Chapter 1

Simulation et estimation de performance des MPSoC au niveau PVT

1.1 Introduction

Dans le chapitre précédent, la problématique du temps de simulation pour évaluer les performances des MPSoC au niveau CABA a été présentée. Nous avons ainsi montré que les temps de simulation obtenus à ce niveau ne sont pas satisfaisants pour une évaluation rapide de solutions architecturales. Cette lenteur de simulation est due principalement à la quantité importante de détails architecturaux que le simulateur au cycle prêt doit prendre en compte à chaque intervalle de temps. Dans ce chapitre, nous proposons une solution qui consiste à décrire le système au niveau transactionnel (TLM), plus spécifiquement au niveau PVT. Le chapitre précédent a montré l'intérêt de l'utilisation de ce niveau pour la conception des systèmes embarqués. Y compris à ce niveau d'abstraction PVT, la résolution de la complexité de conception des systèmes MPSoC nécessite une stratégie d'exploration multi-niveaux. A chaque niveau, un ensemble de solutions pourront être éliminées de l'espace de solutions architecturales. Pour cela, nous allons définir un environnement de simulation composé de 3 sous-niveaux. Ces derniers représentent en quelque sorte une décomposition du niveau PVT. Ainsi, dans notre plateforme le niveau PVT se décline en 3 sous-niveaux "pattern-accurate" noté PVT-PA, "transaction-accurate" noté PVT-TA, et enfin "event-accurate" noté PVT-EA. Ces 3 sous-niveaux sont présentés dans ce chapitre en partant du niveau d'abstraction le plus élevé (PA) vers le niveau d'abstraction le moins élevé (EA). A ce niveau l'architecture est définie de façon beaucoup plus détaillée que PA. L'écart d'abstraction entre PA et EA nous a ramené à définir un niveau intermédiaire TA.

Dans ce chapitre notre évaluation est basée sur le critère d'estimation de performance (temps d'exécution de l'application) ce qui justifie notre développement d'un modèle de temps pour chaque sous-niveau. Pour implémenter une famille de MPSoC, nous avons modélisé plusieurs composants matériels tels que le processeur, la mémoire cache, le réseau d'interconnexion, accélérateur matériel, etc. Ces composants seront décrits pour chacun des sous-niveaux. Les résultats expérimentaux montrent une accélération de simulation et une erreur relative à la précision de description de chaque sous-niveau. A partir de cette simulation nous proposons une méthode de conception de composant à ces niveaux d'abstraction

pour permettre une intégration dans l'environnement de co-design Gaspard.

Ce chapitre est structuré comme suit. La deuxième section présentera un justificatif de l'utilisation du niveau PVT pour la simulation et l'évaluation des systèmes MPSoC. La section 3 détaillera les trois sous-niveaux décrits au niveau PVT utiles pour une exploration multi-niveaux de l'espace de solutions architecturales. La description d'un système MPSoC aux sous-niveaux PVT-PA, PVT-EA et PVT-TA sera détaillée respectivement dans les sections 4, 5 et 6. La section 7 présentera les modèles de différents composants matériels développés pour implémenter l'environnement PVT proposé. Des modèles d'estimation de temps d'exécution sont proposés dans la section 8 afin d'évaluer les performances dans les 3 sous-niveaux PVT-PA, PVT-EA et PVT-TA. L'efficacité de notre proposition sera illustrée à travers l'exécution parallèle de l'application codeur H.263 sur un exemple paramétré système MPSoC. La section 9 présentera les résultats de simulation qui comparent la description d'un système MPSoC décrit avec l'environnement PVT (avec ces 3 sous-niveaux) et le niveau CABA. Les critères de comparaison que nous avons choisis sont : le temps de simulation, la précision dans l'estimation des performances et l'effort à réaliser afin de développer les modèles dans chacun des 3 sous-niveaux de PVT. La section 10 présente une synthèse de ce chapitre et donne un aperçu sur ses généralisations.

1.2 Proposition et justification

Pour réduire le temps de simulation des systèmes embarqués et en particulier ceux qui intègrent plusieurs processeurs, il faut identifier les détails de la micro-architecture du niveau CABA qui peuvent être omis pour alléger la simulation. Parmi ces détails, nous avons identifié ceux qui sont reliés à la partie calcul (ou traitement) et ceux liés à la partie communication. Concernant la partie calcul, les machines d'états finis (Transition, Mealy et Moore) décrivant le composant représentent le facteur prépondérant dans les temps de simulation au niveau CABA. Pour la partie communication, nous avons remarqué que les machines d'états finis qui décrivent l'interface de chaque composant ralentissent aussi la simulation. Notre solution tend vers un compromis équitable entre précision et temps de simulation. Nous proposons à un niveau d'abstraction plus élevé la prise en compte d'indicateurs déduits d'une simulation au niveau CABA. Cette abstraction doit être suffisante pour pouvoir vérifier le comportement de notre système (application déployée sur l'architecture) et permettre aussi l'estimation du temps d'exécution d'une façon assez précise.

