Méthodologie d'évaluation de la sensibilité des microprocesseurs vis à vis des rayonnements cosmiques.

Sabrine Houssany

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Nanoélectronique et Nanotechnologie**

Arrêté ministériel : 7 août 2006

Présentée par

Sabrine HOUSSANY

Thèse dirigée par **Régis LEVEUGLE**

préparée au sein du **Laboratoire TIMA**
dans **l'École Doctorale EEATS**

Méthodologie d'évaluation de la sensibilité des microprocesseurs vis-à-vis des rayonnements cosmiques

Thèse soutenue publiquement le **« 13 septembre 2013 »**,
devant le jury composé de :

M. Ian O'CONNOR
Professeur, École Centrale Lyon, Président
M. Fabrice MONTEIRO
Professeur, Université de Lorraine, Rapporteur
M. Jean-Michel PORTAL
Professeur, Université d'Aix-Marseille, Rapporteur
M. Régis LEVEUGLE
Professeur, Université de Grenoble, Grenoble-INP, Directeur de Thèse
M. Florent MILLER
Ingénieur, EADS IW, Examinateur
M. Jean-Claude LAPERCHE
Ingénieur, AIRBUS, Examinateur

À mes parents et grands-parents,

Remerciements

Les travaux de thèse présentés dans ce mémoire ont été effectués au sein du centre de recherche “EADS Innovation Works”, à Suresnes, et en collaboration avec le laboratoire TIMA de l'Université de Grenoble. Je remercie vivement Monsieur Bruno Foucher responsable du département « Systèmes Électroniques », pour m’avoir accueillie dans le département.

Je témoigne toute ma reconnaissance à Monsieur Florent Miller, responsable de l'équipe "Semi-conductor and Equipment Dependability" à EADS IW. Je le remercie de l’opportunité qu’il m’a offerte d’effectuer cette thèse, et pour la confiance et le soutien qu’il m’a apportés tout au long de mes travaux.

Je remercie très sincèrement Monsieur Regis Leveugle, professeur à l'Université de Grenoble, pour avoir dirigé et encadré ces travaux de thèse, pour ses précieux conseils, et son entière disponibilité.

J’exprime tous mes remerciements à Monsieur Fabrice Monteiro, Professeur à l’Université de Metz et Monsieur Jean-Michel Portal, Professeur à l'Université de Provence, pour avoir accepté d’être rapporteur de ce mémoire.

J’adresse mon remerciement à Monsieur Ian O’Connor, pour m’avoir fait l’honneur de présider le jury de cette thèse.

Merci également à Monsieur Jean-Claude Laperche, Ingénieur à Airbus d’avoir accepté d’être membre de mon jury.

Je remercie très sincèrement l’ensemble de mes collègues qui ont grandement contribué à l’élaboration de ces travaux : Cécile, Sébastien, Thomas, Florian et Antonin. Je remercie tout particulièrement Nicolas, qui m’a fait partager ses connaissances et compétences techniques et pour son soutien.

Je remercie du fond du cœur ma famille, notamment mes parents qui m’ont soutenue tout au long de mes études et mes sœurs qui m’ont toujours encouragée. Un grand merci aussi à Nathalie.

Enfin, je remercie Jean-Philippe, pour avoir été présent à mes côtés durant cette thèse.

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

De nos jours, on utilise de plus en plus de microprocesseurs dits pris sur étagère (Commercial « Off-The-Shelf » ou COTS), dans les applications industrielles où la sûreté de fonctionnement est un facteur important. Ces microprocesseurs COTS, produits en grande série et de faible coût, ont une capacité de calcul haute performance, ce qui les rend très attractifs. Leur utilisation permet de diminuer le poids et la quantité d'équipement embarqué.

Néanmoins, estimer le taux d'erreurs des microprocesseurs COTS est devenu une tâche très difficile en raison de leur complexité et du manque d'informations disponibles sur leur microarchitecture.

Depuis quelques années, les fabricants ont considérablement augmenté la taille des mémoires internes des microprocesseurs, appelées mémoires cache, afin d'améliorer leur performance. Les mémoires cache représentent environ 90 % des cellules mémoires d'un microprocesseur et sont de ce fait un contributeur majeur du taux d'erreurs. Cependant, ces erreurs peuvent être masquées par les mécanismes de protection ou par le comportement dynamique de l'application. Ainsi, des erreurs dans les mémoires cache n'induisent pas forcément des erreurs au niveau système. Par exemple, durant l'exécution d'une application, des données erronées peuvent ne jamais être utilisées ou être remplacées avant d'être lues.

Aujourd'hui, le taux d'erreurs du microprocesseur est estimé simplement, en considérant toutes les cellules comme sensibles et sans prendre en compte le comportement dynamique de l'application. De manière globale ceci conduit donc à une surestimation du taux d'erreurs, qui n'est en général pas représentative de la sensibilité réelle.

Ainsi, il existe un besoin important de disposer d'un outil industriel capable de prédire la sensibilité dynamique d'une application avec une précision meilleure qu'une sensibilité statique.

C'est dans ce contexte que s'inscrivent ces travaux de thèse effectués au centre de recherche IW (Innovation Works) d'EADS *(European Aeronautic Defense and Space Company)* et du laboratoire Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés (TIMA). Cette thèse se focalise sur les méthodes pour prédire la sensibilité des processeurs à l'aide d'un simulateur de processeur.

Ce manuscrit est organisé en quatre parties :

Le chapitre I introduit les environnements radiatifs naturels spatiaux et atmosphériques. Il décrit les effets des particules radiatives sur les composants électroniques. Une présentation de l'architecture des microprocesseurs nous permet d'appréhender les éléments sensibles de ce composant. Ce chapitre présente aussi les principales méthodes existantes permettant de définir la sensibilité des microprocesseurs à différentes étapes de la conception d'un équipement embarqué.

Le chapitre II est dédié à la présentation de la méthodologie de prédiction du taux d'erreurs des microprocesseurs fondée sur l'analyse des mémoires cache. Cette méthode s'appuie sur une étude du comportement des mémoires cache lorsqu'une application s'exécute.

Dans le troisième chapitre, la méthodologie est dans un premier temps appliquée à un processeur dit softcore dont l'implémentation est disponible sous forme de description haut niveau. Puis une deuxième partie est consacrée à la validation de la méthodologie par différents moyens en accélérateur de particules.

Le chapitre IV présente une extension de la méthodologie pour des processeurs matériels. L'avantage de cette méthode est qu'elle peut être étendue à une cible réelle et ne nécessite pas de modèle de processeur. Elle peut être appliquée sur la plupart des microprocesseurs COTS disposant de fonctionnalités avancée telles que les registres de performance.

Chapitre I : Contexte et Etat de l'art

Les systèmes conçus pour le domaine de l'avionique et du spatial embarquent des composants électroniques de plus en plus complexes. Les milieux dans lesquels évoluent ces systèmes sont soumis à des contraintes environnementales sévères. Les composants électroniques sont soumis aux effets de l'environnement radiatif naturel, qui ont pour conséquence de perturber leur fonctionnement. Ces composants sont de plus en plus intégrés, ce qui les rend souvent plus vulnérables. De plus, en ce qui concerne les microprocesseurs, leur complexité croissante pose un problème majeur dans l'élaboration de méthodologies permettant de prédire leur sensibilité. En effet, la sensibilité des microprocesseurs dépend fortement de l'application exécutée. Dans ce chapitre, les environnements radiatifs auxquels sont soumis les microprocesseurs seront abordés, puis l'architecture générale des microprocesseurs et les effets de l'environnement radiatif sur ces composants seront présentés. Enfin, un état de l'art des méthodes d'analyse de la robustesse des processeurs présentes dans la littérature terminera ce chapitre.

I.1 Environnement radiatif naturel et effets sur les circuits intégrés

Les particules de l'environnement radiatif en interagissant avec les composants électroniques peuvent engendrer des perturbations [SCH08]. Dans cette partie, nous proposons une introduction à l'environnement radiatif naturel spatial et atmosphérique puis nous présenterons les mécanismes d'interaction physique et nous terminerons par une présentation des effets des radiations sur les circuits intégrés.

I.1.1 Description de l'environnement radiatif naturel

Cette partie décrit les principales sources de l'environnement radiatif naturel, la quantité et la nature des particules que l'on rencontre dans un environnement spatial et dans un environnement atmosphérique.

I.1.1.1 L'environnement spatial

Dans l'espace, l'électronique embarquée est soumise à un environnement radiatif naturel extrêmement contraignant. Afin de prédire la dégradation due à ces rayonnements, il est primordial de connaître les contraintes radiatives auxquelles vont être confrontés ces composants.

Les principales particules qui interagissent dans cet environnement sont les photons, les électrons, les protons ainsi que les ions (Figure I-1). L'énergie de ces particules s'exprime en électronvolt (eV).

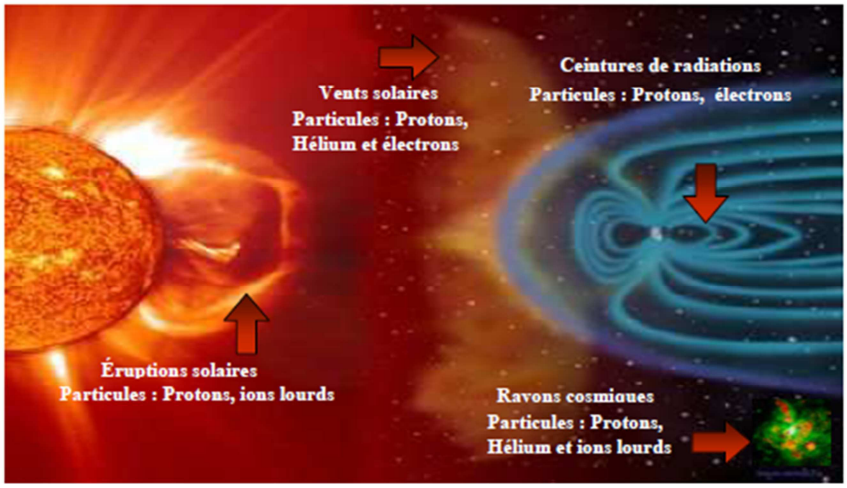


Figure I-1 : Description de l'environnement spatial [WWW04]

Il existe ainsi quatre grandes sources de contraintes radiatives :

- Les éruptions solaires consistent en une émission sporadique de particules liées à des éjections de masse coronale. Le cycle d'activité solaire dure en moyenne 11 ans dont 4 à 5 années de faible activité suivies de 6 à 7 ans de forte activité. Les éruptions solaires sont de 2 types : les éruptions solaires à protons avec un spectre d'énergie assez important pouvant atteindre jusqu'à une centaine de MeV et les éruptions à ions lourds dont l'énergie varie d'une dizaine à une centaine de MeV.

- Le vent solaire est formé d'un plasma continu résultant de l'évaporation de la couronne solaire. Sa vitesse moyenne est de l'ordre de 400 $km.s^{-1}$. Il est constitué d'électrons, de protons et de particules alpha d'énergies inférieures à 100 keV.

- Le rayonnement cosmique provient de sources très éloignées : d'origine galactique ou extragalactique, il a été découvert par Victor Hess en 1912 à partir de ballons-sondes. Il est

constitué de 83 % de protons, 13 % de particules alpha, 1 % d'ions lourds et de 3 % d'électrons [BOU95]. Ces rayonnements sont extrêmement énergétiques.

- Les ceintures de radiations, dans lesquelles les particules issues du vent solaire et des rayons cosmiques sont piégées du fait des lignes du champ magnétique terrestre, dites « ceintures de Van Allen », constituées d'une ceinture extérieure d'électrons et d'une ceinture intérieure de protons. Elles sont soumises à des perturbations causées par les variations de flux du vent solaire, ce qui les rend instables.

Le Tableau I-1 suivant synthétise les différentes particules rencontrées dans l'environnement spatial.

Tableau I-1 : Tableau de la représentation de particules dans l'espace [BOU95]

ORIGINES	PARTICULES	ENERGIES	FLUX
Ceinture de radiations	Protons Électrons	< qq 100MeV <2MeV	10 à 10^6 cm^2s^{-1} 10^{-2} à 10^7 cm^2s^{-1}
Vent solaire	Protons Électrons Particules alpha	<100KeV <qq KeV	10^8 à 10^{10} cm^2s^{-1}
Éruptions solaires	Protons Particules alpha Ions lourds	10MeV à 1GeV 10MeV à qq 100MeV	10^{10} cm^2s^{-1} ~10^2 à 10^3 cm^2s^{-1}
Rayons cosmiques	Protons (87 %) Particules alpha Ions lourds (1 %)	10^2 à 10^6 MeV Fortes énergies 1MeV à 10^{14} MeV	1 cm^2s^{-1} à 100 MeV 10^{-4} cm^2s^{-1} à 10^6 MeV

I.1.1.2 L'environnement atmosphérique

Le champ magnétique et l'atmosphère terrestre protègent la terre de la plupart des rayons cosmiques. Seules les particules du rayonnement cosmique, hautement énergétiques, peuvent traverser le champ magnétique terrestre.

L'environnement radiatif atmosphérique résulte principalement de l'interaction entre les particules issues des rayons cosmiques et les atomes de l'atmosphère (80 % d'azote et 20 % d'oxygène). Les phénomènes induits par de telles réactions donnent lieu à la production de particules secondaires : des neutrons, des protons, des muons, des pions, des électrons et des rayons gamma. Les particules secondaires générées vont elles aussi interagir avec les noyaux d'atomes de l'atmosphère, qui vont subir une cascade de nouvelles réactions avec les molécules

d'air (Figure I-2). Le flux de particules diminue en fonction de l'altitude car à l'issue de chaque interaction, une perte d'énergie importante a lieu.

Il en résulte une prédominance de neutrons aux altitudes de vol commerciaux. Au niveau du sol, on estime le flux de neutrons 300 fois moins sévère qu'à une altitude de 12 km. Des dysfonctionnements de circuits intégrés dans les systèmes avioniques, dus aux neutrons, sont observés et enregistrés à ces altitudes depuis les années 1990 [NOR93].

Les neutrons ne sont pas des particules ionisantes car ils sont électriquement neutres. Cependant, en interagissant avec la matière du composant électronique, ils peuvent engendrer des particules secondaires ionisantes. Les protons d'énergies supérieures à quelques dizaines de MeV peuvent avoir les mêmes effets que ceux des neutrons d'énergie équivalente.

Les autres particules peuvent être négligées du fait de leur faible interaction avec la matière et de leur nombre insuffisant. De plus, elles provoquent beaucoup moins d'effets dans les composants électroniques actuels [ZIE96]. En effet, l'énergie déposée par exemple par les électrons étant assez faible, ils ne peuvent pas induire de défaillance.

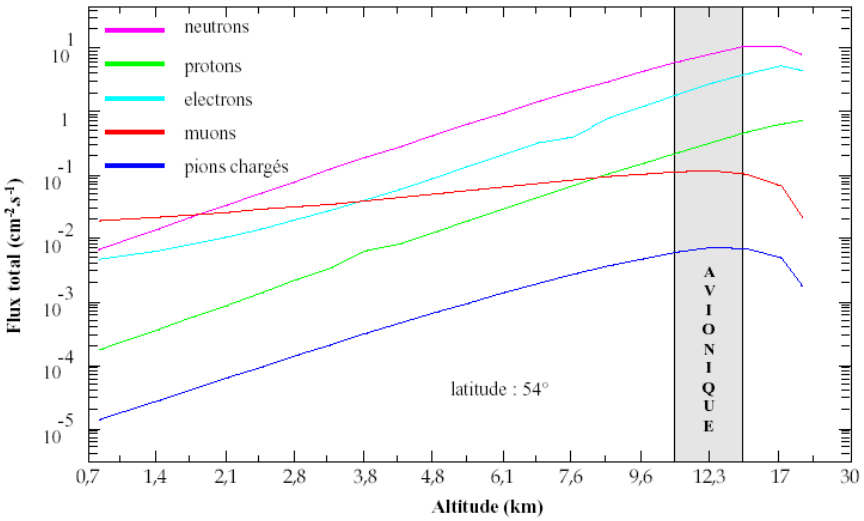


Figure I-2 : Flux total des particules se trouvant dans l'atmosphère en fonction de l'altitude, d'après [BRI71-BRI78]

I.1.2 Effets des radiations sur les composants électroniques

Les radiations peuvent causer des effets temporaires ou des dommages permanents dans les matériaux qu'elles traversent. Suivant l'interaction deux types d'effets se distinguent :

-les effets cumulés : résultent de la somme des contributions d'un grand nombre de particules.

-les effets singuliers : ce sont les effets liés à l'interaction d'une seule particule.

Les effets cumulés résultent de charges piégées dans le volume des oxydes et apparaissent après de longues périodes d'exposition à des radiations. Ce phénomène est appelé dose cumulée. Ses effets concernent plus le domaine du spatial que celui de l'avionique. Les effets singuliers appelés Single Event Effects (SEE) peuvent être provoqués par le passage d'une unique particule chargée à travers des zones sensibles du circuit. On distingue les effets réversibles, non-destructifs et les effets irréversibles, c'est-à-dire destructifs. Notre étude porte sur le domaine avionique : de ce fait, seuls les effets singuliers sont étudiés par la suite.

I.1.2.1 Les effets singuliers

On considère différents types de SEE. Parmi les effets non-destructifs, on distingue :

-Le Single Event Upset (SEU ou alea logique) est le basculement d'un point mémoire. Cet effet concerne essentiellement les composants logiques de type MOS (en particulier les mémoires SRAM et DRAM). Le point de mémoire corrompu peut en général être corrigé par une réécriture.

-Le Single Event Transient (SET) se définit par l'apparition d'un courant parasite dû au passage d'une particule dans un circuit analogique ou un circuit logique combinatoire. Ce courant transitoire peut se propager dans le circuit et entraîner des perturbations s'il est capturé par un élément mémoire de type Flip Flop.

-Les Single Event Functional Interrupt (SEFI) se caractérisent par une interruption fonctionnelle du composant. La récupération de l'état normal du circuit passe souvent par un reset complet, voire un redémarrage de l'alimentation. Cet effet concerne les processeurs, les FPGA et les mémoires haute densité.

Parmi les événements destructifs, on trouve :

-Le Single Event Latchup (SEL) qui se caractérise par le déclenchement du thyristor parasite inhérent à la structure CMOS [BRU96]. Par conséquent, il se révèle intrinsèque au composant logique.

-Le Single Event Burn-out (SEB) qui correspond à la destruction d'une cellule par échauffement thermique non contrôlé suite à un effet de transistor parasite. Il se rencontre en particulier dans les composants de puissance.

Dans cette étude nous nous intéresserons principalement aux SEUs car parmi les effets énoncés les SEUs sont en grande partie responsable des défaillances des microprocesseurs.

I.1.2.2 Le Single Event Upset

Le Single Event Upset a pour effet le basculement du contenu d'une cellule mémoire suite au passage d'une particule. Ce phénomène intervient essentiellement dans les cellules de type SRAM. Un neutron génère une particule secondaire ionisante, capable de déposer suffisamment d'énergie. Cette particule secondaire, en interagissant avec la matière, crée un courant parasite.

Les cellules mémoires sont formées pour la plupart de transistors MOS qui jouent le rôle d'interrupteurs. Pour une technologie CMOS, une cellule mémoire est formée de deux inverseurs sont composés chacun d'un transistor NMOS et d'un transistor PMOS. Ces deux inverseurs sont rebouclés pour former la cellule mémorisant un bit.

La Figure I-3 représente le mécanisme de SEU provoqué par le passage d'un neutron dans un transistor NMOS d'un point mémoire SRAM.

Lorsqu'un neutron interagit dans la zone active du transistor NMOS bloqué N2, les porteurs générés le long de la trace de l'ion peuvent créer des chemins conducteurs ou peuvent être collectés suivant des mécanismes divers. Dans la figure, le nœud A est à 0 et le nœud B à 1, l'ion traverse le drain d'un des transistors OFF de la cellule mémoire, le courant induit sera responsable de commutations parasites dans la cellule. C'est ce pic de courant parasite qui est à l'origine de l'effet SEU. L'erreur induite sur la tension de sortie d'un des deux inverseurs est reportée sur les tensions de grille de l'autre structure inverseuse qui modifiera ainsi l'information stockée.

Le basculement d'une seule cellule dû au passage d'une unique particule est appelé Single Bit Upset (SBU) alors que celui de plusieurs cellules mémoire est appelé Multiple Cell Upset(MCU). Lorsque les cellules appartiennent à un même mot logique, on parle de Multiple Bit Upset (MBU). Il est généralement possible de détecter et de corriger ces erreurs multiples d'un même mot à l'aide d'un code de correction d'erreur dans la mesure où la multiplicité de l'erreur n'est pas plus grande que sa capacité de correction.

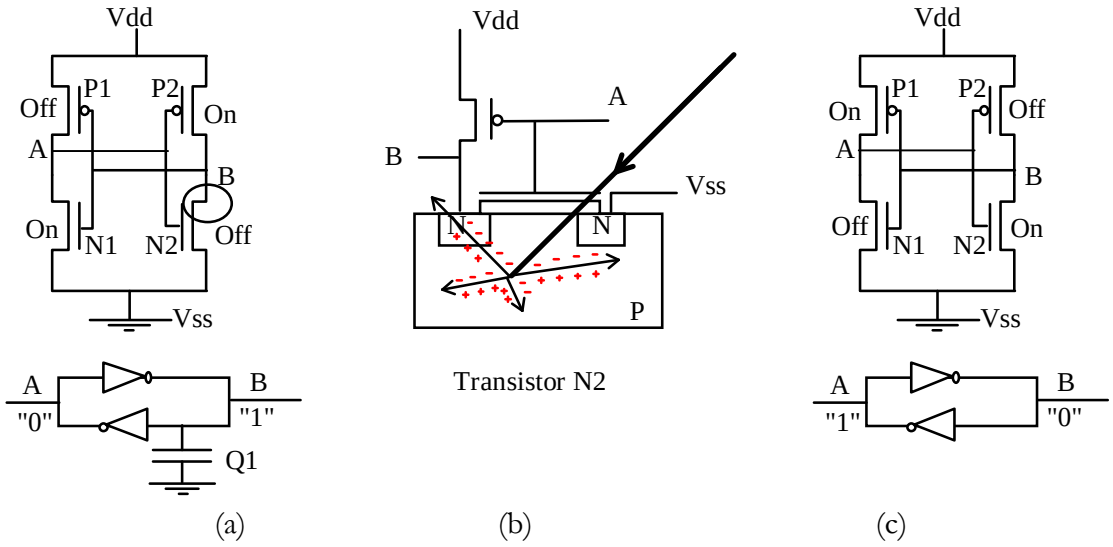


Figure I-3 : Basculement d'un point mémoire dû à l'impact d'un neutron (a) Point mémoire dans son état initial (b) Passage d'une particule ionisante au niveau du transistor N2 (c) Changement d'état du point mémoire [HUB01]

I.1.2.3 Paramètres de quantification de la sensibilité radiatives

Cette partie décrit les paramètres utilisés pour l'évaluation des risques aux SEU.

La section-efficace statique donne une mesure pire cas de la sensibilité aux SEU des zones sensibles du circuit. Elle représente la sensibilité intrinsèque d'un composant électronique vis-à-vis d'un type de particule donné. La section-efficace σ est obtenue en irradiant le composant par un flux de particules. Exprimée en cm^2/bit, elle est égale au quotient entre le nombre de défaillances observées ($N_{défaillances}$) et le nombre total de particules auxquelles le composant aura été exposé appelé fluence (particule/cm^2), multiplié par le nombre total de bits dans le composant (bit) (Eq. I-1).

$$\sigma = \frac{N_{défaillances}}{fluence \times Nbre\ bits} \qquad \textbf{Eq. I-1}$$

La section-efficace est tracée en fonction de l'énergie de la particule incidente (exprimé en MeV) ou du LET (Linear Energy Transfer) qui représente la perte d'énergie dE/dx cédée par unité de longueur et s'exprime en MeV/(mg/cm^2). L'effet d'un ion en incidence normale provoque une défaillance qu'à partir d'une valeur minimale de LET dit LET seuil.

La norme JESD89A [JED06] recommande des tests pour estimer la section-efficace d'un composant, soit en environnement réel, soit en accélérateur de particules avec des ions lourds,

des sources protons ou neutrons. Dans le cas des tests en environnement réel, plusieurs composants sont placés en altitude pendant plusieurs mois et on observe les erreurs survenues. Ces tests doivent être effectués en haute-altitude car le flux de neutron y est plus important. L'inconvénient majeur de cette approche est le nombre de composants nécessaires pour avoir des résultats significatifs et la durée relativement longue des tests.

Les tests en accélérateur de particules permettent de remédier à ces inconvénients et offrent l'avantage d'obtenir une section-efficace beaucoup plus rapidement. Pour cela, le composant est irradié avec un motif prédéfini. Puis pendant l'irradiation, par comparaison du motif irradié avec le motif initial, le nombre d'erreurs est obtenu.

Il existe différents moyens de reproduire l'environnent atmosphérique : une source neutronique à spallation qui reproduit le spectre de neutron terrestre, des sources de neutrons quasi mono-énergétiques et des sources de protons. L'environnement spatial est simulé lui par des accélérateurs d'ions lourds et de protons.

Une courbe de section efficace, obtenue à partir de points expérimentaux à différentes énergies pour un microprocesseur est donnée Figure I-4 [IRO05].

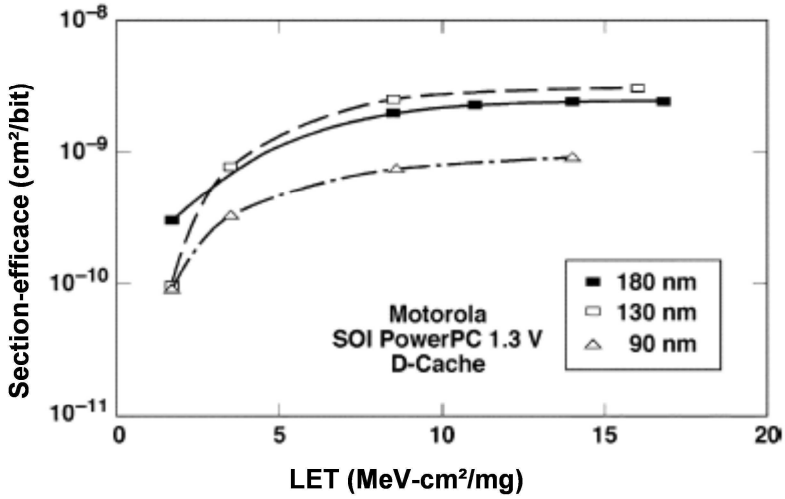


Figure I-4 : Section-efficace aux SEU pour la mémoire cache de données de trois microprocesseurs Motorola : 7455, 7457 et 7448 [IRO05].

Il existe donc dans l'espace et dans l'atmosphère, un environnement radiatif très contraignant provoquant des effets qui perturbent le fonctionnement des composants électroniques. Parmi ces effets, le SEU est l'évènement le plus présent dans les éléments mémoires.

I.2 Présentation de l'architecture globale des microprocesseurs

Les microprocesseurs sont largement utilisés dans divers domaines car ils présentent de nombreux avantages, tant du point de vue de leur coût que de leurs performances qui n'ont cessées de croître ces dernières années. Dans cette partie, une description succincte d'un microprocesseur ainsi que de son fonctionnement interne est proposée afin d'introduire les principales notions qui seront développées dans la suite du manuscrit. L'évolution de leur performance est aussi étudiée afin de mettre en avant l'importance des mémoires cache dans un microprocesseur.

I.2.1 Présentation générale

Un microprocesseur est un composant électronique complexe composé de millions de transistors sur une seule puce. Il fonctionne en exécutant une suite d'instructions spécifiques à son architecture.

I.2.1.1 Introduction

Le premier processeur a été inventé en 1971. Il s'agit du processeur Intel 4004, une unité de calcul de 4 bits cadencé à une fréquence de plus de 100 kHz. Peu de temps après, en 1974, un ordinateur d'IBM intégrait un processeur 8 bits d'Intel : le 8080 avec une capacité d'adressage de 64 Ko. Le 8086, introduit en 1978, a marqué le passage à une architecture 16 bits avec un espace d'adressage de 1 Mo. Depuis cette époque, l'utilisation des microprocesseurs s'est largement généralisée dans l'informatique, dans les appareils de mesure et dans le domaine grand public. Les microprocesseurs dits pris sur étagère (« Commercial Off-The-Shelf » ou COTS) ont ainsi joué un rôle primordial dans le développement de l'informatique et des nouvelles technologies. L'augmentation constante de leur performance et de leur complexité a contribué au développement d'applications de plus en plus élaborées.

Depuis quelques années, les avionneurs marquent une forte volonté d'embarquer des processeurs COTS pour bénéficier de leur performance et de leur faible coût. Cela permet aussi de réduire très fortement le poids et la quantité d'espace nécessaire pour un système embarqué.

I.2.1.2 Architecture interne générale des microprocesseurs

Dans ce paragraphe, une introduction brève de la structure interne des microprocesseurs est présentée. Il existe de grandes familles de microprocesseurs comme les processeurs x86 d'Intel, la

famille de processeurs ARM, MIPS, SPARC ou encore PowerPC : chacune de ces familles se distingue par son type d'architecture, son jeu d'instructions et sa structure interne.

a) Types d'architecture des microprocesseurs.

L'architecture d'un microprocesseur peut appartenir à deux grandes classes :

-L'architecture CISC (Complex Instruction Set Computer), utilisée dans les premiers microprocesseurs de type x86 d'Intel, présente un jeu d'instructions assez important et complexe. En effet, une même instruction gère le calcul, les accès aux mémoires et aux registres et le temps d'exécution pour ce type d'architecture est en général relativement long.

-L'architecture RISC (Reduced Instruction Set Computer) utilisée notamment très tôt par IBM dans l'objectif d'optimiser le temps d'exécution d'un programme, utilise un jeu d'instructions plus simple, ce qui permet des exécutions plus rapides (PowerPC, ARM) et d'avoir de meilleures performances de calcul. Les instructions de calcul se font à l'aide de registres uniquement et les accès mémoires sont gérés par des instructions spécifiques.

b) Jeu d'instructions

Chacune des grandes familles de microprocesseurs possède son propre jeu d'instructions qui est l'ensemble des opérations élémentaires exécutables par un microprocesseur. Il est généralement structuré en plusieurs catégories :

- les accès à la mémoire : stockage, chargement …

- les opérations arithmétiques : addition, soustraction …

- les opérations logiques : OU, ET,

- les opérations de contrôle : branchements …

- les opérations de configuration : registres internes…

Le nombre et le type d'instructions d'un processeur est bien délimité par son jeu d'instructions.

c) Structure interne

Un microprocesseur est généralement constitué d'éléments de base comme des unités de traitement permettant d'effectuer des calculs arithmétiques ou logiques (pour les entiers et pour

les nombres de type flottants), des registres qui sont de petits éléments de stockage temporaire, une unité de contrôle et une ou plusieurs mémoires locales appelées mémoire cache.

La description en blocs fonctionnels de l'architecture interne d'un microprocesseur de type PowerPC e300, telle qu'indiquée dans la fiche technique produit de la société Freescale [FRE07], est donnée sur la Figure I-5. La structure interne du microprocesseur est illustrée par cette figure.

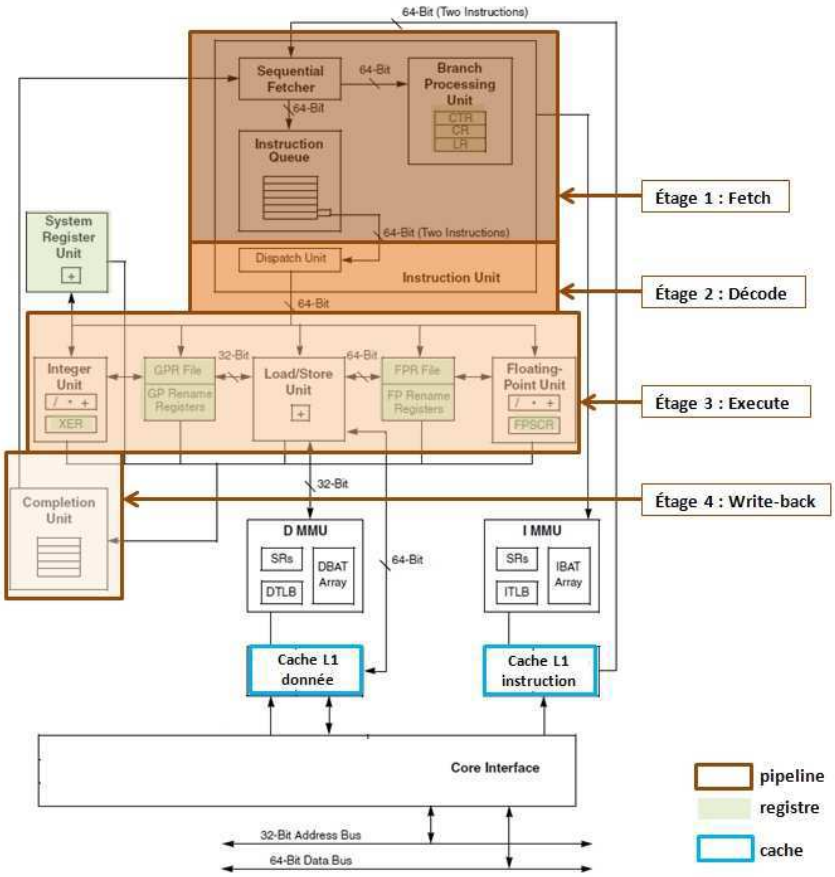


Figure I-5 : Pipeline du cœur PowerPC e300 [FRE07]

La caractéristique majeure de cette architecture est qu'elle contient trois unités d'exécution en parallèle, capables de réaliser des opérations en même temps : unité de calcul des entiers, unité de calcul des flottants et l'unité d'écriture et lecture. Cette architecture possède une mémoire cache de niveau 1, constituée d'un cache L1 de données et d'un cache L1 d'instructions.

Le premier étage consiste à rechercher une instruction, pour cela le CPU a accès à la mémoire pour lire le mot placé à l'adresse contenue par le compteur de programme. Dans un microprocesseur, le compteur de programme, appelé aussi compteur ordinal (ou en anglais « program counter » PC) contient l'adresse mémoire de l'instruction en cours d'exécution ou

prochainement exécutée. Il s'incrémente automatiquement sauf dans le cas d'instructions de branchement ou de saut conditionnel. L'instruction chargée est ensuite décodée afin d'identifier l'unité de calcul appropriée. L'instruction est composée d'un code d'opération appelé « opcode » et d'une partie permettant d'identifier les opérandes. La valeur de l'opérande est chargée selon le mode d'adressage : direct, indirect, immédiat … Puis, le calcul est effectué dans l'étage d'exécution selon les types de données, entiers ou flottants. Cet étage permet aussi d'accéder à la mémoire en lecture ou écriture, dans le cas de types instruction de chargement ou de stockage. Le dernier étage est en général réservé à l'enregistrement du résultat.

Le pipeline de l'architecture e300 [FRE07] est de type « superscalaire », c'est-à-dire que le processeur peut traiter plusieurs instructions en un seul cycle à chaque étage du pipeline. Par exemple, au premier étage du pipeline le processeur peut charger jusqu'à deux instructions à la fois.

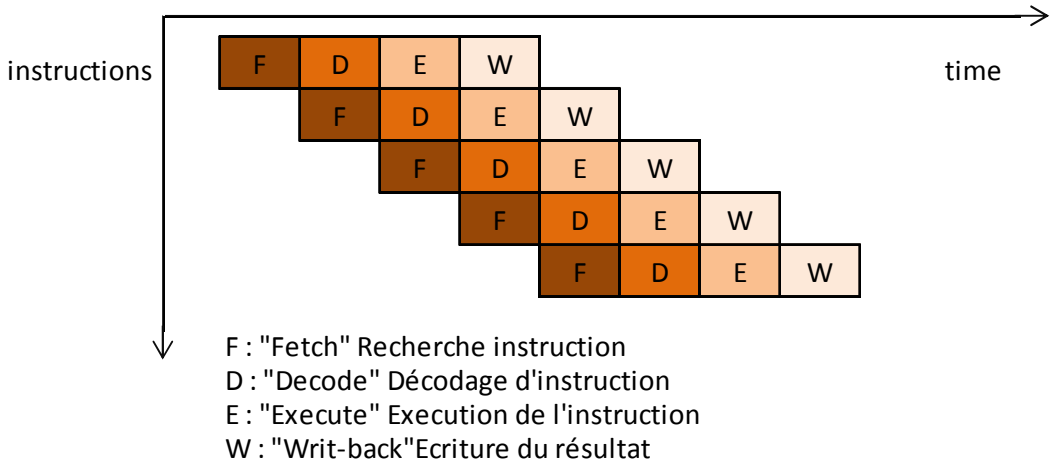


Figure I-6 : Pipeline de base à 4 étages

Les microprocesseurs actuels intègrent des unités plus complexes comme l'unité de prédiction de saut, qui prédit l'effet d'une instruction de branchement dans le déroulement du programme. Ils offrent aussi la possibilité de réorganiser l'ordre dans lequel le processeur va exécuter ses instructions (principe appelé « out-of-order »), contrairement aux « in-order » qui exécutent les instructions dans l'ordre. Ceci permet d'augmenter les performances du processeur en évitant les temps d'attente lors des accès mémoires par exemple.

Les différents éléments mémoires d'un microprocesseur sont en général les registres et les mémoires cache.

Les registres sont des éléments de mémoire interne qui contiennent les données, ou les instructions que le processeur utilise. Il existe ainsi plusieurs types de registres :

- Les registres généraux : registres de stockage temporaire et rapide pour les unités de calcul (entier, virgule flottante ...)

- Les registres de configuration : fournissent des informations sur le modèle de processeur, et configurent le mode de fonctionnement du processeur comme les interruptions par exemple [FRE07].

- Les registres d'état : sauvegarde du contexte du processeur et des informations concernant le résultat de calcul.

- Les registres spécifiques du processeur : ce sont des registres spécifiques à une architecture, comme par exemple des registres donnant des informations sur la performance du processeur.

La mémoire cache est devenue un élément indispensable des processeurs actuels. Elle a pour fonction d'optimiser les accès entre le processeur et la mémoire principale en stockant les informations les plus récentes auxquelles accède le plus souvent le processeur.

Lorsque le processeur requiert une information (instruction ou donnée), il va tout d'abord y accéder en mémoire cache. Si cet élément n'est pas disponible en mémoire cache, le gestionnaire de cache va chercher cette donnée dans la mémoire principale et charger en même temps cette donnée dans la mémoire cache pour un prochain accès. De ce fait, le processeur pourra y accéder plus rapidement. La capacité d'une mémoire cache est en général bien inférieure à celle de la mémoire principale. Il existe communément plusieurs niveaux de cache entre un processeur et la mémoire principale :

- Le cache de niveau 1 (cache L1) se situe au plus près du processeur et est en général constitué de deux parties : l'une stockant les instructions à exécuter et l'autre les données.

- Le cache de niveau 2 (cache L2) sert d'intermédiaire entre le cache L1 et la mémoire principale ou le cache L3. Ce cache contient à la fois les données et les instructions. Il est moins rapide que le cache L1 mais de taille beaucoup plus importante.

- Le cache de niveau 3 (cache L3) est présent sur les derniers microprocesseurs. Sa taille est assez importante.

Les performances de cache sont mesurées à l'aide du ratio de réussite et du temps de latence. Le ratio de réussite, aussi appelé « *cache hit* », correspond au rapport entre le nombre d'accès où l'information était bien dans le cache et le nombre total d'accès. En d'autres termes, c'est le pourcentage de réussite pour trouver l'information dans le cache. La latence correspond au temps d'attente en cas d'échec.

L'unité de gestion de mémoire, aussi appelé MMU (Memory management unit) a pour fonction de traduire les adresses logiques en adresse physiques. Les adresses sollicitées par les CPU sont des adresses virtuelles (logiques). Le « Translation Lookaside Buffer », aussi appelé TLB est une mémoire cache qui contient la traduction des adresses logiques en adresses physiques.

I.2.1.3 Outils de développement

Le développement d'un système autour d'un microprocesseur nécessite un certain nombre d'outils afin d'aider les concepteurs et les développeurs à valider leur système d'un point de vue matériel puis d'un point de vue logiciel.

La simulation est l'un des moyens les plus utilisés pour le développement matériel d'un microprocesseur, ou pour aider les concepteurs dans le choix d'architecture de leurs systèmes. Les simulateurs fonctionnels (appelé aussi « Instruction Set Simulator ») simulent seulement le jeu d'instructions. Ils mettent à jour les registres et exécutent les opérations de l'unité arithmétique et logique. Le simulateur de type « cycle-accurate » doit modéliser précisément le matériel et garantir une exécution représentative au cycle près. Ce type de simulateur ajoute donc à la simulation fonctionnelle une modélisation du processeur au niveau temporel. Il présente l'avantage de fournir des informations concernant les performances : nombre d'instructions exécutées par cycle, ratio de *cache hit*… Il fournit également la consommation de puissance du cœur. Cependant, l'inconvénient majeur de ce type de simulation est la lenteur d'exécution des programmes simulés.

Pour le développement logiciel, les outils de type débogueur sont communément employés. Un tel outil permet d'analyser le comportement du code exécuté sur un microprocesseur. En général, l'utilisateur a accès en lecture ou écriture au contenu des variables, des registres ou des mémoires. Il peut aussi contrôler l'exécution du code en stoppant ou en plaçant des points d'arrêt. La plupart des microprocesseurs possèdent une interface de test compatible avec la norme JTAG (joint test action group ou la norme IEEE 1149.1 (*Standard Test Access Port and*

Boundary Scan Architecture). Ce type d'interface a pour objectif (outre le test du composant) de faciliter la programmation du microprocesseur en donnant accès à l'intérieur du processeur.

Il existe donc ainsi un ensemble d'outils permettant d'aider les concepteurs logiciels et matériels à développer des systèmes à base de microprocesseurs.

I.2.2 Performance et évolution

I.2.2.1 Critères de performance

L'évolution du nombre de transistors des microprocesseurs et l'évolution de la technologie des semi-conducteurs sont représentées à la Figure I-7 [HOU00]. Cette courbe suit la loi de Moore prédite en 1965, selon laquelle le nombre de transistors sur une puce double tous les deux ans.

