes parents; Abdelkefi et Salwa, que tous les mots ne suffisent pas pour leur remercier, ils m'ont toujours soutenu et n'ont jamais cessé de m'encourager et grâce à eux que je suis arrivé à ce stade ! C'est donc tout naturellement que ce manuscrit leur soit dédié.

Mes frères Ahmed et Ghazi, ma sœur Sihem, mes grands-parents je vous aime beaucoup.

Ma chérie Hajer, qui a eu la lourde tâche de me supporter, surtout à la fin de ma thèse. Sa patience, son aide et sa compréhension m'ont permis de mener bien cette thèse...Merci beaucoup.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

CHAPITRE 1. Introduction

Ce travail de thèse a été réalisé au sein du Laboratoire d'Electronique, Antennes et Télécommunications de l'Université de Nice Sophia Antipolis et du CNRS. Il s'intègre dans le cadre du projet ANR Respected[1].

Le thème abordé dans cette étude s'articule autour de l'analyse thermique et de l'ordonnancement temps réel pour des systèmes multiprocesseurs sur puce.

Dans ce chapitre d'introduction, nous présentons tout d'abord le contexte général de l'étude et la problématique abordée. Les principaux objectifs de cette thèse sont ensuite décrits ainsi que le plan de ce mémoire rassemblant la présentation des études et résultats mis en œuvre.

1.1 Contexte général de l'étude

Réduire la consommation d'énergie a été l'un des sujets de recherche importants ces dernières années dans le domaine des systèmes embarqués. En effet, la tendance concernant la technologie des transistors est une réduction continuelle de leur taille avec pour résultat un nombre moyen toujours croissant de transistors intégrés sur une puce (e.g. $7{,}9 \times 10^6$ transistors par mm^2 en 2013 dans le circuit Intel Haswell GT2 4C et $2{,}6 \times 10^6$ transistors par mm^2 en 2009 dans le Intel Lynnfield 4C). La miniaturisation des transistors n'a pas que des effets positifs car elle s'accompagne d'une certaine dégradation de leurs caractéristiques ce qui, conjointement avec le nombre plus élevé de transistors par unité de surface, a conduit à une augmentation de la puissance dissipée. La diminution de la taille des transistors ne s'accompagne plus depuis quelques années d'une augmentation de la fréquence d'horloge des processeurs justement à cause de cette puissance dissipée qui a atteint une limite. Aussi, la complexité croissante des applications temps réel implique de plus en plus l'utilisation d'architectures multiprocesseurs. Ces applications concernent par exemple les domaines du multimédia ou des télécommunications qui nécessitent souvent des calculs intensifs réalisés à des fréquences de fonctionnement élevées. Ainsi, l'augmentation du nombre de cœurs de traitement intégrés dans les architectures multiprocesseurs (Figure 1.1) associée à l'activité de traitement induite par les applications a conduit à une augmentation de la densité de puissance développée dans les architectures pour atteindre presque un doublement tous les trois ans [1][2].

[1] Projet ANR Respected : http://anr-respected.laas.fr/

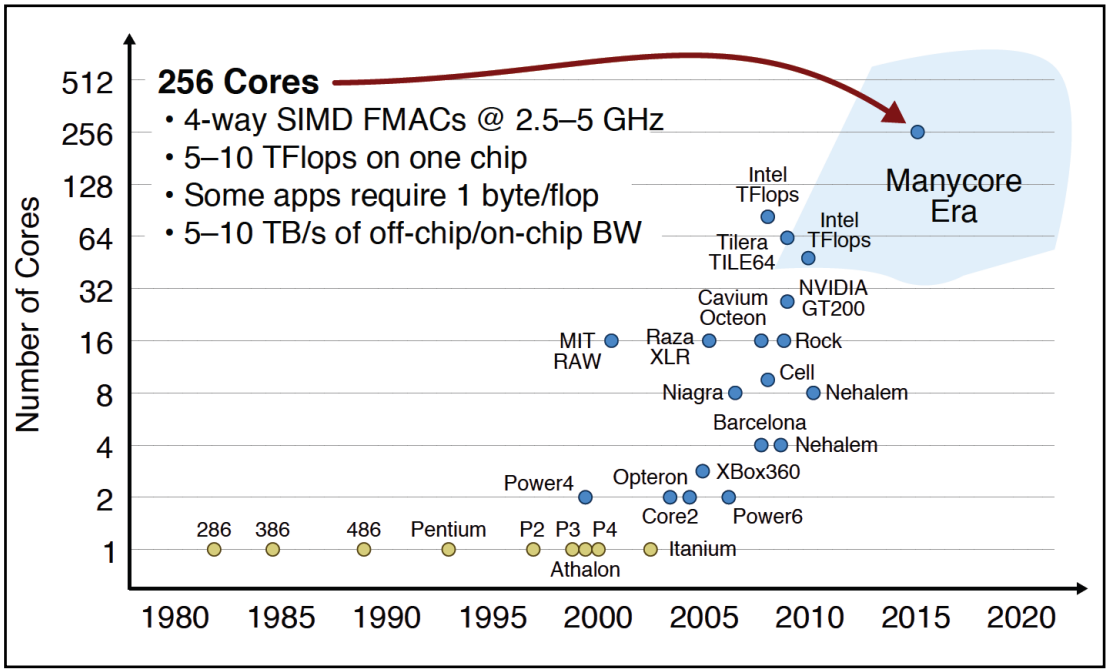


Figure 1.1: Evolution du nombre de cœurs intégrés dans les architectures [3]

La consommation énergétique de ces équipements électroniques qui augmentant de façon constante, il en résulte une élévation de la température du circuit, qui, si elle n'est pas contrôlée, peut causer des erreurs fonctionnelles inattendues ou des dommages permanents aux circuits [4].

Ainsi, de plus en plus, la consommation d'énergie et la température sont des contraintes considérées très tôt lors de la conception de ces systèmes.

Dans la suite de ce chapitre, nous définissons plus précisément la problématique abordée dans cette thèse et les objectifs associés.

1.2 Problématique

Les architectures multiprocesseurs sur puce (MPSoC) ont déjà fait leur apparition il y a quelques années y compris pour des systèmes embarqués. Par exemple, les réalisations MPCore ou plus récemment Cortex A9 de ARM qui possèdent des mécanismes de type DPM (Dynamic Power Management) et DVFS (Dynamic Voltage and Frequency Scaling) sont capables d'adapter la puissance de calcul aux besoins et ainsi réduire la consommation d'énergie dès que la demande en traitement est inférieure à la puissance de calcul maximale fournie par le système.

Cependant, il apparait qu'avec les technologies actuelles, la fiabilité de ces circuits devient plus dépendante des conditions dans lesquelles ils sont utilisés. Ainsi il est montré [5] que le nombre de cycles thermiques (CT) et l'amplitude de ces cycles (suites de montées et descentes en température d'une partie du circuit) a un impact significatif sur la fiabilité du circuit. Ainsi le MTTF (*Mean Time To Failure*) qui représente le temps moyen de bon fonctionnement du système est directement dépendant de ces cycles thermiques. Paradoxalement, le fait de chercher à réduire la consommation d'énergie par utilisation des techniques de type DPM et DVFS conduit à augmenter fortement le nombre de cycles thermiques subits par le circuit. Ainsi dans [6] est illustré l'impact sur la fiabilité de la gestion par DPM et DVFS appliquée à un circuit en technologie 95nm. On peut observer (dans la Figure 1.2) que les cycles thermiques (*Thermal Cycling TC*) ont la plus forte influence sur le MTTF (par comparaison au phénomène d'électro-migration – EM - par exemple) et cette influence ne fait qu'augmenter en considérant des technologies plus fines (e.g. 65nm). L'ampleur des cycles thermiques influe également sur le MTTF comme le mentionne les résultats publiés dans [7]. Du fait de la relation entre puissance statique, due aux courants de court-circuit dans les transistors, et température, cette ampleur dépend de la température ambiante. La technologie utilisée influence également le MTTF. Par exemple, l'échauffement du circuit induit par l'exécution d'un code sur un processeur réalisé en technologie 65nm peut conduire à un MTTF inférieur de 5 ans à celui du même processeur réalisé en 90nm (qui atteint 20 ans dans ce cas) [7].

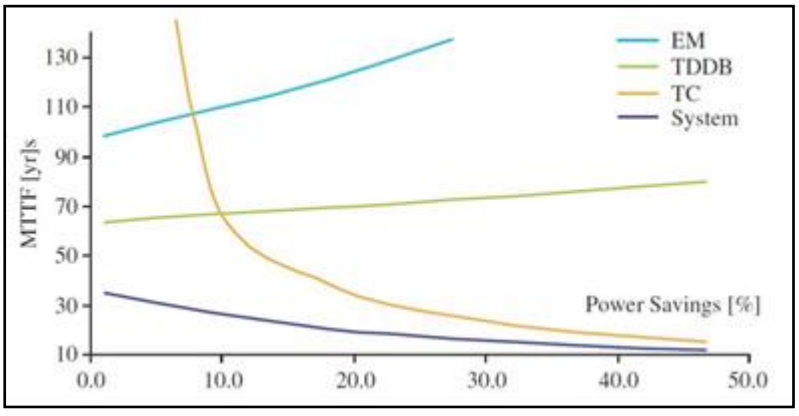


Figure 1.2 : Impact sur le MTTF des techniques classiques de réduction d'énergie/puissance

Dans le cadre de l'électronique embarquée dans l'automobile, les circuits sont soumis à des amplitudes de température qui peuvent être importantes du fait des fortes variations possibles de la température ambiante.

Ces variations sont à la fois dues à l'utilisation du véhicule, à l'emplacement du système électronique au sein du véhicule et aux conditions climatiques (hiver, été, exposition au soleil par exemple). Ces variations induisent des cycles thermiques dits longs. Par ailleurs, avec l'activité en fonctionnement des processeurs embarqués, la température peut fluctuer suivant la charge instantanée à laquelle sont soumis les processeurs : cycles thermiques courts. Comme indiqué ci-dessus, si des mécanismes de type DVFS et DPM sont utilisés alors le nombre de cycles thermiques courts tend à croître. Avec l'augmentation de la fiabilité mécanique moyenne des véhicules, il serait étrange que l'électronique devienne le maillon faible de l'ensemble, du fait d'une absence de contrôle de la température de ses composants.

Outre la fiabilité, les systèmes embarqués peuvent avoir à respecter l'aspect temps réel des applications qu'ils exécutent. En effet, au vu de la sensibilité de certains systèmes, par exemple dans des applications dédiées à l'aéronautique ou l'automobile, le non-respect des échéances des tâches peut engendrer des problèmes graves, et même un disfonctionnement total du système. Ainsi, outre l'objectif de réduction des effets thermiques, il est important voire primordial dans certaines applications de considérer et de vérifier dans la phase de conception que l'ordonnancement des tâches suivant leurs contraintes temporelles soit assuré.

1.3 Objectifs

Deux principaux objectifs sont considérés dans notre travail : réduire les cycles thermiques dans une architecture multiprocesseur sur puce (MPSoC) et satisfaire les contraintes de temps des tâches.

1.3.1 Réduction des cycles thermiques

L'un des deux objectifs dans notre travail est de diminuer les cycles thermiques subis par les cœurs de processeur dans le but d'augmenter la fiabilité des systèmes. Ceci nécessite une étude des modèles thermiques des processeurs et de leurs comportements en température vis-à-vis de l'ordonnancement des tâches. De manière conjointe, on vise également à limiter

l'amplitude des pics de température (*hot spots* [8]) lorsque celle-ci tend à augmenter fortement. Ces phénomènes de *hot spots* localisés sur une partie d'un circuit impactent également négativement la fiabilité des systèmes.

Un modèle thermique permettant d'évaluer les variations en température des différentes parties (cœurs) d'un circuit est ici nécessaire pour identifier les cycles thermiques induits par les variations d'activité de ces cœurs.

L'identification des cycles thermiques permet ensuite d'évaluer leur impact sur la fiabilité du circuit.

La définition du modèle thermique d'une architecture multiprocesseur implique d'identifier et de caractériser l'ensemble des paramètres significatifs qui interviennent dans l'évaluation en température des parties du circuit. Pour mettre en œuvre ce modèle et réaliser des évaluations des comportements thermiques, il est nécessaire de considérer un simulateur capable de résoudre ces modèles en considérant l'aspect dynamique lié aux variations de puissance dissipée par les processeurs qui elles-mêmes dépendent des variations de leurs charges de calcul.

1.3.2 Maintenir l'aspect temps réel

L'objectif, ou plutôt la contrainte dure à considérer dans ce travail, est de maintenir et garantir l'aspect temps réel des applications. En effet, nous cherchons à associer l'évaluation des comportements temporels et énergétique à un modèle de dissipation thermique de l'énergie consommée dans le but d'estimer au cours de l'exécution des tâches l'évolution de la température des différents cœurs de processeurs en fonction de la politique d'ordonnancement considérée. L'évaluation des comportements temporels est alors le premier facteur à considérer dans les résultats : dans le cas d'au moins un dépassement d'échéance par une tâche, la solution est rejetée.

Sur cette base de travail, une stratégie de réduction de la température et des cycles thermiques est proposée. Elle s'appuie sur des techniques de migration de tâches, de DPM ou de DVFS. Cependant considérer ces deux derniers mécanismes de réduction de la puissance dissipée peut engendrer des dépassements des échéances des tâches. Ainsi, la stratégie adoptée doit être capable d'assurer que l'interaction avec l'algorithme d'ordonnancement constitue un ensemble fonctionnel du point de vue des échéances des tâches.

1.4 Plan du mémoire

Le plan de la mémoire est structuré comme suit. Dans le chapitre 2 nous présentons une vue sur les systèmes temps réel et le modèle de tâches considéré. Un état de l'art sur les travaux relatifs aux aspects thermiques, une étude sur les simulateurs associés et les simulateurs d'ordonnancement sont présentés.

Dans le chapitre 3 nous présentons le flot de modélisation proposé sur lequel s'appuient nos travaux. Un modèle thermique de l'architecture multi-cœurs est présenté, ensuite un environnement de co-simulation fonctionnel-thermique (STORM-ATMI) est décrit.

Dans le chapitre 4, une stratégie de réduction des cycles thermiques liée à l'ordonnancement est présentée. L'intérêt de cette stratégie est qu'elle peut interagir avec de nombreux algorithmes d'ordonnancement globaux.

Cette stratégie se base sur les techniques de DVFS et de DPM, ainsi que sur la migration des tâches entre processeurs. Ceci nécessite que l'architecture soit homogène, c'est-à-dire que les cœurs doivent être de type identique et partager une même mémoire commune.

Dans le chapitre 5 nous montrons les résultats de cette stratégie où nous évaluons les améliorations obtenues sur les nombres de cycles thermiques, ainsi que sur la température maximale des cœurs.

Notre technique est appliquée sur trois algorithmes globaux (G-EDF, G-RM et G-EDZL), pour différents scénarios de variation de la température ambiante et sous différentes charges de calcul des processeurs.

Enfin, le chapitre 6 conclut cette thèse et identifie des perspectives pour de futurs travaux de recherches.

CHAPITRE 2. Etat de l’art

2.1 Introduction

Dans ce chapitre nous présentons les caractéristiques des systèmes temps réel, systèmes qui sont de plus en plus présents dans notre vie quotidienne (télécommunication, espace, automobile..). Ces systèmes souffrent de problèmes relatifs à la température qui est la conséquence triviale de l'activité des processeurs et qui peut engendrer des dommages dans la structure physique des circuits et donc agir défavorablement sur leur fiabilité. Ainsi nous détaillons ci-après les travaux qui abordent ce type de problème.

Une étude sur les simulateurs thermiques et d'ordonnancement de tâches est présentée à la fin de ce chapitre.

2.2 Les systèmes temps réel

En informatique temps réel, le comportement correct d'un système dépend, non seulement des résultats logiques des traitements, mais aussi des dates de production de ces résultats [9].

Un système temps réel se compose généralement de plusieurs processus aussi appelés tâches modélisés classiquement par un ensemble de caractéristiques temporelles. Ces tâches sont gérées par un algorithme d'ordonnancement, chacune de ces tâches se traduit alors par plusieurs instances exécutées dans le temps, où chaque instance est appelée *job*.

Ainsi la spécification du comportement d'un système peut se définir par les différentes caractéristiques de ces tâches, soit:

– τ: l'ensemble de n tâches présentes dans le système.

– τ_i: la tâche de numéro i.

– A_i: sa date d'activation.

– S_i: la date de début d'exécution de la tâche.

– F_i: la date de fin d'exécution de la tâche.

– C_i: WCET (worst-case execution time) ou la durée d'exécution pire cas.

– P_i: sa période d'activation.

– D_i: son délai critique (*relative deadline*) ou échéance qui définit la contrainte temporelle de la tâche.

Les caractéristiques temporelles des tâches, ainsi que leurs dates de réveil et leurs échéances absolues sont représentées graphiquement dans la Figure 2.1.

Plusieurs autres paramètres permettent de caractériser les tâches, comme le facteur d'utilisation u_i de chaque tâche τ_i ($u_i=c_i/p_i$) et le facteur d'utilisation total du système : $U=\Sigma u_i$. La laxité d'une tâche τ_i, notée L_i, est un paramètre pertinent dans l'ordonnancement, il se déduit directement des échéances et des valeurs des WCET. On différencie la laxité relative $L_i=D_i-C_i$ et la laxité dynamique absolue $L_i(t) = D_i(t)-C_i + C_i(t)$ où $C_i(t)$ est le temps d'exécution restant de τ_i vis-à-vis de C_i à exécuter à l'instant t.

Tous ces paramètres définissent le modèle classique de tâches, au sein duquel chaque tâche peut avoir l'un des trois modèles d'activation suivants :

Le modèle **Périodique**, introduit par Liu et Layland [10] fournit la notion la plus simple, où les activations de deux instances successives d'une tâche sont séparées par un intervalle de temps constant.

Le modèle **Sporadique** est similaire à celui des tâches périodiques, avec la différence que la période correspond à la durée minimum entre deux activations successives (non nécessairement périodique) d'une tâche τ_i.

Enfin, le modèle **Apériodique** où l'activation des tâches est déclenchée par exemple par un événement externe, ainsi la durée entre deux événements successifs d'activation n'est pas connue a priori.

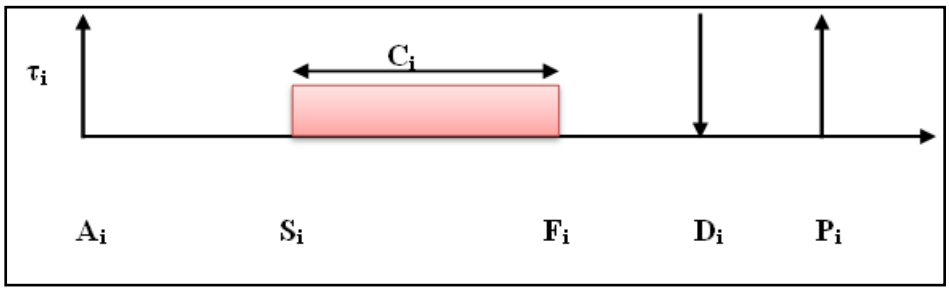


Figure 2.1 : Notations et caractéristiques de base d'une tâche

Ainsi le problème de l'ordonnancement repose sur la définition précise de ces modèles de tâches et des caractéristiques des plateformes d'exécution cibles. Les définitions des algorithmes d'ordonnancement et des tests d'ordonnançabilité associés sont fortement liées aux caractéristiques du modèle de tâches considéré. Nous donnons ci-après les principaux éléments permettant de particulariser un modèle de tâches :

– Échéance sur requête : correspond au cas $D_i = P_i$;

– Tâches concrètes : toutes les dates de réveil (activation) des tâches sont connues ;

– Tâches préemptives : les tâches peuvent être préemptées par toute tâche, à n'importe quel instant de leur exécution ;

– Tâches indépendantes : les tâches ne partagent aucune ressources communes, à part la ressource processeur, et ne sont liées par aucune relation de précédence explicite ;

– Tâches à suspension : une tâche peut se suspendre par exemple durant une opération d'entrée-sortie et reprendre son exécution lorsque celle-ci est terminée.

Selon le respect attendu des contraintes temporelles, les systèmes temps réel peuvent être classés en deux catégories :

Systèmes temps réel dur ou critique : les échéances sont dans ce cas cruciales c'est à dire qu'elles doivent être respectées dans tous les scénarios de travail pour éviter des conséquences catastrophiques.

Systèmes temps réel mou ou souple : sont ceux dont les échéances sont moins critiques tel que le dépassement d'échéances peut être toléré avec généralement un effet réduit sur les performances du système.

2.2.1 Ordonnancement temps réel

L'ordonnanceur joue le rôle d'un planificateur et doit donc attribuer les ressources (processeurs) aux tâches au cours du temps, selon une politique d'ordonnancement. L'objectif premier de l'ordonnancement, donc de l'algorithme qui gère l'affectation dans le temps des exécutions des tâches, est de respecter l'ensemble des contraintes temporelles associées aux tâches.

De plus, si cela est possible, il peut être primordial pour certains systèmes critiques de pouvoir valider le respect des contraintes des tâches par une analyse hors ligne de l'application : c'est l'analyse d'ordonnançabilité. Cette analyse est très fortement dépendante du modèle des tâches de l'application. Elle peut permettre également d'étudier l'optimalité d'un tel algorithme. En effet, un algorithme est dit optimal, s'il est démontré qu'il assure le respect de toutes les échéances des tâches et s'il est capable d'ordonnancer toute configuration ordonnançable déduite du modèle de tâche considéré.

Par exemple, un ensemble de tâches périodiques et indépendantes, à échéances sur requêtes, ordonnancé sur un système monoprocesseur suivant l'algorithme EDF (Earliest Deadline

First) [10] , un algorithme à priorité fixe au niveau *job* qui donne la priorité la plus grande à la tâches qui a l'échéance la plus proche), est ordonnançable si et seulement si la condition suivante est vérifiée [10] :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Classiquement on distingue deux types d'ordonnancement : ordonnancements hors ligne et en ligne.

Dans l'ordonnancement hors ligne, la séquence d'ordonnancement des tâches est pré-calculée dans une table avant l'exécution effective. A l'exécution, l'ordonnanceur est un simple séquenceur. Dans ce cas, il est nécessaire d'avoir une connaissance exacte sur les dates d'activation et les périodicités des tâches, ce qui favorise ainsi le choix d'un modèle périodique. L'ordonnancement hors ligne permet de détecter aisément la présence de dépassements d'échéances mais en contrepartie il n'est pas flexible du moment où l'on doit connaitre à l'avance les instants exacts d'activation des tâches dans le système. De plus, sauvegarder les séquences d'ordonnancement pré-calculées dans des tables peut présenter des contraintes de réalisation significatives.

En revanche, dans l'ordonnancement en ligne, les décisions d'ordonnancement sont prises au cours de l'exécution. A l'exécution, l'ordonnanceur exécute un algorithme d'ordonnancement permettant d'identifier à chaque événement d'ordonnancement quelle tâche il s'agit d'exécuter. Les ordonnanceurs en ligne, même s'ils sont plus complexes à implémenter et à gérer que les ordonnanceurs hors ligne, demeurent plus flexibles vis-à-vis des caractéristiques des tâches.

2.2.2 Stratégies d'ordonnancement

Dans le cas d'un système monoprocesseur, le problème à résoudre est d'affecter la ressource processeur à l'exécution des tâches tel que toutes les échéances des tâches sont respectées. Pour résoudre ce problème on considère classiquement dans les ordonnanceurs une seule file d'attente où les tâches prêtes sont ordonnées selon un critère d'ordonnancement qui vise à satisfaire les contraintes d'échéances. De très nombreux algorithmes d'ordonnancement ont été proposés dans la littérature comme RR (Round Robin) [11], RM

(Rate Monotic) [10], DM (Deadline Monotonic) [10], LLF (Least Laxity First) [12], EDF[10]. Par exemple RM, LLF et EDF sont démontrés optimaux suivant leurs politiques respectives d'attribution de priorités pour des configurations de tâches périodiques à échéances sur requêtes. Ces résultats sont obtenus dans le cas monoprocesseur.

Par contre, dès que l'architecture possède plus d'un processeur, le problème d'ordonnancement devient plus compliqué car il implique de sélectionner sur quel processeur et à quel instant exécuter une tâche donnée. On est ainsi amené à traiter un problème d'affectation de tâches à la fois temporel et spatial. Pour aborder ce problème, deux types d'ordonnancement existent basés sur la manière dont les tâches sont assignées aux processeurs au moment de leur exécution : on parle alors d'ordonnancement partitionné ou d'ordonnancement global. En ce qui concerne les méthodes basées sur l'approche avec partitionnement, toutes les tâches dans une partition sont exécutées sur un même processeur. La répartition des tâches aux processeurs est déterminée via un algorithme de partitionnement qui réalise cette affectation suivant un ou plusieurs critères donnés (e.g. équilibrage de la charge de travail entre les processeurs). Quant aux autres méthodes adaptées à un ordonnancement global, elles considèrent en particulier que chaque tâche peut migrer entre processeurs. Il existe alors deux types de migration :

- Une migration forte où une tâche une fois préemptée peut reprendre son exécution sur un autre processeur.
- Une migration faible où une fois qu'une tâche a démarré son exécution sur un processeur, elle poursuit son exécution jusqu'à sa terminaison sur ce même processeur

2.2.2.1 Ordonnancement avec tâches partitionnées

En réalisant un partitionnement d'un ensemble de tâches, on assigne chacune d'elles à un processeur déterminé. Lors de l'exécution, chaque tâche s'exécute uniquement sur le processeur qui lui a été associé (c'est à dire sans migration) selon une stratégie d'ordonnancement locale et monoprocesseur. Chaque processeur possède dans ce cas sa propre file de tâches prêtes qui ne peut contenir que des tâches prêtes parmi celles qui lui sont assignées. Ainsi, le problème d'ordonnancement multiprocesseur est résolu en deux temps : le partitionnement proprement dit puis l'ordonnancement local. Pour ce dernier problème ce sont des stratégies classiques d'ordonnancement monoprocesseur comme RM et EDF qui sont utilisées. Pour que l'ordonnancement de l'ensemble des tâches soit valide, le partitionnement doit être conduit de telle sorte que pour chaque processeur, l'ordonnançabilité locale des

tâches qui lui sont allouées soit assurée. Dans le cas de tâches indépendantes on peut aisément vérifier l'ordonnançabilité globale par une vérification locale à chaque processeur mais l'optimalité en termes d'utilisation de ressources processeur n'est pas garantie avec cette approche. Cependant la simplicité de l'approche avec partitionnement et son déterminisme en font une solution utilisée fréquemment dans la pratique, comme cela est préconisé dans le standard AUTOSAR pour l'automobile [13].

2.2.2.2 Ordonnancement global des tâches

Contrairement à l'ordonnancement partitionné, une seule file d'attente pour les tâches prêtes est présente dans le système, et un seul ordonnanceur responsable de l'allocation spatiale et temporelle des tâches est utilisé. Pour améliorer l'utilisation des ressources processeur et donner plus de flexibilité à l'ordonnanceur dans ses choix, la préemption des tâches est généralement admise et la migration de ces dernières est aussi autorisée. Ainsi une tâche peut dès lors commencer son exécution sur un processeur puis la reprendre, après préemption, sur un autre processeur (migration forte). Ainsi les algorithmes développés pour le cas monoprocesseur peuvent aisément être étendus au cas multiprocesseur : par exemple global-RM [14], global-EDF [15] ou des adaptations de ces algorithmes comme RM-US [16] ou EDZL [17].

Cependant, l'optimalité des algorithmes d'ordonnancement monoprocesseur n'est généralement pas vérifiée dans le cas multiprocesseur (e.g. RM, EDF). Ainsi l'ajout de processeurs additionnels dans une architecture matérielle introduit une nouvelle dimension au problème de l'ordonnancement [18]. Par conséquent, une nouvelle classe d'algorithmes basée sur l'ordonnancement fluide (ou *fairness*) a été développée permettant d'exploiter de façon optimale les ressources processeurs. Dans cette classe on retrouve par exemple les algorithmes présentés dans [19] ou [20]. Cependant ces algorithmes basés sur un ordonnancement fluide sont reconnus pour produire un nombre important de préemptions et de migrations. Des techniques plus récentes d'ordonnancement global n'exploitant pas la *fairness* tout en conservant cette propriété d'optimalité ont été proposées, par exemple RUN [21] et U-EDF [22].

2.3 Effets de la température

Dans le paragraphe précédent nous avons brièvement décrit les modèles et algorithmes classiques relatifs aux systèmes temps réel et leurs caractéristiques.

Dans ce paragraphe nous décrivons la problématique reliée à la température, qui est une conséquence triviale de l'activité des processeurs avec des conséquences potentiellement néfastes sur la fiabilité des circuits intégrés.

2.3.1 Le problème de la température

Comme indiqué ci-dessus, l'ordonnancement de tâches consiste à décider aux instants définis en particulier par la technique d'ordonnancement choisie, parmi l'ensemble des tâches, celles qui doivent s'exécuter à ces instants et de manière conjointe sur quels cœurs de processeurs elles sont affectées dans le cas d'une architecture multi-cœurs. Ainsi l'évolution en température du circuit sera dépendante de ces choix d'ordonnancement et de l'activité des processeurs produite par l'exécution individuelle des tâches.

Cette influence de la température sur l'architecture se manifeste par [23] :

- Les performances électriques : la température du circuit admet une valeur limite au-delà de laquelle le fonctionnement du système n'est plus garanti. Des dérives des paramètres fonctionnels liées à l'élévation de température provoquent une diminution des performances pouvant aller jusqu'à la défaillance fonctionnelle du système.
- L'activité des unités localisées sur ou dans le circuit génère une puissance dissipée qui produit localement une élévation de température. Ainsi de fortes activités localisées peuvent produire des points chauds dans le circuit avec pour conséquence des gradients de température entre les différentes zones actives et inactives. Les déformations physiques dues aux gradients de température dans le circuit font subir des contraintes dans les matériaux qui produisent des effets accélérés sur le vieillissement du fait des structures plus fines utilisées dans les technologies avancées. Ces déformations dues à la température peuvent conduire à des ruptures d'éléments dans le circuit.
- Les cycles thermiques, auxquels sont soumis les matériaux utilisés dans la réalisation des circuits qui ont des coefficients de dilatation différents, induisent des forces dans

la structure qui peuvent conduire à une rupture instantanée ou créer une fatigue des matériaux qui provoque une rupture à longs termes.

2.3.1.1 Impact de la température sur la fiabilité des systèmes

La fiabilité des composants électroniques est fortement liée aux variations de température qu'ils subissent. En effet, sept mécanismes de défaillance sont cités dans la littérature [24] qui peuvent être les causes de la diminution du MMTF (Mean Time To Failure). Ici nous citons trois principaux mécanismes : *Electromigration* (EM), *Time Dependant Dielectric Breakdown* (TDDB) et *Thermal Cycles* (TC).

- Electromigration

Ce phénomène correspond au déplacement d'atomes dans les conducteurs induit par le flux d'électrons qui provoque des défauts permanents tels que l'ouverture des lignes métalliques ou contacts, le raccourcissement entre les lignes métalliques adjacentes.

Cette défaillance apparaît suite à une forte densité de courant en relation avec la section du conducteur. Généralement l'expression du MTTF associé à ce mécanisme est décrite par le modèle dit de Black :

$$\text{MTTF}= A(J - J_{crit})^{-n} e^{(E_a/KT)}$$

Où A est une constant, J est la densité de courant, J_{crit} est la densité de courant de seuil,

K est la constante de Boltzmann, E_a l'énergie d'activation et T est la température.

- *Time Dependent Dielectric Breakdown* (TDDB)

Ce mécanisme entraîne la formation de pistes conductrices liée à la réduction du diélectrique de grille pouvant conduire à une rupture de ce diélectrique avec création de courants de grille et la perte de contrôle du courant de drain dans le transistor. Il possède comme modèle MTTF :

$$\text{MTTF}= Ae^{-\gamma E_{ox}} e^{(E_a/KT)}$$

Où γ est le paramètre d'accélération du phénomène, E_{ox} est le champ électrique à travers le diélectrique.

- *Temperature cycling*

Ce mécanisme se produit suite à une succession de variations de puissance dissipée entraînant des montées et des descentes locales de température. Des travaux montrent, que ces cycles jouent un rôle majeur dans la fissuration des couches minces métalliques de la puce et induisent des déformations plastiques des matériaux.

Le nombre de cycles thermiques avant la défaillance est classiquement modélisé par :

$$N_f = C_0 \left[C_1(T_{max} - T_{min}) - C_2(T_{avg,s} - T_{mold})\right]^{-q}$$

Avec $(T_{max} - T_{min})$ la plage de température entre les états d'énergie, $T_{avg,s}$ la température moyenne à l'état « *sleep* » du processeur, T_{mold} la température de moulage de toute la puce. L'exposant q est un paramètre technologique dont sa valeur varie généralement de 6 à 9, et $C_{0,1,2}$ sont des constantes de réalisation pour des structures de type système sur puce.

2.3.1.2 Les cycles thermiques

Dans notre travail nous nous intéressons au mécanisme de défaillance dû aux cycles thermiques, qui peuvent être temporels ou spatiaux :

- Temporels : ils sont la conséquence des variations de température en un point particulier du circuit qui apparaissent séquentiellement au cours du temps suite aux variations de la puissance dissipée par les transistors et/ou l'interconnexion situés en ce point du circuit ou en des points voisins (dans ce cas le cycle est obtenu par diffusion de chaleur). Par exemple si un cœur de processeur passe d'un état actif où il exécute une séquence d'instructions à une fréquence élevée à un état de repos (horloge coupée et tension d'alimentation réduite ou nulle) puis à nouveau à un état actif, alors il y a création d'un cycle thermique. Comme indiqué dans l'introduction, la répétition de ces cycles peut conduire à réduire la fiabilité du circuit.
- Spatiaux : un point chaud sur le circuit situé à côté d'un point plus froid (ou même plus chaud) conduit également à des contraintes mécaniques analogues aux variations de température liées aux cycles thermiques.

La Figure 2.2 décrit un cycle thermique (courbe en rouge) suite à une impulsion de puissance dissipée (courbe en vert).