Notre étude de cas est construite autour d'une famille MPSoC sur laquelle une application sera déployée. A partir de ce déploiement sur cette architecture, nous évaluons les performances de notre système ce qui nous permet d'extraire la solution la plus adéquate. A chaque niveau d'abstraction correspond une précision d'évaluation de performance plus au moins coûteuse en temps de simulation. Ainsi le niveau PVT permet d'extraire les solutions pertinentes dans un temps raisonnable tout en offrant un niveau de précision relativement intéressant. Dans nos trois sous-niveaux PVT, certains détails d'implémentation visibles au niveau CABA disparaissent au profit de fonctions d'évaluation basées sur des résultats de simulation au niveau CABA.

Dans ce chapitre, nous nous limitons au critère de performance (temps d'exécution) pour comparer les différentes solutions architecturales. Au chapitre suivant un deuxième critère sera introduit concernant la consommation de puissance.

1.3 Description des MPSoC au niveau PVT

Pour réduire le temps de simulation des systèmes MPSoC, notre solution consiste à décrire le système au niveau PVT. La précision de la description à ce niveau pour les deux parties calcul et traitement dépend des objectifs attendus du système MPSoC à simuler. Dans notre travail nous avons fixé les objectifs suivants:

- La vérification fonctionnelle du système qui consiste à exécuter l'application cible sur une architecture définie et s'assurer de l'exactitude du résultat de simulation.
- Une analyse de performance du système sans trop pénaliser le temps de simulation. En effet, cette analyse consiste à définir un modèle de temps et à l'intégrer dans le niveau d'abstraction cible.
- La vérification du déploiement des tâches de l'application sur les processeurs et des données sur les bancs mémoires. L'estimation du temps d'exécution nous permet de savoir pour chaque tâche si celle-ci est exécutée dans les délais déterminés.
- La mesure des contentions sur le réseau d'interconnexion afin de vérifier l'adéquation du support de communication avec le système. Elle permet de détecter les goulots d'étranglement vers une cible particulière comme les bancs mémoires.
- L'analyse de la consommation d'énergie du système en vue de l'exploration architecturale.
- La diminution de l'effort de développement par rapport aux niveaux les plus bas. Dans notre étude, nous allons nous baser sur le critère *nombre de lignes de code* (LOC) développées pour comparer la modélisation à différents niveaux d'abstraction.

Le nombre important d'objectifs auquel s'ajoute le large espace de solutions architecturales nécessitent une stratégie d'exploration rapide. Les propositions faites dans ce chapitre doivent permettre une approche multi-niveaux ce qui faciliterait l'élimination d'alternatives à chaque niveau jusqu'à converger vers la solution la plus adéquate en bas niveau. Pour atteindre les objectifs précédemment définis, nous avons développé plusieurs sous-niveaux à l'intérieur du niveau PVT à savoir le PVT-PA (*PVT Pattern Accurate*), PVT-TA (*PVT Transaction Accurate*) et le PVT-EA (*PVT Event Accurate*). En effet, certains objectifs ne demandent pas un niveau de détail élevé. Une simulation rapide à un niveau d'abstraction élevé est dans ce cas suffisante.

Chaque sous-niveau propose un compromis accélération/précision différent. Les trois sous-niveaux ont en commun les caractéristiques du niveau PVT à savoir: une architecture matérielle définie et des outils de mesure des performances et de la consommation d'énergie. Néanmoins, chaque sous-niveau se distingue des autres sous-niveaux par les points suivants:

- le sous-niveau PVT-PA permet une exécution fonctionnelle de la partie calcul, donc il offre le facteur d'accélération de la simulation le plus élevé.
- Le sous-niveau PVT-TA utilise un simulateur au niveau d'instructions pour exécuter l'application, il simule fonctionnellement le comportement de la micro-architecture

et ne prend pas en considération les spécifications du protocole de communication. Le sous-niveau PVT-TA présente le meilleur compromis accélération de la simulation/précision.

