# Chapter 1

# État de l'art

## 1.1 Introduction

Comme ceci a été indiqué précédemment, la motivation principale de notre travail est la mise au point d'outils permettant d'un côté de réduire le temps de simulation des performances des différentes alternatives et de l'autre côté de simplifier la phase de conception des MPSoC. En effet, l'une des difficultés rencontrées par les concepteurs de ces systèmes est de trouver la meilleure configuration des différentes unités contenues dans un MPSoC. Cette tâche de conception, appelée exploration, permet d'optimiser les performances et respecter les contraintes imposées par l'application et l'utilisateur. Cet objectif doit être atteint tout en garantissant un temps de conception et de test relativement court pour évaluer un nombre important d'alternatives. Les travaux entrepris dans le cadre de cette thèse visent plusieurs points:

- La co-simulation logicielle/matérielle des systèmes MPSoC à des niveaux d'abstraction élevés.
- L'estimation de performance du système offrant un premier critère de comparaison des solutions architecturales
- L'estimation de la consommation d'énergie
- L'utilisation d'une méthodologie de conception pour réduire la complexité et les temps de développement.

Cet ensemble d'objectifs, nous amène à diviser ce chapitre, sur l'état de l'art, en 4 sections. Une section est dévolue à chaque objectif. Ainsi, la section 2 introduit aux notions de co-simulation et niveaux d'abstraction. Un tour d'horizon des techniques de réduction du temps d'évaluation des performances pour les MPSoC sera présenté dans la section 3. Ensuite, nous survolons dans la section 4 les différentes méthodes d'estimation de la consommation dans les systèmes sur puce. La section 5 synthétise quelques environnements de conception de MPSoC basés sur une approche dirigée par les modèles.

## 1.2 Le codesign et la co-simulation des systèmes sur puce

La complexité de conception des SoC augmente de plus en plus. Les raisons de cette augmentation sont:

- L'augmentation du niveau d'intégration des transistors sur une même puce.
- L'hétérogénéité des architectures proposées qui devient nécessaire pour respecter les contraintes imposées par l'application.
- La taille croissante des nouvelles applications que ces systèmes tentent de supporter.
- Un nombre important de contraintes à respecter telles que le "time-to-market", le coût final du système et la fiabilité.

La conception conjointe logicielle/matérielle, ou le *Codesign*, tente d'apporter une solution efficace à ces problèmes. En 1983, Gajski et Kuhn [22] ont présenté le modèle Y (Y chart) pour décrire les étapes de conception pour les circuits VLSI (*Very Large Scale Integration*). Dans ce modèle, le système est décrit selon trois vues: structurelle (la description électronique), comportementale (la description fonctionnelle) et géométrique (le résultat physique). Aujourd'hui encore, plusieurs travaux s'inspirent de ce modèle Y pour organiser les phases de *codesign* dans un SoC. La figure 1.1 détaille cette organisation. Dans les premières phases de conception, les deux parties matérielle et logicielle sont développées parallèlement et séparément. De ce fait, la conception de la partie logicielle peut commencer avant que la conception de l'architecture ne soit complètement terminée ce qui réduit le temps de conception. Pour une plus grande réduction du temps de simulation, les développeurs ont recourt à la réutilisation de composants déjà existants. Ces derniers, connus sous le nom de blocs IP pour "Intellectual Property", peuvent provenir de développements internes à un groupe de concepteurs ou peuvent être acquis auprès d'un fournisseur tiers et intégrés au flot de conception.

Dès que les premières étapes de spécification sont franchises pour les deux parties, une phase d'association qui consiste à placer l'application sur les composants de l'architecture peut être réalisée. Les tâches de l'application seront placées sur les différents types de processeurs ainsi que les données et les instructions seront placées sur les différentes mémoires disponibles. Cette phase d'association est à la fois topologique (placement) et temporelle (ordonnancement). Une fois l'association est terminée, la description du système est complète. Il est alors possible de vérifier les choix du concepteur sur les performances du système. En d'autres termes, nous vérifions l'adéquation de la capacité de calcul des processeurs avec les demandes des tâches ainsi que la localité des données par rapport aux processeurs qui les manipulent. Plusieurs critères doivent être observés. Le premier est le bon fonctionnement du système, c'est-à-dire s'assurer que le SoC répond toujours correctement aux entrées. Le deuxième critère important est le temps d'exécution qui représente la durée nécessaire pour obtenir les sorties. En outre, des critères physiques peuvent être observés afin de s'assurer que le comportement du SoC est compatible avec son usage, tels que la consommation électrique ou la température de fonctionnement.

L'approche la plus utilisée pour s'assurer de ces critères est la simulation conjointe des deux parties logicielle et matérielle appelée aussi la co-simulation. Elle consiste à simuler l'ensemble du système et à fournir au développeur des informations sur le déroulement de l'application sur la plateforme matérielle (le temps d'exécution, la consommation, etc.). Ces

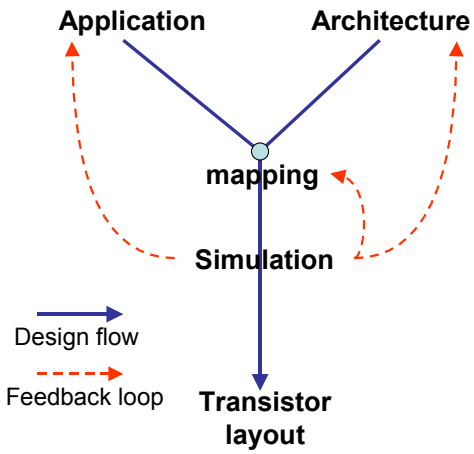


Figure 1.1: *Schéma d'organisation de la conception en Y*

informations permettent de localiser les points à optimiser/modifier dans le logiciel, dans le matériel ou dans l'association qui a été réalisée entre ces deux derniers.

Dans la littérature, le terme co-simulation possède un sens plus large que simuler le logiciel et le matériel ensemble. En effet, il regroupe différents aspects de simulation tels que multi-modèle de calcul, multi-langage et multi-niveau [42]. Au cours de cette thèse, nous nous limitons à la co-simulation logicielle/matérielle. Cependant, la problématique d'interopérabilité entre différents langages et niveaux de simulation a été abordée dans thèse de Lossan bonde [8], ancien doctorant de l'équipe.

Plusieurs environnements de co-simulation logicielle/matérielle issus des universités ou d'industriels ont été proposés. Le plupart de ces environnements permettent la co-simulation pour la validation des systèmes au niveau RTL. Ils offrent la possibilité de simuler en parallèle du logiciel et du matériel. La partie logicielle est exécutée en utilisant un ou plusieurs simulateurs au niveau du jeu d'instructions (ISS) et la partie matérielle est exécutée sur un simulateur matériel au niveau RTL. Comme exemple d'environnement permettant ce type de co-simulation, nous citons Mentor Graphics Seamless CVE [28] et CoWareN2C [14]. Certes que ces outils offrent un bon niveau de précision mais ils sont incapables de s'adapter à la complexité des systèmes multiprocesseurs que nous visons. En l'occurrence, la co-simulation des deux parties logicielle et matérielle doit se faire dans des niveaux d'abstraction plus haut.

Généralement, un flot de conception intègre plusieurs niveaux d'abstraction pour la co-simulation du système. Ces niveaux offrent différent compromis entre la vitesse de simulation et la précision des résultats obtenus. Au fur et à mesure que la modélisation passe d'un niveau élevé à un l'autre bas, l'espace des solutions architecturales devient de plus en plus réduit. Ainsi, à la suite de nombreuses itérations entre la modification du système et la simulation, nous obtenons une solution de bonne qualité qui respecte jusqu'à un certain niveau les contraintes imposées.

### 1.2.1 Niveaux de simulation

L'utilisation de plusieurs niveaux de modélisation offre aux développeurs l'avantage de disposer d'un outil de simulation et d'estimation des performances dès les premières phases de conception. Même si le niveau de précision des estimations (vitesse d'exécution, consommation, etc.) obtenues sont sont relativement faibles, l'objectif est d'éliminer de l'espace de recherche les solutions architecturales les moins performantes. Au prix d'une augmentation des temps de simulation, le niveau de précision peut être augmenté par un raffinement dans la description du système. Plusieurs travaux de recherches proposent une classification des niveaux d'abstraction, en allant du niveau le plus haut vers le niveau le plus bas. Une multitude de définitions des niveaux de modélisation existe. Généralement, les points de différence concernent soit la terminologie ou bien le niveau de précision dans l'estimation des performances dans les parties calcul et communication. Dans notre travail, nous avons adopté la classification présentée par STMicroelectronics [24]. La figure 1.2 résume les niveaux d'abstraction que nous allons détailler dans la suite de cette section.

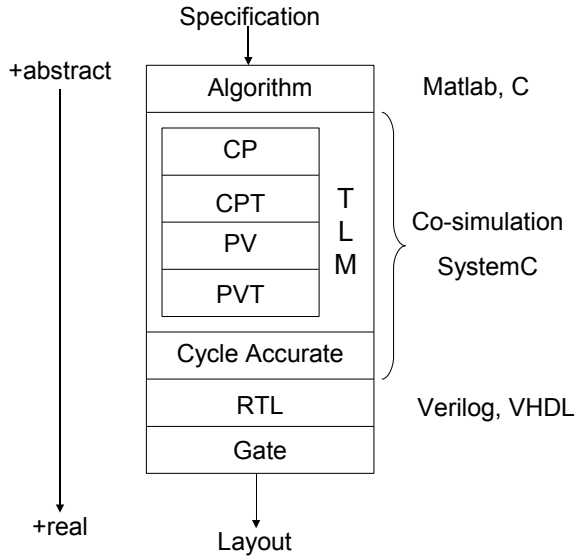


Figure 1.2: *Les différents niveaux d'abstraction pour la description d'un SoC*

#### 1.2.1.1 Algorithme

A ce niveau de la description, nous disposons seulement de la description de l'application sous forme algorithmique. Pour un certain nombre d'applications, l'algorithme, ou les algorithmes, sont connus et documentés. Ceci est le cas lorsque soit ces derniers ont déjà été utilisés dans la réalisation d'une précédente version du SoC ou bien qu'ils font l'objet d'une norme telle que les algorithmes de codage ou décodage vidéo. A ce niveau de la description, l'architecture matérielle du système n'est pas définie et les algorithmes sont décrits dans des langages de haut niveau, comme Matlab ou C++. L'utilité de ce niveau est la possibilité de réalisation une vérification fonctionnelle précoce de l'application du fait qu'une exécution de celle-ci sur la machine hôte est réalisable.

#### 1.2.1.2 Niveau transactionnel: TLM

Depuis les premières publications sur TLM (Transaction Level Modeling) en 2000 [23, 29], plusieurs définitions et classifications des différents niveaux de TLM ont été présentées dans la littérature [18] [11] [15] [21]. Toutes ces propositions ont en commun les points suivants:

- TLM est présenté comme une taxonomie de plusieurs sous-niveaux.
- les aspects communications et calcul au niveau des composants sont séparés.
- Les transactions entre les modules sont simplifiées en utilisant des méthodes de communication qui sont appelées via des canaux (*Channels*) [23].

Parmi les classifications existantes, nous adoptons celle qui a été proposée par Donlin [18].

**Le sous-niveau processus communicant (CP)**: Dans le sous-niveau CP, l'application est découpée en tâches encapsulées dans des processus communicants. Dans CP il est possible donc de mesurer la quantité d'opérations de communication entre les tâches. Ce sous-niveau permet une première expression du parallélisme dans l'application sans se référer à l'architecture du système. Des annotations temporelles peuvent aussi être spécifiées au sous-niveau CP pour estimer le temps d'exécution. Dans ce cas, nous obtenons le sous-niveau CPT ce qui correspond à une modélisation sous forme de processus communicants temporisés.