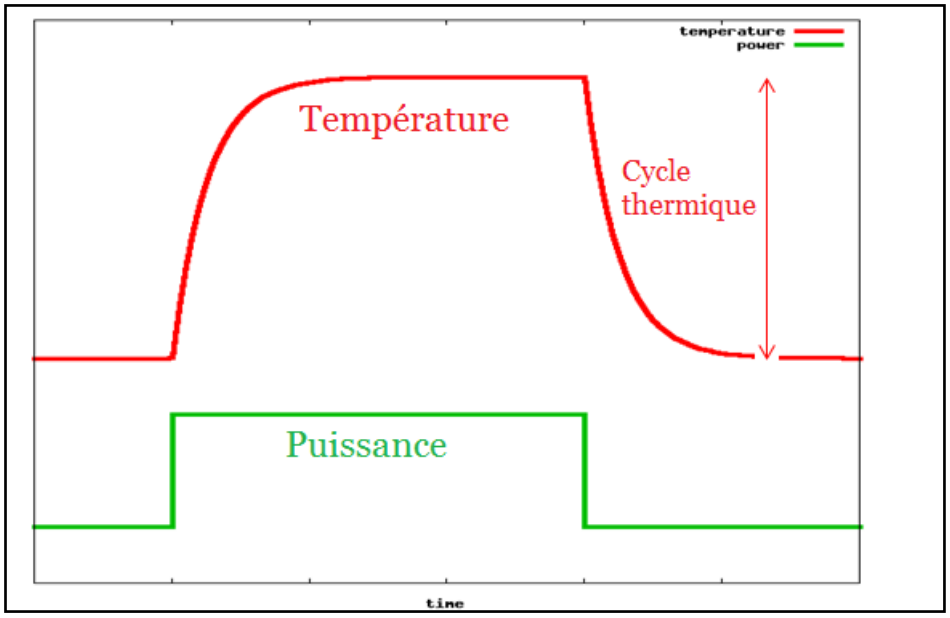


Figure 2.2 : Illustration d'un cycle thermique

2.3.1.3 Variation de la température ambiante

En prenant l'exemple de l'automobile, l'environnement défini par un véhicule peut conduire à considérer dans les phases de conception des variations importantes possibles de la température ambiante que subissent les circuits électroniques embarqués. Cette température ambiante a une influence sur les comportements thermiques des circuits dont il faut tenir compte. On rappelle ici quelques résultats publiés illustrant différentes situations d'exposition à différentes températures de l'électronique embarquée.

Une étude publiée dans [25] consiste à évaluer l'augmentation de la température ambiante à l'intérieur de véhicules automobiles, qui constitue le contexte applicatif support de cette thèse. Dans cette étude le véhicule est exposé au soleil et placé dans un environnement soumis à différentes températures extérieures (entre 73°F et 93°F). La Figure 2.3 illustre la variation de la température à l'intérieur de l'habitacle en fonction du temps. Ainsi, l'augmentation moyenne est de 3°F toutes les cinq minutes et au bout de trente minutes la température interne atteint 140% de la température extérieure. La température finale du véhicule dépend de la température ambiante initiale, par exemple si on prend la température initiale la plus basse

73°F (22.7°C) alors la température intérieure atteint 117°F (47.22°C). De plus, chaque partie du véhicule voit sa température évoluer de manière différente, par exemple le tableau de bord du véhicule est la partie d'habitacle qui subit la plus importante amplitude de température.

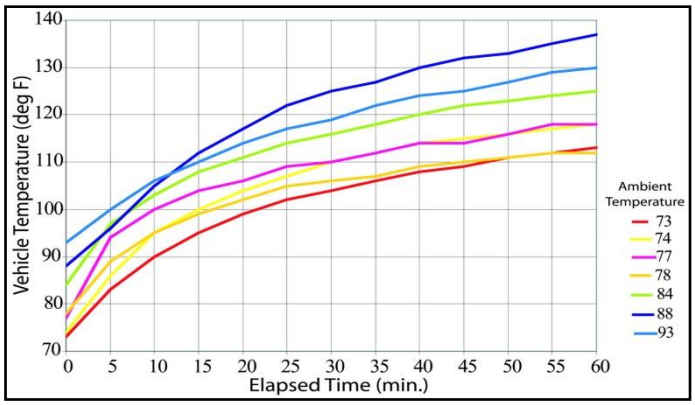


Figure 2.3 : Variation de la température intérieure de l'automobile pour différentes températures extérieures.

Ces études antérieures ont également montré que les jours où la température ambiante dépasse 86°F (30°C), la température intérieure du véhicule peut atteindre rapidement 134 à 154 °F (68°C).

Dans [26], [27] est évalué l'impact de plusieurs paramètres sur la variation de la température intérieure de la voiture. Les paramètres considérés sont par exemple la couleur, l'état des fenêtres (ouvertes ou fermées), l'utilisation d'un pare-soleil posé sur le pare-brise.

La Figure 2.4 montre la variation de la température intérieure du véhicule pour une température ambiante de 32°C avec fenêtres fermées (courbe jaune) et fenêtres ouvertes (courbe rouge) de 4.5cm. Il y a dans cet exemple une différence de plus de 10°C entre ces deux cas.

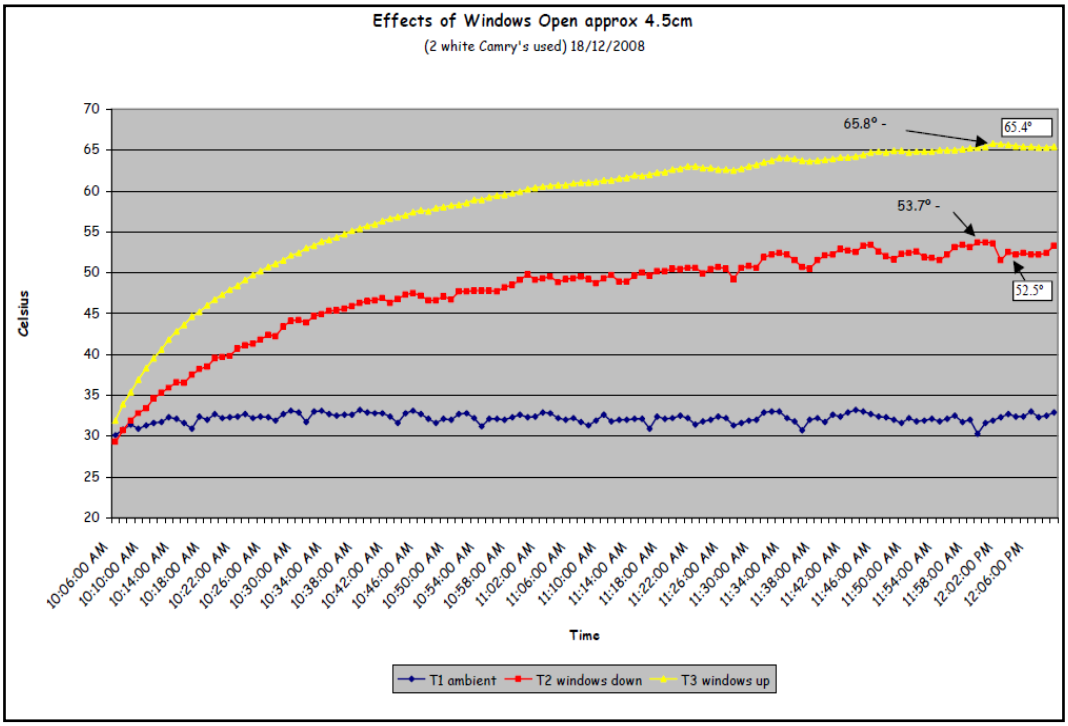


Figure 2.4 : Différences de température intérieure avec fenêtres fermées (courbe jaune) et fenêtres ouvertes (courbe rouge)

Ainsi ces travaux illustrent que les amplitudes de variation de la température ambiante dans les véhicules automobiles peuvent être importantes. Ici la température est mesurée à l'intérieur de l'habitacle. Pour des systèmes électroniques placés proches du moteur thermique les variations seraient encore plus importantes avec de plus des effets oscillants liés à la mise en route de la ventilation forcée.

Pour compléter cette analyse, citons l'étude relative au domaine des processeurs hautes performances [28] réalisée sur un système comportant un Intel Core i7 3.5GHz et un GPU NVIDIA Geforce GTX 670 2GB placé dans une chambre fermée. Une augmentation de la température ambiante de 1°C implique une augmentation de 1.05 °C de la température du CPU et une augmentation de 1.08° C à la température du GPU lorsque le système est en mode repos (*idle*). De plus, l'augmentation de 1°C de la température ambiante produit une augmentation de la vitesse de rotation du système de ventilation de 13,3 tours par minute.

Outre le fait que la température a un effet sur la fiabilité comme montré dans le paragraphe 2.3.1.1, elle joue également sur les courants de fuite des transistors ce qui a un impact sur la puissance dissipée et mène à son tour à une élévation de la température du circuit. Par exemple, la Figure 2.5 [29] illustre l'augmentation significative de la puissance

statique (pour un *die* de 15 mm fabriqué dans une technologie à 100nm et fonctionnant sous une tension d'alimentation de 0,7V) en fonction de la température de la jonction qui est due principalement à la montée exponentielle du courant de fuite (*leackage*).

Si la conductance thermique du boitier et du système de refroidissement n'est pas assez grande, cette dépendance (température, courant de fuite) peut provoquer un emballement thermique [30]. Même si un emballement thermique ne se produit pas, la température de fonctionnement de la puce peut devenir supérieure à la valeur de température maximum préconisée par le constructeur, ce qui par la suite implique soit de dégrader les performances pour limiter l'augmentation de température soit de diminuer la fiabilité à long terme [31].

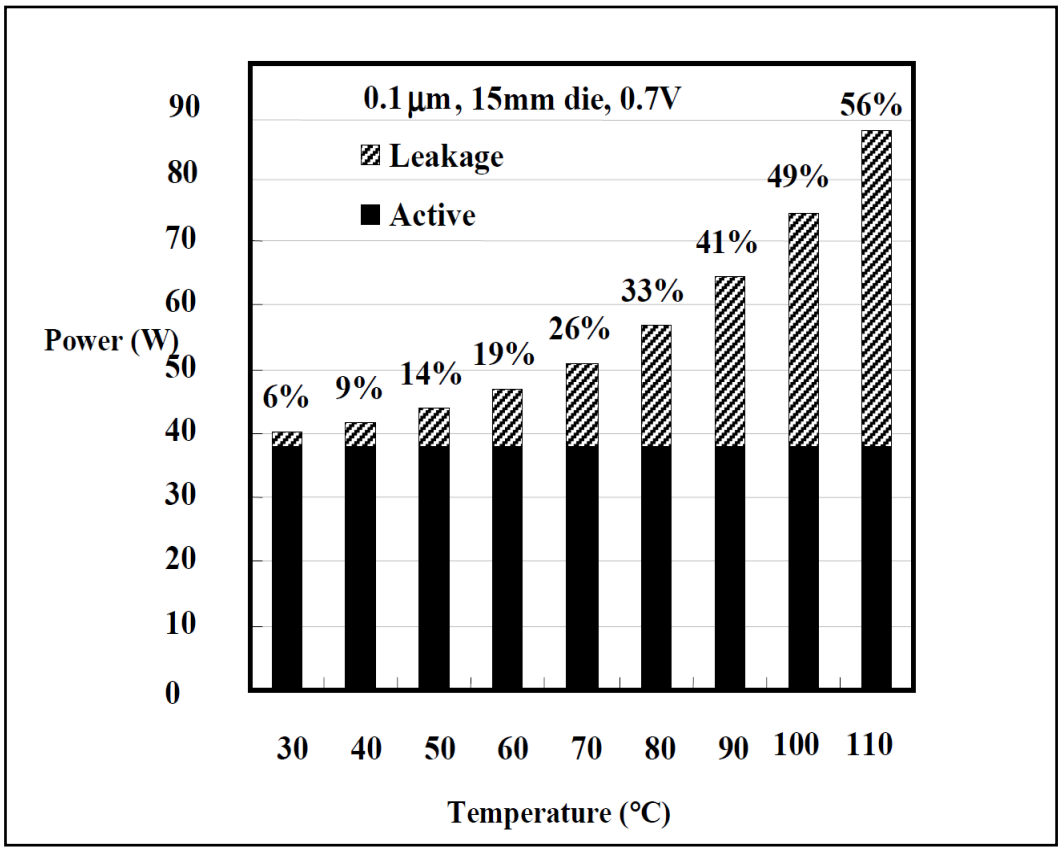


Figure 2.5 : Puissance consommée au niveau du *die* en fonction de la température.

2.3.2 Travaux sur la réduction de la température dans les circuits

Des techniques de gestion thermique ont été explorées à la fois au moment de la conception à travers des boitiers (*packaging*) appropriés et des mécanismes de dissipation de la chaleur, et au moment de l'exécution des applications à travers diverses formes de gestion dynamique de température (DTM : Dynamic Thermal Management).

Cependant, le coût du *packaging* avec système de refroidissement augmente rapidement suivant la quantité de chaleur à dissiper [32]. Des études estiment le surcoût dû à ces systèmes de refroidissement de 1 à 3 dollars par watt dissipé [2].

Comme solution alternative, [33],[34],[2] ont proposé le contrôle de la température pendant l'exécution des traitements en ajustant dynamiquement la consommation de puissance des systèmes.

Pour tenter d'augmenter la fiabilité des circuits en relation avec la température suivant les mécanismes de défaillance définis dans la section 2.3.1.1, des études publiées cherchent principalement à équilibrer la charge (*load balancing*) entre processeurs dans les architectures multiprocesseurs embarquées. Ainsi, dans [35] les auteurs proposent d'utiliser alternativement les deux processeurs de l'architecture considérée pour répartir la dissipation thermique, et ce en prenant comme hypothèse que les tâches s'exécutent en un temps généralement plus petit que leur pire cas. Cependant ce travail ne considère aucune contrainte liée à l'ordonnancement effectif des tâches et à leurs échéances d'exécution. Une technique d'équilibrage de charge entre quatre processeurs est utilisée dans [36] en exploitant les différents états de repos des processeurs.

Une autre technique similaire utilisée dans [37] vise à ordonnancer les tâches en privilégiant le critère température (en utilisant les possibilités de DPM, DVFS et migration) par rapport aux contraintes de temps réel.

Dans [38] l'effet sur la température des migrations de tâches entre processeurs est étudié en tenant compte de l'architecture et du système d'exploitation. Des résultats expérimentaux sur FPGA montrent que la migration est une approche efficace pour contrôler la température, mais aucun objectif de respect d'échéances temporelles n'est considéré dans ces travaux. Des techniques de contrôle optimal sont utilisées dans [39] ou [40] pour réguler la température. Ces techniques donnent de très bons résultats mais d'une part, le contrôle ajouté induit une

dissipation thermique supplémentaire et d'autre part, la garantie du respect des échéances de tâches n'est pas abordée dans ces travaux.

Dans le domaine des systèmes embarqués, des études ont récemment porté sur la synthèse de haut niveau intégrant les phénomènes de température [41] ainsi que l'exploration de l'espace de conception [42]. L'objectif considéré dans ces travaux, est d'optimiser les performances des systèmes en minimisant les pics de température, sans violer les contraintes temporelles des tâches.

Pour les systèmes monoprocesseurs, les auteurs de [43] ont présenté une politique réactive d'ajustement de la vitesse du processeur pour contrôler l'amplitude des pics de température. Afin de garantir les échéances de temps réel, une politique de gestion thermique dynamique a été ensuite proposée [44]. Dans [45] est présentée une politique de contrôle optimal de la vitesse du processeur afin de maximiser sa capacité d'exécution sous contrainte thermique. Le modèle thermique proposé dans cette étude est ensuite amélioré dans [46]. Le travail présenté dans [47] est concentré sur la minimisation de l'énergie sous contraintes thermiques. Les auteurs de [48] ont présenté une condition nécessaire et suffisante d'ordonnançabilité dans un cas monoprocesseur telle que la température du processeur est toujours bornée par une température maximale. Ainsi un nouvel algorithme d'ordonnancement TCEDF (*Temperature-Constrained EDF scheduling*) basé sur EDF est proposé.

Les auteurs de [49] ont été les premiers à travailler sur le problème de la minimisation du pic de température en utilisant la possibilité de changement dynamique et continu de la vitesse du processeur, indépendamment des tâches. Dans [50] est présentée une technique de séquencement des taches afin de minimiser le pic de température pour les systèmes temps réel avec des tâches périodiques.

Pour les systèmes multiprocesseurs, certains travaux se concentrent sur le respect des contraintes des tâches pour les systèmes temps réel dur [51] en considérant des architectures matérielles hétérogènes. D'autres optent pour optimiser la consommation de l'énergie sous contraintes temporelles [52]. La majorité des problèmes d'ordonnancement temps réel est de nature NP-complet, aussi plusieurs heuristiques ont été proposées afin de résoudre ce problème avec différents critères d'optimisation relatifs à la température [53],[54],[55].

Les auteurs de [56] explorent l'ordonnancement temps réel pour les tâches sporadiques afin de minimiser le pic de température dans un système multicœur homogène. Ils dérivent une

vitesse de fonctionnement idéale pour chaque cœur, à travers un système de minimisation. Dans cette étude les algorithmes d'ordonnancement globaux EDF et DM sont considérés.

Les études développées dans [57] présentent une formulation suivant un programme linéaire mixte (MILP) pour partitionner les tâches sur les processeurs en vue de réduire le pic de température. Il est supposé que la consommation de puissance pour chaque tâche sur un processeur est constante.

Malheureusement, tous ces travaux, dans leurs modèles thermiques utilisés, suppose que la température ambiante est constante. De plus ils visent principalement à minimiser le pic de température, et ne tiennent pas compte des mécanismes de défaillance comme par exemple le nombre de cycles thermiques provoqués par les traitements.

2.4 Environnement de l'étude

L'étude sur les algorithmes d'ordonnancement multiprocesseur adaptés à la réduction des cycles thermiques implique de disposer de modèles et/ou de plateformes matérielles d'expérimentation permettant d'évaluer les comportements thermiques d'une architecture multiprocesseur sur laquelle est exécutée un jeu de tâches ordonnancées suivant l'algorithme considéré.

Afin de mener cette étude et de la valider, nous présentons un état de l'art sur les outils de simulation disponibles et l'environnement retenu pour mener nos travaux.

Plusieurs simulateurs thermiques existent dans la littérature, par exemple, HotSpot [58] et Forward Euler (FE) [59] qui utilisent l'analogie classique entre les circuits électriques et la conduction thermique et ce afin de décrire les échanges de chaleurs dans un circuit modélisé par ses résistances et capacités thermiques. Par résolution des équations différentielles associées, l'équilibre thermique dans le modèle peut ainsi être déterminé. Les temps de calcul peuvent être importants suivant la précision souhaitée dans les modèles.

Dans [60] est développée une infrastructure de simulation thermique en proposant une méthode de résolution optimale des équations différentielles tout en réduisant les temps de calcul.

L'outil Magma [61] est un simulateur basé sur l'environnement HotSpot qui vise à étudier le débit maximum que peut fournir une architecture MPSoC sous contrainte de température.

L'outil SESCTherm [62] est un simulateur basé sur la modélisation par différences finies du système en vue de permettre d'évaluer des choix de conception au niveau interconnexion et *packaging*.

Un autre simulateur basé sur la méthode *power blurring* (PB) [63] permet de résoudre le modèle en des temps de calcul de plusieurs ordres de grandeur plus petits par rapport à des approches basées sur la méthode des éléments finis. Cette méthode de *power blurring* a été ensuite étendue dans [64],[65] en appliquant la méthode des images pour tenir compte des symétries dans la diffusion de la chaleur.

Signalons l'existence d'environnements commercialisés de simulation d'effets thermiques dans les circuits électroniques comme AceThermalModeler de la société Docea Power [66].

Une étude dans [67] consiste à comparer l'évolution de la température fournie par trois simulateurs : Hotspot, la méthode PB et SESCTherm. Les erreurs d'estimation de température pour ces trois simulateurs sont définies par rapport aux résultats donnés par le simulateur commercial de ANSYS basé sur la méthode des éléments finis. Les résultats montrent que le simulateur basé sur la méthode *power blurring* produit des estimations de température à l'état stable ou en transitoire les plus précis et dans le temps le plus court pour les trois simulateurs étudiés. Il faut noter que le simulateur HotSpot est très souvent référencé dans les travaux sur l'étude en température des composants électroniques (par exemple [68], [69])

Dans cette étude considérée ici, le simulateur HotSpot produit les résultats les plus éloignés par rapport à ceux donnés par le simulateur ANSYS, au moins sur les exemples étudiés dans [67].

Le simulateur ATMI [65], basé sur la méthode des images a fait l'objet d'une comparaison avec le simulateur HotSpot dans la version 3.0. Dans cette étude [70], il est montré que ATMI, sur les exemples considérés, est plus précis que HotSpot. Par ailleurs, l'approche de modélisation par un réseau de résistances et capacités thermiques implique de connaître l'ensemble des valeurs de ces résistances et capacités qui font du sens vis-à-vis de la modélisation thermique du circuit à étudier. Ainsi un nombre relativement important de paramètres est à renseigner dans le simulateur HotSpot par rapport aux 9 paramètres de ATMI. Classiquement il n'est pas facile d'identifier, et d'accéder à ces valeurs de paramètres car il s'agit de données constructeur sensibles, au moins pour certaines; aussi identifier un grand nombre de ces paramètres s'avère bien évidement plus complexe qu'identifier un nombre plus restreint de paramètres. Les travaux détaillés dans [70] tendent à montrer que le développement d'un modèle thermique sous HotSpot en adéquation avec un système réel

n'est pas aisé et qu'une attention particulière doit être portée sur ce point pour éviter des erreurs de comportements trop importantes.

Par ailleurs, nous nous intéressons dans notre étude aux comportements thermiques issus de l'ordonnancement de tâches sur une architecture multicoeur. Etant donné que les variations de température à l'intérieur d'un composant sont très largement plus lentes comparées au cycle d'horloge du processeur, il n'est pas nécessaire de considérer un modèle thermique ayant une granularité très faible. De plus, pour évaluer les effets des décisions d'ordonnancement de tâches sur la température d'un composant, il est en revanche nécessaire de considérer une fenêtre d'exécution relativement longue (en temps) des tâches. Ceci milite également pour une granularité du modèle relativement élevée afin d'obtenir des temps de simulation acceptables.

Au vu de cette analyse, notre choix d'environnement de simulation pour les aspects thermiques s'est finalement porté sur ATMI. En effet, ce simulateur est simple à prendre en main, il fournit des résultats assez précis, il est aisé de modéliser des circuits du fait des hypothèses de structure qui ont gouverné son développement. De plus, il permet de calculer la dynamique de la température en des temps relativement réduits.

Enfin, pour identifier les cycles thermiques, il est nécessaire de déterminer l'évolution de la température en différents points du circuit. Tenter de réduire ces cycles pendant l'ordonnancement a pour conséquence de devoir calculer les valeurs de température en ces différents points en parallèle avec l'activité du circuit sur lequel les tâches sont exécutées. De manière bouclée, la température impose des contraintes sur la puissance qu'il est possible de dissiper en ces différents points du circuit ce qui implique de contrôler l'exécution des tâches (gestion de puissance par DVFS, DPM et/ou migration de tâche) en fonction de l'évolution de la température. Cet aspect dynamique couplé entre température et gestion de puissance nécessite de calculer simultanément l'état des cœurs de processeurs et leur température. Pour calculer l'état actif ou inactif des cœurs d'une architecture matérielle en liaison avec les décisions d'un algorithme d'ordonnancement il apparait inutile de viser l'exécution réelle du code applicatif étant donné les constantes de temps liées aux échanges de chaleur dans le circuit. Aussi, nous nous intéressons aux simulateurs qui ont été développés dans le but d'analyser les résultats d'algorithmes d'ordonnancement de tâches suivant un modèle abstrait de ces tâches. Parmi ces simulateurs on peut citer Cheddar [71], STORM [52] ou encore TimeTools [73].

L'environnement Cheddar, un environnement développé en ADA à l'Université de Brest implémente la plupart des algorithmes d'ordonnancement temps réels classiques. Il contient un moteur de simulation qui fournit des outils de conception d'algorithmes d'ordonnancement et de modèles de tâches spécifiques.

On peut également citer TORSCHE [74] qui est une boite à outils Matlab librement disponible pour le développement d'algorithmes d'ordonnancement développée à l'Université de Prague.

De même, YASA (*Yet Another Scheduling Analyser*) [75] est un environnement développé par l'Université de Rostock, ayant pour but d'accélérer la décision pour la sélection d'algorithmes d'ordonnancement adéquats (principalement monoprocesseur) pour une application temps réel donnée.

Le simulateur TimeTools propose un environnement pour l'analyse de l'ordonnancement d'un ensemble de tâches muni de relations de précédences et d'un modèle d'occurrence des tâches basé sur un automate temporisé. Des techniques de simulation et d'analyse d'ordonnançabilité (basée sur UPPAAL [76]) sont disponibles dans cet environnement.

Il faut également signaler l'existence des outils commerciaux comme RapidRMA et TimeWiz.

Cependant, malgré les apports intéressants proposés, ces outils ne disposent généralement pas d'interface aisée permettant de construire un environnement de co-simulation, par exemple en association avec ATMI.

L'outil STORM « Simulation TOol for Real-time Multiprocessor scheduling » est une plateforme d'évaluation et de simulation d'ordonnancement multiprocesseur dédiée particulièrement aux architectures multiprocesseurs composées de processeurs hétérogènes ou homogènes (le cas monoprocesseur est le cas le plus simple). STORM est conçu pour être capable de prendre en compte différents paramètres des architectures: par exemple le nombre de cœurs dans l'architecture, l'algorithme d'ordonnancement lui-même. Par ailleurs, il est aisé de représenter des ajustements des fréquences de fonctionnement des processeurs car STORM offre un accès simple à de nombreuses informations du simulateur via une API claire. Ainsi, STORM est flexible, portable, aussi il a été choisi pour modéliser les comportements fonctionnels dans notre approche. Par conséquent, un premier travail réalisé a consisté à étudier et développer les mécanismes qui permettent aux deux simulateurs STORM et ATMI de communiquer. Ceci fait l'objet en particulier du chapitre suivant.

2.5 Conclusion

Dans ce chapitre nous avons introduit les systèmes temps réel et un état de l'art sur les travaux qui abordent les aspects thermiques des processeurs. Ces travaux visent généralement à réduire le pic de température à travers des techniques de *load balancing* ou d'ajustement de vitesses des processeurs. L'ensemble de ces travaux considéret une température ambiante constante, ce qui est loin dêtre le cas ne serait-ce que du fait de l'influence de la chaleur dissipée par le système en fonctionnement sur la température ambiante à l'intérieur de l'espace clos contenant le système électronique.

Dans le chapitre suivant nous décrivons la modélisation thermique du circuit utilisé (le MPC5517 de la société Freescale) ainsi que l'environnement de co-simulation basé sur le couplage entre ATMI et STORM.

CHAPITRE 3. Modélisation thermique

3.1 Introduction

L'objectif global est de déterminer une technique de réduction des cycles thermiques subits par les unités matérielles de l'architecture, c'est-à-dire au niveau silicium, et ce afin de limiter l'influence négative de la température sur la fiabilité du circuit. Aussi, si on se limite à proposer des techniques d'optimisation pour réguler la température au niveau du boitier externe alors il n'est pas assuré qu'au niveau du silicium l'effet désiré de réduction des cycles thermiques soit effectivement obtenu. Ainsi, il est important de considérer la variation de la température au niveau du circuit (*die*) pour évaluer l'impact réel des optimisations envisagées sur les cycles thermiques et in fine sur la fiabilité. Ceci impose de développer un modèle thermique du système prenant en compte les différents niveaux de matériaux, du silicium au boitier englobant le circuit. Une fois ce modèle déterminé, il doit être intégré dans un environnement permettant d'évaluer à partir des comportements fonctionnels du système les évolutions de température du circuit.

3.2 Environnement pour l'analyse thermique

Notre objectif est de permettre la mise en œuvre d'un environnement de simulation qui relie l'aspect fonctionnel d'une application (représenté par un jeu de tâches qui s'exécutent sur un processeur ou des processeurs selon un algorithme d'ordonnancement donné) et l'aspect dynamique de la température lié à cet aspect fonctionnel.

Comme mentionné dans le chapitre 2, nous avons choisi de modéliser les comportements fonctionnels par le simulateur STORM, et les comportements en température par le simulateur ATMI [65]. Ces deux simulateurs nécessitent un ensemble d'informations et de paramètres afin qu'ils soient représentatifs d'un système réel. Pour y arriver nous devons renseigner dans STORM les caractéristiques des tâches, la description abstraite de l'architecture et définir l'algorithme d'ordonnancement utilisé. Concernant ATMI, il s'agit de renseigner l'architecture matérielle cible support de la partie applicative (aspect fonctionnel) et pour cela il est nécessaires de :

- Disposer d'une description abstraite du placement des unités (*layout*) de l'architecture matérielle. En effet, connaitre la position relative des unités qui composent

l'architecture (cœurs de processeurs, mémoires caches, interconnexion par exemple) permet de faciliter l'identification des sources de chaleurs lorsque les tâches s'exécutent sur ces unités. Dans le cas où le placement des unités sur le circuit n'est pas fourni par le constructeur, cette identification peut être éventuellement effectuée à l'aide d'une caméra thermique placée au-dessus du circuit sur lequel sont exécutés des programmes spécifiques permettant d'activer plus particulièrement les unités à identifier.

- Connaitre les caractéristiques thermiques (capacités, résistances..) du circuit contenant l'architecture matérielle pour dériver un modèle thermique du système. Si toutes les caractéristiques ne sont pas fournies par le constructeur, disposer d'au moins un sous-ensemble significatif permettrait d'obtenir ce modèle en complétant les données constructeur par des expérimentations.

La Figure 3.1 décrit l'approche suivie afin d'arriver à modéliser et à concevoir une plateforme de simulation valide (c'est-à-dire représentative des aspects fonctionnels, des aspects température et de leur couplage). Dans cette approche, une caméra thermique est utilisée pour identifier les différentes unités ou blocs (*layout*) du processeur cible et les renseigner dans ATMI avec les caractéristiques thermiques associées. D'autres paramètres thermiques peuvent être identifiés à l'aide de STiMuL [77] en comparant la température simulée avec celle observée par la caméra. Ces paramètres, une fois validés, sont reportés dans ATMI qui couplé avec le simulateur fonctionnel STORM, constituent l'environnement de simulation.

Toutes ces démarches et étapes seront détaillées par la suite en s'appuyant sur l'étude d'un circuit comportant deux cœurs : le MPC5517 de la société Freescale.

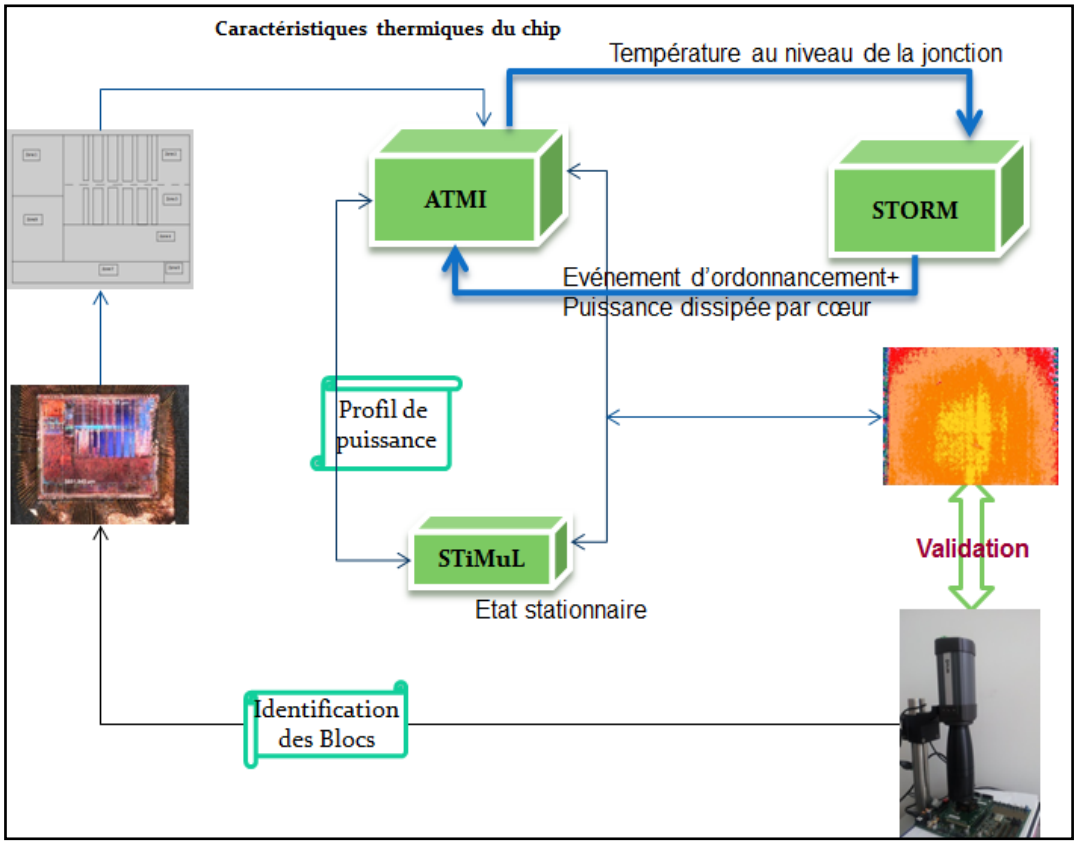


Figure 3.1 : Flot de modélisation retenu

3.2.1 Architecture étudiée : le MPC5517

Le circuit qui a servi à notre étude est le MPC5517 de la famille MPC55xx des microcontrôleurs 32 bits. Ce composant est développé par la société Freescale pour réaliser des contrôleurs intégrés dans le domaine de l'automobile. En effet, les processeurs basés sur l'architecture PowerPC sont très populaires dans l'automobile et Freescale est un partenaire historique des industriels de l'automobile en général.

Plusieurs solutions multicœurs sont disponibles sur cette architecture (MPC551x, MPC5668G/E), mais la plupart sont asymétriques : il y a un cœur principal pour les calculs et un cœur plus simple (e200z0 par exemple) pour les opérations d'entrées/sorties, réseau, etc.

Le seul processeur double cœurs symétrique présent au moment de notre étude dans la feuille de route commerciale de Freescale est le MPC5643L, basé sur deux cœurs de type e200z4. Cependant le MPC5643L annoncé par Freescale fin 2010 n'a été disponible commercialement que fin 2012.

Aussi nous avons considéré le circuit MPC5517 également bi-cœurs mais l'inconvénient de ce circuit vis-à-vis de notre étude est son asymétrie qui ne permet de faire fonctionner les deux cœurs sous un ordonnancement global en particulier du fait de l'absence de mémoire partagée rendant impossible la migration efficace de tâches.

Le MPC5517 comporte deux cœurs de type PowerPC : e200z1 et e200z0 qui peuvent fonctionner sous 4 fréquences différentes : 16 Mhz, 48 Mhz, 66 Mhz et 80Mhz.

La société Freescale fournit une carte d'évaluation et de développement (MPC55xx EVB Kit) pouvant accueillir ce circuit. Le MPC5517 encapsulé dans un boitier de type LQFP144 en plastique y est monté sur un support et peut être remplacé à volonté. Cette disposition facilite la mise en place d'un circuit décapsulé permettant l'emploi de la caméra thermique. Les outils de développement de Freescale (CodeWarrior) sont disponibles pour cette carte d'évaluation.

3.2.2 Caméra thermique

Nous nous sommes équipés d'une caméra thermique de type FLIR SC325 (Figure 3.2) avec un support et un objectif microscope (résolution 25µm) adaptés permettant d'acquérir des images de taille 320x240 pixels à la fréquence maximale de 60Hz.