- Le sous-niveau PVT-EA tient compte des délais entre les événements internes de l'architecture (exemple: exécution d'une instruction, accès en lecture mémoire, etc.) et implémente un protocole de communication ce qui lui permet d'atteindre le maximum de précision. Dans ce qui suit, nous allons détailler la description de ces trois sous-niveaux.

1.4 Le PVT Pattern Accurate (PVT-PA)

Nous avons défini deux objectifs pour le sous-niveau PVT-PA. En effet, ce sous-niveau doit d'un côté permettre au développeur la vérification fonctionnelle du système et de l'autre côté il doit rendre possible l'observation des contentions sur le réseau d'interconnexion et la récupération des informations temporelles correspondantes. Ce deuxième objectif représente la majeure différence entre PVT-PA est le niveau PV qui n'inclut pas des spécifications temporelles. Les résultats de simulation à ce sous-niveau vont permettre au développeur de prendre des décisions sur l'adéquation du réseau d'interconnexion et du flux de communication de l'application. Il en est de même pour le déploiement des tâches sur les processeurs et le placement des données sur les bancs mémoires.

1.4.1 Structure de données

La proposition d'un sous-niveau PVT-PA est justifiée par le domaine d'application, à savoir le traitement de données intensif. Dans ce contexte le nombre d'accès à la mémoire d'instructions est négligeable par apport au nombre d'accès aux mémoires de données. Pour cela, la partie calcul est exécutée à un niveau d'abstraction assez haut. A titre d'exemple, les processeurs ne sont pas décrits à travers des ISS (*Instruction Set Simulator*) mais réalisent plutôt l'ensemble de tâches de l'application. Ces tâches seront ensuite exécutées par la machine de simulation. Néanmoins, nous allons simuler les accès aux mémoires de données au niveau de l'architecture. La figure 1.1 illustre la simulation logicielle/matérielle au niveau PVT-PA.

La structure de données traitée par les tâches est nommée *Motif* (*Pattern*). Un motif peut être une variable simple ou bien un tableau à dimension variable. A titre d'exemple, pour une application de multiplication de deux matrices 32x32, le motif de données traité peut être un tableau à deux dimensions (32x32) ou bien un vecteur de dimension (32). Avant l'exécution d'une tâche, une phase de lecture des motifs d'entrées à partir de la mémoire est réalisée. De même après l'exécution de la tâche, un transfert des données résultats dans l'autre sens pour stocker les motifs de sorties est effectué. En comparaison au niveau CABA, le sous-niveau PVT-PA permet de simuler les mêmes tailles de transfert de données. D'où notre appellation PVT "précis au niveau motif" ou bien PVT Pattern Accurate (PVT-PA).

1.4.2 Synchronisation des tâches et ordonnancement

Pour assurer la cohérence des données lors de l'exécution de l'application, un mécanisme de synchronisation entre les tâches est nécessaire. Ce mécanisme est nécessaire pour vérifier

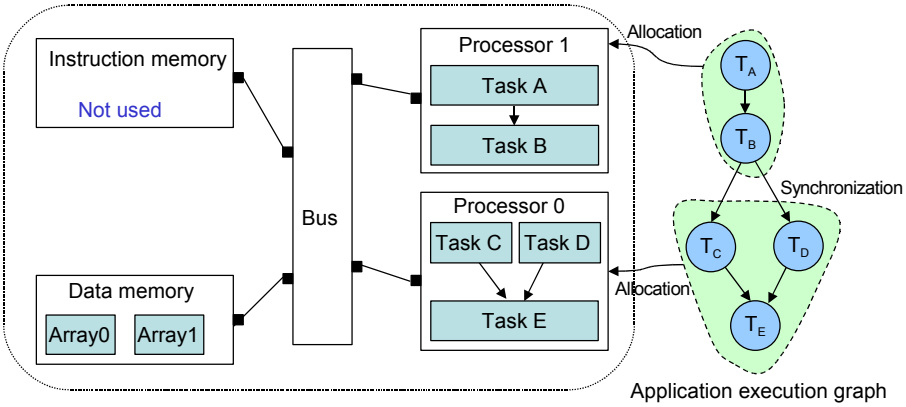


Figure 1.1: *Simulation logicielle/matérielle au sous-niveau PVT-PA*

la cohérence de données entre deux tâches successives exécutées sur des processeurs différents. Dans notre exemple de la figure 1.1, le mécanisme de synchronisation vérifie que la tâche C ou D du processeur 0 n'est exécutée qu'après la fin d'écriture du motif issue de la tâche B du processeur 1. Dans ce cas la tâche B est appelée producteur et les tâches C et D sont appelées consommateurs. Le mécanisme de synchronisation doit pouvoir gérer le cas de plusieurs producteurs et plusieurs consommateurs. C'est le cas si les tâches B, C ou D sont parallélisées sur plusieurs processeurs. Cette synchronisation des tâches n'est utile que lorsque les tâches sont placées sur des processeurs différents. Autrement, lorsque les tâches sont exécutées sur le même processeur, il est possible de les ordonnancer statiquement selon un ordre respectant les dépendances de données.