**Le sous-niveau Vue du Programmeur (PV)**: C'est uniquement à ce niveau que l'architecture du système est prise en compte et où la co-simulation logicielle/matérielle devient possible. La partie matérielle est représentée sous forme de composants de différents types (processeurs, mémoires, réseau d'interconnexion, etc.). Des modules de communication sont utilisés comme mécanismes de transport des données et des politiques d'arbitrage entre les requêtes sont appliquées au niveau des ressources partagées. La description des composants doit être suffisamment précise pour que le concepteur puisse exécuter l'application de façon similaire au système final. Ainsi, le sous-niveau PV permet une première vérification du système entier (logiciel et matériel). Néanmoins, les propriétés non fonctionnelles telles que le temps d'exécution ou la consommation d'énergie, sont soit omises ou bien approximées. Dans ce dernier cas, des annotations temporelles sont ajoutées à la description pour obtenir le sous-niveau PVT, qui correspond à un niveau "vue du programmeur temporisé". L'objectif dans PVT est d'atteindre des estimations de performance proches de celles obtenues par les bas niveaux de description.

#### 1.2.1.3 Niveau Cycle Précis Bit Précis (CABA)

Au niveau CABA, le système est décrit de façon précise d'un point de vue temps d'exécution des opérations. Il permet de simuler le comportement des composants au cycle près. Au niveau calcul, une description de la micro-architecture interne du processeur (pipeline, prédiction de branchement, cache, etc.) est réalisée. Au niveau communication, un protocole de communication précis au bit près est adopté. Cette description détaillée et fine du système permet d'améliorer la précision de l'estimation des performances.

#### 1.2.1.4 Niveau RTL

La modélisation au niveau RTL correspond à la description de l'implémentation physique du système sous forme de blocs élémentaires (bascules, multiplexeurs, UALs, registres, etc.) et à utiliser la logique combinatoire pour relier les entrées/sorties de ces blocs. En général, nous utilisons les outils de synthèse standard pour obtenir le masque (*layout*) du système à partir du niveau RTL.

### 1.2.2 Langages de description

Comme nous venons de le voir, il est important de pouvoir modéliser un système complet, comprenant des parties logicielles et matérielles, à plusieurs niveaux d'abstraction. Pour faciliter cette approche et permettre un passage rapide entre les niveaux, il serait intéressant d'utiliser un même langage de description pour couvrir tout le flot de conception tout en restant exécutable à tous les niveaux d'abstraction. Malheureusement, ce langage n'existe pas encore [43]. Etant donné que le logiciel est typiquement développé en C/C++ et le matériel en langage HDL (Hardware Description Language), plusieurs initiatives font aujourd'hui de l'effort pour unifier le langage de description. Certaines initiatives tentent d'adapter le C/C++ à la description matérielle, d'autres cherchent à étendre les HDL à la description système [40]. Une troisième voie appelle à la spécification de nouveaux langages.

Dans le cadre de cette thèse, nous nous intéressons à utiliser la co-simulation logicielle/matérielle dans la conception des MPSoC. Notre objectif est de réussir à évaluer les performances du système avant d'atteindre les phases finales de conception (au-delà de RTL). Le succès dans cet objectif nécessite tout d'abord le choix d'un langage bien adapté à la description de haut niveau, permettant ainsi l'exécution de notre système tout entier dans un temps raisonnable. Il sera intéressant aussi que le langage de description choisi permette de visualiser le comportement dynamique du système. Au cours de notre développement, le standard SystemC a été adopté comme langage de description des systèmes MPSoC. En effet, les spécificités de ce langage répondent le mieux à notre problématique de co-simulation comme nous le détaillerons dans ce qui suit.

#### 1.2.2.1 SystemC et la librairie TLM

SystemC est une plate-forme de modélisation composée de librairies de classes C++ et d'un noyau de simulation. Il introduit de nouveaux concepts (par rapport au C++ classique) afin de supporter la description du matériel et ses caractéristiques inhérentes comme la concurrence et l'aspect temporel. Ces nouveaux concepts sont implémentés par des classes C++ tels que les modules, les ports, les signaux, les FIFO, processus (threads et methods), etc., permettant ainsi de faire la conception à différents niveaux d'abstraction. L'initiative SystemC se développe dans le cadre de l'OSCI (Open Source systemC Initiative) [51] qui est chargée de diffuser, promouvoir et rédiger les spécifications de SystemC. Depuis décembre 2005, SystemC est standardisé auprès de l'IEEE sous le nom de IEEE 1666-2005.

Au cours de ces dernières années, SystemC a suscité un intérêt de plus en plus grand auprès des chercheurs et des industriels relativement à d'autres langages. La première raison de cet intérêt est le fait que SystemC soit basé sur le standard C++ largement utilisé pour le développement logiciel. Il devient donc possible d'utiliser les outils conventionnels de débogage qui sont généralement maîtrisés par les développeurs pour la vérification

fonctionnelle. Par ailleurs, l'utilisation de SystemC pour modéliser les différents composants d'un système embarqué autorise l'utilisation d'un seul moteur de simulation. Ceci se traduit par un gain en temps de simulation et évite principalement l'utilisation des techniques de synchronisation entre différents moteurs de simulation. Les expériences réalisées par exemple chez NEC [71], montre que la conception des systèmes sur puces basée sur le standard C ou C++ offre des résultats intéressants en terme de gain en temps de développement et vérification.

La librairie TLM est parmi les principaux attraits de SystemC. Elle définit de nouveaux concepts permettant de supporter la description du matériel et du logiciel au niveau transactionnel. Principalement, nous identifions les *fonctions de transport* décrites dans les modules de communication qui servent à convoyer les transactions entre les initiateurs et les cibles et vice versa. Ces transactions se présentent sous formes soit des requêtes de commande ou bien des requêtes de réponse. Afin de préciser le format de chaque type de requête ainsi que la liste des arguments nécessaires, un *protocole de communication* est définie. Ce dernier fourni un ensemble de *méthodes de communication* jouant le rôle d'API (Application Program Interface) pour l'utilisateur, facilitant ainsi la modélisation TLM. A titre d'exemple, pour initier une opération de lecture ou d'écriture, des appels fonctions de type *read()* ou *write()* sont utilisés. Ces fonctions sont transmises à travers les *ports* des composants connectés à des canaux de communication (appelés aussi *Channels*).

En résumé, SystemC possède des caractéristiques intéressantes pour la description de l'architecture à haut niveau. Cependant, ce langage n'est, pour le moment, pas synthétisable. Il n'est donc pas possible de l'utiliser au-delà du niveau RTL. Pour cette raison SystemC est utilisé en général pour les tâches de modélisation et de vérification rapide du système. De plus, comme il n'a été standardisé que récemment, son adoption par les grands consortiums de systèmes sur puce risque de demander un certain temps.

#### 1.2.2.2 Autres langages de description

D'autres langages que SystemC peuvent être aussi utilisés pour une co-simulation logicielle/matérielle des systèmes sur puce monoprocesseur ou multiprocesseur. Nous citons à titre d'exemple, SpecC [63], VHDL et SystemVerilog [1]. SpecC est un langage de description et de spécification système basé sur le standard C, avec des extensions syntaxiques. Ces extensions comprennent, par exemple, des types de données adéquats, des commandes pour supporter la concurrence et la description de machines à états. Il a été développé à l'Université de Californie à Irvine. Au-delà du langage lui-même, les travaux menés par l'équipe de D. Gajski introduisent des concepts intéressants pour la modélisation système ainsi que des techniques de raffinement [23]. Le développement de SpecC se fait dans le cadre du STOC (SpecC Technology Open Consortium). En comparaison avec SystemC, SpecC n'est pas un langage "open source" limitant ainsi les travaux de recherche qui développent des outils autour de ce langage. SpecC est plus utilisé par les consortiums de SoC asiatiques et très peu répondu en Europe.

VHDL est le langage de description matériel le plus répondu dans le monde. Il est destiné à décrire le comportement et/ou l'architecture d'un système électronique numérique. L'utilisation de ce langage dans le monde industriel ou académique est fortement liée à une phase avancée de la conception du système. VHDL permet une description du système au niveau RTL. L'intérêt d'une telle description réside dans son caractère exécutable pour vérifier le comportement réel du circuit. En outre, des outils de CAO existants permettent de

passer directement d'une description en VHDL à un schéma en porte logique et par las suite au masque final. En comparaison avec SystemC, le langage VHDL ne permet pas la vérification de la conception à un haut niveau d'abstraction notamment TLM. Ainsi, dans un même environnement de co-simulation, ces deux langages peuvent être utilisés dans des phases de conception différentes suivant le niveau d'abstraction. Ils peuvent donc être considérés comme deux langages complémentaires. Le langage SystemVerilog présente une autre approche que SystemC ou VHDL. En effet, il résulte de l'extension du langage standard de description matérielle Verilog. A ce dernier, il a été associé de nouveaux concepts afin que SystemVerilog puisse supporter la description de haut niveau et la vérification des modèles développés. C'est ce dernier point qui fait l'avantage par rapport à SystemC. En 2005, SystemVerilog est standardisé auprès de l'IEEE sous le nom de IEEE 1800-2005.

### 1.2.3 Synthèse

Au cours de cette thèse, nous avons mis l'accent sur l'importance de la co-simulation logicielle/matérielle pour la validation comportementale des MPSoC. Face à l'obstacle des temps de simulation résultant des outils traditionnels au niveau RTL, de nouvelles approches doivent être adoptées. Aujourd'hui, l'utilisation du niveau CABA pour co-simuler les MPSoC est devenue une solution préférée pour les chercheurs académiques et industriels [**?**, 4]. En utilisant SystemC comme environnement de co-simulation, cette approche sera étudiée en détail dans notre travail afin de profiter du critère de précision offert par le niveau CABA. Cette étude fait l'objectif du chapitre suivant. Néanmoins, nous énonçons à l'avance que les temps de simulation à ce niveau ne sont pas encore satisfaisants. Pour cela, nous étudions dans la section suivante les techniques permettant d'accélérer l'évaluation des MPSoC à partir du niveau CABA.

## 1.3 Techniques d'accélération de l'évaluation des MPSoC

Toute technique visant l'accélération de la simulation pour une évaluation rapide des performances des MPSoC doit tenir compte des facteurs suivants: la vitesse de simulation pour l'évaluation de chaque alternative et le niveau de précision souhaité d'un côté et le niveau d'abstraction dans lequel est décrit le système de l'autre coté. Notons, que dans notre thèse, nous nous intéressons principalement à l'aspect architecture. Nous supposons par conséquent que l'application est disponible et qu'elle est décrite dans un langage de programmation de haut niveau (C, C++, Java ou autre) pouvant être exécutée ou interprétée par le simulateur. Les techniques existantes et visant la réduction des temps de simulation au niveau CABA peuvent être divisées en 4 grandes catégories.

### 1.3.1 Technique d'accélération par échantillonnage

Cette première technique réduit le temps de simulation en considérant un nombre limité d'instructions de l'application. Ainsi, au lieu d'exécuter l'application dans sa totalité, quelques instructions seulement sont exécutées. Cette approche peut être abordée de 2 façons:

- En générant un programme synthétique ayant le même comportement que l'application sur la plateforme mais contenant moins d'instructions.

- En sélectionnant un nombre réduit d'intervalles d'instructions de l'application mais qui sont les plus représentatifs du comportement total de l'application.

Dans la première approche, appelée "simulation statistique", le programme est simulé afin de calculer des statistiques qui caractérisent l'exécution de l'application sur la plateforme matérielle. Habituellement, dans ces méthodes nous trouvons comme statistiques mesurées: les types et les occurrences des instructions exécutées (instructions UAL, lecture/écriture, branchement, flottant, etc.), le degré de localité de l'application et des références aux données (nombre de défauts dans les caches). Ces statistiques sont alors utilisées pour générer un programme synthétique ayant un nombre d'instructions plus réduit et permettant d'estimer donc rapidement le temps d'exécution [67, 45].

La deuxième approche part du constat que dans la plupart des applications, une même partie du code est exécutée plusieurs fois. Pour réduire le temps de simulation, il suffit donc d'exécuter au niveau CABA, une seule fois les blocs d'instructions qui reviennent le plus souvent. Il est nécessaire de générer un profil de l'application au niveau des blocs de base exécutés et regrouper les intervalles d'exécution contenant les mêmes blocs de bases. Une fois, les intervalles les plus représentatifs définis, il suffit de sélectionner un échantillon de chaque groupe et d'exécuter les échantillons sur le simulateur CABA. Il a été montré que, plus grand est le nombre d'échantillons meilleures seront les estimations. Néanmoins, cette approche pose deux problèmes:

- La construction du contexte du programme avant le démarrage de chaque intervalle.
- La construction de la structure interne du processeur (contenus des caches, du pipeline, etc.).