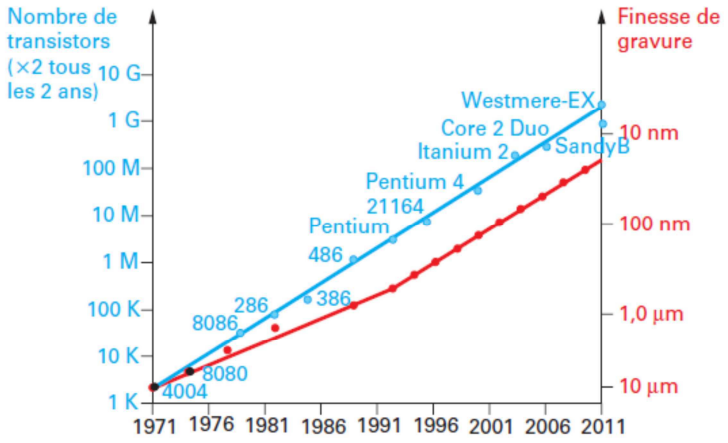


Figure I-7 : Évolution des performances des microprocesseurs et de la technologie [HOU00]

L'augmentation de la performance des microprocesseurs est due à l'amélioration de plusieurs facteurs comme le nombre de cœurs, l'architecture interne, la fréquence de fonctionnement, les mémoires caches, la parallélisation et l'optimisation des séquences d'instructions…

Le type d'architecture est un facteur déterminant dans la performance des processeurs : les processeurs COTS actuels sont pour la plupart de type RISC.

L'accroissement du nombre de transistors sur une même puce a permis d'intégrer des unités de calcul plus complexes, des mémoires internes de plusieurs niveaux…

L'augmentation de la fréquence des processeurs a permis d'exécuter plusieurs instructions en moins de temps et donc d'augmenter la vitesse de calcul.

Les mémoires cache améliorent la rapidité d'accès aux données les plus utilisées. Les fabricants recherchent un compromis entre la taille, le type de mémoire cache, et les techniques d'accès aux données pour améliorer le taux de réussite.

La parallélisation et l'optimisation des séquences d'instructions se font au niveau du pipeline. Par exemple, vers les années 2000, le Pentium III d'Intel avait 10 étages de pipeline et dans la version suivante le Pentium 4 avait 30 étages. Le but est de réduire au maximum les temps de traitement, et d'exécuter plusieurs instructions simultanément. Les difficultés majeures de cette optimisation résident dans le fait que les calculs sont souvent dépendants, et donc il est nécessaire d'attendre la valeur du résultat du calcul précédent.

Le nombre d'instructions exécutées par seconde, ou MIPS (Millions d'instructions par seconde), est l'une des manières de caractériser la performance générale d'un processeur. Le nombre moyen d'instructions exécutées par cycle d'horloge. L'IPC (« Instruction Per Cycle ») est une autre mesure qui exprime cette performance.

Ainsi, à titre d'illustration, le Tableau I-2 montre le déploiement de la famille de PowerPC. Ce tableau montre la manière dont les microprocesseurs couvrent la gamme de performances (MIPS) en utilisant différentes dimensions :

- La fréquence,

- La taille de la mémoire cache,

- Les différents types d'architecture.

Tableau I-2 : Évolution des microprocesseurs Freescale

Nom de micro-processeur	Fréquence (MHz)	Cache	Architecture interne	MIPS
MPC8260	333-450	L1 I/D : 16 KB	603	570 MIPS à 300 MHz
MPC8349	400-667	L1 I/D : 32 KB	e300	1260 MIPS à 667 MHz
MPC8569	800-1333	L1 I/D : 32 KB L2 : 512 KB	e500	2799 MIPS à 1,33 GHz
P4080	1200-1500	L1 I/D : 32 KB/cœur L2 : 128 KB/cœur L3 : 2 MB	e500mc (8 cœurs)	Non communiqué
T4240	1800	L1 I/D : 32 KB/cœur L2 : 2 MB/cœur L3 : 512 MB	e6500 (16 cœurs physiques et 24 cœurs virtuels)	6000 MIPS par cœur

I.2.2.2 Évolutions des mémoires cache dans les familles de processeurs

Parmi les éléments décrits dans la section précédente, les mémoires cache ont un rôle important et sont régulièrement améliorées par les fabricants. L'évolution des mémoires cache dans les familles de processeurs INTEL, PowerPC et ARM est présentée par la suite comme exemple.

Depuis les années 2000, les microprocesseurs de la famille Intel ont vu le nombre de leurs transistors considérablement augmenter. Le Tableau I-3 donne à titre indicatif le nombre de transistors pour deux familles différentes : le Pentium 4 et le Pentium M. L'estimation du nombre de transistors dans les mémoires cache de niveau L1 et L2 a été calculée sur la base de cellules SRAM de type 6-T [THO02]. Le pourcentage de transistors lié à la mémoire cache de niveau L1 et de niveau L2 ne cesse de croître. Pour un processeur de type « Prescott-2M », 60 % des transistors de la puce sont liés aux mémoires cache.

Les processeurs de la famille de Freescale [WWW02] sont largement utilisés dans l'industrie aéronautique. Il existe une première génération de microprocesseurs de type PowerQUICC. Depuis, les années 1995, on remarque que chaque famille de processeurs Freescale a vu sa taille de cache augmenter. La Figure I-8 donne l'évolution de la taille des mémoires cache des microprocesseurs Freescale pour chaque cœur depuis les années 1995. Le cache de niveau L1 de données est représenté en bleu foncé et le cache d'instructions en bleu clair. La taille de la mémoire cache de niveau L1 a augmenté jusqu'aux derniers PowerQUICC III, où elle se limite à 32 KB. Ceci s'explique du fait que le premier niveau de cache doit garder un compromis entre une taille relativement grande et un accès assez rapide, afin d'optimiser au maximum les

performances du processeur. Ainsi, un deuxième niveau de cache (L2) (vert) est présent sur tous les microprocesseurs de la famille multicoeurs « QorIQ ». Les dernières générations de processeurs ont vu l'apparition d'un dernier niveau de cache L3 (couleur jaune), qui est plus lent mais permet d'accéder à la donnée dans cette mémoire plutôt qu'en mémoire principale. Elle possède une taille relativement importante de l'ordre de 2 MB pour les 4 cœurs du processeur « QorIQ » P5.

Tableau I-3 : Répartitions du nombre de transistors dans les microprocesseurs de la Famille Intel

Famille d'Intel	Nom du processeur	Année	Techno-logie	Nombre de transistors total	Estimations du nombre de transistors dans le cache L1	Estimations du nombre de transistors dans le cache L2	Pour-centage de transistors liés à la mémoire cache
Pentium 4	« Willanette »	2000	180 nm	42 millions	0,39 million (8 KB)	13 millions (256KB)	31%
	« Nothwood »	2002	130 nm	55 millions	0,39 million (8 KB)	25 millions (512KB)	46%
	« Prescott »	2004	90 nm	125 millions	0,79 million (8 KB)	50 millions (1 MB)	41%
	« Prescott -2M »	2005	90 nm	169 millions	0,79 million (8 KB)	100 millions (2 MB)	60%
Pentium M	« Banias »	2003	130 nm	77 millions	0,39 million (8 KB)	50 millions (1 MB)	66%
	« Dothan »	2004	90 nm	144 millions	0,39 million (8 KB)	101 millions (2 MB)	70%

La Figure I-9 montre l'évolution des mémoires cache dans la famille de processeurs ARM [WWW03]. Depuis les années 2006, la série des Cortex-R a augmenté ses mémoires cache jusqu'à 2 MB pour les mémoires de taille configurable. La figure représente les tailles maximum possibles pour chaque mémoire. À partir des années 2010, les séries Cortex-A ont intégré une mémoire cache L2 allant jusqu'à 4 MB. De plus, la multiplication des cœurs a entraîné l'accroissement de la proportion de mémoire cache.

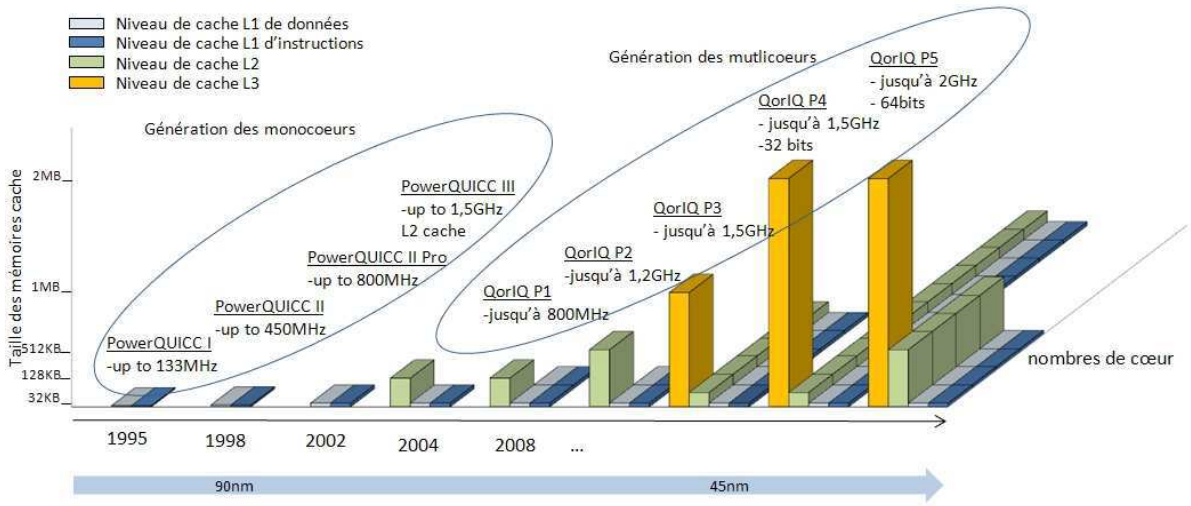


Figure I-8 : Évolution des mémoires cache dans la famille de processeurs Freescale

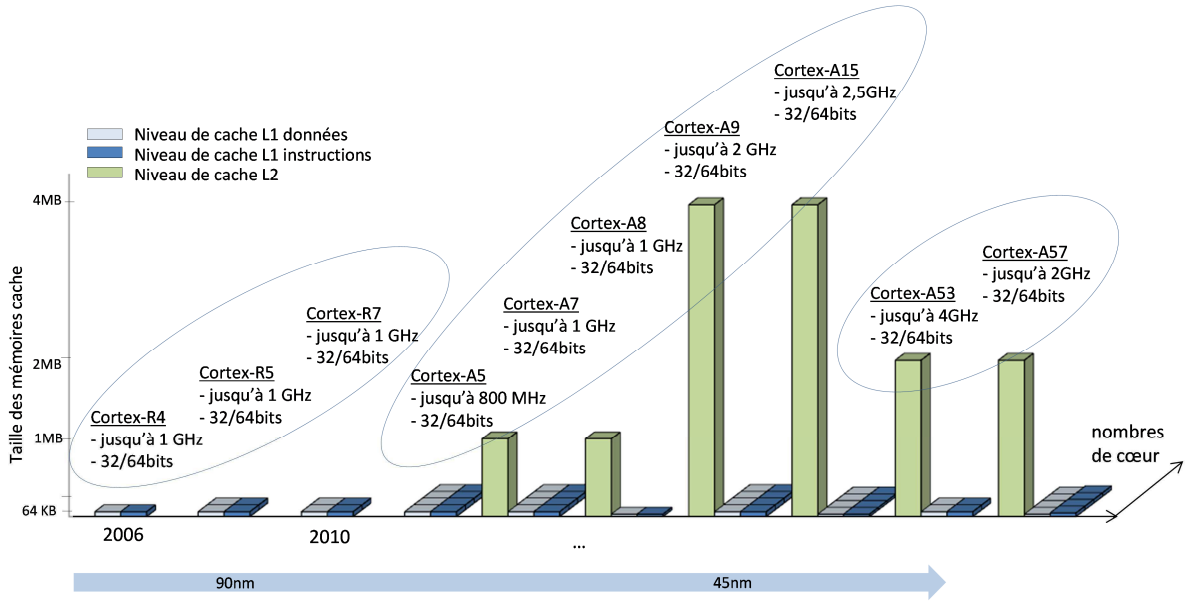


Figure I-9 : Évolution des mémoires cache dans la famille de processeurs ARM

En conclusion, la complexité des microprocesseurs COTS n'a cessé d'augmenter depuis leur création. Les mémoires cache jouent un rôle important dans leur évolution. Elles représentent la majeure partie des cellules mémoires d'un microprocesseur. La haute performance de calcul, le faible coût et la diminution de taille sont les raisons pour lesquelles les avionneurs souhaitent intégrer les microprocesseurs dans leurs équipements. Il est donc important d'étudier les risques liés aux SEU dans un microprocesseur et savoir comment les évaluer.

I.3 Effets des radiations sur les microprocesseurs

I.3.1 Identification des éléments sensibles aux SEU

Les éléments mémoires sont les composants les plus susceptibles aux SEU. Ainsi, dans un microprocesseur tous les éléments mémoires comme les registres et les mémoires caches sont sensibles aux SEU. L'exemple du processeur LEON3 FT montre bien que les éléments mémoires sont les plus critiques dans un microprocesseur. Le LEON3 FT est une version tolérante aux fautes du processeur LEON3 standard. Les éléments mémoires ont été les plus protégés dans ce processeur car ils sont considérés comme les plus sensibles. De ce fait, les registres internes sont protégés par des codes correcteurs d'erreurs et la mémoire cache du LEON3 FT, constituée de caches séparés de données et d'instructions, est protégée par un mécanisme de protection de type parité. En revanche, l'unité de contrôle, de calcul et les bus de données ne sont pas durcis.

Irom *et al.* [IRO05], ont testé plusieurs processeurs de type PowerPC tels que le Motorola 7455. Les résultats des tests d'irradiation aux ions lourds ont montré que les mémoires cache ont une section-efficace de $9{,}2 \times 10^{-10}$ cm²/bit et les registres internes ont une section-efficace estimée autour de 9×10^{-8} cm²/bit pour une technologie de 90 nm. De plus, la section-efficace liée aux MBU a aussi été mesurée. Elle montre que le taux de MBU est 200 fois inférieur aux taux de SBU pour les mémoires caches et il est 50 fois inférieur pour les registres internes.

Les éléments les plus sensibles dans un microprocesseur sont donc les mémoires caches compte tenu de leur taille relativement importante par rapport à celles des registres. Notre étude se focalisera donc sur les mémoires cache.

I.3.2 Stratégies de qualification des processeurs aux effets singuliers

Dans le but de prédire le taux d'erreurs, un composant électronique est généralement exposé à un flux de particules durant son fonctionnement. Pour les microprocesseurs, aucune stratégie n'est établie. Les conséquences d'un SEU sont difficiles à prédire, et dépendent de la nature de l'information et de l'instant d'occurrence de la perturbation. Ainsi, les fautes induites dans un microprocesseur embarqué dans des applications avioniques ou spatiales auront une conséquence différente selon l'application exécutée. Le JEDEC [JED06] recommande de tester individuellement chaque élément du processeur pour évaluer la sensibilité par élément, c'est l'approche statique, ou bien de calculer le taux d'erreur global du microprocesseur, c'est

l'approche dynamique. Cette partie commencera par aborder en premier les stratégies de qualification des microprocesseurs utilisées pour un équipement avionique ou spatial, puis comment la caractérisation de la sensibilité qui en découle influence le développement d'un tel système embarqué.

I.3.2.1 Approche statique

L'approche statique est une stratégie généralement utilisée pour estimer la sensibilité des composants de types mémoires.

À partir de la section efficace, on peut calculer le taux d'erreurs appelé Soft Error Rate (SER) pour un environnement donné. Le calcul du SER est obtenu en multipliant la section efficace par le flux moyen de particules où E est l'énergie des neutrons en MeV, E_{min} et E_{max} sont les bornes du spectre d'énergie exprimé en MeV, $d\phi(E)/dE$ est le flux différentiel neutron exprimé en cm^{-2}. $MeV^{-1}.s^{-1}$ et σ_{SEU} est la section-efficace (Eq. I-2).

$$SER = \int_{E_{max}}^{E_{min}} (d\,\phi(E)/dE)\sigma_{SEU}(E)\mathrm{dE} \qquad \textbf{Eq. I-2}$$

Il peut aussi être exprimé en FIT (Failure in Time) pour représenter 10^9 heures de fonctionnement.

Pour un environnement atmosphérique, l'IEC62396 [IEC12] définit un flux de 6000 n/cm²/h (pour une énergie de neutron supérieure à 1-10 MeV) comme valeur nominale pour 12,2 km d'altitude et 45° de latitude. Ainsi, on obtient le calcul de taux d'erreurs suivant (Eq. I-3) :

$$SER = 8000\ (\mathrm{n/cm^2/h}) \times \sigma_{SEU} \qquad \textbf{Eq. I-3}$$

De plus, le calcul du SER doit aussi prendre en compte le taux d'erreurs lié aux MBU à partir de la section-efficace de ceux-ci.

La stratégie utilisée pour caractériser la sensibilité d'un processeur consiste à irradier chaque type de cellule mémoire (registre et mémoires internes). Cette méthode, héritée du test sur les composants mémoire, permet d'évaluer la sensibilité des différentes zones mémoire du processeur.

La mesure de la section-efficace d'un élément mémoire consiste à exposer le composant à un flux de particules pendant un temps donné et à observer les évènements dans les registres et les éléments mémoires (registres, caches et mémoire interne). Les éléments mémoires doivent au préalable être chargés d'un motif prédéterminé comme le motif échiquier, le motif tout à 0 ou le motif tout à 1. Pour accéder aux éléments mémoires, il existe des moyens tel que le contrôleur JTAG ou le débogueur. Les zones observées sont limitées, soit par le jeu d'instructions, soit par l'architecture du processeur [PEA98].

Cependant, l'approche statique n'est pas la stratégie la mieux adaptée aux microprocesseurs. En effet, l'application qui s'exécute utilise différemment les cellules mémoires du processeur dans le temps et l'espace.

De nombreux travaux ont montré que la sensibilité dynamique était différente de celle obtenue par test statique et était dépendante de l'application [KAR93] [VEL92] [BEZ96] [ELD88] [CUS85] [ASE98].

I.3.2.2 Approche dynamique : notion de facteur de sensibilité

Le comportement du processeur dépend de l'utilisation de ses ressources internes. L'approche dynamique consiste donc à exposer le composant à un flux de particules pendant qu'il exécute une application. Ce cas de test est plus proche du cas réel dans lequel les cellules mémoires ne sont pas toutes utilisées durant le cycle complet d'exécution d'un programme. Les logiciels exécutés sont généralement complexes, leurs séquences d'instructions ne sont pas toujours prévisibles (exemple : les sauts conditionnels).

Le taux d'erreurs dynamique peut être beaucoup plus faible que celui estimé en test statique car un microprocesseur n'utilise pas ses ressources à 100 %.

Le cas idéal de test consiste à tester le processeur dans son environnement réel (périphériques et exécution de l'application finale). Cependant, cela impose des contraintes de test assez importantes tel que la durée du test (partie I.4.2.3).

La sensibilité dynamique d'un microprocesseur (sensibilité de l'application) ($SER_{dynamique}$) s'obtient à partir de la sensibilité statique ($SER_{statique}$) pondérée par des facteurs de décote Π appelés « facteurs de sensibilité » ou encore « vulnerability factor » [MUK03] (Eq. I-4) :

$$SER_{dynamique} = SER_{statique} \times \pi \qquad \textbf{Eq. I-4}$$

Les facteurs de sensibilité représentent à la fois le pourcentage d'éléments sensibles dans le processeur et le temps pendant lequel ils sont considérés comme sensibles. De nombreuses méthodes existent pour estimer ces facteurs de sensibilité. La section I.3.2 résume les principales méthodes pour évaluer la sensibilité dynamique des microprocesseurs.

I.3.2.3 Exemple de stratégie de qualification d'un équipementier.

Dans le milieu de l'aéronautique, les risques liés aux rayonnements cosmiques sont décrits dans les exigences de sûreté de fonctionnement définies par l'autorité de certification. Il s'agit de caractériser la fiabilité ou la disponibilité du système en prenant en compte l'impact de l'environnement radiatif sur les composants. Ces exigences de fiabilité pour le développement des systèmes embarqués sont décrites dans les documents ARP4754 [ARP4754] et ARP4761 [ARP4761]. Chaque fonction ou équipement aéronautique est caractérisé par un niveau de criticité (« *Design Assurance Level* » DAL).

Une stratégie communément utilisée chez un équipementier de système avionique pour analyser et quantifier les risques liés aux SEU dans un équipement avionique est brièvement présentée ci-dessous [RAD10]. Une méthode similaire existe aussi au niveau spatial [RAD11] mais n'est pas étudiée dans cette partie.

Cette analyse est liée à l'impact des SEU sur une application (un ensemble de tâches logicielles) exécutée sur un microprocesseur. Elle consiste à quantifier le taux de SEU au niveau du composant (approche statique) puis à évaluer le taux par tâche exécutée sur le microprocesseur.

Ainsi, la première étape est donc de calculer le taux de SEU statique pour le microprocesseur. La section-efficace est en général soit fournie par le fabriquant soit issue de la littérature avec différentes marges appliquées suivant la précision de la donnée. Sur un calculateur avionique et selon la norme ARINC 653, l'exécution conjointe de plusieurs tâches nécessite la décomposition des ressources accessibles en sous-ensembles robustes. Les tâches sont séparées en différentes partitions selon leur criticité, et elles ne doivent pas interférer entre elles. Une allocation spatiale et temporelle est alors attribuée pour chaque partition exécutée par le microprocesseur.

Le taux d'erreurs peut alors être calculé pour chacune des partitions par rapport aux ressources spatiales ou temporelles qu'elles utilisent dans chaque composant, aussi appelé facteur d'exposition. La quantité d'éléments utilisés ou le temps pendant lequel la partition exécute son code sont généralement définis dans les spécifications.

Ainsi, le taux d'erreurs SER d'une partition n, SER (partition n) est calculé de la manière suivante (Eq. I-5) :

$$SER\ (partition\ n) = SER\ statique \times facteur\ d'exposition\ (partition\ n) \qquad \textbf{Eq. I-5}$$

L'équipementier doit ensuite identifier les évènements qui pourront être détectés et corrigés par l'implémentation de son architecture globale.

L'état de l'art propose des méthodes permettant à un concepteur de préciser le facteur d'exposition d'un produit au fur et à mesure de son développement.

I.4 État de l'art des méthodes d'analyse de la robustesse face aux SEU

La complexité des microprocesseurs ne cesse de croître avec le degré d'intégration de la technologie. Ils sont embarqués dans des systèmes de plus en plus complexes pour des applications avioniques ou spatiales qui imposent de fortes contraintes de fiabilité. C'est pourquoi, il existe un besoin constant de développer ou d'adapter les techniques de prédiction de taux d'erreurs pour les processeurs. Contrairement aux composants électroniques de type mémoire, le test sous accélérateur de particules n'est pas le moyen le mieux adapté pour prédire le taux d'erreurs des processeurs car celui-ci dépend fortement de l'application exécutée. Cependant, il reste l'outil de référence si les tests sont effectués sur des durées suffisamment longues et avec plusieurs applications pour l'exhaustivité du test. De plus, un test en accélérateur nécessiterait que le système soit à sa phase finale de conception, c'est-à-dire que le développement matériel et logiciel soit déjà achevé. Or à ce niveau, les modifications de conception peuvent être coûteuses. De ce fait, des méthodes de prédiction ont été développées en tenant compte de cette problématique. La plupart des techniques étudiées ces dernières années, sont fondées sur des méthodes d'injections de fautes sur des cibles réelles, sur des modèles ou dans des simulateurs de processeur. Ces méthodes peuvent être appliquées à différents niveaux du cycle de développement du système.

Les systèmes aéronautiques sont développés selon le principe du cycle en V. Ce modèle conceptuel de gestion de projet permet de limiter un retour aux étapes précédentes. Il est constitué d'une phase descendante avec des étapes de spécifications, conception architecturale et logicielle. Puis une phase ascendante avec des étapes de validation et de vérification. La référence [DUR09] donne un aperçu du développement de processus chez Airbus.

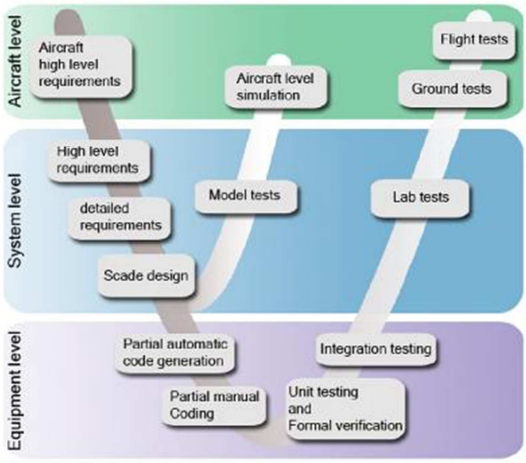


Figure I-10 : Développement d'un processus avionique [DUR09]

Les ingénieurs utilisent de nombreux outils pour s'assurer que les spécifications sont correctes et complètes. Ces outils permettent de valider la partie spécification de chaque niveau, ainsi que de valider les hypothèses faites (probabilité des taux de pannes...). Plus particulièrement, des simulateurs de système et de microprocesseur sont utilisés tout au long du processus de vérification et validation d'un système dont voici les différents types :

- les simulateurs de bureau qui simulent un système ainsi que son environnement. Ils analysent le comportement de l'architecture, celui de la logique opérationnelle ou la performance du système.

- les simulateurs de développement qui valident les exigences et les spécifications système avec un niveau de représentativité de l'environnement suffisant.

- les bancs d'intégration systèmes, qui sont des outils pour assembler le système à partir des sous-éléments physiques du système. Cela nécessite un environnement complet du système.

- les simulateurs d'intégration qui sont un moyen d'intégrer plusieurs sous-systèmes réels où l'environnement est simulé avec un niveau de représentativité suffisant pour évaluer l'opérationnalité du système entier.

Dans les pages qui suivent sont présentées les méthodes qui permettent de caractériser la sensibilité des processeurs durant la phase de conception (phase descendante du cycle en V), puis durant la phase de validation de l'implémentation (phase montante du cycle en V).

I.4.1 Phase de conception

Durant la phase de conception, des outils caractéristiques sont utilisés pour valider les spécifications comme des simulateurs de processeur qui permettent aux ingénieurs de commencer leurs évaluations sans attendre le prototype final. Les méthodologies de caractérisation de la sensibilité d'un microprocesseur au niveau de la phase de conception nécessitent d'avoir un modèle très proche du comportement de la puce. Les fabricants de microprocesseur conçoivent des modèles de processeur ou des environnements de simulation qui permettent aux développeurs d'avancer dans la phase de conception logicielle. Les méthodes évoquées dans cette partie ont pour objectif d'assister les concepteurs dans le choix de leurs composants, de l'architecture de leur produit ou dans le choix du logiciel.

I.4.1.1 Architectural Vulnerability Factor

Somani *et al.* [SOM97] ont développé un modèle probabiliste qui évalue la sensibilité de la mémoire cache et des registres internes d'un processeur. Cette méthode repose sur la connaissance interne de la microarchitecture de celui-ci pour vérifier la probabilité de propagation d'erreur dans sa structure. L'avantage de cette méthode est qu'il n'y a pas besoin d'injecter de fautes mais seulement d'exécuter le code une seule fois pour analyser le comportement de la microarchitecture du processeur. L'inconvénient est qu'il est nécessaire d'avoir une bonne connaissance (et une parfaite visibilité) de la microarchitecture, et une grande capacité d'enregistrement de l'activité de chaque structure interne du microprocesseur.

Mukherjee *et al.* [MUK03] ont introduit la notion de facteur de vulnérabilité de l'architecture ou encore « Architectural Vulnerability Factor » (AVF), comme la probabilité qu'une faute dans une structure du microprocesseur engendre une erreur d'application. Une structure du microprocesseur correspond par exemple à un élément physique du microprocesseur. Cette méthode est fondée sur la connaissance de la microarchitecture et l'identification des bits qui

n'induiront pas d'erreurs s'ils sont corrompus. Le taux d'erreurs du microprocesseur est ainsi estimé en multipliant le taux d'erreurs statistique par l'AVF calculé. Pour calculer l'AVF d'une structure, le principe est d'identifier les bits ACE « Architecturally correct execution », aussi appelés bits d'exécution d'architecture corrects. Une corruption de ces bits provoque une erreur d'application, leur intégrité est donc nécessaire à l'exécution sans erreur du code. Par opposition, les bits « un-ACE » sont les bits dont l'inversion ne conduira pas à une erreur d'application. L'AVF d'une structure est donc la fraction de temps pendant laquelle cette structure contient des bits ACE. En appliquant ce principe, les auteurs estiment l'AVF de chaque structure d'un microprocesseur soit en suivant la trace de chaque bit ACE tout le long du code à l'aide d'un modèle de performance, soit en analysant la pertinence de chaque structure. Par exemple, l'élément de prédiction de branchement aura un AVF de 0 % car cet élément permet de prédire les instructions qui pourraient être exécutées mais il n'est pas essentiel. En revanche, le compteur de programme aura lui un AVF de 100 % car il est jugé indispensable à l'exécution correcte d'un programme tout le temps.

De nombreuses études sur le facteur de vulnérabilité des mémoires cache ont été proposées. La plupart repose sur l'estimation des durées de vie des variables critiques dans les mémoires cache et nécessite un modèle détaillé du processeur. Les auteurs de [BIS05] proposent de déterminer les facteurs d'AVF pour les éléments mémoires (cache de données, buffer de translation de données et buffer de sauvegarde) en identifiant les mots ACE du cache. Biswas *et al.* ont donc introduit la notion d'analyse de durée de vie pour des cellules mémoires. Le principe consiste à diviser le cycle de vie d'un bit mémoire en fonction de ses différents états (lecture, écriture…) pour identifier les intervalles de temps où une corruption de bit ne sera pas critique. Cette analyse a aussi été effectuée avec le même modèle de performance.

L'analyse de la sensibilité d'une application par la méthode d'AVF présente l'avantage d'obtenir rapidement un premier niveau de sensibilité lié à l'application à l'aide d'un simulateur. Cependant, elle nécessite en général des modèles de performance détaillés qui ne sont pas toujours disponibles selon l'architecture du processeur. De plus, on ne peut calculer que l'AVF des composants dont la modélisation existe.

Dans l'état de l'art, on remarque que toutes les études ont été réalisées pour des processeurs de type Alpha 21464 ou des processeurs Intel. La méthode est difficilement transposable à d'autres types d'architecture.

I.4.1.2 Injection de fautes par émulation à partir de descriptions synthétisables

Une méthode communément employée pour estimer les facteurs de décote (« derating ») est la méthode d'injection de fautes dans le composant, afin de simuler l'effet d'une inversion de cellule. Pendant les phases de conception, le produit final n'étant pas disponible, il est toujours possible d'injecter des fautes dans un modèle de processeur pour améliorer ou tester la fiabilité de l'application.

Nous rappelons que l'injection de fautes consiste à perturber le fonctionnement du composant lorsqu'il exécute une application donnée. L'injection de fautes peut être de type logicielle ou matérielle mais indépendamment du type de faute, on peut modéliser l'environnement global d'injection de fautes par trois modules : un générateur de fautes, un module d'injection de fautes et enfin un module d'analyse.

Le rôle principal du module de génération de fautes est de créer une liste de fautes à injecter dans le processeur. L'injection de fautes dans un système complexe peut-être spatiale et/ou temporelle, c'est-à-dire que pour avoir un taux de couverture et de représentativité maximum, il faut injecter des fautes dans chaque élément du processeur et aussi à n'importe quel moment pendant le déroulement de l'application pour tenir compte de l'aspect dynamique. De ce fait, la liste de fautes contient les positions et les cycles où le bit doit être corrompu. Le module injection de fautes doit permettre d'injecter la faute dans un point mémoire. Il doit aussi veiller à ce que le système reste fonctionnel. Enfin, le module de surveillance et d'analyse des résultats génère un rapport sur l'ensemble des résultats de la simulation. Il observe l'impact de l'injection de fautes sur le comportement du processeur, en surveillant la propagation des erreurs ou en examinant le résultat final de l'application. Les erreurs suite à l'injection de fautes peuvent être classées par exemple de cette manière : erreur de résultats, perte de séquencement, pas d'erreurs.

L'injection de fautes peut être réalisée au niveau logiciel à l'aide d'un simulateur de processeur. Mais cela nécessite que le simulateur offre cette fonctionnalité. L'inconvénient d'une telle méthode est la lenteur des simulateurs. L'injection de fautes fondée sur l'émulation répond essentiellement à ce critère de temps. Cette approche utilise un circuit programmable de type Field Programmable Gate Array (FPGA) pour émuler le comportement d'un microprocesseur. Les FPGA présentent l'avantage d'être des composants à forte densité d'intégration. L'injection de fautes par émulation requiert toutefois un modèle RTL synthétisable du processeur (par exemple en Verilog ou VHDL).

On distingue deux approches pour l'injection de fautes par émulation : la reconfiguration du composant et l'instrumentation pour introduire une faute.

L'injection de fautes par reconfiguration consiste à modifier le contenu du fichier de configuration. Ceci peut être fait de manière statique ou de manière dynamique. Dans le cas d'une reconfiguration statique, il est nécessaire de re-synthétiser le design et de le recharger dans le FPGA à chaque modification. Pour une reconfiguration dynamique, le FPGA peut être reconfiguré durant l'exécution de son application ce qui permet de gagner un temps considérable par rapport à la reconfiguration statique qui elle nécessite le rechargement du fichier modifié. Cette technique permet d'injecter des fautes dans les blocs combinatoires et dans les éléments mémoire [ANT03]. Cependant l'inconvénient majeur de cette méthodologie est qu'il est indispensable de bien connaître la configuration du FPGA pour ne modifier que les bits qui sont essentiels pour l'injection. C'est pourquoi nous décrirons seulement la deuxième méthode qui est plus facilement réalisable sans connaissance précise du FPGA : l'instrumentation [LOP07].

Le principe de l'instrumentation consiste à modifier le circuit à étudier afin d'accéder aux éléments mémoires pour modifier leur contenu durant l'exécution du programme. L'instrumentation a pour but de contrôler individuellement chaque bascule pour pouvoir émuler un bit flip sur chacune d'entre elle. Diverses approches d'instrumentation ont été proposées dans la littérature. On ajoute par exemple un « saboteur » pour modifier la valeur d'un signal et éventuellement une « sonde » pour obtenir des indicateurs sur leurs états en surveillant un bus externe.

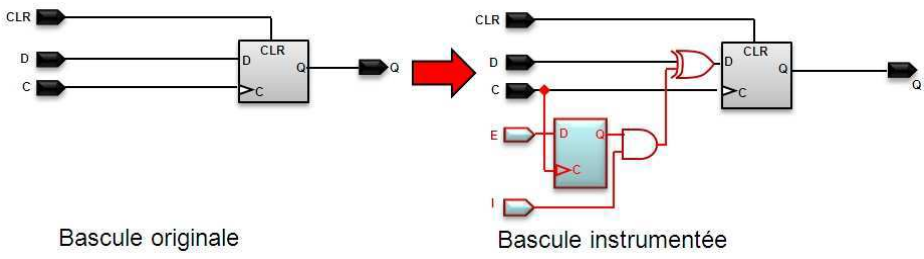


Figure I-11 : exemple d'instrumentation d'une bascule

Le taux d'erreurs estimé à l'aide de cette méthode est assez précis car l'instrumentation permet d'accéder à toutes les cellules du processeur, ainsi on peut injecter des fautes dans toutes les localisations. Cependant, la limitation majeure de cette technique est la disponibilité des modèles HDL des processeurs cibles. Cette méthode est aussi intrusive et elle nécessite d'avoir suffisamment de ressource dans le FPGA pour ajouter l'instrumentation.

De plus, les performances du « design » cible peuvent également être un élément limitant puisque l'ajout d'éléments sur le chemin de la donnée peut venir modifier les temps de propagation. Ainsi, si la fréquence est trop élevée certain éléments ne peuvent pas être adressé à moins de diminuer la vitesse de fonctionnement du « design ». De plus, le temps relativement important pour réaliser une campagne de test exhaustive est aussi à prendre en compte. Pour un système avec n bascules et une application exécutée en N cycles, il faudrait réaliser « N x n » injections. Ceci devient rapidement impossible en pratique sur les circuits actuels. Pour résoudre ce problème, des injections de fautes statistiques sont souvent réalisées et permettent de réduire significativement le temps d'injection de fautes [LEV09]. Le nombre d'injections à réaliser est défini à partir de la marge d'erreurs et du niveau de confiance souhaités.

De récents résultats sur l'injection de fautes par émulation sur le processeur NIOSII sur différentes applications sont présentés dans [GUI12].

Ces tests ont permis de déterminer un pourcentage de sensibilité assez précis pour les mémoires internes du processeur : cache d'instruction, mémoires internes et registres internes. La technique d'émulation est utilisée pour évaluer une sensibilité dynamique du processeur mais aussi pour identifier les parties sensibles du processeur.

I.4.2 Phase de validation de l'implémentation

I.4.2.1 Code Emulated Upset

La méthode de Code Emulated Upset (CEU) [REZ01] est fondée sur l'injection de fautes par l'activation d'un signal d'interruption d'un système matériel externe pour modifier le contenu d'une cellule mémoire en exécutant une routine d'interruption associée.

Le système externe a pour rôle de choisir les paramètres d'injection de fautes c'est-à-dire l'instant d'activation de l'interruption et la cellule mémoire à perturber. Le code CEU [VEL00] permettant le basculement de bits est stocké à une adresse spécifique de la mémoire, il est chargé par le système externe dans la mémoire SRAM du microprocesseur.

Le mécanisme d'injection de fautes est représenté sur la Figure I-12. L'injection de fautes se déroule de la manière suivante. Lors d'une interruption, le programme s'arrête après l'instruction en cours. Le contexte est sauvegardé dans une pile. Le processeur saute vers l'adresse indiquée par le vecteur d'interruption et exécute le code provoquant un *bitflip*. Une fois le traitement de

l'interruption terminé, la routine se termine par plusieurs instructions de retour d'interruption, qui restaurent l'état sauvegardé et font repartir le processeur à l'endroit où il avait été interrompu. Une comparaison des résultats du programme et des résultats d'un même programme non corrompu est ensuite effectuée pour analyser l'impact de la corruption mémoire.

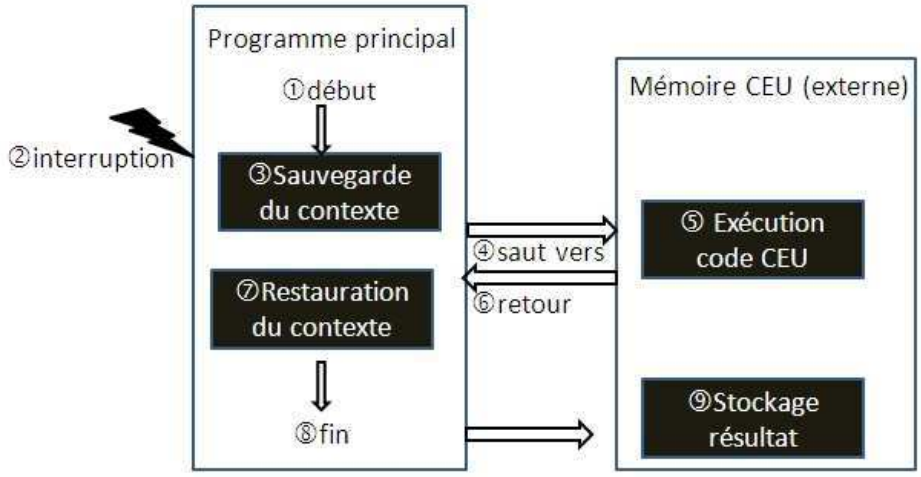


Figure I-12 : Principe du code CEU

Pour provoquer un basculement de bit d'un élément mémoire, le programme doit exécuter certaines étapes :

- lecture du contenu de l'élément mémoire.
- opération de type OU EXCLUSIF entre la valeur lue et une valeur contenant la position du bit à changer.
- écriture de la valeur corrompue dans l'élément mémoire.

Ces tâches sont effectuées grâce au jeu d'instructions du processeur. La Figure I-13 donne un exemple de basculement de bit pour une cellule mémoire d'un processeur LEON. Le « mask » indique la position du bit à changer, R représente un registre temporaire du processeur et « addr1 » l'adresse de la donnée.

De nombreux résultats ont été publiés sur la méthode CEU. Les résultats de [VAL07] montrent que la méthode CEU est applicable au niveau des mémoires cache du processeur. Ainsi, l'injection de fautes sur un processeur LEON a permis d'estimer la sensibilité des éléments mémoires à 8,4 % pour un algorithme de type « bubble sort ». Cette méthode permet donc d'avoir une estimation de la sensibilité de l'application lors de la phase de développement logiciel, lorsque le processeur physique est disponible mais nécessite un dispositif de test assez important.

De plus, les principaux inconvénients de cette technique sont la limitation de la liste potentielle de cellules cibles due au jeu d'instructions du processeur, et l'intrusivité de cette méthode car elle nécessite la modification du code d'application. Le temps nécessaire à l'injection de fautes est relativement court, mais chaque instruction doit être terminée pour prendre en compte une interruption.

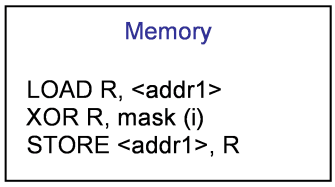


Figure I-13 : Exemple d'inversion de bits dans le processeur LEON

I.4.2.2 Injection de fautes par interface de débogage intégré (on-chip debugger)

La plupart des microprocesseurs modernes comprennent une logique dédiée pour le débogage. Cette fonctionnalité permet au concepteur d'accéder aux ressources internes du processeur à l'aide d'un logiciel spécifique. L'utilisateur a en général accès en écriture et en lecture aux cellules mémoires, de ce fait il est possible d'injecter des fautes dans le microprocesseur.

Le mécanisme global d'injection de fautes reste le même. Il s'agit de provoquer le basculement d'un contenu mémoire pendant que le processeur exécute son application et d'observer le résultat de cette perturbation (Figure I-14).