Le mécanisme que nous avons implémenté peut être décrit comme suit: au début tous les processeurs de type consommateur sont bloqués. Pour chaque tâche de type producteur, lorsqu'elle est terminée elle indique la disponibilité du motif qu'elle doit fournir. Lorsque toutes les tâches producteurs ont indiqué la disponibilité de leurs motifs, chacune des connexions de sortie reçoit l'autorisation de débloquer les lecteurs. Une fois qu'une tâche consommateur a toutes ses connexions débloquées, elle commence à s'exécuter. Pendant ce temps-là, les tâches qui ont produit les motifs sont bloquées afin de ne pas récrire par-dessus les motifs qui sont en cours de lecture. Les variables de synchronisation sont déclarées au niveau du simulateur SystemC sans exécuter réellement les accès mémoires correspondants. Ceci nous permet d'accélérer la simulation au sous-niveau PVT-PA.

Pour traiter l'ordonnancement des tâches d'un même processeur, c'est le cas des tâches C et D du processeur 0, plusieurs possibilités existent. La solution la plus simple est d'effectuer un ordonnancement statique séquentiel en respectant le graphe de dépendance entre les tâches. Cette solution est simple à implémenter, mais ne permet pas d'obtenir le maximum de performance de l'application. En effet, nous ne pouvons pas savoir à l'avance quel motif de la tâche sera disponible en premier. Ceci ne dépend pas seulement du temps d'exécution de la tâche, mais dépend aussi des contentions sur le réseau d'interconnexion. Donc avec un ordonnancement statique, nous pouvons nous retrouver dans le cas où une première tâche attend qu'un motif soit disponible alors que nous avons déjà une deuxième tâche qui peut être exécutée. Pour remédier à cet inconvénient, la solution consiste à réaliser un ordonnancement dynamique. Seule cette dernière solution, a été implémentée comme ordonnanceur de notre simulateur. En résumé, la partie calcul au sous-niveau PVT-PA est

structurée comme suit:

- Réaliser une synchronisation au début de la tâche avec les autres tâches exécutées sur des processeurs différents.
- Lire les motifs d'entrées à partir de la mémoire pour l'exécution de chaque tâche.
- Exécuter une ou plusieurs tâches.
- Écrire des motifs de sorties dans la mémoire après l'exécution de chaque tâche.
- Réaliser une synchronisation à la fin de la tâche avec les autres tâches exécutées sur des processeurs différents.

Au sous-niveau PVT-PA, cette description est intégrée dans le composant processeur en utilisant le processus SC_THREAD de SystemC. Ce choix est dû principalement à la nécessité d'utiliser des instructions *wait()* dans notre description pour différentes raisons. En premier lieu, nous utilisons les instructions *wait()* lorsque toutes les tâches d'un processeur sont en attente des motifs d'entrées. A ce moment, l'ordonnanceur a besoin de vérifier s'il y a une tâche prête à être exécutée à chaque période de temps, comme nous allons le décrire dans la section 1.8.1. En second lieu, nous avons besoin d'utiliser les instructions *wait()* pour les intégrer dans le modèle de temps pour l'estimation de performance.

Le texte ci-dessous est un exemple de thread pour implémenter la fonctionnalité de la partie calcul décrite au sous-niveau PVT-PA. Cet exemple concerne la description du processeur 0 de la figure 1.1.

```
void MIPSCore::run()
{
//Tâche_C
 void task_C (struct tlmpa_task* a)
 {
// Déclaration des tableaux correspondant aux motifs de la tâche.
   float Tab_in[16]; float Tab_out[16];
// Synchronisation avec les autres tâches exécutées sur d'autres processeurs
   sync_1.startReading(B);
// Lecture du motif d'entrée de la tâche C
   for ( int i=0 ; i<=15 ; i++ ){
   Tab_in[i]=Port.read(addr+i, data, t);
   Wait(Transaction_delay,NS)
   }
// Exécution de la Tâche_C
   Tâche_C(Tab_in, Tab_out );
   Wait(Task_delay,NS)
// Ecriture du motif de sortie de la tâche C
   for ( int i=0 ; i<=15 ; i++){
   Port.write(addr+i, Tab_out[i], t);
   Wait(Transaction_delay,NS)
   }
// Synchronisation avec les autres tâches exécutées sur d'autres processeurs
```

```
   sync_1.finishWriting(B);
 ...}

  void task_D (struct tlmpa_task* a)
  {
  ...
  }
  void task_E (struct tlmpa_task* a)
  {
  ...
  }
}
```