Ces deux problèmes réduisent le facteur d'accélération potentiel de cette méthode par échantillonnage de l'application. Dans le cas des architectures multi-processeurs, la plupart des techniques existantes sont difficilement applicables du fait qu'il est impossible de connaître à l'avance le recouvrement des codes parallèles dans les différents processeurs, rendant impossible la génération des intervalles [5, 6]. Même si récemment des solutions ont été proposées pour remédier à ces limites [**?**, **?**], il reste que leur application nécessite la réalisation d'un simulateur CABA afin d'estimer les performances.

### 1.3.2 Technique d'accélération par augmentation de la granularité

La deuxième technique propose de réduire le temps de simulation en augmentant la granularité des événements considérés comme significatifs pour l'estimation des performances. En effet, habituellement, la simulation au niveau CABA des MPSoC est réalisée en commutant entre les différents contextes des processeurs à chaque cycle du simulateur. Cette méthode garantit la prise en compte des événements dès que ces derniers arrivent (cycle par cycle) ce qui offre une grande précision.

L'accélération de la simulation par l'augmentation de la granularité, utilise la spéculation sur les événements futurs. Ainsi, si l'analyse nous garantit que le prochain événement significatif pour un processeur se produira dans $N$ cycles, la simulation pour ce processeur peut avancée de $N$ cycles avant de commuter vers les contextes des autres processeurs. Cette diminution du nombre de changements de contexte par cycle permet par conséquent d'accélérer la simulation, puisque plusieurs instructions sont exécutées consécutivement

sans changement de contexte [34]. Kim et al. [35] propose une technique intéressante pour appliquer cette idée et permettre une co-simulation logicielle/matérielle rapide des MPSoC hétérogènes. L'idée ici est de combiner une simulation contrôlée par la trace et la synchronisation virtuelle. Dans une simulation contrôlée par la trace d'exécution (*trace driven simulation*), l'application est d'abord simulée pour enregistrer les événements considérés significatifs intéressants au niveau de chaque processeur pris individuellement. L'enregistrement se fait dans un fichier trace sur disque. Dans la deuxième étape, les fichiers traces des différents processeurs sont fusionnés afin d'aligner les événements et évaluer les performances. Cette étape nécessite l'utilisation d'un simulateur au niveau CABA (ISS) qui lit les fichiers traces et avance événement par événement dans les fichiers. Un problème supplémentaire qui concerne l'espace mémoire exigé pour stocker toutes les traces est posé avec cette approche. Pour éviter ce problème, les auteurs proposent de construire les traces et les consommer à la volée. Pour cela, la trace est construite partie par partie au niveau des processeurs puis consommée par le simulateur. Une partie correspond aux événements qui se sont produits entre deux accès à la mémoire partagée.

### 1.3.3 Technique d'accélération par parallélisation de la simulation

Cette technique se base sur l'utilisation des machines SMP (Symmetric Multi-processor) permettant d'exécuter en parallèle un grand nombre de processus décrivant le système [2]. Ces processus sont décrits à travers des threads. L'ordonnancement de ces derniers est assuré par un contrôleur qui se charge de vérifier les threads prêts à être exécutés et de les allouer aux processeurs disponibles. Ce type de simulation, souvent appelé "multi-thread", nécessite trois types de synchronisation. Le premier type est assuré entre le contrôleur de la simulation et les processus afin de déclencher l'exécution au bon moment. Le deuxième type de synchronisation est défini entre les processeurs de la machine lors des accès simultanés à des ressources partagées. Le dernier type est spécifié par le concepteur dans les processus afin de garantir le comportement correct du composant. Les analyses formelles des performances montrent que l'accélération est proportionnelle au nombre de processeurs de la machine tant que ce nombre est supérieur au nombre de processus du système. Dans le cas inverse, l'accélération peut atteindre au maximum la valeur du nombre de processeurs.

### 1.3.4 Technique d'accélération par abstraction du niveau de description

Enfin, dans la quatrième technique, l'accélération de la simulation des MPSoC est obtenue par l'utilisation de niveaux d'abstractions plus élevés que le niveau CABA. L'approche TLM, que nous avons présentée dans la section précédente fait partie de ces niveaux. En effet, l'accélération de la simulation par TLM est obtenue par la réduction des surcharges (ou overheads) associées à la synchronisation et la communication entre composants. Il faut noter, néanmoins que la plupart des travaux réalisés dans ce cadre se limitent à définir une classification sans proposer d'outils permettant de réaliser une estimation des performances aux différents niveaux d'abstractions définis. A notre connaissance très peu de travaux se sont intéressés à l'implémentation de TLM dans le cas des MPSoC. Deux projets méritent d'être détaillés ici. Il s'agit de Pasrisha et al. [52] et Viaud al. [70].

Pasrisha et al. ont défini et implémenté un niveau abstraction au dessus du niveau RTL. Ce niveau nommé "*Cycle Count Accurate at Transaction Boundaries*" (CCATB) correspond au niveau PVT Event Accurate que nous proposons dans le chapitre suivant. Au niveau

CCATB, dans un souci de précision, des annotations sur les délais des opérations et le protocole de communication ont été ajoutées. Néanmoins, la mise en oeuvre de CCATB nécessite la connaissance de l'état du système, comme le nombre de requêtes en lectures/écritures à un moment donnée, qui n'est possible que dans le cas des architectures à bus partagé. Les niveaux d'accélération obtenus par CCATB sont assez limités.

Viaud et al. proposent aussi une méthode visant à réduire le temps de simulation en se basant sur la théorie des événements discrets parallèles (ou Parallel Discrete Event Simulation PDES). L'objectif est d'enrichir la description TLM par un certain nombre de mécanismes et de règles permettant de déduire l'horloge des processeurs sans pour autant avoir un fonctionnement cycle par cycle. Lorsque le processeur désire envoyer une requête vers la mémoire, celle-ci comprend en plus des informations habituelles, la valeur de l'horloge locale. Au niveau du réseau d'interconnexion et du module mémoire partagé cible, un paquet contenant l'horloge locale sur chaque canal d'entrée doit être reçu avant de faire avancer l'horloge locale du réseau ou de la mémoire. De cette façon, le comportement dynamique de l'application sur la plateforme matérielle peut être construit. De cette façon, il devient facile de retrouver les événements architecturaux, tels que les contentions dans le réseau d'interconnexion ou la mémoire partagée, nécessaire à l'estimation des performances de l'architecture. Cette approche nécessite néanmoins l'utilisation de message "nuls" afin de faire avancer les horloges ce qui peut entraîner une augmentation de la surcharge sur le simulateur. Par ailleurs, cette méthode semble difficile à généraliser dans le cas des MPSoC hiérarchiques ou distribués. Les valeurs des accélérations obtenues sont assez importantes, mais celles-ci n'ont pas été mesurées sur des applications réelles.

### 1.3.5 Synthèse

Parmi les techniques déjà citées, nous avons prêté une grande attention à la technique d'accélération par abstraction du niveau de description. En effet, le développement des composants au niveau transactionnel peut se faire d'une façon plus rapide que celle au niveau CABA. Pour cela, cette technique permet d'obtenir une co-simulation du système dans les premières phases de conception en comparaison par rapport aux autres techniques. Par ailleurs, des modèles d'estimation de performance et de consommation d'énergie peuvent être intégrés dans le simulateur de haut niveau permettant ainsi une comparaison rapide entre les différentes alternatives. Enfin, cette technique offre la possibilité d'observer le comportement dynamique du système au cours de l'exécution de l'application.

## 1.4 Estimation de la consommation dans les systèmes sur puce

La réduction de la consommation d'énergie est devenue un objectif de premier plan dans la conception des SoC. En effet, du fait des taux d'intégration et des fréquences d'horloges de plus en plus élevés, il est devient nécessaire de concevoir des techniques pour réduire la consommation d'énergie. Dans le cas des systèmes embarqués, ces techniques sont intéressantes pour satisfaire les critères d'autonomie, de fiabilité et de coût. La solution à ces défis commence par le choix d'une technologie bien adaptée à la spécificité des systèmes embarqués en terme de consommation. Aujourd'hui, la technologie CMOS (Complementary Metal-Oxide Semiconductor) est la plus utilisée pour la fabrication de ces systèmes. En comparaison à d'autres technologies telles que la BiCMOS (Bipolar-CMOS) [25] ou la SOI (Silicon On In-

sulator) [7], la CMOS permet d'implanter l'électronique numérique et analogique au même niveau de performance. En outre, elle offre des meilleurs compromis entre les propriétés suivants: haute intégration, coût de fabrication et faible consommation électrique.

### 1.4.1 Consommation statique et Consommation dynamique

Depuis longtemps, la majeure partie de la consommation d'énergie a lieu pendant les transitions des nIJuds logiques dans un circuit. Ainsi, cette source de consommation est appelée dynamique tandis que la deuxième source qui correspond à l'état de repos du circuit est appelée statique. Cette dernière est considérée pratiquement nulle. Néanmoins, cela n'est plus vrai pour les nouvelles technologies submicroniques où les courants de fuite deviennent de plus en plus importants.

En général, la consommation des circuits CMOS en terme de puissance est calculée par l'équation suivante [69]:

$$P = P_{statique} + P_{dynamique} \tag{1.1}$$

#### 1.4.1.1 La consommation statique

La consommation statique dans un circuit est due principalement à trois types de courants de fuites dans le transistor [59, 69]:

- Le courant sous le seuil, $I_{seuil}$ (subthreshold current) qui est considéré le composant le plus important de la consommation statique [59]. Ce courant circule entre le drain et la source du transistor quand il y a une différence de potentiel drain-source et le transistor se trouve à l'état bloquant. Cet état correspond à une tension grille-source $V_{gs}$ inférieure à la tension de seuil $V_{th}$ [69]. A ce moment, le transistor CMOS laisse passer un courant qui dépend exponentiellement de la tension $V_{th}$.
- Le courant à travers la grille due à la réduction des dimensions du transistor et donc de l'épaisseur de l'oxyde de la grille, provoque la rupture de l'impédance infinie entre la grille et le substrat. Dans ce cas, la différence de potentiel grille-substrat donne lieu à un effet tunnel avec un transfert d'électrons. Un nouveau courant de fuite apparaît entre la grille et le substrat, $I_{grille}$ (gate leakage current), qui circule toujours quand il y a une tension grille-substrat. Ce courant peut se produire à tout moment quel que soit l'état du transistor (passant ou bloquant) [59].
- Le courant de fuites des jonctions, $I_{diode}$ ou courant d'injection dans le substrat correspond aux fuites dans les diodes des jonctions PN (illustré dans la figure 1.3). Ce courant circule à travers ces jonctions continuellement, quel que soit l'état du transistor (passant ou bloquant) [59].

A température constante, la puissance statique d'un transistor dépend donc principalement de ces trois courants $I_{seuil}$, $I_{diode}$ et $I_{grille}$ et de la tension d'alimentation $V_{dd}$ selon l'expression suivante [59]:

$$P_{statique} = I_{fuites} \times V_{dd} = (I_{seuil} + I_{diode} + I_{grille}) \times V_{dd} \tag{1.2}$$

Pour les technologies en dessus de 0.25$\mu$m, les courants de fuite étaient faibles ce qui implique que la consommation statique n'est pas prise en compte au cours de la conception des circuits. A l'opposé, dans les nouvelles technologies submicroniques ces courants

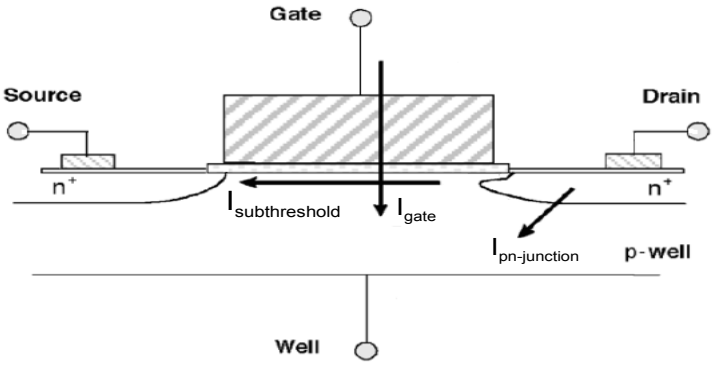


Figure 1.3: *Courants de fuites dans un transistor CMOS*

occupent une grande importance pour déterminer les caractéristiques finales des circuits. Donc, il devient nécessaire de les prendre en compte. A titre d'exemple, pour le processeur Itanium d'Intel conçu avec une technologie 0.13$\mu$m, cette consommation occupe 14% de la dissipation totale du circuit [31].