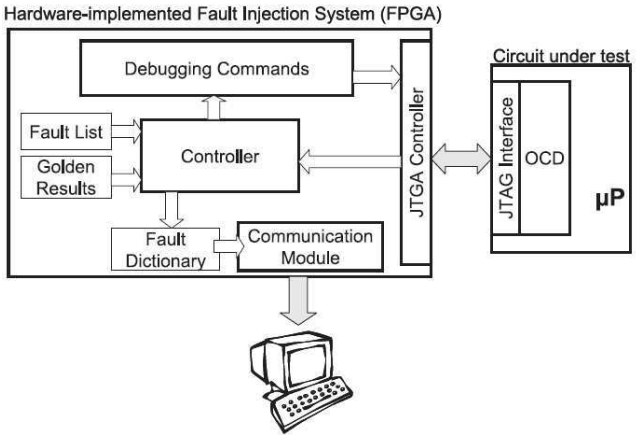


Figure I-14 : Injection de fautes par interface de débogage intégré [FID06]

Le standard communément utilisé pour communiquer avec l'interface de débogage intégré est l'IEEE 1149.1 [IEE01], ou JTAG. Les microprocesseurs avec une interface JTAG de

débogage intégré offrent la capacité d'arrêter l'exécution d'un programme, d'écrire ou lire le contenu d'un registre ou d'une mémoire. C'est pour cela que ce type d'interface est parfaitement adapté à l'injection de fautes dans un microprocesseur. De nombreuses approches utilisant la technique d'injection de fautes par JTAG ont été développées [REB99] [PEN06] [FID06]. Ces méthodes sont applicables à différentes familles de processeurs dès lors qu'il existe un logiciel de débogage qui permet d'accéder à la mémoire.

Le mécanisme d'injection de fautes est constitué d'un module de génération de fautes, d'injection de fautes et d'observation. Le principe est donc à partir d'une liste de fautes, d'exécuter le programme d'application, de stopper le programme, puis de lire la valeur d'une cellule cible et de corrompre sa valeur avant de la réécrire. Puis, on continue d'exécuter l'application pour observer l'impact de la corruption.

La méthode d'injection de fautes par JTAG est connue mais elle contient des limitations au niveau des cellules cibles qui contribuent ainsi à l'imprécision de l'estimation du SER. Certaines interfaces JTAG ne permettent pas de corrompre la valeur des registres d'état du microprocesseur par exemple.

Le temps nécessaire à l'exécution de la méthodologie est assez important, car il nécessite l'arrêt de l'exécution du code source par le processeur pour pouvoir corrompre une valeur et relancer l'exécution du programme.

Il existe dans les processeurs actuels de nouvelles fonctionnalités qui permettent de corrompre une information interne du processeur aléatoirement, comme le P4080 [FRE12] de chez Freescale. Cependant, cette méthode n'est applicable qu'aux mémoires cache du processeur et le caractère aléatoire présente un inconvénient assez important en terme de taux de couverture du test.

De plus, les processeurs modernes ont maintenant des architectures complexes comme un pipeline superscalaire, une exécution dans le désordre des instructions. L'utilisation du mode de débogage, c'est-à-dire l'interruption du processeur, modifie le comportement général du processeur et les instructions ne sont pas exécutées de la même façon en mode normal ou selon l'emplacement du point d'arrêt. Cette méthodologie a été appliquée aux registres et aux cellules mémoires d'un processeur ARM 7 [POR11]. Le taux de couverture de fautes n'est pas indiqué, mais on peut noter que selon le type d'application le taux d'erreurs est diffèrent.

I.4.2.3 Test accélérateur dynamique / test laser

La technique la plus pertinente lorsque le circuit est disponible est d'effectuer des tests en accélérateur de particules. Elle ne requiert pas une connaissance détaillée de la microarchitecture du processeur. Ce test est effectué sur cible réelle, il prend en compte les contraintes réelles de l'application comme le système d'exploitation et l'application avec sa gestion d'entrées / sorties.

L'inconvénient de cette technique est qu'elle ne peut être appliquée que sur cible finale, c'est-à-dire à la fin du développement du produit final.

En outre, prendre en compte le comportement dynamique d'une application nécessite une expérimentation longue et coûteuse pour avoir des résultats pertinents avec un nombre suffisant d'erreurs observées. De plus, les tests doivent être effectués à chaque modification de l'application. Un autre inconvénient est le développement d'un système de test automatique (ATE) comme précisé dans le JESDEC89. Dans le cas des processeurs, l'automatisation de l'équipement de test peut avoir un coût de développement très important.

Pour finir, les tests accélérateurs présentent de nombreuses contraintes expérimentales comme la protection des autres éléments de la carte et la vitesse d'exécution du code. De plus, le contrôle et le pilotage du test sont en général situés à plusieurs mètres du faisceau.

La NASA a listé un certain nombre d'étapes à respecter pour tester les microprocesseurs sous accélérateurs [IRO08].

La première étape est bien sûr la préparation physique du composant. Il s'agit de vérifier que le type de boîtier est bien adapté au type de test accélérateur. Il faut s'assurer que l'implémentation physique de la puce n'empêchera pas la particule de traverser le silicium. Par exemple, pour un test aux ions lourds il est souvent nécessaire d'ouvrir le composant en face avant ou d'amincir la puce. Les processeurs fonctionnant à haute fréquence ont en général un système de dissipation important sur le boîtier du processeur. Il faut aussi vérifier que le radiateur thermique soit retiré de la puce et que malgré toutes ces modifications, le composant reste fonctionnel. De plus, les tests pour le spatial nécessitent des tests sous vide pour une irradiation aux ions lourds, ce qui implique des contraintes de dissipation de chaleur et d'interfaçage avec le système de contrôle. Il est parfois nécessaire de limiter les temps d'exécution de l'application pour ne pas trop chauffer le composant.

La contrainte matérielle réside aussi dans le fait d'avoir un composant qui soit implanté sur une carte électronique et qui soit placé directement en face du faisceau. Le composant doit avoir un système de diagnostic comme le JTAG pour s'assurer qu'il est toujours fonctionnel.

Pour calculer une sensibilité statique, il faut pouvoir accéder au plus grand nombre d'éléments mémoires en écriture et en lecture, c'est-à-dire les mémoires cache, les registres internes …

L'ensemble de l'équipement de test doit en général être testé pour vérifier que le microprocesseur ainsi que le système de contrôle ou de surveillance soit fonctionnel en condition réelle de test sous faisceau : longueur des câbles, bruit de l'enceinte sous tests…

Les processeurs sont de plus en plus performants, ils fonctionnent à des fréquences élevées et leur complexité augmente le risque de « Hang », aussi appelé SEFI. Les SEFI correspondent à des erreurs liées à la perte de fonctionnalité du composant ce qui démontre la nécessité d'avoir des systèmes capables de détecter ces dysfonctionnements mais aussi de pouvoir les « réparer » soit par un simple reset soit par un redémarrage complet du composant sous test. De plus, la difficulté de ce type d'évènement est d'identifier la source principale de la défaillance car lorsque le processeur n'est plus fonctionnel, il est ardu d'accéder à ses ressources internes.

D'autres moyens de test comme le laser peuvent être utilisés pour aider à caractériser la sensibilité dynamique du processeur. Les tests laser ont permis de simuler certains effets radiatifs dans le silicium [MIL06]. Le principe du test laser est de générer un pic de courant transitoire dans le point mémoire pour inverser le contenu. Ce photo-courant est généré par l'apport d'énergie localisée induit par le faisceau laser.

L'inconvénient du test laser est la profondeur de pénétration du faisceau. Pour pallier à cette limitation, l'irradiation par face arrière est souvent employée. Cependant, cela nécessite d'ouvrir le boîtier du composant. L'avantage du test laser est de pouvoir cibler avec précision la zone où injecter une erreur.

I.4.3 Récapitulatif des méthodes

La Figure I-15 résume les différentes méthodes énoncées dans les parties précédentes. Elles sont classées en fonction de leur utilisation dans le cycle de production d'un équipement. Pour chacune des méthodes énoncées, le tableau comporte une évaluation des méthodes selon plusieurs critères :

- le niveau de précision du taux d'erreurs,

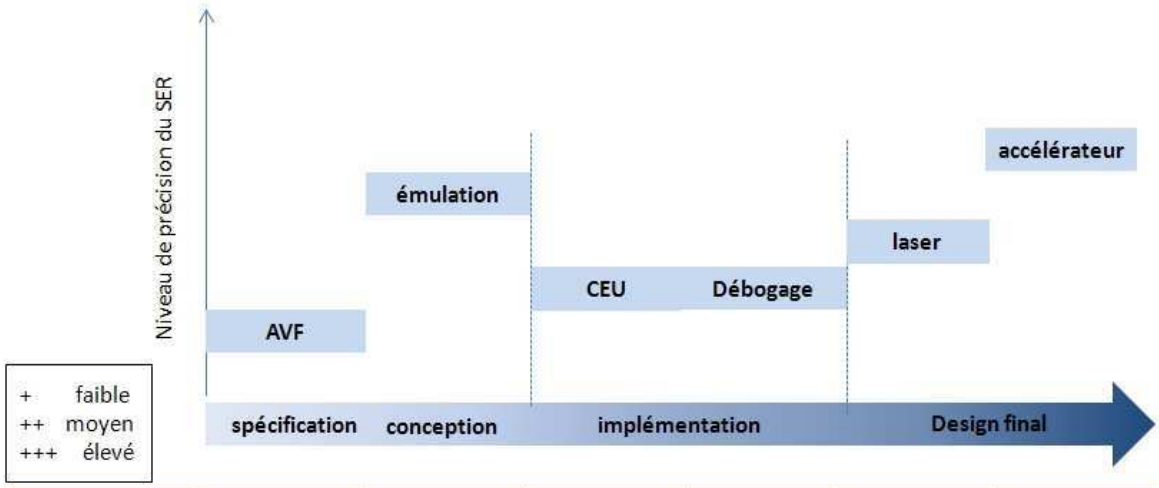
- le temps d'exécution nécessaire de la méthodologie,

- la facilité de mise en œuvre,

- la disponibilité du moyen de test,

- le coût avantageux,

- l'exhaustivité des résultats, la couverture de test.



	Architectural Vulnerability Factor	Injection de faute par émulation	Méthode Code Emulated Upset	Injection de faute par interface débogage intégré	Injection de faute par laser	Test dynamique en accélérateur de particules
Précision	+	+++	+	+	++	+++
Rapidité	++	+	+	+	++	++
Mise en œuvre	+	+++	++	++	+	+
Disponibilité	+	+	+++	+++	++	+
Coût	+++	+++	+++	+++	++	+
Exhaustivité	+	+++	+	+	++	++

Figure I-15 : Résumé des méthodes de prédictions de la sensibilité des processeurs.

Lors de la phase de spécification et de conception, à l'aide d'une plateforme de simulation ou de prototypage, une méthode comme l'Architectural Vulnerability Factor peut être appliquée. Elle permet d'avoir des premiers résultats approximatifs assez rapidement. Néanmoins, cette solution est difficile à mettre en œuvre car elle nécessite une simulation détaillée de la microarchitecture du microprocesseur.

L'injection de fautes par émulation peut être employée lors la phase de conception du produit lorsque le modèle HDL du processeur est disponible ce qui n'est pas le cas pour tous les processeurs. Cependant, la couverture de test pour cette méthode est assez correcte en dépit de la durée de test relativement longue.

Au moment de la phase d'implémentation, lorsque la plateforme matérielle est disponible, les approches CEU ou d'injection de fautes par débogueur sont privilégiées. Elles permettent d'obtenir des résultats sans complexité majeure d'implémentation malgré une faible couverture de test.

Dans la phase finale du produit, les tests accélérateurs et les tests laser offrent la possibilité de conforter le concepteur dans le choix du matériel et du logiciel de son produit. Les conditions de test sont plus proches de la réalité. Mais en pratique, ce type de test n'est pas réalisé par les équipementiers à cause des délais classiques des projets industriels. De plus, la disponibilité des installations et le coût des campagnes de test sont les principaux inconvénients de ce type de test. Les tests laser pallient en partie aux contraintes des tests en accélérateur de particules mais cela exige l'accessibilité à la puce.

En conclusion, cette analyse des différentes méthodes de prédiction montre que la plupart nécessitent des outils spécifiques ou des modèles qui ne sont pas disponibles pour les utilisateurs finaux. La méthode d'injection de fautes par émulation reste la plus exhaustive et la plus précise mais requiert un modèle RTL de processeur. En général, les méthodes d'injection de fautes demandent des durées de test relativement longues.

Les méthodes d'analyse d'architecture reposent toutes sur l'instrumentation de simulateur spécifiques. De plus, ces simulateurs ne sont que des modèles approximatifs fondés sur le jeu d'instructions spécifique. Il n'existe donc pas de méthode capable de fournir à un concepteur un taux d'erreurs précis et exhaustif avec un moyen de test aisément disponible quel que soit le type de microprocesseur.

I.5 Conclusion

Dans ce chapitre, une brève description des environnements radiatifs spatiaux et atmosphériques a été présentée. Les composants électroniques sont soumis aux effets des particules qui peuvent être sources d'erreurs pour un équipement embarqué. Les

microprocesseurs sont au cœur des évolutions technologiques d'aujourd'hui, et leur performance les rend très attrayants pour les systèmes embarqués avioniques.

Au cours de ces dernières années, la capacité de calcul des microprocesseurs n'a cessé d'augmenter, en partie grâce aux mémoires caches. Ces mémoires internes de différents niveaux, proches du cœur de l'unité de calcul, ont permis d'augmenter les vitesses de calcul en améliorant les temps d'accès aux données et aux instructions. Les derniers microprocesseurs de la famille des PowerPC intègrent jusqu'à trois niveaux de mémoire cache. Néanmoins, les mémoires cache sont de grands contributeurs aux taux d'erreurs des microprocesseurs car étant des éléments mémoires ils sont sensibles aux SEUs. L'estimation du taux d'erreurs dû aux rayons cosmiques est une tâche difficile car les microprocesseurs sont des composants très complexes. La notion de sensibilité dynamique d'un microprocesseur a été évoquée dans ce chapitre. Il s'agit de caractériser la sensibilité du processeur en fonction de l'application exécutée.

Le moyen de test classique consiste à exposer le microprocesseur à un faisceau de particules. Il est difficilement réalisable et n'est pas la solution la mieux adaptée pour prédire le taux d'erreurs, qui varie selon l'application. En effet, ce test impliquerait qu'un système développé à base de microprocesseur soit en phase finale de son cycle de développement. Par conséquent tout changement logiciel et matériel éventuel est extrêmement coûteux.

Ainsi, de nombreuses méthodes ont été étudiées pour aider le concepteur dans le choix des mécanismes de protection à implémenter. La plupart de ces méthodes reposent sur l'injection de fautes et demandent une durée de test très longue. D'après l'analyse de l'état de l'art, il n'y a pas de méthode qui permette d'estimer une sensibilité des mémoires caches sans instrumenter un simulateur spécifique ou réaliser une émulation.

Nous allons donc proposer dans le chapitre II, une nouvelle méthodologie de prédiction du taux d'erreurs des microprocesseurs fondée sur l'analyse des mémoires cache à l'aide de simulateurs standards de type « cycle–accurate ».

Chapitre II : Méthodologie de prédiction du taux d'erreurs des microprocesseurs fondée sur l'analyse des mémoires cache

La prédiction du taux d'erreurs des microprocesseurs en environnement radiatif est une tâche difficile à réaliser comme le montre l'analyse de l'état de l'art au chapitre 1. La taille des mémoires cache étant en constante évolution, elles sont un contributeur majeur d'erreurs. Il existe une grande disparité entre les résultats obtenus en test accélérateur selon que l'on utilise une approche statique ou dynamique. Afin de mettre en évidence les difficultés de mise en œuvre d'un test accélérateur sur les microprocesseurs, un test a été réalisé et sera présenté dans la partie II.1. Il met en évidence la nécessité d'avoir un outil industriel capable de fournir un taux d'erreurs plus précis. Le test accélérateur a permis de déterminer les limites d'une telle approche sur des composants complexes comme un microprocesseur. Ainsi, une méthodologie basée sur les mémoires cache a été mise en place, et est présentée dans ce chapitre. Le fonctionnement des mémoires cache est étudié en détail afin d'obtenir des facteurs de décote. Le premier facteur de décote découle directement de l'architecture de la mémoire cache, tandis que le second dépend du comportement de la mémoire cache lors de l'exécution de l'application.

II.1 Étude de cas : limites des tests en accélérateur de particules

Les tests en accélérateur de particules sont aujourd'hui insupportables pour estimer les sections efficaces à différentes énergies et déterminer les taux d'erreurs réel. L'analyse de l'état de l'art, présentée au paragraphe I.4.2.3, a révélé que les tests en accélérateurs de particules sont le moyen qui permet d'obtenir les résultats les plus proches de ceux obtenus en environnement réel. Cependant, afin de mettre en évidence les contraintes liées à l'irradiation des microprocesseurs exécutant une application, une campagne de test a été réalisée à l'UCL (Université Catholique de Louvain La Neuve, Belgique).

II.1.1 Présentation du cas d'application

Les tests d'irradiation aux protons 63 MeV ont été effectués sur un microprocesseur de la famille des PowerPC, le MPC8349E [FRE06]. Le choix du microprocesseur s'est porté sur cette cible car les PowerPC sont une cible d'intérêt dans l'avionique. De plus, ce type de composant possède une grande quantité de mémoires cache, ce qui en fait un candidat idéal pour notre étude. Ce microprocesseur de technologie 130 nm possède des mémoires cache L1 d'instructions et de données de 32 KB chacune protégée par un mécanisme de parité. Ce microprocesseur ne possède pas de niveau L2 de cache. Cependant, il contient 157 registres dont 32 registres généraux (GPR) de 32 bits ainsi que 32 registres pour les variables de virgules flottantes (FPR) de 64 bits (Tableau II-1).

Tableau II-1 : Répartition des éléments mémoires du MPC8349

Éléments mémoires sensibles aux SEU	Nbre bits	Taux d'occupation
Cache de données L1	296960	48,8 %
Cache d'instructions L1	296960	48,8 %
MMU	8192	1,3 %
Registres	6048	1,0 %

Le système expérimental est constitué d'une carte d'évaluation commerciale de chez Freescale : MPC8349E-miTX-GP [FRE06-1]. L'équipement de test utilisé est constitué d'un PC de contrôle et d'une sonde JTAG reliée à la carte de test, afin de permettre l'accès en lecture aux registres ainsi qu'aux mémoires cache du PowerPC.

Ce test a deux objectifs : le premier est de déterminer un taux d'erreurs statique lié aux mémoires caches, puis le second est d'évaluer la sensibilité dynamique du microprocesseur exécutant une application.

II.1.2 Caractérisation de la sensibilité statique

Dans un premier temps, un test statique a été réalisé pour évaluer la sensibilité des éléments mémoires comme les mémoires cache et les registres internes.

Les normes JESD89A [JESD89A] préconisent de tester les composants mémoire avec des motifs de type « échiquier » (« 0xAAAAAAAA ») ou « échiquier inverse » (« 0x55555555 »), c'est pourquoi les mémoires cache et les registres ont été testés avec ces motifs. Un autre motif de type

échiquier combiné (« 0xAA55AA55 »), a aussi été testé. Le test consiste à initialiser les mémoires cache et les registres avec un motif prédéfini, puis à exposer le composant au faisceau de particules. Le motif est ensuite relu à la fin de l'irradiation afin de déterminer le nombre d'erreurs observé. Le composant a été irradié avec deux flux : un flux de $2x10^7$ proton/cm²/s et un flux plus élevé à $2x10^8$ proton/cm²/s.

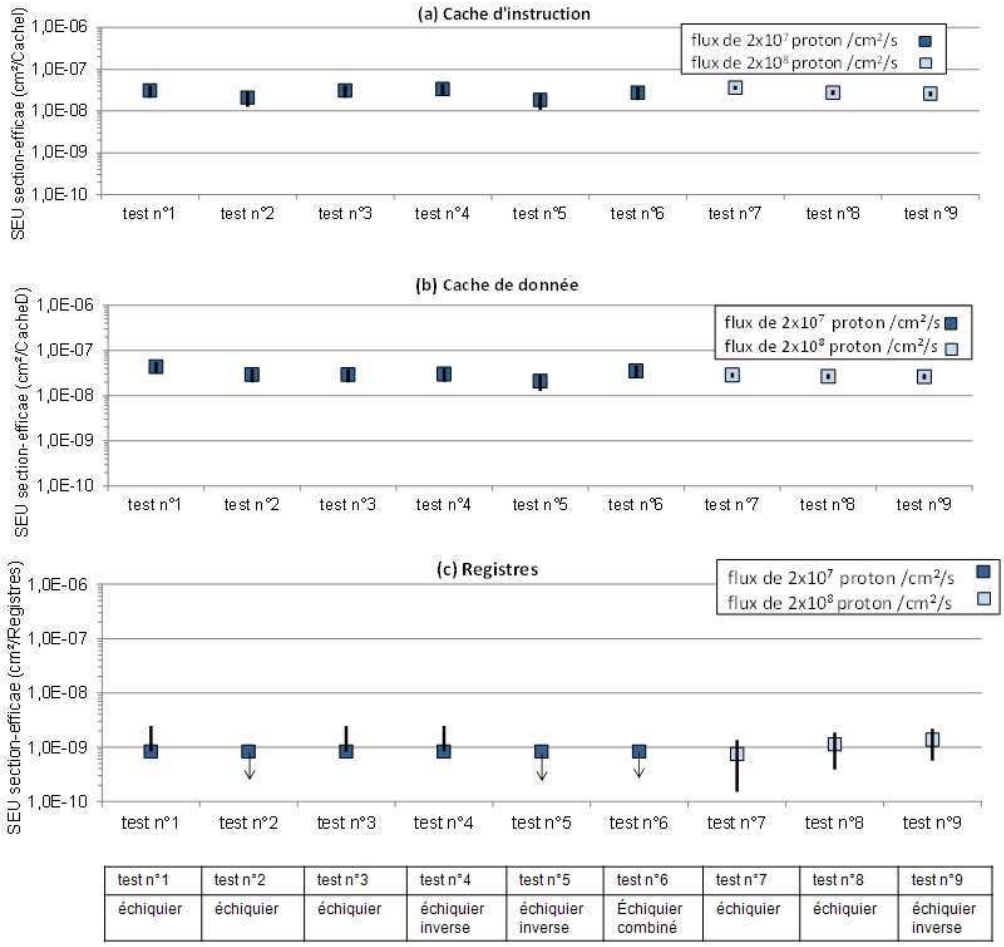


Figure II-1 : Section efficace statique du MPC8349E (a) cache d'instructions, (b) cache de données et (c) registres

La Figure II-1 présente les résultats obtenus lors des tests : les tests n°1, 2 et 3 ont été effectués avec un flux de $2x10^7$ proton/cm²/s et le motif échiquier, les tests n°4 et 5 avec un flux de $2x10^7$ proton/cm²/s et un motif échiquier inverse. Les tests n°7 et 8 ont été réalisés avec un flux de $2x10^8$ proton/cm²/s et le motif échiquier alors que le n° 9 l'a été avec le motif échiquier inverse. Les sections efficaces sont représentées en cm² par élément testé dans le microprocesseur : cache d'instructions, cache de données et registres. Les résultats montrent que le cache L1 de données et le cache L1 d'instructions ont une section efficace similaire en

moyenne de $9{,}7x10^{-14}$ cm²/bit comme indiqué dans la Figure II-1. Il est indispensable de noter qu'aucun évènement multiple n'a été observé durant les tests.

Les sections efficaces obtenues pour les registres sont représentées en Figure II-1 (c) : la proportion du nombre de bits constituant les registres étant relativement faible, cela explique que les barres d'erreur sont assez importantes pour ce type de test.

Néanmoins, l'irradiation du microprocesseur aux protons de 63 MeV, a permis d'estimer une section efficace statique pour les mémoires caches et donc de calculer un taux d'erreurs pire cas (SER_{RAW}).

II.1.3 Caractérisation de la sensibilité dynamique : influence des mémoires cache

Afin de mesurer le nombre d'erreurs lorsque le microprocesseur exécute une application, un test dynamique a été réalisé à un flux de $2x10^7$ proton/cm²/s^{-1}. Le principe du test dynamique est d'irradier le composant alors qu'il exécute une application et d'en observer l'effet sur la sortie du résultat final. L'objectif est d'évaluer l'impact des mémoires cache sur la sensibilité du microprocesseur. Ainsi, deux types de tests ont été effectués avec et puis sans les mémoires cache.

L'objectif du test dynamique est aussi d'évaluer l'influence des différentes données résidant en mémoire cache (instructions, ou données utilisées par le calcul). Ainsi pour évaluer le comportement des caches, un programme de multiplication de vecteurs a été développé. Ce programme, consistant à multiplier deux vecteurs de mille mots de 32 bits, a été implémenté de deux manières afin de montrer l'influence du cache d'instructions. La première application (appli 1) a été codée avec une boucle « for » et la deuxième (appli 2) sans boucle « for », le code s'exécute donc de manière séquentielle. Ainsi, avec la première application, le cache d'instructions L1 est rempli partiellement mais les mêmes instructions sont lues constamment. La deuxième application permet d'occuper totalement le cache. Pour les deux applications, le cache de données est rempli entièrement. Deux types d'erreurs sont observés : le premier est un résultat du calcul erroné et le second est la perte de fonctionnalité du microprocesseur.

La Figure II-2 illustre la section efficace dynamique obtenue pour les différents tests. Le test n°1 a été réalisé à l'aide de l'appli1 et le test n°2 avec l'appli 2. Ces deux tests ont été réalisés sans

mémoire cache. Le test n°3 a été effectué avec l'appli 1 alors que les tests n° 4 et 5 avec l'appli 2, ces trois tests comprennent les mémoires cache.

Le test n°2 ne permet pas de conclure car aucun n'événement n'a été observé. Cependant le test n°1 montre, malgré des barres d'erreurs importantes, qu'il y a très peu d'erreurs lorsque les mémoires cache ne sont pas utilisées. Les caches sont donc un contributeur majeur d'erreurs.

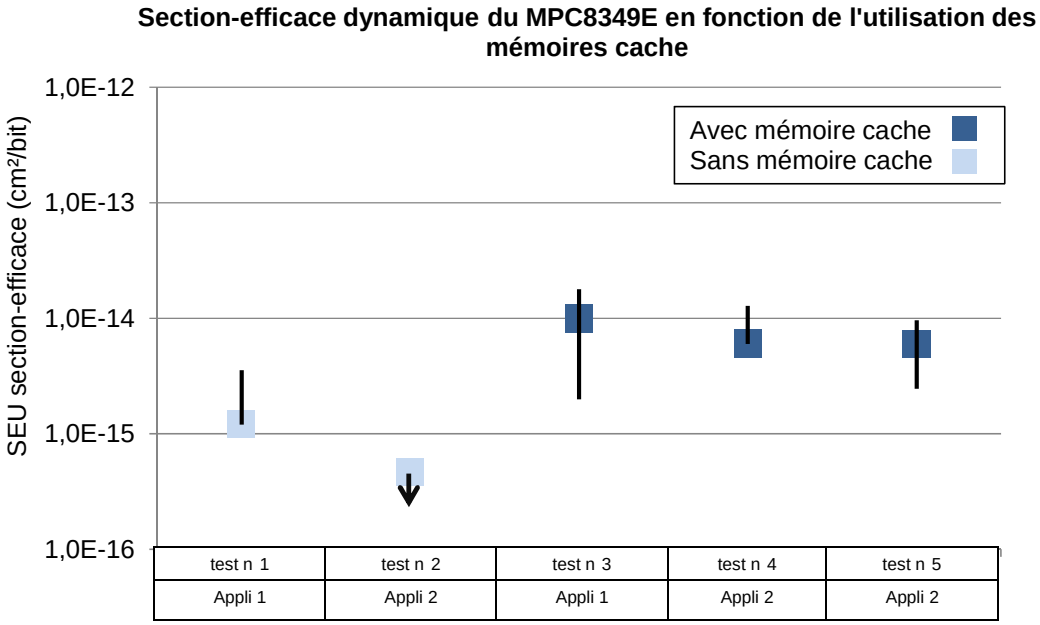


Figure II-2 : Résultat des tests dynamiques du MPC8349E

De plus, le taux d'erreurs des microprocesseurs dépend fortement de l'utilisation de la mémoire cache. Pour un temps d'exposition à peu près semblable, on peut dire que l'application 1 est plus sensible que l'application 2, ceci étant probablement dû à ce que la mémoire cache d'instructions de l'application 1 est souvent sollicitée (Tableau II-2). Ceci reste une hypothèse car les barres d'erreurs sont assez importantes pour les tests effectués. En outre, il est assez difficile d'évaluer la sensibilité des registres. Néanmoins, ce test accélérateur a démontré qu'il est difficile d'obtenir une estimation de la sensibilité des différentes applications en test accélérateur pour un microprocesseur.

Tableau II-2 : Nombre d'erreurs durant l'irradiation

Test	1	2	3	4	5
Applications	Appli1	Appli2	Appli1	Appli2	Appli2
Nombre d'erreurs	1	0	6	3	11
Nombres de bits	6048	6048	600960	600960	600960
Flux (proton/cm²/s)	$2x10^7$	$2x10^7$	$2x10^7$	$2x10^7$	$2x10^8$
Temps total	1 min	3 min	50 s	41 s	15 s

La Figure II-3 représente le taux d'erreurs SER estimé au sol en FIT (20 neutron/cm²/h) pour les 5 tests effectués et le taux d'erreur pire cas. Pour les tests n°1 à 5, le taux d'erreur est calculé en prenant en compte la section efficace dynamique de chacun des tests. Le taux d'erreurs pire cas correspond aux nombres d'erreurs théoriques en fonction de la section efficace statique. La différence entre le nombre d'erreurs observé et le nombre d'erreurs théorique est relativement importante. Cette analyse montre que les mémoires caches sont d'importants contributeurs au nombre d'erreurs. Il est donc important d'avoir un outil qui donne une indication plus précise du taux d'erreurs car il est difficile à estimer.

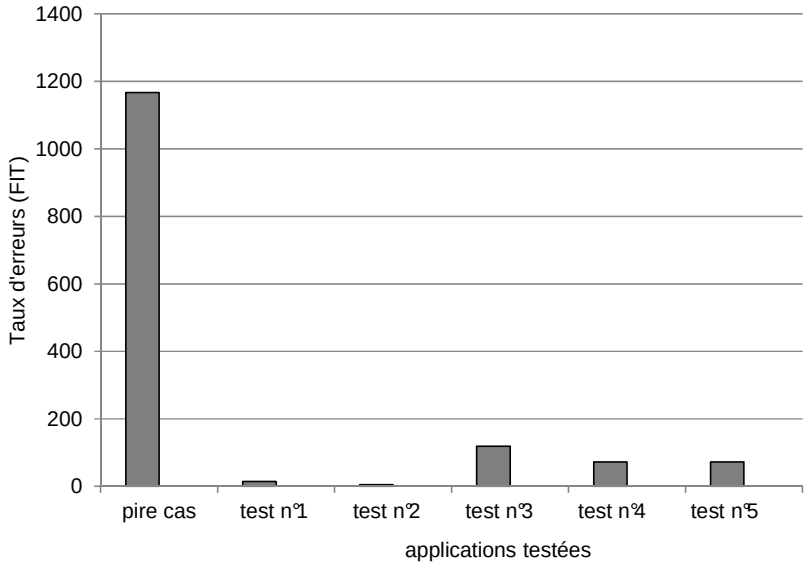


Figure II-3 : Comparaison des taux d'erreurs pire cas estimés par rapport aux taux d'erreurs mesurés en accélérateur pour chaque test.

Cette étude de cas a montré que l'estimation du taux d'erreurs pour une application est difficile à quantifier et assez complexe à mettre en œuvre. En effet, le taux d'erreurs pire cas

obtenu grâce à la section efficace statique est largement supérieur aux taux d'erreurs dynamiques obtenus en accélérateur. Il y a donc nécessité d'avoir un outil permettant d'estimer le facteur de décote entre une sensibilité statique et une sensibilité dynamique. Les tests accélérateurs ont aussi montré que le principal élément contributeur aux taux d'erreurs est bien la mémoire cache. C'est pourquoi la mémoire cache est notre point focal dans l'étude.

De plus, les tests accélérateurs sont assez lourds et difficile à mettre en œuvre, l'incertitude des résultats obtenus est un paramètre important et les barres d'erreurs peuvent être grandes. Il est donc primordial pour l'ingénieur devant développer une application sous fortes contraintes environnementales de disposer d'un outil capable de raffiner cette prédiction du taux d'erreurs. Cet outil doit être basé sur l'architecture du cache ainsi que sur la prise en compte de la sollicitation de celui-ci par une application. Il doit être mis en place tôt dans la phase de conception, lorsque que la cible matérielle n'est pas accessible, donc à l'aide d'un simulateur permettant de simuler le comportement de la cache. Dans la partie suivante, nous allons étudier en détail le fonctionnement des mémoires caches.

II.2 Analyse du fonctionnement des mémoires cache

Comme indiqué dans le chapitre I, les mémoires cache sont de petites mémoires implantées dans la puce même du processeur. Leur rôle principal est d'augmenter la vitesse d'exécution des calculs du processeur. Elles permettent de stocker des données fréquemment utilisées et de diminuer les temps d'accès aux données qu'il faudrait normalement rechercher en mémoire externe.

II.2.1 Description des mémoires cache

II.2.1.1 Organisation du cache

Une mémoire cache est en général composée d'une partie pour le stockage de données, d'une partie pour le stockage d'adresse et d'informations complémentaires sur les données, appelées bits d'état.

La partie données est constituée d'un ensemble de lignes, aussi appelées blocs, et chacune de ces lignes contient un ensemble de mots de 8, 16 ou 32 bits selon la configuration du cache. Le

nombre de mots contenu dans une ligne est aussi dépendant de la configuration du cache : souvent 4, 8, 16 ou 32 selon l'architecture.

Chaque ligne est associée à une étiquette appelée « tag » et l'ensemble des tags constitue la partie stockage d'adresse. Le microprocesseur cherche une donnée dans la mémoire cache en comparant l'adresse de cette donnée avec le tag de la ligne de cache. Les lignes de cache sont aussi associées à des bits d'état donnant diverses informations sur la ligne en mémoire cache comme sa validité, la modification éventuelle de la donnée …

Une ligne de mémoire principale est placée dans le cache selon une organisation spécifique. Chaque adresse est liée à un ou plusieurs emplacements précis de la mémoire cache. L'adresse d'une donnée en mémoire principale est organisée en plusieurs champs et chacun de ces champs va permettre de chercher précisément la donnée en mémoire cache. Cette adresse est généralement composée d'une partie contenant le numéro de la page de mémoire cache (« index ») pour une mémoire organisée en plusieurs pages, une autre contenant le tag, et enfin l' « offset » qui correspond à la position de la donnée dans la ligne. Lors d'une recherche de donnée en cache, le microprocesseur va d'abord sélectionner une page de la mémoire cache à partir de la valeur du champ index contenue dans son adresse. Puis, dans la page choisie, il va comparer les champs tag de chacune des lignes avec le tag contenu dans son adresse. Si le tag correspond, le microprocesseur sélectionne ensuite le mot de la ligne identifié par le tag, à l'aide de la valeur indiquée dans le champ offset.

II.2.1.2 Associativité

La mémoire cache étant de taille inférieure à la mémoire principale, elle ne peut pas contenir toutes les informations et il est nécessaire de définir une correspondance entre les informations présentes dans le cache et celles présentes en mémoire principale. Il existe donc différentes organisations de mémoire cache afin de déterminer la manière dont les blocs vont être chargés. En général, trois types d'accès se distinguent :

- Pour une mémoire **cache à accès direct**, chaque ligne de la mémoire principale a une adresse unique en mémoire cache. Ce type de cache présente l'avantage de rechercher facilement une donnée en mémoire cache. Cependant, lorsque le processeur utilise deux parties de la mémoire qui utilisent le même emplacement cache, cela peut provoquer une perte de performance due au rechargement du cache.

- Pour une mémoire **partiellement associative**, chaque adresse aura un nombre défini d'emplacement dans la mémoire cache. Un cache n-associatif bénéficie de n possibilités de chargement de ligne. Ce type de mémoire est en général préféré car il correspond à un bon compromis entre la complexité et la rapidité de la recherche de donnée.

- Dans le cas d'un **cache pleinement associatif**, chaque donnée peut être mise dans n'importe quelle ligne du cache, ce qui permet d'éviter les conflits de placement mais augmente le temps de recherche des données.

II.2.1.3 Les politiques de remplacement

Il existe différents types d'algorithmes dits politiques de remplacement de cache pour déterminer la ligne qui doit être remplacée lorsque le processeur a besoin d'accéder à d'autres données alors que le cache est occupé (tout au moins sur les lignes pouvant être occupées par la donnée à charger).

Les politiques de remplacement des mémoires cache jouent un rôle essentiel car elles permettent d'anticiper les données qui seront utilisées par le microprocesseur. Cependant, la complexité de l'algorithme de remplacement engendre un surcoût de calcul. Il existe plusieurs politiques de remplacement des mémoires cache comme le tirage aléatoire, FIFO (First In First Out)… Le plus couramment utilisé de nos jours est l'algorithme du moins récemment utilisé, ou LRU (Least Recently Used). Cet algorithme consiste à remplacer la ligne dont le dernier accès est le plus ancien. Le pseudo LRU est un algorithme LRU amélioré basé sur un arbre de décision binaire.

II.2.1.4 Cas d'étude : mémoires cache du PowerPC e300

L'organisation de la mémoire cache L1 de données de l'architecture interne d'un microprocesseur de type PowerPC e300, telle qu'indiquée dans la documentation de Freescale [FRE06], est donnée sur la Figure II-4. Il s'agit d'un cache de données de niveau 8 d'associativité et de politique de remplacement de ligne de type pseudo LRU. Une ligne de cache est composée de 8 mots de 32 bits, de bits d'état et d'un champ associé au tag. Lorsque le microprocesseur veut accéder à une donnée en cache, il décompose l'adresse de cette donnée en plusieurs champs pour retrouver tout d'abord la page correspondante puis la ligne et enfin le mot. Ainsi, pour une adresse de 32 bits (de A_0 à A_{31} bit), les bits A_{20} à A_{26} représente le numéro de page à sélectionner,

les bits A_0 à A_{20} correspondent au tag à comparer au tag de chaque ligne, et les bits A_{27} à A_{31} indiquent le mot à choisir dans la ligne.

A_0 A_{19}	A_{20} A_{26}	A_{27} A_{31}
Sélection de la ligne	Sélection de la page	Sélection du mot

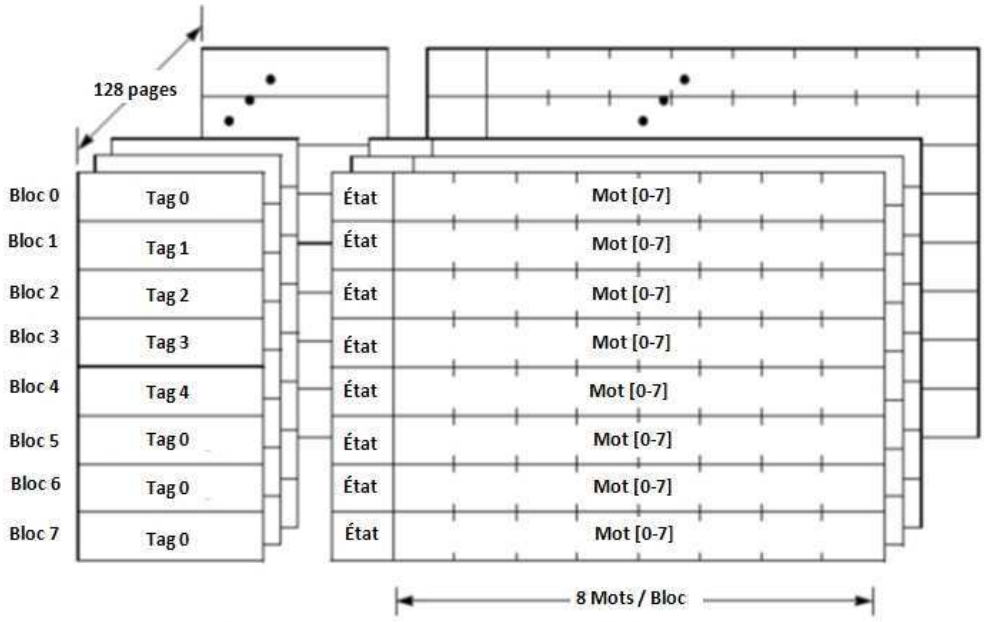


Figure II-4 : Organisation de la mémoire cache de données d'un processeur PowerPC e300

II.2.2 Fonctionnement

Comme indiqué précédemment, la mémoire cache contient les données fréquemment utilisées par le CPU. Lorsque ce dernier a besoin d'une donnée stockée en mémoire principale, il va tout d'abord vérifier que cette donnée n'est pas au préalable stockée en mémoire cache. À l'issu de cette demande, deux possibilités sont envisageables :

- La donnée est dans la mémoire cache ce qui se traduit par un accès réussi appelé « **cache hit** ». Le CPU peut alors accéder à cette donnée en mémoire cache directement.

- La deuxième possibilité est que la donnée n'est pas stockée en mémoire cache. Il y alors un échec lors de l'accès, dit « ***cache miss*** » et le processeur va rechercher la donnée en mémoire principale.

L'évènement de type « *cache miss* » est géré différemment selon le type et la configuration de mémoire cache, mais dans la plupart des cas, un « *cache miss* » engendre un remplissage de la mémoire cache avec la donnée manquante en provenance de la mémoire principale.

En résumé, on peut accéder à une ligne de cache de plusieurs manières dont les principaux accès sont :

- l'écriture lors d'une opération de stockage en assembleur,

- la lecture d'une donnée lors d'une opération de chargement en assembleur,

- le remplissage d'une ligne de cache par le contrôleur de cache lors d'un *cache miss*,

- l'éviction d'une ligne de cache avant la copie en niveau inférieur

Le fonctionnement des mémoires cache est fondé sur le principe de localité spatiale et temporelle :

- Le principe de localité spatiale consiste à dire qu'une donnée dans une ligne de cache sera sûrement utilisée. Ainsi le chargement de la mémoire cache s'effectue ligne par ligne c'est-à-dire que lorsque le CPU a besoin d'accéder à une donnée, il charge aussi les données voisines.