1.4.3 Les communications

Au sous-niveau PVT-PA, la communication entre les composants est décrite à un niveau plus haut que le niveau CABA. Les transactions sont réalisées à travers des canaux de communication (appelés aussi *Channels*) au lieu des signaux comme ceci est réalisé au niveau CABA. Les canaux de communication implémentent une ou plusieurs interfaces. Chaque interface est définie par un ensemble de méthodes *read()* et *write()*. Pour le chargement (*load*) et le stockage (*store*) des données, les initiateurs (processeurs, contrôleur DMA, etc.) instancient des appels de fonctions *read()* et *write()* qui seront transmis via le port. A titre d'exemple dans le texte ci-dessus nous utilisons les deux fonctions suivantes: *Port.read(addr+i, data, t)* et *Port.write(addr+i, Tab_out[i], t)* où le premier paramètre présente l'adresse de la donnée, le deuxième paramètre présente la variable qui reçoit la donnée et le dernier paramètre présente le temps mis par la transaction. Au niveau des cibles (mémoire, accélérateur matériel, etc.), les transactions seront récupérées pour exécuter la méthode correspondante et envoyer la réponse à l'initiateur. A ce sous-niveau, un modèle de temps a été défini pour estimer le temps d'exécution de l'application. Ce modèle sera détaillé dans la section 1.8.1.

En résumé, le sous-niveau PVT-PA présente l'avantage d'un gain important en terme d'accélération de la simulation. Ceci est dû principalement au fait que nous avons remplacé l'implémentation de l'architecture des processeurs du système par le code de l'application. Cela permet de réaliser une simulation logicielle/matérielle même si le composant processeur n'est pas disponible. Des résultats préliminaires peuvent être obtenus assez tôt dans le processus de développement ce qui peut réduire le temps de développement. En plus, lorsque nous exécutons les tâches de l'application directement sur la machine de simulation, il est possible d'utiliser les outils conventionnels de débogage qui sont généralement maîtrisés par les développeurs pour la vérification fonctionnelle. En contre-partie de ces avantages, nous pensons que le transfert de données au niveau des motifs ne reflète pas ce qui se passe réellement dans le réseau d'interconnexion au niveau des transactions ce qui altère l'estimation de performance au sous-niveau PVT-PA. Le choix d'un motif à granularité fine peut être une solution pour remédier à cette source d'erreur. Mais ceci nous ramène à une décomposition fine des tâches de l'application ce qui va surcharger la simulation et contredit le premier objectif défini pour le sous-niveau PVT-PA.