La figure 1.4 (source:EETimes June 2002) donne des prévisions sur la contribution de la consommation statique en fonction de la technologie de fabrication. Selon cette source, à partir d'un processus de 70 nm l'augmentation prévue des valeurs des courants de fuite sera tellement considérable, que la consommation statique dépassera la consommation dynamique. Des nouvelles technologies devront être mises en place pour faire face à ce problème.

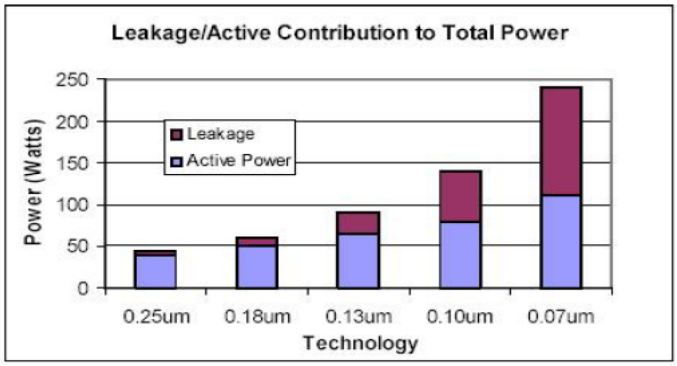


Figure 1.4: *Contribution de la consommation statique et dynamique dans un SoC*

#### 1.4.1.2 La consommation dynamique

La consommation dynamique se produit à chaque transition d'un nIJud logique dans un circuit CMOS. Elle est due principalement à deux types de courants: le courant de court-circuit et celui de charge [54]. La puissance dynamique est exprimée avec l'équation suivante:

$$P_{dynamique} = P_{court-circuit} + P_{charge} \tag{1.3}$$

Lors de la commutation d'une porte CMOS, il arrive un moment où des transistors de

type PMOS et NMOS sont passants en même temps, ce qui crée un courant de court-circuit entre l'alimentation et la masse. La puissance de court-circuit qui en résulte est donnée par l'équation suivante:

$$P_{cc} = f \times I_{cc} \times V_{dd} \tag{1.4}$$

Où $I_{cc}$ est la valeur moyenne du courant de court-circuit lors d'une transition, $f$ est la fréquence des transitions et $V_{dd}$ est la tension d'alimentation. Les études réalisées par Nose [44] nous montrent que la consommation de court-circuit peut augmenter avec l'évolution des technologies submicroniques et des tensions de seuil $V_{th}$ très basses. Cela se produit dans les circuits fonctionnant à basse tension $V_{dd}$ pour augmenter la vitesse de commutation. Dans ce cas, les courants de court-circuit augmentent considérablement et deviennent une source importante de consommation, environ 20% de la dissipation totale du circuit. Pour diminuer cette source, les circuits doivent être correctement conçus en utilisant des valeurs de tension $V_{dd}$ et $V_{th}$ bien calibrées.

La consommation de charge dynamique d'un circuit CMOS représente la puissance nécessaire pour charger la capacité $C$ de la sortie de la porte logique à un niveau correspondant à la tension d'alimentation $V_{dd}$, puis pour la décharger à la masse. Cette partie est la composante la plus importante de la consommation d'énergie. C'est elle que nous appelons souvent puissance dynamique. La puissance moyenne dissipée dans une porte logique est calculée de la façon suivante [54]:

$$P_{dynamique} = F \times C \times V_{dd}^2 \tag{1.5}$$

Avec $F = \frac{1}{T}$, T est le temps de réponse de la porte. Pour un système synchrone composé de plusieurs millions de portes, la puissance totale est déduite par l'équation suivante:

$$P_{dynamique} = F \times \alpha \times N \times C \times V_{dd}^2 \tag{1.6}$$

Où N est le nombre total de portes et $\alpha$ est l'activité du circuit et correspond au pourcentage de portes qui changent de niveau.

### 1.4.2 Outils d'estimation de la consommation

L'estimation de la consommation dans les systèmes sur puce nécessite des outils d'analyse capables d'évaluer les différentes sources statiques et dynamiques lors de l'exécution de l'application. Cette analyse peut être faite d'une façon plus ou moins fine, suivant le niveau d'abstraction auquel notre système est décrit. Dans la littérature, il existe une variété d'outils permettant l'estimation de la consommation à différents niveaux d'abstraction. Ces outils offrent différents compromis entre la précision de l'estimation et sa rapidité.

Dans cette thèse, l'objectif n'est pas de proposer des méthodes d'optimisation de la consommation de puissance mais plutôt de développer des outils d'estimation de la consommation pour les systèmes MPSoC à différents niveaux d'abstraction. Ces outils peuvent être utilisés dans un flot de conception de MPSoC pour réaliser une exploration architecturale et trouver l'alternative optimale en terme de performance et de consommation.

Dans la suite de cette section, nous présentons de façon synthétique les travaux d'estimation existants à tous les niveaux de conception. Nous positionnerons par la suite nos travaux et nos contributions par rapport aux autres travaux dans le domaine d'estimation de la consommation pour les MPSoC.

#### 1.4.2.1 Niveau transistors

Les outils d'estimation à ce niveau de la modélisation permettent le calcul de la consommation prenant en compte les courants électriques circulant dans les transistors. La consommation dépend des caractéristiques physiques des transistors telles que les valeurs des capacités, la tension d'alimentation et de seuil, etc. Comme nous pouvons l'imaginer pour réaliser ces estimations, il est nécessaire de disposer de la description du SoC au niveau transistor. Le concepteur pourra utiliser ces estimations pour choisir les cellules (transistors) qui répondent au mieux à ses objectifs [13]. Il est aussi possible d'utiliser ces estimations pour optimiser le placement/routage de ces cellules [3]. Parmi les outils existants à ce niveau, nous pouvons citer SPICE [50], PowerMill de Synopsys [64] ou Lsim Power Analyst de Mentor Graphics [27]. Même si les estimations obtenues à ce niveau sont très proches des valeurs du circuit réel, à l'exception des petites variations causées par l'environnement de fonctionnement telles que la température, le temps de simulation est très important. Cet inconvénient constitue un obstacle majeur de ces outils. Dans le cas des MPSoC intégrant des centaines de millions de transistors, l'utilisation de cette approche pour évaluer la consommation totale de plusieurs centaines d'alternatives devient irréalisable.

#### 1.4.2.2 Niveau portes logiques

La réalisation de l'estimation de la consommation de puissance à ce niveau est réalisée par l'utilisation d'une bibliothèque de portes logiques. Cette bibliothèque donne pour chaque porte ces caractéristiques en terme de consommation de puissance[1]. Cette approche permet une première réutilisation de blocs afin de réduire la complexité de conception. Le concepteur peut visualiser les courants au niveau les différentes portes logiques et ainsi contrôler la consommation du circuit. La fréquence d'horloge et les tensions d'alimentation des portes sont des exemples de paramètres qui peuvent être explorés à ce niveau. Pour ce faire, des techniques telles que les horloges inhibées (clock gating) ou l'adaptation dynamique de la fréquence et de la tension (frequency/voltage scaling) sont appliquées. Comme exemple d'outils à ce niveau, nous trouverons, entre autres, PowerGate de Synopsys [65] et Diesel de Philips [53].

#### 1.4.2.3 Niveau RTL (Register Transfer Level)

Afin d'accélérer la simulation du système, ce niveau de conception vise la réutilisation de blocs de granularité plus grossière que les deux niveaux précédents. La modélisation revient à décrire l'implémentation sous forme d'éléments logiques et séquentiels tels que les registres ou les bascules et à relier les entrées/sorties de ces éléments pour construire un composant du SoC. Plusieurs outils industrielles, tel que Petrol [39] de Philips, permettent l'évaluation de la consommation au niveau RTL. Les concepteurs emploient généralement la simulation avec des stimuli en entrée pour obtenir les valeurs exactes de la consommation des éléments, ce qui permet d'avoir une faible erreur d'estimation. Cette erreur peut varier entre 10 et 15% par rapport aux résultats obtenus sous SPICE [39, 60].

En résumé nous pouvons dire, la simulation d'un système MPSoC complet aux trois niveaux d'abstraction précédents souffre d'une certaine lenteur. Ce problème s'accentue avec l'utilisation d'un nombre important de processeurs. Nous sommes contraints à faire des

[1] nous dirons par la suite qu'une caractérisation des composants a été réalisée.

estimations en composant le système sur plusieurs modules. En mesurant la consommation de cette façon, l'influence des interactions inter-blocs n'est pas prise en considération, ainsi que l'évolution de la consommation pendant le fonctionnement de l'application ne peut pas être visualisée. De ce fait, l'utilisation d'outils de modélisation au niveau transistor, portes logiques ou RTL n'est pas appropriée pour la réalisation de l'exploration des architectures MPSoC.

#### 1.4.2.4 Niveau architectural

Pour contourner l'obstacle du temps, des niveaux d'abstraction plus élevés que ceux présentés précédemment doivent être utilisés pour l'estimation de la consommation de puissance. Plusieurs travaux de recherche proposent des solutions visant à abstraire la description de certains détails d'implémentation. Au prix d'une certaine perte de précision, l'estimation de la consommation de puissance pourra se faire en considérant des événements de niveaux de granularité plus élevés que la commutation dans un transistor ou le changement d'état d'une porte logique. Les événements identifient en réalité les activités pertinentes dans le niveau de modélisation considéré et qui consomment une part importante d'énergie. Ces activités sont spécifiques à chaque type de composant (processeur, mémoires, etc.). Elles concernent généralement le type et le nombre d'instructions exécutées, les accès mémoires en lecture et en écriture, etc. L'estimation de la consommation du système est obtenue en deux étapes. La première consiste à obtenir les occurrences des activités pertinentes pendant ou après l'exécution de l'application. Ensuite, les valeurs de ces occurrences sont transmises aux modèles de puissance pour calculer la consommation dans chaque composant du système. Pour implémenter cette méthodologie, plusieurs approches, offrant différents compromis entre la précision et rapidité, existent. Dans ce qui suit, nous détaillons les approches qui nous ont parues comme étant les plus importantes:

- **Analyse de la consommation par trace:** La première approche est proposée par l'outil AVALANCHE [38, 30]. Elle consiste à estimer la consommation en se basant sur une analyse de la trace d'exécution de l'application. Cette dernière est exécutée en utilisant un simulateur de processeur décrit au niveau du jeu d'instructions (Instruction Set Simulator ou ISS). Les instructions exécutées servent à estimer la consommation du processeur. Les accès mémoire sont aussi détectés par un traceur de requêtes, qui envoie ces informations à l'outil Dinero (figure 1.5) pour les traiter. Ce dernier décide s'il s'agit d'un succès de cache ou bien d'un défaut de cache nécessitant l'accès à la mémoire de données ou d'instructions. Les occurrences de ces activités seront transmises à des modèles de puissance de chacun des composants pour calculer la consommation totale. Cette approche a été l'une des premières méthodes traitant le problème de l'estimation de la consommation des SoC au niveau système. Elle présente l'avantage de la précision des modèles de puissance des composants. Le modèle du processeur est composé de la consommation de toutes les instructions et des cycles d'attente dus aux défauts de cache. Le courant consommé par chaque instruction est mesuré de façon très précise avec la méthode proposée par Tiwari [68]. Nous présenterons de façon détaillée cette dernière méthode dans la section 1.4.2.5. Cette approche souffre cependant de quelques inconvénients. Le premier concerne les effets des interactions temporelles entre les composants qui ne sont pas évaluées. A titre d'exemple, les délais dus aux conflits lors des accès simultanés à la même cible (mémoire) ne sont pas pris en compte.

Ce type d'événements à une importance majeure pour la précision dans l'estimation de la performance et de la consommation des MPSoC. L'absence de la modélisation des interconnexions est une deuxième source d'erreur dans l'estimation. Par ailleurs, cette approche ne permet pas d'analyser l'évolution de la consommation dans le temps. Les valeurs de consommation ne sont obtenues qu'à la fin de la simulation et pour des paramètres fixés au début de la simulation.