Les analyses menées avec cette caméra thermique ont été faite sur le composant MPC5517 encapsulé ainsi que sur le même composant décapsulé. Ces analyses sont détaillées dans le paragraphe 3.3 Modélisation et Validation.

Figure 3.2 : Caméra thermique SC325

3.2.3 Le simulateur ATMI

ATMI, un acronyme de « **A**nalytical model of **T**emperature in **MI**croprocessors », est développé par l'IRISA comme un ensemble de fonctions implémentées en C. Il permet de calculer la température au niveau de la couche la plus basse (le silicium) d'un circuit électronique. Pour ce faire, ATMI prend comme entrée :

- une description de l'architecture matérielle d'un circuit avec ses paramètres divisés en deux catégories : des paramètres dimensionnels, et des paramètres thermiques (conductance, conductivité ...) ;
- la puissance dissipée par chaque unité ou bloc (*layout*) de cette architecture au cours du temps.

La description de l'architecture matérielle, c'est-à-dire l'emplacement des différentes unités et leurs dimensions, a été faite par une analyse thermographique sur le composant Freescale.

Pour compléter cette description structurelle d'un modèle sous ATMI, il est nécessaire d'identifier les valeurs de neuf paramètres reliés à cette architecture, afin de compléter le modèle thermique de l'architecture. Certains de ces paramètres sont déterminés à la fois par une analyse des *datasheet* fournis par Freescale et par une analyse effectuée avec la caméra thermique. Les autres paramètres ont été identifiés à l'aide du simulateur STIMuL.

L'autre entrée de ATMI, à savoir la puissance dissipé par chaque unité, est évaluée à partir des images prisent par la caméra et également à partir de l'analyse des *datasheet* du composant. On en déduit alors un modèle de puissance dissipée par les unités de l'architecture, modèle qui est ensuite intégré dans STORM.

Une fois tous les modèles paramétrés, les valeurs de températures instantanées retournées par ATMI vers STORM sont celles au niveau jonction (i.e. silicium) et pour chaque unité renseignée.

3.2.4 Le simulateur STiMuL

Ce simulateur, développé également par l'IRISA, permet de calculer la température en régime permanent dans des circuits multicouches. Contrairement à ATMI, STiMuL est destiné principalement à la modélisation de la conduction de chaleur en régime permanent. Il peut être utilisé pour modéliser la température dans les circuits 2D et 3D à partir de densités de

puissance fournies par l'utilisateur. Il peut aussi être utilisé pour obtenir la densité de puissance à partir de la température.

Dans notre approche de modélisation, STiMUL, est utilisé afin de :

- Déterminer les valeurs de certains paramètres pour les reporter ensuite dans ATMI.
- Valider indirectement ces valeurs de paramètres par des mesures prises par la caméra thermique, pour ainsi valider le modèle développé dans ATMI.

3.2.5 Le simulateur STORM

La phase d'ordonnancement des tâches fonctionnelles est réalisée sous STORM « Simulation TOol for Real time Multiprocessor scheduling » qui est un outil de simulation développé par l'IRCCyN et destiné à l'analyse du comportement et l'évaluation de performances de politiques d'ordonnancement. Ainsi, STORM permet de décrire à la fois l'algorithme d'ordonnancement considéré, la partie applicative (les jeux de tâches) et une description gros grain de l'architecture matérielle représentée par les cœurs de processeurs.

Afin de caractériser en puissance les cœurs de processeur de l'architecture, nous introduisons un modèle de puissance dans STORM. Ainsi, l'objectif est de transmettre à ATMI à chaque évènement d'ordonnancement, la puissance dissipée par chaque cœur, en fonction de l'activité des cœurs déduite de l'ordonnancement.

3.3 Modélisation & Validation

3.3.1 Modèle de puissance

Le modèle intégré dans STORM est un modèle classique où la puissance totale dissipée est la somme sur l'ensemble des unités de l'architecture de la puissance dynamique (dépendant directement de l'activité du circuit) et de la puissance statique (induite par les courants de fuite à l'intérieur du circuit) de chaque unité :

$$P_{total} = \sum_{i} P_{dynamique} + \sum_{i} P_{statique}$$

Au niveau STORM, une unité est un cœur de processeur. La puissance dynamique dissipée dépend des changements d'états des différents transistors de l'unité :

$$P_{dynamique} = C * V^2 * F$$

Où C représente la capacité de commutation équivalente qui est chargée ou déchargée à chaque cycle d'horloge, V est la tension d'alimentation et F la fréquence de fonctionnement du circuit. On peut considérer que la puissance dynamique est indépendante de la température [78]. Dans notre approche nous considérons que tous les cœurs de l'architecture fonctionnent à la même fréquence. Ceci est guidé par le fait que les circuits étudiés fonctionnent suivant ce principe. Cependant adapter le modèle pour tenir compte de fréquences différentes par cœur ne poserait pas de difficulté.

La puissance statique dépend principalement du courant de fuite $I_{leackage}$:

$$P_{statique} = I_{leackage} * V$$

Où $I_{leackage}$ est le courant de fuite global pour une unité.

Pour expliciter la puissance statique dissipée on reprend la formulation décrite dans [79] et [80]. Le principal courant de fuite à considérer est le courant I_{sub} dit courant de *subthreshold* :

$$I_{sub} = \mu \cdot C_{ox} \cdot V_T^2 \cdot \frac{W}{L} \cdot e^{\left(\frac{V_{GS} - V_{th}}{n \cdot V_T}\right)}$$

V_{th} est la tension de seuil, V_T = KT/q est la tension thermique, T la température, V_{GS} la tension de gille, les autres paramètres étant spécifiques à la technologie utilisée et à la conception réalisée. Le courant de fuite intervient quand le transistor est bloqué c'est-à-dire quand V_{GS} = 0. En considérant I_0= I_{sub} à une température de référence T_0 (par exemple la température ambiante 25°), on peut exprimer le rapport I_{sub}/ I_0:

$$\frac{I_{sub}}{I_0} = \frac{T^2}{T_0^2} \cdot \frac{e^{\alpha/T}}{e^{\alpha/T_0}}$$

Où $\alpha = (V_{GS} - V_{th})/n$. Cette expression peut se mettre sous la forme :

$$I_{sub} = I_0 * \frac{T^2}{T_0{}^2} * e^{\alpha * \frac{T_0 - T}{T_0 * T}}$$

On obtient alors l'expression de la puissance totale dissipée :

$$P_{total} = C * V^2 * F + I_0 * \frac{T^2}{T_0{}^2} * e^{\alpha * \frac{T_0 - T}{T_0 * T}} * V$$

Ainsi, connaissant le courant de fuite I_0 du circuit à la température ambiant T_0, on peut en déduire le courant de fuite à une température $T > T_0$.

On remarque que le courant de fuite varie quadratiquement avec la température. Ainsi lorsque la température augmente le courant de fuite augmente également ce qui génère une puissance statique plus élevée qui à son tour fait augmenter la température. Si on arrive à un niveau où le flux thermique produit par le circuit est supérieur à celui extrait par le système de refroidissement alors ce phénomène bouclé conduit à un emballement thermique avec pour conséquence la destruction du circuit [30].

Par conséquent, pour des températures proches de la température ambiante, l'influence de ce courant de fuite sur la température est négligeable; par contre lorsque la température augmente il faut prendre en compte la variation de ce courant de fuite dans la puissance dissipée.

Afin d'intégrer notre modèle de puissance, trois paramètres sont à identifier : C, I_0 et α.

Dans la suite nous montrons à travers les analyses comment ces paramètres ont été déterminés ainsi que les valeurs des autres paramètres reliés à l'architecture et nécessaires pour développer le modèle sous ATMI.

3.3.2 Analyse thermographie sur MPC5517E

La thermographie infrarouge est une technique qui permet de mesurer sans contact la température et ses variations temporelles et spatiales sur la surface d'un objet. Lorsqu'un objet est sollicité thermiquement il émet un rayonnement infrarouge plus au moins intense selon sa température.

Le spectre de rayonnement thermique s'étend de 0,4 à 30 μm mais les moyens d'analyses infrarouges opèrent généralement dans la bande 3 à 15 μm.

Les relevés de thermographie permettent d'obtenir, au moyen d'une caméra sensible aux rayonnements infrarouges, une image mettant en évidence les variations de la température à la surface de l'objet ainsi que la puissance dissipée au niveau de cette surface.

Nos objectifs par cette analyse thermique sont :

- Identification des blocs les plus importants dans le circuit (plus précisément les cœurs de processeur) afin de déterminer les paramètres dimensionnels nécessaires à ATMI.
- Evaluation de la puissance dissipée par chacun de ces blocs.

Ainsi nous avons développé plusieurs programmes de test basique permettant d'activer différentes unités d'un des deux cœurs (le e200z1 et/ou e200z0) afin d'identifier si des valeurs différentes de puissance consommée sont observables. Ces programmes sont constitués de quelques instructions (par exemple des additions et multiplications) qui opèrent sur les registres du processeur et exécutés dans une boucle infinie. Ces programmes sont exécutés à différentes fréquences, en activant un seul cœur dans un premier temps et en activant le deuxième cœur dans deuxième temps.

Cette analyse est menée sur le circuit MPC5517E en deux étapes : circuit dans son boitier (encapsulé, Figure 3.3) et circuit décapsulé (Figure 3.4).

3.3.2.1 Analyse avec circuit encapsulé

Sur l'image du circuit fournie par la caméra, nous enregistrons l'évolution de la température au cours du temps à partir d'une grille de 9x9 capteurs de température définie sur la surface, comme indiqué sur la Figure 3.3 (droite).

Ces mesures sont prises en exécutant les différents programmes considérés avec les quatre fréquences admises (16, 40, 64, 80 MHz) par le MPC5517. La température varie selon la fréquence utilisée, elle atteint 45 degrés sous la fréquence maximale (à partir d'une température ambiante de 25°C) et environ 30 degrés sous la fréquence minimale.

On remarque, sur la Figure 3.3 (gauche), la présence de deux blocs, à droite la température est toujours plus élevée. Par contre à gauche la température reste relativement faible.

L'avantage de cette analyse avec le circuit encapsulé est qu'il permet d'exploiter toute les fréquences de fonctionnement admises et ainsi obtenir différents niveaux de puissance dissipée. Cependant, les résultats obtenus ne sont pas satisfaisants car ils ne fournissent pas d'informations pertinentes afin d'identifier l'emplacement des unités importantes à la surface du circuit. Cela est dû à la dissipation de chaleur au niveau du plastique (qui est un bon

conducteur de chaleur), conduisant à une distribution de température quasiment homogène sur la surface, avec un écart entre la température maximale et la température minimale de 1 degré, ce qui rend difficile la différentiation entre les différentes unités.

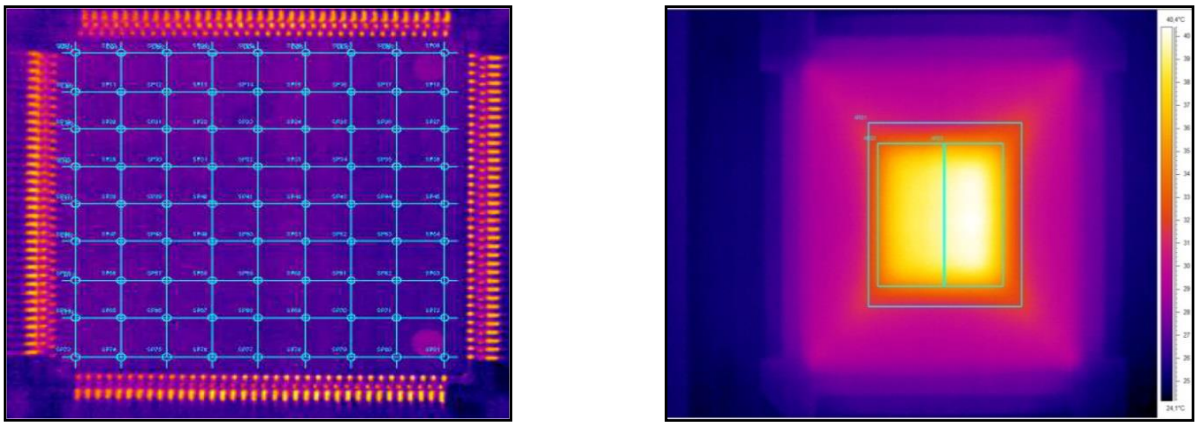


Figure 3.3 : Circuit encapsulé vu par la caméra thermique

3.3.2.2 Analyse avec circuit décapsulé

Outre l'objectif d'identifier les blocs et unités à la surface du circuit, et comme expliqué dans la section 3.2, il est également important de déduire un modèle thermique au niveau silicium afin d'évaluer les variations de température à ce niveau afin de s'assurer que les effets de réduction des cycles thermiques sont obtenus. Ces deux objectifs nous ont conduit à décapsuler quelques composants MPC5517E afin d'enlever la partie plastique et de disposer le circuit directement sous la camera comme le montre la Figure 3.4.

Une fois décapsulé, on peut mesurer dans un premier temps les dimensions du circuit soit 5,8mm par 7,1mm.

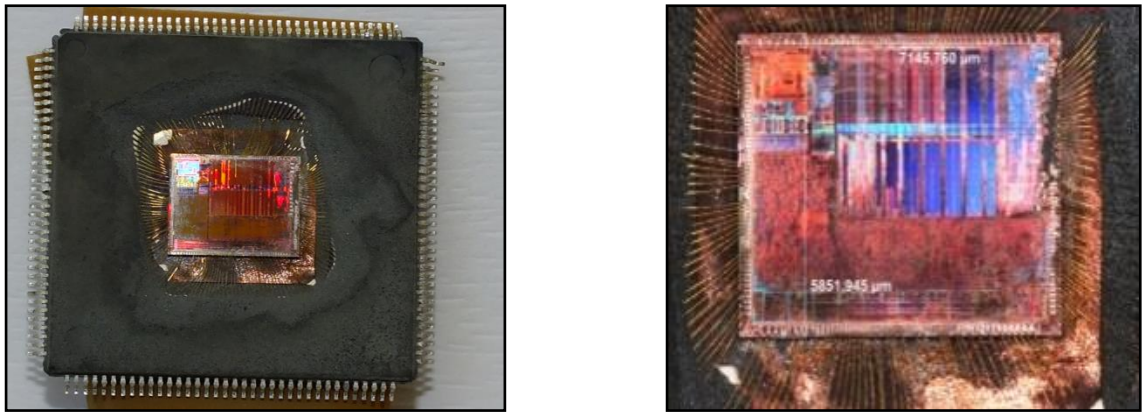


Figure 3.4 : Circuit MPC5517 décapsulé

Il est à noter qu'une fois décapsulé, le composant est directement en contact avec l'air qui possède une conductance thermique beaucoup plus faible que celle du plastique. Aussi, il n'est plus possible de faire fonctionner le circuit à sa fréquence maximale de 80MHz sans produire un échauffement important du circuit qui conduit rapidement (quelques secondes) à une destruction du composant par emballement thermique. Aussi, toutes les analyses réalisées avec la caméra thermique ont été faites à la fréquence minimale du circuit à savoir 16MHz.

Etant donné la taille réduite du composant (environ 35mm²), la faible conductance thermique de l'air qui en contact avec les couches de métallisation, une faible puissance dissipée (45mW à 16MHz), il s'en suit une diffusion rapide de la chaleur à la surface observable du composant (couche métal) ce qui réduit fortement les possibilités d'identification des blocs significatifs dans le circuit.

La Figure 3.5 donne un exemple d'image capturée par la caméra thermique du MPC5517 où le cœur z1 exécute une suite d'additions entre valeurs contenues dans les registres du processeur et le cœur z0 est simplement sous tension mais n'exécute aucun code applicatif. Il y a deux parties plus chaudes dans le circuit (de couleur jaune dans la Figure 3.5), celle au centre doit correspondre au cœur z1 et la tâche plus petite en bas de l'image au centre doit correspondre au périphérique JTAG. En changeant le code exécuté par le cœur z1 (par exemple une suite de lectures/écritures en mémoire) il n'apparaît pas de différences significatives dans la puissance dissipée observée ce qui limite les possibilités d'identifier de façon plus fine d'autres blocs dans le circuit.

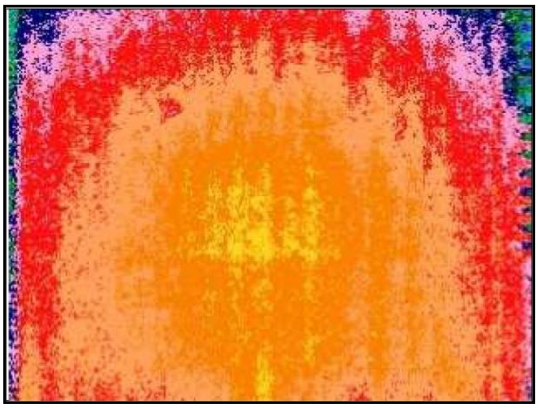


Figure 3.5 : Image thermique du MPC5517 avec le cœur z1 actif (suite d'additions entre registres) et le cœur z0 sous tension.

Toutefois par l'intermédiaire de ces observations et en tenant compte de la topologie du circuit observée sur la Figure 3.4, il est ainsi possible de retracer le « *layout* » du circuit afin d'identifier la position approximative des cœurs sur le circuit (le cœur z0 centré dans la zone 3, et le cœur z1 centré dans la zone 2) et leurs dimensions approchées comme le montre la Figure 3.6

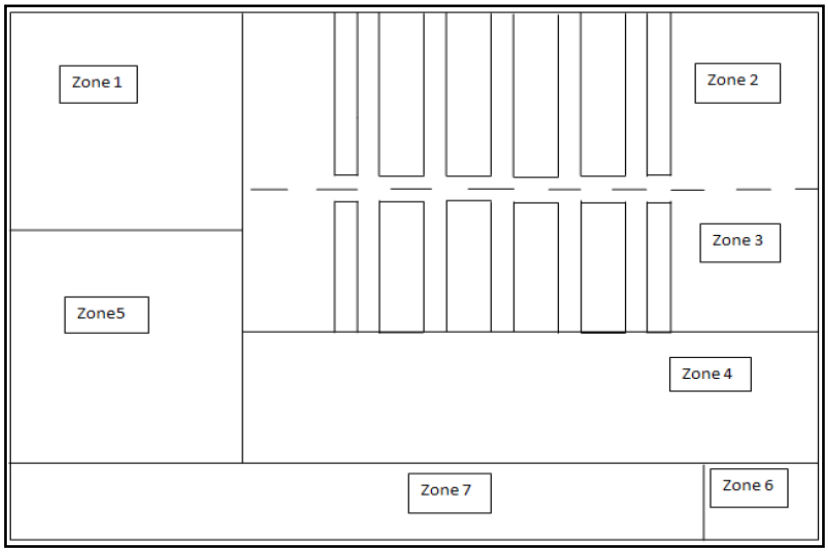


Figure 3.6 : *Layout* du circuit MPC5517

Nous considérons aussi, pour notre étude en simulation, une version étendue à 4 cœurs du circuit MPC5517, illustrée par la Figure 3.7 où nous prenons en compte des caractéristiques et des tailles des composants multipliées par un facteur 2.

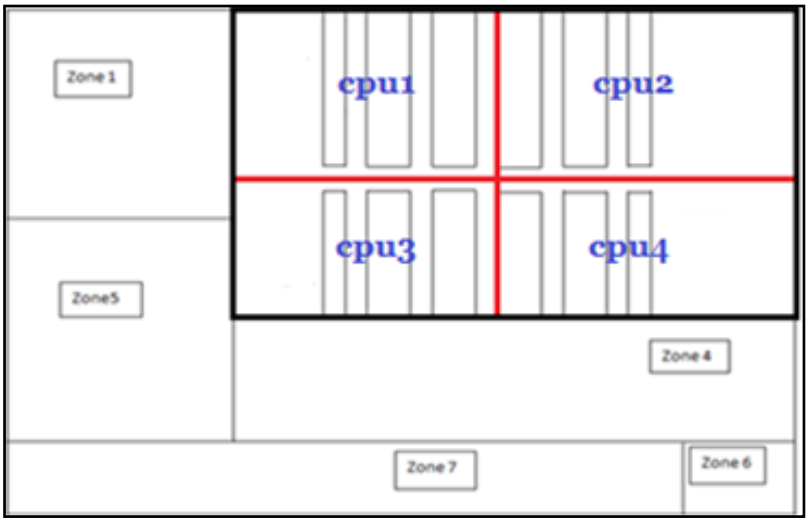


Figure 3.7 : *Layout* du circuit MPC5517 extrapolé à 4 cœurs

3.3.3 Analyse des données constructeurs (*datasheet*)

Cette analyse a pour but de déterminer les paramètres présentés dans le modèle de puissance, et introduits dans STORM par la suite. Ces paramètres sont la capacité de commutation équivalente C, le courant de fuite référence I_0 et la constante reliée à l'architecture α.

Les *datasheets* de Frescale relatifs à la famille MPC551x [81] donnent des informations utiles permettant de calculer le courant de fuite à différentes températures. Ainsi, les spécifications en courant fournies dans ce document indiquent d'une part des courants consommés à 25°C et 70°C pour une même configuration et d'autre part, l'évolution du courant consommé à 25°C et à 70°C pour quatre fréquences de fonctionnement (16MHz, 48MHz, 66MHz et la fréquence maximale autorisée 80MHz).

Cette évolution est présentée dans les deux figures Figure 3.8 et Figure 3.9 sous les températures 25° et 70° respectivement.

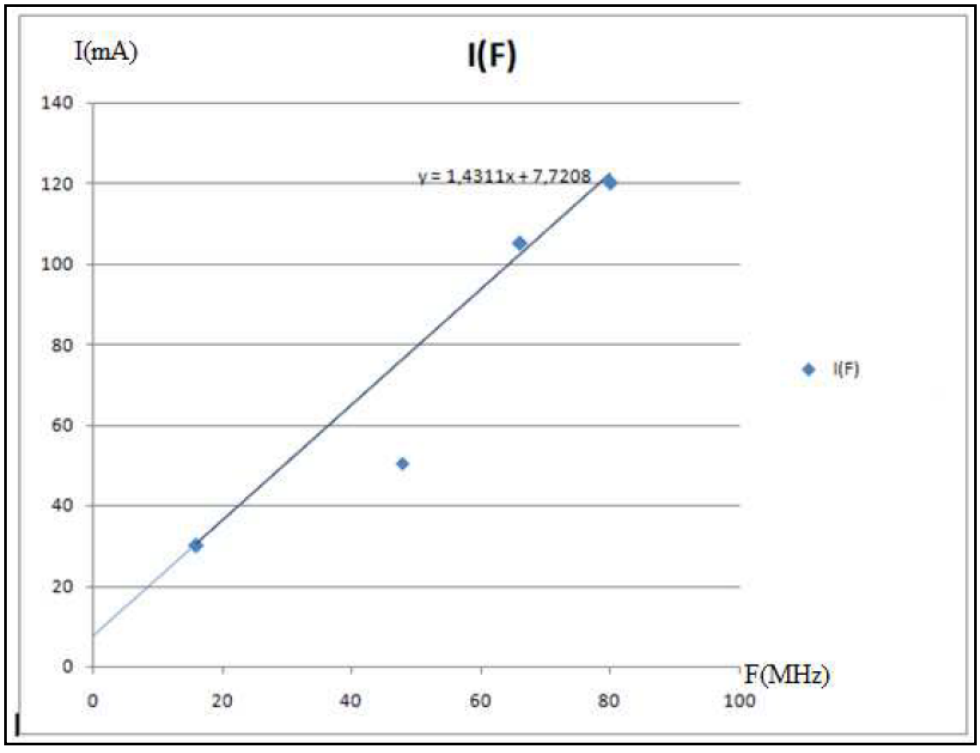


Figure 3.8 : Courbe de courant en fonction des fréquences à température ambiante 25°C

La puissance dynamique évolue linéairement en fonction de la fréquence. Ainsi à partir de ces deux figures de courant en fonction de la fréquence, on peut identifier la valeur du courant de fuite du circuit : cette valeur est déterminée en considérant une fréquence nulle (aucune activité dans le circuit). Ainsi on trouve la valeur du courant de fuite I_0 à 25°C égale à 10 mA environ. De même, on en déduit la valeur du courant de fuite I_{sub} à 70°C qui est d'environ 22mA. A partir de ces deu-x valeurs on peut déterminer la valeur du coefficient α intervenant dans la formule de I_{sub} en fonction de I_0 soit α=35,6. Connaissant le courant de fuite et le courant total (120mA à 80MHz et à 25°C lorsque les deux cœurs sont actifs, sous une tension d'alimentation V_{dd} = 1,5V) on peut déterminer la valeur de la capacité de commutation équivalente C, soit C = $9.17*10^{-10}$.

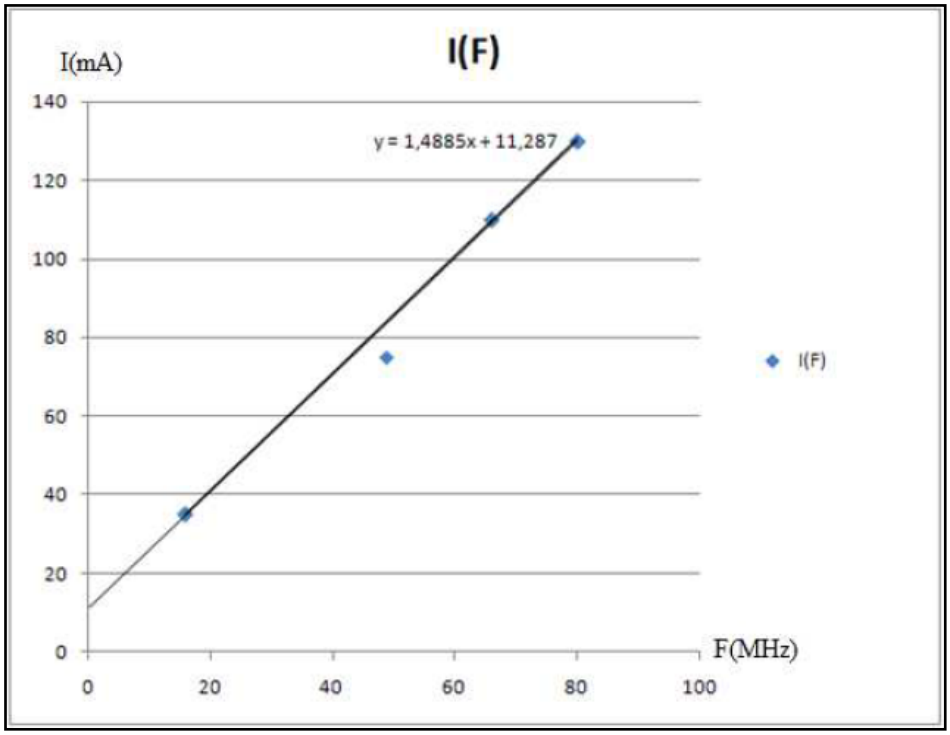


Figure 3.9 : Courbe de courant en fonction des fréquences à température ambiante 70°

3.3.4 Modélisation sous ATMI

Comme indiqué ci-dessus, nous avons fait le choix d'utiliser l'environnement ATMI pour modéliser les comportements thermiques du circuit. La Figure 3.10 décrit le modèle de circuit défini dans ATMI.

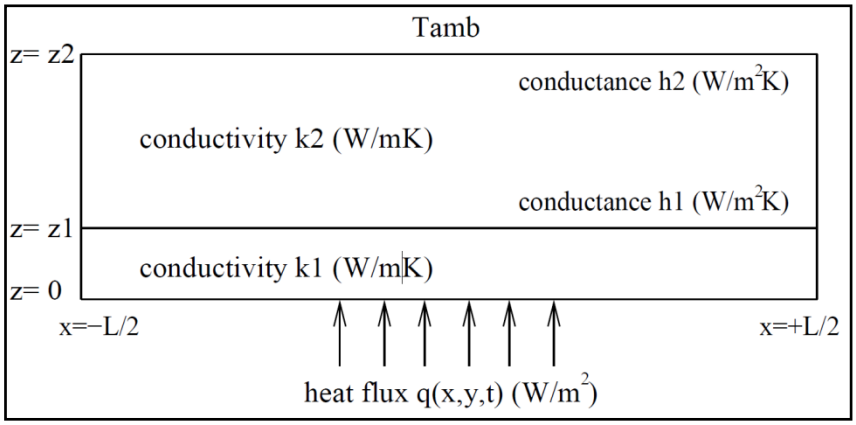


Figure 3.10 : Principe de la modélisation d'un circuit par ATMI

Pour obtenir un modèle du circuit, l'environnement ATMI impose de définir un ensemble de paramètres présentés dans le Tableau 3.1.

Ce modèle considère deux couches de matériaux chacune représentant un carré de taille L. La première de taille z1 représente le silicium et la seconde de taille z2-z1 représente le système de dissipation de chaleur et/ou la base du système de dissipation de chaleur (par exemple du cuivre), ce qui est notre cas. Ces deux couches sont caractérisées chacune par leur conductivité thermique k1 et k2. Le flux de chaleur produit à l'instant t par les transistors et les connexions situées en (x,y) dans la couche basse est représenté par q(x,y,t). La conductance h1 représente l'interface entre les deux matériaux et la conductance h2 décrit le coefficient de transfert de chaleur, ce dernier coefficient modélise le flux qu'il est possible d'extraire à travers la surface du matériau supérieur vers l'air ambiant. Cette description simplifiée du système néglige en particulier les échanges thermiques qui s'opèrent sur les côtés verticaux des deux couches de matériaux.

Les sources de chaleur ont été modélisées dans ATMI par des rectangles situés en z=0. Avec ce type de description des sources de chaleur, ATMI impose de définir ces rectangles ayant des côtés parallèles aux axes x et y du circuit. Il est nécessaire de définir également la taille et la position des rectangles dans le plan z=0 et la densité de puissance (en W/m²) produite à partir de l'instant t=0 par chacun de ces rectangles.

La description de ces rectangles dans ATMI est basée sur l'analyse thermographique faite précédemment (Figure 3.6).

Compte tenu des analyses effectuées précédemment, les premiers paramètres du modèle ATMI sont :

- la taille L = 6mm (le circuit a en réalité une taille de 5x7mm).
- Les épaisseurs des matériaux sont z1 = 0,3mm et d = 0,4mm.
- Les surfaces des deux cœurs de processeurs sont estimées chacune à 1,8mmx3mm.
- Les valeurs de conductivité k1 et k2 pour les matériaux de niveaux 1 (silicium) et 2 (plastique) données par Freescale sont indiquées dans [82]. Elles sont respectivement égales à 148 W/mK et 360 W/mK.
- La diffusivité thermique d'un matériau est égale à k/ρ*C où k est sa conductivité, ρ sa masse volumique et C sa capacité thermique. Ainsi, à partir des données fournies dans [82], on peut déduire aisément les deux valeurs a1= 87.10^{-6} et a2 = 106.10^{-6}.

Tableau 3.1 : Paramètres dans ATMI

Paramètre	Description
z1	épaisseur du matériau de niveau 1
d2	épaisseur du matériau de niveau 2 (z2 = z1 + d)
k1	conductivité thermique du matériau de niveau 1 (en W/mK)
a1	diffusivité thermique du matériau de niveau 1 (en m^2 / s)
k2	conductivité thermique du matériau de niveau 2 (en W/mK)
a2	diffusivité thermique du matériau de niveau 2 (en m^2 / s)
h1	conductance thermique de l'interface niveau 1/niveau 2 (en W/m^2K)
h2	conductance thermique de l'interface niveau 2/ambiant (W/m^2K)
L	taille du circuit (en m)

Pour construire le modèle complet, il est nécessaire de déterminer les valeurs des autres paramètres en particulier h1 et h2 qui ont un rôle majeur sur la diffusion de la chaleur dans le circuit et entre le circuit et l'air ambiant.

Pour déterminer maintenant la puissance dissipée au niveau du circuit, ATMI dispose d'une technique pour extraire la puissance rayonnée à partir d'une image thermique donnée. En effet, en utilisant la caméra thermique, son capteur mesure la puissance du rayonnement thermique d'un objet dans un certain spectre. Cette mesure est ensuite convertie en valeurs de température puis visualisée par l'intermédiaire d'une palette numérique pour l'affichage. Ainsi, la caméra thermique est d'abord un capteur de puissance rayonnée, puissance qui dépend elle-même de la température de l'objet. Par conséquent, si la transformation effectuée

Ainsi, sur la Figure 3.11 (a), à partir de l'image fournie par la caméra via la palette « Rainbow » il est possible de recalculer en inversant la fonction palette, la valeur de la puissance de rayonnement mesurée pour chaque pixel de l'image source. La Figure 3.11 (b)

donne le résultat non pas pour chaque pixel mais par blocs carrés de 10x10 pixels. Cette matrice de valeurs de puissance est en relation directe avec la puissance dissipée par le composant. En sommant les puissances ainsi trouvées pour l'ensemble des pixels, on doit retrouver la puissance mesurée sur l'alimentation du circuit. N'ayant pas directement accès à cette mesure on a considéré que cette puissance est égale à celle indiquée dans les *datasheets* du composant à savoir 180 mW à T=25°C. Ainsi par cette méthode il est possible de déterminer pour chaque partie du circuit sa contribution à la consommation de puissance. Pour vérifier ce calcul, nous avons effectué une mesure de température par la caméra thermique en utilisant la même palette que l'outil Gnuplot (Figure 3.11 (c)). Ainsi il y a concordance entre la visualisation de la température fournie par la caméra thermique à partir de la puissance mesurée par le capteur de la caméra et l'affichage de la puissance recalculée et affichée via Gnuplot. Ainsi cette méthode de mesure de puissance est utile pour renseigner la consommation des cœurs de processeurs, ici du MPC5517, dans les étapes suivantes de simulation.

Dans la suite on indique l'approche qui a permis de déterminer les valeurs des paramètres h1= 300W/m^{2}K; h2 = 190 W/m^{2}K, étant données leurs sensibilités et leurs influences dans les résultats –leurs valeurs doit être évaluée le mieux possible. Cette approche se base sur l'utilisation du simulateur STIMuL.