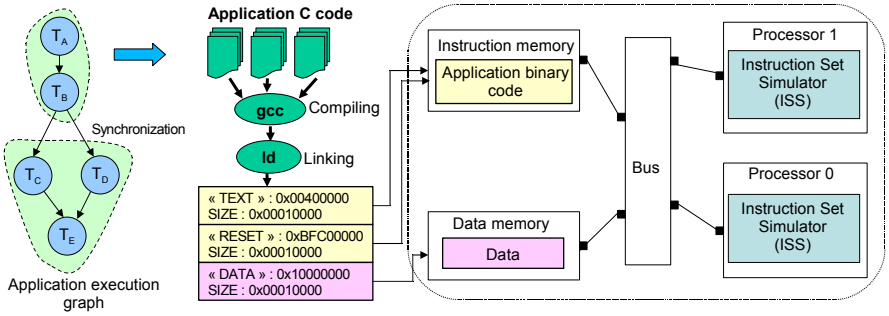


Figure 1.2: *Simulation logicielle/matérielle au sous-niveau PVT-TA*

1.5 Le PVT Transaction Accurate (PVT-TA)

L'objectif de ce sous-niveau est d'obtenir des estimations de performance plus précises que celles obtenues par le sous-niveau PVT-PA. Les solutions architecturales retenues du sous-niveau PVT-PA par le développeur peuvent être explorées une deuxième fois au sous-niveau PVT-TA afin d'obtenir des estimations plus précises par la prise en compte du flux d'instructions. Par rapport au sous-niveau PVT-PA, notre idée majeure est d'utiliser des simulateurs de processeurs au niveau des instructions (ISS) pour l'exécution de l'application afin d'augmenter la précision d'estimation de performance. Les tâches de l'application sont compilées afin d'obtenir un code binaire qui sera ensuite exécuté à partir de la mémoire d'instructions. La figure 1.2 illustre la simulation logicielle/matérielle au niveau PVT-TA. Avant de commencer la simulation, nous initialisons les différents segments de données et d'instructions dans les mémoires appropriées. En plus, pour chaque processeur nous définissons l'adresse du début de programme de chaque tâche à exécuter.

1.5.1 Structure de données

Au niveau PVT-TA, l'ISS qui simule la fonctionnalité d'un processeur sera exécuté sur la machine de simulation. Il modélise le processeur cible au niveau du jeu d'instructions. Pendant la simulation, l'application sera exécutée instruction par instruction en suivant ces 4 étapes: la lecture du code de l'opération à partir de la mémoire d'instructions, le décodage, la lecture des opérandes à partir de la mémoire de données et l'exécution de l'instruction. Pour supporter les ISS au sous-niveau PVT-TA dans un environnement de simulation MPSoC, cette fonctionnalité est décrite dans un module SystemC en utilisant le processus SC_THREAD. Ce choix est dû principalement à la nécessité d'utiliser des instructions *wait()* dans notre description pour deux raisons. La première est liée à notre choix d'utiliser des canaux de communication de type "bloquant" pour connecter les différents composants (plus de détails dans la section 1.7.1). Autrement dit, nous utilisons des instructions *wait()* sensitives sur des événements (dans notre cas c'est l'arrivée des réponses) ce qui permet au noyau SystemC d'ordonnancer les différents processeurs au niveau de chaque instruction. Ceci a comme avantage d'éviter qu'un processeur avance dans le temps sans que les autres le fassent. La deuxième raison est que nous avons besoin d'utiliser les instructions *wait()* dans le modèle de temps pour l'estimation de performance comme il sera détaillé dans la section 1.8.2.

Le texte ci-dessous est un exemple de thread pour implémenter la fonctionnalité de la partie calcul (processeur) décrite au sous-niveau PVT-TA.

```
void MIPSCore::run()
{
// Boucle d'exécution
  while(next_pc!=0)
  {
// Lecture du code de l'opération à partir du cache d'instructions
  initiator_ins_port.read( next_pc , instruction , time );
  Wait(Transaction_delay,NS)
// Décodage des instructions
  IDecode(instruction, ins_opcode);
// Execution des instructions
  switch (ins_opcode) {
    case OP_ADD:
    {...
    Wait(ADD_delay,NS)
    }
    break;
    case OP_BEQ:
    {...
    Wait(BEQ_delay,NS)
    }
    break;
// Cas de chargement de données
    case OP_LB:
    {...
     initiator_data_port.read( adr , d , time );
     Wait(Transaction_delay,NS)
    }
    break;
// Cas de stockage de données
    case OP_SB:
    {...
     initiator_data_port.write( adr , d , time );
     Wait(Transaction_delay,NS)
    }
    break;
    ...
}
```

Dans ce texte nous identifions les étapes nécessaires pour l'exécution de l'application au niveau des instructions. Le processeur commence par une phase de lecture de la prochaine instruction à partir du cache d'instructions en initialisant une requête *initiator_ins_port.read(next_pc, instruction, time)*. Le paramètre *next_pc* spécifie l'adresse contenue dans le registre compteur programme, *instruction* représente la variable dans la quelle nous devons recevoir l'instruction et le paramètre *time* est utilisé pour l'estimation du temps d'exécution comme il sera détaillé dans la section 1.8.2. La deuxième phase consiste à décoder l'instruction pour identifier le type de l'opération via la fonction *IDecode(instruction,*

ins_opcode). La phase suivante est la lecture des opérandes à partir de la mémoire si c'est nécessaire. La dernière phase consiste en l'exécution de l'instruction courante et la mise à jour des registres du processeur et du compteur programme.

La description au sous-niveau PVT-TA produit le même comportement du point de vue des transactions que le niveau CABA, d'où notre appellation PVT "précis au niveau transaction" ou bien PVT Transaction Accurate (PVT-TA).