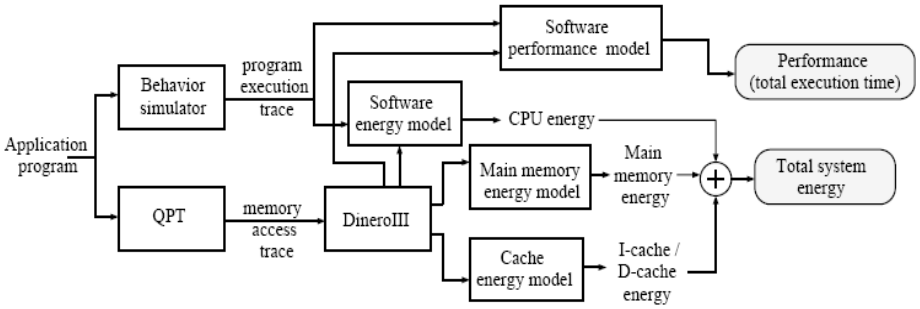


Figure 1.5: *Estimation de l'énergie avec l'outil Avalanche [38]*

- **Analyse de la consommation au cycle près:** Pour remédier à l'imprécision de l'approche précédente, plusieurs travaux de recherche proposent l'évaluation du système au niveau architectural en se basant sur une simulation cycle précis. L'approche consiste à modéliser le comportement des composants avec un langage de description comme C ou SystemC, et à assurer la synchronisation entre les différents composants du système. La description doit être suffisamment précise pour pouvoir simuler le comportement au cycle près des composants avec des données réelles. L'estimation des performances à ce niveau sera donnée par le simulateur d'architecture en nombre de cycles. Pour estimer la consommation, un modèle de consommation correspondant à chaque composant est intégré dans le simulateur d'architecture. Au cours de la simulation, la consommation est calculée à chaque cycle en se basant sur les occurrences des activités pertinentes des composants. Parmi les outils décrits avec cette approche, nous citons Wattch [10] et SimplePower [73]. L'architecture du système est constituée principalement d'un processeur Superscalaire et d'une hiérarchie de mémoires. Le but de ces outils est de servir d'outil pour optimiser la micro-architecture du processeur pour une application donnée. Cette optimisation concerne aussi la hiérarchie mémoire afin de trouver la meilleure configuration en performance et en consommation. Cette approche permet d'obtenir une précision acceptable et un temps de simulation raisonnable. Néanmoins, elle ne traite que des systèmes monoprocesseur. La plateforme MPARM [4], développée à l'Université de Vérone, présente un environnement de simulation des systèmes MPSoC au niveau CABA (le niveau architectural le plus précis). Elle intègre un modèle de puissance pour chaque composant, y compris le réseau d'interconnexion, permettant l'évaluation de la consommation au cycle près. Cette description offre les estimations les plus précises au prix d'une augmentation du temps de simulation comme il sera détaillé dans le chapitre suivant.

- **Analyse de la consommation par co-simulation:** L'outil Ptolemy [36] présente une troisième approche en se basant sur une co-estimation matérielle/logicielle à plusieurs

niveaux d'abstraction. Cet outil permet la simulation des systèmes MPSoC dont les composants sont décrits à différents niveaux d'abstraction. Pour simuler la partie logicielle du système, un simulateur au niveau du jeu d'instructions (ISS) est utilisé. La partie matérielle est décrite et simulée au niveau RTL ou portes logiques. L'outil Ptolemy permet ainsi la simulation conjointe de tout le système par la prise en compte de la synchronisation et les transferts d'informations entre les différents simulateurs. La co-estimation de la consommation au niveau architectural est obtenue en utilisant plusieurs outils d'estimation correspondant aux différents composants. Ces outils fonctionnent de façon concurrente et synchronisée. Le principal avantage de cette approche est l'augmentation de la précision de l'estimation. Cela permet de prendre en compte les effets des interactions temporelles entre composants grâce à la synchronisation des simulateurs. Un autre avantage vient du fait que cette approche permet de suivre l'évolution de la consommation au cours du temps de tout le système. Par conséquent, elle permet de trouver le maximum de dissipation dans chaque composant en vue d'optimiser la consommation (figure 1.6). L'inconvénient majeur de cette approche est la lenteur de l'estimation provenant de la simulation de quelques composants au niveau RTL ou portes. Ce problème est amélioré en partie par les techniques d'accélération proposées, mais qui diminuent la précision de l'estimation. En outre, l'utilisation de la synthèse rapide des composants matériels dédiés peut introduire des erreurs dans les estimations.

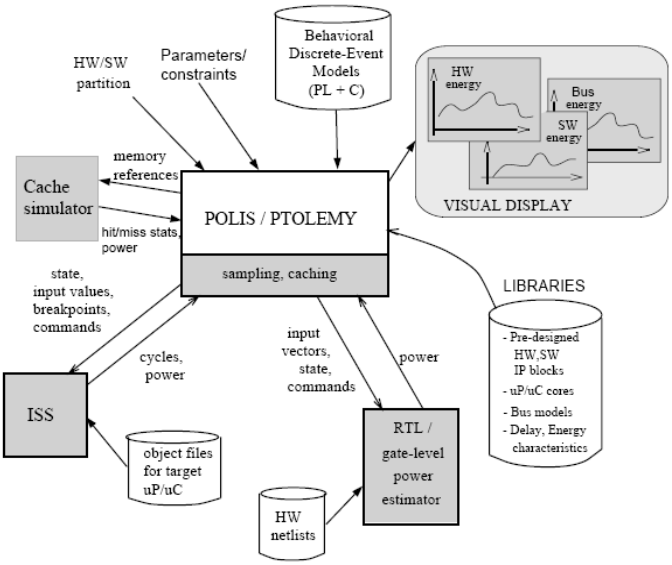


Figure 1.6: *Co-estimation de la consommation avec l'outil Ptolemy [36]*

#### 1.4.2.5 Niveau fonctionnel ou algorithmique

Encore plus haut que le niveau architecture, nous trouvons le niveau fonctionnel. Dans ce cadre, Tiwari et al. [68] ont proposé la première méthode d'estimation de la consommation d'un programme logiciel. Cette méthode est souvent considérée comme référence dans l'estimation de la consommation des processeurs et elle est applicable théoriquement à tous

types de processeurs (processeurs généralistes comme Pentium ou PowerPc, ou processeurs dédiés tel que les DSP). Sachant qu'un programme est une suite d'instructions, cette méthode consiste à cumuler l'énergie dissipée par chaque instruction. Pour évaluer les différentes instructions, il faut disposer d'abord d'un environnement expérimental réel contenant le processeur à modéliser et exécuter l'instruction pour mesurer le courant moyen consommé. La précision de cette approche est améliorée en prenant compte aussi l'interaction entre deux instructions successives au niveau des valeurs des courants. L'approche de Tiwari est très précise mais nécessite beaucoup d'analyses et de mesures et par conséquent le temps de développement d'un modèle pour un processeur cible est considérable. L'outil JouleTrack [62] développé par Sinha et al. de l'Institut de Technologie du Massachusetts (USA) fait un exemple d'environnement basé sur l'approche de Tiwari. L'outil permet l'estimation de la consommation à partir du code C pour les processeurs StrogARM SA-1100 et Hitachi SH-4. Sinha montre que pour un modèle de processeur simple, ne prenant en compte que la tension d'alimentation et la fréquence, cette approche peut donner des résultats relativement précis. Donc, elle peut être appliquée pour l'estimation de la consommation logicielle d'un processeur RISC simple dans un système embarqué. Dès que l'architecture du processeur devient plus complexe, cette approche n'est plus intéressante puisqu'elle génère des erreurs d'estimation importantes.

Plusieurs extensions ont été apportées aux travaux de Tiwari en vue de diminuer le temps de développement et l'étendre pour des processeurs plus complexes. Nous citons principalement les travaux du laboratoire LESTER et leur outil SoftExplorer [61] qui a été conçu sur la base des travaux de Tiwari. La méthode d'estimation dans SoftExplorer est basée sur une analyse fonctionnelle de la consommation dans le processeur (Functional Level Power Analysis : FLPA) [37, 32]. Grâce à cette analyse, un nombre réduit de mesures est suffisant pour déterminer le modèle de consommation. Cette méthode prend en compte toutes les fonctions du processeur tel que les étages du pipeline, les unités de traitement, les mémoires internes ainsi que les communications du cache avec la mémoire. Cette méthode est illustrée dans la figure 1.7. FLPA est réalisée en 2 étapes: la définition du modèle et l'estimation de la consommation.

- La définition du modèle est basée sur l'analyse fonctionnelle du processeur. Cette étape sera faite une seule fois. Elle consiste au début à identifier les paramètres algorithmiques pertinents agissant sur la consommation totale. Parmi ces paramètres nous citons le taux de parallélisme des instructions (ou instruction level parallelism ILP), le taux d'occupation des unités de traitement, le taux de défaut de cache du programme, etc. Le modèle de puissance du processeur sera composé d'un certain nombre de lois de consommation. Ces dernières sont calculées à partir d'un ensemble réduit de mesures expérimentales appliquées sur le processeur avec des applications élémentaires. Les lois seront ensuite décrites avec des fonctions mathématiques paramétrées en fonction des paramètres algorithmiques et d'autres paramètres de configuration tels que la fréquence, le placement des données en mémoire, etc.

- Le processus d'estimation est fait pour chaque application permettant de déduire la consommation totale. Pour ce faire, une analyse du code est faite pour extraire les paramètres algorithmiques par l'outil SoftExplorer. Ces paramètres sont par la suite insérés dans les équations du modèle de puissance pour obtenir la consommation de l'application.

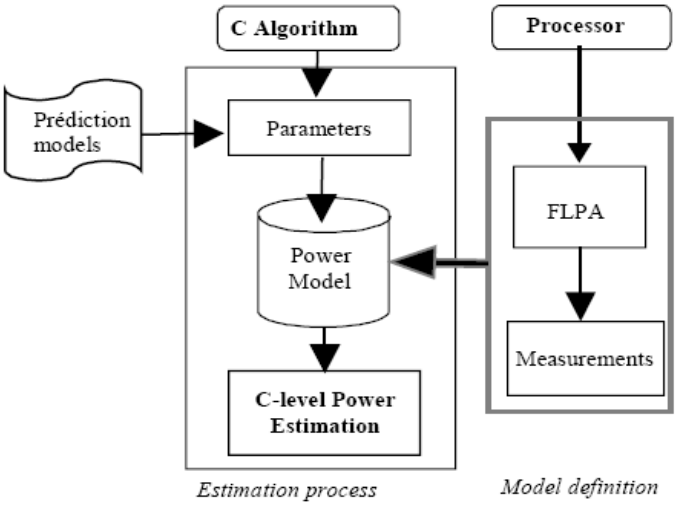


Figure 1.7: *FLPA [32]*

Cette approche a été appliquée sur plusieurs processeurs commerciaux tels que les DSP TI TMS320C6201 et TI TMS320C6701 ansi que sur le processeur ARM7TDMI [33]. L'outil SoftExplorer dispose d'une interface graphique facilitant son utilisation. L'entrée de cet outil est le code de l'application en langage C ou en assembleur. L'outil fournit comme résultat la consommation totale ainsi que la répartition de la consommation sur les différents blocs du code de l'application. Le temps d'estimation est de quelques secondes. L'avantage majeur de la méthode FLPA est la réduction du temps de développement du modèle de puissance (de l'ordre d'un mois) pour des estimations de consommation très précises. Elle est applicable aussi bien sur des processeurs VLIW que sur des processeurs RISC. Néanmoins, elle est restreinte à des systèmes monoprocesseur. Dans le contexte MPSoC, nous pensons que la méthode FLPA ne peut pas être directement appliquée. En effet, plusieurs autres paramètres sont mis en jeu tels que la cohérence des caches, les contentions sur le réseau d'interconnexion, etc. Par ailleurs, cette approche nécessite la disponibilité d'une plateforme intégrant le processeur cible à modéliser ainsi que que les appareils de mesure nécessaires tels qu'un ampèremètre, un oscilloscope, etc.