- Le principe temporel dit qu'une donnée sollicitée sera sûrement sollicitée de nouveau dans un intervalle de temps assez court.

Ces principes garantissent l'intérêt que la mémoire cache garde les données récemment utilisées par le CPU et les données adjacentes.

La hiérarchie d'une mémoire cache est en général organisée en plusieurs niveaux pour offrir une meilleure performance au processeur. Plus la mémoire cache est petite, plus elle permet d'accéder rapidement à la donnée. De ce fait les concepteurs ont rajouté un deuxième niveau de cache pour permettre de stocker d'autres données. Le premier niveau de cache est appelé niveau de données L1. Il est en général très proche du CPU que celui-ci puisse accéder rapidement aux données. Ce niveau 1 de cache est en général composé de deux mémoires cache différentes : un cache contenant les données et un cache contenant les instructions. Le cache d'instructions n'est accessible par le processeur qu'en lecture alors que l'accès en cache de données peut être en écriture ou en lecture.

La performance d'une mémoire cache se mesure par son taux de *cache miss* : plus ce taux est faible plus le cache est optimisé. On classifie les caches miss selon trois catégories :

- Le *cache miss* de première référence (« compulsory miss »). Ce défaut de cache est inévitable, il s'agit de charger la donnée ou la ligne pour la première fois.

- Le *cache miss* de capacité (« capacity miss »). Ce défaut de cache traduit le fait que la mémoire cache ne peut contenir toutes les données utilisées dans le programme. Il peut être réduit en augmentant la taille de la mémoire cache.

- Le *cache miss* de conflit (« conflict miss »). Ce défaut de cache est dû au fait qu'une ligne de cache a été retirée pour pouvoir charger une autre ligne de même référence. Ce défaut peut être corrigé en augmentant l'associativité de la mémoire cache.

Cette partie a permis d'introduire les notions indispensables dans la suite du manuscrit. Elle permet aussi de mettre en évidence que les mémoires cache sont des éléments essentiels car lorsque les données sont disponibles, le microprocesseur accède directement à leur contenu sans accès à la mémoire principale. De plus, la corruption d'un bit ou plusieurs bits peut entrainer des erreurs relativement importantes dans l'exécution du programme pour le cache d'instructions qui n'est accessible qu'en lecture. D'autre part, du fait de leur petite taille, les mémoires caches ont un comportement dynamique assez important, ce qui est représenté par des évènements de type *cache hit* et *cache miss*.

II.3 Présentation de la méthodologie générale proposée

La méthodologie proposée permet d'obtenir une estimation plus précise du taux d'erreurs aux radiations des microprocesseurs. En effet, en tenant compte de l'application exécutée, de l'environnement et de la sensibilité de la technologie, le taux d'erreurs peut être raffiné. Cette méthode consiste à obtenir dans une première partie la sensibilité intrinsèque de la technologie du composant et dans une seconde partie, analyser l'application et l'architecture du composant pour avoir une sensibilité dynamique.

Dans cette partie, nous proposons d'introduire la méthodologie générale qui a été développée en présentant les deux étapes fondamentales : la sensibilité de la technologie et la sensibilité dynamique.

II.3.1 Sensibilité de la technologie

Dans le chapitre 1, la notion de sensibilité statique pour quantifier le taux d'erreurs en environnement radiatif pour un microprocesseur a été introduite. Elle est liée à la sensibilité de la

technologie du composant. La sensibilité statique peut s'estimer de plusieurs manières dont les principales sont évoquées ci-dessous :

-Tout d'abord, la section efficace liée aux SEU peut être obtenue par irradiation du composant à un flux de particules donné (neutron, proton, ions lourds …).

-Puis, il existe aussi des outils de prédiction comme MC DASIE [WEU11] qui nécessite en entrée des bases de données nucléaires et des informations simples relatives au design des structures et composants à semi-conducteurs. Il permet de prédire la sensibilité d'un composant dans un environnement radiatif donné ou de la déduire pour un autre type d'environnement.

-Enfin, la dernière méthode pour évaluer la section efficace statique est d'obtenir directement cette donnée auprès des fabricants, si ce dernier a réalisé des tests expérimentaux et est prêt à les partager [SAN08].

À partir de la section efficace du composant, pour une particule donnée, la sensibilité pire cas $SER_{RAW,}$ peut être estimée pour un environnement donné à partir de sa fluence moyenne selon l'équation suivante (Eq. II-1) :

$$SER_{RAW} = \sigma\,(cm^2/bit) \times particule\ fluence\ (particule/cm^2) \qquad \textbf{Eq. II-1}$$

D'après l'IEC [IEC12], le calcul de sensibilité d'un microprocesseur en environnement atmosphérique est établi en multipliant la section efficace par un flux de neutrons de 8000 n/cm² pour une énergie supérieure à 1-10 MeV. Le flux de neutrons est défini pour une valeur nominale pour une altitude de 12,2 km et une latitude de 45°. L'IEC recommande d'appliquer des marges au calcul du taux d'erreurs liées entre autres à la précision ou à la provenance de l'estimation de la section efficace (Eq. II-2).

$$SER_{RAW}\ (bit\ /\ h) = \sigma\,(cm^2\ /\ bit) \times 8000\ (n/\ cm^2/h) \times Marges_{IEC} \qquad \textbf{Eq. II-2}$$

La section efficace liée aux SEU comprend aussi la sensibilité du composant aux évènements multiples. En intégrant le calcul de section efficace à un environnement donné, on peut estimer le taux d'erreurs statique SER_{RAW}.

II.3.2 Sensibilité dynamique

L'estimation de la sensibilité dynamique consiste à prendre en compte l'application et son influence sur le comportement des mémoires cache. Ainsi, en appliquant des facteurs de décote aux taux d'erreurs pire cas, une estimation plus raffinée de la sensibilité des éléments mémoires peut être réalisée.

Le premier facteur de décote est lié à l'architecture des mémoires cache. Ce premier facteur de décote « Π_{arch} » est fondé sur les mécanismes de protection et les politiques d'écriture de cache.

Le facteur de décote lié à l'application « Π_{appli} » donne un pourcentage de sensibilité des mémoires cache pour une application et une cible données. Ce paramètre est obtenu en exécutant l'application et en analysant le contenu du cache pour obtenir une liste de cellules sensibles. Ceci nous permet de calculer plus précisément la sensibilité du processeur (Eq. II-3).

$$SER(FIT) = SER_{RAW} \times \pi_{Arch} \times \pi_{Appli} \qquad \textbf{Eq. II-3}$$

II.3.1 État de l'art des méthodes d'analyse de cache

Plusieurs études [SRI06] [BIS05] [WAN09] ont été développées pour identifier les données critiques des mémoires cache. Biswas *et al.* ont par exemple introduit le concept de l'analyse de durée de vie pour déterminer l'AVF d'une structure de cache. Toutes ces approches identifient des données critiques et non-critiques dans les mémoires cache. Cependant, elles reposent sur l'instrumentation de simulateurs, comme Simplescalar [BUR97], pour l'observation du contenu d'une ligne de cache. De plus, la plupart des simulateurs de processeurs étant des logiciels propriétaires, il est difficile pour un utilisateur final de mettre en œuvre les méthodes de l'état de l'art. En outre, d'autres techniques sont fondées sur l'instrumentation spécifique des modèles RTL pour surveiller les accès en cache. Ces techniques peuvent être très contraignantes car ces modèles ne sont pas toujours disponibles.

De plus, les méthodes présentes dans l'état de l'art ne prennent pas en compte dans leur calcul de taux d'erreurs, les différentes tailles de données, les différents mécanismes de protection, la technologie du composant ainsi que l'environnement dans lequel se situe le processeur.

Il est donc nécessaire de définir une méthode qui ne nécessite pas l'instrumentation des simulateurs et n'a pas besoin d'un modèle de processeur de niveau de détail supérieur.

II.4 Facteur de décote lié à l'architecture du cache : π_{Arch}

Dans cette partie, les deux principales caractéristiques des architectures de mémoires cache sont étudiées afin d'expliquer l'estimation du premier facteur de décote. Dans la première partie, les différentes politiques d'écriture des mémoires cache sont décrites et dans la seconde, les principaux mécanismes de protection des mémoires sont évoqués. Enfin, un calcul du facteur de décote lié à l'architecture du cache est appliqué à un processeur.

II.4.1 Politique d'écriture dans les mémoires caches

Une mémoire cache est une unité de stockage pour les données fréquemment utilisées. Cette mémoire est une mémoire temporaire supposée cohérente et complétement transparente au niveau utilisateur. Ce qui signifie que lorsqu'une ligne de cache est modifiée, la mémoire principale doit aussi avoir son contenu modifié. Il existe des politiques de gestion des écritures visant à optimiser les performances des microprocesseurs en optimisant les durées d'écritures.

Les deux politiques d'écritures couramment utilisées sont la politique dite « ***write-through*** », l'écriture immédiate, et la politique dite « ***write-back »***, écriture différée.

La manière la plus simple de garantir une cohérence de cache est l'implémentation d'un cache en mode « *write-through* », le contenu de chaque cache de niveau bas étant recopié immédiatement en niveau supérieur ainsi qu'en mémoire principale. Lors d'une opération d'écriture, le microprocesseur écrit donc cette valeur dans tous les niveaux de mémoires internes et ainsi qu'en mémoire principale. Ceci permet d'assurer la synchronisation des mémoires cache et de la mémoire principale. Ce type de mémoire offre l'avantage d'avoir une gestion d'écriture relativement simple. Cependant, le fait d'écrire une donnée plusieurs fois est assez couteux en terme de temps, par exemple lorsqu' une même donnée est constamment modifiée.

Un cache configuré en mode « *write-back* » a une politique d'écriture un peu plus complexe. Cela implique que le microprocesseur n'écrit pas immédiatement en mémoire principale la valeur d'une donnée modifiée en cache. Cette configuration requiert un bit additionnel, appelé « dirty bit », mentionnant l'état de la ligne ou du mot (modification ou non de la donnée). Lorsque le

processeur remplace une ligne modifiée (« dirty bit » actif), il recopie cette ligne en mémoire principale avant de la supprimer de son cache. Ce type de mémoire est généralement plus avantageux en termes de performance car il n'y a pas d'écriture systématique de la donnée et on évite ainsi les écritures intempestives. En effet, une donnée qui est souvent modifiée en cache n'implique pas une écriture systématique en mémoire principale. En revanche, ce type de configuration présente un inconvénient majeur dans le cas de ressources partagées, ce qui est souvent le cas pour les systèmes multicœurs par exemple. La cohérence du cache doit être maintenue lorsque la mémoire principale est sollicitée par plusieurs processus qui s'exécutent en parallèle et qui partagent le même espace mémoire.

La plupart des microprocesseurs multicœurs implémentent un protocole qui leur permettent de s'assurer de la cohérence des données du cache, appelé protocole de cohérence de cache MESI (Modified Exclusive Shared Invalid) [FRE12]. Il est fondé sur les différents états qu'une ligne de cache peut avoir :

- modifié : un des CPU a modifié le contenu de la ligne de cache et cela implique que c'est la seule valeur à jour de la donnée,

- exclusive : le contenu de la ligne n'est pas modifié mais seul ce CPU a chargé cette ligne,

- partagé : le contenu de la ligne n'a pas été modifié, et cette ligne peut résider dans d'autres mémoires caches,

- invalide : la ligne de cache n'est pas utilisée car elle n'est pas valide.

Lorsque le microprocesseur souhaite écrire une donnée en mémoire cache et qu'il y a un *cache miss*, il existe deux types d'opérations couramment utilisées : allocation en écriture, et non-allocation en écriture. Le principe d'allocation en écriture consiste, lors d'une écriture d'une donnée qui n'est pas en cache, à allouer un emplacement en mémoire cache pour cette donnée. Pour un cache configuré en mode « non-allocation en écriture », la donnée est écrite directement en mémoire principale et n'est pas chargée en cache.

Les mémoires caches peuvent être configurées en mode « *write-through* » ou mode « *write-back* ». Une mémoire cache en mode « *write-through* » offre une protection additionnelle, car elle permet de retrouver la valeur dans un niveau de cache inférieur. Un cache en mode « *write-back* » peut ne pas contenir une copie à jour de la donnée dans d'autres niveaux de cache, et est de cette façon moins fiable.

II.4.2 Mécanismes de protection des mémoires cache

Les principales protections utilisées pour la fiabilité du cache sont la parité et les codes de correction d'erreurs ECC (« *Error Correcting Codes* »). Les microprocesseurs modernes de type COTS contiennent pour la plupart des mécanismes de protection pour les éléments de mémoire internes : registres, TLB, MMU, caches.

II.4.2.1 Parité

Le mécanisme de parité est généralement utilisé pour la détection des erreurs simples. Son principe est simple, il consiste à calculer la parité d'une donnée, en cas d'écriture en mémoire. Ce bit de parité correspond au nombre de 1 ou de 0 dans la donnée : il peut être pair ou impair.

Ce bit de parité se calcule simplement à l'aide d'un «ou exclusif » (« XOR ») sur tous les bits du mot. Il est ensuite stocké en mémoire cache avec le tag. Lors de la lecture du mot, le bit de parité est recalculé et comparé à celui stocké en mémoire. Il existe alors deux possibilités : les deux bits de parité sont identiques, ce qui signifie qu'il n'y a pas d'erreur simple détectée. Dans l'autre cas, les bits de parité sont différents, le microprocesseur a détecté une erreur simple (ou multiple impaire) et peut prévenir le système de la détection d'une donnée corrompue.

L'avantage de ce mécanisme de protection est qu'il est en général assez simple à mettre en place et n'est pas couteux en terme de charge CPU. Il est donc souvent implémenté pour les mémoires cache L1, qui sont les mémoires les plus utilisées.

Les microprocesseurs modernes COTS proposent des techniques de recouvrement d'erreurs simples. Par exemple, le processeur e500 de Freescale qui possède un cache L1 protégé par de la parité, offre une possibilité d'implémentation de méthodes pour corriger d'éventuelles erreurs de parité détectée. Lors d'une détection d'erreur de parité, le microprocesseur génère une interruption et sauvegarde l'adresse en erreur, afin d'exécuter par la suite une routine qui permet d'invalider la ligne de cache et recharger à nouveau cette ligne de la mémoire principale, si la mémoire cache est mode « *write-through* » [GEN07].

II.4.2.2 ECC

La technique la plus communément utilisée pour protéger les mémoires des erreurs éventuelles est d'utiliser des codes de correction d'erreurs (ECC « Error-correcting code »). Elle

permet de détecter et de corriger un évènement simple et offre la possibilité de détecter certaines erreurs multiples pour les codes ECC couramment utilisés.

Ce type de détection requiert une logique de calcul complexe et des bits additionnels de stockage. La plupart des mémoires cache de niveau 2 possèdent des codes de correction d'erreurs simples et de détection d'erreurs doubles (SECDED « Single Error Correcte Double Error Detected »).

Le principe de détection d'erreurs par le code ECC est le même que celui de la parité, c'est à dire qu'à chaque écriture d'une donnée, l'algorithme spécifique au code ECC calcule les bits ECC qui sont sauvegardés, par la suite, en mémoire avec les données du cache. À la lecture de la donnée, les bits ECC stockés sont relus et recalculés afin d'être comparés. À l'issu cette comparaison, en cas d'erreurs le processeur peut alerter le système.

Les codes de correction d'erreur ECC sont employés depuis plusieurs décades pour la protection. Les codes ECC de type « Hamming » [HAM50] permettent en général la détection et la correction d'une erreur simple et la détection d'une erreur double. Les codes Hamming sont spécifiés de la manière suivante : « (n,k) » où k représente le nombre de bits de la donnée, et n le nombre de bits après encodage de la donnée. Par exemple, un code de Hamming de type (7,4) corrige une erreur simple et peut détecter jusqu'à deux erreurs en encodant une donnée de 4 bits avec 3 bits additionnels.

Il existe d'autres codes de correction fondés sur les codes polynomiaux, comme les codes Reed-Solomon, Bose, Ray-Chaudhuri, Hocquenghem … Ces codes sont très peu utilisés pour les mémoires cache. L'inconvénient de ce type de code est le temps de calcul et le temps de lecture. De ce fait, les mémoires cache L1 sont en général de type détection d'erreur et non correction. Par exemple, les microprocesseurs de la famille PowerQUICC III possèdent une mémoire cache L1 protégée par de la parité et une mémoire L2 protégée par de l'ECC capable de détecter jusqu'à deux erreurs.

II.4.3 Synthèse du calcul du facteur de décote

Comme décrit dans les travaux de S. Mukherjee *et al.* [MUK03], l'effet d'un SEU peut être classé de la manière suivante :

- Erreur de type « silencieuse » ou « Silent Data Corruption » (SDC) : ce qui signifie qu'une faute provoque des erreurs dans le résultat du programme exécuté mais qu'il n'y a aucun mécanisme interne pour détecter la corruption.

- Erreur de type « détectée » ou « Detected Uncoverable Error » (DUE) : le processeur peut détecter la présence d'erreurs dans le système mais aucun mécanisme n'est implémenté pour corriger la ou les erreurs.

-Pas d'erreur « No failure » : l'impact du SEU n'a pas d'effet au niveau applicatif. Il a pu être corrigé soit par un mécanisme interne ou soit par un effet de masquage.

Dans cette partie de la méthodologie, une distinction est faite entre les événements simples (SBU) et les évènements multiples au sein d'un même mot (MBU). Les mécanismes de protection ont un comportement différent selon le nombre d'erreurs à détecter ou corriger. Les protections comme la parité et l'ECC sont en général très efficaces pour la détection des SBU. Cependant, le taux de MBU a tendance à augmenter au cours de ces dernières années du fait de la diminution des dimensions du nœud technologique [WRO01]. L'effet d'un SBU ou d'un MBU dépend fortement de la configuration du cache et des moyens de protection mis en œuvre.

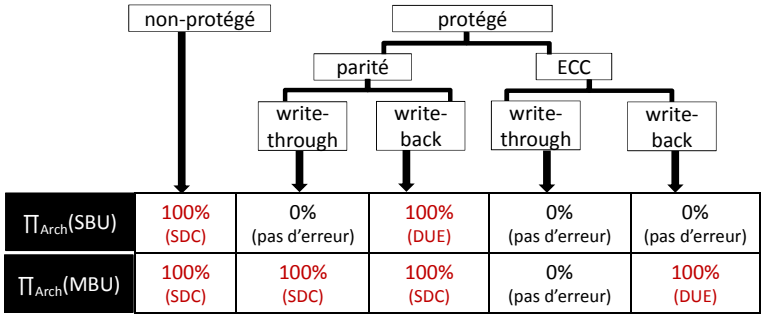


Figure II-5 : Facteur de décote lié à l'architecture de la configuration de la mémoire cache (étude pire cas supposant que l'erreur n'apparait que dans le cache)

La Figure II-5 représente la manière d'estimer le facteur de décote dû aux SBU et MBU en prenant en compte d'une part les mécanismes de protection comme l'ECC et la parité et d'autre part la configuration du cache : « *write-through* » ou « *write-back* ». Ainsi, nous avons défini le facteur de décote Π_{arch} (SBU), lié à l'architecture dû aux SBU et Π_{arch} (MBU), le facteur de décote dû aux MBU.

Dans le premier cas de figure, pour les mémoires cache n'ayant pas de système de protection, toutes les erreurs dues à l'inversion de bits sont classés comme étant des erreurs de type SDC, c'est-à-dire que, comme il n'y aucun moyen interne de détecter les erreurs, elles résident dans le système et peuvent provoquer une panne ou une erreur de calcul. Le facteur de décote pour les erreurs SDC est donc de 100 %.

Dans un second cas, lorsque les mémoires cache sont protégées par de la parité, il est possible de détecter les erreurs simples, donc tous les SBU sont en théorie détectés. De ce fait, pour un cache de configuration « *write-through* », une fois l'erreur détectée et signalée par le contrôleur de cache, un système d'interruption simple de recouvrement peut-être mis en place pour recharger la ligne en erreur. Le facteur de décote lié à ce type d'architecture est de 0 %. En revanche, pour un cache en mode « *write-back* », il est plus difficile de pratiquer cette méthode car les données ne sont pas systématiquement à jour, les erreurs sont classées en DUE et le facteur de décote est de 100 %. La parité ne permet pas de détecter les évènements multiples, ce qui implique que les erreurs potentielles sont de type SDC et le facteur de décote est à 100 %.

Dans un dernier cas, des caches protégés par des codes ECC permettent de corriger les erreurs simples, de ce fait quelle que soit la configuration de la mémoire cache, le facteur de décode est de 0 %. Les évènements de type MBUs (avec un niveau de multiplicité compatible avec l'ECC) induisent des erreurs de types DUE pour un cache protégé avec de l'ECC et configuré en mode « *write-back* ». En supposant que la multiplicité des erreurs n'est pas plus grande que la capacité de détection du mécanisme de protection ECC, les évènements de type MBUs induisent des erreurs de type DUE pour un cache protégé avec de l'ECC et configuré en mode « *write-back* ».

En conséquence, le facteur de décote lié à l'architecture Π_{arch} est calculé en se basant sur le mécanisme de protection choisi et la distribution des erreurs de multiplicité, nommée $RATIO_{MBU}$ (Eq. II-4) :

$$\Pi_{arch} = \Pi_{arch}(SBU) + \Pi_{arch}(MBU) \times RATIO_{MBU} \quad \textbf{Eq. II-4}$$

Le graphe précèdent nous montre que certaines configurations peuvent conduire à ce qu'il n'y ait pas d'erreurs, alors que d'autres configuration sont plus critiques. C'est pourquoi il y a un fort intérêt à étudier le comportement de la mémoire cache.

II.4.4 Estimation du facteur de décote pour différentes cibles

La méthode présentée précédemment est appliquée à titre d'exemple aux microprocesseurs IBM 750FX et Freescale 7447A. Les deux processeurs ont une technologie de 130 nm.

- Le microprocesseur 750FX d'IBM possède un niveau L1 composé d'un cache d'instructions et d'un cache de données de 32 KB chacun, en configuration « *write-through* » et protégé par de la parité. Le cache de niveau 2 est de taille 1 MB, fonctionne en mode « *write-back* » et utilise l'ECC comme mécanisme de protection. Il peut corriger une erreur simple et détecter jusqu'à deux erreurs au sein d'un même mot.

- Le microprocesseur 7447A de Freescale possède un niveau L1 composé d'un cache d'instructions et d'un cache de données de 32 KB chacun, et un cache L2 de 512 KB. Toutes les mémoires cache sont protégées par de la parité. Les mémoires cache de ce processeur peuvent être programmées en mode « *write-through* » ou en mode « *write-back* ». À titre d'exemple, les mémoires cache de niveau L1 sont configurées en mode « *write-through* » alors que la mémoire cache de niveau L2 est en mode « *write-back* ».

À l'aide de toutes ces caractéristiques et en appliquant la méthode, le facteur de décote Π_{arch} (SBU) et Π_{arch} (MBU) pour chaque niveau de cache peut-être estimé dans le Tableau II-3.

Tableau II-3 : Estimation du facteur de décote pour les mémoires cache

Microprocesseur	Type de cache	Π_{arch}(SBU)	Π_{arch} (MBU)
IBM 750FX	Cache L1 de données	0%	100%
	Cache L1 d'instructions	0%	100%
	Cache L2	0%	100%
Freescale 7447A	Cache L1 de données	0%	100%
	Cache L1 d'instructions	0%	100%
	Cache L2	100%	100%

Pour le microprocesseur 750FX, le facteur de décote pour le cache d'instructions Π_{arch} (cache L1 I), le cache de données Π_{arch} (cache L1 D), et le cache de niveau L2 Π_{arch} (cache L2) est estimé à :

Π_{arch} (cache L1 D) = RATIO $_{MBU}$

Π_{arch} (cache L1 I) = RATIO $_{MBU}$

Π_{arch} (cache L2) = RATIO $_{MBU}$

Le ratio de MBU pour une technologie de 130 nm peut être supposé à 1,2 % [SEI06].

Des travaux récents [GUE10] ont conduit à estimer la sensibilité des mémoires cache des microprocesseurs IBM 750FX et Freescale 7447A avec leur mécanisme de protection. Les résultats des tests d'irradiation avec les caches protégés ont montré que pour le processeur IBM 750FX aucun basculement de bits n'a été observé pour les trois mémoires cache. L'irradiation du cache L1 de données du microprocesseur Freescale 7447A protégé par de la parité n'a provoqué aucune erreur visible. En revanche, il n'y a pas eu de test des autres éléments de cache.

La méthode d'estimation de facteur de décote à partir de l'architecture de la mémoire cache permet de réduire considérablement la sensibilité pire cas et d'obtenir ainsi une première estimation de la sensibilité dynamique du microprocesseur. En revanche, il existe des configurations de mémoire cache, pour lesquelles on ne peut pas estimer de facteur de décote significatif. C'est pourquoi l'analyse du comportement dynamique des mémoires cache peut apporter une réduction du taux d'erreurs par rapport au pire cas.

II.5 Facteur de décote lié à l'application exécutée : π_{Appli}

Le facteur de décote lié à l'application exécutée consiste à analyser le comportement de l'application dans la mémoire cache pour évaluer la sensibilité de ses cellules. Dans cette partie, les notions de variable critique et de temps critique seront définies, afin d'expliquer la méthode qui permet de les identifier. L'identification de données critiques sera tout d'abord étudiée pour la partie propre aux données, puis pour la partie tag.

II.5.1 Définition des variables critiques et temps critiques

Les mémoires cache améliorent la performance selon le principe temporel et spatial. Ce sont des unités de stockage temporaire pour des données susceptibles d'être utilisées par le processeur. En pratique, il peut exister des cas où des données sont chargées dans le cache, sans que le CPU ne puisse solliciter ces données à cause du manque de localité spatiale. De plus, il est possible que certaines données ne soient sollicitées (en lecture ou écriture) qu'une première fois ou résident dans le cache après seulement un ou quelques accès. Ceci dépend de l'application et de la politique de remplacement de la mémoire cache.

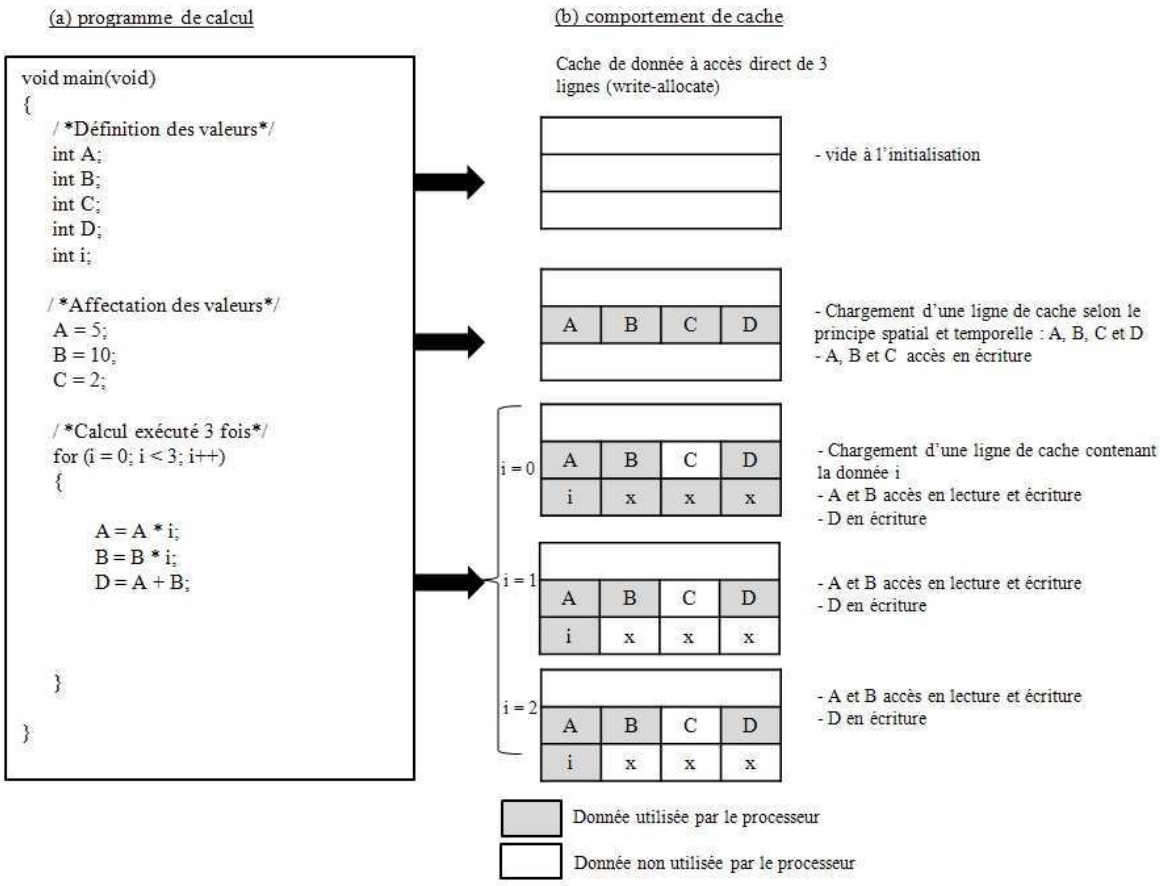


Figure II-6 : Exemple de données non-utilisées dans une mémoire cache dû à l'application
(a) Exemple d'un programme de calcul simple
(b) Exemple des données présentes en cache de données en fonction de l'exécution du code

La Figure II-6 illustre un exemple de donnée non utilisée par le processeur mais chargée en mémoire cache de données. Il s'agit d'un programme simple de calcul en C qui multiplie les valeurs de deux opérandes à chaque incrémentation, puis additionne ces deux valeurs (a). À titre d'exemple, on suppose que le compilateur est configuré sans option d'optimisation, qui aurait permis que la donnée « C » soit supprimée du code exécuté. Le cache de données (b) est configuré en mode « *write-allocate* », ce qui signifie que lors d'un *cache miss* et d'une écriture, la donnée est chargée en cache. À l'initialisation, le cache de données est inoccupé. Lors de l'affectation des valeurs qui correspondent à une opération d'écriture, le contrôleur de cache charge toute une ligne contenant les données A, B, C et D, puis stocke les valeurs correspondantes. Ainsi les 4 données sont utilisées une première fois. La partie calcul est exécutée trois fois. A la première exécution, une ligne contenant la donnée i est chargée (les autres données sont nommées « x » à titre d'exemple et ne sont pas prises en compte dans l'explication). Durant toute l'exécution de cette partie du code, la donnée C réside en cache mais n'est jamais utilisée.

Par conséquent, une inversion de bit de la donnée C n'impliquera aucune erreur dans le résultat final.

Ainsi, il peut donc résider dans le cache des données qui ne sont pas utilisées, définies aussi comme des données « mortes ». Un basculement de bit dans ce type de donnée n'engendrera pas d'erreur visible dans le système.

En pratique, une mémoire cache ne contient que quelques données non utilisées par le CPU. La méthode proposée permet donc d'identifier d'une part les données qui sont et qui ne sont pas sollicitées par le CPU mais aussi d'identifier les données critiques et non critiques.

Une donnée critique [SRI06] est définie comme étant une donnée lue par le CPU ou écrite dans un niveau de mémoire inférieur. Un SEU dans cette donnée est susceptible de se propager dans le système. Le temps critique est défini comme le temps durant lequel la donnée est critique dans la mémoire cache. Dans l'exemple précédent, la donnée D peut être facilement identifiable comme donnée non-critique car elle n'est jamais lue par le CPU.

Pour une mémoire cache configurée en mode « *write-through* », le cycle de vie d'une ligne de cache ou d'un mot de cache peut être divisé en plusieurs phases [BIS05]. Parmi ces phases (Figure II-7), le temps critique (pour une même taille de donnée) correspond aux intervalles suivant : « lecture-lecture », « écriture-lecture », et « remplissage-lecture ». Toutes les phases avant l'éviction sont incluses pour un cache en mode « *write-back* » (b) ;

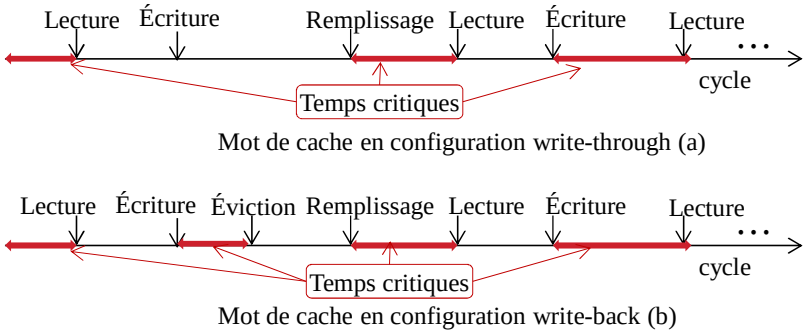


Figure II-7 : Exemple de phases critiques dans un mot de cache [BIS05]

Pour des données ayant une taille inférieure à celle de la taille normale d'un mot (exemple d'accès à un octet), les phases « écriture-écriture » ou « lecture-écriture » peuvent aussi être considérées comme sensibles. En effet, pour un mot de 32 bits par exemple, une instruction de

type stockage d'un octet n'utilisera que 8 bits parmi les 32 bits disponibles du mot. Une analyse qui prend en compte la taille d'un octet permet de pallier à ce problème [BIS05] [WAN09].

La Figure II-8 illustre le fait que les phases de « remplissage-écriture », « écriture-écriture » peuvent être considérées comme sensibles pour une architecture de type PowerPC e500. Les instructions utilisées dans l'exemple sont de type :

- une instruction de stockage d'un octet : « stb rS,d(rA) » où rS correspond au registre source où sera stocké le contenu de l'adresse effective indiquée par le registre rA.

- une instruction de chargement d'un mot : « lwz rD, d (rA) » où rD est le registre de destination.

- une instruction de chargement immédiat de valeur : « li rD,imm » où imm est la valeur immédiate.

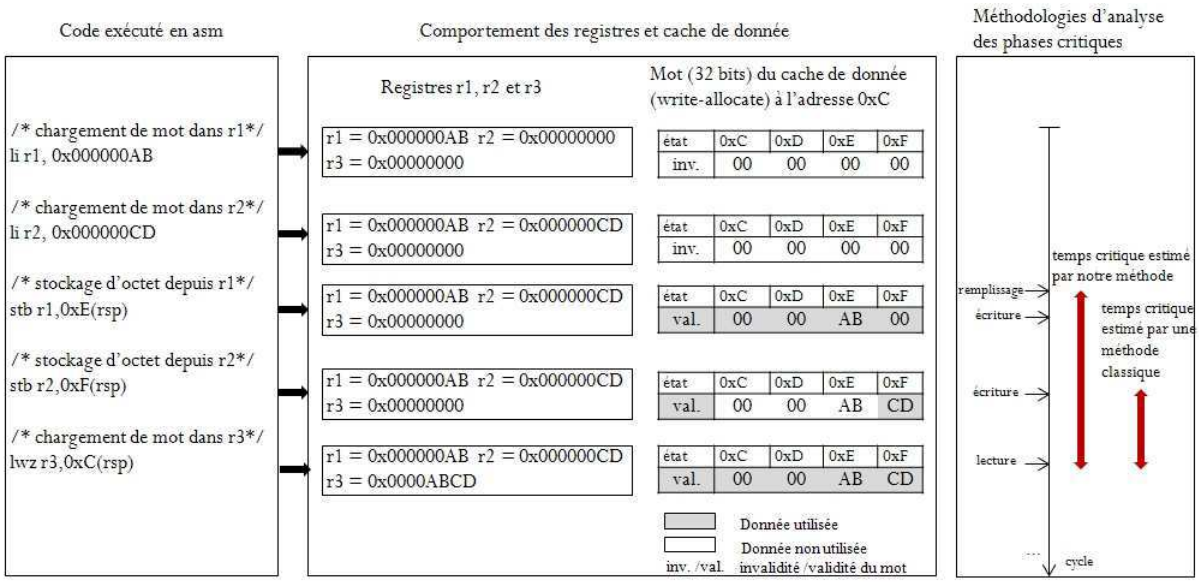


Figure II-8 : Exemple de phase critique

Le code en assembleur indique tout d'abord, un chargement de deux valeurs dans les registres r1 et r2. Puis, le dernier octet de ces registres est stocké à une adresse en mémoire. Et enfin, cette donnée est chargée dans un registre r3. À chaque étape de l'exécution, les valeurs contenues dans les registres sont représentées. Le contenu d'un mot du cache de données est illustré sur la partie droite. Il s'agit d'une mémoire cache de type « *write-allocate* ». Sur la dernière partie de la figure, deux méthodes différentes d'identification de temps critique d'un mot de

cache sont représentées. Notre méthode d'identification prend en compte les accès en écriture de tailles différentes ce qui rend l'analyse plus couvrante qu'une méthode classique [SRI06]. La granularité de la donnée analysée n'a pas seulement un impact sur la précision du facteur de décote mais aussi sur la criticité de la donnée.

Notre approche permet d'identifier les phases sensibles en se basant sur plusieurs tailles de mots pour l'analyse des données critiques. Ainsi, dans une configuration « *write-through* », on peut aussi considérer les buffers d'écriture qui sont sensibles pendant un intervalle de temps assez court.

II.5.2 Présentation de la méthode Cache Analyzer

La méthode que nous proposons pour l'identification des données critiques et des temps critiques consiste à exécuter une application dans un simulateur de microprocesseur, et surveiller les accès en mémoire cache grâce aux signaux de performance de type *cache hit* ou *cache miss* disponibles dans ces simulateurs, comme indiqué dans le paragraphe II.2.2. Ces informations sont ensuite traitées dans un outil, développé dans le cadre de la thèse, appelé Cache Analyzer afin d'en extraire un facteur de décote lié à l'application (Figure II-9).

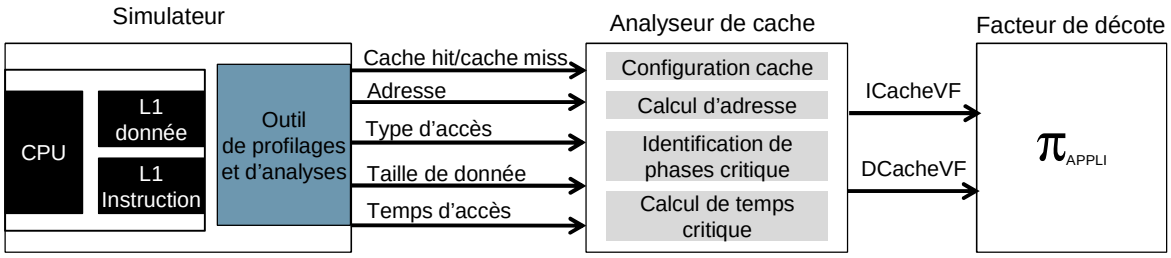


Figure II-9 : Schéma de la méthode Cache Analyzer

a) Signaux de performance spécifiques au simulateur

Il existe des simulateurs de type « cycle accurate », qui modélisent en détail le fonctionnement de la mémoire cache ainsi que les temps d'accès. Toutes les opérations sont simulées avec les temps précis d'exécution : chargement d'instruction, temps d'exécution du pipeline, temps de latence dus aux *caches miss* ... Ce type de simulateur aide les développeurs logiciels en collectant un certain nombre de paramètres et métriques relatifs à l'exécution du code source. Des simulateurs comme « Simics » [SIM05] ou le simulateur « IBM Full Simulator » [BOH04] fournissent une collection d'évènements liés aux performances du processeur. Ces

évènements (Figure II-10) retracent l'activité du processeur comme les *cache hit* et *miss*, les références mémoires, les instructions exécutées …

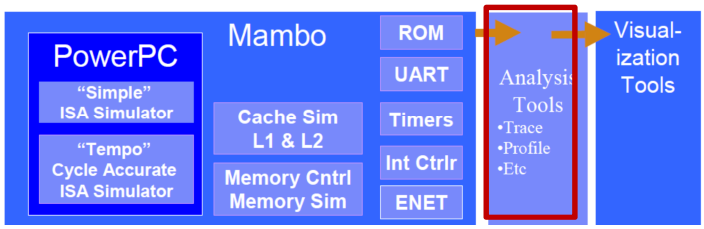


Figure II-10 : Simulateur pour processeur PowerPC [BOH04]

Le principal avantage de cette méthode est qu'il existe de nombreux simulateurs « cycle-accurate » sur le marché. D'autres sont en développement, comme il y a un fort intérêt à avoir un modèle de performance. De plus, les développeurs ont en général besoin de simulateur pour démarrer le développement logiciel avant que la cible soit disponible.

b) Paramètres nécessaires pour la détection de données critiques

Afin d'identifier en mémoire cache une donnée critique et les temps critiques associés, il est nécessaire de connaître les instants où la donnée est dans le cache et les types d'accès effectués par le processeur pour déterminer les phases critiques.

Dans la partie II.2.2, nous avons vu que lorsque le processeur a besoin d'une donnée, il va tout d'abord chercher cette donnée dans le cache. Si la donnée est en cache, cela correspond à un *cache hit* sinon en cas d'échec, il s'agit d'un *cache miss*. Ces événements sont donc un élément essentiel de la méthodologie car ils permettent de déterminer la présence ou non d'une donnée en cache. Les signaux les plus importants pour identifier les phases critiques sont donc :

- les signaux « *cache hit* » : qui permettent de détecter la présence d'une donnée dans le cache,

- les signaux « *cache miss* » : qui nous indiquent qu'une donnée n'est pas dans le cache, mais que selon la configuration du cache, elle peut être chargée en mémoire centrale,

- les signaux de types d'accès : qui nous informent sur le type d'accès à la mémoire cache (lecture ou écriture),

- les signaux d'adressage : qui nous donnent l'adresse de la donnée du cache.

Les temps d'accès et les tailles de données sollicitées sont aussi des paramètres nécessaires pour l'analyse de la criticité des données.