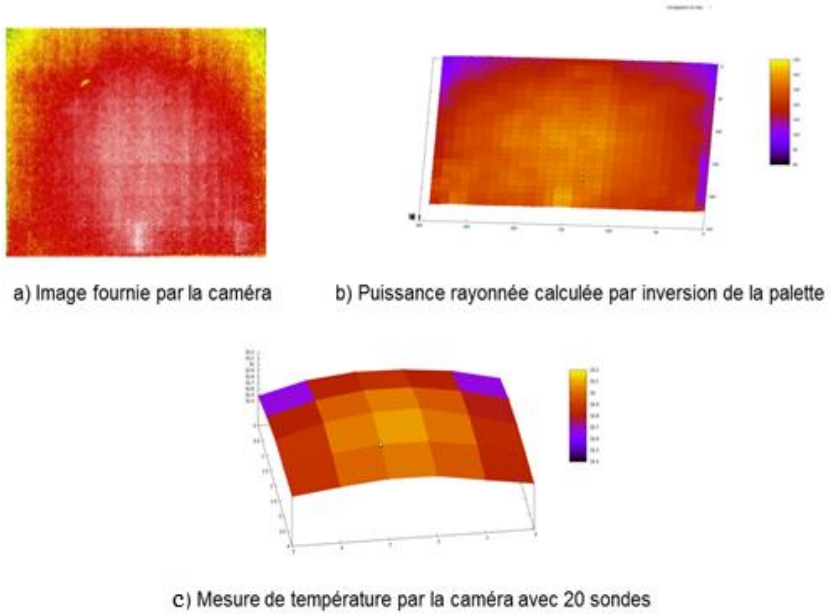


Figure 3.11 : Calcul de la puissance rayonnée à partir de l'image fournie par la caméra thermique

3.3.5 Modélisation sous STIMUL

Parmi les paramètres dans ATMI qui restent à ajuster, figurent les conductances thermiques h_i relatives aux interfaces. Obtenir les valeurs de ces conductances n'est pas aisé alors qu'elles sont nécessaires à ATMI pour calculer la température au niveau jonction.

Avec la caméra thermique nous pouvons observer la température du circuit dans deux niveaux : au niveau du plastique (dans le cas où le circuit est encapsulé) et au niveau du cuivre (*die*) (dans le cas où le chip est décapsulé).

L'approche suivie pour déterminer les valeurs des conductances est d'opérer par comparaison entre les mesures par caméra à ces deux niveaux et les températures obtenues par simulation à l'état d'équilibre à ces deux niveaux. Pour ce faire, on utilise le simulateur STiMuL qui utilise les mêmes principes et paramètres de simulation que ATMI mais permet d'évaluer la température (à l'état d'équilibre) et les flux thermiques aux interfaces entre couches.

L'environnement STiMuL permet une description d'un circuit suivant un système de couches empilées elles-mêmes de type multi-niveaux (Figure 3.12). Chaque couche peut avoir une taille différente mais les niveaux à l'intérieur d'une couche ont tous la même taille. Les couches sont supposées centrées verticalement (axe z). Il est fait l'hypothèse que la couche de niveau 1 (sa surface la plus basse, aussi appelée le côté 0 de la couche multi-niveaux, est en z=0) est de taille plus petite que celle de la couche de niveau 2 et ainsi de suite. La résolution du modèle consiste à calculer la température sur chaque côté 0 de chacune des couches de telle sorte que la température calculée sur le côté 0 de la couche n sert de conditions aux limites sur le côté 1 (opposé au côté 0) de la couche n-1. Ainsi par un processus itératif la solution converge vers un état d'équilibre global en température. Ainsi STiMuL permet de calculer la température à l'état d'équilibre et d'estimer la température à chaque interface entre les couches du circuit modélisé.

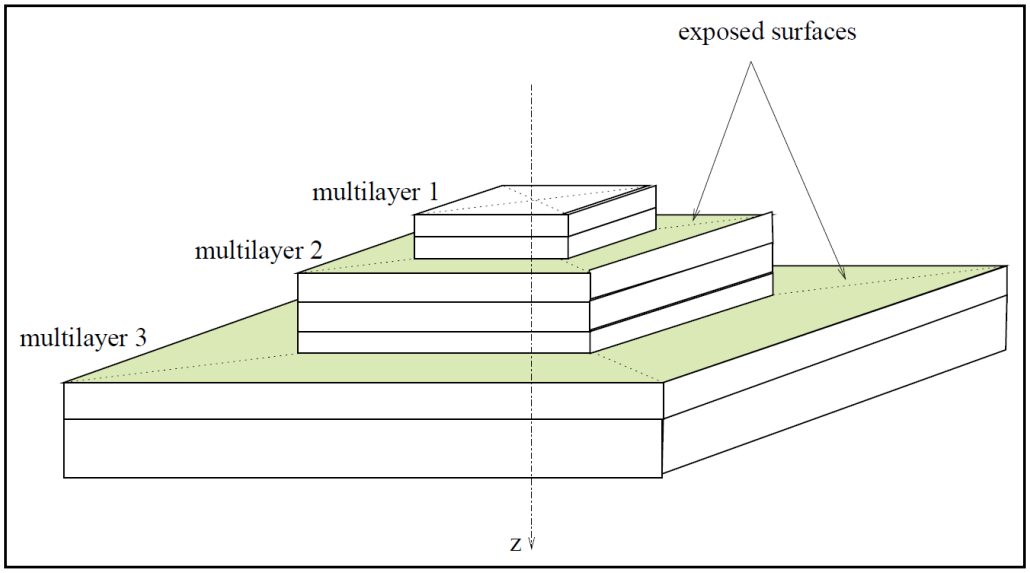


Figure 3.12 : Principe de modélisation d'un circuit avec STIMuL.

Le point intéressant de STiMuL et de ATMI est que ces deux environnements partagent majoritairement les mêmes paramètres physiques pour décrire un circuit. Ainsi, si les paramètres sont validés par une modélisation avec STiMuL, alors ces valeurs de paramètres peuvent être reportées dans ATMI pour une étude de la dynamique en température. C'est l'approche qui a été choisie afin de déterminer les valeurs de h1 et h2.

Pour obtenir un modèle ATMI en deux couches du type de celui de la Figure 3.10 où h2 représente le flux qu'il est possible d'extraire à travers la surface du boitier plastique vers l'air ambiant, on cherche à décrire dans STiMuL le circuit suivant un modèle en trois couches : silicium, métal, plastique.

Dans un premier temps nous avons décrit dans STiMuL le circuit sous la forme de deux couches, l'une de silicium et l'autre de métal (Figure 3.13). Ce modèle simple correspond au circuit décapsulé considéré ci-dessus avec ATMI. Nous connaissons la température à la surface supérieure (mesurée par la caméra thermique) et la puissance dissipée au niveau silicium (méthode déterminée par ATMI). La valeur de h1 est grande car les couches silicium et métal sont intimement liées. Les valeurs de k1 et k2 sont données par la documentation Freescale et les dimensions du circuit sont connues. La valeur de h2 peut être estimée de manière empirique pour une surface au contact de l'air [83] : $h=K*(\Delta T/l)^{0.25}$ où K est une constante de valeur 1.35 (pour une surface horizontale avec dissipation de chaleur verticale), ΔT est la différence entre la température à la surface (celle donnée par la caméra thermique) et la température ambiante mesurée à une distance de l au-dessus de cette surface d'échange. Ainsi ce calcul donne une valeur de h2 égale à environ 20 W/m^2K.

Toutes ces valeurs étant déterminées, nous les avons introduites dans le modèle développé sous STiMuL.

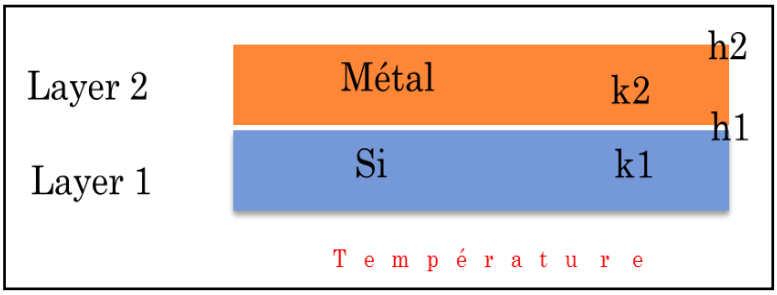


Figure 3.13 : Premier modèle en deux couches sous STiMuL correspondant au chip décapsulé.

En simulation nous avons également cherché à faire varier le paramètre h2 et la valeur h2=50 W/m^2K est effectivement la plus pertinente pour se rapprocher du comportement observé avec la caméra thermique. Le calcul de la température par STiMul à la surface du circuit décapsulé est décrit sur la Figure 3.14 (seul de cœur z1 est actif, puissance dissipée du cœur cadencé à 16MHz, le cœur z0 est sous tension mais n'exécute pas de code).

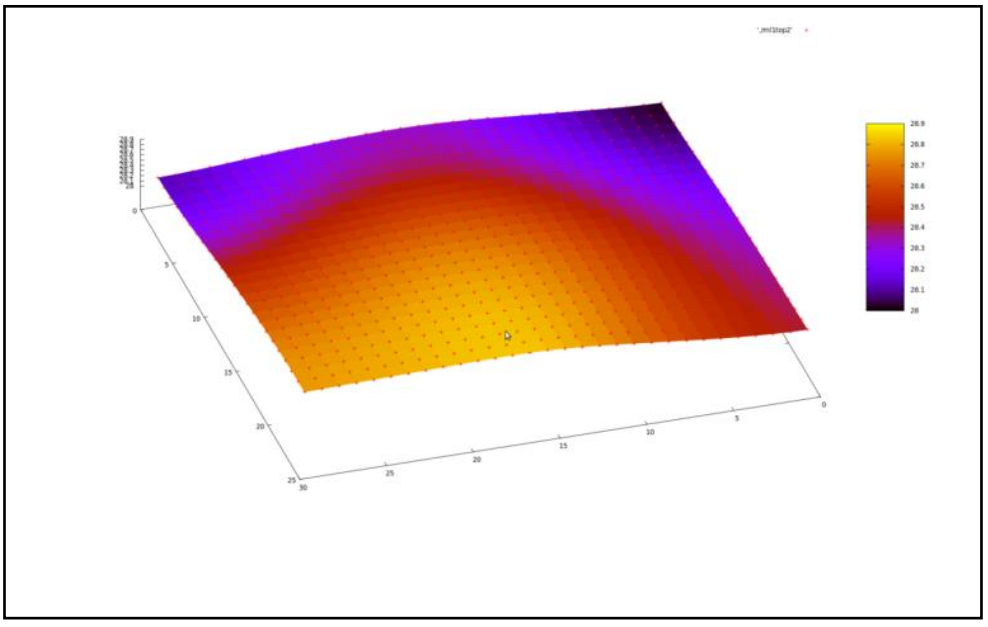


Figure 3.14 : Résultat de STiMul sur le modèle du MPC5517E décapsulé.

Ainsi on a pu déterminer la valeur de h2 permettant de calculer la température au niveau supérieur de la couche métal du modèle STIMuL correspondant à la température mesurée par la caméra thermique au niveau métal du circuit décapsulé. Parmi les analyses faites sur le circuit décapsulé, nous avons fait l'expérience qu'un circuit MPC5517E décapsulé

fonctionnant à 80MHz (au lieu de 16MHz) est hors d'usage en quelques secondes de fonctionnement après avoir été mis sous tension et activé sur un code effectuant une suite de multiplications. Ceci est dû au fait que dans ce cas la valeur de h2 est faible car le *die* est en contact direct avec l'air qui est un mauvais conducteur de la chaleur. Lorsque le circuit est dans son boitier plastique la valeur de h2 est plus élevée, évitant ainsi l'emballement thermique lorsque le circuit est placé dans les conditions d'utilisation prévues par le constructeur. Nous avons changé dans le modèle de simulation décrit ci-dessus la valeur de la puissance dissipée par le cœur de processeur fonctionnant à 80MHz. Par simulation on obtient également un emballement thermique du circuit (Figure 3.15) qui conduit rapidement à une température supérieure à 145°C correspondant à la température jonction maximum de fonctionnement du circuit autorisée d'après le constructeur.

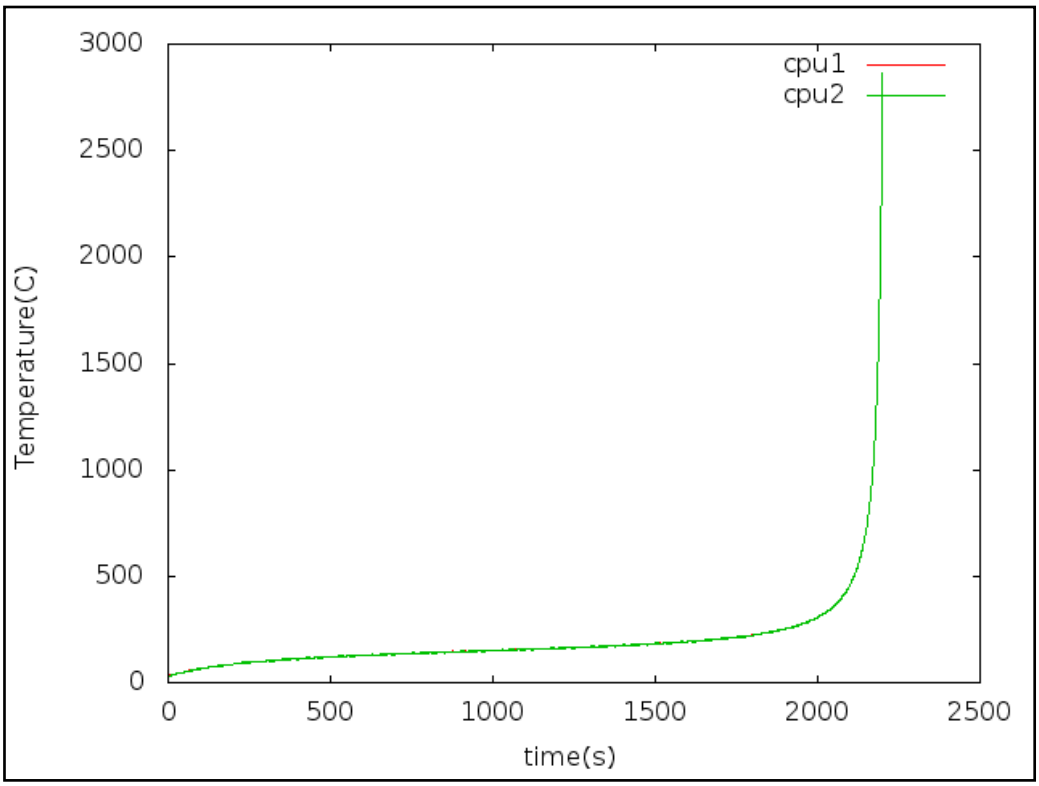


Figure 3.15 : Simulation par ATMI de l'emballement thermique observé sur un MPC5517E

L'étude ci-dessus a permis de vérifier que notre approche de modélisation permet de reproduire à un niveau de granularité adapté le comportement thermique du composant. Cependant le modèle du circuit décapsulé n'est pas celui qui convient à notre étude car dans la pratique on utilise le circuit dans son boitier, avec la capacité de fonctionner normalement à toutes des fréquences données dans les *datasheets*. Aussi nous avons modifié le modèle sous

STiMuL pour passer à un modèle de simulation en trois couches : silicium, métal et plastique (Figure 3.16).

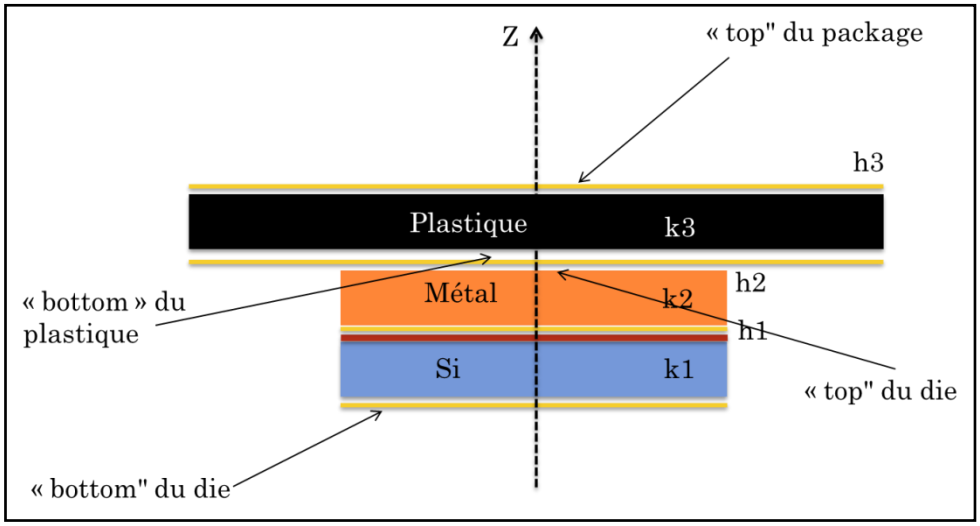


Figure 3.16 : Modèle en trois couches du circuit sous STiMuL.

Avec cette modélisation sous STiMuL il est ainsi possible de calculer la température à l'état d'équilibre en différents points du circuit, en particulier au niveau jonction (*bottom du die*) et au niveau supérieur du boitier (*top du package*).

La valeur de la conductance h2 entre métal et plastique est relativement élevée car le plastique adhère à la surface du *die*. Ceci est confirmé par la valeur du paramètre de caractérisation thermique du boitier 144LQFP utilisé par Freescale pour le MPC5517. En effet, ce paramètre caractérise la différence de température entre la surface supérieure du boitier et la température de jonction, sa valeur donnée dans les *datasheets* du circuit est de 2°C/W. Par contre la conductance thermique h3 de l'interface plastique/air est relativement faible. En utilisant la même approche empirique que celle décrite ci-dessus ($h=K*(\Delta T/l)^{0.25}$) à partir de la température mesurée à la surface du boitier par rapport à celle de l'air ambiant situé au-dessus du boitier, on obtient une valeur de h3=190 W/m^2K. Sur la base de ce modèle en trois couches, on obtient par STiMuL le résultat (cœur z1 fonctionnant à 80MHz, cœur z0 inactif mais sous tension) présenté sur Figure 3.17, image de gauche. A droite de la figure est donnée l'image capturée par la caméra du circuit dans son boitier.

Nous remarquons que : i) les températures fournies par le modèle sont très proches de celle mesurées concernant la partie centrale du boitier (au-dessus du *die*) et ii), le modèle néglige les fils de connexion en cuivre qui relient les *pads* du circuit avec les broches en périphérie du boitier. Or ces fils diffusent de la chaleur, phénomène qui n'est pas représenté dans notre modèle ce qui explique la différence observée sur la partie du boitier entourant le *die*.

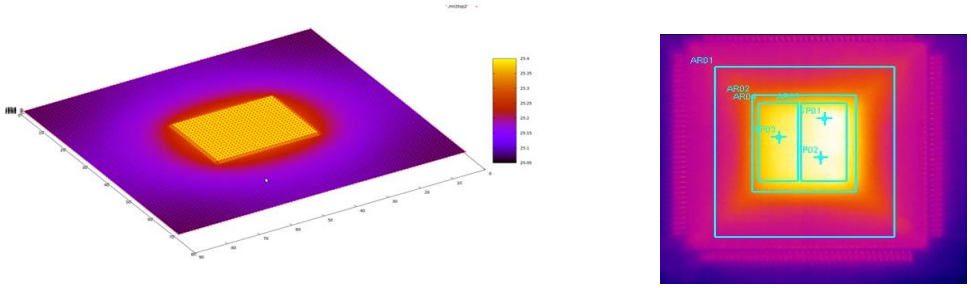


Figure 3.17 : Simulation sous STiMuL (à gauche) et image issue de la caméra thermique (à droite) du MPC5517

Ainsi, par cette étude en simulation par STiMuL nous avons identifié un modèle dont les paramètres sont reportés dans ATMI. La valeur de h2 dans le modèle ATMI (en deux couches comme décrit par la Figure 3.13) est de 190 W/m^{2}K, toutes les valeurs des autres paramètres sont données dans la section 3.3.4.

Pour l'extrapolation du circuit à 4 cœurs, la valeur de h_2 est fixée à 300 W/m^{2}K pour tenir compte de l'accroissement de taille du circuit. Une fois ce modèle de simulation réalisé et validé, les actions menées ensuite ont consisté à connecter les deux simulateurs STORM et ATMI dans le but d'obtenir par simulation les comportements thermiques de l'architecture en fonction de l'activité des cœurs de processeurs de l'architecture induite par les exécutions des tâches suivant l'algorithme d'ordonnancement choisi.

3.4 Contrôleur thermique

Comme expliqué précédemment et afin d'évaluer la dynamique de température du système, nous avons couplé les deux simulateurs STORM (aspects fonctionnels) et ATMI (dynamique de la température). En effet, on cherche à évaluer les évolutions de la température des cœurs en fonction des variations de leurs activités déterminées par les décisions d'ordonnancement des tâches. Le couplage entre STORM et ATMI servira ensuite de support pour intégrer l'approche proposée de réduction des cycles thermiques, et représentée par le Contrôleur Thermique dans la Figure 3.18.

La puissance dissipée à un instant donné par un cœur de processeur dépend :

- Du jeu de tâches et de leurs caractéristiques,

- Des choix de la politique d'ordonnancement (en particulier du nombre de tâches dans l'état *running* à un instant donné)
- De la stratégie d'affectation des tâches dans l'état *running* aux cœurs de processeur, autrement dit, quel cœur choisir pour exécuter cette tâche.

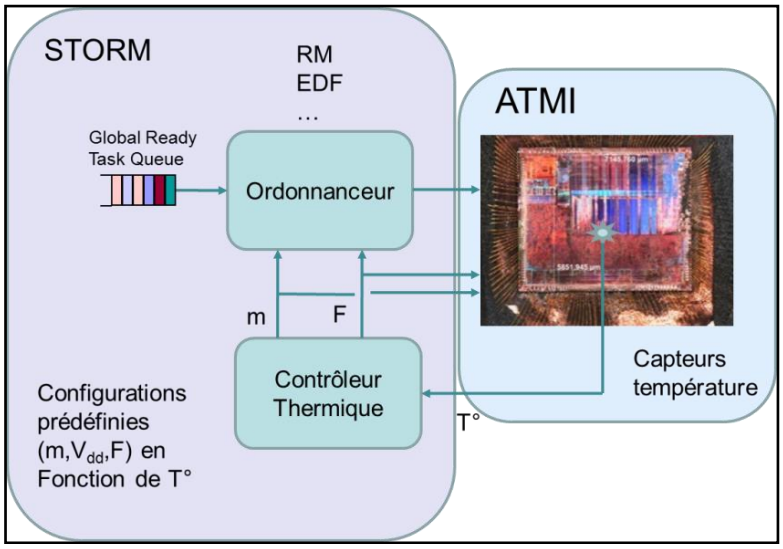


Figure 3.18 : Contrôleur thermique construit par couplage de STORM et ATMI.

Dans STORM sont décrites les architectures matérielle (nombre et caractéristiques des cœurs) et logicielle (jeu de tâches avec leurs propriétés), ainsi que la technique d'ordonnancement.

Côté ATMI le modèle thermique présenté dans la partie 3.3.4 est renseigné.

La connexion entre les deux simulateurs permet à STORM de transmettre à ATMI à chaque événement d'ordonnancement significatif les informations suivantes :

- la date absolue de cet événement,
- pour chaque cœur de processeur la puissance dissipée à partir de cette date,
- quels cœurs de processeurs sont actifs ou inactifs (mode basse consommation de type *sleep*). Cette information est utilisée par la stratégie proposée de réduction des cycles thermiques.

Un événement d'ordonnancement significatif est un événement d'ordonnancement avec changement d'état. Un changement d'état intervient lorsqu'au moins une des situations suivantes est rencontrée :

- Une nouvelle tâche s'exécute.
- Une tâche termine son exécution.
- Un changement de contexte suite à une préemption.
- Une migration de tâche.
- Un changement de fréquence des processeurs.
- Un cœur de processeur change d'état (de *sleep* à *running* ou vice versa)

Dans ATMI on a la possibilité de définir la valeur de la température ambiante dans laquelle se situe le circuit et plus précisément les variations de cette température ambiante. L'intérêt de faire varier cette température est qu'il est ainsi possible de décrire un scénario d'utilisation du circuit dans un environnement où les amplitudes thermiques peuvent être importantes (e.g. dans un véhicule automobile) dans un intervalle de temps limité comme cela a été présenté dans le chapitre 2.

ATMI calcule les valeurs de températures de chaque cœur qui sont ensuite retournées à STORM où est implémentée la stratégie de réduction des cycles thermiques.

Ainsi ATMI joue le rôle d'un capteur ou de capteurs de température.

Il faut noter que la température est déterminée par ATMI avec un décalage d'un événement, comme cela est explicité sur la Figure 3.19. En effet, ATMI calcule la variation de température entre deux instants t_i et t_{i+1} qui correspondent à deux événements d'ordonnancement successifs définis par STORM. A l'instant t_i STORM communique à ATMI la nouvelle valeur de la puissance dissipée $P_{k,ti}$ par chaque cœur de l'architecture. On considère que ces valeurs de puissance restent constantes jusqu'au prochain événement d'ordonnancement t_{i+1}. Cependant à l'instant t_i ATMI ne peut calculer l'évolution de la température après cette date t_i tant que la date t_{i+1} n'est elle-même pas connue, instant auquel les puissances dissipées par les cœurs sont recalculées par STORM. Ainsi ATMI est toujours en retard sur STORM d'un événement ce qui provoque un biais dans les décisions prises par l'algorithme d'ordonnancement ou par le contrôleur thermique.

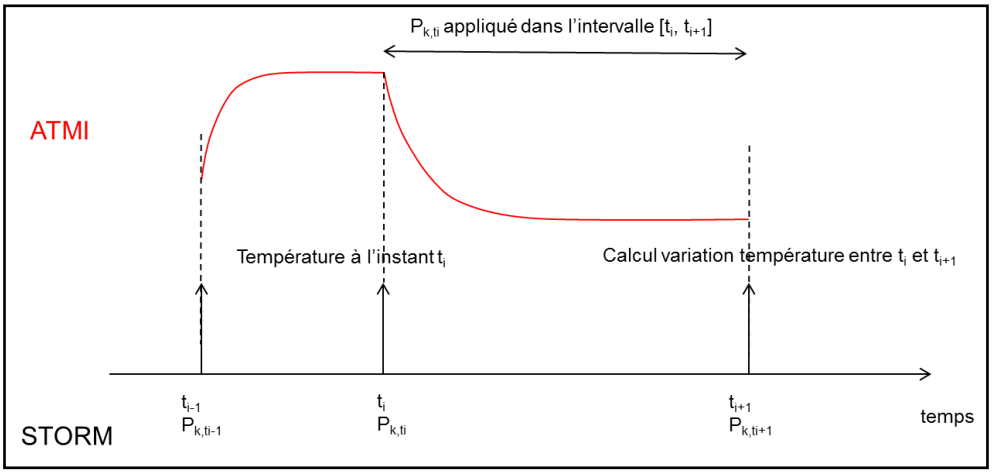


Figure 3.19 : Principe du calcul de la température par ATMI à partir des informations fournies par STORM

Il faut noter toutefois qu'un biais existe effectivement dans la pratique du fait de la constante thermique du circuit. Ainsi la température mesurée à un instant donné dans le circuit réel dépend de l'écart Δt entre cet instant de mesure et la date t_i du ou des derniers changements de puissance dissipée par les différents composants du circuit. Par conséquent, en effectuant l'hypothèse que l'écart entre deux événements d'ordonnancement qui provoquent un changement de puissance dissipée par les cœurs est relativement faible par rapport à la constante thermique du circuit, le décalage du calcul de la température par ATMI n'a que peu d'impact sur le contrôle de la température et des cycles thermiques fait dans STORM.

D'un point de vue pratique, la connexion entre STORM (codé en Java) et ATMI (codé en C) est réalisée avec la classe « ProcessBuilder » (apparue depuis Java 1.5) implémentée dans la technique d'ordonnancement.

En effet, l'échange des données entre ATMI et STORM est assuré à travers un fichier partagé, créé avec le déclenchement du premier événement d'ordonnancement par STORM afin de sauvegarder les informations nécessaires pour ATMI (date absolue, puissance dissipée de chaque cœur).

Côté ATMI on a défini le modèle thermique et les règles pour extraire les données à partir du fichier partagé.

Côté STORM, à chaque événement d'ordonnancement correspond à une écriture d'une ligne de données dans le fichier partagé, puis à l'activation de ATMI avec les méthodes de la classe « ProcessBuilder ». Cette interface maintient la connexion entre ATMI et STORM et permet

de récupérer les valeurs de température associées aux différents cœurs et la date présente à travers un « BufferReader » qu'on l'initialise sur un « InputStream ».

Du point de vue du temps de simulation, ATMI dispose d'une fonction qui permet de créer un fichier spécifique (initialisé dans le premier lancement de simulation) contenant la description, les paramètres et l'état du modèle thermique à l'équilibre avec les conditions initiales pour la configuration choisie de telle sorte qu'à la prochaine simulation il est inutile de refaire tous les calculs nécessaires à l'initialisation du modèle thermique (qui correspond à la même configuration). Ainsi un gain significatif sur le temps de simulation est obtenu lorsque plusieurs simulations sont réalisées utilisant les mêmes conditions initiales.

3.5 Conclusion

Dans ce chapitre nous avons décrit une nouvelle approche pour la modélisation thermique de circuit et à titre d'exemple pour le MPC5517 (bi-cœurs). Nous avons aussi considéré une version étendue (dans notre étude) à 4 cœurs pour permettre d'étudier une stratégie de réduction des cycles thermiques sur une architecture plus complexe. Cette modélisation a été faite en utilisant à la fois les données mesurées sur le circuit par une caméra thermique, les données constructeur disponibles et les outils de simulation thermique ATMI et StiMuL.

Un environnement de co-simulation a été défini et développé (ATMI-STORM) pour mettre en évidence les comportements thermiques associés aux comportements fonctionnels.

CHAPITRE 4. Approche de réduction des cycles thermiques

4.1 Introduction

Dans ce chapitre nous présentons l'approche proposée de réduction des cycles thermiques d'une architecture multi-cœurs placée dans un environnement à température ambiante variable.

Cette technique proposée nécessite que l'architecture matérielle supporte les possibilités de DPM (Dynamic Power Management) et de DVFS (Dynamic Voltage and Frequency Scaling) afin d'agir sur les puissances dissipées par chaque unité de l'architecture pendant l'exécution des tâches. Le contrôle du DPM et du DVFS est assuré par les algorithmes présentés dans ce chapitre.

4.2 Principe de réduction des cycles thermiques

Le principe de la méthode proposée est d'adapter dynamiquement les paramètres de l'architecture matérielle en fonction de la température instantanée des cœurs qui est la conséquence de la température ambiante et de la température due à l'activité de fonctionnement de chaque cœur.

Cette adaptation consiste à ajuster le couple de fonctionnement (V, F) pour : i), réduire le nombre total de cycles thermiques, ii) limiter la température maximale des cœurs et iii), vérifier les échéances des tâches.

Pour illustrer ces points, prenons un exemple d'une architecture monoprocesseur qui exécute 3 tâches tel que présenté dans la Figure 4.1. Les 3 tâches s'exécutent à une fréquence et une tension données qui conduisent à des intervalles de temps d'inactivité (mode *idle*) entre chaque activation de tâche. La réponse en température suite à cette exécution est illustrée par la courbe en rouge sur la même figure où on remarque la création de 3 cycles thermiques dus aux passages du cœur d'un état *run*, où il exécute une tâche (ou un ensemble d'instructions) à fréquence élevée, à un état *idle* (par exemple horloge coupée et tension d'alimentation réduite) puis à nouveau à un état *run*.

Plus le cœur reste longtemps en état *idle* (respectivement en état run), après un état *run* (respectivement en état *idle*), plus la descente (respectivement la montée) en température est importante ce qui augmente l'amplitude du cycle thermique.

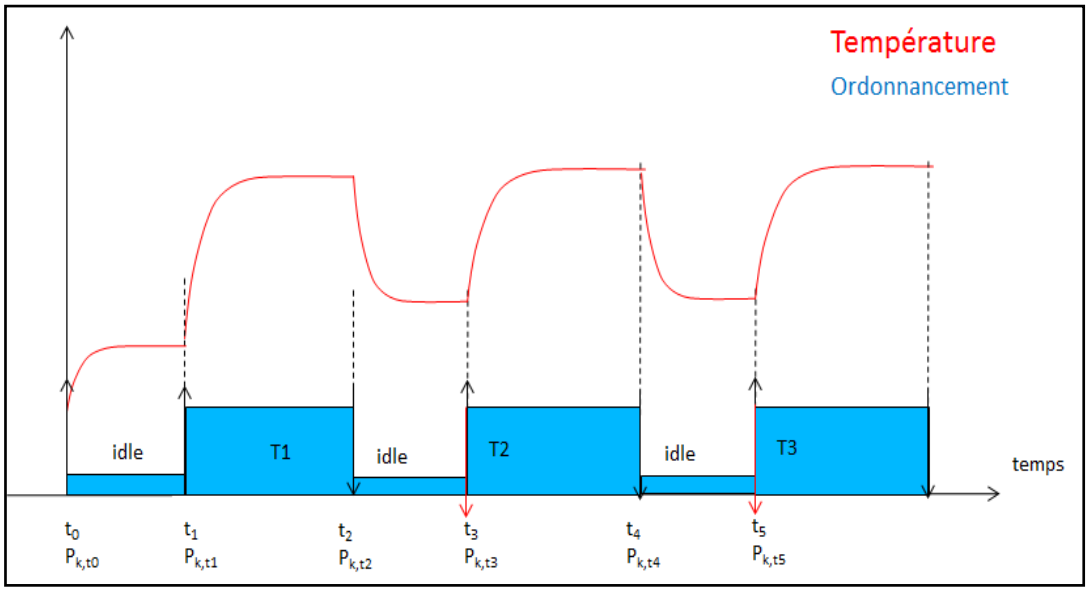


Figure 4.1 : Ordonnancement de 3 tâches sur un seul cœur à fréquence élevée

Maintenant, si on prend le même exemple et on adapte la fréquence et la tension à leurs valeurs minimale de telle sorte que l'augmentation de la durée d'exécution des tâches ne conduise pas à une violation de leurs échéances, comme le montre la Figure 4.2, alors les 3 cycles thermiques observés précédemment sont éliminés. Ceci, est dû à la suppression des passages successifs entre modes *run* et *sleep*.

Puisqu'on s'intéresse à une architecture multi-cœurs alors l'adaptation du couple (V, F), peut conduire à une évolution dans le temps du nombre de cœurs *running* dans cette architecture. On considère ici le cas où la charge des tâches est plus faible que le total des puissances de calcul cumulées sur l'ensemble des cœurs de l'architecture fonctionnant à fréquence maximum autorisée.

En effet, si on augmente la fréquence et la tension de l'ensemble des cœurs, la charge induite par les tâches est réduite en proportion de la variation de fréquence ce qui implique moins de ressources de calcul pour satisfaire les échéances et ainsi peut permettre de diminuer le nombre de cœurs *running* nécessaires.

La diminution du nombre des cœurs *running*, i.e mettre certains cœurs en mode *sleep*, a pour effet de baisser la puissance statique globale et donc limiter l'effet d'emballement thermique à température élevée.

Vice versa, une diminution de la fréquence et de la tension implique une augmentation de la charge globale des tâches ce qui nécessite a priori d'accroître le nombre de cœurs *running* pour répondre à cette augmentation.