#### 1.4.2.6 Synthèse

Dans ce qui précède, nous avons présenté différentes méthodes et outils existants pour l'estimation de la consommation des systèmes sur puce à différents niveaux d'abstraction. Nous pouvons dire en guise de conclusion de cette section que, l'évaluation à des bas niveaux (au-dessous du niveau RTL) est précise mais coûteuse en termes de temps de simulation. L'estimation à des niveaux plus hauts est plus avantageuse puisqu'elle permet d'accélérer la phase d'exploration. Cependant, plus la modélisation s'éloigne du niveau RTL, plus il devient difficile de tenir compte des détails de l'architecture et plus grande devient l'erreur d'estimation. Dans notre travail, nous proposons une approche qui profite des avantages des différents niveaux. En effet, nous nous appuyions sur un nombre réduit d'expérimentations de bas niveau pour caractériser les activités pertinentes de nos composants en terme de consommation. Les coûts énergétiques qui en résultent seront ensuite

utilisés dans le niveau architectural (CABA ensuite PVT). Ceci correspond à une méthode d'estimation hybride qui satisfait au critère de précision et de rapidité. D'un autre coté, nous avons constaté qu'une attention est de plus en plus grande est accordée à la consommation statique. Cette dernière devient un facteur essentiel notamment dans les nouvelles technologies submicroniques. Pour cette raison, cette source a été prise en compte dans tous les modèles de consommation qui ont été développés dans notre travail.

La majorité des outils présentés sont destinés à l'estimation de la consommation dans des systèmes monoprocesseur. Dans la littérature, peu de plateformes MPSoC intègrent ce critère pour comparer différentes configurations architecturales. Celles qui existent sont décrites au niveau CABA offrant des estimations assez précises. De ce fait, cette approche sera utilisée dans notre environnement de simulation MPSoC. Néanmoins, les accélérations de simulation obtenues à ce niveau sont assez limitées et ne s'adaptent pas avec la complexité de ces systèmes. Face à ce défi qui sera détaillé dans le chapitre suivant, nous proposons d'aborder le critère de consommation à un niveau d'abstraction plus haut. Dans ce contexte, nous sommes parmi les premiers qui intègrent ce critère dans des systèmes MPSoC au niveau transactionnel. Nous allons démontrer que ce niveau est efficace pour l'estimation de la consommation tout autant que pour la vérification rapide du système.

L'estimation de l'énergie ou de la puissance se base principalement sur le développement de différents modèles de consommation. Dans les outils existants, la méthodologie de développement de ces modèles était négligée ou bien détaillée pour un composant spécifique tel que le processeur dans SoftExplorer. Nous considérons cette phase est importante afin de permettre aux concepteurs d'adapter facilement ces modèles à des nouvelles architectures de composants. Dans notre travail, nous proposons une méthodologie pour le développement des modèles de consommation à différents niveaux d'abstraction. Cette méthodologie sera appliquée aux différents types de composants constituant le système MPSoC afin d'obtenir des modèles paramétrables et précis.

## 1.5 Environnements de conception basés sur une approche modèles

A ce point nous avons introduit la co-simulation logicielle/matérielle et son attrait pour accélérer l'évaluation des MPSoC. En outre, la prise en considération des critères de performance et de consommation d'énergie offre une meilleure fiabilité à l'exploration des solutions architecturales. Maintenant, nous allons nous attacher à l'obtention automatique du système à co-simuler en favorisant autant que possible l'aspect de réutilisation de modèles. Nous avons donc besoin d'adopter une méthodologie de conception orientée modèles en permettant la représentation de la structure d'un composant indépendamment de son environnement d'utilisation. En effet, à partir d'une description de haut niveau du système, plusieurs environnements d'implémentation peuvent être projetés tels que une simulation SystemC à un niveau d'abstraction donné, une description RTL du système, etc.

Une des lignes directrices suivies par nos travaux est le recours à l'Ingénierie Dirigée par les Modèles (IDM) pour organiser la conception des MPSoC et la génération automatique du système à co-simuler. L'IDM doit permettre de bénéficier à la fois d'une approche orientée modèle, permettant d'abstraire et de simplifier la représentation des systèmes, et d'une chaîne complète de compilation capable d'exploiter directement les modèles. Dans ce qui suit, nous allons introduire au début le principe et les concepts fondamentaux de l'IDM.

Ensuite nous survolons quelques environnements de conception des MPSoC basés sur une approche modèle utilisant de façon implicite ou explicite les concepts de l'IDM.

### 1.5.1 Ingénierie dirigée par les modèles

Afin de pouvoir comprendre et agir sur le fonctionnement d'un système, il est nécessaire de disposer d'un modèle de ce dernier. Un modèle est une simplification et/ou une abstraction de la réalité. Modéliser consiste à identifier les caractéristiques intéressantes ou pertinentes d'un système afin de pouvoir l'étudier. Au nombre des évolutions dans le développement des logiciels, il faut citer l'apparition de plusieurs méthodes de modélisation : Merise [66], SSADM (Structured Systems Analysis and Design Methodology) [19], UML (Unified Modeling Language) [47], etc. Ces méthodes vont permettre d'appréhender le concept de modèle en informatique. Elles proposent des concepts et une notation permettant de décrire le système à concevoir. En général, à chaque étape du cycle de vie du système, un ensemble de documents généralement constitués de diagrammes permettent aux concepteurs, développeurs, utilisateurs et autres entités impliquées de partager leur perception du système. Ces méthodes de modélisation ont été parfois critiquées pour leur lourdeur et leur manque de souplesse face à l'évolution rapide du logiciel. Elles ont conduit à la notion de modèles contemplatifs qui servent essentiellement à communiquer et comprendre, mais reste passif par rapport à la production. Ainsi, après un demi-siècle de pratique et d'évolution, on constate aujourd'hui que le processus de production de logiciels est toujours centré sur le code.

L'Ingénierie Dirigée par les Modèles (IDM) vient pallier la déficience des méthodes traditionnelles de modélisation. Elle oeuvre à fournir un cadre de développement logiciel dans lequel les modèles passent de l'état passif (contemplatif) à l'état actif (productif) et deviennent les éléments de première classe dans le processus de développement des logiciels.

#### 1.5.1.1 Principe et concepts fondamentaux de l'IDM

Notons d'abord que le terme Ingénierie Dirigée par les Modèles (IDM) possède plusieurs synonymes tels que Model Driven Engineering (MDE), Model Driven Development (MDD), Model Driven Software Development (MDSD), etc. L'IDM est encore un domaine de recherche récent, dynamique et en pleine évolution. Cela se traduit par une pluralité d'idées, de concepts et de terminologies compétitives qui tendent à créer une confusion dans ce domaine. Dans cette section, nous tentons de dégager le principe et les concepts de base formant le socle de l'IDM et qui sont généralement acceptés par toute la communauté.

#### 1.5.1.2 Principe de L'IDM

L'IDM représente une forme d'ingénierie générative, par laquelle tout ou partie d'une application est générée à partir de modèles dans un processus de conception. Cette approche offre un cadre méthodologique et technologique qui permet d'unifier différentes façons de faire dans un processus homogène. Il est ainsi possible d'utiliser la technologie la mieux adaptée pour chacune des étapes du développement, tout en ayant un processus de développement global et unifié. Le principe de l'IDM consiste à utiliser intensivement et systématiquement les modèles tout au long du processus de développement logiciel. Les modèles devront désormais non seulement être au cœur même du processus, mais également devenir des entités

qui peuvent être interprétées par les machines. Un système peut être vu sous plusieurs angles ou points de vue. Les modèles offrent la possibilité d'exprimer chacun de ces points de vue indépendamment des autres. Cette séparation des différents aspects du système permet non seulement de se dégager de certains détails, mais permet également de réaliser un système complexe par petits blocs plus simples et facilement maîtrisables. Ainsi, on pourrait utiliser des modèles pour exprimer les concepts spécifiques à un domaine d'application et des modèles pour décrire des aspects technologiques. Chacun de ces modèles est exprimé dans la notation ou le formalisme les plus appropriés.

Dans le cadre de la conception des systèmes embarqués, l'approche IDM ouvre de nouvelles perspectives. Son principe de séparation des différents aspects du système, augmente la réutilisation et aide les membres de l'équipe de conception à partager et s'échanger leurs travaux indépendamment de leurs domaines d'expertise. Ce concept est particulièrement intéressant dans le domaine de conception des systèmes sur puce où les deux technologies logicielle et matérielle vont interagir pour la définition du comportement global du système.

#### 1.5.1.3 Concepts fondamentaux de l'IDM

De façon générale, l'IDM peut être définie autour de trois concepts de base : les modèles, les méta-modèles et les transformations.

**Modèle:** Un modèle est une abstraction d'un système étudié, construite dans une intention particulière. Il doit pouvoir être utilisé pour répondre à des questions sur le système. Comme le montre cette définition, la notion de modèle va de pair avec la notion de système. En effet, un modèle est conçu pour représenter quelque chose que l'on désigne ici par le terme système. Il est important de noter qu'en soi la notion de modèle apportée par l'IDM n'a absolument rien de novateur. Comme l'a fait remarquer Favre [20], on peut considérer que la notion remonte à plusieurs millénaires. Cette notion était déjà présente en informatique dans les années 1970.

**Méta-modèle:** Un méta-modèle est un langage qui permet d'exprimer des modèles. Il définit les concepts ainsi que les relations entre concepts nécessaires à l'expression de modèles. Un modèle écrit dans un méta-modèle donné sera dit conforme à ce dernier. La relation entre modèle et méta-modèle (conforme à) peut être par analogie aux langages de programmation comparée à la relation entre une variable et son type ou un objet et sa classe (dans le cas des langages objets).

**Transformation de modèles:** Comme nous l'avons déjà indiqué, l'IDM oeuvre à fournir un cadre de développement logiciel dans lequel les modèles passent de l'état passif (contemplatif) à l'état actif (productif). Un modèle est productif soit parce qu'il est directement exécutable par une machine, soit parce qu'il permet de produire d'une façon directe ou indirecte des modèles exécutables. Le second cas suppose la possibilité de pouvoir réaliser des opérations sur le modèle pour produire un autre modèle exécutable. Cette opération sur les modèles est connue dans l'IDM sous le concept de transformation de modèles.

Hubert Kadima citeKadima nous donne cette définition: Une transformation de modèles est la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources conformément à une définition de transformation. Cette définition est définie par un ensemble de règles de transformation qui décrivent globalement comment un modèle décrit dans un langage source peut être transformé en un modèle décrit dans un langage cible. La figure 1.8 illustre ce processus principal de l'IDM.

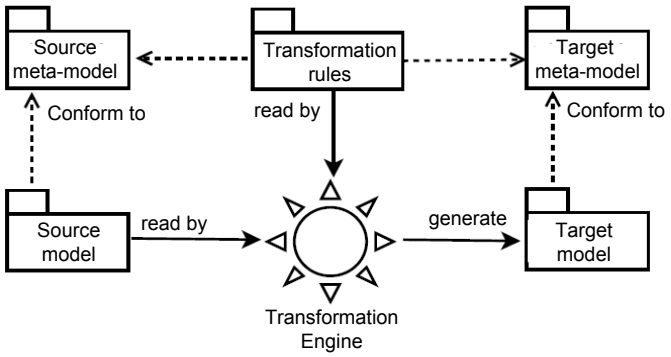


Figure 1.8: *Processus de transformation de modèles*

### 1.5.2 Gaspard

Gaspard (Graphical Array Specification for Parallel and Distributed Computing) [72] est un environnement basé sur le schéma en Y de Gajski. Il est orienté pour les applications de traitement de signal intensif. Ce type d'application est parmi ceux qui nécessitent le plus de puissance de calcul et se prête généralement bien à la parallélisation des traitements. Le traitement de signal est aussi parmi les types d'applications les plus utilisés dans les systèmes embarqués. Gaspard vise la génération de différents codes à partir d'un même placement d'une application sur une architecture à savoir:

- La génération du code VHDL correspondant à un accélérateur matériel capable d'exécuter l'application initialement modélisée à haut niveau.
- La génération de langages synchrones déclaratifs (tels que Lustre ou Signal) permettant de vérifier formellement la modélisation d'une application.
- La génération de langages procéduraux tels que Fortran/OpenMP rend possible l'exécution concurrente de différents processus sur une architecture multiprocesseur (architectures à mémoire partagée dans l'état actuel de Gaspard).
- Enfin, la génération de code SystemC permet la simulation du comportement d'un SoC à différents niveaux d'abstraction.