1.5.2 Synchronisation des tâches et ordonnancement

Au sous-niveau PVT-TA, pour assurer la cohérence des données entre plusieurs tâches exécutées sur différents processeurs nous avons implémenté un mécanisme de synchronisation. Dans ce mécanisme, les processeurs partagent des variables de synchronisation qui peuvent être lues et écrites seulement à partir de la mémoire de données partagée et qui ne sont pas mises dans les caches. D'une autre façon, ces variables peuvent être vues comme des verrous au niveau des tâches qui nécessitent de la synchronisation. C'est le cas des tâches B et C ou B et D dans la figure 1.1. L'avantage de ce mécanisme est qu'il peut être utilisé pour différentes topologies de réseaux d'interconnexion. Néanmoins, la synchronisation avec des variables partagées ne permet pas d'obtenir le maximum des performances en comparaison avec des solutions liées à des types de réseaux d'interconnexion particuliers. C'est le cas du mécanisme d'espionnage *snoop* utilisé pour le bus.

Au sous-niveau PVT-TA, la simulation des accès mémoires des variables de synchronisation va nous permettre d'éviter l'utilisation de l'ordonnanceur de tâches. En effet, une tâche synchronisée avec d'autres tâches sera exécutée lorsque son verrou est débloqué. Lorsque plusieurs tâches peuvent être ordonnancées en même temps, celle qui va avoir en premier son verrou débloqué sera exécutée. Ceci au sous-niveau PVT-PA n'était pas possible, car les variables de synchronisation sont déclarées au niveau du simulateur SystemC donc au point de vue architecture nous ne pouvons pas savoir leur instant de déblocage contrairement au sous-niveau PVT-TA.

1.5.3 Les communications

La communication entre les composants au sous-niveau PVT-TA est la même que celle décrite au sous-niveau PVT-PA. Comme au sous-niveau PVT-PA, les spécifications du protocole de communication ne sont pas définies. Les événements internes de l'architecture se produisent successivement sans respecter les délais entre eux ce qui présente une source d'erreur pour l'estimation de performance.

En résumé le sous-niveau PVT-TA intègre des simulateurs de processeurs au niveau des instructions (ISS) pour l'exécution de l'application. Il permet de visualiser les mêmes transactions que celles simulées au niveau CABA ce qui minimise l'erreur d'estimation de performance. Néanmoins, ceci est obtenu au prix d'une augmentation du temps de simulation. En effet, à l'exception du composant processeur, la description des autres composants aux sous-niveaux PVT-PA et PVT-TA est identique. De ce fait, la vitesse de simulation dans ces deux sous-niveaux est semblable. Le passage du sous-niveau PVT-PA au PVT-TA peut se faire avec une étape de raffinement automatique en compilant les tâches de chaque processeur avec le compilateur approprié. Il nécessite néanmoins la disponibilité du même composant processeur à deux niveaux d'abstraction différents. Les segments du code binaire générés seront ensuite placés dans la mémoire d'instructions. Nous remarquons que l'utilisation des

ISS rend difficile l'intégration des outils de débogage au cours de la simulation puisque le jeu d'instructions du processeur cible est à priori différent de celui de la machine de simulation.

1.6 Le PVT Event Accurate (PVT-EA)

L'objectif du sous-niveau PVT-EA est de pouvoir atteindre des estimations de performance très proches de celle du niveau CABA. Pour ce faire, nous avons identifié les informations intéressantes à ajouter au sous-niveau PVT-TA concernant les deux parties calcul et communication pour augmenter la précision sans pénaliser de façon importante le temps de simulation. Parmi les détails de la micro-architecture, la prise en compte des spécifications du protocole de communication et le respect des délais entre les événements ont montré expérimentalement les conséquences sur l'amélioration de la précision d'estimation.

En effet, dans un système MPSoC plusieurs tâches sont exécutées simultanément. Pour assurer une exécution dans l'ordre des événements entre les processeurs, il est nécessaire de prendre en considération les délais d'exécution correspondants (exécution des instructions, préparation des requêtes commande et réponse, transmission des requêtes via le réseau d'interconnexion, etc.). Cette attention aux délais d'exécution permet d'établir le même séquencement d'événements que celui obtenu par simulation au niveau CABA, d'où notre appellation PVT "précis au niveau événements" ou bien PVT Event Accurate (PVT-EA). Néanmoins, certaines activités peuvent avoir un délai d'exécution imprévisible. Comme exemple, nous citons le cas de l'exécution désordonnée des instructions dans un processeur superscalaire. Le temps de terminaison d'un tel type d'événement étant imprévisible, cela constituera une source d'erreur possible pour l'estimation de performance. Au sous-niveau PVT-EA, la structure de données traitées, la synchronisation et l'ordonnancement des tâches se présentent de la même façon qu'au sous-niveau PVT-TA.