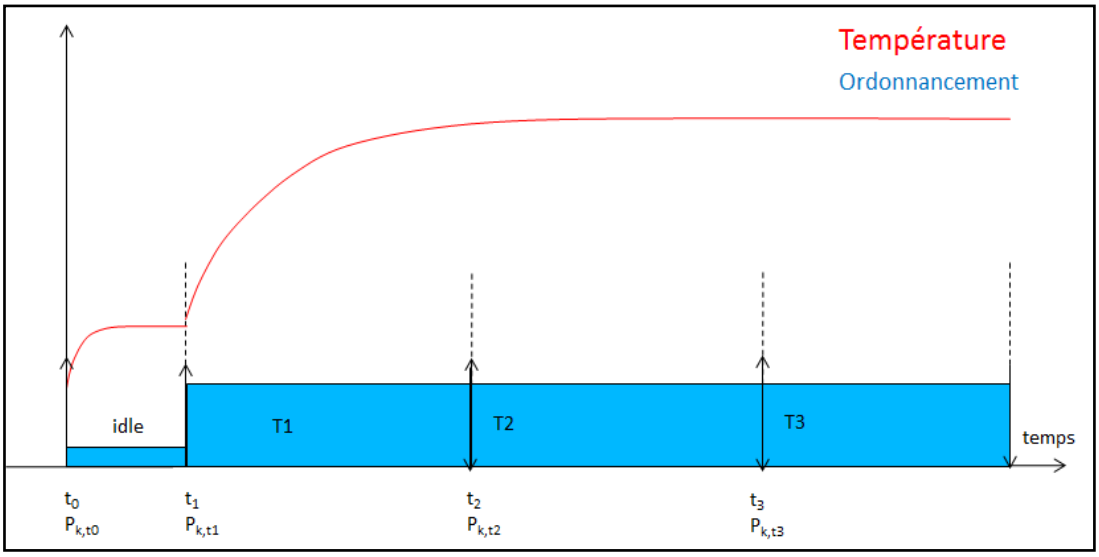


Figure 4.2 : Ordonnancement de 3 tâches sur un seul cœur à fréquence minimal

Pour se situer maintenant par rapport à la variation de la température ambiante, et en se basant sur les observations précédentes, deux cas se présentent :

- **Température ambiante élevée :** dans ce cas la puissance statique est importante (puisqu'elle dépend exponentiellement de la température, comme expliqué dans 3.3.1), ce qui peut provoquer un emballement thermique [30]. Alors il est nécessaire de diminuer les sources de puissance (le nombre de cœurs *running*), c'est-à-dire mettre le maximum de cœurs en état *sleep*. Cette diminution du nombre de ressources nécessite une augmentation de la fréquence de fonctionnement afin d'assurer la fin d'exécution des tâches avant leurs échéances, au risque de créer des cycles thermiques.
- **Température ambiante faible :** dans ce cas il est préférable d'augmenter le nombre de cœurs *running* dans l'architecture ce qui permet de diminuer la fréquence de fonctionnement des cœurs en s'assurant que les échéances des tâches sont respectées. Ainsi la puissance dynamique diminue ce qui conduit à réduire à la fois l'amplitude

des cycles thermiques et leur nombre du fait de la réduction du nombre de passages *run* /*idle* des cœurs. Dans ce cas il n'y a pas de risque d'emballement thermique puisqu'on se situe à température ambiante faible.

Ainsi trois états sont associés aux différents cœurs de l'architecture, à partir du modèle de puissance décrit dans 3.3.1:

- *Sleep* : dans cet état l'horloge du cœur est coupée de même que la tension d'alimentation sur l'essentiel de la logique du cœur et du ou des caches. La consommation de puissance est alors négligeable, on la suppose nulle.
- *idle* : le cœur est en état d'attente d'exécution d'instructions aussi, l'horloge peut être coupée sur une partie importante du cœur. Dans ce cas il consomme seulement une puissance statique.
- *run* : le cœur est totalement actif et exécute un ensemble d'instructions, alors il consomme une puissance statique et une puissance dynamique.

4.3 Heuristique globale de réduction des cycles thermiques

En se basant sur les observations décrites précédemment, notre approche de réduction des cycles thermiques se compose de deux étapes :

- **Une étape dite hors ligne :** cette étape consiste à établir un ensemble de configurations {T, m_T, F_T} tel que m_T représente le nombre de cœurs *running* à la température instantanée T, et F_T est la fréquence de fonctionnement des m_T cœurs.
- **Une étape dite en ligne :** cette étape correspond à l'exécution de la technique proposée de sélection dans cet ensemble de la configuration qui correspond à la température mesurée par le ou les capteurs situés dans le circuit (et modélisés par ATMI dans notre cas).

4.3.1 Détermination de l'ensemble {T,m_T,F_T}

Les valeurs de m_T et F_T de chaque configuration doivent en premier lieu être déterminées pour que l'ensemble des tâches soit ordonnançable suivant leurs échéances. Ainsi il n'apparaît pas judicieux de déterminer en ligne (pendant l'exécution) le choix des couples

m_T et F_T étant donné la complexité de déterminer ces valeurs qui vérifient l'ordonnançabilité des tâches. Aussi le calcul hors ligne de ces configurations est le plus adapté. Pour ce faire, on considère dans un premier temps une approche du calcul de m_T et F_T qui garantit l'ordonnançabilité des tâches. Ensuite on propose une technique qui s'appuie sur la simulation pour trouver ces valeurs de m_T et F_T., sans garantie de faisabilité de l'ordonnancement dans ce cas.

4.3.1.1 Approche basée critère d'ordonnançabilité

Ce problème d'ordonnançabilité d'un ensemble de tâches sur une architecture multiprocesseurs où les processeurs ont une fréquence ajustable a été par exemple étudié en profondeur dans [84] . Nous reprenons dans un premier temps l'approche développée dans cette thèse en considérant un système placé dans un environnement où la température ambiante est basse (e.g. T = 25°C). Notons que le principe d'ajuster globalement la fréquence des processeurs d'une architecture multicoeur tout en vérifiant l'ordonnançabilité des tâches a été par exemple étudié dans [85]. Cependant ce type de technique s'appuie sur un ordonnancement fluide qui en pratique n'est que peu exploitable dans la réalisation de systèmes temps réel.

L'algorithme proposé dans [84] cherche à déterminer la vitesse de fonctionnement des processeurs la plus basse possible telle que pour cette vitesse de fonctionnement l'ensemble des tâches soit ordonnançable sur m processeurs identiques suivant la technique d'ordonnancement considérée. La vitesse d'un processeur est définie par le rapport $\omega = F/F_{max}$ où F est la fréquence effective de fonctionnement des processeurs et F_{max} la fréquence maximum admissible des processeurs.

Etant donné un ensemble de tâches τ, un algorithme d'ordonnancement S, l'Algorithme 4.1 de recherche de la vitesse minimale *find_lowest_speed*(S, τ, m, ω_{max}, ω_{min}) opère comme suit :

Algorithme 4.1 : Algorithme de recherche de la vitesse minimale

```
1.  begin
2.  if (sched(S, τ, m, ω_max) == False)
3.     then return #APP_NOT_SCHEDULABLE;
4.  ω_current = ω_max;
5.  while (ω_current ≥ ω_min and sched(S, τ, m, ω_current)) do
6.        ω_current = ω_current - ω_step ;
7.  end while
8.  ω_current = ω_current + ω_step ;
9.  return ω_current;
10. end
```

La valeur de ω_{step} est aussi petite que possible, une valeur très faible permet d'approcher la vitesse minimale au prix d'un nombre d'itérations a priori élevé. Inversement augmenter cette valeur accélère la convergence mais conduit à une solution qui peut s'écarter d'une distance ω_{step} de la fréquence minimale. La fonction *sched(S, τ, m, ω)* renvoie vrai si l'ensemble de tâches τ est ordonnançable sur m processeurs fonctionnant à la vitesse ω. Tout critère d'ordonnançabilité défini pour S est utilisable afin de garantir la faisabilité de l'ordonnancement. Ainsi dans [84] quatre critères d'ordonnançabilité définis pour l'algorithme global-EDF ([15],[86],[87],[88]) sont considérés : si l'un d'eux au moins est vérifié alors la fonction sched(S, τ, m, ω) retourne la valeur vrai.

L'Algorithme 4.2, ci-dessus, ne tient pas compte de la puissance statique des cœurs de processeurs qui dépend de la température. Son objectif est de réduire uniquement la puissance dynamique. Aussi on considère la technique suivante pour déterminer les valeurs de m_T et F_T pour différents seuils de température pour lesquels la puissance statique devient non négligeable.

Algorithme 4.2 : Détermination des configurations {T, m_T, F_T}

1. **begin**
2. F_{prev} = F_{max} * *find_lowest_speed*(S, τ, m, ω_{max}, ω_{min});
3. m_T = m ; F_T = F_{prev} ;
4. T = T_{min} + ΔT;
5. **while** (T $\leq$ T_{max}) **do**
6. F = F_{max} * *find_lowest_speed*(S, τ, m-1, ω_{max}, ω_{min}) ;
7. **If** (*Ptot*(m-1, F ,T) < *Ptot*(m, F_{prev},T) **then**
    a. m_T = m-1;
    b. F_T = F;
    c. *Push*(T, m_T, F_T);
    d. F_{prev} = F;
    e. m = m-1 ;
    f. end if
8. T = T + ΔT;
9. **end while**
10. **end**

La fonction *Ptot*(m, F, T) retourne la puissance totale (statique et dynamique) dissipée par m cœurs de processeurs fonctionnant à la fréquence F et sous une température T.

Dans cet algorithme on part d'une valeur faible de la température (ambiante), par exemple T_{min} = 25°C pour laquelle on sait que la puissance statique est faible ce qui conduit à utiliser l'ensemble des m cœurs de processeur de l'architecture à fréquence minimale. Ensuite, on fait varier cette température par pas de ΔT. Pour chaque valeur de température ainsi obtenue on calcule (ligne 6) la fréquence de fonctionnement minimale en considérant un nombre de processeur égal à m-1. Si on obtient une diminution de la puissance totale avec m-1 cœurs par rapport à la configuration à m cœurs (ligne 7) alors on retient cette nouvelle configuration (ligne 7.c). Ainsi, à chaque fois que la condition à la ligne 7 est vérifiée cela signifie que la configuration où un cœur supplémentaire est mis en mode repos permet de réduire la puissance dissipée à la température T considérée. L'algorithme fournit donc les couples m_T et F_T pour chaque seuil de température T qui produit une baisse de puissance en réduisant le nombre de processeurs actifs lorsque la température augmente. La fréquence F_T est la plus basse pour le nombre de processeur m_T considéré.

Si cette approche permet de trouver les configurations m_T et F_T qui garantissent le respect des échéances des tâches, il n'en demeure pas moins qu'on risque d'obtenir des taux d'utilisation des processeurs relativement faibles. En effet, les critères d'ordonnançabilité pour les algorithmes d'ordonnancement de type global-EDF imposent des taux d'utilisation des ressources processeurs relativement faibles. Ceci implique que des temps *idle* vont se produire sur chaque processeur conduisant à des cycles thermiques.

Cependant la recherche de la garantie de l'ordonnançabilité impose a priori cette situation. Une amélioration de la solution pourrait être alors d'utiliser des approches de gestion en ligne de la fréquence et de la tension (DVFS).

Notons que dans la technique proposée de calcul des configurations m_T et F_T, il est nécessaire de disposer d'au moins un critère d'ordonnançabilité pour l'algorithme d'ordonnancement global considéré. Par exemple pour global-EDF quatre critères d'ordonnançabilité sont utilisés dans [84], pour EDZL le critère développé dans [89] peut être utilisé.

Une approche plus pragmatique permettant d'obtenir des fréquences de fonctionnement a priori plus faibles par rapport à celle calculées dans l'approche précédente est d'opérer en simulation en utilisant l'environnement de simulation présenté dans le chapitre 3.

4.3.1.2 Approche basée simulation

Dans cette approche on perd la garantie sur le respect des échéances des tâches mais on obtient des solutions a priori plus optimisées du point de vue puissance, énergie et cycles thermiques.

Comme ci-dessus, on part d'une configuration où les m cœurs de l'architecture sont à l'état *running* et fonctionnent à température basse (par exemple 25°C). Dans ce cas, les cœurs sont cadencés à la fréquence F_{min} telle que :

$$\sum_i \frac{C_i}{P_i} = m$$

Où P_i représente la période de la tâche τ_i et C_i représente le pire temps d'exécution de la tâche τ_i lorsque les cœurs de processeur fonctionnent à la fréquence F_{min}. Choisir une fréquence inférieure à F_{min} conduirait nécessairement à des dépassements d'échéances de tâches. Cependant, suivant l'algorithme d'ordonnancement choisi, cette fréquence F_{min} peut conduire

également à des dépassements d'échéances de tâches. Aussi il est nécessaire de déterminer la valeur de fréquence $F \geq F_{min}$ telle que l'ensemble des tâches exécuté suivant l'algorithme d'ordonnancement considéré respecte en simulation les contraintes d'échéances.

Le principe de la détermination des configurations $\{T, m_T, F_T\}$ est le suivant :

1- On cherche la fréquence initiale et minimale qui assure le fonctionnement de la totalité des cœurs à la température ambiante (minimale).

2- On calcule la puissance dissipée avec cette configuration retenue (Fréquence, nombre de cœurs en état *running*, température).

3- On diminue le nombre de cœur *runninig* et on augmente la fréquence, tout en vérifiant par simulation sur une hyper-période les échéances des tâches, et on compare la nouvelle puissance dissipée avec la précédente valeur.

4- Si la nouvelle valeur de puissance dissipée est inférieure à la précédente on retient la nouvelle configuration (soit une fréquence plus élevée avec moins de cœurs en mode *running* pour cette valeur de température), sinon on maintient l'ancienne configuration.

5- On fait varier la température ambiante d'un pas ΔT et on revient au point 2.

Ainsi l'Algorithme 4.3 décrit cette procédure orientée simulation. Les lignes 4 à 6 permettent de calculer la fréquence minimale de fonctionnement pour les *m* cœurs de processeur *running* telle qu'aucun dépassement d'échéance n'est observé en simulation. La fonction *dealine_miss()* renvoie vrai si au moins un dépassement d'échéance est observé. A cette étape on fait l'hypothèse qu'avec les *m* cœurs *running* il existe une fréquence de fonctionnement valable (c'est-à-dire inférieure à la fréquence maximum F_{max} autorisée par le constructeur) qui permet aux tâches de s'exécuter suivant leurs échéances.

La boucle des lignes 8 à 22 simule l'augmentation de la température ambiante par pas de ΔT. Dans cette boucle on teste si avec $m_1 = m$-1 cœurs de processeur *running*, on obtient une puissance dissipée inférieure à celle obtenue avec *m* cœurs *running* pour cette même température (ambiante). Pour ce faire, il faut en premier lieu calculer la fréquence de fonctionnement pour les m_1 cœurs qui ne fait apparaître aucun dépassement d'échéances. Si la puissance totale avec *m-1* cœurs de processeur est plus faible qu'avec *m* cœurs alors on enregistre cette configuration $\{m_1, F_{m-1}\}$ pour le seuil de température T_{amb}, qui sera par la suite notée $\{T_{amb}, m_T, F_T\}$. On teste si F_{m-1} est inférieur à F_{max}. Si ce n'est pas le cas il est

inutile de poursuivre car, continuer à diminuer le nombre de cœurs *running* ne peut conduire à une solution qui vérifie les échéances des tâches.

La variation de la température ambiante simulée dans la boucle des lignes 8 à 22 et modélisée dans ATMI, est représentée sur la Figure 4.3.

La durée d'un palier de température ambiante est au minimum égale à $H*(F_{max} - F_{min})/ \Delta F$ où H est l'hyper-période des tâches de τ_i. Ce temps permet à chaque incrément de fréquence ΔF d'évaluer, sous STORM pendant une hyper-période, si toutes les échéances des tâches sont respectées.

La température T_{max} est au plus égale à celle définie dans les *datasheets* du circuit ou correspond à la température ambiante maximum prévue pour l'environnement cible dans lequel doit opérer le circuit.

En faisant varier ainsi la température ambiante par pas de ΔT on détermine itérativement tous les seuils de températures qui induisent une configuration différente. Dans la suite, chaque configuration $\{T_{amb}, m_T, F_T\}$ associée au $i^{ème}$ seuil de température identifié est référencée $\chi_T(i)$.

L'intérêt d'utiliser STORM avec ATMI est qu'à partir de cette température ambiante et des valeurs de puissance fournies par STORM, la température du circuit au niveau jonction est calculée par ATMI qui est ensuite retournée à STORM qui à son tour met à jour les valeurs des puissances statiques.

Ainsi par simulation on obtient les valeurs des puissances totales dissipées en fonction de la température ambiante.

Algorithme 4.3 : Détermination des configurations {T, m_T, F_T}

```
1.  begin
2.      F = Fmin ;
3.      m = mmax ;
4.   while (deadline_miss(m,F) == true) do
5.      F = F + ΔF ;
6.   end while
7.   Tamb = Tmin ; Push(Tamb, m, F);
8.   while (Tamb ≤ Tmax) do
9.      Pm = Ptot(m,F,Tamb) ;
10.     m1 = m-1 ; Fm-1 = F ;
11.     while (deadline_miss(m1, Fm-1) == true) do
12.         Fm-1 = Fm-1 + ΔF ;
13.     end while
14.     if (Fm-1 ≤ Fmax) then
15.        Pm-1 = Ptot(m1,Fm-1,Tamb) ;
16.        if (Pm-1 < Pm ) then
17.           m = m1 ; F = Fm-1 ; Push(Tamb, m, F);
18.        end if
19.        T amb = T amb + ΔT;
20.        else exit();
21.     end if
22.  end while
23. end
```

Il faut noter que cette étude en simulation n'est faite qu'une seule fois pour un jeu de tâches et un algorithme d'ordonnancement donnés. Elle correspond à la première étape hors ligne de l'approche.

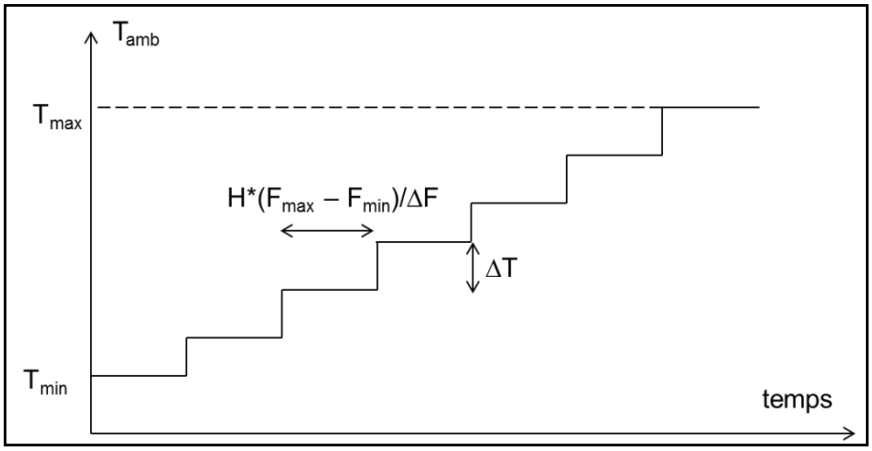


Figure 4.3 : Variation de la température ambiante dans ATMI

4.3.2 Algorithme de réduction des cycles thermiques

On dispose d'un ensemble de triplets (T, m_T, F_T) déterminés hors ligne et qui permettent de fixer, pour un niveau de température donné, le nombre de cœurs de processeur à utiliser dans l'architecture et leur fréquence de fonctionnement. Pour ce niveau de température, cette configuration conduit à réduire la puissance dissipée (pour une architecture où une seule fréquence est la même pour tous les cœurs). De plus, comme la fréquence de fonctionnement est la plus faible possible tenant compte des échéances des tâches, les cycles thermiques sont globalement minimisés. Il reste à déterminer comment ces configurations sont effectivement utilisées en ligne, ce qui est illustré dans l'Algorithme 4.4, dont ses principes sont les suivants:

1- On suppose que la température ambiante initiale du système est basse. On part de la configuration initiale qui correspond à la fréquence minimale et le nombre maximum de cœurs actifs.

2- Pour chaque évènement d'ordonnancement, la température au niveau jonction est calculée (compte tenu de la température ambiante et de son augmentation éventuelle).

3- Le contrôleur thermique observe les valeurs de température calculées et les compare par rapport à la valeur de température seuil de la configuration active.

4- Il sélectionne parmi les configurations, déterminées par l'Algorithme 4.3.

5- la configuration qui correspond à la température mesurée.

Ainsi, à l'initialisation on utilise la configuration $\chi_T(0)$ qui active l'ensemble des m cœurs de processeur à la fréquence $F_a = \chi_T(0).\ F_T$.

Ici, on suppose que notre système se trouve initialement dans un environnement à température ambiante peu élevée. Pour tenir compte de la température ambiante réelle il suffirait de sélectionner la configuration initiale adaptée par rapport à une mesure de température ambiante.

La fonction *activate*(m_a, F_a, $\chi_T(0).S_T(0)$) active tous les cœurs de processeur et l'ordonnanceur dispose d'une architecture à m processeurs pour ordonnancer les tâches suivant l'algorithme d'ordonnancement retenu.

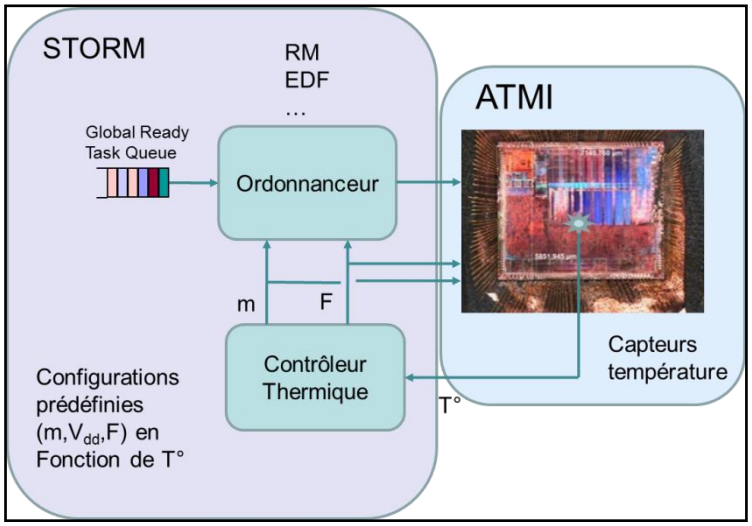


Figure 4.4 : Contrôleur thermique

Pendant le fonctionnement du système, à chaque événement d'ordonnancement significatif, le contrôleur thermique (Figure 4.4) lit la température des cœurs (calculée dans ATMI) et retient la température la plus élevée, soit *max_temp*. A partir de cette température on identifie la configuration ayant l'index *config* le plus grand dans l'ensemble χ_T telle que $\chi_T(config).T \leq max\_temp$. Ceci est réalisé par la fonction *identify_config*(χ_T,*max_temp*). Par exemple si les températures de seuil pour les configurations retenues sont égales à 25°C, 55°C, 70°C et 85°C, si la température mesurée est de 65°C alors c'est la configuration ayant le seuil de 55°C qui est appliquée.

Algorithme 4.4 : Heuristique de réduction des cycles thermiques

```
1.  begin
2.  m_a = χ_T(0). m_T ; F_a = χ_T(0). F_T ;
3.  activate(m_a, F_a, χ_T(0).S_T(0));
4.  current_config = 0 ;
5.  t_prev = current time ();
6.  for each scheduling event do
7.     max_temp = read_max_temp(m) ;
8.     config = identify_config(χ_T,max_temp) ;
9.     if (config != current_config) then
10.       m_a = χ_T(config). m_T ; F_a = χ_T(config). F_T ;
11.       index_active_core = 0 ;
12.       activate(m_a, F_a, χ_T(config).S_T(index_active_core));
13.        t_prev = current time ();
14.    else
15.      t = current time ();
16.    if (t – t_prev ≥ χ_T(config).θ_T) then
17.      index_active_core = (index_active_core) +1 % r ;
18.      activate(m_a, F_a, χ_T(config).S_T(index_active_core));
19.      t_prev = t ;
20.    enf if
21.    end if
22.  end for
23. end
```

Deux cas se présentent alors. Si la configuration identifiée diffère de celle en cours d'utilisation alors cette nouvelle configuration ayant m_a cœurs *running* fonctionnant à la fréquence F_a est appliquée. Pour ce faire, on active les m_a cœurs de processeurs dont les numéros sont renseignés dans $\chi_T(config).S_T(0)$, les $m - m_a$ autres cœurs sont mis en mode repos (lignes 10 à 12). Dans le cas contraire on reste avec la même configuration. Cependant si cette configuration est active depuis un temps supérieur ou égal à $\chi_T(config).\theta_T$ alors on

procède à une permutation sur les processeurs *running* en utilisant les numéros de cœurs présents dans la liste $\chi_T(config).S_T(index\_active\_core + 1 \% r)$ où r est le nombre de listes de numéros présents dans $\chi_T(config).S_T$ (lignes 16 à 20). Cette permutation est réalisée dans le but de limiter l'apparition de points chauds sur le circuit (cycles thermique spatiaux). Bien sûr, l'architecture matérielle doit supporter la migration de tâches de manière à autoriser la mise en mode repos de certains processeurs *running* et au contraire activer d'autres processeurs qui étaient en mode repos.

Dans la partie qui suit, nous détaillons la technique de détermination des cœurs actifs dans une configuration donnée.

4.3.3 Détermination des cœurs actifs dans une configuration donnée

Lorsque dans la configuration appliquée χ_T (*config*).S_T(*index_active_core*) le nombre de cœurs actifs est plus petit que le nombre total de cœurs disponibles dans l'architecture, il est nécessaire d'identifier les numéros de cœurs à activer dans l'architecture de manière à ce que la température et les gradients de température soient réduits à la surface du circuit. Si pour des circuits contenant deux cœurs cette question ne se pose pas réellement, nous avons étudié cependant ce problème dans le cas de circuits plus complexes (possédant un plus grand nombre de cœurs). Du fait de la migration possible des tâches, cette optimisation ne dépend principalement que de la topologie définie pour le circuit considéré. On cherche donc à identifier à partir des caractéristiques des processeurs et de la topologie du circuit différentes combinaisons de m_a cœurs actifs parmi m telles que la température et les gradients de températures soient les plus faibles. Il n'est pas nécessaire de considérer systématiquement toutes les combinaisons possibles, en effet, on cherche à équilibrer dans le temps la répartition de la charge sur les processeurs induite par les tâches et leur impact sur la température.

L'Algorithme 4.5 décrit la technique des cœurs à activer et à désactiver dans l'architecture, en se basant sur les mesures fournies par ATMI pour calculer les températures minimale et maximale des cœurs. Pour chaque configuration sélectionnée avec m_a cœurs, nous effectuons une permutation des cœurs actifs les plus chauds avec les cœurs inactifs les moins chauds.

Par exemple, avec la topologie d'un circuit à 16 cœurs illustrée sur la Figure 4.5, en considérant 15 cœurs actifs parmi 16, les 4 configurations qui minimisent les gradients de température sont celles où l'un des 4 cœurs 5, 6, 9 ou 10 est inactif (c'est-à-dire la

list_inactive_core de l'Algorithme 4.3 qui contient forcément l'un de ces cœurs). En effet, ceux sont les cœurs au centre du circuit qui induisent et subissent la température la plus élevée par rapport aux cœurs situés à la périphérie du circuit.

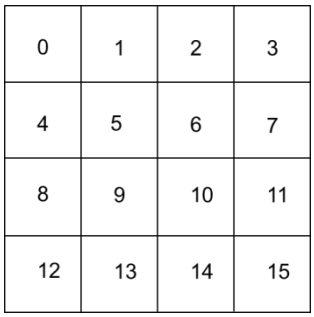


Figure 4.5 : Exemple de topologie à 16 cœurs de processeurs

Ce problème de réduction des gradients de température a été abordé par exemple dans [90],[91]. Dans [90], l'étude porte sur la définition d'une topologie visant à réduire les points chauds (*hot spots*) à partir de la connaissance des puissances dissipées par chaque composant. Cette prise en compte de la topologie du circuit est également relevée dans [92].

Dans [91], le problème abordé est plus général car il intègre la modélisation des tâches à exécuter et cherche également à associer une fréquence de fonctionnement à chaque cœur en tenant compte de la topologie du circuit. Les auteurs modélisent le problème suivant un programme linéaire en nombres entiers dans lequel est décrit à la fois l'impact sur la température de la puissance dissipée par chaque cœur actif et l'influence réciproque en termes de température d'un cœur sur ses voisins, voisins au sens de la topologie. Pour modéliser cette interaction entre cœurs une matrice $a_{i,,j}$ est utilisée pour représenter la conductance entre le cœur i et le cœur j. Cette conductance dépend de la conductivité thermique du silicium (approximée par $150*(300/T)^{4/3}$ en W/mK dans [90]) ou du métal (400 W/mK pour le cuivre) et des tailles en trois dimensions des blocs qui relient physiquement deux cœurs adjacents [90]. Ainsi la température d'un cœur i à un instant k+1 est exprimée selon [91] par :

$$T_{k+1,i} = T_{k,i} + \sum_{j \neq i} a_{i,j} \cdot (T_{k,j} - T_{k,i}) + b_i \cdot p_i$$

Où p_i est la puissance dissipée par le cœur i lorsqu'il est actif et b_i est une constante dépendant de la capacitance thermique du cœur i [90].

Ce type de modélisation, interaction et dépendance en température entre cœurs, est supporté par ATMI à travers la technique de superposition [65].

Algorithme 4.5 : Permutation des cœurs actif/inactif

```
1.  begin
2.  list_active_core=m_a .core ;
3.  list_inactive_core= (m - m_a ).core;
4.  if ((list_active_core >= list_inactive_core) then
5.      activate(list_inactive_core) ;
6.      for index in list_active_core do
7.          i=read_max_temp_core(index) ;
8.          desactivate( core.i) ;
9.      end for
10. end if
11. else (list_active_core < list_inactive_core)
12.     desactivate(list_active_core) ;
13.     for index in list_active_core do
14.         i=read_min_temp_core(index) ;
15.         activate(core.i) ;
16.     end for
17. end
```

4.4 Conclusion

Dans ce chapitre nous avons présenté une heuristique globale afin de réduire les cycles thermiques et les pics de températures.

Cette heuristique se base sur deux étapes, une étape est dite hors ligne où on détermine un ensemble de triplets : la fréquence, le nombre de cœurs actifs et la température associée pour chaque configuration. La deuxième étape, en ligne, détermine la manière dont ces configurations sont effectivement utilisées.

De plus si le nombre de cœurs actifs est plus petit que le nombre total de cœurs disponibles dans l'architecture, une technique de permutation entre ces cœurs (actifs/inactifs) est présentée afin de limiter les gradients de températures à la surface du circuit et d'équilibrer les cycles thermiques entre les cœurs.

Dans le chapitre suivant nous présenterons les résultats obtenus en appliquant cette technique proposée, pour trois algorithmes d'ordonnancement global : EDF, RM et EDZL.

CHAPITRE 5. Résultats expérimentaux

5.1 Introduction

Dans ce chapitre nous présentons les résultats obtenus en appliquant la technique proposée de réduction des cycles thermiques en se basant, dans la simulation, sur l'environnement décrit dans le chapitre 3.

Nous nous intéressons aux cycles thermiques ainsi qu'à la température maximale subis par les cœurs de l'architecture, résultants de l'ordonnancement de jeux de tâches suivant les trois algorithmes : global-RM, global-EDF et global-EDZL.

Nous avons considéré l'architecture du circuit MPC5517 de la société Freescale, étendue à 4 cœurs afin de mettre en évidence l'influence du nombre de cœurs actifs sur les résultats en température. Nous avons aussi considéré un autre type de processeur (le UltraSparc T1 de Sun Microsystems) qui consomme plus de puissance par rapport au circuit MPC5517 et ce afin d'évaluer notre dans un contexte plutôt orienté calcul intensif.

5.2 Considérations préliminaires

5.2.1 Jeux des tâches

Les jeux de tâches considérés dans les expérimentations effectuées sont produits par un générateur de tâches développé par l'IRCCyN. Ces jeux de tâches induisent différentes valeurs de charge des processeurs. Cette charge est bien-sûr dépendante de la fréquence appliquée au processeur. Nous avons fait fonctionner les processeurs à des fréquences différentes (DVFS) de manière à faire évoluer, pour un même algorithme d'ordonnancement, les intervalles de temps où chaque processeur est en mode *running* ou en mode *idle*. En effet, une augmentation de la fréquence conduit à une diminution du temps d'exécution des tâches et par conséquent à une augmentation a priori des intervalles *idle* du ou des processeurs.

Les caractéristiques de chaque ensemble de tâches considéré dans cette étude sont données dans le Tableau 5.1.

Tableau 5.1 : jeux des tâches considérés

Taux d'utilisation / Caractéristiques Des tâches	90%	80%	60%
Périodes (ms)	30, 36, 40, 45, 50, 30	30, 36, 40, 45, 50, 30	30, 36, 40, 45, 50, 30
WCET (ms)	22, 9, 21, 28, 46, 16	24, 25, 28, 8, 20, 12	26, 10, 18, 8, 7, 14

Les valeurs des temps d'exécution pire cas des tâches (donnés en millisecondes) et des taux d'utilisation indiqués dans le tableau correspondent à une architecture monoprocesseur de type MPC5517 qui fonctionne à sa fréquence maximale, soit 80MHz.

5.2.2 Scenarios de variation de la température ambiante

Comme mentionné dans la section 2.3.1.3, la température ambiante dans les automobiles peut subir de grandes variations, par exemple si on considère que le moteur du véhicule est à l'arrêt ou en fonctionnement ou si le véhicule est en plein soleil l'été ou à l'ombre l'hiver.

Deux scénarios de variation de température ambiante sont considérés :

- Le premier : on a adopté une variation linéaire croissante de la température comme la montre la Figure 5.1 , où la température croit de 25°C à 100°C pendant 70s. Cette montée en température dans un intervalle de temps de 70s n'est pas très réaliste. Cependant, avec des tâches qui ont des temps d'exécution de l'ordre de quelques millisecondes, le nombre d'événements d'ordonnancement à considérer dans la simulation thermique sur plusieurs minutes conduit à des temps de simulation qui peuvent devenir prohibitifs. On s'est donc limité à un intervalle de temps simulé permettant d'observer les phénomènes souhaités sans entraîner des simulations trop longues. A noter que la valeur de température maximum considérée (100°C) est cependant tout à fait possible dans une automobile. La montée linéaire en température est utilisée dans ce premier scénario pour valider les décisions relatives aux seuils de température utilisés pour les changements de configuration.