En effet, Gaspard regroupe des experts de différents domaines ayant comme point commun l'usage de l'IDM comme méthodologie de conception en allant de la modélisation en UML jusqu'à la génération de code. Notre travail dans Gaspard consiste à générer du code SystemC pour la co-simulation logicielle/matérielle d'un système sur puce multiprocesseurs, voire massivement parallèle, modélisé à haut niveau. Gaspard doit permettre au concepteur de modéliser indépendamment les différents aspects de notre système; application, architecture et association. A partir de ces informations et en se basant sur les recommandations de l'IDM, plusieurs transformations de modèles doivent être développées afin de générer le code du MPSoC complet, à la fois matériel et logiciel. Par suite, ce code nous permettra des simulations à des niveaux de plus en plus précis. La figure 1.9 illustre cette organisation du développement, et fait le rapprochement avec le flot de conception en Y.

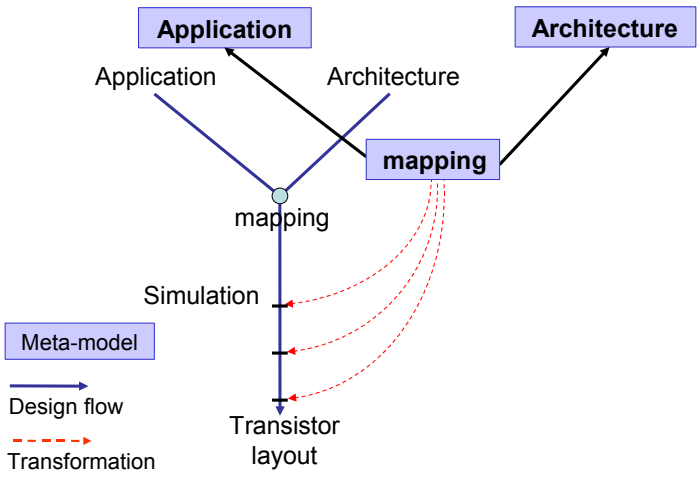


Figure 1.9: *Schéma du flot de conception en utilisant l'environnement Gaspard*

Au début de ces travaux de thèse, seul le méta-modèle permettant à l'utilisateur de spécifier un MPSoC était défini, les transformations étant encore uniquement des perspectives. Ce méta-modèle a pour origine les travaux d'Arnaud Cuccuru lors de son doctorat dans l'équipe. Il est en partie inspiré de standards OMG tels que SPT (Schedulability, Performance, and Time) [49] pour représenter l'architecture matérielle, SysML (System Modeling Language) [48] pour la mise en oeuvre de l'association, et UML pour l'approche composant. Notons au passage que le méta-modèle Gaspard fait en partie à la base du tout jeune standard MARTE (Modeling and Analysis of Real-Time and Embedded systems) [56]. Notre méta-modèle pour l'expression des MPSoC sera détaillé dans le chapitre **??**. Gaspard est également basé sur le langage Array-OL (Array Oriented Language) [17, 9] comme modèle de calcul dédié aux applications de traitement de signal intensif. Comme nous allons le montrer dans les deux derniers chapitres de cette thèse, notre environnement est capable de générer du code SystemC présentant le système à co-simuler au niveau PVT. Ultérieurement, d'autres niveaux (CABA et RTL) seront intégrés dans Gaspard. Un autre attrait essentiel de notre environnement est l'intégration des outils d'estimation de performance et de consommation d'énergie.

### 1.5.3 Profil UML 2.0 pour SystemC

Différents chercheurs de l'Université de Milan, de l'Université de Catane et du consortium STMicroelectronics ont réuni les efforts pour réaliser le profil UML 2.0 pour SystemC [57]. Leur objectif est d'arriver à générer du code exécutable SystemC à partir d'une description de haut niveau. Différentes notations UML ont été associées aux concepts SystemC comme le montre la figure 1.10. Cette notation inclut la structure, les communications, les types de données, le comportement, la synchronisation, etc. La description d'un modèle de composant avec ce profil peut contenir les moindres détails sur le comportement. L'utilisateur peut modéliser le comportement de son composant avec une machine à états finis. Néanmoins pour des architectures complexes, ceci peut causer un temps de modélisation supérieur au temps de codage à la main. Un environnement [58] a été conçu en vue de

la génération automatique du code à partir du modèle de haut niveau. Il contient trois transformations de modèles indépendantes, chacune ramène à un type de génération de code. La première transformation permet une génération squelette (*skeleton*) contenant seulement la structure statique du système. Ce type de génération est fréquemment adopté par les outils de transformation. La deuxième transformation offre une génération partielle intégrant une spécification plus complète du système. Le comportement peut être modélisé en utilisant une machine à états finis, un diagramme d'activités, etc. La dernière transformation aboutit à une génération totale du code SystemC. En comparant cet environnement avec Gaspard, ce dernier ne permet pas la génération du code d'un composant matériel ou logiciel mais plutôt favorise la réutilisation de ces composants à partir des bibliothèques.

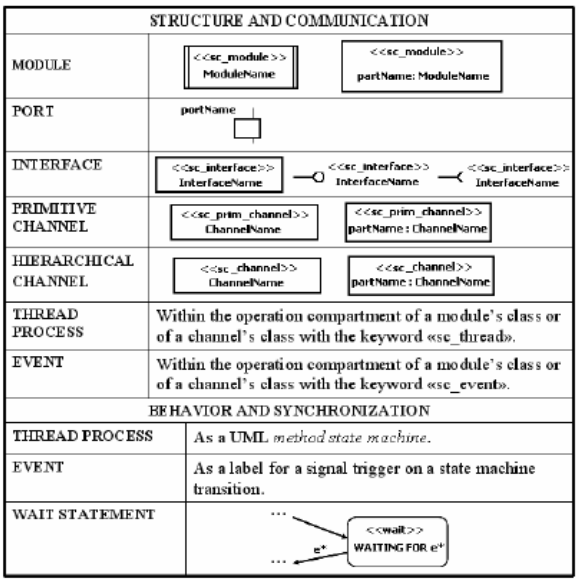


Figure 1.10: *Notation UML pour les concepts SystemC*

### 1.5.4 ROSES

L'environnement ROSES [12] permet une description hétérogène des systèmes MPSoC englobant plusieurs modèles de description et aussi plusieurs modèles de simulation: à des niveaux d'abstraction différents et/ou décrits en différents langages. Le cadre qui offre cette possibilité est l'utilisation du concept de l'architecture virtuelle. Cette dernière est constituée principalement de modules virtuels, ports virtuels et canaux virtuels. Le langage de description utilisé pour l'architecture virtuelle est VADeL permettent de décrire les systèmes d'une manière hétérogène. ROSES permet la co-simulation logicielle/matérielle avec plusieurs simulateurs en générant automatiquement les interfaces logicielles, matérielles et de co-simulation. Pour ce faire, les outils de génération automatique se servent des éléments prédéfinis des bibliothèques et des annotations du concepteur dans l'architecture virtuelle pour sélectionner les composants des bibliothèques (figure 1.11). ROSES fait un exemple d'environnement qui permet la génération du code de bas niveau (RTL) à partir d'une description de haut niveau. Les différents concepts de l'IDM sont utilisés de façon implicite dans ROSES. Ce dernier utilise un langage de description spécifique (textuel) tandis que la

recommandation de l'IDM est d'utiliser un langage visuel tel que UML. Les outils de génération de code jouent le rôle des transformations de modèles permettant de passer d'une description abstraite à une autre plus détaillée.

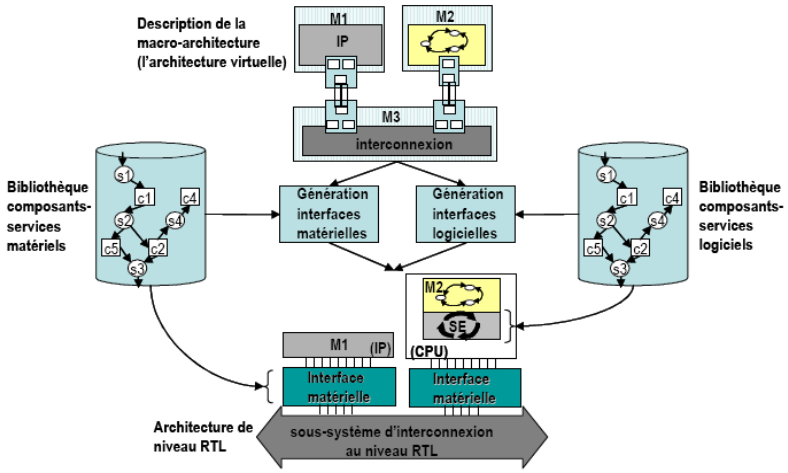


Figure 1.11: *Le flot de conception logiciel/matériel dans ROSES*

### 1.5.5 Le projet OMEGA

Le projet Européen OMEGA[2] s'inscrit dans le cadre d'une approche basée sur des modèles dédiés au développement des systèmes temps réel embarqués critiques. La modélisation est basée sur le langage UML. Pour exprimer les exigences spécifiques au domaine du temps réel, un profil UML a été défini [46]. Ce dernier représente un raffinement formel du profil UML standard Scheduling, Performance and Time (SPT). La boîte à outils IFx (figure 1.12) a été développée pour supporter l'utilisation de ce profil (dit profil OMEGA). Elle est utilisée en complément d'éditeurs UML compatibles avec le format d'échange XMI, tels que Rational Rose ou I-Logix Rhapsody. Le modèle UML est transformé ensuite en un modèle IF (langage spécifique), ce qui permet de simuler et de valider formellement des propriétés dynamiques des modèles. Pour que ce passage par IF soit transparent pour l'utilisateur, les fonctionnalités d'analyse et de vérifications ont été remontées au niveau de la spécification UML à travers une interface fournie par IFx. Ainsi, la connaissance de l'outil IF utilisé en dessous de l'interface n'est pas essentielle. L'inconvénient de cet environnement est qu'il ne supporte pas la description de l'architecture d'un système sur puce et reste restreint à un domaine spécifique d'application.

### 1.5.6 Autres propositions

D'autres environnements, basés sur une approche modèles, existent dans la littérature tels que Milan [41], Ptolemy [16], Artemis [55], Metropolis [2], GRACE++ [26], etc. Ils offrent une sémantique de description respectant un méta-modèle de référence, ensuite proposent des outils permettant l'analyse, la simulation ou la synthèse du système. Les différences

[2]Projet Européen OMEGA (IST-33522). Voir aussi http://www-omega.imag.fr

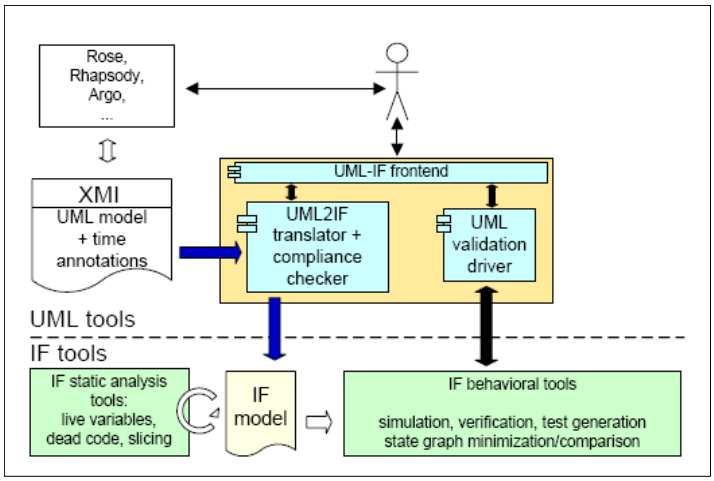


Figure 1.12: *Architecture de la boite à outils IFx*

entre ces environnements concernent le langage de spécification de haut niveau, le modèle de calcul, le type d'application envisagée et le niveau d'abstraction du code généré.