La Figure II-11 donne un exemple d'exécution de deux instructions dans un simulateur « cycle-accurate ». Au cycle 0, une instruction de type lecture de la donnée A est exécutée, la donnée A n'étant pas dans la mémoire cache. Ainsi, le CPU accède à la mémoire principale. Le simulateur nous renseigne sur cet évènement, c'est-à-dire que la donnée n'est pas présente dans la mémoire cache, par un évènement de type *cache miss*. Cette information est donc sauvegardée avec le temps d'accès, l'adresse de la donnée A et le type d'accès. Comme il y a eu un *cache miss* pour la lecture de cette donnée, une ligne de la mémoire contenant la donnée A est chargée de la mémoire principale vers une ligne du cache de données. Au cycle 5, une instruction de type lecture de la donnée B est chargée. Le CPU accède à la donnée B en mémoire cache directement car la donnée B est présente dans la mémoire depuis le chargement de la donnée A. Le temps critique de la donnée B a donc été prolongé lors de son chargement. Le simulateur envoie l'information du « *cache hit* » indiquant que la donnée est en cache, le type d'accès, l'adresse et le temps d'accès.

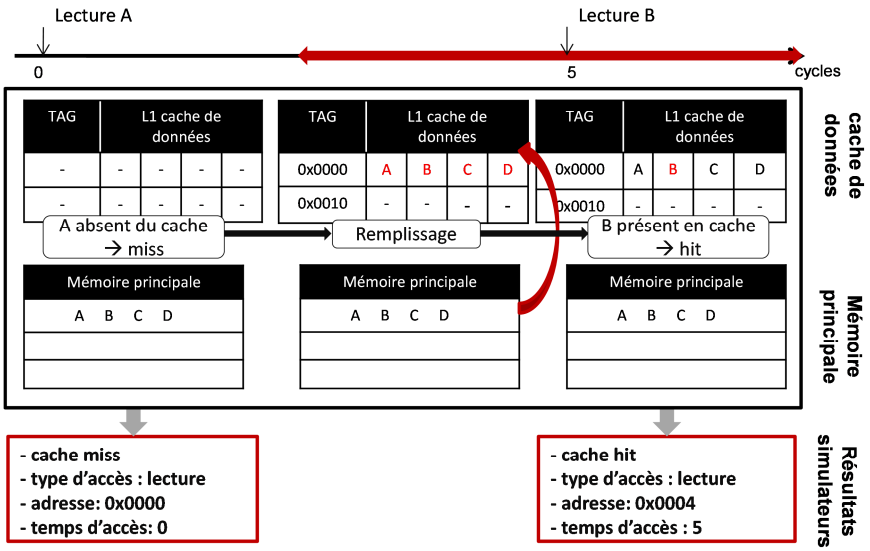


Figure II-11 : Identification des temps critiques grâce à un simulateur « cycle-accurate ».

c) Développement de l'outil de traitement d'information : « l'analyseur de cache »

Un logiciel, appelé « analyseur de cache » a été implémenté dans le cadre de la thèse pour analyser toutes les informations recueillies à l'aide du simulateur afin de calculer les temps

critiques et non-critiques. Cet outil a été développé avec la librairie JRE System Library JavaSE-1.6. Il prend en compte les paramètres pertinents comme la taille de la mémoire cache, son fonctionnement et sa configuration (associativité, taille de la ligne, taille du mot …) (Figure II-12).

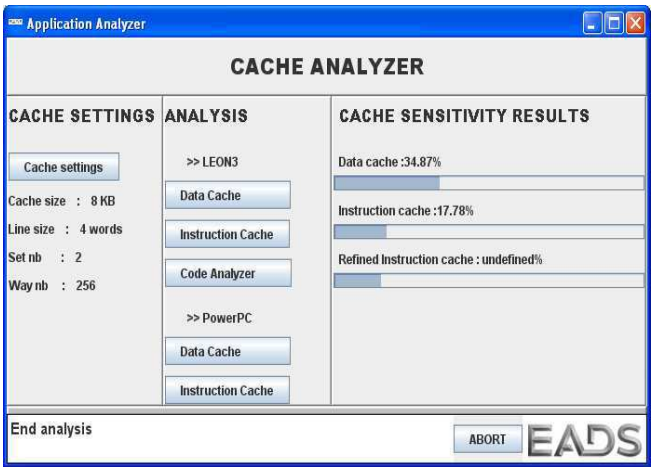


Figure II-12 : Interface logicielle d'analyse de sensibilité de cache

L'algorithme utilisé pour calculer les temps critiques a pour fonction de retrouver les adresses sollicitées, d'identifier les phases critiques et de calculer les temps critiques.

- Calcul des adresses des mots de la ligne du cache du mot sollicité :

Un « *cache miss* » engendre le chargement d'une ligne ce qui implique de sauvegarder dans la base de données les temps d'accès pour chacun des mots de la ligne. Il est donc nécessaire de calculer ou d'identifier les adresses correspondant à chaque mot de cette ligne. Le nombre de mots contenus dans une ligne de cache, la taille de la ligne et la taille du mot sont des paramètres qui permettent de recalculer la position du mot dans le cache.

- Identification des phases critiques à partir des signaux de performance et des signaux de type d'accès :

Comme indiqué dans la partie II.5.1, une phase est identifiée comme critique lorsqu'elle se termine par un accès en lecture. Le Tableau II-4 contient toutes les phases critiques possibles entre deux accès : accès précèdent et accès actuel.

Tableau II-4 : Étude de criticité d'une phase entre deux accès en mémoire cache consécutif

			Accès actuel			
			Lecture d'une donnée		Écriture d'une donnée	
			Cache hit	Cache miss	Cache hit	Cache miss*
Accès précédent	Lecture d'une donnée	Cache hit	Critique	Non critique	Non critique	Non critique
		Cache miss	Critique	Non critique	Non critique	Non critique
	Écriture d'une donnée	Cache hit	Critique	Non critique	Non critique	Non critique
		Cache miss*	Critique	Non critique	Non critique	Non critique

* ce cas est valable lors d'un cache configuré en « allocation en écriture »

** la bordure représente les phases critiques pour le cache d'instructions

- Prise en compte de la politique d'écriture du cache :

La politique d'écriture du cache est un élément qui indique la manière de traiter l'information issue du simulateur lorsqu'il y a une instruction de stockage (ou d'écriture) de donnée et que celle-ci n'est pas en cache (« *cache miss* »).

Dans le cas, d'un cache configuré en « non allocation en écriture », le contrôleur de cache ne charge ni la donnée, ni une ligne de cache, cela signifie que la donnée n'est pas ou n'est plus en cache. Il n'y a donc pas de temps critique à calculer pour cette donnée.

Pour un cache « allocation à l'écriture », le contrôleur charge le mot en cache et donc aussi tous les mots de la ligne. L'algorithme de traitement va donc recalculer les adresses de chaque mot.

- Calcul des temps critiques :

Le temps d'accès sauvegardé pour chaque accès d'une donnée va permettre de calculer le temps critique total de cette donnée (« TempsCritique »). Dès lors qu'une phase est identifiée

comme critique, le temps critique d'exposition est calculé comme étant la différence entre le temps de l'accès courant et le temps d'accès précédent. Cette durée est ensuite additionnée au temps critique total de la donnée. La Figure II-13 illustre la manière dont est calculé le temps critique pour une donnée en cache. Au temps t_0, la donnée entre en cache, elle a donc pour le moment un temps critique d'exposition nulle. À l'accès suivant, la phase est identifiée comme critique, le temps critique correspond à la différence entre t_1 et t_0 (accès initial). A l'instant t_2, il n'y a pas de période critique, le temps critique global reste inchangé. Au denier accès t_3 critique, ce temps total est recalculé.

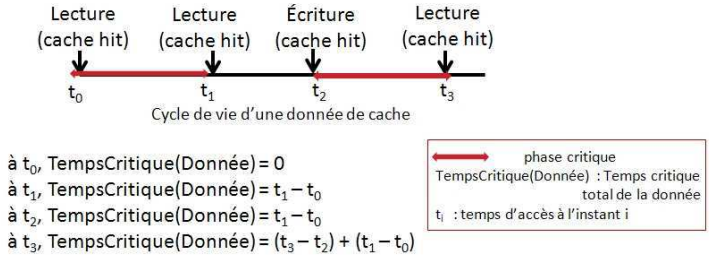


Figure II-13 : Calcul de temps critique d'exposition pour une donnée en cache

Dans le cas d'une écriture inférieure à la taille de la donnée (par exemple une instruction de stockage d'un octet), le temps entre cet accès en écriture et l'accès précédent est sauvegardé jusqu'au prochain accès.

En résumé, pour un cache de donné L1, le traitement des informations issues du simulateur est le suivant :

- Lors d'une lecture manquée, « *cache miss* », impliquant que le CPU charge une ligne entière de cache, l'adresse de chaque donnée de la ligne est alors recalculée par l'outil et le temps de chargement est aussi sauvegardé.

- Lors d'une lecture réussie, « *cache hit* », l'outil d'analyse de cache met à jour les temps critiques d'une ou plusieurs données selon le type d'instruction. Par exemple une instruction de type « load double », chargement d'un mot double, engendrera la lecture de deux mots de la mémoire cache de données.

- Lors d'une écriture réussie, le temps critique de la donnée écrite est redémarré.

- Lors d'une écriture manquée, si la mémoire cache est configurée en mode « écriture non alloué », la donnée n'est pas prise en compte par l'outil d'analyse de cache. Dans le cas contraire, l'écriture manquée est considérée comme une écriture réussie.

Le cache d'instructions de niveau 1 n'est en général accessible qu'en lecture, c'est pourquoi seuls les évènements de lecture sont enregistrés et analysés par l'outil. Un évènement de lecture de la cache d'instructions s'identifie par le chargement d'une donnée dans le 1er étage du pipeline, l'étage appelé « *Fetch* ». L'adresse de l'instruction correspond au compteur de programme.

- Lors d'un événement *cache miss* associé au cache d'instructions, une ligne entière est chargée dans la mémoire cache. Le temps critique de chaque mot de la ligne de cache ainsi que l'adresse sont enregistrés.

- Lors d'un évènement *cache hit*, on met à jour le temps d'accès de l'instruction pour pourvoir recalculer la durée du temps critique de ce mot.

d) Calcul du facteur de décote

Le facteur de vulnérabilité du cache, « CacheVF » est calculé en tenant compte du temps critique « TempsCritique » de chaque donnée critique, la taille des données critiques (octet, mots …) « TailleMot », la valeur du temps d'application « TempExecutionTotal » et la taille total du cache (Eq. II-5) :

$$CacheVF = \frac{\sum TempsCritique \times TailleMot}{TempsExecutionTotal \times TailleCache} \qquad \textbf{Eq. II-5}$$

II.5.3 Extension de la méthode au tag du cache

La méthode présentée dans la partie II.5.2 décrit comment calculer la sensibilité de la mémoire cache pour la partie données. Cependant, comme indiqué lors de la description d'une mémoire cache, le cache contient une partie adresse nommée « Tag ». Pour accéder à une mémoire cache, le CPU envoie l'adresse effective au contrôleur de cache, ou l'adresse physique selon la configuration de la MMU.

La Figure II-14 illustre la recherche d'un mot dans un cache à partir de son adresse physique : le champ d'offset sélectionne l'octet ou le mot de la ligne de cache, le champ « index » sélectionne une page, et la partie tag identifie la ligne.

Lorsque la ligne est recherchée, le tag de chaque ligne de cache est comparé aux tags de l'adresse effective. Si les tags sont identiques, il y a alors un *cache hit* sinon il s'agit d'un *cache miss*. Ainsi pour évaluer la sensibilité de la partie tag de la mémoire cache, les informations suivantes sont collectées :

- l'accès réussi ou l'accès manqué,

- le type d'accès,

- l'adresse du tag.

À partir de ces informations, le temps critique pour chaque tag peut être estimé. Le temps critique pour une donnée débute lorsqu' il y a un accès manqué et il est mis à jour à chaque fois qu'il y a un accès réussi pour le cache d'instructions ou le cache de données. Si l'adresse de la partie tag n'est pas facilement accessible avec un simulateur, on peut aisément retrouver l'adresse de chaque mot de la ligne car ils ont la même adresse de tag.

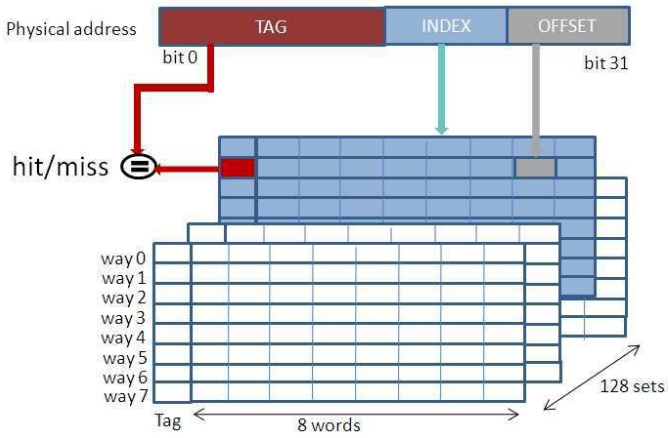


Figure II-14 : Identification du tag dans un cache d'associativité 8 de 32 KB

Le facteur de vulnérabilité du tag « TagVF » est calculé selon le temps critique « TempsCritique » de chaque tag critique identifié, la taille du tag « TailleTag », le temps total d'exécution « TempExecutionTotal » de l'application et la taille totale d'un bloc (Eq. II-6).

$$TagVF = \frac{\sum TempsCritique \times TailleTag}{TempsExecutionTotal \times TailleBlockTag} \quad \textbf{Eq. II-6}$$

II.5.4 Identification des bits « silencieux »

Une nouvelle approche pour améliorer la précision d'identification de bits critiques a été étudiée. Ceci peut être appliqué en extrapolant la définition des bits « un-ACE » (Unnecessary for architectural Correct Execution bits) définie par S. Mukherjee *et al.* [MUK03] au jeu d'instructions des microprocesseurs.

Nous avons développé une analyse statique du jeu d'instructions pour identifier les bits silencieux du cache d'instructions. Les bits silencieux sont des bits pour lesquels la corruption n'a pas d'impact sur l'exécution de l'instruction.

L'analyse statique consiste à créer une base de données qui contient tous les bits silencieux pour chaque instruction du jeu d'instructions et ensuite utiliser cette base pour déterminer les bits silencieux de chaque instruction identifiée comme critique.

Il existe plusieurs possibilités pour identifier un bit dit silencieux :

- Le moyen le plus simple est d'identifier tous les bits de type non-utilisé en regardant le format du jeu d'instructions. En effet, certains champs de l'instruction sont spécifiés comme « non-utilisé » Cela signifie qu'ils ne sont pas lus. Par exemple, dans la Figure II-15 pour une architecture PowerPC [FRE07], l'instruction de synchronisation contient 16 bits « non –utilisés », c'est pourquoi il n'y a que 50 % de l'instruction qui est sensible.

- Certains bits d'une instruction peuvent aussi ne pas être utilisés sous certaines conditions spécifiques. Par exemple, dans la Figure II-15, pour un jeu d'instructions SPARC, les bits 5 à 12 de l'instruction LOAD sont toujours non-utilisés. Si le champ « rs2 » est égal à 0 alors le bit 13 est aussi de type silencieux.

- Une autre méthode consiste à inverser chaque bit d'une instruction et décoder l'instruction pour analyser le résultat. L'injection de fautes peut-être réalisée à l'aide des méthodes de l'état de l'art comme l'injection de fautes par émulation, la méthode CEU ou encore la méthode d'injection de fautes par interface de débogage. Lorsque la corruption de bit n'engendre pas d'effet dans l'exécution de l'instruction, il peut être classé comme bit silencieux. Les corruptions

de bits qui modifient le registre adressé ne sont pas prises en compte pour les bits silencieux. L'effet peut ensuite être classé selon trois groupes :

-Faute : instruction qui n'est pas implémentée

-Silencieux : pas d'effet dans l'exécution de l'instruction

-Dépendant du code : le bit sensible dépend de l'opérande.

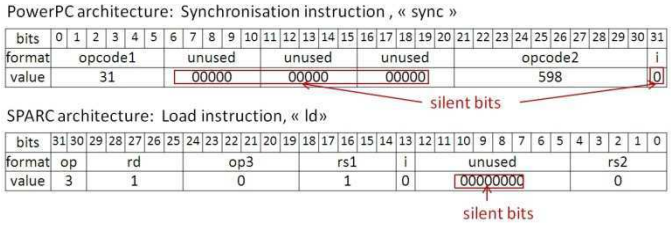


Figure II-15 : Identification des bits non utilisé

Enfin, après avoir généré une base de données des bits silencieux, les étapes suivantes sont réalisées :

Identifier les données critiques et leur temps critiques associés pour le cache d'instructions avec l'analyseur de Cache. Analyser pour chaque instruction identifiée comme critique, le nombre de bits silencieux, et ainsi réduire le nombre de bits critiques. Une sensibilité plus raffinée pour le cache d'instructions "ICacheVF", est calculée en tenant compte du nombre de bits critiques pour chaque instruction (Eq. II-7) :

$$ICacheVF = \frac{\sum TempsCritique \times NbreBitCritique}{TempsExecutionTotal \times TailleCache} \qquad \textbf{Eq. II-7}$$

De plus, une analyse binaire dynamique peut aussi être faite pour le nombre de bits « Dépendant du code » et pour identifier les bits silencieux. En combinant l'analyse statique avec un simulateur, le masquage logique est facilement détectable. Cela consiste à observer le contenu des opérandes et à évaluer si l'inversion d'un bit ne modifie pas le résultat. Ce cas concerne principalement les instructions logiques de type : xor, or, and …

De plus, certaines instructions logiques peuvent aussi apporter un facteur de décote supérieur car il existe un certain nombre d'instructions logiques qui donnent le même résultat

pour les mêmes opérandes. Par exemple, dans une architecture SPARC, une instruction « XOR » peut donner une instruction « OR » lors d'une inversion de bit mais le résultat de « 0 xor 1 » est équivalent à « 0 or 1 ». Néanmoins, l'effet du masquage logique n'a pas été implémenté car on estime que le facteur de décote obtenu par cette méthode est insignifiant alors que l'analyse serait relativement longue.

Le facteur de décote lié à l'application, « Π_{appli} », pour une mémoire cache inclut aussi la partie adresse c'est-à-dire tag de la mémoire cache. Ainsi le facteur de décote peut être estimé en prenant compte de CacheVF et du TagVF, prenant en compte le nombre total d'éléments de chaque sous-bloc (Eq. II-8) :

$$\pi_{appliTotalCache} = \frac{CacheVF \times TailleCache + TagVF \times TailleBlockTag}{TailleCache \times TailleBlockTag} \quad \textbf{Eq. II-8}$$

Cependant, en considérant les bits silencieux, le facteur de décote du cache d'instructions lié à l'application est estimé grâce à l'équation suivante (Eq. II-9) :

$$\pi_{appliTotalCacheInst} = \frac{ICacheVF \times TailleCache + TagVF \times TailleBlockTag}{TailleCache \times TailleBlockTag} \quad \textbf{Eq. II-9}$$

Ainsi, le facteur de décote total prend en compte le facteur de décote lié au cache d'instructions et le facteur de décote lié à au cache de données, à savoir (Eq. II-10) :

$$\pi_{appli} = \left\lceil \frac{\pi_{appliTotalCacheInst} \times TailleTotalCacheI}{TailleTotalCacheI + TailleTotalCacheD} \right\rceil + \left\lceil \frac{\pi_{appliTotalCacheD} \times TailleTotalCacheD}{TailleTotalCacheI + TailleTotalCacheD} \right\rceil \quad \textbf{Eq. II-10}$$

La méthode peut être appliquée facilement dans un simulateur précis au niveau cycle (cycle accurate) qui fournit les informations de type accès réussi ou accès manqué avec des informations d'accès mémoire.

II.6 Conclusion

Dans ce chapitre, les résultats d'un test dynamique en accélérateur ont été présentés. Ils ont permis de mettre en avant d'une part les contraintes liées aux accélérateurs de particules et la nécessité d'avoir un outil ingénieur capable d'estimer un taux d'erreurs plus précis pour les mémoires cache. De plus, l'état de l'art des techniques d'analyse de sensibilité de cache montre que ces approches sont difficilement applicables pour un utilisateur final qui ne dispose pas des codes sources pour modifier le comportement du simulateur. Ainsi, l'objectif envisagé est d'établir une méthodologie générale, fondée sur les mémoires cache, qui permette d'estimer un taux d'erreurs plus précis non fondé sur l'instrumentation d'un simulateur spécifique et ne nécessitant pas un modèle détaillé du processeur.

Afin de mettre en place une telle méthodologie, une analyse détaillée du fonctionnement des mémoires cache a été présentée. Cette étude a permis d'introduire les concepts essentiels relatifs aux mémoires cache, mais aussi de mettre en évidence que, selon la configuration et l'architecture des mémoires cache, les données résidant en cache sont plus ou moins importantes pour l'exécution correcte du code. L'analyse du fonctionnement du cache a permis de déterminer un facteur de décote lié à l'architecture et un facteur de décote dépendant de l'application exécutée.

La méthodologie générale présentée dans ce chapitre a pour finalité d'obtenir une estimation plus précise du taux d'erreurs aux radiations des microprocesseurs. En effet, en tenant compte de l'application exécutée, de l'environnement et de la sensibilité de la technologie, le taux d'erreurs peut-être raffiné. Cette méthode consiste à obtenir dans une première partie la sensibilité intrinsèque de la technologie du composant et dans une seconde partie, à analyser l'application et l'architecture du composant pour avoir une sensibilité dynamique.

Le facteur de décote lié à l'architecture du cache prend en compte les mécanismes de protection et les politiques d'écriture du cache. Les mécanismes de protection comme la parité détectent les erreurs simples qui peuvent être corrigées si le cache est configuré en mode « *write-through* », c'est-à-dire si le cache recopie systématiquement les données dans d'autres mémoires. Mais il existe des cas où le facteur de décote ne permet pas de réduire la sensibilité pire cas, ainsi une analyse du comportement dynamique des mémoires cache peut apporter un deuxième facteur de décote.

En exécutant une application sur un simulateur de processeur et en surveillant les signaux propres aux simulateurs, comme les signaux de performance et d'accès cache, il est possible d'évaluer une sensibilité des mémoires cache plus précise. Ces signaux issus du simulateur aident l'utilisateur à identifier des données critiques dans le cache. En calculant les temps d'exposition des données critiques, on obtient un facteur de décote lié à l'application.

Dans le chapitre suivant, la méthode proposée est appliquée à un processeur LEON3. Puis une deuxième partie est consacrée à la validation de la méthodologie par différents moyens en accélérateur de particules.

Chapitre III : Application de la méthodologie à un microprocesseur softcore et validation de la méthodologie

Dans le chapitre précédent, une méthodologie de prédiction de taux d'erreurs dans un microprocesseur a été proposée. Cette méthode est axée sur les mémoires cache, qui sont les éléments les plus sensibles dans le processeur en raison de leur grand nombre de cellules et de leur rôle primordial dans l'exécution d'une application.

La méthode proposée dans le chapitre II consiste à estimer des facteurs de décote afin d'obtenir un taux d'erreurs dynamique, plus précis qu'un taux d'erreurs pire cas. L'estimation de ces facteurs repose sur l'architecture et la configuration des mémoires cache ainsi que le comportement des données au sein de ces mémoires. L'analyse de l'architecture des mémoires cache permet de déterminer l'effet produit par un SEU à partir des mécanismes de protection implantés et des politiques d'écriture de cache. Cependant, il existe des configurations où l'architecture de la mémoire cache n'apporte pas de facteur de décote. De plus, l'occurrence d'événements multiples augmente avec la diminution de la taille du nœud technologique. Les mécanismes de protection ne sont plus suffisants pour avoir un taux d'erreurs satisfaisant les exigences de fiabilité des applications critiques. Ainsi, l'analyse du comportement du ou des caches exécutant une application permet d'obtenir un deuxième niveau de précision. En exécutant une application sur un simulateur de processeur et en observant le type d'accès et les signaux de performance du cache, il est possible d'identifier des données dites critiques.

Dans ce chapitre, nous proposons de valider expérimentalement la méthode d'analyse de cache par deux des moyens de tests les plus pertinents de l'état de l'art, identifiés dans le chapitre1 : l'injection de fautes par émulation et les tests en accélérateur. Une première étape de validation a donc été réalisée en comparant les résultats obtenus par la technique classique d'injection de fautes par émulation. Le taux d'erreurs estimé par cette technique est assez précis mais nécessite un modèle synthétisable de processeur. Par conséquent, la méthodologie d'analyse de cache a été appliquée à un système sur puce basé sur le processeur « softcore » LEON3.

La deuxième validation expérimentale par test en accélérateur a nécessité le développement d'un banc de test spécifique qui est décrit dans ce chapitre. Les résultats obtenus à l'issu des tests en accélérateur aux sources de neutrons et de protons sont présentés dans la dernière partie.

III.1 Présentation du microprocesseur cible : LEON3

La méthodologie a été appliquée et validée pour un processeur LEON3. Ainsi, dans cette partie, le choix de la cible est expliqué plus en détails, ainsi que le fonctionnement du processeur et de ses mémoires cache.

III.1.1 Choix de la cible

Afin de valider expérimentalement la méthode d'analyse de cache proposée au chapitre II, nous avons choisi de comparer cette méthode avec deux autres issus de l'état de l'art. Comme indiqué dans la partie I.4, la technique d'injection de fautes par émulation à partir de descriptions synthétisables fournit un taux d'erreurs très précis et offre la possibilité d'injecter des fautes dans toutes les parties du processeur. Néanmoins, elle requiert un modèle HDL. C'est pourquoi le choix du processeur LEON3 répond à ce critère car il dispose d'un modèle HDL libre.

De plus, les caractéristiques hautement configurables de ce processeur ont renforcé sa sélection. En effet, ce processeur présente une architecture de processeur COTS moderne : il possède un cache de données et un cache d'instructions. La facilité de portabilité sur des cibles matérielles différentes est aussi un avantage majeur.

Ce processeur est utilisé dans des applications ayant de forts objectifs de fiabilité avec une version tolérante aux fautes : le LEON3FT. Cette version du processeur offre des mécanismes de détection et de correction d'erreurs dans tous les éléments mémoires du processeur comme les registres et les caches. Il est largement utilisé par des programmes spatiaux européens et internationaux [KOE10] [PHA11]. Toutefois cette version n'est pas disponible librement et le travail a donc été effectué sur la version sans mécanisme de protection.

III.1.2 Présentation générale

Le processeur LEON3 est un processeur 32 bits qui est fondé sur le jeu d'instructions SPARC V8 [SPA92]. Le code source est disponible sous licence GNU GPL. Il est configurable à

l'aide de paramètres génériques VHDL. Il est ainsi possible d'implanter plusieurs configurations de mémoire cache.

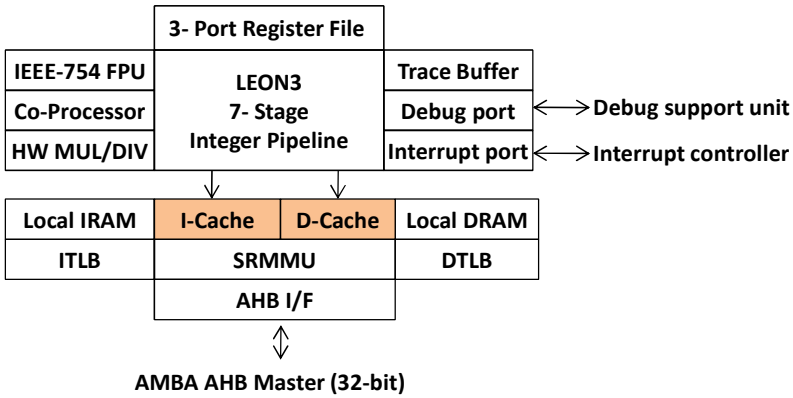


Figure III-1 : Schéma du processeur cible LEON3

La Figure III-1 illustre l'architecture du LEON3, qui possède les caractéristiques suivantes :

- un pipeline entier constitué de 7 étages,

- une architecture Harvard : un cache de données et un cache d'instructions séparés, de tailles configurables,

- un multiplieur et un diviseur matériels (optionnels),

- une unité de débogage intégrée,

- une extension multiprocesseur.

Le LEON3 dispose un système de mémoires configurables. Le cache L1 de données et le cache L1 d'instructions peuvent être paramétrés en nombre de pages (1 à 4), taille de page (1 à 256 KB), nombre de mots par ligne (4 ou 8), et politiques de remplacement des lignes (LRU, aléatoire …).

Par ailleurs, les mémoires cache sont configurées par défaut en mode « *write-through* » et il n'y a pas d'allocation en mémoire cache de données lors d'une écriture avec un *cache miss*. La MMU et les buffers de translation TLB n'ont pas été activés pour le système implanté.

III.2 Application de la méthodologie

Le simulateur associé à ce processeur est le simulateur cycle-accurate TSIM. Il est disponible en version d'évaluation mais est limité et ne permet pas d'appliquer la méthodologie proposée [TSI13]. En effet, ce simulateur ne dispose pas de signaux de performance accessibles dans sa version d'évaluation. Ainsi, le simulateur ModelSIM a été préféré pour appliquer la méthodologie. Celui-ci ne dispose pas de signaux de performance, mais il offre la possibilité de simuler tous les signaux du processeur LEON3 et donc de disposer des signaux relatifs aux accès en mémoire cache. Les signaux de performance et d'accès en cache ont été créés à partir de la lecture des différents signaux du LEON3 ; cette méthode présente l'avantage d'être non intrusive.

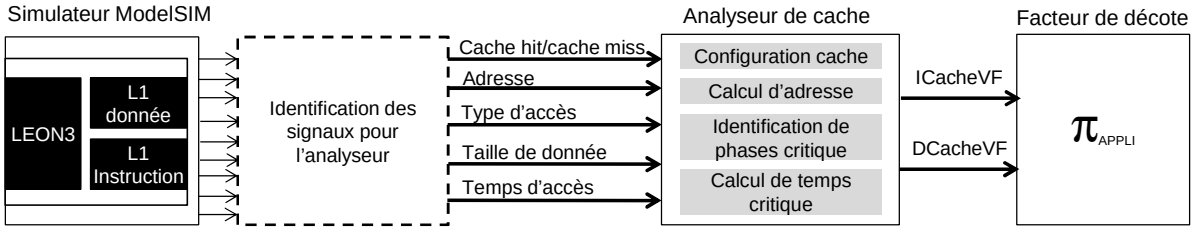


Figure III-2 : Schéma de la méthode appliquée au processeur LEON3

L'application de la méthodologie proposée pour un processeur LEON3 consiste à exécuter une application dans le simulateur MODELSIM puis, à partir des signaux accessibles du LEON3, de recréer des informations de profilage. Ces informations sont ensuite envoyées à l'analyseur de cache pour calculer le facteur de décote de l'application (Figure III-2).

III.2.1 Identification des signaux de performance et d'accès en cache

Pour appliquer la méthodologie présentée dans le chapitre précédent, il est nécessaire de contrôler, durant l'exécution d'un programme, les signaux de performance « *cache hit* » et « *cache miss* », qui indiquent si la donnée sollicitée par le CPU est présente ou non dans le cache. Pour le cache de données, les signaux d'accès sont surveillés pour indiquer le type d'accès (lecture ou écriture) afin d'identifier la criticité de la donnée. L'adresse et la taille de la donnée sont aussi des paramètres à prendre en compte.

Le cache d'instructions n'étant accessible qu'en lecture seule, le compteur de programme est le principal élément à analyser.

Afin de retrouver les principaux éléments, une étude détaillée du comportement des mémoires cache du processeur LEON3 a été réalisée et est présentée ci-dessous.

Tout d'abord, il est important d'identifier l'étage du pipeline où les mémoires caches sont sollicitées.

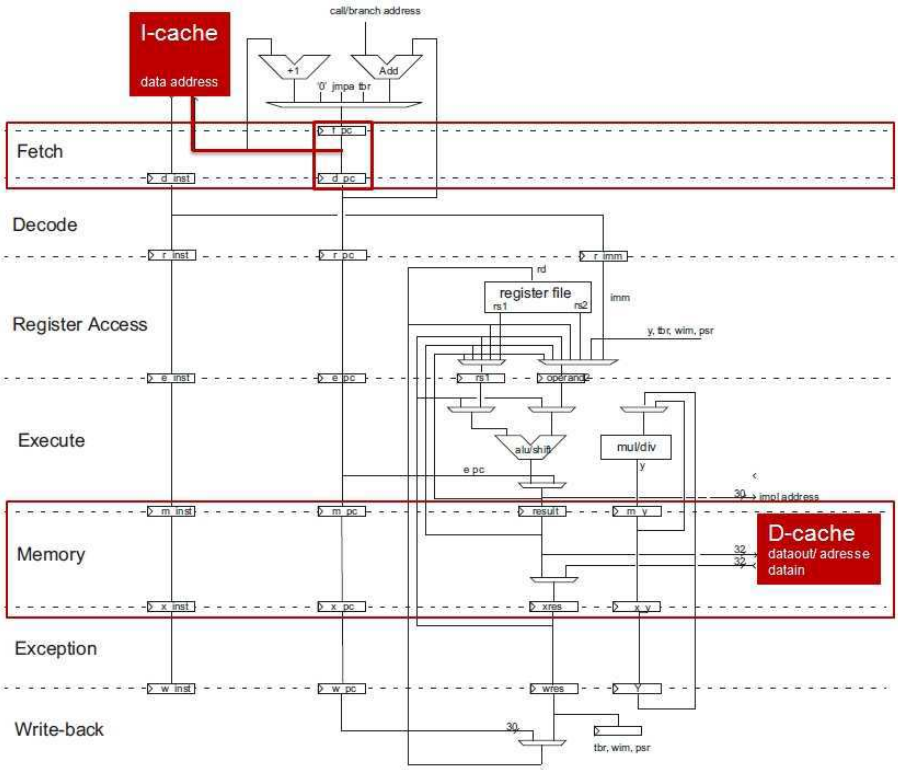


Figure III-3 : Pipeline du LEON3 avec accès aux mémoires caches

Le pipeline du LEON3 est composé de 7 étages : *Fetch*, *Decode*, Accès aux registres, Exécution, Accès mémoire (« Memory »), Exception et Complétion (« *write-back* »). Les accès en mémoire cache d'instructions se font uniquement à l'étage « *Fetch* » et les accès en mémoire cache de données à l'étage « Memory ». De ce fait, l'identification des signaux nécessaires à l'élaboration de la méthode proposée se concentrera uniquement sur ces deux étages.

La Figure III-4 illustre le fonctionnement simplifié de la mémoire cache d'instructions du LEON3. À l'étage « *Fetch* » du pipeline, une requête au contrôleur de cache est effectuée. Pour l'analyse du cache, les signaux en sortie de l'étage « *Fetch* » du pipeline correspondant au compteur de programme PC, sont utilisés pour identifier l'adresse de l'instruction recherchée en mémoire cache. La machine d'états au sein du contrôleur de cache permet d'indiquer le résultat de la requête.

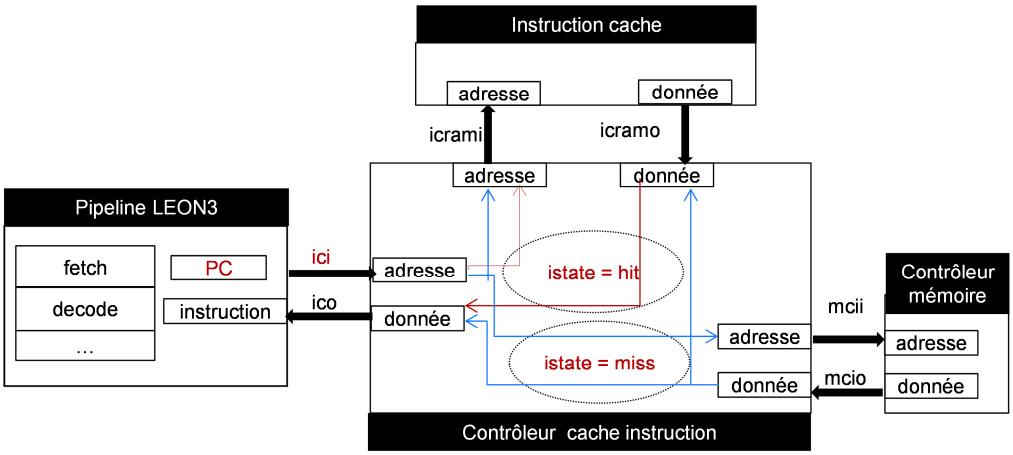


Figure III-4 : Schéma simplifié d'accès au cache d'instructions

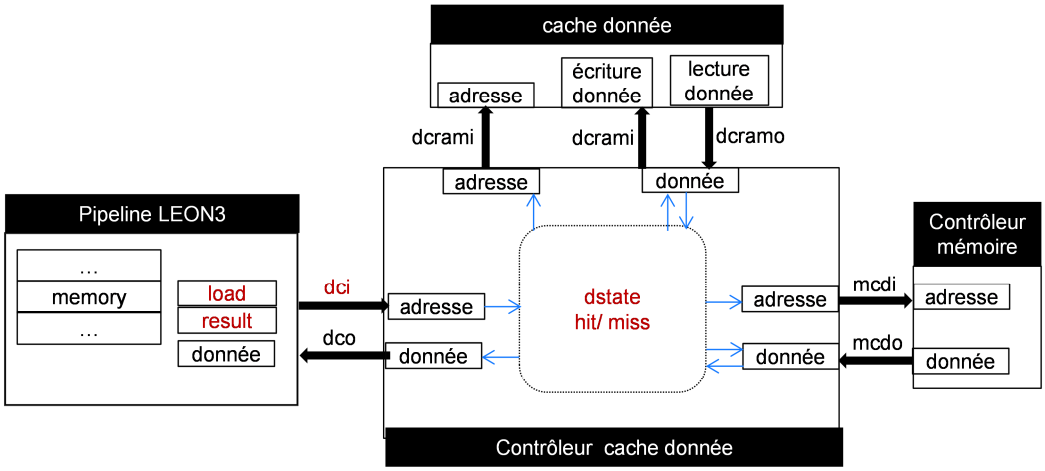


Figure III-5 : Schéma simplifié d'accès au cache de données

La Figure III-5 montre les signaux du LEON3 qui permettent d'écrire ou de lire en mémoire cache de données. Comme pour le cache d'instructions, les signaux de la machine d'états du contrôleur de cache sont utilisés pour identifier les accès « *cache hit* » et « *cache miss* ». Par ailleurs, les signaux indiquant s'il s'agit d'un chargement ou d'un stockage sont aussi nécessaires pour identifier la criticité d'une donnée.

III.2.2 Simulation du processeur et résultats

Le processeur exécutant une application est simulé avec ModelSIM. Il existe plusieurs méthodes pour récupérer le contenu des signaux à analyser.

Une première approche a été implantée pour récupérer le contenu des signaux en cours d'exécution. Elle repose sur une interface FLI (*Foreign Language Interface*). Les fonctions FLI sont

des fonctions écrites en langage C qui donnent un accès aux informations dans le simulateur. Ainsi, en utilisant les fonctions suivantes, il est possible d'obtenir la valeur d'une donnée en cours d'exécution de l'application pour identifier l'adresse, le type d'accès et l'information *cache hit* et *cache miss* :

- mti_FindSignal() : retrouver n'importe quel signal simulé,

-mti_Sensitize() : détecter la modification d'une valeur,

-mti_SignalImage() : obtenir la valeur d'un signal,

-mti_SetSignalValue() : définir la valeur d'un signal

Cette méthode présente l'avantage de sauvegarder uniquement les informations utiles pour l'analyse et de les traiter pendant l'exécution de l'application sans la ralentir.

La deuxième approche est plus simple à mettre en place, car il s'agit de sauvegarder, dans un fichier de sortie, les valeurs d'un certain nombre de signaux. Pour cela il suffit de lancer une simulation en spécifiant dans une liste le nom des signaux à surveiller.

Pour l'application de la méthodologie, il a été choisi de configurer le processeur de la manière suivante :

- un cache L1 de données et L1 d'instructions de 8 KB,

- un cache composé de deux pages, 256 lignes par page et 4 mots par ligne.

La méthode d'analyse de cache a été appliquée à plusieurs programmes issus de MiBench [GUT01] :

- L'algorithme « FFT » calcule une transformée de Fourrier rapide, et son inverse.

- L'application « basicmath » exécute de simples calculs mathématiques tels qu'une fonction cubique, des conversions d'angles et d'entiers au carré.

- L'application « stringsearch » recherche une chaîne de caractères dans un ensemble de phrases.

- L'algorithme « bitcount » compte le nombre de bits dans un tableau d'entiers de cinq manières différentes.

- L'application « qsort » trie un ensemble de chaînes par ordre croissant.

La Figure III-6 illustre les facteurs de décote obtenus pour le cache d'instructions et le cache de données. La partie gauche représente les résultats des applications de MiBench. La partie droite montre des résultats pour des applications personnalisées. Les applications nommées AES et « additions de vecteurs » sont présentées en détails dans la partie III.3.

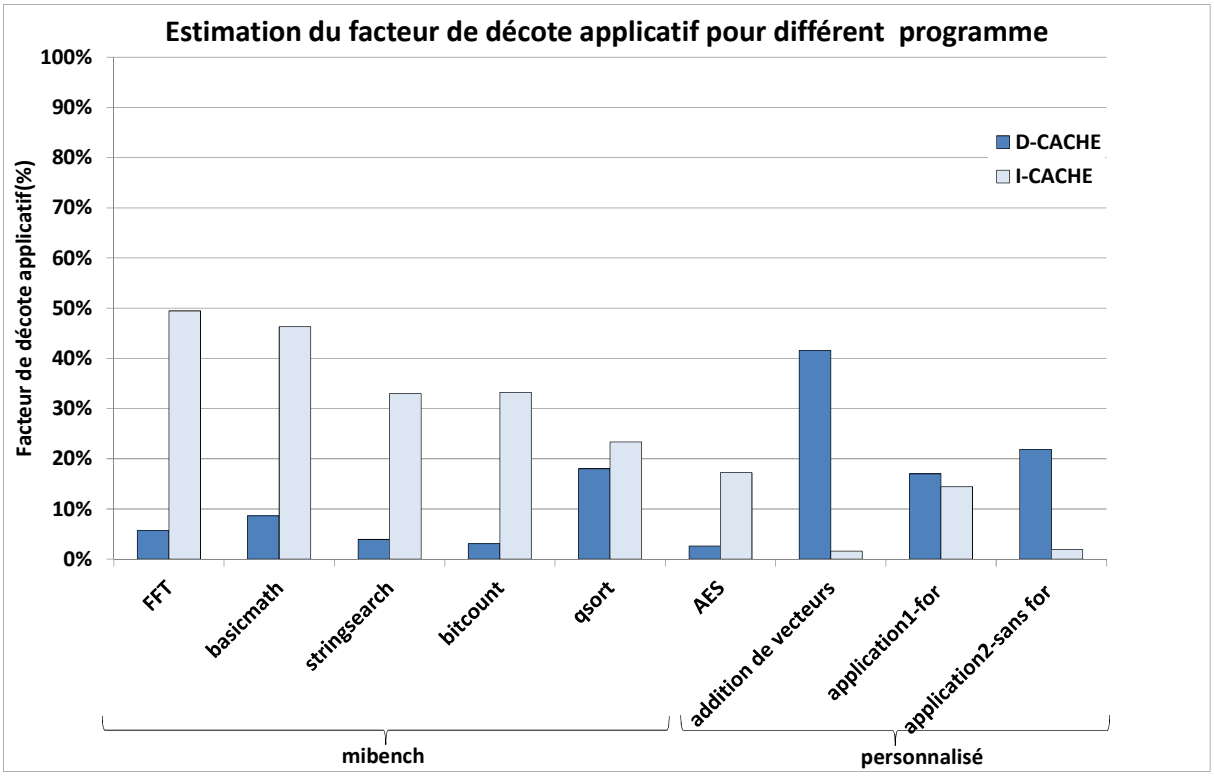


Figure III-6 : Facteur de décote du cache de données et du cache d'instructions estimé pour plusieurs applications.