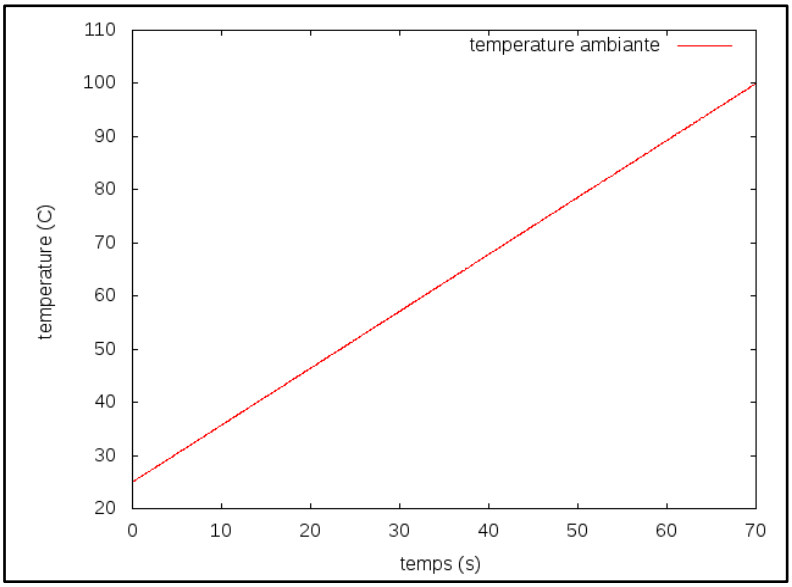


Figure 5.1 : Variation linéaire de la température ambiante en simulation

- Le deuxième scénario proposé consiste en une variation de la température ambiante suivant un schéma différent, afin d'évaluer si les algorithmes d'ordonnancement ont un impact sur les cycles thermiques (en fonction des positions des temps *idle* des processeurs et en fonction de la puissance statique dépendante de la température). Ce deuxième scénario sert de base de comparaison en vue de l'évaluation de la technique proposée de réduction des cycles thermiques (chapitre 4). La Figure 5.2 montre la courbe de température ambiante considérée. De même que dans le premier scénario, pour limiter les temps de simulation (essentiellement sous ATMI) nous avons fait varier la température dans des proportions importantes (de 25° à 100°) en un temps limité.

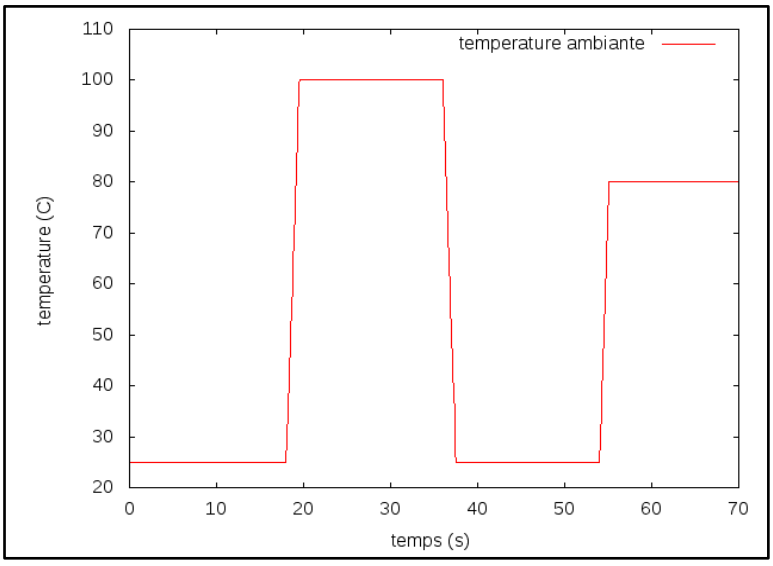


Figure 5.2 : Exemple considéré de variation de la température ambiante

Ces deux scénarios sont utilisés dans le paragraphe suivant dans le but de comparer, sur ces cas, l'effet de la gestion de la technique proposée de réduction des cycles thermiques par rapport à une technique classique d'ordonnancement de tâches (sans utilisation du mode DPM et de la technique DVFS).

5.3 Analyse avec une technique classique d'ordonnancement

Nous montrons tout d'abord dans ce paragraphe les résultats, avec les deux scenarios considérés, en utilisant soit tous les cœurs actifs (soit 4 cœurs) soit un seul cœur actif fonctionnant à sa fréquence maximale. On obtient ainsi les deux situations extrêmes qui permettent ensuite de comparer ces résultats avec ceux issus de notre technique adoptée.

5.3.1 Analyse en simulation de la température

Cette étude a pour but d'analyser la température maximale atteinte au niveau jonction en fonction de la température ambiante, sans utiliser les techniques DPM et/ou DVFS.

En utilisant le jeu de 6 tâches ayant un taux d'utilisation de 60% du Tableau 1 avec une architecture à 4 cœurs fonctionnant à 16MHz sous EDF, dans l'environnement décrit par le premier scenario (Figure 5.1), on obtient la variation en température illustrée dans la Figure 5.3. Ces variations de température sont ici très faibles ce qui s'explique par la faible puissance dissipée de l'architecture qui s'élève à 90mW lorsque la fréquence de fonctionnement est fixée à 16MHz.

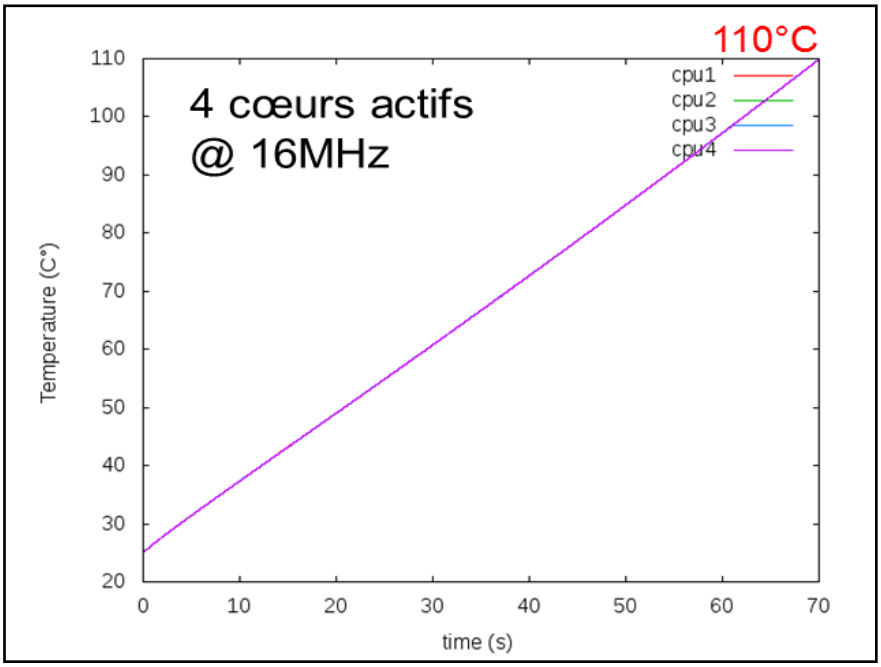


Figure 5.3 : Variation en température de l'architecture à 4 cœurs actifs à 16MHz (EDF)

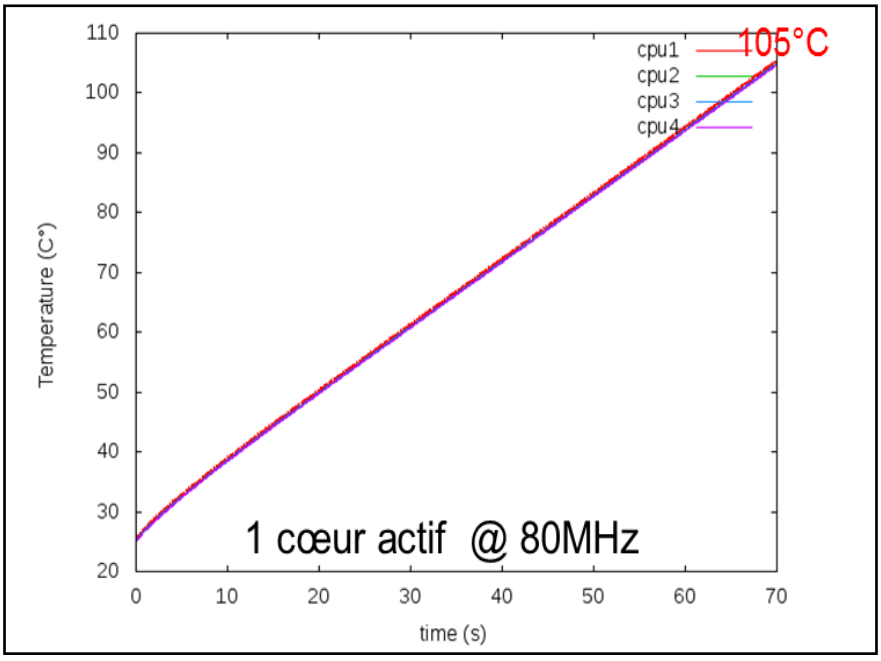


Figure 5.4 : Variation en température de l'architecture à 4 cœurs à 80 MHz (EDF) dont un seul est actif.

En utilisant un seul cœur fonctionnant à 80MHz (et les 3 autres cœurs en mode *sleep*) sur le même jeu de tâches ordonnancé par EDF, on obtient l'évolution de température illustrée sur la Figure 5.4.

En effectuant un zoom sur un intervalle de temps de 5 secondes (entre l'instant 50s et 55s), on peut vérifier l'apparition des cycles thermiques indiqués sur la Figure 5.5. Seul le premier cœur (CPU 1) est actif et son activité induit des variations de température par diffusion thermique sur les autres cœurs qui sont en mode *sleep*. On peut noter que la température maximale atteinte par l'architecture à 4 cœurs fonctionnant à 16Mhz est de 110°C (Figure 5.4) alors qu'avec un seul processeur la température maximum n'est que de 105°C, pour une température ambiante maximale de 100°C. Cet écart est dû à la puissance statique qui dans le premier cas est supérieure étant donné que tous les cœurs sont actifs dans ce cas.

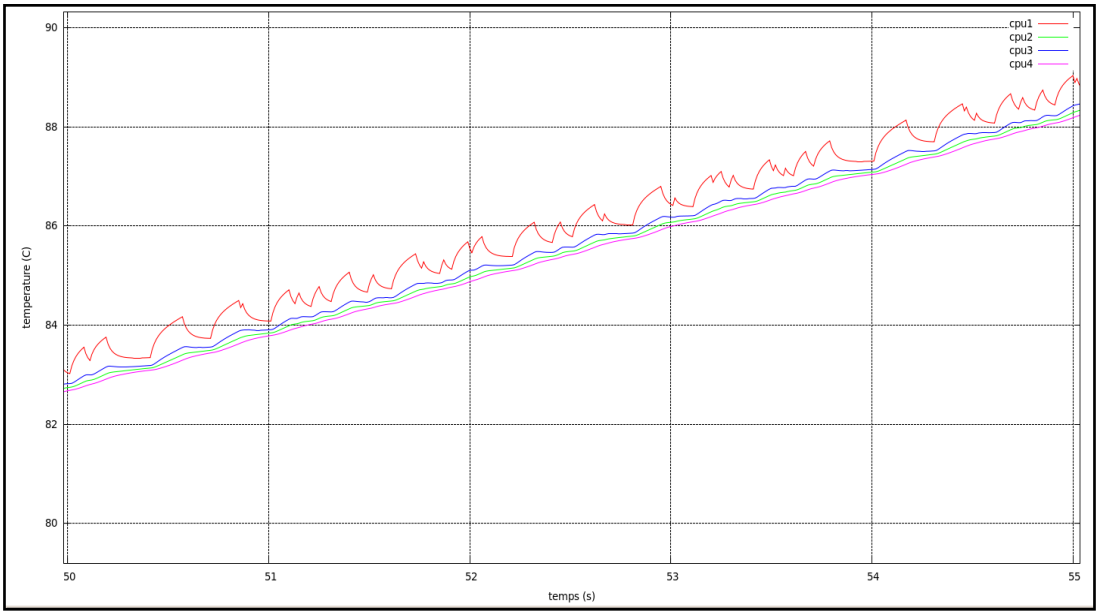


Figure 5.5 : Cycles thermiques d'un cœur fonctionnant à 80MHz (EDF)

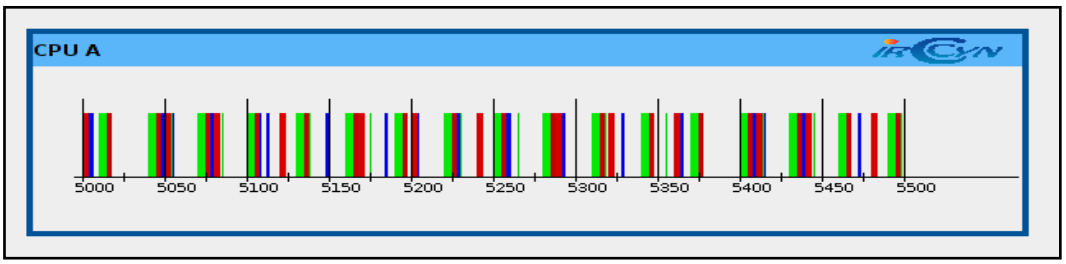
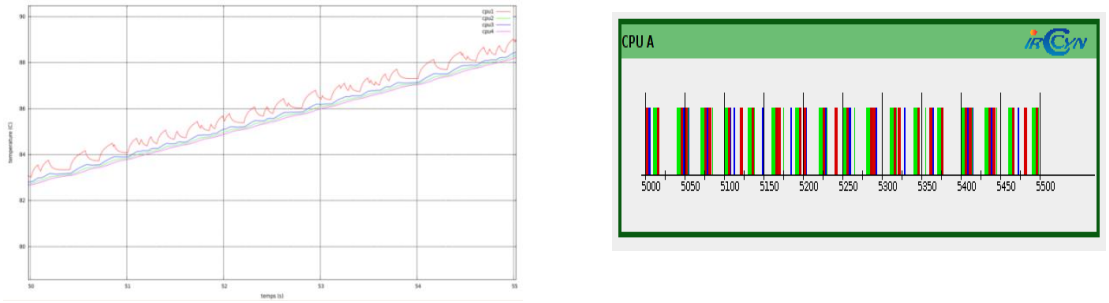


Figure 5.6 : Ordonnancement relatif au cœur actif fonctionnant à 80 MHz (EDF)

La Figure 5.6 présente l'ordonnancement de l'ensemble des tâches sur le CPU A (représenté par cpu1 sur la Figure 5.5). Le temps en abscisse est donné en milisecondes. Nous remarquons les temps d'inactivité et activité du cœur qui correspondent bien aux variations de température illustrées dans la Figure 5.5.



a) b)

Figure 5.7 : Cycles thermiques (a) et Ordonnancement (b) pour un cœur fonctionnant à 80MHz (EDZL)

De même, en utilisant un seul cœur fonctionnant à 80MHz (et les 3 autres cœurs en mode *sleep*) sur le même jeu de tâches (60%) cette fois ordonnancé par EDZL et RM, nous obtenons les évolutions de température illustrées sur les Figure 5.7 et Figure 5.8 respectivement (représentées ici entre les instant 50s et 55s). On peut remarquer les fortes similitudes entre les résultats en température obtenus suivant les trois algorithmes d'ordonnancement étudiés.

a) b)

Figure 5.8 : Cycles thermiques (a) et Ordonnancement (b) pour cœur fonctionnant à 80MHz (RM)

Maintenant, nous considérons le même cas d'un taux d'utilisation 60% et le jeu de 6 tâches ordonnancées sous RM, avec le deuxième scénario de variation de température ambiante (Figure 5.2). La Figure 5.9 donne le résultat de simulation de la température lorsqu'un seul cœur est actif à la fréquence de 80MHz. L'amplitude de variation importante de la température (Figure 5.9. a) ne permet pas de visualiser les cycles thermiques. La Figure 5.9 b

détaille l'évolution de la température entre les instants 19s et 36s. On remarque que le cœur actif (la courbe en rouge) est celui qui produit les plus importantes variations de température. La température des autres cœurs fluctue en raison de la conductance thermique au niveau *die* entre les cœurs de processeur. Cette conductance dépend du matériau, de la taille des cœurs et de la position des cœurs sur le *die*.

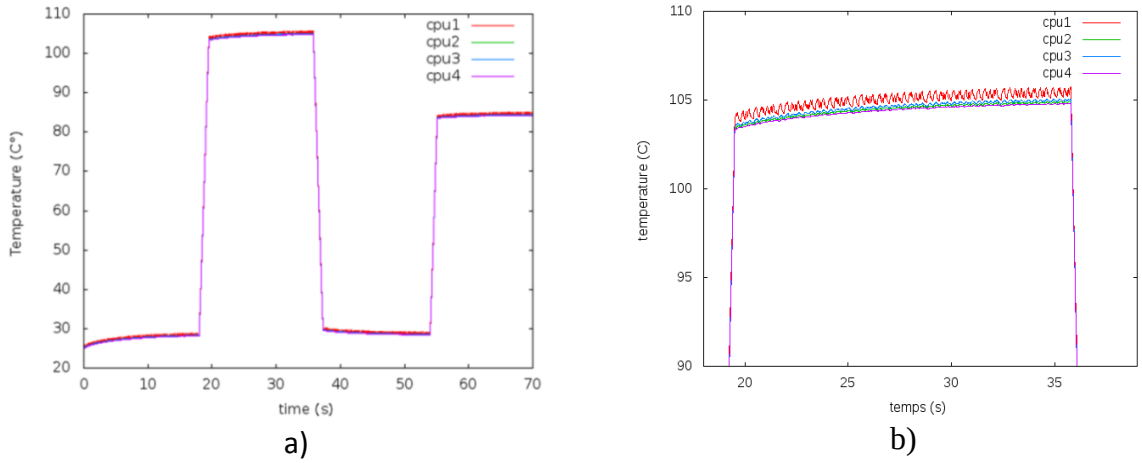


Figure 5.9 : Température des cœurs : 1 cœur actif à 80MHz suivant RM

Nous avons effectué une simulation du même jeu de tâches dans les mêmes conditions avec un ordonnancement G-EDF et un ordonnancement G-EDZL. Les courbes de températures obtenues sont très similaires à celles fournies par G-RM (Figure 5.9).

5.3.2 Analyse des cycles thermiques

Dans cette partie on présente les résultats relatifs à la variation des cycles thermiques. Pour évaluer ces cycles, on calcule la somme cumulée pour les différents cœurs de l'architecture des amplitudes de ces cycles au cours de la durée de simulation, et pour chacun des deux cas de variation de température ambiante (Figure 5.1 et Figure 5.2).

Pour la variation linéaire de la température ambiante (Figure 5.1), on calcule les cycles thermiques (CT) relatifs à l'évolution de la température des Figure 5.3 (4 cœurs actifs à 16MHz) et Figure 5.4 (1 seul cœur actif à 80MHz) pour obtenir les résultats présentés dans la Figure 5.10. Dans la configuration à 4 cœurs actifs à 16MHz (Figure 5.10 gauche), aucun cycle thermique n'est identifié : la température croît de manière continue et sa dérivée ne

s'annule jamais. Ceci est dû à la faible valeur de la fréquence utilisée qui induit une charge de calcul élevée pour les processeurs limitant les intervalles où les processeurs sont en mode *idle*. Dans le cas d'un seul processeur actif à 80MHz (Figure 5.10 droite), des cycles thermiques apparaissent et leur cumul conduit à un total de 140°C. On remarque que le CPU3 (courbe bleue) subit quelques cycles thermiques, ceux-ci sont induits par les variations en température du CPU1.

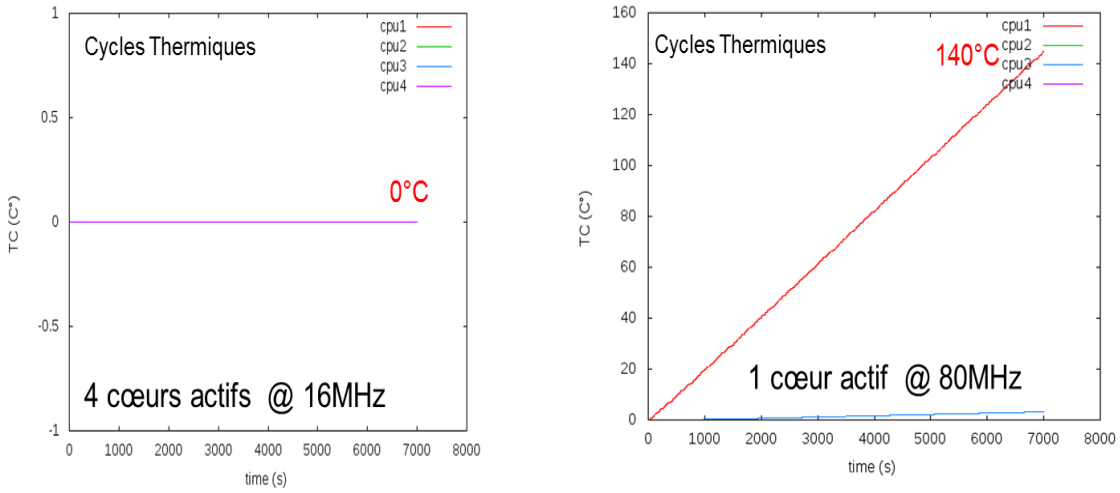


Figure 5.10 : Cycles thermiques cumulés par processeur

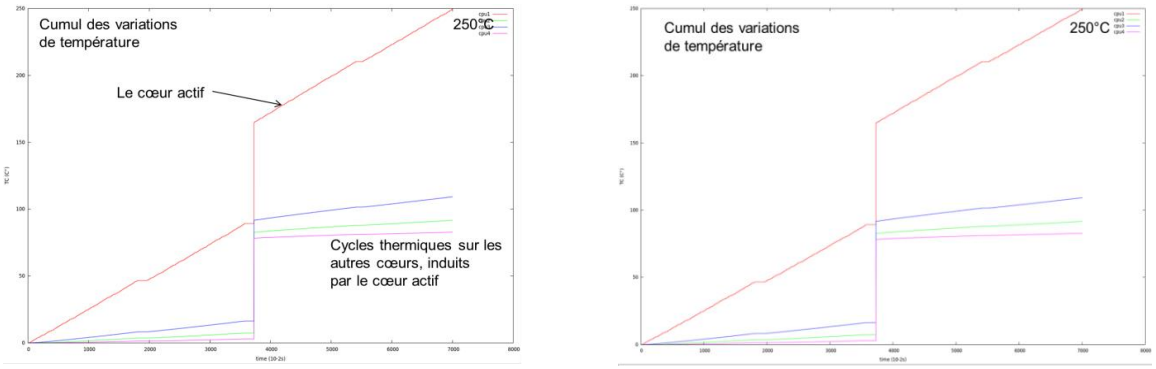


Figure 5.11 : Cycles thermiques avec 1 cœur actif à 80MHz ordonnancé suivant RM (figure de gauche) et EDF (figure de droite)

En prenant maintenant le deuxième cas de la variation dans la température ambiante (Figure 5.2), on calcule la somme des cycles thermiques pour G-RM et G-EDF relatifs aux évolutions de la température des Figure 5.9 (ordonnancement G-RM). On obtient les résultats de la Figure 5.11 où on observe une augmentation des CT plus importante pour le processeur actif (courbe rouge, CPU1) comparée à celle des autres processeurs. Le saut en température

qui intervient à partir de 37s est dû à la variation de la température ambiante qui réalise un cycle complet entre les instants 18s et 37s. La somme totale des cycles thermiques sur la durée de simulation pour le CPU1 atteint 250°C dans les deux cas. On remarque également que les cycles thermiques des autres processeurs dépendent du positionnement des cœurs à la surface du *die*, et de leurs tailles. De même nous avons effectué la même analyse avec EDZL avec un seul processeur, on obtient alors les mêmes résultats qu'avec EDF ce qui est logique car EDZL est à la base un ordonnancement de type EDF.

Les similarités entre G-EDF et G-RM au niveau des CT sont confirmées par l'évolution quasi identique des variations de température observées dans les deux cas.

En effectuant les mêmes analyses avec le jeu de tâches ayant un taux d'utilisation de 80% (au lieu de 60%) on observe les mêmes similarités entre RM, EDF et EDZL.

Lorsque la charge processeur augmente à 90%, les deux algorithmes RM et EDF font apparaître des dépassements des échéances ses tâches, et ce contrairement à EDZL qui reste opérationnel dans ce cas. La Figure 5.12 illustre les fluctuations de température et les cycles thermiques des processeurs dans le cas EDZL et un taux d'utilisation de 90%.

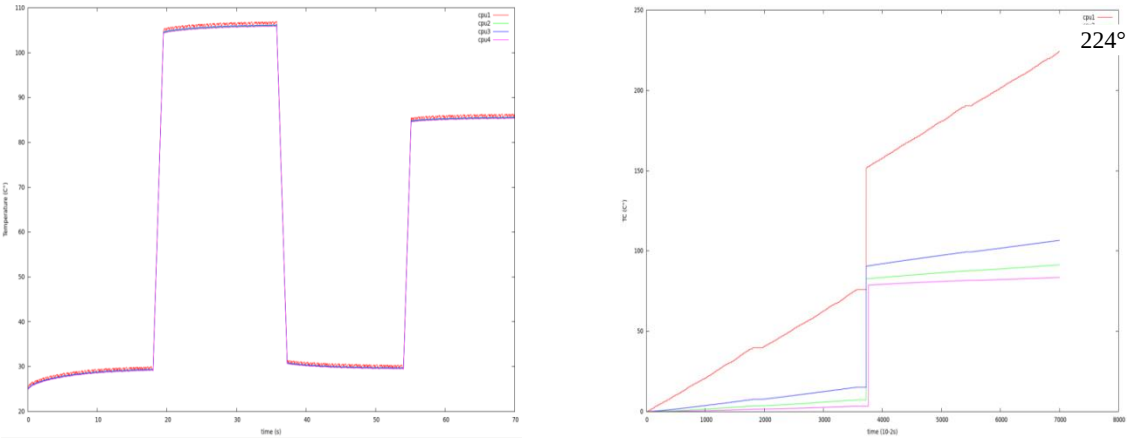


Figure 5.12. Résultats pour EDZL, 1 processeur à 80MHz, taux d'utilisation de 90% : variation de température (figure de gauche) et cycles thermiques subis (figure de droite).

Ainsi on observe ici que le nombre de cycles thermiques subis par les processeurs avec EDZL est plus faible (224°) par rapport aux nombres de cycles observés lorsque la charge processeur est de 60% (250°). Ceci confirme bien les observations déjà réalisées précédemment, à savoir que quel que soit l'algorithme d'ordonnancement utilisé, moins il y a de temps *idle* dans l'ordonnancement, moins il y a de cycles thermiques subis par les processeurs.

On peut également en déduire la règle empirique suivante :

> *Si pour un jeu de tâches donné, et un algorithme d'ordonnancement de type work-conserving*[2]*, en augmentant le ou les temps d'exécution d'une ou de plusieurs tâches (e.g. algorithmes de calcul plus précis), ou en diminuant la fréquence de fonctionnement des processeurs (tous les temps d'exécution des tâches augmentent de façon proportionnelle), si le nombre de cycles thermiques subis par les processeurs ne diminue pas alors des dépassements d'échéances des tâches risquent fortement d'apparaître.*

Ceci résulte logiquement de la corrélation observée entre nombre de cycles thermiques subis et intervalles de temps *idle* des processeurs.

Considérons maintenant le cas de l'architecture à 4 cœurs actifs fonctionnant à la fréquence de 16MHz. Avec un ordonnancement global EDF sur le jeu de tâches ayant un taux d'utilisation de 60%, on obtient la variation en température et les cycles thermiques décrits par la Figure 5.13. On remarque que les cycles thermiques sont nettement plus faibles par rapport à un seul processeur fonctionnant à la fréquence maximum (Figure 5.12).

Ces cycles sont très faibles (le cycle induit par la température ambiante comptant pour 100-25 = 75°) et très bien répartis entre les processeurs ce qui va dans le sens d'accroître la fiabilité du circuit. Les différences d'occurrence des fortes variations des cycles thermiques (Figure 5.13 droite) sont dues aux instants différents de détection par l'algorithme de calcul des CT des changements de signe des dérivées des courbes de température. En effet, avec des variations de température des cœurs très faibles, un cycle thermique produit par un CPU A, après diffusion de chaleur vers un autre CPU B, peut se trouver inhibé sur le CPU B du fait du début de la montée rapide de la température ambiante. Par ailleurs, on observe que la température maximale atteinte est de 110°C, soit 5°C de plus par rapport à la configuration à un seul cœur actif. Ainsi on remarque qu'après l'instant 19s la température monte progressivement d'environ 103° jusqu'à 110°C (instant 36s). Ceci est dû à l'influence réciproque entre la puissance statique dissipée et la température au niveau jonction. Ainsi le couplage entre STORM (qui calcule les valeurs de puissances statique et dynamique en fonction de l'ordonnancement et de la température) et ATMI permet de mettre en évidence ce phénomène.

[2] Dans un ordonnancement de type *work-conserving*, un tâche prête ne peut rester en attente d'exécution si au moins un processeur de l'architecture est dans l'état *idle*.

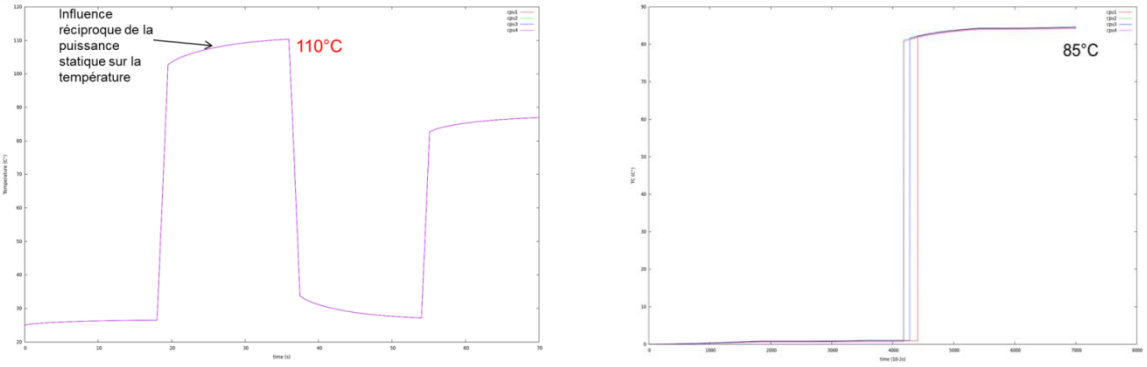


Figure 5.13 : Variation en température (figure de gauche) et cycles thermiques (figure de droite) avec 4 cœurs actifs à 16MHz et le jeu de tâches ayant un taux d'utilisation de 60%

Comme cela a été présenté dans le chapitre 2, la fiabilité d'un composant diminue lorsque ce composant est soumis à des températures élevées [31]. Une température du circuit élevée favorise l'apparition de phénomènes d'électromigration et augmente le courant statique qui à son tour accélère l'apparition de courts-circuits dans les diélectriques (phénomène de TDDB : *time-dependent dielectric breakdown*). Par conséquent, lorsque la température s'élève, il est important de trouver un compromis entre diminution des cycles thermiques et limitation de la température maximale.

Dans la suite on cherche d'une part à évaluer les gains apportés sur le nombre de cycles thermiques des processeurs par la technique proposée et d'autre part, à limiter autant que possible la température maximale de chaque cœur en situation de température ambiante élevée.

5.4 Analyse de la technique de réduction des CT

Dans cette partie nous décrivons les résultats obtenus concernant les cycles thermiques et les températures maximales produites sous les trois algorithmes globaux, en illustrant l'effet de la technique de permutation entre les cœurs actif et inactifs.

Tout d'abord, nous appliquons la première partie de la technique qui consiste à calculer les différents triplets {F,m_t,T}.

Le Tableau 5.2 illustre les différentes configurations retenues. Par exemple, si la température mesurée au niveau jonction (retournée par ATMI) ne dépasse pas 80°C, alors la fréquence de fonctionnement est de 16MHz avec un nombre de cœurs actifs maximum, soit 4 cœurs. Au-delà de cette température (80°C) le système passe à une fréquence de 48MHz et 2 cœurs actifs et maintient cette configuration tant que la température ne dépasse pas 100°C.

A une température supérieure à 100°C le système passe à un seul cœur actif et à la fréquence 66MHz. Pour rappel, les données constructeur du MPC5517 mentionnent que ce circuit est opérationnel dans la gamme de température [-40,+145°C] au niveau jonction.

Tableau 5.2 : Les différentes configurations calculées par l'Algorithme 4.3

Fréquence (Mhz)	Voltage (V)	m_t	T
16	0.6	4	80
48	1.05	2	100
66	1.3	1	-
80	1.5	1	-

5.4.1 Analyse de la température et CT

Dans un premier temps on utilise la variation linéaire de la température illustrée par la Figure 5.1. Avec le jeu de 6 tâches conduisant à un taux d'utilisation de 60% et un ordonnancement de type RM on obtient l'évolution de la température donnée par la Figure 5.14.

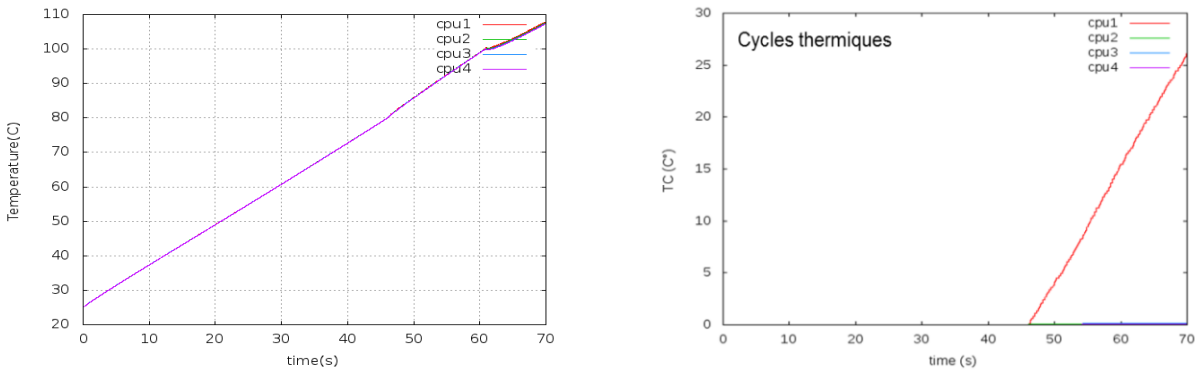


Figure 5.14 : Action du contrôleur thermique sur la température des cœurs, U= 60%, RM

Avec l'élévation linéaire de température on observe les changements de configuration déclenchés à chaque franchissement de seuil : à la température de 80°C on passe de 4 à 2 cœurs actifs et à la température de 100°C on passe de 2 à 1 cœur actif. Sur la courbe de température on remarque que lorsque la température dépasse le seuil activant la solution à un seul processeur actif, les fluctuations mesurées de température augmentent légèrement.

Ceci est illustré par la Figure 12 droite sur les cycles thermiques : à partir de l'instant 46s les cycles thermiques commencent à croître pour donner un total de 25°C à 70s. Les résultats avec EDF et EDZL sont très similaires à ceux de la Figure 5.14 , de ce fait ils ne sont pas détaillés.

Sur la Figure 5.15 sont présentées les courbes relatives à la variation de température et aux cycles thermiques avec le jeu de tâches à 60% ordonnancé sous RM et suivant la variation de température ambiante donnée Figure 5.2.