## 1.6 Conclusion

Dans ce chapitre, nous avons présenté une analyse détaillée des aspects qui seront étudiés dans cette thèse à savoir: la co-simulation, l'accélération de la simulation, l'estimation de performance, l'estimation de la consommation d'énergie et la méthodologie de conception. Au début, il semble difficile d'aborder ces différents aspects ensemble. Chacun d'entre eux représente un domaine de recherche nécessitant une étude plus approfondie. Néanmoins, l'analyse dans ce chapitre a montré une forte dépendance entre ces aspects. En effet, l'estimation de performance ne peut pas être étudiée indépendamment du type de co-simulation réalisée, l'estimation de la consommation fait recourt à la description du système et la méthodologie de conception regroupe les concepts et les outils développés par l'utilisateur. En conclusion tous ces aspects se réunissent ensemble pour réussir l'objectif final: résoudre la complexité de conception des MPSoC et réduire l'effort de développement. Dans le chapitre suivant, il s'avère nécessaire d'étudier de façon plus détaillée la description des MPSoC au niveau CABA afin de retirer les critères de précision qui peuvent être remontés vers des niveaux d'abstraction plus haut.

# Bibliography


[1] . SystemVerilog. `http://www.systemverilog.org/`, 2007.

[2] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *IEEE Computer*, 36(4), Apr. 2003.

[3] A. Bellaouar and M. I. Elmasry. *Low-Power Digital VLSI Design Circuits and Systems*. Kluwer Academic Publishers, 1995.

[4] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. Mparm: Exploring the multi-processor soc design space with systemc. *J. VLSI Signal Process. Syst.*, 41(2):169–182, 2005.

[5] M. V. Biesbrouck, L. Eeckhout, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *ISPASS'04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, Texas, USA, 2004.

[6] M. V. Biesbrouck, L. Eeckhout, and B. Calder. Considering all starting points for simultaneous multithreading simulation. In *ISPASS'06: Proceedings of the 2006 IEEE International Symposium on Performance Analysis of Systems and Software*, Texas, USA, 2006.

[7] M. Blagojevic. *SOI mixed mode design techniques and case studies*. PhD thesis, Ecole Polytechnique Fédérale de Laussane, 2005.

[8] L. Bonde. *Transformations de Modèles et Interopérabilité dans la Conception de Systèmes Hétérogènes sur Puce àă Base d'IP*. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Universitéde Lille 1, 2006.

[9] P. Boulet. Array-OL revisited, multidimensional intensive signal processing specification. Technical Report RR-6113, Feb. 2007.

[10] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual international symposium on Computer architecture*, pages 83–94, 2000.

[11] L. Cai and D. Gajski. Transaction level modeling: an overview. In *CODES+ISSS'03*, New York, USA.

[12] W. Cesário, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, and M. Diaz-Nava. Component-based design approach for multicore socs. In *DAC'02: Proceedings of the 39th conference on Design automation*, New Orleans, Louisiana, USA, 2002.

[13] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*, chapter 3. Kluwer Academic Publishers, 1995.

[14] COWARE. N2c. `http://www.coware.com/cowareN2C.html`.

[15] CoWare Inc, ConvergenSC. `http://www.coware.com/products`.

[16] U. de Berkeley. Ptolemy project. `http://ptolemy.eecs.berkeley.edu`.

[17] A. Demeure, A. Lafarge, E. Boutillon, D. Rozzonelli, J.-C. Dufourd, and J.-L. Marro. Array-OL : Proposition d'un formalisme tableau pour le traitement de signal multi-dimensionnel. In *Gretsi*, Juan-Les-Pins, France, Sept. 1995.

[18] A. Donlin. Transaction level: flows and use models. In *CODES+ISSS'04*, Stockholm, Sweden.

[19] M. Eva. *SSADM Version 4: A User's Guide*. McGraw-Hill Publishing Co, Apr. 1994.

[20] J.-M. Favre. Concepts fondamentaux de l'IDM. De l'ancienne egypte à l'ingénierie des langages. In *2èmes Journées sur l'Ingénierie Dirigée par les Modèles (IDM'06)*, Lille, France, 2006.

[21] F. Fummi, S. Martini, G. Perbellini, and M. Poncino. Native ISS-SystemC integration for the co-simulation of multi-processor SoC. In *Date'04: Proceedings of the conference on Design, automation and test in Europe*, Paris, France.

[22] D. D. Gajski and R. H. Kuhn. Guest editor introduction : New vlsi-tools. *IEEE Computer*, 16(12):11Ű14, 1983.

[23] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao. *SpecC:Specification Language and Methodology*. Kluwer, 2000.

[24] F. Ghenassia. *Tansaction level modeling with SystemC*. Springer, 2005.

[25] E. A. Gonzalez. BiCMOS technology processes, trends, and applications. Technical report, November 2004.

[26] GRACE++. *System Level Design Environment for Network-on-Chip (NoC) and Multi-Processor platform (MP-SoC) exploration*.

[27] M. Graphics. Lsim power analyst: Transistor-level simulation, 2004. `http://www.mentor.com/lsmpoweranalyst/datasheet.html`.

[28] M. Graphics. Seamless cve. `http://www.metorg.com/semless`.

[29] T. Grotker, S. Liao, and G. Martin. *System Design with SystemC*. Kluwer, 2003.

[30] J. Henkel. A low power hardware/software partitioning approach for core-based embedded systems. In *The 36th Design automation conference*, 1999.

[31] Intel. 3rd generation itanium : power budget. In *The 40th Design automation conference*, 2003.

[32] N. Julien, J. Laurent, E. Senn, and E. Martin. Power estimation of a c algorithm based on the functional-level power analysis of a digital signal processor. In *ISHPC '02: Proceedings of the 4th International Symposium on High Performance Computing*, London, UK, 2002.

[33] N. Julien, J. Laurent, E. Senn, and E. Martin. Power consumption modeling and characterization of the ti c6201. *IEEE Micro*, 23(5), 2003.

[34] D. Kim, S. Ha, and R. Gupta. Cats: cycle accurate transaction-driven simulation with multiple processor simulators. In *DATE'07: Proceedings of the conference on Design, automation and test in Europe*, Nice, France, 2007.

[35] D. Kim, Y. Yi, and S. Ha. Trace-driven HW/SW cosimulation using virtual synchronization technique. In *The 38th Conference on Design Automation*, Las Vegas, USA.

[36] M. Lajolo, A. Raghunathan, and S. Dey. Efficient power co-estimation techniques for system-on-chip design. In *DATE'00: Proceedings of the conference on Design, automation and test in Europe*, NY, USA, 2000.

[37] J. Laurent, N. Julien, E. Senn, and E. Martin. Functional level power analysis: An efficient approach for modeling the power consumption of complex processors. In *DATE'04: Proceedings of the conference on Design, automation and test in Europe*, DC, USA, 2004.

[38] Y. Li and J. Henkel. A framework for estimating and minimizing energy dissipation of embedded hw/sw systems. In *The 35th Design automation conference*, 1998.

[39] R. P. Llopis and K. Goossens. The petrol approach to high-level power estimation. In *ISLPED '98: Proceedings of the 1998 international symposium on Low power electronics and design*, CA, USA, 1998.

[40] D. Maliniak. Design languages vie for system-level dominance. Technical report, Oct. 2001.

[41] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *Conference on Languages, compilers and tools for embedded systems*, Berlin, Germany, 2002.

[42] G. Nicolescu. *Specification and validation for heterogeneous embedded systems*. PhD thesis, Laboratoire TIMA, UJF Grenoble, 2002.

[43] G. Nicolescu, K. Svarstad, W. O. Cesário, L. Gauthier, D. Lyonnard, S. Yoo, P. Coste, and A. A. Jerraya. Desiderata pour la spécification et la conception des systèmes électroniques. *Technique et Science Informatiques*, 21(3):291–314, 2002.

[44] K. Nose and T. Sakurai. Analysis and future trend of short-circuit power. *Transactions on Computer-Aided design of integrated circuits and systems*, 19(9):1023Ű1030, September 2000.

[45] S. Nussbaum and J. E. Smith. Statistical simulation of symmetric multiprocessor systems. In *35th Annual Simulation Symposium*, 2002.

[46] I. Ober, I. Ober, S. Graf, and D. Lesens. Projet OMEGA : un profil UML et un outil pour la modélisation et la validation de systèmes temps réel . 73:33–38, juin 2005.

[47] Object Management Group, Inc., editor. *UML 2 Infrastructure (Final Adopted Specifcation)*. `http://www.omg.org/cgi-bin/doc?ptc/2003-09-15`, Sept. 2003.

[48] Object Management Group, Inc., editor. *SysML v0.9*. `http://www.omg.org/cgi-bin/doc?ad/05-01-03`, Jan. 2005.

[49] Object Management Group, Inc., editor. *(UML) Profile for Schedulability, Performance, and Time Version 1.1*. `http://www.omg.org/technology/documents/formal/schedulability.htm`, Jan. 2005.

[50] U. of Berkeley (USA). Spice manual. `http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/`.

[51] Open SystemC Initiative. SystemC. `http://www.systemc.org/`, 2007.

[52] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Fast exploration of bus-based on-chip communication architectures. In *CODES+ISSS'04*, Stockholm, Sweden.

[53] Philips Electronic Design and Tools Group, Philips Research. DIESEL User Manual, version 2.5 edition, June 2001.

[54] C. Piguet. *Low-Power Electronics Design*. CRC Press, 2004.

[55] A. D. Pimentel. The artemis workbench for system-level performance evaluation of embedded systems. *Journal of Embedded Systems*, 1(7), 2005.

[56] ProMarte partners. UML Profile for MARTE, Beta 1. `http://www.omg.org/cgi-bin/doc?ptc/2007-08-04`, Aug. 2007.

[57] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A uml 2.0 profile for systemc: toward high-level soc design. In *EMSOFT'05: Proceedings of the 5th ACM international conference on Embedded software*, NJ, USA, 2005.

[58] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A model-driven design environment for embedded systems. In *DAC'06: Proceedings of the 43rd annual conference on Design automation*, CA, USA, 2006.

[59] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. *Proceedings of the IEEE*, 91(2):305Ű327, February 2003.

[60] C. V. Schimpfe, S. Simon, and J. A. Nossek. High-level circuit modeling for power estimation. In *6th IEEE ICECS, International Conference on Electronics, Circuits and Systems*, 1999.

[61] E. Senn, J. Laurent, N. Julien, and E. Martin. Softexplorer: estimation, characterization and optimization of the power and energy consumption at the algorithmic level. In *IEEE PATMOS*, Santorin, Grèce, 2004.

[62] A. Sinha and A. P. Chandrakasan. Jouletrack: a web based tool for software energy profiling. In *The 38th conference on Design automation*, NY, USA, 2001. ACM.

[63] SpecC Technology Open Consortium. SpecC. `http://www.specc.gr.jp/eng/index.html`, 2007.

[64] Synopsys. NanoSim datasheet (timemill and powermill) : Memory and mixed signal verification. `http://www.synopsys.com/products/mixedsignal/nanosim/nanosim_ds.pdf`.

[65] Synopsys. Power-Gate(TM) : a dynamic, simulation-based, gate-level power analysis tool. `Synopsys,http://www.synopsys.com/news/pubs/rsvp/fall97/rsvp_fall97_5.html`.

[66] H. Tardieu, A. Rochfeld, and R. Colletti. *La Méthode Merise : Principes et outils*. Editions d'Organisation, 1991.

[67] J. Tayeb and S. Niar. Adapting the epic register stack for an efficient execution of forth. In *The 22nd EuroForth Conference (EuroForth 2006)*, Cambridge, England.

[68] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. In *Transactions on VLSI Systems*, 1994.

[69] A. Turier. *Etude, conception et caractérisation des mémoires CMOS, faible consommation, faible tension en technologies submicroniques*. PhD thesis, Université Pierre et Marie Curie, Paris, France, 2000.

[70] E. Viaud, F. Pecheux, and A. Greiner. An efficient TLM/T modeling and simulation environment based on parallel discrete event principles. In *DATE'06: Proceedings of the conference on Design, automation and test in Europe*, Munich, Germany.

[71] K. Wakabayashi and T. Okamoto. C-based soc design flow and eda tools: An asic and system vendor perspective. *IEEE Computer on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1507Ű1522, 2000.

[72] WEST Team LIFL, Lille, France. Graphical Array Specification for Parallel and Distributed Computing (GASPARD-2). `http://www.lifl.fr/west/gaspard/`, 2005.

[73] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. Irwin. The Design and Use of SimplePower: A Cycle Accurate Energy Estimation Tool. In *Design Automation Conf*, June 2000.