Les deux applications testées avec des faisceaux de particules, et présentées dans le chapitre II, ont été soumises à l'analyseur de cache : « application1-for » et « application2-sans for ». Ces deux applications permettent d'évaluer l'impact de la manière de programmer sur le facteur de décote. Elles concernent essentiellement le cache d'instructions. Les résultats de l'analyseur de cache indiquent que même si le cache d'instructions n'est pas occupé à 100 %, il est sensible car les mêmes instructions sont utilisées constamment.

L'application « qsort » présente un facteur de décote pour le cache de donnes assez élevé, cela peut s'expliquer par le fait qu'un nombre important de données sont triées.

La durée de ces analyses statiques ne dépend que du temps d'exécution de l'application sur le simulateur. Le Tableau III-1 présente les temps d'exécution de l'application et non la durée de l'analyse qui est négligeable.

Tableau III-1 : Caractéristiques des applications testées

Applications	Option de compilation	Temps d'exécution total
FFT	-o3	50,0 ms
basicmath	-o3	56,7 ms
stringsearch	-o3	14,2 ms
bitcount	-o3	15,0 ms
qsort	-o3	11,2 ms
AES	-o2	13,4ms
additions de vecteur	-o2	2,8 ms
appli1-for	aucune	9,9 ms
appli2-sans for	aucune	11,7 ms

III.3 Validation de la méthodologie par injection de fautes

Afin de valider la méthode d'analyse de cache par la technique d'injection de fautes par émulation, on vérifie qu'une donnée (ou instruction) corrompue par l'injection de fautes et provoquant une erreur d'application est bien identifiée comme critique par l'analyseur de cache. A l'inverse, une information corrompue qui ne génère pas d'erreur peut être identifiée comme critique par l'analyseur de cache. D'une part, des phénomènes de masquage fonctionnels peuvent ne pas être identifiés par l'analyseur de cache. D'autre part, il s'agit d'obtenir une évaluation conservative pour ne pas sous-estimer le taux de défaillance de l'application. L'objectif est d'être le plus proche possible des résultats d'injection, mais de conserver une valeur supérieure du facteur de décote.

III.3.1 Présentation de la méthode d'émulation

Comme indiqué dans le chapitre 1, le principe de l'injection de fautes par émulation consiste à implanter un prototype de la cible sur un système à base de FPGA, à partir de la description RTL, et d'injecter ensuite des fautes dans le prototype.

EADS IW [GUI12] a développé une plateforme d'injection de fautes fondée sur l'instrumentation [LOP07]. Ainsi, les cellules mémoires sont instrumentées avec des saboteurs dont l'état peut être altéré grâce à un mécanisme externe de contrôle. Les cellules mémoire sont instrumentées au niveau de la netlist. La plateforme d'émulation, ELECTRE (ELaborated

Electronic Component Tester for Radiation Effects) est composée de deux cartes avec un FPGA Virtex-4 de Xilinx et implémentant le processeur LEON3. Une carte est dédiée au contrôle de l'injection de fautes et l'autre contient le composant sous test. Les cibles et les cycles d'injection de fautes sont sélectionnés de façon aléatoire. Une faute est injectée par exécution d'une application. La plateforme d'émulation donne une trace complète des conditions de l'injection de fautes et les résultats qui sont utilisés pour la validation.

III.3.2 Comparaison des résultats obtenus

Environ 100 000 erreurs ont été injectées dans chaque cache (données et instructions), donnant une marge d'erreur de 0,31 % pour l'analyse des résultats avec un niveau de confiance de 95 %. L'injection de fautes a duré 4 jours, alors que l'analyse de cache a été faite en 30 minutes.

Pour valider la méthodologie, il a été vérifié que toutes les adresses détectées comme sensibles par la technique d'émulation ont aussi été détectées sensibles par l'analyseur de cache. Il a aussi été vérifié que pour une adresse donnée, l'instant d'injection de fautes provoquant une erreur de l'application par la technique d'émulation se trouvait bien dans une période de temps critique.

La validation de la méthode d'analyse de cache a été effectuée pour deux applications : l'application « AES » et une addition de vecteurs. La première application consiste à encrypter et décrypter une chaine de caractères, puis à dérouler un algorithme de code de redondance cyclique (CRC). L'algorithme implémenté est l'AES (Advanced Encryption Standard) avec une clé de 128bits développé par EADS selon le standard FIPS-197 [FIP02]. La deuxième application, nommée « addition de vecteurs », est composée de deux parties. La première partie est constituée d'additions de vecteurs successives, et la seconde partie consiste à calculer un CRC sur le résultat. Cette application permet de maximiser la sensibilité du cache de données et du cache d'instructions, en utilisant un grand nombre de données.

Les résultats obtenus par émulation et par l'analyseur de cache sont représentés sur la Figure III-7 pour l'application AES. Cette figure est inspirée d'une représentation des auteurs de [HAT71] qui illustre le principe de localité temporelle et spatiale. La Figure III-7 représente les adresses détectées comme étant sensibles. Le graphe supérieur représente toutes les adresses sensibles du cache d'instructions (environ 2048 adresses pour un cache de 8 KB) pour chaque cycle d'horloge de l'application. La même représentation est donnée pour le cache de données dans le graphique inférieur. Seules les données qui provoquent une erreur d'application par la

technique d'émulation sont représentées pour des raisons de lisibilité. Ce graphique montre à la fois que toutes les adresses critiques identifiées par l'injection de fautes par émulation sont comprises dans l'ensemble des adresses identifiées comme critiques par l'analyse de cache et que pour chacune de ces adresse critiques, les temps d'injection de fautes qui conduisent à une erreur d'application lors de l'injection de fautes par émulation sont tous inclus dans les temps critiques déterminés par l'analyse de cache. Ceci vaut à la fois pour le cache de données et pour le cache d'instructions.

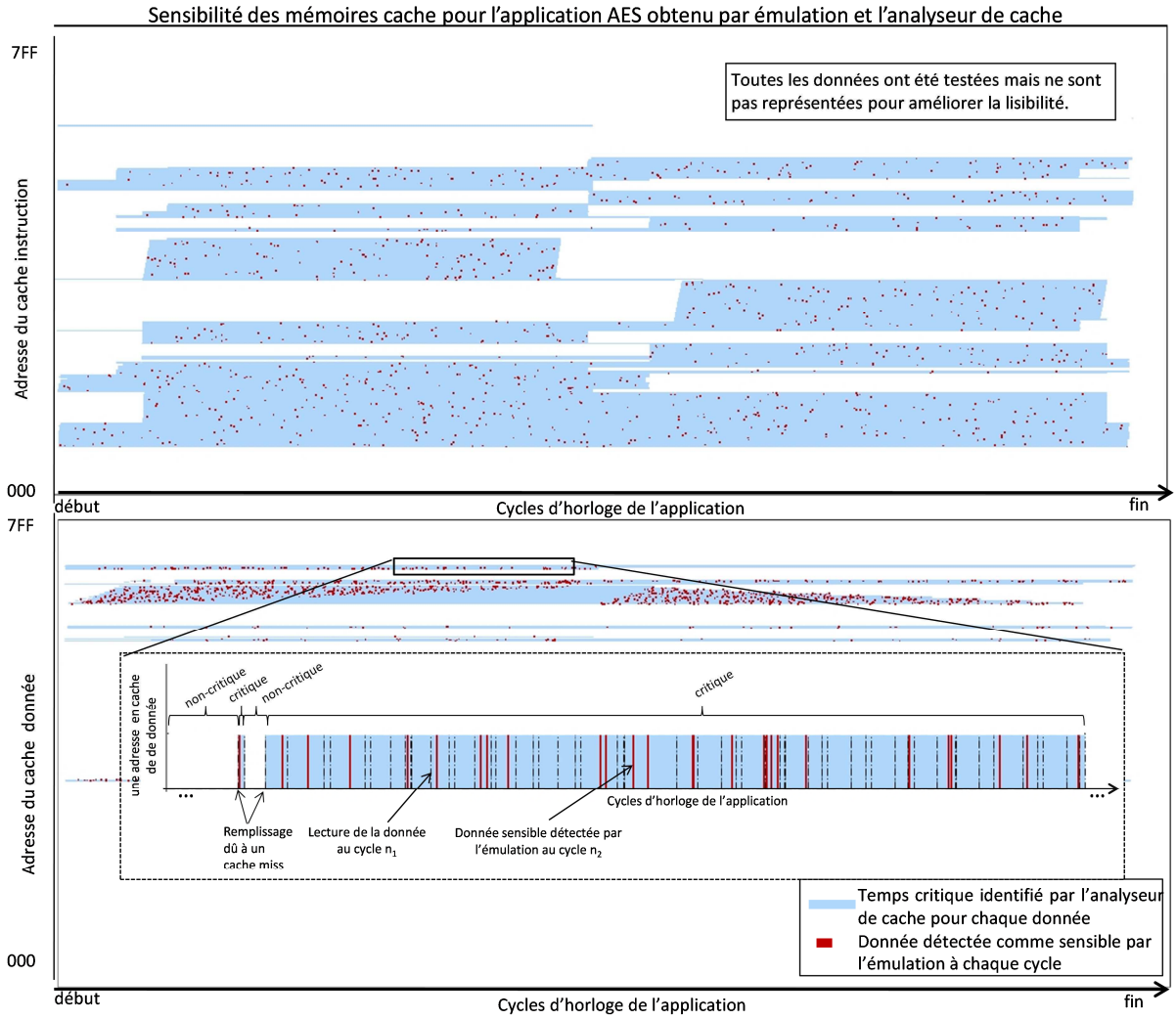


Figure III-7 : Représentation des données détectées comme sensibles par l'émulation et l'analyseur de cache pour l'application AES

La surestimation, qui garantit le caractère conservatif de l'analyse de cache, peut être observée dans la Figure III-7, l'émulation donnant une précision plus importante (bit par bit au lieu de la résolution de l'adresse). Le temps de vie d'une donnée en cache est représenté sur la Figure III-7 dans l'encadré, tous les cycles détectés comme sensible par l'émulation étant inclus

dans les temps critiques identifiés par l'analyseur de cache. L'application d'addition de vecteurs a aussi été validée mais les détails du résultat ne sont pas représentés.

La Figure III-8 présente le facteur de décote, en pourcentage, pour les caches de données et d'instructions, obtenu par trois méthodes : l'émulation, l'analyseur de cache et l'analyseur de cache avec identification des bits silencieux. Dans le chapitre II, une méthode pour améliorer la précision de l'estimation de la sensibilité du cache d'instructions a été présentée, en identifiant les bits silencieux du jeu d'instructions du microprocesseur. Ainsi, dans le cas d'une architecture SPARC, une base de données contenant tous les bits silencieux du jeu d'instructions a été développée et a permis de raffiner la connaissance des bits réellement sensibles.

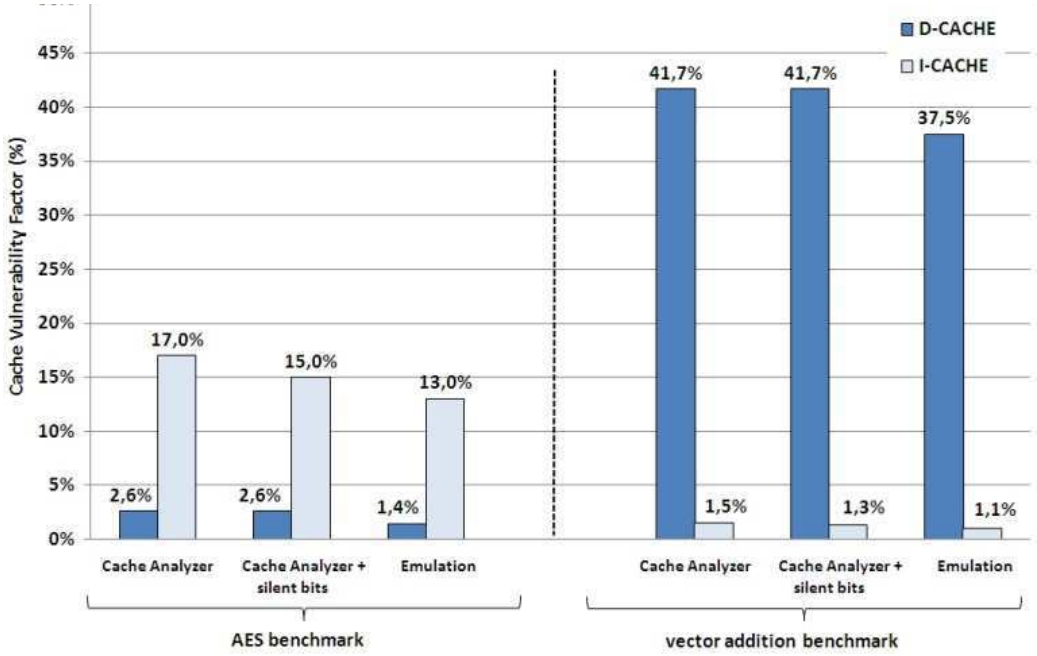


Figure III-8: Estimation du facteur de décote du cache de données et du cache d'instructions pour deux applications

Pour l'application AES, l'outil d'analyse de cache estime un facteur de décote pour le cache d'instructions de 17,0 % alors que l'émulation donne 13,0 %. Concernant l'application d'addition de vecteurs, le cache de données est estimé sensible à 41,7 % par l'analyseur de cache et 37,5 % par l'émulation. La contribution des bits silencieux est aussi représentée sur cette figure ; la sensibilité raffinée du cache d'instructions est 15,0 % pour l'AES et 1,3 % pour l'application d'addition de vecteurs. Même si ce facteur de décote est relativement faible, il permet d'obtenir une estimation plus précise de la sensibilité du cache d'instructions.

L'application d'addition de vecteurs est plus sensible à cause d'un effet de masquage inférieur des données, car il y a beaucoup plus de données utilisées. L'analyse de cache reste tout de même conservative.

La méthodologie n'a pu être validée que pour un cache en mode « *write-through* » sans protection, car, dans la version libre du LEON3, les mémoires cache ne peuvent pas être configurées en mode « *write-back* », et n'ont pas de moyen de protection disponible.

Si on suppose que les mémoires cache sont protégées par de la parité, on considère qu'un SEU dans ces mémoires cache peut être corrigé en implantant un mode de recouvrement. C'est pour cela que le cache de données et le cache d'instructions auront une sensibilité nulle pour des erreurs simples mais resteront plus sensibles aux MBU. Dans ce cas, les particules de l'environnement et la sensibilité intrinsèque de la technologie doivent être prises en compte.

La méthode proposée a ainsi été validée par la technique d'injection de fautes par émulation qui est reconnue comme étant très précise puisque toutes les cellules peuvent être corrompues. De plus, l'analyseur de cache permet de déterminer un facteur de décote pour les mémoires cache pour un ensemble d'applications dans un temps relativement court. L'autre avantage de cette méthode est qu'elle peut utiliser n'importe quel simulateur de type cycle-accurate donnant des informations sur la performance du cache.

III.4 Validation par des tests en accélérateur de particules

La deuxième étape de validation de la méthode d'analyse de cache consiste à comparer les résultats avec le moyen de test expérimental. Ainsi, deux essais en accélérateur ont été réalisés : le premier est l'irradiation aux neutrons 14 MeV, ce test ayant permis de vérifier la mise en œuvre du banc de test, et le second est un test d'irradiation aux protons 63 MeV.

III.4.1 Préparation des tests

La validation de la méthode d'analyse de cache sous faisceau de particules nécessite le même dispositif que celui simulé. Cette exigence conduit donc à plusieurs problèmes techniques.

Le premier est le choix du composant à tester : le DUT (Device Under Test). Nous avons choisi d'implanter le LEON3 sur un FPGA. Il existe trois grandes familles de FPGAs :

- les FPGAs à base d'anti-fusibles : la configuration du FPGA repose sur des fusibles et n'est donc programmable qu'une seule fois. Ces FPGAs sont donc totalement inadaptés pour des expérimentations pendant lesquelles différentes configurations doivent être évaluées.

- les FPGAs à base de mémoire SRAM : la configuration du circuit est mémorisée dans une mémoire SRAM qui est chargée à partir d'une mémoire externe non-volatile. Les principaux avantages de cette technologie sont son faible coût et la possibilité d'une programmation facile et rapide. Cependant, les FPGAs à base de mémoire SRAM ne conviennent pas aux tests sous faisceau de particules car les bits de configuration peuvent être corrompus, modifiant le comportement du SoC. De plus, le développement d'une technique de protection pour les bits de configuration n'était pas une option, car nous voulions seulement valider l'analyseur de cache.

- les FPGAs à base de mémoire Flash : la configuration du FPGA est stockée dans une mémoire non-volatile de type Flash, réputée pour sa très bonne tenue vis-à-vis des évènements radiatifs. De plus, ce composant peut être effacé ou reprogrammé à volonté et la sensibilité aux particules d'une mémoire Flash est très faible (même si non nulle). Ainsi en comparant tous les avantages et inconvénients des types de FPGAs, un FPGA à base de mémoire Flash a été sélectionné pour les essais sous faisceaux.

La seconde et principale difficulté pour valider la méthodologie a été le développement d'un système de détection de fautes permettant en temps réel, durant l'expérimentation, d'obtenir les mêmes informations qu'en simulation. En effet, pour valider qu'une donnée est bien critique, il est nécessaire de détecter précisément le lieu et l'instant où un SEU est apparu dans les mémoires cache, et s'il y a eu un impact sur le résultat final de l'application.

Enfin, les évènements cumulés durant l'exécution d'un cycle de l'application doivent être évités car il s'agit d'étudier l'effet d'un seul évènement (SEU) ou plusieurs (MBU) sur une donnée et sa propagation au niveau de l'application. Ainsi, un compromis sur le flux de particules entre la statistique d'évènements et la probabilité d'évènements cumulés doit être trouvé.

III.4.1.1 Caractéristiques de la cible

Les expériences ont été réalisées sur une carte de développement « Actel Cortex-M1 Enabled ProASIC3TM », [ACT08] équipée d'un FPGA à base de mémoire Flash de référence Proasic3 M1A3P1000 de chez Actel.

Le LEON3 est synthétisé sur ce FPGA de technologie 130 nm et utilise 18 KB de bloc RAM (BRAM), incluant 8 KB de BRAM utilisés pour les caches L1 de données et d'instructions. Les mémoires cache sont configurées de la manière suivante :

- un cache L1 de données et un cache L1 d'instructions de 4 KB chacun,

- chaque cache est composé d'une page, de 128 lignes par page et de 8 mots par ligne.

La carte de développement est aussi composée d'une mémoire Flash externe de 16 MB qui permet de stocker les applications à tester. Au démarrage, l'application est chargée de la mémoire Flash vers les blocs RAM. Une mémoire SRAM de 1 MB est aussi disponible sur la carte. Les GPIOs (General Purpose Input Output) sont utilisées pour envoyer les données nécessaires pour l'analyse des résultats et décrites par la suite. La fréquence d'horloge est limitée à 3MHz pour augmenter le temps d'exécution de l'application, et ainsi augmenter la probabilité d'obtenir une erreur durant un cycle d'application.

La comparaison entre l'analyseur de cache et l'accélérateur a été faite en utilisant deux applications. La première est l'application personnalisée d'addition de vecteurs décrite dans la partie III.3 et la seconde est un algorithme de transformée de Fourrier (FFT) de MiBench. Cette application a été choisie car elle présente une sensibilité du cache d'instructions assez importante (Figure III-6). Les résultats de ces deux applications sont envoyés à travers une interface série.

III.4.1.2 Développement du banc de test

a) Système de détection de fautes

Un bloc réutilisable (*Intellectuel Property*-IP) a été développé pour détecter la corruption des données en mémoire cache. Cette IP est basée sur le mécanisme de parité. De ce fait, seuls les évènements simples au sein des mêmes mots (SBU et MCU) sont détectés par notre système de détection de faute. Ainsi, les MBU de multiplicités paires ne sont pas détectés et ne sont pas pris en compte dans la validation.

Le système de détection est composé de trois modules : un module de détection de fautes, un module d'injection de fautes et un module de transmission d'erreurs. Comme illustré dans la Figure III-9, l'IP est localisée dans le FPGA entre le LEON3 et les mémoires cache, et surveille les transactions entre ces deux entités.

Le module de détection de fautes identifie les accès en lecture et en écriture dans la mémoire cache. Ensuite, il calcule les bits de parité de la donnée sollicitée. L'avantage d'utiliser des bits de parité est qu'il n'y a besoin que d'un seul bit par mot et son implémentation ne nécessite que quelques portes logiques de type « ou exclusif ».

Les bits de parité sont calculés à chaque écriture d'une donnée en cache puis sauvegardés dans la BRAM à la même adresse que la donnée. Cependant, les blocs RAM n'étant pas protégé, ils sont susceptibles d'être corrompus. Pour cette raison, afin de détecter des erreurs potentielles dans les BRAM, les bits de parité sont dupliqués. Lorsque le module de détection identifie un accès en lecture, il recalcule le bit de parité de cette donnée et compare le résultat à ceux sauvegardés dans l'IP à la même adresse. Trois cas sont possibles :

- Si les deux bits de parité sauvegardés sont identiques et sont égaux à celui calculé, cela signifie qu'il n'y a pas de corruption visible (en supposant qu'il n'a que des erreurs simples ou impaires).

- Si les deux bits de parité sont identiques mais sont différents de celui calculé, cela signifie qu'un SEU est apparu en mémoire cache. L'erreur est alors communiquée au module de transmission.

- Si les deux bits sauvegardés sont différents, cela signifie qu'il y a eu un SEU dans les BRAM contenant les bits de parité. Les résultats de ce cycle d'application ne sont pas pris en compte car seule la sensibilité des mémoires cache est étudiée.

Un module de transmission envoie les informations suivantes (Figure III-9) :

- l'horloge : le signal pour obtenir une information temporelle et pour synchroniser les données envoyées,

- le type d'accès en cache : lecture ou écriture,

- l'erreur de parité : pour signaler la corruption d'une donnée,

- l'adresse et la donnée corrompue.

Enfin, un module d'injection de fautes est implanté pour valider que l'IP fonctionne correctement, avant de passer sous faisceaux. Le principe est d'inverser le bit de parité calculé avant de le sauvegarder dans l'IP. Nous pouvons ensuite vérifier que l'IP transmet correctement les erreurs.

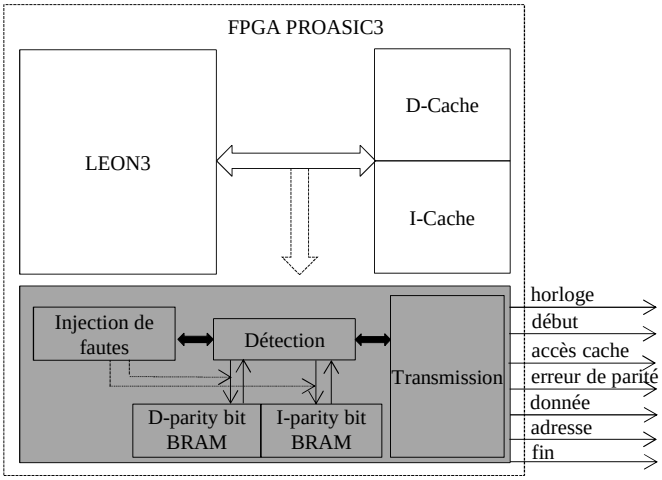


Figure III-9 : Description du système de détection de fautes intégré dans le processeur

b) Équipement de contrôle des tests

Après avoir développé le système de détection de fautes au sein des mémoires cache, il a fallu développer le banc de test dont le rôle est de récupérer les informations issues du système de détection et de piloter la carte sous faisceau de particules (Figure III-10).

Un programme dédié en LabVIEW (un environnement de programmation graphique), a été codé pour contrôler l'automatisation complète du banc de test composé de trois unités :

- L'unité de contrôle est fondée sur une plate-forme PXI (PCI eXtensions for Instrumentation), qui est destinée aux systèmes de mesure d'automatisation. Cette plateforme intègre un FPGA Virtex-5 LX3 programmable à l'aide du logiciel LabVIEW. Le logiciel développé dans le cadre de la thèse a pour fonction d'allumer, d'éteindre ou de redémarrer la carte en cas de non réponse du processeur, et d'arrêter le test à n'importe quel cycle. Le PXI gère aussi la réception des signaux en provenance du système de détection de fautes et le résultat final de l'application.

- La carte de développement est composée du DUT, le FPGA M1A3P1000. Il envoie aussi des signaux de synchronisation pour indiquer le début et la fin de l'exécution de l'application.

- L'unité d'interface utilisateur est une station de travail qui envoie des commandes à l'unité de contrôle et sauvegarde les résultats.

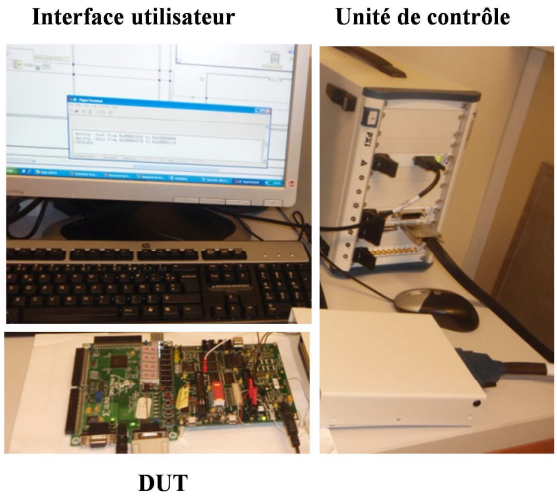


Figure III-10 : Banc de test

c) Les limites

L'équipement de contrôle du test et le système de détection de fautes présentent des limitations au niveau temporel et spatial. En effet, en termes de contrainte spatiale, le système de détection de fautes n'est capable de vérifier les erreurs que dans les 8 KB de mémoire cache, et non dans l'ensemble des BRAM. De plus, parmi l'ensemble des cellules des mémoires caches, seules les cellules sollicitées par le processeur sont surveillées par l'IP de détection de fautes. Ainsi un SEU impactant des cellules non sollicitées par le processeur ne sera pas identifié dans notre test. Ce qui signifie qu'un certain nombre de résultats n'est pas pris en compte.

La contrainte temporelle concerne le temps effectif d'observation et de détection de la faute. L'analyse de cache n'est réalisée que durant le temps de l'exécution de l'application, c'est pourquoi seules les erreurs apparaissant durant cette période sont observées. Un SEU qui apparaît en dehors de cette période n'est pas pris en compte dans notre expérimentation. La Figure III-11 montre les temps d'exécution $t_{exécution}$ de l'application par rapport au temps d'exécution d'un cycle complet de test. Le temps d'attente $t_{attente}$ correspond au temps nécessaire pour transmettre les résultats de l'application et les données provenant de l'IP de détection de fautes telles que les erreurs potentielles de parité, avec l'adresse et la donnée corrompue. Les autres temps morts correspondent au temps du reset et au temps de chargement de l'application de la mémoire Flash vers la RAM.

Figure III-11: Probabilité d'obtenir une erreur dans la mémoire cache selon le temps d'exécution d'un cycle de test.

En considérant ces deux principales contraintes, la probabilité d'obtenir un événement dans chaque cache pendant l'exécution de l'application est évaluée. Ainsi, on prend en compte la taille des caches par rapport à la taille totale des BRAM et le temps d'exécution de l'application par rapport au temps total d'un cycle de test. Cette probabilité est présentée dans la première ligne du Tableau III-2 pour les deux applications.

Les probabilités qu'un événement apparaisse dans d'autres éléments BRAM (hors mémoires cache) durant l'exécution de l'application et qu'un évènement apparaisse dans toutes les BRAM en dehors du temps d'exécution de l'application (durant le chargement, le reset ou le temps d'attentes) ont été estimées. C'est deux types d'évènements ne sont pas surveillés pendant les essais en accélérateur. La durée du temps de reset étant relativement faible, il est un peu probable d'avoir un évènement à ce moment-là.

Tableau III-2 : Probabilité d'obtenir un évènement en fonction du cache et du temps d'exécution de l'application

Type d'évènement	Addition de vecteurs	FFT
Évènement en cache durant l'exécution de l'application	33,8%	32,8%
Évènement dans d'autres éléments du processeur pendant l'exécution de l'application	31,2%	30,2%
Évènement en dehors de l'exécution de l'application	35,0%	37,0%

III.4.2 Test en accélérateur de particules neutrons

III.4.2.1 Conditions expérimentales

Pour valider le facteur de décote obtenu avec notre méthodologie, une première expérimentation a été réalisée aux neutrons 14 MeV, à la SODERN [SOD04]. Ce moyen de test a donné l'opportunité de vérifier le bon fonctionnement du système de détection de fautes et de l'équipement de contrôle. Les composants sous test doivent être disposés de manière radiale par rapport au flux de particules car la source émet les neutrons de manière iosotropique. Il est donc

nécessaire de calculer la distance du composant par rapport au centre du tube pour obtenir le flux de neutrons (qui varie en $1/r^2$). La Figure III-12 montre l'installation de l'expérimentation. Les neutrons mono-énergétiques sont produits grâce à la réaction de fusion du deutérium-tritium. Cette installation produit jusqu'à 10^9 neutron/s sur 4π stéradian.

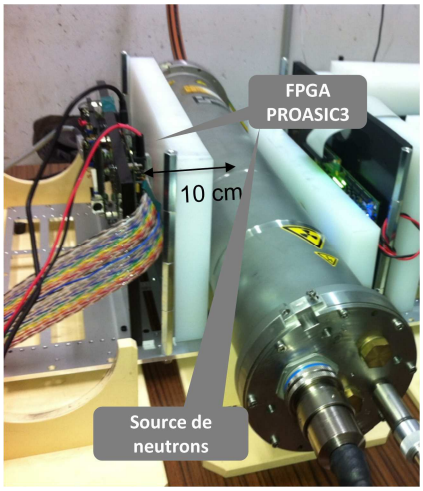


Figure III-12 : Positionnement de la carte par rapport au tube (gauche) à la SODERN

L'avantage de ce moyen de test est la disponibilité et la proximité de l'installation. De plus, il n'est pas nécessaire de procéder à l'ouverture du boîtier du composant, les neutrons pouvant traverser la matière. Cependant, la principale contrainte pour réaliser des tests avec une source de neutrons est le blindage des autres éléments de la carte. En effet, il a été nécessaire de trouver un compromis entre l'irradiation sous un faible flux du FPGA et la protection des autres composants de la carte sous test. Ainsi, le FPGA est situé à 4 cm du tube.

Pendant la moitié des tests (n°1 à 3), la SRAM de 1 MB et les autres composants de la carte étaient protégés par 3 cm de polyéthylène. Le flux de neutrons à la surface du FPGA était donc de $8x10^5$ n/cm²/s. Pour les tests n° 4 à 7, la protection de polyéthylène a été retirée pour augmenter le flux de neutrons à la surface de la puce. Il était donc de $1x10^6$ n/cm²/s néanmoins la probabilité d'obtenir une erreur dans la SRAM était supérieure.

III.4.2.2 Résultat obtenus

La section efficace pour une technologie 130 nm de FPGA est estimée à $2x10^{-14}$ cm²/bit [NOR10]. Elle correspond à la section-efficace d'une mémoire SRAM de 130nm. Les résultats obtenus sont exposés dans le Tableau III-3. Pour chaque test numéroté de 1 à 7, la fluence, le nombre d'erreurs attendues et le nombre d'erreurs observées durant l'irradiation sont donnés. Le

nombre d'erreurs attendues « global » est le nombre d'erreurs dans toutes les BRAM utilisées pour une fluence donnée. Comme mentionné précédemment, en raison des inconvénients de l'IP de détection, seules les erreurs en mémoire cache durant le temps d'exécution de l'application sont observées. Ainsi, une estimation de ce nombre d'erreurs est donnée dans la quatrième colonne du tableau. Dans la dernière colonne du Tableau III-3 , le nombre d'erreurs observées pendant l'exécution de l'application est présenté. Deux types d'erreurs se distinguent : les erreurs liées au résultat final de l'application et les erreurs détectées dans la mémoire cache par le système de détection de faute.

L'irradiation du composant exécutant l'application FFT n'a pas pu être réalisée par manque de temps. De ce fait, seul le programme « addition de vecteurs » a été testé aux neutrons. Le résultat final attendu à la fin de chaque exécution de ce programme est un mot de 32 bits « 0x0181F060 », correspondant au CRC calculé. Après irradiation, le résultat final peut être de trois types :

- Erreur de résultat final d'application : une valeur différente est obtenue,

- Une perte de séquencement : le programme se termine de façon inattendue,

- Pas d'erreur : la valeur du résultat final est correcte.

Tableau III-3 : Comparaison entre les erreurs détectées et les erreurs prédites

N° test	Fluence (n/cm^2)	Nombre d'erreurs attendues		Nombre d'erreurs observées durant l'irradiation	
		Global	Dans le cache durant l'application	Erreurs d'application	Erreurs de parité en cache
1	$4{,}7x10^8$	1,2	0,3	-	-
2	$1{,}7x10^9$	4,3	1,0	-	-
3	$9{,}5x10^8$	2,4	0,5	-	-
4	$1{,}7x10^9$	4,2	1,0	2	-
5	$9{,}8x10^8$	2,4	0,6	-	-
6	$9{,}1x10^8$	2,3	0,5	-	-
7	$4{,}5x10^9$	11,1	2,5	1	2
Temps total d'exposition aux radiations				123 min	

Pour les tests n° 1, 2, 3, 5 et 6, aucune erreur n'a été observée. Durant le test n°4, nous avons détecté deux erreurs de résultat final. La première erreur était « 0xD0C2B135 » et la seconde erreur était « 0xBD71B175 ». Cependant, le signal indiquant une erreur de parité n'a pas été déclenché. Pendant le test n°7, nous avons observé deux types d'erreurs :

- Au cycle d'application 3611, une erreur de parité ($erreur_3$) dans le cache d'instructions et une erreur de résultat d'application.

- Au cycle 4468, une erreur de parité ($erreur_4$) dans le cache d'instructions mais pas d'erreur de résultat d'application.

Le Tableau III-4 résume les principaux résultats de notre expérimentation.

Tableau III-4 : Comparaison entre les erreurs prédites et les erreurs détectées

Erreurs	Adresse	Fautes	Prédites par l'analyseur de cache
$erreur_1$	pas applicable		
$erreur_2$	pas applicable		
$erreur_3$	0x4000A958	oui	oui
$erreur_4$	0x4000A968	non	oui

Pour l'$erreur_1$ et l'$erreur_2$, il n'y a pas eu d'erreurs de parité détectées. Nous pouvons en déduire que ces erreurs ne sont pas reliées à la mémoire cache mais sont probablement dues à d'autres éléments du FPGA comme les bascules ou les BRAM (hors cache). Ainsi, ces résultats ne peuvent pas être utilisés pour valider la méthode d'analyse de cache. L'$erreur_3$ correspond à une erreur de parité dans le cache d'instructions à l'adresse "0x4000A958". Après vérification dans les résultats issus de l'analyseur de cache, cette adresse est incluse dans nos prédictions comme étant critique à ce cycle d'horloge. L'$erreur_4$ est apparue à l'adresse "0x4000A968" et n'a pas engendré d'erreur dans le résultat final de l'application. L'analyseur de cache avait considéré cette erreur comme critique.

Nos essais effectués aux neutrons 14 MeV n'ont donné que quelques résultats à cause du temps d'irradiation très court associés à un flux de particules relativement faibles. Cependant, plusieurs modes de défaillance ont été observés en cohérence avec les prédictions de l'analyseur de cache. Ce test expérimental a surtout permis de vérifier l'exécution correcte du dispositif développé pour la détection de fautes dans les mémoires cache.

III.4.3 Test en accélérateur de particules protons

III.4.3.1 Conditions expérimentales

Une autre campagne de test a été réalisée avec une source de protons 63 MeV, permettant notamment de disposer de flux de près de 100 fois plus élevés, à l'Université Catholique de

Louvain (UCL). La Figure III-13 montre la mise en place du dispositif de test sous faisceau de protons.

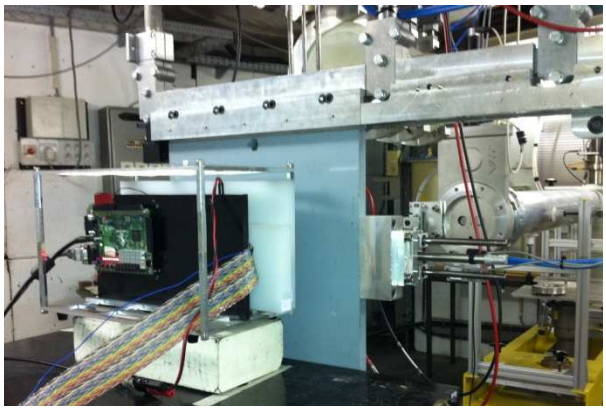


Figure III-13 : Composant sous le faisceau (UCL)

Tous les autres composants de la carte de test sont protégés avec 6 cm de polyéthylène. Le flux de protons à la surface de la puce est de $2x10^7$ p/cm²/s.

III.4.3.2 Résultat obtenus

La section efficace pour un FPGA d'une technologie de 130 nm est estimée à $9x10^{-14}$ cm²/bit [REZ08]. Pendant l'irradiation, deux types d'erreurs sont surveillés comme pour les tests d'irradiation aux neutrons : les erreurs en mémoire cache détectées par l'IP présentée à la section précédente et les erreurs de résultat final d'application.

Le nombre d'erreurs observées pendant les tests protons est beaucoup plus important que celui obtenu durant les tests neutrons. Par ailleurs, les erreurs observées pendant l'irradiation sont classées et vérifiées de la manière suivante :

- cas n°1 : une erreur de parité en cache et une erreur de résultat final d'application sont détectées, et l'adresse de la donnée corrompue en mémoire cache correspond à une donnée critique identifiée par l'analyseur de cache. Ce cas permet de valider les prédictions de l'analyseur de cache.

- cas n°2 : une erreur de parité en cache et une erreur de résultat final d'application sont détectées mais l'adresse de la donnée corrompue correspond à une donnée identifiée comme non-critique par l'analyseur de cache.

- cas n°3 : une erreur de parité en cache est détectée mais aucune erreur sur le résultat final n'est observée. Ainsi, c'est une donnée non-critique. Si l'analyseur de cache prédit l'adresse comme critique, cela signifie que l'analyseur de cache a surestimé la sensibilité de la mémoire cache.

- cas n°4 : une erreur de parité en cache est détectée, il n'y a pas d'erreur observée sur le résultat final. Si l'analyseur de cache identifie cette adresse comme non-critique, ceci valide l'analyseur de cache.

- cas n°5 : pas d'erreur de parité en cache mais une erreur d'application est identifiée. Ce cas n'est pas pris en compte dans notre étude. En effet, comme nous voulons seulement valider l'analyseur de cache, les erreurs de la logique du FPGA, des registres, du pipeline et d'autres parties du processeur ne sont pas considérées dans notre étude (mais restent néanmoins comptabilisées, ce qui permet de vérifier qu'elles ne sont pas prépondérantes).

Tableau III-5 : Classification des erreurs observées durant l'irradiation

cas	Erreurs détectées durant l'irradiation		Prédites par l'analyseur de cache	Résultat
	Mémoire cache	Résultat de l'application		
cas n°1	erreur	erreur	critique	validation
cas n°2	erreur	erreur	pas critique	sous-estimation
cas n°3	erreur	pas d'erreur	critique	sur-estimation
cas n°4	erreur	pas d'erreur	pas critique	validation
cas n°5	pas d'erreur	erreur/ pas d'erreur	pas applicable	

Ainsi, pour valider la méthode proposée, nous avons vérifié que toutes les adresses détectées comme critiques par l'accélérateur sont aussi détectées par l'analyseur de cache (cas n°1). Nous avons aussi vérifié que l'impact d'un SEU sur une donnée identifiée comme non-critique par l'analyseur de cache ne conduit pas à une erreur de résultat d'application (cas n°4).

Tableau III-6 : Validation des données critiques et non-critiques du cache d'instructions

	Donnée critique		Donnée non-critique	
Programme	Erreurs obtenues en accélérateur	Prédites par l'analyseur de cache	Erreurs obtenues en accélérateur	Prédites par l'analyseur de cache
FFT	56	56	1	1
addition de vecteurs	30	30	42	42

Tableau III-7 : Validation des données critiques et non-critiques du cache de données

	Donnée critique		Donnée non-critique	
Programme	Erreurs obtenues en accélérateur	Prédites par l'analyseur de cache	Erreurs obtenues en accélérateur	Prédites par l'analyseur de cache
FFT	1*	0	26	26
addition de vecteurs	23*	21	0	0

*Seulement 3 erreurs critiques sous faisceau ont été identifiées comme non-critiques par l'analyseur de cache. Il peut s'agir d'évènements multiples dans le cache ou dans les bascules.