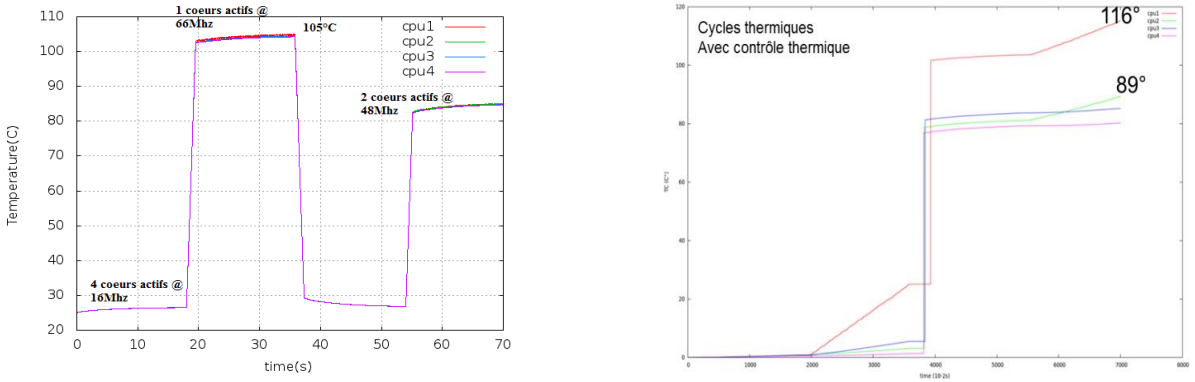


Figure 5.15 : Résultats obtenus avec gestion des cycles thermiques, U=60%, RM

On obtient un résultat (Figure 5.15 de gauche) assez proche de la combinaison des résultats illustrés dans les Figure 5.9 et Figure 5.13 . En effet, à température basse le comportement est celui relatif à une architecture à 4 cœurs actifs fonctionnant à fréquence basse alors qu'à température élevée le comportement est celui d'une architecture ayant un seul cœur actif à fréquence plus élevée (ici une fréquence à 66 MHz suffit à ordonnancer les tâches suivant leurs échéances.

Avec le jeu de 6 tâches conduisant à un taux d'utilisation de 80%, on obtient les résultats donnés par la Figure 5.16.

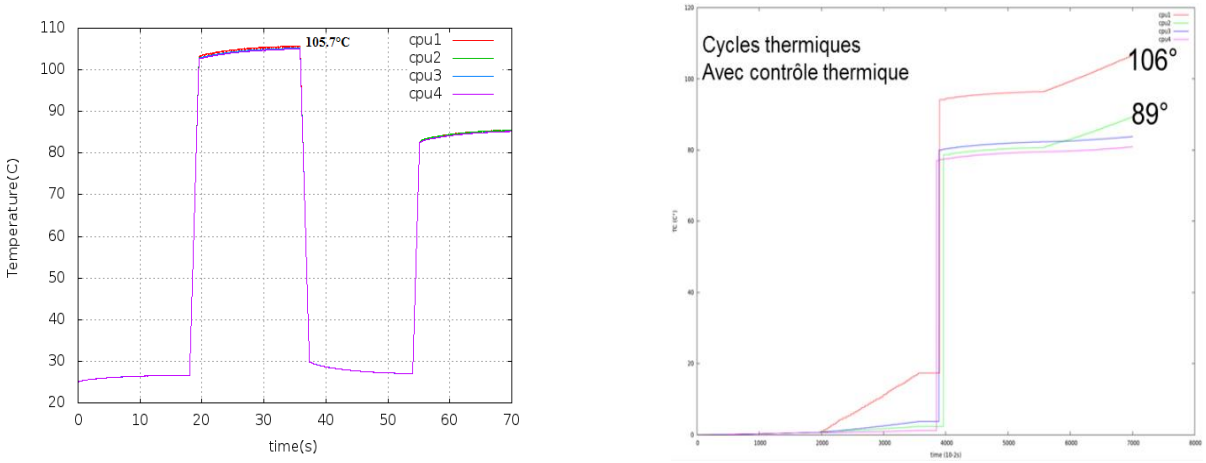


Figure 5.16 : Résultats avec RM sur le jeu de 6 tâches avec U = 80%

Ces résultats montrent la tendance déjà observée : lorsque le taux d'utilisation des processeurs augmente les temps *idle* diminuent et les cycles thermiques diminuent également (106° au lieu de 116°).

En passant au jeu de tâches ayant un taux d'utilisation égal à 90%, seul l'ordonnancement EDZL permet d'ordonnancer les tâches suivant leurs échéances. Dans ce cas on obtient les résultats en température illustrés sur la Figure 5.17. On constate le même phénomène que précédemment, les cycles thermiques diminuent du fait de l'augmentation du taux de charge (101°).

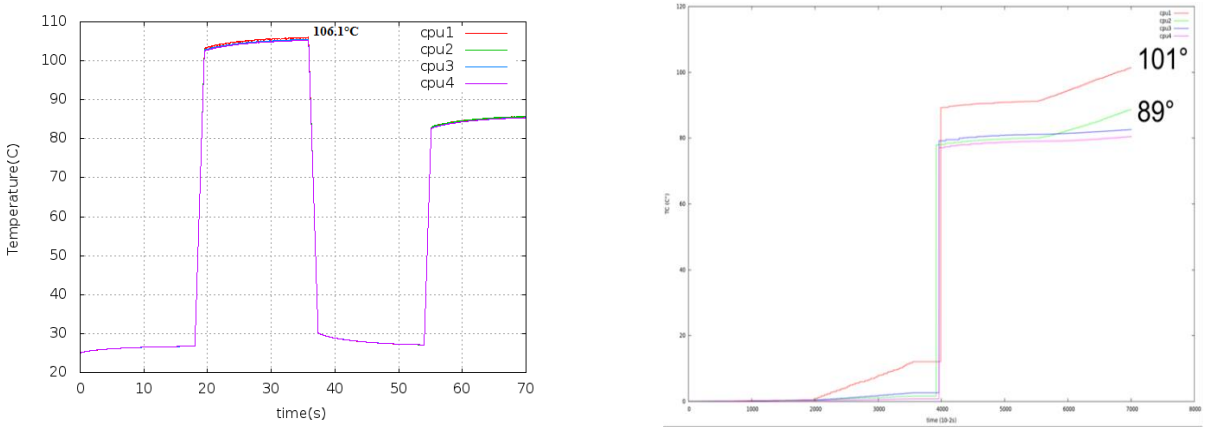


Figure 5.17 : Résultats avec EDZL et U=90%

Sur la Figure 5.18 est effectué un zoom des évolutions des cycles thermiques pour EDF et EDZL avec le jeu de tâches à U=90% dans l'intervalle de 44s à 54s, sachant qu'avec EDF ce jeu de tâches n'est pas ordonnançable. On remarque sur cette figure qu'avec EDF les cycles thermiques sont toujours plus nombreux que ceux produits avec EDZL (la courbe verte EDF est toujours située au-dessus de la courbe rouge EDZL). La stratégie d'ordonnancement développée dans STORM lors de dépassements d'échéances induit des temps *idle* des processeurs, aussi il est normal d'observer que la technique EDF conduise à un nombre de cycles thermiques plus élevé qu'EDZL sur cet exemple.

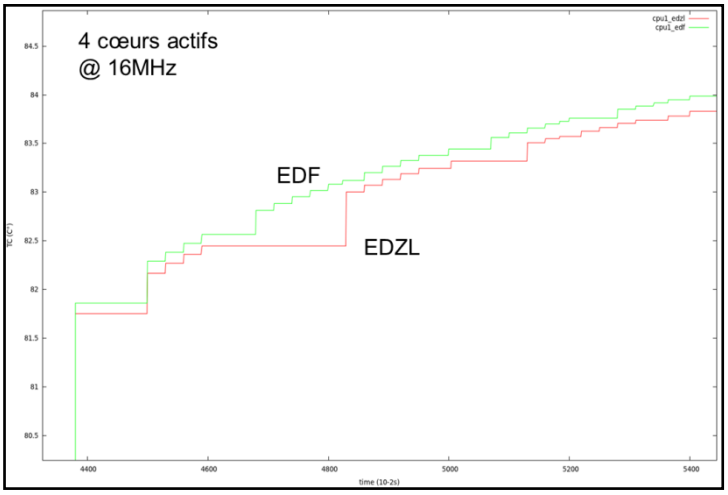


Figure 5.18 : Cycles thermiques pour EDF et EDZL pour U=90% entre les instants 44s et 54s.

5.4.2 Analyse de l'effet des permutations entre cœurs actifs/inactifs

Nous avons proposé dans la section 4.3.3, lorsque dans une fenêtre de temps donnée le nombre de cœurs actifs reste constant et inférieur à m_{max}, d'effectuer des permutations entre les cœurs actifs et inactifs afin de répartir les cycles thermiques sur l'ensemble des processeurs. Nous avons appliqué cette approche à l'exemple précédent. Nous avons considéré le cas du jeu de 6 tâches avec U=90% ordonnancé sous EDZL avec au plus m=4 processeurs. Sur cet exemple, lorsqu'une même configuration avec un nombre k de processeurs actifs inférieur à m_{max} est appliquée depuis plus de 4,5s alors on procède à une permutation pour changer l'état actif/inactif de m-k processeurs.

Cette période de 4,5s a été déduite après analyse de la constante thermique du circuit qui se situe entre 12s et 15s pour le circuit décapsulé. Nous avons choisi le tiers de cette valeur de manière à ce qu'en présence d'une période longue d'activité induite par les tâches, on change de processeur actif avant que celui-ci n'atteigne sa température maximale.

La Figure 5.19 montre la variation de température obtenue. On atteint au maximum 106° ce qui correspond à la température maximum observée sans changement de configuration (Figure 5.17). A température basse les 4 cœurs sont actifs, il n'y a pas de changement de configuration. Ces changements interviennent lorsque la température augmente c'est-à-dire dans les intervalles où seulement un ou deux processeurs sont seulement actifs.

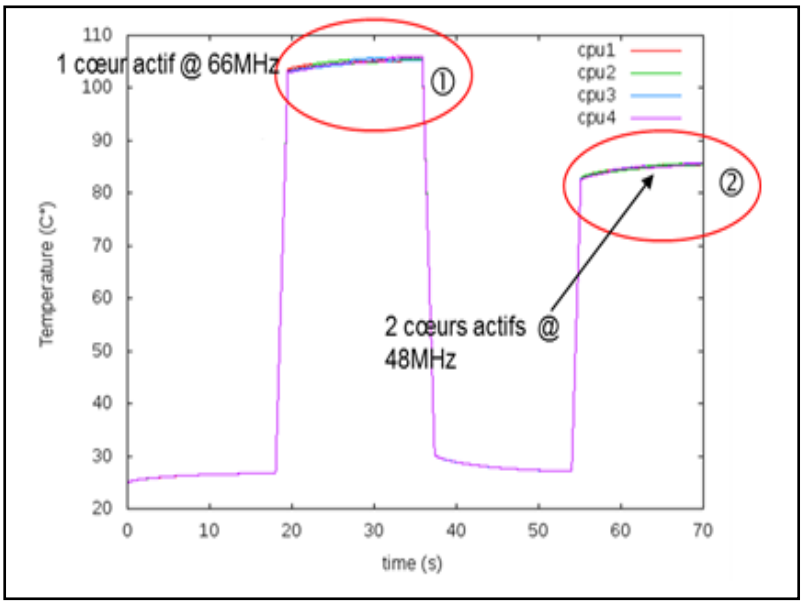


Figure 5.19 : Changement périodique de configuration

En effectuant un zoom sur les intervalles ① et ② de la Figure 5.19 on observe (sur la Figure 5.20) qu'il y a effectivement permutation du processeur actif dans la zone ① et que dans la zone ② deux processeurs sont périodiquement concernés par les changements d'états actif/inactif.

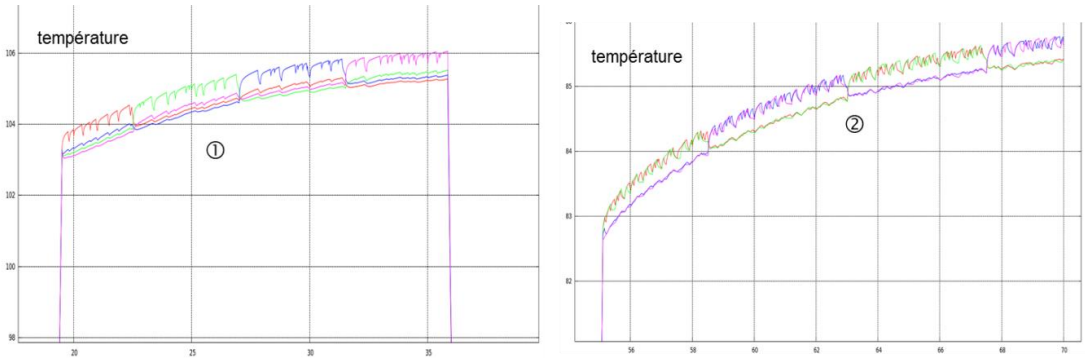


Figure 5.20 : Détail des variations en température des intervalles ① et ② de la

En effectuant ces changements d'états on obtient la répartition des cycles thermiques illustrée sur la Figure 5.21. On remarque que cette technique de permutation permet ainsi d'équilibrer entre les cœurs les cycles thermiques produits par l'ordonnancement des tâches. La somme des cycles thermiques subis par chaque cœur est alors de 89°C ce qui correspond à la valeur des processeurs les moins actifs de la Figure 5.17.

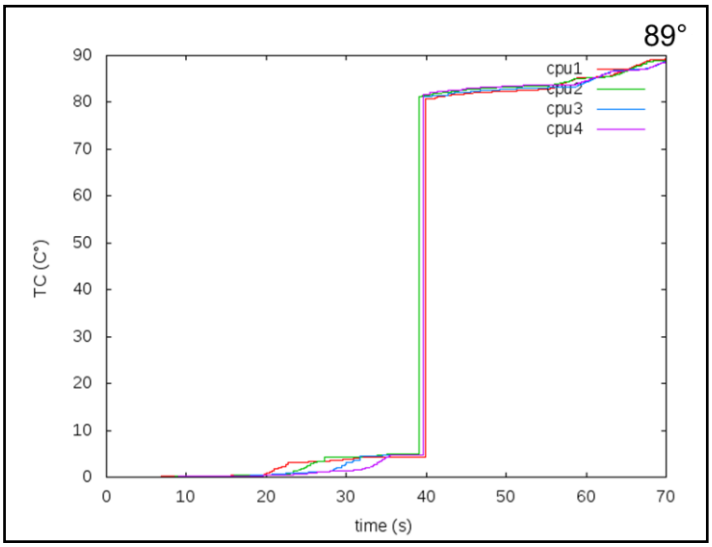


Figure 5.21 : Effet des changements périodiques d'états actifs/inactif sur les cycles thermiques

Pour illustrer l'impact de permutation entre les cœurs actifs et inactifs sur la variation des cycles thermiques, nous considérons la Figure 5.22 qui montre pour les trois algorithmes globaux EDF, RM et EDZL le nombre maximal de cycles thermiques subis par chaque cœur. Les figures de droite utilisent la technique proposée de permutation respectivement pour chaque algorithme et celles de gauche n'utilisent pas cette technique, et ce, pour les différentes charges de processeur soit 60%, 80% et 90%.

On observe une similitude des résultats entre les trois algorithmes vis-à-vis du nombre de cycles thermiques. Lorsque la technique de permutation n'est pas appliquée, le CPU1 atteint 116° de CT alors que les CPU2, CPU3 et CPU4 atteignent respectivement 89°, 85° et 80° et ce pour une charge de 60%.

Ainsi un écart important d'environ 30°C existe entre le CPU1 et les autres cœurs, ce déséquilibre pourrait sur une longue période conduire à une fragilisation du circuit lié à ces cycles thermiques majoritairement centrés sur le même CPU.

Cette différence de CT entre le CPU1 et les autres cœurs est due à la sélection des cœurs à activer en situation de température élevée Ainsi dans notre cas lorsque un seul cœur est actif dans la situation à température ambiante la plus élevée (m=1), alors uniquement le CPU1 est utilisé. De plus, même avec m=4 lorsque la température ambiante est basse, le processeur ayant l'index le plus faible est celui le plus sollicité du fait de l'ordre d'affectation des cœurs aux tâches prêtes.

Avec la technique de permutation des cœurs actifs et inactifs, et une charge de 60%, nous réduisons les CT de 20% sur le CPU1 (92,4°). Les autres cœurs voient leurs cycles thermiques augmenter : CPU2 à 92,7°, CPU3 à 93,7° et CPU4 à 92°C. Ainsi l'écart total entre les cœurs est réduit à 1°C.

En augmentant la charge des processeurs, les CT diminuent ainsi que l'écart observé entre les CT des différents cœurs. Ainsi de manière triviale, on peut noter qu'avec une charge 100% (en admettant que les tâches soient ordonnançables dans ce cas) l'écart en CT sera nul entre les cœurs.

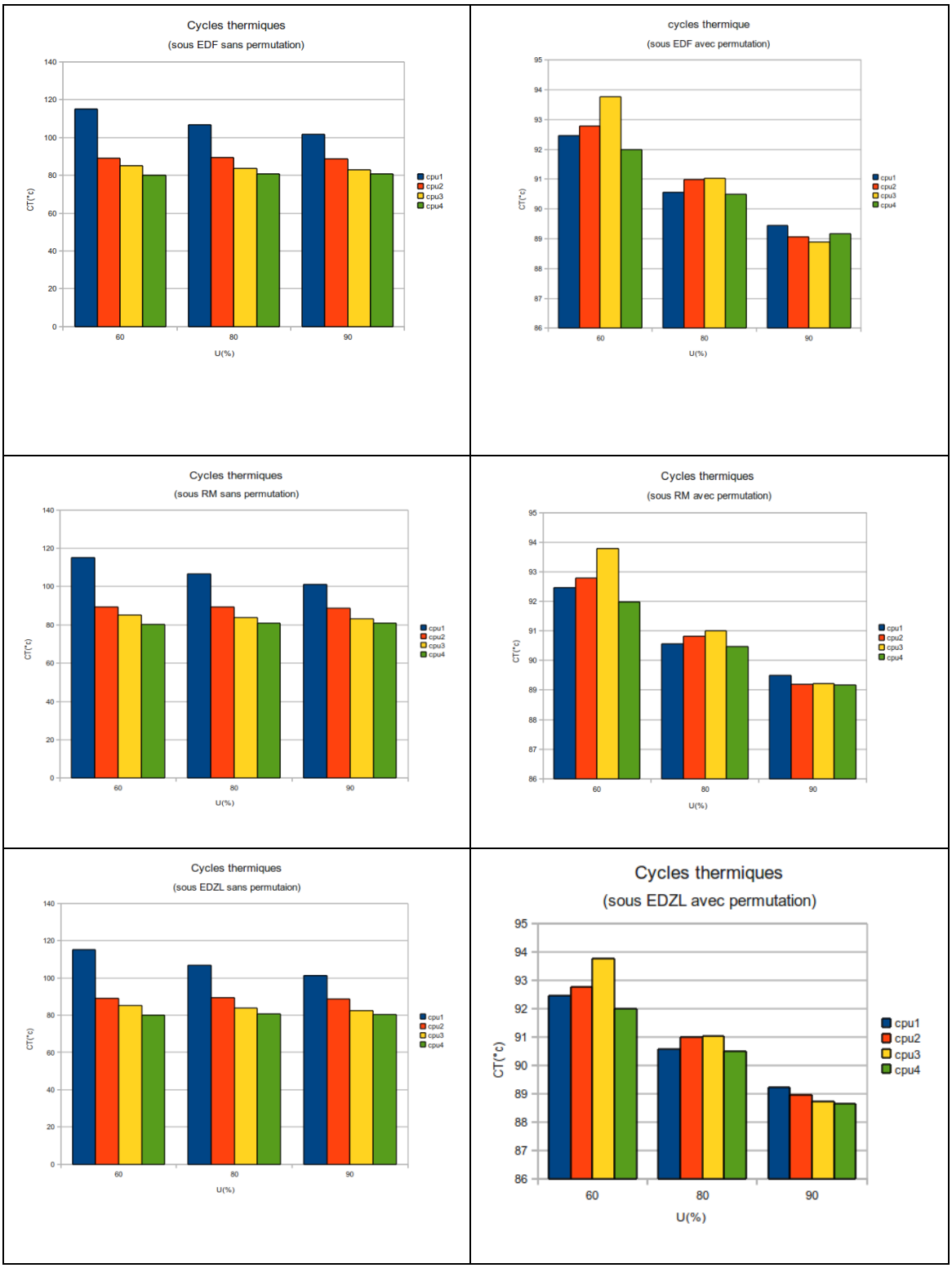


Figure 5.22 : Cycles thermiques subi par les différents cœurs suivant les 3 algorithmes globaux considérés, et sous les différentes charges processeurs.

5.5 Etude sur UltraSparc T1

Une autre étude a été menée en utilisant l'Ultrasparc T1 de Sun Microsystems [93]. Contrairement au MPC5517, ce composant a été développé dans un objectif de recherche de puissance de calcul élevée sur un spectre large d'applications. L'Ultrasparc T1 n'est pas utilisé dans des applications temps réel mais nous avons souhaité évaluer notre approche sur une architecture beaucoup plus sensible aux problèmes thermiques que le MPC5517.

La Figure 5.23 illustre le descriptif du *layout* de l'Ultrasparc T1[93]. Ce circuit contient 8 cœurs placés de façon symétrique avec 4 cœurs (cœurs 0, 2, 4, 6) en face des autres 4 cœurs (1, 3, 5,7) alignés sur chacun des bords haut et bas du composant. Ce circuit consomme 63 W à fréquence 1.2Ghz sous une tension de 1.3V. Il occupe une surface de silicium de 379 mm².

Nous considérons 3 points de fonctionnement (V, F) dans notre étude : (1,05V ; 720MHz) (1,2V ; 960MHz) (1,3V; 1200MHz). Les différents caractéristiques thermiques qui ont permis de développer le modèle de ce circuit dans ATMI ont été déterminées à partir de [94].

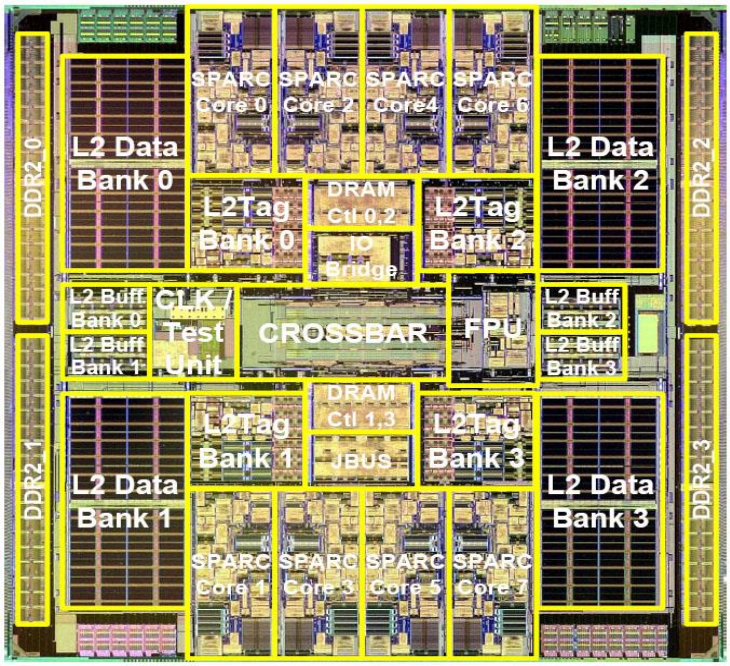


Figure 5.23 : Le *floorplan* de l'UltraSparc considéré

Nous avons considéré un scénario de variation de température ambiante illustré dans la Figure 5.24, ou la température ambiante maximale atteinte est de 50 °C. Contrairement au

MPC5517, ce type de processeur n'est pas conçu pour fonctionner à des températures ambiantes élevées

Tableau 5.3 : Caractéristiques des tâches considérées pour lUltraSparc T1

Caractéristiques / Tâches	WCET	Période
Tâche 1	19	30
Tâche 2	3	36
Tâche 3	3	40
Tâche 4	10	45
Tâche 5	9	50
Tâche 6	20	30
Tâche 7	24	36
Tâche 8	5	40
Tâche 9	39	45
Tâche 10	38	50
Tâche 11	2	30
Tâche 12	16	36

Nous avons choisi l'algorithme d'ordonnancement RM avec une charge globale des processeurs de 60% suivant 12 tâches dont les caractéristiques sont citées dans le Tableau 5.3

En utilisant la totalité des cœurs à la fréquence minimale, soit 720Mhz, on obtient l'évolution de la température présentée dans la Figure 5.25. La température maximale atteinte (au niveau jonction) dépasse les 120°C pour une température ambiante de 50°C.

Nous remarquons dans cette figure qu'avec une température ambiante de 50°C, l'état stationnaire n'est pas atteint à 35s et que la température continue à croître. Au contraire, pour une température ambiante de 25°C, la température atteint son état stable plus vite. Cette

différence est due à l'augmentation du *leackage* dans le circuit liée à la dépendance avec la température.

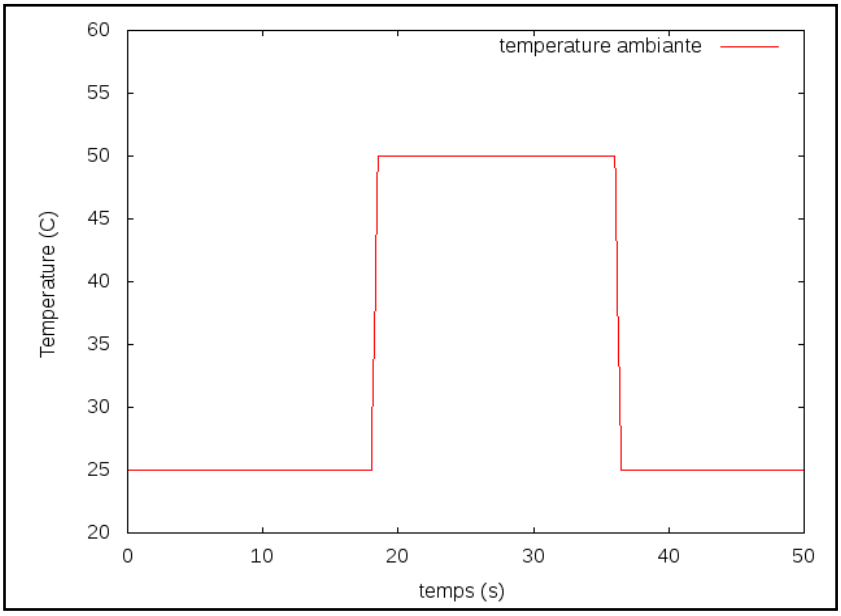


Figure 5.24 : Exemple considéré de variation de la température ambiante pour l'Ultra Sparc T1

Nous considérons aussi l'exécution avec le minimum de cœurs et à la fréquence maximale (1.2Ghz), et qui nous donne une évolution en température comme le montre la Figure 5.26 où la température maximale est de 95°C, pour une proportion de température ambiante de 50°C.

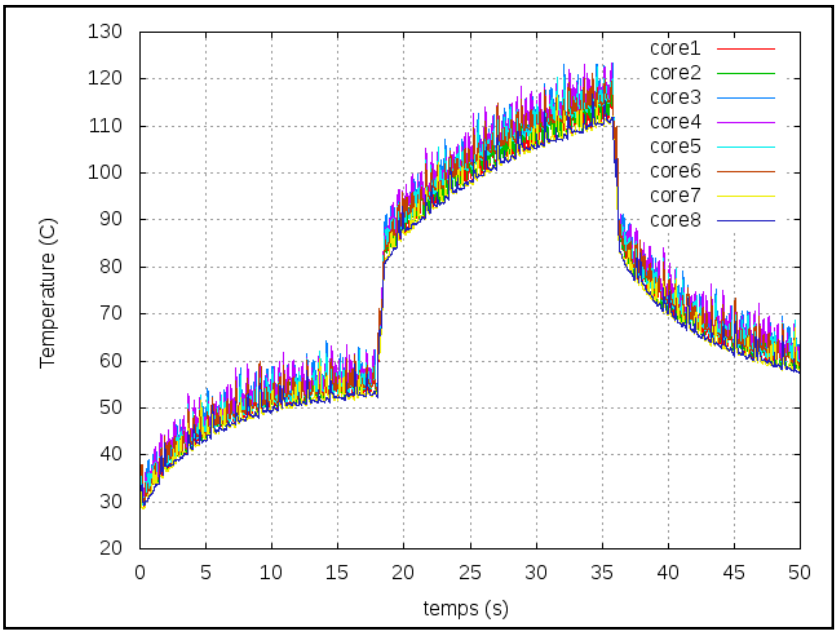


Figure 5.25 : Variation en température à 8 cœurs à 720 MHz (avec RM).

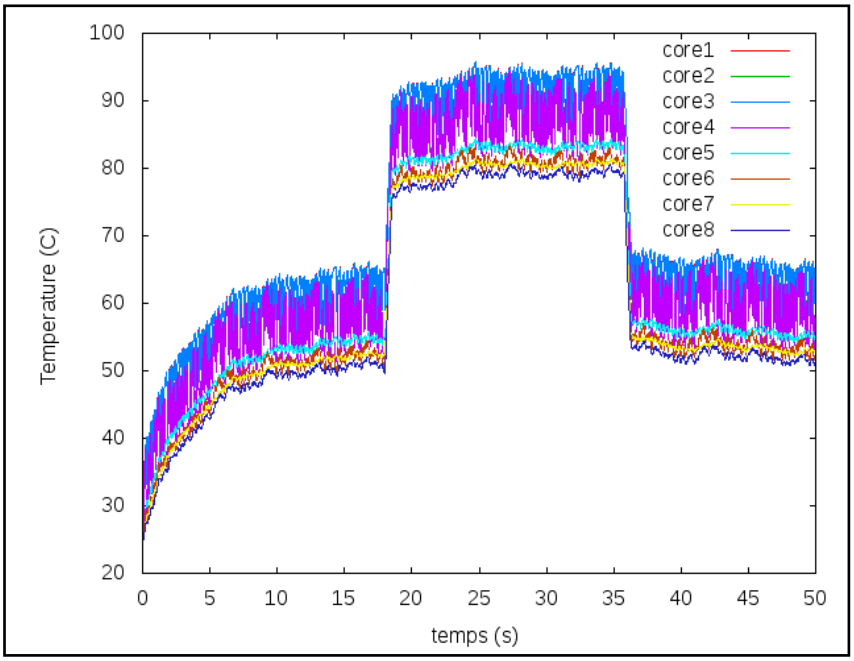


Figure 5.26 : Variation en température de l'architecture à 2 cœurs à 1200 MHz (avec RM).

En appliquant notre approche, nous obtenons les résultats de la Figure 5.27 où le changement de configuration intervient à 78°C pour passer à 2 cœurs actifs fonctionnant à 1,2GHz.

Nous observons que la température maximale est de 113°C (avec un minimum de 80°C) dans la période où la température ambiante est de 50°C, et que la température du processeur atteint son état stationnaire plus vite que dans le cas de la Figure 5.25.

Autre point intéressant, est que dans la 3ème partie où la température ambiante est de 25°C, (Figure 5.27), la température au niveau jonction retourne plus vite à l'état équilibre que la Figure 5.25. Cette différence est due au fait que lorsque la température passe de 50°C à 25°C la température des cœurs est plus faible dans le cas de la Figure 5.27 ce qui réduit les courants de fuite et donc favorise une décroissance plus rapide avec une meilleure diffusion de la chaleur à la surface du circuit. Même si la décroissance rapide en température ambiante telle qu'elle est considérée dans notre expérimentation n'est pas très réaliste, on peut cependant constater que partir d'une valeur moyenne de température jonction plus faible à température ambiante élevée va dans le sens d'une décroissance en température du silicium qui sera plus proche de celle de la température ambiante.

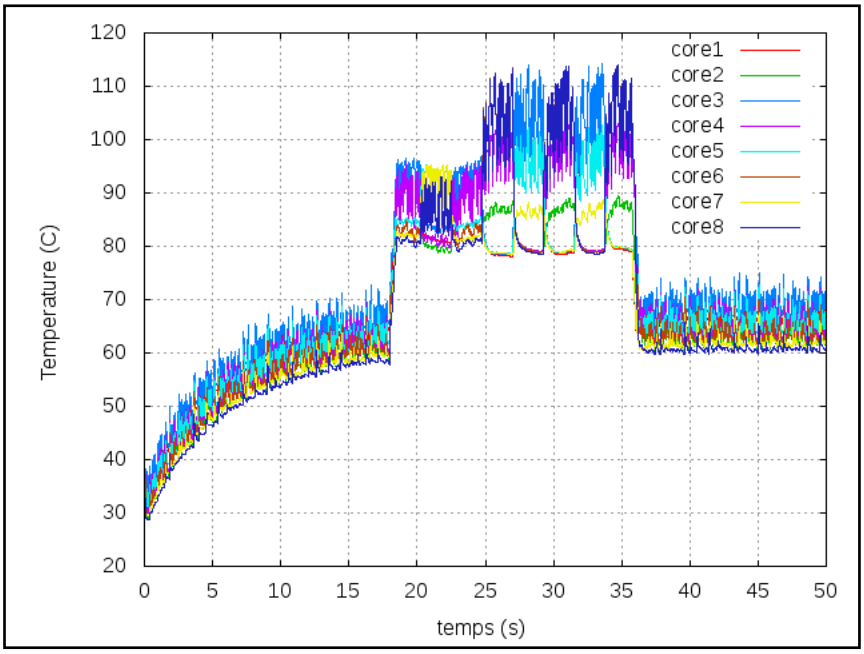


Figure 5.27 : Variation en température de l'architecture en appliquons notre approche.

Du point de vue des cycles thermiques, la Figure 5.28 illustre le cumul de ces cycles dans le temps. Ces résultats correspondent à l'exécution des tâches par les 8 cœurs à la fréquence de 720Mhz (Figure 5.25). On peut constater que le cœur core1 subit des cycles thermiques à hauteur de 1750°C.

Par contre en utilisant seulement la configuration à 2 cœurs avec une fréquence de 1200Mhz on observe sur la Figure 5.29 un cumul en cycles thermiques de 2886°C pour les core1 et core2 (les deux courbes ont superposées) (ce qui correspond à l'évolution en température de la Figure 5.26).

Dans la Figure 5.30 nous présentons les cycles thermiques subis par les cœurs dans le cas où est appliquée notre approche. Le processeur core3 (core 2 dans la Figure 5.23) subit le plus de cycles thermiques avec un total cumulé de 1700° ce qui réduit de façon significative ces cycles par rapport à la situation précédente.