Le Tableau III-6 présente le nombre de données critiques et non-critiques dans le cache d'instructions détectées en accélérateur de particules et par la technique d'analyse de cache pour les applications « addition de vecteurs » et FFT. La partie de gauche représente le nombre d'adresses sensibles du cache d'instructions détectées durant les tests en accélérateur et le nombre de données critiques détectées par l'analyseur de cache. La représentation du tableau ne permet pas de voir que ce sont bien les mêmes éléments qui sont identifiés mais en revanche, nous nous sommes assurés de ce point

Pour l'application d'addition de vecteurs, les 30 erreurs du cache d'instructions détectées durant les tests en accélérateurs ont toutes été prédites correctement par l'analyseur de cache. L'application FFT a aussi été validée. Un cas de donnée non-critique a été identifié durant les tests en accélérateur et cette donnée en particulier a aussi été détectée comme non-critique par l'analyseur de cache.

La même représentation est illustrée pour le cache de données dans le Tableau III-7. Une donnée critique a aussi été détectée dans le cache de données durant les tests en accélérateur mais cette donnée n'a pas été identifié comme sensible par l'analyseur de cache. Notre supposition est que la plupart des éléments logiques du FPGA n'est pas protégée : une erreur ou aussi un possible évènement multiple peut apparaître dans la logique ou dans les registres. Ce cas n'est apparu que trois fois durant tous les tests.

Une analyse plus détaillée des résultats a été réalisée pour valider notre méthodologie pour le cache de données et pour l'application « addition de vecteurs ». Pour cela, l'application a été exécutée en simulation et il a été vérifié que pour un certain nombre de données, l'inversion logicielle de bit produit bien la même erreur de résultat final d'application que celle obtenue durant les essais. Ainsi, le Tableau III-8 décrit quelques résultats obtenus à l'issu de cette analyse. La valeur de la donnée corrompue ainsi que son adresse dans l'application sont indiquées. Par

exemple, l'adresse "0x4000E004" correspond à la valeur de "x [0]"de l'application d'addition de vecteurs. Comme nous n'exécutons que des opérations de lecture, la valeur de la donnée "x [0]" reste à "0" jusqu'à la fin de l'application. Puis en simulation, la valeur de "x [0]" est corrompue à plusieurs endroits et nous obtenons la même erreur de résultat d'application qu'en accélérateur.

Tableau III-8 : Vérification des valeurs corrompues dans le cache de données pour l'application « addition de vecteurs »

Adresse	valeur correspondante dans l'application	valeur originale	valeur corrompue	Erreur d'application	Vérifié en simulation
4000E004	x [0]=0	0x00000000	0x04000000	0x0af52cb4	oui
4000E010	x [3]=3	0x00000003	0x00000007	0x0beb746a	oui
4000E014	x [4]=4	0x00000004	0x00000204	0xea013259	oui
4000E020	x [7]=7	0x00000007	0x00000047	0x18979d2a	oui
4000E00C	x [2]=2	0x00000002	0x00080002	0x7cecc122	oui
4000E01C	x [6]=6	0x00000006	0x00004006	0x8b7298e3	oui
4000E76C	x [487]=1E7	0x000001e7	0x008001e7	0x1d7459a2	oui
4000E790	x [484]=484	0x000001e4	0x000021e4	0x9e75f295	oui
4000E7C0	x [496]=1F0	0x000001f0	0x000009f0	0xe35be74b	oui

I.4.2.2.1. Surestimation de l'analyseur de cache

La sensibilité des mémoires caches a été évaluée avec l'analyseur de cache pour les deux applications. Ainsi, la méthode décrite dans le chapitre II a été appliquée à l'aide de ModelSim pour le LEON3. Les données critiques de chacune des mémoires cache ont été identifiées et les temps critiques calculés.

Ensuite à partir de ces informations, les facteurs de décote ont été calculés en pourcentage pour chaque mémoire cache, en tenant compte de la taille des données, du temps d'exécution total de l'application et de la taille totale du cache (Eq. II-5).

$$CacheVF = \frac{\sum TempsCritique \times TailleMot}{TempsExecutionTotal \times TailleCache} \qquad \textbf{Eq. II-5}$$

Les résultats sont présentés dans le Tableau III-9. Pour chaque application, « addition de vecteurs » et FFT, le pourcentage obtenu avec l'analyseur de cache est présenté dans la colonne « Sensibilité du cache obtenue avec l'analyseur de cache pour le cache d'instructions « I-Cache » et le cache de données « D-cache ».

La sensibilité du cache estimée durant les tests en accélérateur « $CacheVF_{accelerateur}$ » est obtenue en tenant compte du nombre d'erreurs détectées (l'ensemble des données critiques) durant l'irradiation sur le nombre éventuel (cas n°1 du Tableau III-5). Ce nombre d'erreurs éventuel est calculé en fonction de la sensibilité statique du FPGA aux protons ($9x10^{-14}$ cm²/bit), du flux de protons 63 MeV, du temps d'exposition de l'application et de la taille totale du cache (Eq. III-1) :

$$CacheVF_{accelerator} = \frac{\sum donnéeCritique}{TempsExecutionTotal \times fluence_{proton} \times TailleCache \times \sigma} \quad \textbf{Eq. III-1}$$

La différence d'estimation est illustrée dans le Tableau III-9 ; ceci montre la différence de niveau de précision entre les deux méthodologies. Par exemple, l'analyseur de cache permet d'estimer un facteur de décote pour le cache d'instructions de 52 % alors que l'accélérateur donne 18,5 % pour l'application FFT. Comme le nombre d'évènement pris en compte en accélérateur est bien inférieur à celui prédit, la différence n'est pas surprenante, et a été observée dans des comparaisons précédentes.

Nous pouvons observer que le cache de données est plus sensible pour l'application addition de vecteurs. Ceci est dû à l'effet de masquage, sachant que l'on a utilisé un grand nombre d'additions sur beaucoup de données. L'analyse reste conservative.

Tableau III-9 : Surestimation de l'analyseur de cache

Application	Mémoire cache	Sensibilité du cache obtenue avec l'analyseur de cache	Sensibilité du cache obtenue avec l'accélérateur
FFT	I-Cache	52,0%	18,5% (+/-26,7%)
	D-Cache	5,2%	<0,3%
addition de vecteurs	I-Cache	7,9%	4,6% (+/-35,7%)
	D-Cache	94,7%	3,2% (+/-42,7%)

Dans la Figure III-14, nous résumons une estimation du taux d'erreurs des mémoires cache de données et d'instructions, pour les applications FFT et « addition de vecteurs », obtenue avec trois méthodes : pire-cas, l'analyseur de cache, et le test en accélérateur. L'estimation pire cas est le nombre d'erreurs théorique obtenu avec la même fluence qu'en accélérateur (nous considérons que toutes les cellules de la mémoire cache sont sensibles). Il est visible que l'analyseur de cache, malgré son exécution très rapide, permet dans beaucoup de cas d'obtenir une évaluation plus réaliste tout en restant conservative.

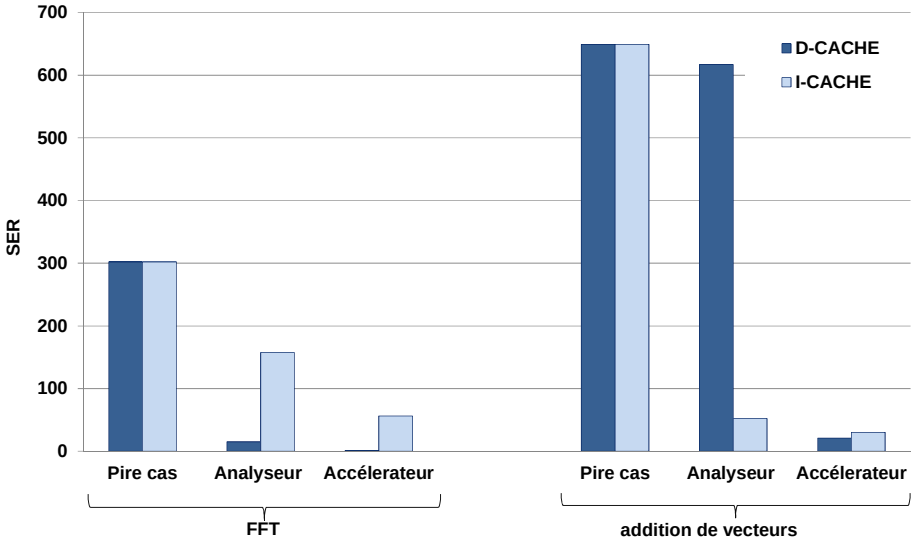


Figure III-14 : Estimation du taux d'erreur des mémoires cache avec trois moyens

III.5 Conclusion

La méthode proposée au chapitre précédent a été validée sur un prototype basé sur le processeur LEON3 par une technique d'injection de fautes, reconnue comme étant assez précise car toutes les cellules peuvent être corrompues. Les résultats ont montré des évaluations cohérentes avec celles de l'analyseur de cache pour deux applications. L'avantage de la méthode d'analyse de cache est qu'on peut déterminer un facteur de décote pour les mémoires cache pour un ensemble d'applications dans un temps relativement court. Ce temps ne dépend que de la vitesse du simulateur. L'autre bénéfice de cette méthode est qu'elle peut être implémentée avec n'importe quel simulateur de type cycle-accurate donnant des informations sur la performance du cache. Elle ne repose pas sur l'injection aléatoire de fautes et est de ce fait plus couvrante.

Une deuxième étape de validation a été réalisée. Les tests en accélérateur de particules sont aujourd'hui incontournables pour déterminer les sections efficaces statiques des composants et calculer les taux d'erreurs. Les résultats obtenus avec l'analyseur de cache ont été confrontés aux résultats issus de l'irradiation du LEON3 en mode dynamique, c'est-à-dire exécutant une application. Pour cela, un système de détection de fautes fondé sur le mécanisme de parité a été développé pour les mémoires cache et un banc de test spécifique pour récupérer les résultats en temps réel a été mis en place.

Les essais avec le générateur de neutrons 14 MeV ont permis de valider quelques données mais ont surtout aidé à valider l'ensemble du système de contrôle de test. Les tests réalisés à l'aide

d'une source de protons 63 MeV ont permis de valider un plus grand nombre de données critiques. Cependant, on peut noter une surestimation de la méthodologie proposée car elle est fondée sur le comportement des mémoires cache et non pas sur l'impact de la faute sur l'application.

Ce chapitre a donc présenté une validation expérimentale de la méthode d'identification de donnée critique en se basant sur les signaux de performances issus d'un simulateur. Dans le chapitre suivant, une extension de la méthode est proposée pour des processeurs matériels. Elle peut être appliquée sur la plupart des microprocesseurs COTS disposant de registres de performance.

Chapitre IV : Adaptation de la méthodologie sur cible matérielle

Dans le chapitre III, la validation expérimentale de la méthode d'analyse de cache, fondée sur l'utilisation d'un simulateur, a été présentée. Cette approche a été validée grâce à deux moyens pertinents : l'injection de fautes et les tests en accélérateurs. Cependant, certains processeurs COTS ne disposent pas de simulateur de type « cycle-accurate » ou ces simulateurs sont difficilement accessibles. C'est pourquoi la méthode d'analyse des données critiques dans les mémoires cache a été adaptée à des cibles matérielles de processeurs. La plupart des processeurs COTS d'architecture récente (notamment les processeurs multi-cœurs) possèdent des unités de débogage assez complexes visant à aider les développeurs. Ainsi l'approche que nous proposons ici repose sur l'exploitation des fonctionnalités internes de ces systèmes et notamment les registres de performance. Dans ce chapitre, une première partie est consacrée à l'étude de la problématique. Puis, la mise en œuvre de cette méthode sur une cible matérielle sera présentée ainsi qu'une validation partielle des résultats par une méthode d'injection de fautes par interface de débogage.

IV.1 Problématique

IV.1.1 Idée de base

Comme démontré dans le chapitre III, la méthode d'analyse de cache permet d'obtenir une sensibilité plus précise des mémoires cache. Elle consiste à exécuter une application sur un simulateur de processeur et à surveiller les signaux propres à celui-ci, comme les signaux de performance et les signaux d'accès en cache. Ces informations issues du simulateur aident l'utilisateur à identifier les données critiques du cache. En calculant les temps d'exposition de ces données critiques, on obtient un facteur de décote lié à l'application : $\boldsymbol{\pi_{Appli}}$. Ce facteur permet donc de réduire le taux de défaillance par rapport au pire cas.

Cette méthode nécessite un simulateur de type cycle-accurate qui, selon la famille et le type de processeur peut être parfois difficile à obtenir pour un utilisateur final. Prenons l'exemple du simulateur du processeur PowerPC970 d'IBM : celui-ci n'est disponible que dans sa version fonctionnelle et non cycle-accurate [BOH04]. Par conséquent, en raison de cette limitation, il a été

envisagé d'appliquer notre méthode d'analyse directement sur la cible si celle-ci est disponible durant le cycle de production d'un équipement sans le simulateur correspondant. Cette deuxième méthode repose sur l'utilisation des fonctionnalités complexes qui sont intégrées dans les processeurs modernes, tels que les compteurs de performance et l'unité de sauvegarde de l'activité du processeur appelée « hardware trace ». L'avantage de ce déploiement sur cible est qu'il n'est plus nécessaire de disposer d'un simulateur cycle accurate du processeur, l'analyse s'effectuant directement sur la cible matérielle (Figure IV-1). Elle nécessite en revanche la plateforme matérielle.

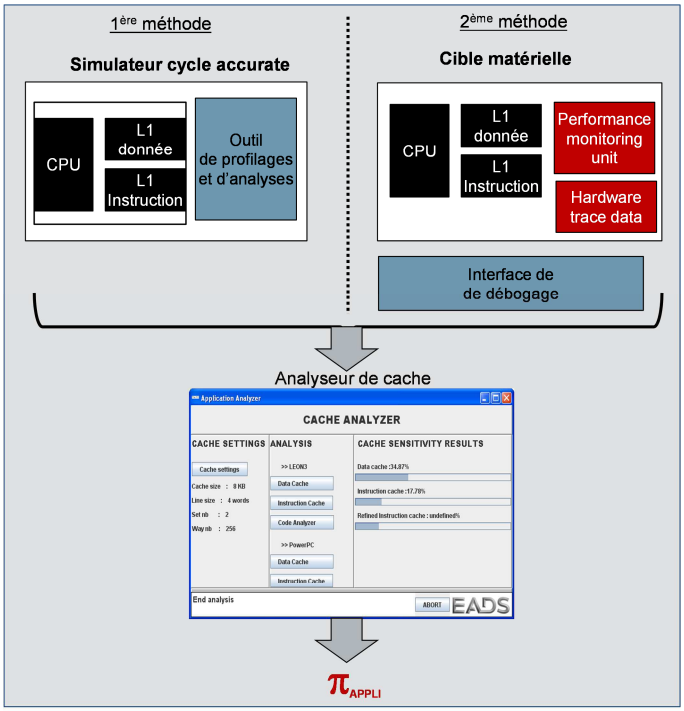


Figure IV-1 : Estimation du facteur de décote lié à l'application par deux méthodes selon la cible disponible : simulateur cycle accurate ou cible matérielle

IV.1.2 Les fonctionnalités avancées des processeurs modernes

La complexité des systèmes embarqués étant en constante augmentation, il y a un besoin croissant de débogage efficace pour réduire le temps de développement. On peut distinguer deux outils généralement utilisés par les développeurs : le débogueur et l'analyseur de performance. Le débogage permet d'aider le programmeur à identifier les erreurs dans son code. L'analyse de performance permet d'améliorer le comportement d'une application exécutée en mesurant des métriques. De nombreux outils existent permettant d'aider les développeurs à améliorer les performances d'un système mais la plupart d'entre eux nécessitent la modification de l'application exécutée sur le processeur. Néanmoins, depuis quelques années, les fabricants de processeur ont

intégré, au sein des processeurs, des éléments matériels de débogage n'ayant pas d'impact sur le système. Parmi ces éléments, on distingue le « hardware trace » et les compteurs de performance.

IV.1.2.1 Hardware Trace

La trace est définie comme une suite d'états observables, ordonnés dans le temps. L'unité de sauvegarde de l'activité du processeur, « hardware trace » est une unité qui enregistre l'historique des instructions exécutées par le processeur. Cette unité permet aussi de sauvegarder les adresses et les valeurs correspondant aux accès en mémoire. Ainsi, elle apporte une visibilité du comportement interne du processeur. Une logique dédiée est utilisée pour diffuser les informations liées à la trace, et il n'y a pas d'impact sur l'exécution du code. Ceci permet de collecter la trace de manière non-intrusive. Il existe des interfaces comme les sondes ou les analyseurs de traces pour se connecter directement au microprocesseur afin de recueillir les informations enregistrées. Cette fonctionnalité offre aussi la possibilité d'ajouter un marqueur temporel à chaque information collectée.

IV.1.2.2 Les compteurs de performance

L'unité de suivi de performance, appelée « Performance Monitoring Unit » (PMU), est utilisée en général pour l'analyse de performance et pour identifier les causes d'une perte de performance. Cette unité examine les occurrences d'évènements spécifiques, liés à la microarchitecture du composant, en incrémentant les registres de compteurs de performance lorsque l'évènement apparait.

Ces évènements peuvent être les *cache miss*, les pré-chargements de mémoire, les branchements prédits et non réalisés, …. Les compteurs sont accessibles par l'utilisateur et permettent ainsi d'avoir des informations de bas niveau de l'état du processeur lorsqu'il exécute son code.

IV.1.3 Exemple de processeurs

Notre étude porte sur une cible de processeur multicœur : le QorIQ P4080 d'architecture PowerPC [FRE12]. Cependant, de plus en plus de familles de processeurs ajoutent des unités de performance et de trace matérielle. La plupart des processeurs modernes PowerPC possèdent des unités de débogage assez complexes qui seront présentées plus en détails dans la partie suivante.

Les processeurs ARM incluent une interface standardisée pour la trace appelée Embedded Trace Macrocell (ETM) qui génère des informations liées aux instructions et aux données utilisées par le CPU. Ainsi grâce aux débogueurs du marché, comme par exemple l'outil TRACE32 de

Lauterbach [LAU11], il est possible de collecter la trace de toutes les instructions exécutées. Les processeurs ARM7, ARM9 et ARM11 possèdent cette unité. D'autres processeurs ARM (ARM11, Cortex-R et Cortex-A) contiennent aussi une unité de suivi de performance HPC (« Hardware Performance counters »). Les HPC sont configurables pour compter un certain nombre d'évènements. Les processeurs possèdent entre 2 et 6 registres de compteurs d'évènements. Ils peuvent être programmés soit de manière logicielle soit par une interface de débogage.

Les processeurs d'Intel des dernières générations disposent de compteurs de suivi de performance au sein des PMU. Ceci concerne les processeurs de types Intel Xeon 5500, 5600, 7500, E5, E7 et Core i7 [INT13]. Il est ainsi possible de surveiller le nombre de *cache hit* et de *cache miss* pour chaque niveau de cache.

Les processeurs AMD disposent de compteurs de suivi de performance (PMC) pour mesurer des évènements matériels dans un programme exécuté. Ils peuvent entre autre mesurer le nombre d'occurrences d'un évènement comme le « *cache miss* » pour le cache de données, ou le cache d'instructions, mais aussi le chargement d'instructions, ou l'accès en cache de données.

Ainsi, on peut noter que la grande majorité des processeurs COTS récents comporte des fonctionnalités internes de débogage et d'optimisation.

IV.2 Mise en œuvre de l'approche

La méthode d'estimation du facteur de décote lié à l'application a été mise en œuvre pour le processeur P4080. Le principe est d'exécuter l'application sur la cible matérielle, après avoir configuré la trace matérielle et les compteurs de performance pour identifier les données critiques du cache.

IV.2.1 Description des fonctionnalités internes

IV.2.1.1 Présentation générale

Le microprocesseur multi-cœur QorIQ P4080 de chez Freescale, aussi appelé « Multiprocessor System on Chip » (MP-SoC) est destiné aux applications à très haut débit nécessitant une grande puissance de calcul. Il peut fonctionner jusqu'à une fréquence de 1,5 GHz. La plateforme intègre des unités matérielles pour le traitement rapide d'envoi des données. Le P4080 contient huit cœurs d'architecture PowerPC e500mc partageant une mémoire principale et des périphériques internes. Le

réseau d'interconnections, appelé « CoreNet », est utilisé pour connecter tous les blocs matériels du P4080 (Figure IV-2).

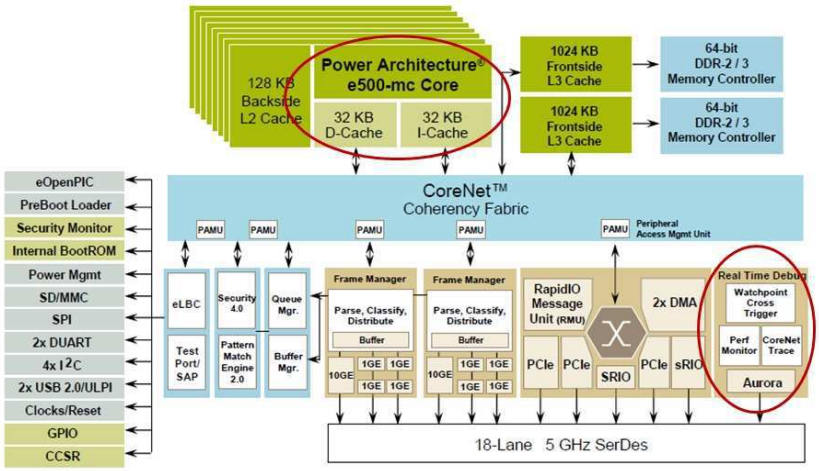


Figure IV-2 : Schéma du processeur cible P4080 [FRE12]

L'étude n'a porté que sur l'un des cœurs du microprocesseur, c'est pourquoi seule l'architecture d'un cœur est décrite dans la suite du manuscrit.

Il s'agit d'un processeur 32 bits d'architecture superscalaire qui possède les caractéristiques suivantes :

- un pipeline entier constitué de 7 étages,
- une architecture Harvard : un cache de données et un cache d'instructions séparés de taille 32 KB,
- un cache de niveau L2 de taille 128 KB,
- des mémoires tampon internes de lecture et écriture (DLFB).

Chaque cache de niveau L1 est configuré en 64 pages de 8 lignes et chaque ligne contient 16 mots de 32 bits. Les caches de niveau L1 sont protégés par un mécanisme de parité avec un bit de parité par octet pour le cache de données, et un bit par mot pour le cache d'instructions. Le cache L2 n'est pas décrit dans le manuscrit car nous n'avons pas eu le temps d'appliquer la méthode pour ce niveau de cache pour ce processeur.

Par ailleurs, les mémoires cache sont configurées par défaut en mode « *write-through* » et il n'y a pas d'allocation en mémoire cache de données lors d'une écriture et d'un *cache miss*. En plus des mémoires cache, chaque cœur intègre des mémoires tampon de stockage et de chargement (Data Line Fill Buffer - DLFB) qui se comportent comme des mémoires cache qu'il n'est pas possible de désactiver. Cependant, dans le cas de notre analyse des mémoires cache, ces mémoires tampon ne

sont pas prises en compte, bien qu'elles soient des éléments sensibles du processeur car aucun moyen de protection n'est mis en œuvre.

IV.2.1.2 Architecture de débogage du P4080

La plateforme P4080 intègre une infrastructure APDA (« Advanced QorIQ Plateform debug Architecture ») [FRE10] de débogage permettant le traçage, le suivi de performance et le contrôle du processeur. Chaque élément matériel (DDR, CoreNet…) et chaque cœur e500mc de la plateforme a une logique dédiée de débogage. La Figure IV-3 représente l'architecture de débogage d'un cœur e500mc et illustre les unités de débogage utiles à notre étude.

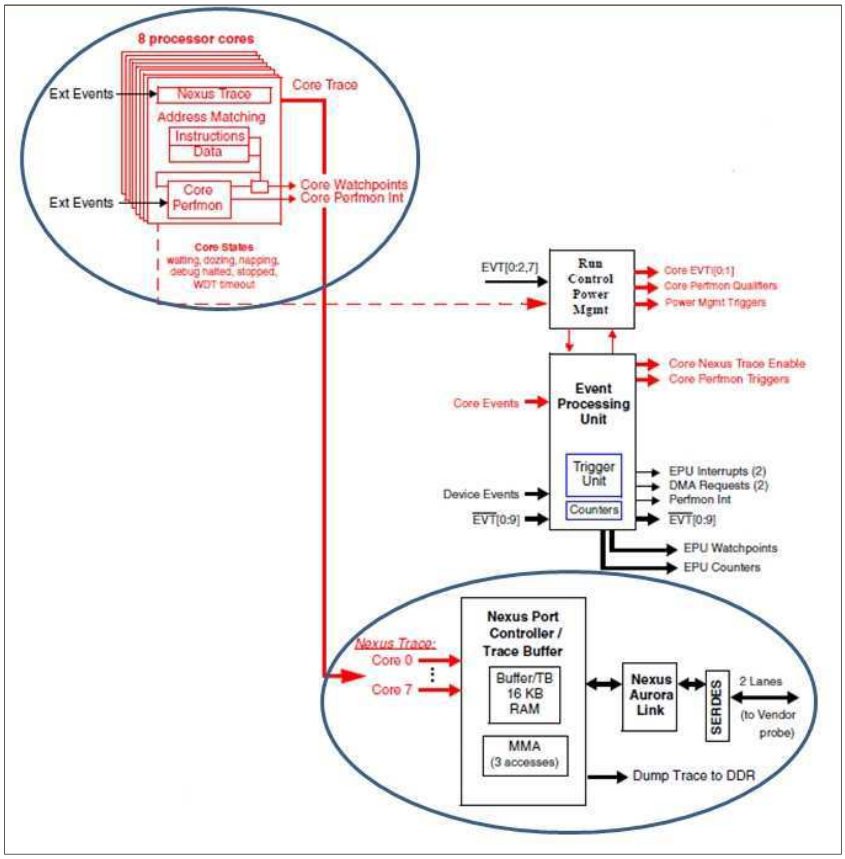


Figure IV-3 : Vue détaillée de l'architecture de débogage du P4080 [FRE10]

L'unité EPU (« Event Processing Unit ») permet de combiner ou de filtrer un certain nombre d'événements en provenance de plusieurs unités du processeur. Elle permet aussi de générer d'autres fonctionnalités de débogage.

L'unité « Nexus Port Controller » contient une mémoire tampon de 16 KB dédiée à la trace et gère la connexion vers la sonde de débogage.

L'unité de débogage interne au cœur e500 va être plus particulièrement détaillée. Les fonctionnalités internes du processeur sont représentées en Figure IV-3 : l'unité de suivi de performance, l'unité de trace (« Nexus Trace »), ainsi que l'unité de comparateur d'adresses de donnée et d'instruction.

a) Les compteurs de performance :

L'architecture e500mc contient un ensemble de registres dédiés au suivi de performance PMR (« Performance monitor register ») : des registres de comptage et des registres de configuration. Pour cette architecture, il est possible d'avoir jusqu'à 4 compteurs de performance (PMC0-PMC3). Pour chacun de ces compteurs, la configuration s'effectue en deux étapes : la sélection du type d'évènement à compter avec le registre « PMLCa » et celle des conditions de déclenchement ou d'arrêt du compteur « PMLCb ». Il est possible de compter jusqu'à 180 évènements. Le Tableau IV-1 montre les évènements les plus pertinents pour l'analyseur de cache.

Tableau IV-1 : Exemples d'évènements liés à la performance des PMR

Référence	Évènement	Description
Réf 1	Cycles du processeur	Cycle d'horloge du processeur.
Réf 2	Instructions terminées	Lorsqu'une instruction a fini d'être exécutée (dernier étage du pipeline).
Com 4	Instructions chargées	Instruction chargée au niveau du pipeline.
Com 60	Rechargement du cache d'instructions L1	Lorsqu'une ligne de cache est rechargée dû à l'étage de chargement du pipeline.
Com 41	Rechargement du cache de données L1	Lorsqu'une ligne de cache est rechargée.
Com 9	Opération de chargement terminée	À chaque fois qu'une instruction de chargement a fini d'être exécutée.
Com 10	Opération de stockage terminée	À chaque fois qu'une instruction de stockage a fini d'être exécutée.

Il n'est pas possible de surveiller le nombre de *cache hit* pour ce type d'architecture pour les caches L1. Cependant, les taux de *cache miss* pour le cache de données et le cache d'instructions se mesurent de la manière suivante [FRE08] :

$L1_{instruction}$*cache miss* = Com60 / Ref2

= rechargement du cache d'instructions L1/ instructions terminées

$L1_{data}$*cache miss* = Com 41 / (Com 9 + Com 10)

= rechargement du cache de données L1 / (opération de chargement terminée+ opération de stockage terminée)

b) L'unité de trace hardware

Le module de trace « Nexus Trace » est conforme au standard IEEE-ISTO 5001. Ce module permet en particulier de tracer le programme exécuté *(« Program trace »)*, les données sollicitées *(« Data trace »),* et des points de surveillance (« *Watchpoint trace* »). Les autres options ne sont pas détaillées car elles ne sont pas utilisées dans l'implantation de la méthodologie.

L'utilisation de la trace se décompose en trois étapes :

- la configuration de la trace,

- la collection et le décodage de la trace,

- la lecture de la trace.

Le module de trace se configure par le registre « Nexus Development Control Register » (DC1). Il est possible de sélectionner les options suivantes :

- La trace d'exécution d'un programme : elle permet de ne tracer que les instructions de branchement et de prédiction.

- La trace des données : le cœur e500mc présente une limitation au niveau de la trace des données car il ne peut tracer que les instructions de stockage.

- La trace des points de surveillance permet de surveiller lors de l'exécution d'un programme des évènements spécifiques tels que décrits dans le Tableau IV-2. Ces évènements spécifiques sont indiqués dans la fiche technique produit de la société Freescale [FRE08].

Tableau IV-2 : Exemples des évènements spécifiques de débogage

Nom de l'évènement	Description
Event IAC1	Évènement lié au registre 1 du comparateur d'adresses d'instructions
Event IAC2	Évènement lié au registre 2 du comparateur d'adresses d'instructions
Event DAC1	Évènement lié au registre 1 du comparateur d'adresses de données
Event DAC2	Évènement lié au registre 2 du comparateur d'adresses de données
Event PMW0	Évènement lié au registre 0 des compteurs de performance

La collection de la trace c'est-à-dire la récupération des messages de trace peut s'effectuer à l'aide de trois types de mémoires tampon :

- le « buffer Nexus » qui a une taille de 16 KB mais présente l'avantage d'être non-intrusif,

- le « buffer DDR » qui a une taille moins limitée mais opère de manière intrusive,

- la trace Aurora qui permet de sauvegarder jusqu'à 4 GB de données et est non-intrusive mais nécessite une sonde « Gigabit TAP» [FRE11-1].

Pour lire le contenu des mémoires tampon, l'environnement de développement de chez Freescale, CodeWarrrior [FRE10-1], peut être utilisé. Il existe cependant d'autres environnements de développement permettant de lire la trace.

c) Les registres de comparaison d'adresses d'instructions et de données.

Le processeur possède une option de comparaison d'adresses. Ceci permet de spécifier une adresse spécifique ou une plage d'adresses à comparer pour les instructions (IAC - « instruction address compare ») et les données (DAC - « data address compare »). Ainsi, un évènement de débogage tel que l'envoi d'un message peut être activé lorsque cette option de comparaison est activée.

La programmation de cette fonctionnalité se fait au travers des registres de contrôle de débogage « DBCR » (Debug Control Register). Les registres IAC et DAC contiennent respectivement l'adresse de l'instruction et l'adresse de la donnée à comparer. Il existe quatre registres IAC mais seulement deux sont disponibles pour cette version de processeur.

Dans le cas du registre IAC, lorsqu'une instruction est chargée dans le pipeline et que l'adresse de cette instruction est soit égale à une valeur spécifique soit incluse (ou non) dans un intervalle donné, un évènement de débogage est généré. Dans le cas des registres DAC, la comparaison s'effectue lorsqu'une donnée est stockée ou chargée en mémoire. Il est alors possible de générer un événement en effectuant des combinaisons de comparaisons entre deux registres.

Ainsi, au travers de la description des différentes fonctionnalités de débogage, il est possible de tirer bénéfice d'une telle architecture pour implémenter la méthode d'analyse de cache sur cible matérielle.

IV.2.2 Implémentation de la méthodologie

Le principe de la méthode est le suivant : lors de l'exécution d'une application, il faut sauvegarder d'une part la liste des instructions exécutées ou les adresses des accès en mémoire et

d'autres part les compteurs de performance. Comme présenté dans la partie IV.2.1, le processeur P4080 dispose d'un certain nombre de fonctionnalités de débogage qui peuvent fournir les paramètres d'entrée nécessaires pour l'analyse de cache. En combinant ces différentes fonctionnalités, deux options sont envisageables pour implémenter la méthodologie ; elles sont présentées dans la Figure IV-4.

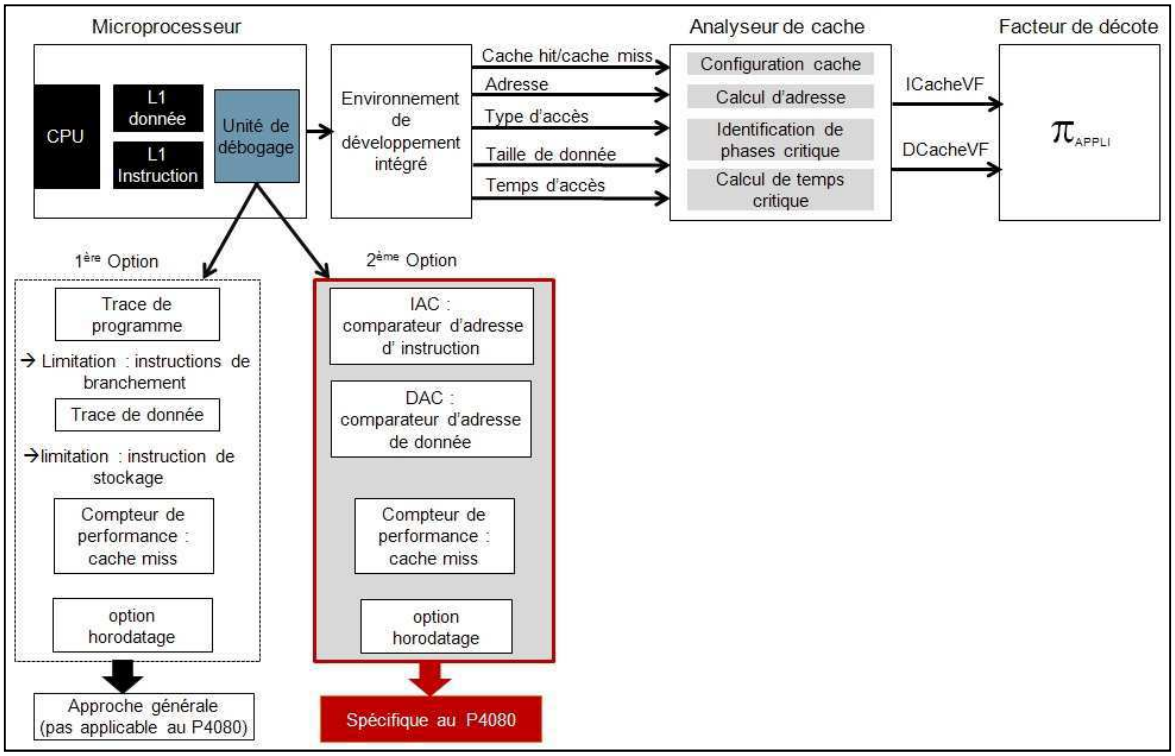


Figure IV-4 : Schéma de la méthode appliquée sur une cible matérielle

IV.2.2.1 Première option

La première solution consiste à récupérer d'une part la trace complète du programme afin d'analyser la sensibilité du cache d'instructions et d'autre part la trace de données pour obtenir les adresses de chargement et de stockage du cache de données.

Le P4080 offre la possibilité de tracer soit le programme soit les données grâce au module « Nexus Trace » comme indiqué dans la partie IV.2.1.b. Cependant, la version actuelle du cœur e500mc contient certaines limitations au niveau de ce module de trace. En effet, le cœur e500mc ne peut tracer d'une part que les instructions de branchement et d'autre part, concernant les données, les sauvegardes sont limitées au accès de stockage.

Il est alors difficile d'implémenter la première solution du fait des limitations spécifiques à ce cœur. Ainsi, une deuxième solution a été mise en œuvre pour pallier ce problème. Néanmoins, cette

solution semble envisageable pour une architecture ne limitant pas la trace aux instructions de branchement ou aux instructions de stockage.

IV.2.2.2 Deuxième option

Une deuxième solution a été mise en œuvre pour obtenir les adresses des données sollicitées par le CPU. L'environnement de développement et de débogage CodeWarrior a été utilisé afin d'automatiser la séquence de test. Cette solution est fondée sur l'utilisation des comparateurs d'adresses de données et d'instructions, l'unité de trace des points de surveillance « watchpoint trace » et enfin l'unité des compteurs de performance.

a) Configuration des comparateurs d'adresses

Comme présenté dans la partie IV.2.1.c, l'unité de comparateur d'adresses permet de générer un évènement de débogage lorsqu'une donnée spécifiée par une adresse ou une plage d'adresses est sollicitée. Cet évènement est ensuite tracé grâce à l'unité de trace des points de surveillance. Ainsi, une information temporelle liée à ces évènements peut être obtenue.

Pour le cache d'instructions, afin d'obtenir la trace contenant les temps d'accès de toutes les instructions qui ont été sollicitées par le CPU, les comparateurs d'adresses sont programmés en mode « comparaison exacte des adresses ». Ce mode de comparaison permet de générer un évènement de débogage lié au comparateur d'adresses IAC (« Event IAC »), lorsque l'adresse de l'instruction chargée est égale à la valeur spécifiée dans le registre IAC.

Le cœur e500mc possède deux registres de comparaison d'adresses d'instructions IAC1 et IAC2, ainsi il est possible de spécifier deux adresses différentes dans chaque registre IAC et d'obtenir la trace de deux instructions en une seule exécution.

En résumé, la procédure consiste à activer les modules de comparateurs d'adresses IAC1 et IAC2, et à spécifier pour chacun des registres l'adresse à surveiller. Le mode de comparaison exact doit aussi être choisi. Finalement, la trace des points de surveillance « watchpoint trace » est activée pour pouvoir générer un message à chaque événement « Event IAC1 » et « Event IAC1 » (Figure IV-5). La principale contrainte de cette technique est qu'il faut relancer l'application pour chacune des adresses à pister.

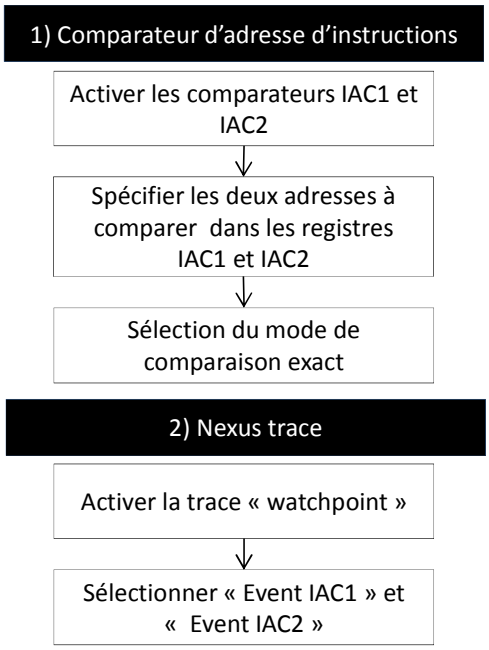


Figure IV-5 : Procédure pour obtenir la trace des instructions

Pour le cache de données, le même principe est appliqué. Les comparateurs d'adresses de données DAC sont activés en mode de comparaison exact pour pouvoir générer un évènement de débogage lié au comparateur d'adresses DAC (« Event DAC »). Cependant, les deux comparateurs sont programmés pour observer la même adresse de donnée mais le premier identifie les accès en lecture et le deuxième les accès en écriture.

b) Configuration des compteurs de performance

L'unité de suivi de performance est programmée pour surveiller les accès *cache miss* du cache de données et du cache d'instructions. Le principe de programmation de cette unité du processeur est illustré par la Figure IV-6. Pour obtenir tout d'abord les accès *cache miss* du cache de données, on initialise le compteur PMC0, puis on sélectionne l'évènement « com41 » qui correspond au rechargement du cache de données L1, comme indiqué dans le Tableau IV-1. Le registre PMLCb0 est programmé pour générer un évènement de débogage « Event PMW0 » à chaque incrémentation du compteur. Le même principe est appliqué pour obtenir les accès *cache miss* du cache d'instructions. Puis, la trace des points de surveillance « watchpoint trace » est activée pour pouvoir générer un message à chaque fois que cet événement apparaît.

```
#Cache de données : miss

#Initialisation du compteur à 0 :
PMC0  = 0x00000000

# Sélection de  l'événement  : « com41 »
 PMLCA0 = 0x00290000

# Sélection de la condition de déclenchement  : incrémentation de PMC0
PMLCB0 = 0x0000E000
```

```
#Cache d'instructions: miss

#Initialisation du compteur à 0 :
PMC0  = 0x00000000

# Sélection de  l'événement  : « com60 »
 PMLCA0 = 0x003C0000

# Sélection de la condition de déclenchement  : incrémentation de PMC0
PMLCB0 = 0x0000E000
```

Figure IV-6 : Programmation des compteurs de performance

c) Présentation de la méthodologie

La méthode mise en œuvre utilise l'unité des comparateurs d'adresses et nécessite de spécifier à chaque exécution l'adresse à surveiller. Ainsi, il est nécessaire d'obtenir au préalable toutes les adresses des données ou des instructions utilisées par le CPU. Pour cela, on exécute l'application à caractériser en activant la trace d'exécution d'un programme. Même si cette fonction ne trace que les instructions de branchement, elle fournit toutes les adresses utilisées par le CPU sans indiquer les temps d'accès. On obtient ainsi la plage d'adresses à observer durant l'exécution d'un programme.

Un script en langage TCL a été développé pour automatiser toute la méthodologie proposée. Il permet de lancer l'exécution d'une application en modifiant les valeurs des adresses à pister. La Figure IV-7 illustre l'algorithme utilisé pour automatiser la procédure d'obtention des informations nécessaires pour l'analyse du cache d'instructions. La première étape consiste à calculer les deux adresses à examiner pour pouvoir configurer les registres IAC1 et IAC2 avec ces adresses. En effet, on incrémente automatiquement les adresses à observer à partir de la borne inférieure de la plage d'adresses. Puis, on modifie le fichier de configuration P4080.xml. Ce fichier permet de configurer les options de débogage et de sélectionner les options de trace de points de surveillance. On lance ensuite le mode de débogage et l'exécution du code jusqu'à la fin de la routine d'initialisation du cache. Lorsque le cache a été initialisé, on exécute le code de lancement de trace. Il va permettre de configurer la collecte de traces qui s'exécute au démarrage du code applicatif. Puis on redémarre le programme afin d'observer les deux adresses suivantes.

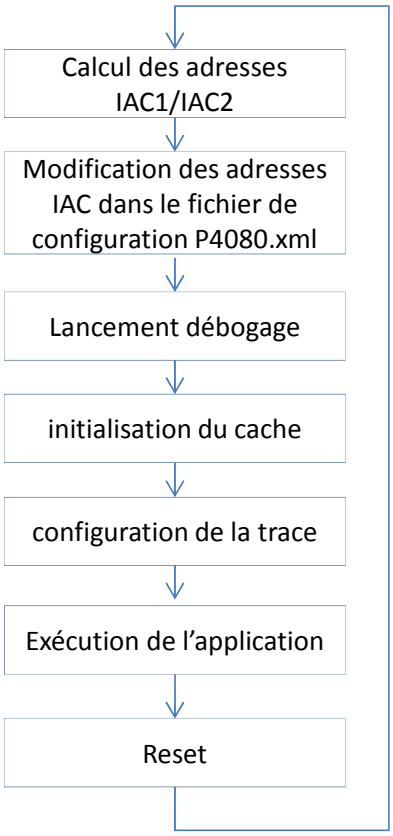


Figure IV-7 : Algorithme du script d'automatisation

La Figure IV-8 synthétise la procédure pour obtenir la sensibilité du cache d'instructions d'un cœur du P4080. La phase d'initialisation consiste à programmer toutes les fonctionnalités de débogage. Puis on exécute le code. Une fois tous les fichiers collectés pour toutes les adresses, on peut exécuter l'application en activant cette fois-ci les compteurs de performance. Ensuite, l'outil analyseur de cache peut analyser toutes les informations recueillies. Le logiciel a donc été adapté pour prendre en considération les fichiers issus du débogueur pour un processeur P4080. Cette mise à jour consiste à combiner tous les fichiers et à réévaluer les temps d'accès sur la même base de temps.

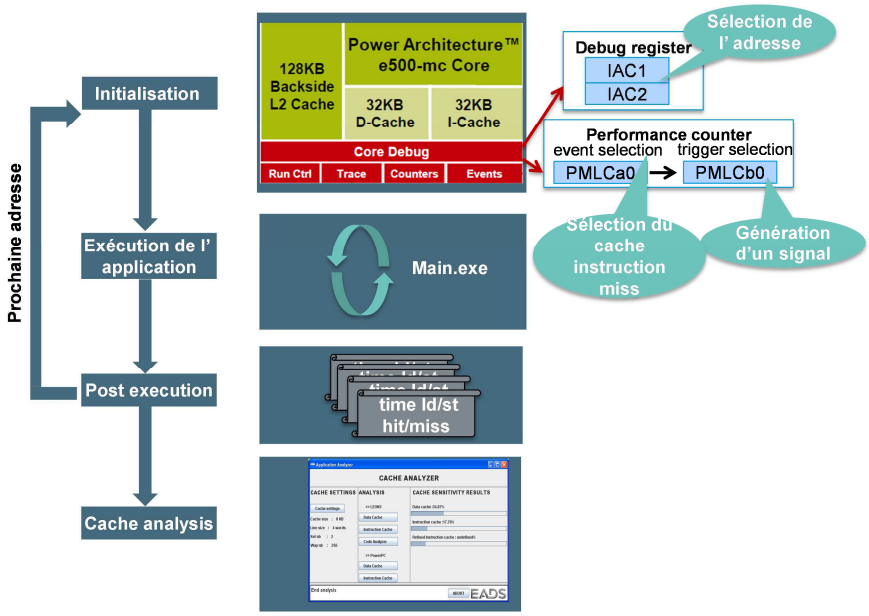


Figure IV-8 : Méthode d'analyse de cache pour un cœur du P4080

IV.3 Résultats obtenus et validation de l'approche

La méthode d'analyse de cache proposée a été appliquée à l'application FFT de MiBench [GUT01]. Cet algorithme « FFT » calcule une transformée de Fourrier rapide, et son inverse. L'application a été exécutée uniquement sur le cœur « core 0 » du P4080, les autres cœurs étant désactivés.

IV.3.1 Résultat

La sensibilité des mémoires caches a été évaluée avec l'analyseur de cache. Les données critiques de chacune des mémoires cache ont été identifiées et les temps critiques calculés.

Ensuite à partir de ces informations, les facteurs de décote ont été calculés pour chacune des mémoires cache, en tenant compte de la taille des données, du temps d'exécution total de l'application et de la taille totale du cache((Eq. II-5).

Les résultats sont présentés dans la Figure IV-9. Pour chaque mémoire cache, le pourcentage obtenu avec l'analyseur de cache est présenté dans la colonne « Sensibilité du cache obtenue avec l'analyseur de cache ». Par ailleurs, cette figure contient aussi le taux d'occupation des mémoires cache, obtenu avec le débogueur qui permet de visualiser le contenu des caches.

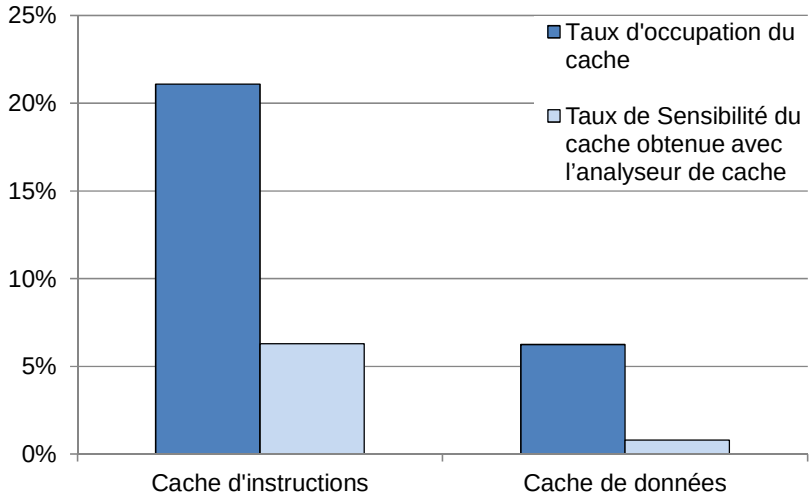


Figure IV-9 : Estimation du taux de sensibilité du cache de données et du cache d'instructions d'un cœur du P4080 pour l'application FFT

Le temps nécessaire pour obtenir les temps d'accès d'une adresse est estimé à environ 30 s. Ce temps est relativement long contenu du temps d'exécution de l'application qui est inférieur à la seconde. Ceci est dû principalement au temps de reset et de sauvegarde du ficher de résultat.

Pour le cache d'instructions, environ 13400 adresses ont été surveillées par les comparateurs d'adresses. Pour le cache de données, seules les adresses issues de l'étude du taux d'occupation ont été observées.

Il existe une variation au niveau de la précision des temps d'accès issus de l'unité Nexus Trace. Cependant, cette variation de temps est relativement faible, de l'ordre de moins d'une nanoseconde. Cela implique donc de calculer des marges d'erreur liées à cette variation de temps. Cependant, la documentation de Freescale ne permet pas de calculer une marge d'erreur ; il faudrait plus de données du fabricant pour évaluer cette différence.

Cette variation peut quelques fois impacter la corrélation entre le temps issu du compteur de performance indiquant les *cache miss* et les temps d'accès des adresses sollicitées. C'est pourquoi, compte tenu du faible nombre de miss, une procédure spécifique a permis de retrouver l'adresse de cache exacte provoquant le *cache miss* en stoppant le programme à chaque accès miss pour retrouver l'adresse de la ligne chargée en mémoire cache.

IV.3.2 Validation

La validation sous faisceau accélérateur n'était pas envisageable à cause du temps nécessaire pour le développement d'un banc de test, des disponibilités des installations et du coût des

campagnes. Les résultats ont été validés grâce une campagne d'injection de fautes par interface de débogage intégré. Il s'agit de provoquer le basculement d'un contenu mémoire avec l'interface de débogage pendant que le processeur exécute son application et d'observer le résultat de cette perturbation.

Cependant, comme présenté dans le chapitre I, cette méthode présente un inconvénient majeur qui est le temps relativement important nécessaire pour les injections. En effet, cette méthode nécessite l'arrêt de l'exécution du code source pour pouvoir corrompre une valeur et relancer l'exécution du programme. En conséquence, seuls les résultats de la mémoire cache d'instructions ont été validés. De plus, pour réduire le temps de la campagne, l'injection de fautes n'a été réalisée que dans les cellules occupées.

Un script en langage TCL a été développé pour injecter des fautes dans les mémoires cache en tenant comptes des limitations de l'environnement de débogage CodeWarrior. En effet, il n'y a pas de commande qui permette de modifier le contenu d'un mot du cache. Il faut donc réécrire toute une ligne de cache. Ainsi, pour modifier un bit d'un mot du cache, il faut tout d'abord lire toute la ligne contenant le mot, puis modifier la valeur du mot cible et ensuite réécrire toute la ligne.

Le deuxième inconvénient majeur est qu'il n'est pas possible d'arrêter l'exécution du code pour une durée inférieure à la seconde. La mise en place de points d'arrêts permet de pallier partiellement ce problème. Des points d'arrêt ont donc été placés dans une ligne de code générée aléatoirement, puis retiré une fois que la valeur a été corrompue. Cette méthode ne permet pas de garantir un taux de couverture satisfaisant car les valeurs ne sont pas corrompues aléatoirement dans le temps mais seulement lorsque le code s'arrête la première fois qu'il rencontre le point d'arrêt. De plus, il existe des cas où le programme ne s'arrête pas au point d'arrêt spécifié.

Ainsi, pour valider la méthode, deux campagnes de tests ont été réalisées :

- Durant la première campagne de test, l'injection de fautes a été réalisée au même moment dans le code, c'est-à-dire que le point d'arrêt a toujours été placé au même endroit. Ainsi, environ 447 fautes ont été injectées.

- Durant la deuxième campagne de test, le placement aléatoire de point d'arrêt a été mis en place. Il n'y a eu que 188 injections de fautes pour ce test, car de nombreuses fois, le programme ne s'est pas arrêté aux points d'arrêt spécifiés.

Il a été vérifié que toutes les adresses détectées comme sensibles par l'injection de fautes ont aussi été identifiées par la méthode Cache Analyzer. Les résultats du Tableau IV-3 montrent que

cette condition a été satisfaite. Cependant, on peut noter que les inconvénients liés à la méthode d'injections de fautes limitent fortement le nombre d'injections de fautes possibles.

Tableau IV-3 : Validation des données critiques du cache d'instructions

	Nombre d'injections de fautes provoquant une erreur d'application	Prédit par l'analyseur de cache
1ère campagne de test	158	158
2ème campagne de test	13	13

IV.3.3 Les limitations et perspectives

Cette méthode présente quelques inconvénients en termes de temps et de stockage d'informations, et un certain nombre de limitations et d'axes d'amélioration ont été identifiés.

L'inconvénient majeur de cette méthode est le temps nécessaire pour récupérer l'ensemble de la trace du programme, ce qui est dû à l'utilisation du comparateur d'adresses. La durée nécessaire pour obtenir cette trace peut être réduite en supprimant les fenêtres d'affichage de l'interface de débogage et en lançant le code en ligne de commande. Il est par ailleurs possible d'utiliser une sonde de type Aurora [FRE11-1] pour augmenter la vitesse de communication entre la plateforme P4080 et le PC de contrôle, afin d'augmenter la vitesse d'exécution des commandes de type redémarrage, arrêt, continuation … De plus, cette sonde permet aussi d'augmenter la capacité de stockage de la trace. La mémoire tampon utilisée est limitée à 16 KB alors que la sonde permet de stocker 4 GB.

Par ailleurs, l'architecture de débogage du P4080 dispose d'une unité EPU qui permettrait de combiner et de filtrer les évènements de débogage pour réduire la taille des données stockées ou envoyées au PC. Cette étude a permis cependant de montrer qu'en exploitant quelques-unes des fonctionnalités de débogage, il est possible d'obtenir des premiers résultats sur la sensibilité du cache. Toutefois, il existe d'autres solutions que celles proposées qui n'ont pas encore été étudiées.

IV.4 Conclusion

Dans ce chapitre, une adaptation de la méthode d'analyse de cache appliquée au simulateur a été proposée. En effet, certains processeurs ne disposent pas de simulateur cycle-accurate, c'est pourquoi il a été présenté une méthode qui peut s'appliquer directement sur la cible de processeur. La plupart des processeurs COTS récents possèdent des fonctionnalités de débogage complexes

visant à aider le développeur. En tirant profit de différentes possibilités qu'offrent les unités de débogage internes du processeur, une méthodologie d'analyse de cache a été mise en place.

Cette approche a été appliquée à un cœur du processeur P4080 et le fonctionnement des unités de débogage comme la trace et les compteurs de performance ont été présentés. Le principe est d'exécuter l'application sur la cible matérielle, et de configurer la trace matérielle et les compteurs de performance pour identifier les données critiques du cache. Pour obtenir la trace complète du programme, l'unité de comparateur d'adresses de données et d'instructions a été programmée. Cela consiste à spécifier une adresse à observer pendant l'exécution de l'application.

La méthode a été testée pour évaluer la sensibilité du cache de données et du cache d'instructions pour une application FFT. Ensuite une première étape de validation grâce une campagne d'injection de fautes par interface de débogage intégré, a été effectuée pour la mémoire cache d'instructions. Il a été vérifié que toutes les adresses détectées comme sensibles par l'injection de fautes ont aussi été identifiées par la méthode Cache Analyzer.

Cependant, l'inconvénient majeur de cette méthode est le temps relativement long, car il faut exécuter le code pour toutes les adresses à surveiller. Un certain nombre d'axes d'amélioration ont été identifiés.

Conclusion générale et perspectives

Les microprocesseurs COTS sont de plus en plus utilisés dans les applications du domaine de l'avionique et du spatial. Ces composants, évoluant dans des milieux difficiles, sont soumis à des contraintes environnementales sévères. En outre, la complexification des microprocesseurs n'a cessé d'augmenter au cours de ces dernières années, ce qui rend difficile l'élaboration de méthodologies pour prédire le taux d'erreurs dû aux rayons cosmiques. En effet, la sensibilité des microprocesseurs dépend fortement de l'application exécutée ; elle est aussi appelée sensibilité dynamique.

À travers l'étude de l'évolution des microprocesseurs et de leurs architectures, nous avons vu que la puissance de calcul des microprocesseurs n'a cessé de croître, en partie grâce aux mémoires caches. En effet, ces mémoires sont devenues des éléments indispensables des processeurs actuels et sont de grands contributeurs des taux d'erreurs des microprocesseurs. La synthèse de l'état de l'art a montré que les moyens de test classiques, comme les accélérateur de particules, ne sont pas la solution la mieux adaptée pour prédire le taux d'erreurs, car cela implique que le système développé soit en phase finale de son cycle de développement. Les autres méthodes reposent sur l'injection de fautes et requièrent une durée de test assez longue.

Pour répondre aux besoins industriels de disposer d'un outil, utilisable par les ingénieurs, et capable de prédire suffisamment tôt et rapidement la sensibilité dynamique de leurs applications pendant la phase de développement, une nouvelle méthodologie fondée sur l'analyse des mémoires cache à l'aide de simulateurs standard de type « cycle –accurate » a été proposée. Cette méthode consiste à obtenir dans une première phase la sensibilité intrinsèque de la technologie du composant et, dans une seconde phase, à analyser l'application et l'architecture du composant pour obtenir un facteur de décote lié à l'architecture et un autre lié à l'application exécutée.

Le facteur de décote lié à l'architecture du cache prend en compte les politiques d'écriture du cache et les mécanismes de protection, et ainsi permet de déterminer l'effet produit par un SEU. Néanmoins, il existe des configurations où les mécanismes de protection ne sont plus suffisants pour avoir un taux d'erreurs satisfaisant les exigences de fiabilité des applications critiques. En effet, l'occurrence d'événements multiples augmente avec la diminution du nœud technologique. Ainsi, un deuxième facteur de décote lié au comportement dynamique des mémoires cache a été étudié. Il s'obtient en exécutant une application sur un simulateur de processeur et en observant

les signaux propres aux simulateurs, comme les signaux de performance et d'accès cache. Ces signaux, indiquant la présence ou non d'une donnée en cache, nous servent à identifier les données critiques du cache. En calculant ensuite les temps d'exposition de chaque donnée critique, on obtient un facteur de décote lié à l'application. Une approche complémentaire a été développée pour améliorer la précision d'identification de bits critiques dans le cache d'instructions. Elle est basée sur une analyse statique du jeu d'instructions pour identifier les bits n'ayant pas d'impact sur l'exécution de l'instruction.

Par ailleurs, la méthode proposée a été validée par les deux moyens de test les plus pertinents issus de l'analyse de l'état de l'art : l'injection de fautes par émulation et les tests en accélérateur. La validation a donc été réalisée sur un prototype basé sur le processeur LEON3 pour deux applications. Les résultats issus de la technique d'injection de fautes ont montré des évaluations cohérentes avec celles de l'analyseur de cache. La deuxième validation expérimentale, par test en accélérateur, a nécessité le développement d'un banc de test spécifique. Un système de détection de fautes fondé sur le mécanisme de parité a été développé pour les mémoires cache et un banc de test spécifique pour récupérer les résultats en temps réel a été mis en place. Les résultats des tests en accélérateur ont permis de valider les données critiques identifiées par l'analyseur de cache. Ces expériences ont aussi montré une surestimation de la méthodologie proposée car elle est fondée sur le comportement des mémoires cache et non sur l'impact de la faute sur l'application.

L'avantage de la méthode d'analyse de cache est qu'on peut obtenir un facteur de décote pour les mémoires cache pour un ensemble d'applications dans un temps relativement court, qui ne dépend que de la vitesse du simulateur. De plus, cette méthode peut être implémentée avec n'importe quel simulateur cycle-accurate donnant des informations sur la performance du cache. Elle ne repose pas sur l'injection aléatoire de fautes et est de ce fait plus couvrante.

Pour compléter notre étude, nous avons étendu la méthode d'analyse des données critiques dans les mémoires cache à des cibles matérielles de processeur, dans le cas où le simulateur cycle-accurate n'est pas disponible. Nous avons donc exploité les fonctionnalités internes de débogage des processeurs modernes, notamment les registres de performance. La méthode d'analyse des temps critiques a été appliquée à un processeur d'intérêt pour les applications avioniques : le QorIQ P4080. L'approche a été mise en œuvre sur un cœur du processeur en utilisant les unités de débogage interne comme les compteurs de performances et les comparateurs d'adresses de données et d'instructions. Les premiers résultats de sensibilité du cache de données et du cache

d'instructions de niveau L1 semblent prometteurs. Puis, à partir d'une technique d'injection de fautes par interface de débogage, les résultats concernant le cache d'instructions ont pu être en partie validés. Un certain nombre d'axes d'amélioration ont été identifiés pour pallier l'inconvénient majeur de cette méthode qui est le temps relativement long nécessaire pour obtenir toutes les informations d'accès en cache. L'avantage principal de ce portage est qu'il n'est plus nécessaire de disposer d'un simulateur cycle accurate du processeur car l'analyse s'effectue directement sur la cible matérielle quand celle-ci est disponible. Cependant, soulignons ici que la validation doit être poursuivie en confrontant les prédictions avec des résultats de tests plus nombreux. La méthode d'analyse de cache sur cible matérielle n'est pas figée. Les unités de débogage des processeurs étant assez complexes, il est possible de mettre en œuvre d'autres solutions.

Des perspectives ont été identifiées à l'issue de ces travaux selon trois axes :

- Il serait d'intérêt d'évaluer la sensibilité des mémoires cache du P4080 par test laser en injectant des fautes par face arrière.

- Il est aussi envisageable d'appliquer la méthode aux composants multi-coeurs qui sont les prochaines cibles d'intérêt pour les équipements avioniques. Les mémoires cache de niveau L1 n'étant pas partagées par les autres cœurs, il est possible d'appliquer la méthode pour chaque cœur. Les mémoires cache de niveau partagé requiert de prendre en compte d'autres signaux indiquant l'état de la ligne.

- Enfin, il serait intéressant de développer un outil utilisable par les ingénieurs pour fournir une aide lors de la phase de développement des futures applications, sur ces processeurs multi-coeurs. Un tel outil, utilisable facilement chez les équipementiers, peut être fondé sur la méthodologie développée dans cette thèse, car la méthode proposée ne nécessite pas d'outils en dehors de ceux classiquement utilisés lors d'un développement d'équipement.

Bibliographie

[ACT08] Cortex-M1 Enabled ProASIC3 Development Kit User's Guide, 2008.

[ANT03] L. Antoni, R. Leveugle, B. Fehér, "Using Run-Time Reconfiguration for Fault Injection Applications", IEEE Trans. on Instrumentation and Measurement, vol 52, n° 5, pp. 1468-1473, Oct. 2003.

[ARP4754] ARP 4754: "Certification Considerations for Highly Integrated or Complex Aircraft Systems", 1996.

[ARP4761] ARP 4761: "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment", 1996.

[ASE98] V. Asenek, C. Underwood, S. Rezgui, M. Oldfield, Ph. Cheynet, R. Ecoffet, "SEU induced errors observed in microprocessor systems,", IEEE Trans. on Nuclear Science, vol.45, no.6, pp.2876,2883, Dec 1998.

[BEZ96] F. Bezerra, R. Velazco, A. Assoum, D. Benezech, "SEU and latch-up results on transputers, "IEEE Trans. on Nuclear Science", vol.43, no.3, pp.893,898, Jun 1996.

[BIS05] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing Architectural Vulnerability Factors for Address-Based Structures," Proc. Of the 32nd Annual Intl. Symp. on Computer Architecture (ISCA'05), pp. 532-543, Jun 2005.

[BOH04] P. Bohrer, M. Elnozahy, A. Gheith, C. Lefurgy, T. Nakra, J. Peterson, R. Rajamony, R. Rockhold, H. Shafi, R. Simpson, E. Speight, K. Sudeep, E. V. Hensbergen, , and L. Zhang. Mambo - A Full System Simulator for the PowerPC Architecture. ACM SIGMETRICS Performance Evaluation Review, 31(4), March 2004.

[BOU95] J. C. Boudenot, "L'Environnement Spatial", Collection "Que sais-je?," n° 3032, Ed. Presses Universitaires de France, 1995.

[BRIE71] K. O'Brien, "The Natural Radiation Environment", Report N°720805-P1, United States Department of Energy, p.15, 1971.

[BRIE78] K. O'Brien, Report N°EML-338, United States Department of Energy, 1978.

[BRU96] G. Bruguier and J-M. Palau, "Single Particle-Induced Latchup", IEEE Transactions on Nuclear Science, vol. 43, n°. 2, April, 1996.

[BUR97] D. A. Burger, T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report #1342, University of Wisconsin-Madison, Computer Sciences Department, June 1997.

[CIV01] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Exploiting FPGA for accelerating fault injection experiments", Proc. 7th Int. On Line Testing Workshop (IOLTW'01), Taormina, Italie, pp. 9-13, Juillet 2001.

[CUS85] J. Cusick, R. Koga, W. A. Kolasinski, C. King, "SEU Vulnerability of the Zilog Z80 and NSC-800 Microprocessors", "IEEE Trans. on Nuclear Science", vol. NS-32, no. 6, pp.4206-4211, Dec. 1985.

[DAV09] J.-M Daveau, A. Blampey, G. Gasiot, J. Bulone, P. Roche, "An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip," Reliability Physics Symposium, 2009 IEEE International, vol., no., pp.212-220, 26-30 April 2009.

[DUR09] G. Durrieu, H. Waeselynck, and V. Wiels, "LETO - A Lustre-Based Test Oracle for Airbus Critical Systems". In D. Cofer and A .Fantechi, editors, Formal Methods for Industrial Critical Systems, volume 5596 of Lecture Notes in Computer Science, pages 7–22. Springer Berlin / Heidelberg, 2009.

[ELD88] J. H. Elder, J. Osborn, W.A Kolasinski, R. Koga, "A method for characterizing a microprocessor's vulnerability to SEU," IEEE Transactions on Nuclear Science, vol.35, no.6, pp.1678-1681, Dec 1988.

[FER12] F. Ferlini, F. A. Da Silva, E.A Bezerra, D. V. Lettnin, "Non-intrusive fault tolerance in soft processors through circuit duplication," Test Workshop (LATW), 2012 13th Latin American , vol., no., pp.1,6, 10-13 April 2012.

[FID06] A.V. Fidalgo, G.R. Alves, and J.M. Ferreira, "Real Time Fault Injection Using a Modified Debugging Infrastructure," Proc. 12th IEEE Int'l Online Testing Symp., July 2006.

[FIP01] FIPS-197: Advanced Encryption Standard (AES), Federal Information Processing Standards Publications, Nov. 2001.

[FRE06] Freescale MPC8349E PowerQUICC™ II Pro Integrated Host Processor Family Reference Manual, Aug 2006.

[FRE06-1] Freescale Semiconductor "MPC8349E-mITX-GP Reference Design Platform User's Guide", Rev. 0, 10/2006.

[FRE07] Freescale Semiconductor, e300 Power Architecture Core Family Reference Manual, 2007.

[FRE10] Freescale, "Advanced QorIQ Debug and Performance Monitoring Reference Manual", rev. c ed., 2010.

[FRE12] Freescale, "P4080 QorIQ Integrated Multicore Communication Processor Family Reference Manual", rev. 1 ed., 2010.

[FRE11] Freescale, "e500mc Core Reference Manual", rev. 0 ed., 2011.

[FRE08] Freescale, "PowerQUICC III Performance Monitors" rev.1 08/2008

[FRE11-1] Freescale, "Gigabit TAP for the Suite of CodeWarrior Development Tools" rev.0 2011.

[GEN07] P. Genua "Error Correction and Error Handling on PowerQUICCTM III Processors", Application Note AN3532, Freescale Semiconductor, Nov. 2007.

[GUE10] S.M. Guertin, F. Irom, "Recent Results for PowerPC Processor and Bridge Chip Testing," Radiation Effects Data Workshop (REDW), 2010 IEEE , vol., no., pp.8,8, 20-23 July 2010.

[GUI12] N. Guibbaud, F. Miller, F. Molière and A. Bougerol, "Efficiency evaluation of the Soft-Errors Analysis techniques in complex ICs", RADECS 2012.

[GUT01] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on, vol., no., pp.3,14, 2 Dec. 2001.

[HAM50] R. Hamming. "Error detecting and error correcting codes ». Bell System Technical Journal, 29(2), April 1950.

[HAT71] D. J. Hatfield, J. Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192. 1971.

[HOU00] D. Houzet "Microprocesseurs : approche générale," Techniques de l'ingénieur, Vol. 2, p. 1-17, février 2000.

[HUB01] G. Hubert, "Élaboration d'une méthode de prédiction du taux d'aléas logiques dans les SRAMs induits par les neutrons atmosphériques" Thèse de Doctorat de l'Université Montpellier II, 2001.

[IEC12] International Electrotechnical Commission (IEC) TC107, TS62396-1, "Process Management for Avionics Atmospheric Radiation Effects, Accommodation of atmospheric radiation effects via single event effects within avionics electronic equipment", Edition 2012.

[IEE01] IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Standard 1141.1-2001, 2001.

[INT13] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide", Part 2. June 2013.

[IRO05] F. Irom, F. H. Farmanesh, "Single-Event Upset in Highly Scaled Commercial Silicon-on-Insulator PowerPC Microprocessor," IEEE Trans. on Nuclear Science, vol.52, no.5, pp. 1524-1529, Oct. 2005.

[IRO08] F. Irom, "Guideline for Ground Radiation Testing of Microprocessors in the Space Radiation Environment" Jet Propulsion Laboratory, Tech. Rep. 13, 2008.

[JED06] JEDEC standard, "Measurement and reporting of alpha particle and terrestrial cosmic ray-induced Soft Errors in semiconductor devices," JESD89A, Oct. 2006.

[KAR93] S. Karoui, "étude du comportement de circuits complexes en environnement radiatif spatial" , Thèse de doctorat, INPG, Dec. 1993.

[KOE10] F. Koebel, J.-F. Coldefy, "SCOC3: a space computer on a chip," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010, vol., no., pp.1345, 1348, 8-12 March 2010.

[KOO02] P. Koopman, "32-Bit cyclic redundancy codes for internet applications," in Proc. Int. Conf. Dependable Systems Networks, Jul. 2002, pp. 459–468.

[LAU11] Lauterbach, "Intelligent Trace Analyses for Cortex-M3/M4", 2011.

[LEV09] R. Leveugle, A. Calvez, P. Maistri, P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence" Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09. , vol., no., pp.502-506, 20-24 April 2009.

[LOP07] C. Lopez-Ongil, M. Garcia-Valderas "Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation", IEEE Trans. on Nuclear Science, vol.54, no.1, pp. 252-261, Feb. 2007.

[LUU09] A. Luu, Méthodologie de prédiction des effets destructifs dus à l'environnement radiatif naturel sur les MOSFETs et les IGBTs de puissance, Thèse de doctorat de l'université de Toulouse, EADS IW, Novembre 2009.

[MIL06] F. Miller, "Étude Expérimentale et Théorique des Effets D'un Faisceau Laser Pulse sur les Composants Électroniques et Comparaison avec les Évènements Singuliers Induits par L'Environnement Radiatif Naturel", Thèse de Doctorat de l'Université Montpellier II, 2006.

[MUK03] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," Proc. of the Intl. Symp. on Micro-architecture (MICRO-36), pp. 29-40, 2003.

[NOR93] E. Normand, A. Taber, "Single Event Upset in avionics," IEEE Trans. on Nuclear Science, vol. 40, no.2, pp. 120-126, Apr. 1993.

[NOR10] E. Normand and L. Dominik, "Cross comparison guide for results of neutron SEE testing of microelectronics applicable to avionics," in Proc. IEEE Radiat. Effects Data Workshop, 2010, pp.

8–8.

[PEA98] R. L. Pease, A. H. Johnston, J. L Azarewicz, "Radiation testing of semiconductor devices for space electronics," Proceedings of the IEEE , vol.76, no.11, pp.1510,1526, Nov 1988.

[PEN06] J. Peng, J. Ma, B. Hong, and C. Yuan, "Validation of Fault Tolerance Mechanisms of an Onboard System," Proc. First Int'l Symp. Systems and Control in Aerospace and Astronautics, pp. 1230-1234, Jan. 2006.

[PHA11] C. Pham, H. Malcom, R. Maurer, D. Roth, K. Strohbehn, "LEON3FT Proton SEE Test Results for the Solar Probe Plus Program," Radiation Effects Data Workshop (REDW), 2011 IEEE , vol., no., pp.1,4, 25-29 July 2011.

[POR11] M. Portela-García, C. López-Ongil, M. G. Valderas, and L. Entrena,"Fault injection in modern microprocessors using on-chip debugging infrastructures," IEEE Trans. Depend. Secure Comput., vol. 8, no. 2, pp. 308–314, Sep. 2011.

[RAD11] Short course session 8 "Radiation analysis to be performed at Equipment level", Radiation and Its Effects on Components and Systems (RADECS), 12th European Conference on, Sevilla, 2011.

[RAD10] J. M. Hess, " Radiation en avionique Expérience d'un équipementier", RADSOL 2010, 3 – 4, juin 2010.

[REB99] M. Rebaudengo and M. S. Reorda, "Evaluating the Fault Tolerance Capabilities of Embedded Systems via BDM," VLSI Test Symposium, 1999. Proceedings. 17th IEEE., pp. 452-457, Apr. 1999.

[REZ01] S. Rezgui "Prédiction du taux d'erreur d'architectures digitales : une méthode et des résultats expérimentaux" Thèse de Doctorat Institut National Polytechnique de Grenoble - INPG, 2001.

[REZ08] S. Rezgui, J. J. Wang, S. Yinming, B. Cronquist, and J. McCollum, "New reprogrammable and non-volatile radiation tolerant FPGA: RTA3P," in Proc. Aerospace Conf., Mar. 2008, pp. 1–11.

[SAN08] Sanda, P.N.; Kellington, J.W.; Kudva, P.; Kalla, R.; McBeth, R. B.; Ackaret, J.; Lockwood, R.; Schumann, J.; Jones, C. R., "Soft-error resilience of the IBM POWER6 processor," IBM Journal of Research and Development , vol.52, no.3, pp.275,284, May 2008.

[SEI06] N. Seifert , P. Slankard , M. Kirsch , B. Narasimham , V. Zia , C. Brookreson , A. Vo , S. Mitra , B. Gill and J. Maiz "Radiation-induced soft error rates of advanced CMOS bulk devices", Proc. IRPS, pp.217 -225, 2006.

[SIM02] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, F. Larsson, A. Moestedt; B.Werner..,"Simics: A full system simulation platform," Computer , vol.35, no.2, pp.50,58, Feb 2002.

[SCH08] J. R. Schwank, M. R. Shaneyfelt, P.E. Dodd, "Radiation hardness assurance testing of microelectronic devices and integrated circuits: radiation environments, physical mechanisms and foundations for hardness assurance," Sandia National Laboratories Document SAND-2008-6851P1.

[SOD04] SODERN, Neutron Generator GENIE 16GT, ref. DAN/FP/16- 12/04, 2004. Disponible en ligne : http://www.sodern.com/sites/docs_wsw/RUB_79/Genie16GT.pdf

[SOM97] A. K. Somani, K. S. Trivedi, "A Cache Error Propagation Model", Proc. of the Pacific Rim International Symposium Fault-Tolerant Systems, pp. 15-21, 1997.

[SPA92] "The SPARC Architecture Manual Version 8", SPARC International, Prentice Hall, 1992.

[SRI06] V. Sridharan, H. Asadi, M.B. Tahoori, and D. Kaeli, "Reducing Data Cache Susceptibility to Soft Errors," IEEE Trans. Dependable and Secure Computing, vol. 3, no. 4, pp. 353-364, Oct.-Dec. 2006.

[THO02] S. Thompson et al., "A 90-nm logic technology featuring 50-nm strained silicon channel transistors, 7 layers of Cu interconnects, low k ILD, and 1 m 2 SRAM cell," in IEDM Tech. Dig., pp. 61–64,2002.

[TSI13] http://www.gaisler.com/index.php/products/simulators/tsim.

[VAL07] M.G. Valderas, P. Peronnard, C.L Ongil, R. Ecoffet, F. Bezerra, R. Velazco, "Two Complementary Approaches for Studying the Effects of SEUs on Digital Processors," Nuclear Science, IEEE Transactions on , vol.54, no.4, pp.924,928, Aug. 2007.

[VEL92] R. Velazco, S. Karoui, T. Chapuis, D. Benezech, L.H. Rosier, "Heavy ion test results for the 68020 microprocessor and the 68882 coprocessor," Nuclear Science, IEEE Transactions on , vol.39, no.3, pp.436,440, Jun 1992.

[VEL00] R. Velazco, S. Rezgui, "Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection," Nuclear Science, IEEE Transactions on , vol.47, no.6, pp.2405,2411, Dec 2000.

[WAN09] Shuai Wang; Jie Hu; Ziavras, S.G., "On the Characterization and Optimization of On-Chip Cache Reliability against Soft Errors," Computers, IEEE Transactions on, vol.58, no.9, pp.1171-1184, Sept. 2009.

[WEU11] C. Weulersse, "Développement et validation d'outils Monte-Carlo pour la prédiction des basculements logiques induits par les radiations dans les mémoires SRAM très largement submicroniques ", Thèse de Doctorat de l'Université Montpellier II, 2011.

[WRO01] F. Wrobel, J.M. Palau, M.C. Calvet, O. Bersillon, H. Duarte, "Simulation of Nucleon-induced Nuclear Reactions in a Simplified SRAM Structure: Scaling Effects on SEU and MBU cross sections" , IEEE Transactions on Nuclear Science 48, pp. 1946-1952, December 2001.

[ZIE96] J. F. Ziegler, "Terrestrial cosmic rays," IBM Journal of Research and Development, vol. 40, n° 1, pp. 19-40, 1996.

Liens Internet

[WWW01] http://en.wikipedia.org/wiki/Microprocessor

[WWW02] http://www.freescale.com/

[WWW03] http://www.arm.com/

[WWW04] http://sohowww.nascom.nasa.gov/home.html

Liste des acronymes

BRAM	Block RAM
CMOS	Complementary MOS
COTS	Component Off The Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DAC	Data Address Compare
DAL	Design Assurance Level
DDR	Double Data Rate
DUT	Device Under Test
EADS	European Aeronautic Defense and Space Company
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
IAC	Instruction Address Compare
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
JEDEC	Joint Electron Devices Engineering Council
LET	Linear Energy Transfer
LRU	Least Recently Used
MBU	Multiple Bit Upset
MCU	Multi Cell Upset
MMU	Memory Management Unit
SEE	Single Event Effect
SEFI	Single Event Functional Interrupt
SEL	Single Event Latchup
SER	Soft Error Rate
SET	Single Event Transient
SEU	Single Event Upset
SRAM	Static Random-Access Memory
SRAM	Static Random Access Memory
TCL	Tool Command Language
UCL	Université Catholique de Louvain
VHDL VHSIC	Hardware Description Language

Publications de l'auteur

Revues internationales :

S. Houssany, N. Guibbaud, A. Bougerol, R. Leveugle, F. Miller, N. Buard, "Microprocessor soft error rate prediction based on cache memory analysis," IEEE Transactions on Nuclear Science (TNS), vol.59, no.4, pp.980-987, August. 2012.

S. Houssany, N. Guibbaud, A. Bougerol, R. Leveugle, T. Santini, F. Miller, "Experimental Assessment of Cache Memory Soft Error Rate Prediction Technique," IEEE Transactions on Nuclear Science (TNS), vol.60, no.4, pp.2734-2741, August. 2013.

Conférences internationales :

S. Houssany, N. Guibbaud, A. Bougerol, R. Leveugle, F. Miller, N. Buard, "Microprocessor soft error rate prediction based on cache memory analysis," 12th European Conference on Radiation Effects on Components and Systems (RADECS), Sevilla, Spain, September 19-23 2011.

S. Houssany, N. Guibbaud, A. Bougerol, R. Leveugle, T. Santini, F. Miller, "Experimental Assessment of Cache Memory Soft Error Rate Prediction Technique," 13th European Conference on Radiation Effects on Components and Systems (RADECS), Biarritz, France, September 24-28 2012.

Brevets :

Procédé de caractérisation de sensibilité d'un composant électronique pour procédé de conception d'équipement électronique
Inventeurs : A. Bougerol, S. Houssany
Publication n° : WO/2013/014239, FR2011/056867.
Date de publication : 01 Février 2013.

TITRE : Méthodologie d'évaluation de la sensibilité des microprocesseurs vis-à-vis des rayonnements cosmiques

RESUME Les microprocesseurs dits pris sur étagère (Commercial « Off-The-Shelf » ou COTS) sont de plus en plus utilisés dans les systèmes avioniques et spatiaux. La prédiction du taux d'erreurs réel dus aux rayons cosmiques d'un microprocesseur exécutant un logiciel d'application, est devenue une préoccupation majeure car les méthodes actuelles conduisent à une surestimation du taux réel. De plus, les mémoires cache sont un contributeur majeur du taux d'erreur du fait de l'augmentation significative de leur taille. Cette thèse porte sur la prédiction plus précise de la sensibilité des mémoires cache en tenant compte de l'application opérée sur le microprocesseur. Une nouvelle méthode a été développée ; elle est fondée sur la surveillance des accès en mémoire cache et nécessite un simulateur de processeur. Cette approche est mise en œuvre sur un processeur LEON3 pour plusieurs applications. Les résultats obtenus ont été validés par une campagne d'injection de fautes par émulation, puis par des tests en accélérateurs de particules. Le moyen de test expérimental utilisé pour les tests en accélérateur est aussi présenté, il comprend un système de détection de fautes au sein des mémoires caches. Enfin, une deuxième méthode a été implémentée pour une cible matérielle de processeur multicoeur d'intérêt pour des applications avioniques. Cette méthode est fondée sur l'utilisation des fonctionnalités complexes intégrées dans les processeurs modernes tels que les compteurs de performances. L'avantage de ce portage est qu'il n'est plus nécessaire de disposer d'un simulateur, l'analyse s'effectuant sur la cible réelle.

Mots clés : Microprocesseur, mémoire cache, simulateur, taux d'erreurs

TITLE: Methodology to evaluate microprocessor sensitivity towards cosmic radiations

ABSTRACT Component Off-the-Shelf (COTS) microprocessors are widely used in space and aeronautic systems. Prediction of the real failure rate due to cosmic radiations of microprocessor running a specific application has become a major concern since current methodologies lead to huge overestimations of the soft error rate (SER). Moreover, cache memories are a major contributor to SER due to a significant increase of cache memories size. This work focuses on the predictions of more accurate cache memory sensitivity by taking into account the running application. A new methodology has been developed; it is based on the monitoring of cache accesses, and requires a microprocessor simulator. This approach is applied to the LEON3 soft-core with several benchmarks. Results obtained have been validated thanks to an RT-level fault injection technique and accelerated test experiments. The experimental setup used during the accelerator tests are also presented, it includes a fault detection system inside cache memories. Finally, a second methodology has been implemented for a hardware-based multicore processor, targeted to aeronautic applications. This methodology is based on the use of complex integrated features of modern processor such as performance counters. The advantage of this is implementation is that simulator is no longer required, as the analysis can be done directly on the hardware target.

Keywords: Microprocessor, cache memory, simulator, failure rate

INTITULE ET ADRESSE DU LABORATOIRE

Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France.

n°ISBN 978-2-11-129180-5