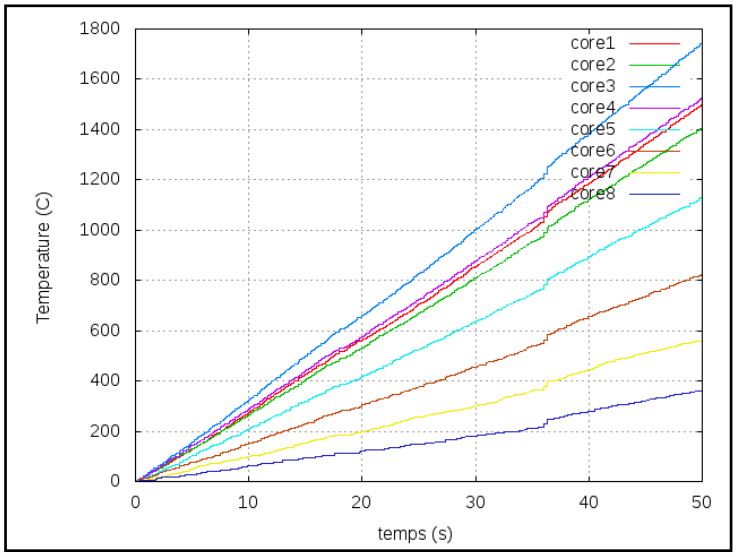


Figure 5.28 : Cycles thermiques correspondent à l'évolution en température de la Figure 5.25

L'avantage d'effectuer des permutations entre les cœurs se voit entre les instants 20s et 35s où l'algorithme 4.3 choisit (à partir de l'instant 25s) alternativement les cœurs (core1, core3) et (core6, core8). Nous avons considéré un temps entre deux permutations égal au huitième de l'hyper-période des tâches. Une autre avantage constaté, est que l'écart entre les cycles thermiques des cœurs est réduit significativement par rapport à la configuration où on ne considère que deux cœurs actifs seulement (2886 CT). Une diminution et une répartition plus homogène des cycles thermiques entre les cœurs permettent a priori d'aller dans le sens d'augmenter la fiabilité d'un système.

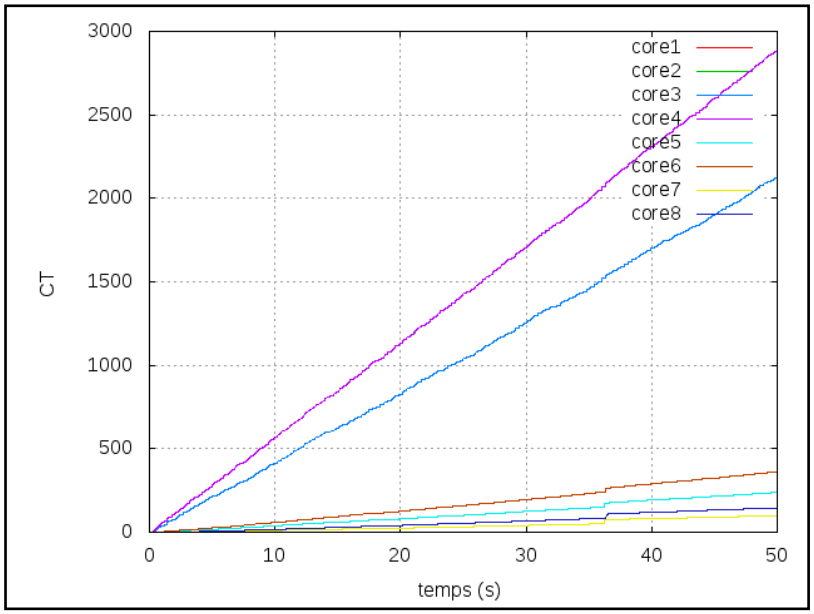


Figure 5.29 : Cycles thermiques correspondent à l'évolution en température de la Figure 5.26

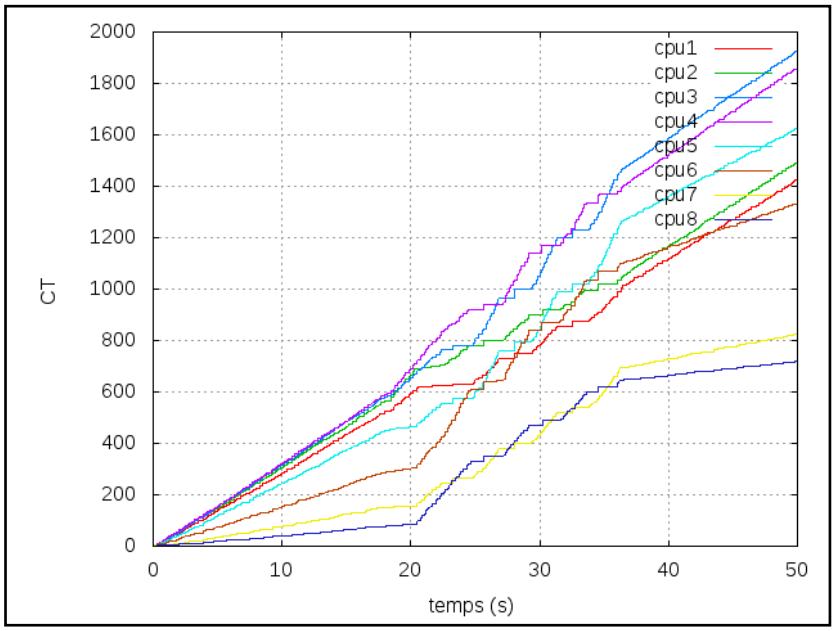


Figure 5.30 : Cycles thermiques correspond à l'évolution en température de la Figure 5.27

Si par rapport à la Figure 5.29 les résultats de la Figure 5.30 illustrent une amélioration dans la répartition des cycles thermiques sur les processeurs, il demeure une certaine disparité qui montre que des améliorations sont toujours possible. Par exemple dans notre approche le choix des processeurs actifs à chaque permutation est basé sur la température courante des cœurs de processeurs. Cette température courante ne représente pas nécessairement les cycles

thermiques effectivement subis. Pour obtenir une répartition beaucoup plus homogène des cycles thermiques une solution consisterait à comptabiliser à l'exécution et pour chaque cœurs ces cycles. Cependant, ceci suppose un échantillonnage continu des valeurs des températures fournies par les capteurs et une détection des dérivées nulles des températures suivant une période de l'ordre de quelques millisecondes ou dizaines de millisecondes ce qui peut dans certains cas induire un *overhead* non négligeable vis-à-vis des caractéristiques des tâches à exécuter.

5.6 Conclusion

Dans ce chapitre nous étudié les résultats des deux systèmes MPC5517 étendu à 4 cœurs et Ultrasparc T1 en appliquant notre approche de sélection des configurations couplée à une technique de permutations entre cœurs pour réduire et équilibrer les cycles thermiques subis par les cœurs d'une architecture sur laquelle s'exécute un ensemble de tâches ordonnancées suivant un algorithme d'ordonnancement global.

Les résultats obtenus, par rapport à une exécution qui n'utilise pas de DVFS ni de DPM, montrent un gain sur le nombre de cycles thermiques et une réduction de la température maximale atteinte dans le cas d'une température ambiante élevée. Diminuer la fréquence des processeurs permet de diminuer les cycles thermiques, quelle que soit la valeur de la température ambiante mais si la température ambiante croît de manière importante la température au niveau silicium peut vite devenir prohibitive du fait des courants de fuite, ce qui n'est pas souhaité de point de vue fiabilité.

Aussi, notre technique constitue un bon compromis entre l'objectif de réduction des cycles thermiques et de contrôle de la temprature maximale du circuit dans des situations de variations importantes de température ambiante, et ce tout en respectant les échéances des tâches.

CHAPITRE 6. Conclusion générale et perspectives

6.1 Conclusion générale

Dans ce manuscrit nous avons abordé le problème de la réduction des cycles thermiques subis par les cœurs d'architectures multiprocesseurs utilisées dans des applications temps réel embarquées. Les cycles thermiques ont un effet négatif sur la fiabilité des systèmes électroniques et cet effet est d'autant plus marqué que la technologie des transistors utilisée est avancée et que les conditions d'utilisation des circuits les expose à des températures ambiantes pouvant être élevées. Etudier simultanément les cycles thermiques et le caractère temps réel d'une exécution de tâches sur une architecture multiprocesseur impose de modéliser les deux comportements : évolution de la température dans le temps et ordonnancement des tâches sur les processeurs. Ainsi une part importante de l'étude a consisté dans un premier temps à se doter d'un environnement de co-simulation permettant de construire conjointement ces deux comportements afin d'analyser à la fois le caractère temps réel de l'exécution et les cycles thermiques subis par les processeurs. Cet environnement de co-simulation a été construit sur la base de deux simulateurs : le simulateur STORM développé à l'IRCyNN relatif aux aspects fonctionnels et le simulateur ATMI développé à l'IRISA pour les aspects thermiques. Le domaine d'étude abordé est relatif à l'automobile qui fait appel à de nombreux systèmes électroniques pouvant être placés dans des conditions de température souvent sévères. Ainsi nous avons proposé un modèle thermique du circuit MPC5517 de la société Freescale souvent utilisé dans l'électronique embarquée pour l'automobile. . Ce modèle opère à un degré de granularité relativement élevé. Il a été validé par une série de mesures effectuées par caméra thermique sur le circuit décapsulé. Une difficulté rencontrée est qu'il n'a pas été possible d'identifier à un niveau plus fin que le niveau cœur de processeur les éléments qui interviennent dans la dissipation de puissance et de chaleur du circuit. Ceci est essentiellement dû à la fois à l'absence d'information sur le *layout* du circuit et au comportement thermique du circuit qui dissipe peu de puissance sur une surface assez homogène. En particulier, les différences de température observées à la surface du silicium sont assez faibles ce qui n'a pas permis d'identifier des blocs plus fins que les cœurs de processeur.

Toutefois nous avons pu dériver des modèles thermiques sous ATMI (dynamique de température) qui donnent des résultats satisfaisants par rapport aux observations faites, par la caméra thermique et le simulateur STIMuL (évaluation de la température du circuit à l'état stable), et en accord avec les *datasheets* du constructeur.

Ensuite nous avons réalisé le couplage entre les deux outils STORM et ATMI pour définir un environnement de co-simulation fonctionnel-thermique qui opère au niveau tâche. Du côté STORM nous avons développé un contrôleur thermique qui peut adapter à chaque événement d'ordonnancement significatif (avec changement d'état) la configuration (nombre de cœurs de processeur actifs et fréquence de fonctionnement des cœurs selon la température mesurée) utilisée par l'algorithme d'ordonnancement. La contrainte est que l'algorithme d'ordonnancement doit utiliser une politique globale d'ordonnancement et permettre la migration de tâches entre les processeurs. Ceci impose de considérer uniquement le cas d'architectures à processeurs homogènes.

Nous avons proposé une méthode pour déterminer ces configurations qui permettent à la fois de réduire la puissance dissipée (puissance dynamique et puissance statique) en tenant compte de la température des cœurs et de réduire les temps inactifs des processeurs qui engendrent en particulier les cycles thermiques.

Ces configurations ainsi déterminées, permet de garantir le respect des échéances des tâches par simulation dans le but d'optimiser la réduction des cycles thermiques.

Enfin nous avons proposé une approche pour réaliser le contrôleur thermique qui s'appuie sur la possibilité de faire varier la fréquence de fonctionnement en fonction de la température mesurée. Ce contrôleur sélectionne alors la configuration déterminée hors ligne qui correspond à la valeur de température mesurée des cœurs. A température ambiante élevée (qui correspond à une configuration avec un nombre minimal de cœurs actifs par rapport à la charge de travail), une rotation est opérée entre les processeurs actifs et les processeurs en mode repos afin de limiter les gradients de température entre processeurs sur le circuit.

Ce contrôleur thermique est a priori compatible avec tout algorithme d'ordonnancement qui effectue un ordonnancement global des tâches et permet la migration des tâches. Cette propriété a été illustrée sur les trois algorithmes globaux G-EDF, G-RM et G-EDZL.

Les résultats obtenus montrent que les techniques proposées permettent de réduire très efficacement les cycles thermiques subis par les processeurs d'une architecture multicoeur, en tenant compte des variations de la température ambiante. On peut noter que cette température ambiante est une contrainte souvent considérée comme constante dans la majorité des travaux de recherches.

Afin de valider l'étude sur une autre architecture matérielle que celle du MPC5517, nous avons aussi considéré le cas de l'UltraSparc T1 capable de fournir des puissances de calcul élevées. Ce circuit n'est pas a priori prévu pour des applications temps réel mais nous avons souhaité évaluer le comportement du contrôleur thermique développé pour une architecture révélant des dynamiques de températures plus importantes.

La technique proposée de réduction des cycles thermiques s'appuie sur la possibilité de faire varier la fréquence de fonctionnement en fonction de la température mesurée. Dans le cas où la charge induite par les tâches est proche du nombre de processeurs m de l'architecture il n'est pas possible d'appliquer ce type d'approche. Ceci implique que pour fonctionner à fréquence élevée et dans une gamme de température ambiante large, il est sans doute nécessaire d'équiper le circuit avec un système de refroidissement afin de limiter la température au niveau jonction à une valeur inférieure à la contrainte fixée par le constructeur. On peut alors se poser la question de l'intérêt de considérer plutôt un circuit ayant un nombre plus important de cœurs car le prix du silicium pour répliquer un cœur de processeur peut s'avérer moins élevé que celui relatif à placer un radiateur (éventuellement avec ventilation) sur le circuit. Aussi la méthode proposée pourrait être étendue pour permettre de déduire le nombre de cœurs à considérer dans une architecture afin d'obtenir un système sans radiateur satisfaisant les contraintes de temps réel et de fiabilité compte-tenu des exigences d'environnement du système (e.g. système confiné, valeur de température ambiante).

6.2 Perspectives

Le travail réalisé dans cette étude ouvre différentes perspectives dont certaines sont présentées ci-dessous.

6.2.1 Autres modèles de défaillances

Les cycles thermiques dans les systèmes embarqués sont liés à l'augmentation de la densité des transistors par puce en raison de la miniaturisation. L'augmentation de la densité des transistors sur la puce et les exigences de calcul à haute vitesse conduisent à l'augmentation de la dissipation de chaleur dans les systèmes multiprocesseurs. Cette augmentation de la température provoque plusieurs problèmes de défaillance, autres que celui dû aux cycles thermiques. Comme mentionné dans la section 2.3.1.1, plusieurs autres modèles de

défaillances existent dans la littérature qui dépendent directement de la température ou des variations de cette température, il s'agit par exemple de l'électromigration ou le phénomène de *Time Dependent Dielectric Breakdown*.

Ce serait certainement une contribution intéressante et utile d'étudier ces mécanismes de près vis-à-vis de l'ordonnancement des tâches sur les processeurs et d'élaborer une technique afin d'améliorer la fiabilité globale du système.

6.2.2 Impact sur la fiabilité

La majorité des travaux existants qui considèrent les aspects fiabilité pour des systèmes temps réel, visent à réduire principalement le pic de température dans le but d'augmenter la fiabilité et ce à travers des techniques comme la migration de tâches ou le DVFS.

Cependant, comme il est montré dans [36] optimiser la consommation de puissance peut avoir des impacts négatifs sur la fiabilité, aussi il serait judicieux de compléter l'analyse de l'ordonnancement et l'analyse thermique présentées dans notre étude par une étude réelle de fiabilité. Ceci pourrait être effectué en intégrant par exemple l'outil RAMP [5] afin de quantifier l'apport d'une telle stratégie sur la fiabilité de l'ensemble du système.

6.2.3 Etude du *Floorplaning*

Dans la section 4.3.3 nous avons décrit une technique de permutation entre les cœurs afin de limiter les gradients en température (des cycles thermiques spatiaux).

Ces gradients dépendent fortement de l'emplacement des cœurs (ainsi des autres blocs du chip comme la mémoire) sur le die comme de leurs activités ainsi leurs interdépendances. Par exemple les cycles thermiques spatiaux obtenus avec 4 cœurs placés sur les 4 coins du *die*, ne sont pas les mêmes cycles (ou même comportement thermique) obtenus en les plaçant au centre du die.

Alors une étude thermique sur le floorplanning au niveau de la conception en considérant les différentes combinaisons de placement des cœurs sur le *die* serait très utile et augmente la fiabilité du chip.

6.2.4 Les processeurs 3D

Actuellement des études se concentrent sur la conception d'architectures de plate-formes multicoeurs exploitant la technologie en devenir 3D [95] [96] et ce afin de satisfaire aux exigences de calcul imposées par l'évolution des applications. Du fait de la miniaturisation, la surface disponible sur puce pour la dissipation thermique est réduite ce qui se traduit par une augmentation de la densité de puissance dissipée. En technologie 3D la problématique est exacerbée du fait de la concentration dans un volume réduit de la capacité de calcul et des fonctions mémoire et communication.

Une modélisation thermique de ce type de structure ainsi que l'étude de l'impact des ordonnanceurs (l'affectation des tâches aux cœurs) sur l'évolution de la température du système en considérant l'ensemble des facteurs qui agissent sur la fiabilité (dont les cycles thermiques spatiaux et temporels) seraient intéressante.

Bibliographie

[1] S. Irani and K. R. Pruhs, “Algorithmic problems in power management,” *SIGACT News*, vol. 36, no. 2, pp. 63–76, juin 2005.

[2] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture,” in *In Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2003, pp. 2–13.

[3] C. B, “Enabling Exascale Programming: A System-Wide Challenge,” Paris, Mar. 2013.

[4] Y.-W. Wu, C.-L. Yang, P.-H. Yuh, and Y.-W. Chang, “Joint exploration of architectural and physical design spaces with thermal consideration,” in *Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005. ISLPED ’05*, 2005, pp. 123–126.

[5] S. J, A. S, B. P, R. J, and H. C, “RAMP: A Model for Reliability Aware MicroProcessor Design,” IBM Research Report, RC23048 (W0312- 122), Dec. 2003.

[6] T. S. Rosing, K. Mihic, and G. De Micheli, “Power and reliability management of SoCs,” *IEEE Trans Very Large Scale Integr Syst*, vol. 15, no. 4, pp. 391–403, avril 2007.

[7] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, “Lifetime Reliability: Toward an Architectural Solution,” *IEEE Micro*, vol. 25, no. 3, pp. 70–80, 2005.

[8] V. N. A. NAIKAN, *RELIABILITY ENGINEERING AND LIFE TESTING*. PHI Learning Pvt. Ltd., 2008.

[9] J. A. Stankovic, “Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems,” *Computer*, vol. 21, no. 10, pp. 10–19, Oct. 1988.

[10] C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” *J ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[11] D. C. Blest and D. G. Fitzgerald, “Scheduling sports competitions with a given distribution of times,” *Discrete Appl. Math.*, vol. 22, no. 1, pp. 9–19, décembre 1988.

[12] A. K. Mok, “FUNDAMENTAL DESIGN PROBLEMS OF DISTRIBUTED SYSTEMS FOR THE HARD-REAL-TIME ENVIRONMENT,” Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[13] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion, “Multi-source and multicore automotive ECUs - OS protection mechanisms and scheduling,” in *2010 IEEE International Symposium on Industrial Electronics (ISIE)*, 2010, pp. 3734–3741.

[14] T. P. Baker, “An Analysis of Fixed-Priority Schedulability on a Multiprocessor,” *Real-Time Syst*, vol. 32, no. 1–2, pp. 49–71, février 2006.

[15] T. P. Baker, “Multiprocessor EDF and deadline monotonic schedulability analysis,” in *24th IEEE Real-Time Systems Symposium, 2003. RTSS 2003*, 2003, pp. 120–129.

[16] B. Andersson, S. Baruah, and J. Jonsson, “Static-priority scheduling on multiprocessors,” in *22nd IEEE Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings*, 2001, pp. 193–202.

[17] M. Cirinei and T. P. Baker, “EDZL Scheduling Analysis,” in *19th Euromicro Conference on Real-Time Systems, 2007. ECRTS ’07*, 2007, pp. 9–18.

[18] R. I. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Comput Surv*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011.

[19] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, “Proportionate progress: A notion of fairness in resource allocation,” *Algorithmica*, vol. 15, no. 6, pp. 600–625, Jun. 1996.

[20] D. Zhu, D. Mosse, and R. Melhem, “Multiple-resource periodic scheduling problem: how much fairness is necessary?,” in *24th IEEE Real-Time Systems Symposium, 2003. RTSS 2003*, 2003, pp. 142–151.
[21] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, “RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor,” in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, 2011, pp. 104–115.
[22] G. Nelissen, V. Berten, V. Nelis, J. Goossens, and D. Milojevic, “U-EDF: An Unfair But Optimal Multiprocessor Scheduling Algorithm for Sporadic Tasks,” in *2012 24th Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 13–23.
[23] 167918, K. Mihic, G. De Micheli, and T. Simunic, “Optimization of Reliability and Power Consumption in Systems on a Chip,” *Proc. Int. Work. Power Timing Model. Optim. Simul. PATMOS*, pp. 237–246, 2005.
[24] JEP122-B, “Failure Mechanisms and Models for Semiconductor Devices.” JEDEC standard, Aug-2003.
[25] C. McLaren, J. Null, and J. Quinn, “Heat Stress From Enclosed Vehicles: Moderate Ambient Temperatures Cause Significant Temperature Rise in Enclosed Vehicles,” *Pediatrics*, vol. 116, no. 1, pp. e109–e112, Jul. 2005.
[26] D. I.R, A. I, F. N.D, K. G., and V. K., “Temperature Variations in a Parked Car,” 2009.
[27] M. Russell and E. John, “Temperature in cars Survy.” Feb-2009.
[28] M. Bach, *How Ambient Temperatures Affect Your PC*. 2012.
[29] F. Fallah and M. Pedram, “Standby and Active Leakage Current Control and Minimization in CMOS VLSI Circuits,” *IEICE Trans.*, vol. 88–C, no. 4, pp. 509–519, 2005.
[30] M. L. Mui, K. Banerjee, and A. Mehrotra, “Power supply optimization in sub-130 nm leakage dominant technologies,” in *5th International Symposium on Quality Electronic Design, 2004. Proceedings*, 2004, pp. 409–414.
[31] “Int’l Technology Roadmap for Semiconductors,” ITRS 2004.
[32] N. Bansal and K. Pruhs, “Speed scaling to manage temperature,” in *Proceedings of the 22nd annual conference on Theoretical Aspects of Computer Science*, Berlin, Heidelberg, 2005, pp. 460–471.
[33] D. Brooks and M. Martonosi, “Dynamic thermal management for high-performance microprocessors,” in *The Seventh International Symposium on High-Performance Computer Architecture, 2001. HPCA*, 2001, pp. 171–182.
[34] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas, “A framework for dynamic energy efficiency and temperature management,” in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, New York, NY, USA, 2000, pp. 202–213.
[35] A. K. Coskun, T. S. Rosing, K. Mihic, G. De Micheli, and Y. Leblebici, “Analysis and Optimization of MPSoC Reliability,” *J. Low Power Electron.*, vol. 2, no. 1, pp. 56–69, Apr. 2006.
[36] J. Haase, M. Damm, D. Hauser, and K. Waldschmidt, “Reliability-Aware Power Management of Multi-Core Processors,” in *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, B. Kleinjohann, L. Kleinjohann, R. J. Machado, C. E. Pereira, and P. S. Thiagarajan, Eds. Springer US, 2006, pp. 205–214.
[37] A. K. Coskun, R. Strong, D. M. Tullsen, and T. Simunic Rosing, “Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors,” *SIGMETRICS Perform Eval Rev*, vol. 37, no. 1, pp. 169–180, juin 2009.
[38] S. Carta, A. Acquaviva, P. G. Del Valle, D. Atienza, G. De Micheli, F. Rincon, L. Benini, and J. M. Mendias, “Multi-processor operating system emulation framework

with thermal feedback for systems-on-chip," in *Proceedings of the 17th ACM Great Lakes symposium on VLSI*, New York, NY, USA, 2007, pp. 311–316.

[39] F. Zanini, D. Atienza, L. Benini, and G. De Micheli, "Multicore thermal management with model predictive control," in *European Conference on Circuit Theory and Design, 2009. ECCTD 2009*, 2009, pp. 711–714.

[40] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Proactive temperature management in MPSoCs," in *Proceedings of the 13th international symposium on Low power electronics and design*, New York, NY, USA, 2008, pp. 165–170.

[41] R. Mukherjee, S. O. Memik, and G. Memik, "Temperature-aware resource allocation and binding in high-level synthesis," in *Proceedings of the 42nd annual Design Automation Conference*, New York, NY, USA, 2005, pp. 196–201.

[42] T. Kim and P. Lim, "Thermal-aware high-level synthesis based on network flow method," in *Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference*, 2006, pp. 124–129.

[43] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: a power-performance simulator for interconnection networks," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, Los Alamitos, CA, USA, 2002, pp. 294–305.

[44] J.-J. Chen, S. Wang, and L. Thiele, "Proactive Speed Scheduling for Real-Time Tasks under Thermal Constraints," in *Proceedings of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, Washington, DC, USA, 2009, pp. 141–150.

[45] R. Rao, S. Vrudhula, C. Chakrabarti, and N. Chang, "An optimal analytical solution for processor speed control with thermal constraints," in *Proceedings of the 2006 international symposium on Low power electronics and design*, New York, NY, USA, 2006, pp. 292–297.

[46] R. Rao and S. Vrudhula, "Performance optimal processor throttling under thermal constraints," in *Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*, New York, NY, USA, 2007, pp. 257–266.

[47] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. De Micheli, and R. Gupta, "Processor Speed Control With Thermal Constraints," *IEEE Trans. Circuits Syst. Regul. Pap.*, vol. 56, no. 9, pp. 1994–2008, 2009.

[48] G. Quan, Y. Zhang, W. Wiles, and P. Pei, "Guaranteed scheduling for repetitive hard real-time tasks under the maximal temperature constraint," in *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, New York, NY, USA, 2008, pp. 267–272.

[49] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," in *45th Annual IEEE Symposium on Foundations of Computer Science, 2004. Proceedings*, 2004, pp. 520–529.

[50] R. Jayaseelan and T. Mitra, "Temperature aware Task Sequencing and Voltage Scaling," 2008.

[51] P. Gai, L. Abeni, and G. Buttazzo, "Multiprocessor DSP scheduling in system-on-a-chip architectures," in *14th Euromicro Conference on Real-Time Systems, 2002. Proceedings*, 2002, pp. 231–238.

[52] K. Bhatti, C. Belleudy, and M. Auguin, "Power Management in Real Time Embedded Systems through Online and Adaptive Interplay of DPM and DVFS Policies," in *2010 IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC)*, 2010, pp. 184–191.

[53] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli, “Temperature control of high-performance multi-core platforms using convex optimization,” in *Proceedings of the conference on Design, automation and test in Europe*, New York, NY, USA, 2008, pp. 110–115.

[54] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. D. Micheli, “Temperature-aware processor frequency assignment for MPSoCs using convex optimization,” in *In Proc. of CODES/ISSS ’07*, 2007.

[55] W.-L. Hung, Y. Xie, N. ViJ’aykrishnan, M. Kandemir, and M. J. Irwin, “Thermal-aware task allocation and scheduling for embedded systems,” in *Design, Automation and Test in Europe, 2005. Proceedings*, 2005, pp. 898–899 Vol. 2.

[56] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, “Thermal-Aware Global Real-Time Scheduling on Multicore Systems,” in *15th IEEE Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009*, 2009, pp. 131–140.

[57] T. Chantem, X. S. Hu, and R. P. Dick, “Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs,” *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 19, no. 10, pp. 1884–1897, 2011.

[58] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-Aware Computer Systems: Opportunities and Challenges,” *IEEE Micro*, vol. 23, no. 6, pp. 52–61, 2003.

[59] D. Atienza, P. G. Del Valle, G. Paci, F. Poletti, L. Benini, G. D. Micheli, J. M. Mendias, and R. Hermida, “HW-SW emulation framework for temperature-aware design in MPSoCs,” *ACM Trans Autom Electron Syst*, vol. 12, no. 3, pp. 26:1–26:26, mai 2008.

[60] F. Zanini, D. Atienza, A. K. Coskun, and G. De Micheli, “Optimal multi-processor SoC thermal simulation via adaptive differential equation solvers,” in *2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*, 2009, pp. 139–146.

[61] R. Rao and S. Vrudhula, “Fast and Accurate Prediction of the Steady-State Throughput of Multicore Processors Under Thermal Constraints,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1559–1572, 2009.

[62] J. Nayfach-Battilana and J. Renau, “SOI, interconnect, package, and mainboard thermal characterization,” in *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, New York, NY, USA, 2009, pp. 327–330.

[63] T. Kemper, Y. Zhang, Z. Bian, and A. Shakouri, “Ultrafast Temperature Profile Calculation in Ic Chips,” Sep. 2007.

[64] V. M. Heriz, J.-H. Park, T. Kemper, S.-M. Kang, and A. Shakouri, “Method of images for the fast calculation of temperature distributions in packaged VLSI chips,” in *13th International Workshop on Thermal Investigation of ICs and Systems, 2007. THERMINIC 2007*, 2007, pp. 18–25.

[65] P. Michaud and Y. Sazeides, “ATMI: Analytical Model of Temperature in Microprocessors,” in *Third Annual Workshop on Modeling, Benchmarking and Simulation (MoBS),*, 2007.

[66] *acethermalmodeler http://www.doceapower.com/products-services/acethermalmodeler.html*. DoceaPower.

[67] A. Ziabari, E. K. Ardestani, J. Renau, and A. Shakouri, “Fast thermal simulators for architecture level integrated circuit design,” in *2011 27th Annual IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, 2011, pp. 70–75.

[68] M. Bao, A. Andrei, P. Eles, and Z. Peng, “Temperature-aware voltage selection for energy optimization,” in *Proceedings of the conference on Design, automation and test in Europe*, New York, NY, USA, 2008, pp. 1083–1086.

[69] R. M. C and V. Pangracious, “Thermal analysis &;optimization of a 3 dimensional heterogeneous structure,” *CoRR*, vol. abs/1203.1799, 2012.
[70] D. Fetis, “An evaluation of hotspot-3.0 block-based temperature model,” in *In Proc. 2006 WDDD, in conjuction with ISCA 2006*, 2006.
[71] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, “Cheddar: a flexible real time scheduling framework,” *Ada Lett*, vol. XXIV, no. 4, pp. 1–8, Nov. 2004.
[72] R. Urunuela, A. Déplanche, and Y. Trinquet, “STORM a simulation tool for real-time multiprocessor scheduling evaluation,” in *2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2010, pp. 1–8.
[73] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, “TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems,” in *Formal Modeling and Analysis of Timed Systems*, K. G. Larsen and P. Niebert, Eds. Springer Berlin Heidelberg, 2004, pp. 60–72.
[74] P. Sucha, M. Kutil, M. Sojka, and Z. Hanzalek, “TORSCHE Scheduling Toolbox for Matlab,” presented at the IEEE Computer Aided Control Systems Design Symposium (CACSD’06), Munich, Germany, 2006, pp. 1181–1186.
[75] F. Golatowski, J. Hildebrandt, J. Blumenthal, and D. Timmermann, “Framework for validation, test and analysis of real-time scheduling algorithms and scheduler implementations,” in *13th IEEE International Workshop on Rapid System Prototyping, 2002. Proceedings*, 2002, pp. 146–152.
[76] K. G. Larsen, P. Pettersson, and W. Yi, “Uppaal in a nutshell,” *Int. J. Softw. Tools Technol. Transf.*, vol. 1, no. 1–2, pp. 134–152, Dec. 1997.
[77] P. Michaud, “STiMuL: a Software for Modeling Steady-State Temperature in Multilayers - Description and user manual,” INRIA, Rapport Technique RT-0385, Apr. 2010.
[78] W. Liao, L. He, and K. M. Lepak, “Temperature and supply Voltage aware performance and power modeling at microarchitecture level,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 7, pp. 1042–1053, 2005.
[79] F. Fallah and M. Pedram, “Standby and Active Leakage Current Control and Minimization in CMOS VLSI Circuits,” *IEICE Trans. Electron.*, vol. E88–C, no. 4, pp. 509–519, Apr. 2005.
[80] D. Helms, E. Schmidt, and W. Nebel, “Leakage in CMOS Circuits – An Introduction,” in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, E. Macii, V. Paliouras, and O. Koufopavlou, Eds. Springer Berlin Heidelberg, 2004, pp. 17–35.
[81] Freescale Semiconductor, “Data Sheet: Technical Data, MPC5510 Rev. 3,” 2009.
[82] “Freescale : Thermal Analysis of Semiconductor Systems.” N° BasicthermalWP/Rev0-2008.
[83] B. W. Williams, *Power Electronics: Devices, Drivers, Applications, and Passive Components*. McGraw-Hill, 1992.
[84] V. Nelis, “Energy-Aware Real-Time Scheduling in Embedded Multiprocessor Systems,” Thèse de l’Université Libre de Bruxelles, 2012.
[85] K. Funaoka, A. Takeda, S. Kato, and N. Yamasaki, “Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors,” in *International Symposium on Industrial Embedded Systems, 2008. SIES 2008*, 2008, pp. 27–33.
[86] T. P. Baker, “An analysis of EDF schedulability on a multiprocessor,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 8, pp. 760–768, 2005.
[87] T. P. Baker and S. K. Baruah, “Schedulability analysis of multiprocessor sporadic task systems,” in *Handbook of Realtime and Embedded Systems*, 2007.

[88] M. Bertogna, M. Cirinei, and G. Lipari, “Improved schedulability analysis of EDF on multiprocessor platforms,” in *17th Euromicro Conference on Real-Time Systems, 2005. (ECRTS 2005). Proceedings*, 2005, pp. 209–218.
[89] J. Lee and I. Shin, “EDZL Schedulability Analysis in Real-Time Multicore Scheduling,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 7, pp. 910–916, 2013.
[90] G. Paci, P. Marchal, F. Poletti, and L. Benini, “Exploring ‘temperature-aware’ design in low-power MPSoCs,” in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, 3001 Leuven, Belgium, Belgium, 2006, pp. 838–843.
[91] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. P. Boyd, L. Benini, and G. D. Micheli, “Temperature Control of High-Performance Multi-core Platforms Using Convex Optimization,” in *DATE*, 2008, pp. 110–115.
[92] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, “Temperature-aware microarchitecture: Modeling and implementation,” *ACM Trans Arch. Code Optim*, vol. 1, no. 1, pp. 94–125, Mar. 2004.
[93] A. S. Leon, K. W. Tam, J. L. Shin, D. Weisner, and F. Schumacher, “A Power-Efficient High-Throughput 32-Thread SPARC Processor,” *IEEE J. Solid-State Circuits*, vol. 42, no. 1, pp. 7–16, 2007.
[94] A. S. Leon, J. L. Shin, K. W. Tam, W. Bryg, F. Schumacher, P. Kongetira, D. Weisner, and A. Strong, “A Power-Efficient High-Throughput 32-Thread SPARC Processor,” in *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, 2006, pp. 295–304.
[95] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, “3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling,” in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010, pp. 463–470.
[96] A. K. Coskun, D. Atienza, T. S. Rosing, T. Brunschwiler, and B. Michel, “Energy-efficient variable-flow liquid cooling in 3D stacked architectures,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 111–116.