1.6.1 Les communications

Au sous-niveau PVT-EA, les spécifications du protocole de communication utilisé doivent être précisées par l'utilisateur ainsi que les délais d'exécution correspondants. Les délais de la micro-architecture, comme le temps de réponse du réseau d'interconnexion, doivent être précisés. Ces délais peuvent être estimés par l'utilisateur ou bien mesurés à un bas niveau comme cela sera détaillé dans la section 1.8.3. Cette opération est très importante pour obtenir un comportement réaliste de la partie communication en comparaison avec le niveau CABA. La figure 1.3 illustre le séquencement de deux transactions de tailles différentes. La transaction 1 contient un seul mot alors que la deuxième contient trois mots. L'instant auquel chaque transaction arrive au réseau d'interconnexion intervient pour définir l'ordre des accès aux ressources partagées vis-à-vis des autres transactions venant d'autres processeurs. Ceci permet d'obtenir un séquencement d'événements réaliste et d'offrir le maximum de précision. Le défi au sous-niveau PVT-EA est de détecter les instants auxquels se produisent les transactions et ceux auxquels elles arrivent au réseau d'interconnexion. Or ceci ne peut se faire qu'en prenant compte du nombre de mots dans la requête, du temps de préparation du paquet et des contentions sur le réseau d'interconnexion.

L'avantage majeur du sous-niveau PVT-EA est qu'il permet de prendre en considération les spécifications du protocole de communication au plus tôt dans le processus de développement (avant le niveau CABA). Il offre aussi la possibilité d'estimer le temps

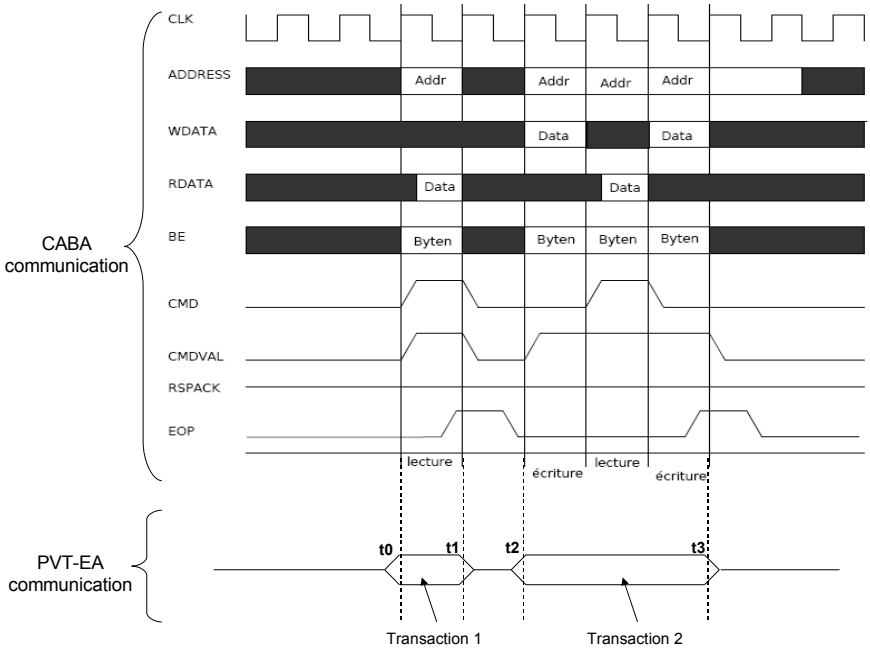


Figure 1.3: *Communications aux niveaux CABA et PVT-EA*

d'exécution d'une façon plus précise que PVT-PA et PVT-TA. En contrepartie, PVT-EA génère une accélération de simulation plus faible que les deux autres sous-niveaux. Le passage du sous-niveau PVT-TA au sous-niveau PVT-EA peut se réaliser avec une étape de raffinement en ajoutant les détails du protocole et de temps à la description du système MPSoC au sous-niveau PVT-TA. La prise en compte de ces détails correspond à insérer des instructions *wait()* avec les arguments appropriés dans la description de l'architecture du système.

1.7 Modèles de composants pour la conception des MPSoC

Dans cette section nous allons présenter les modèles de composants qui ont été développés aux 3 sous-niveaux. Ces composants sont génériques et nous permettent d'évaluer les performances pour une classe de systèmes MPSoC à mémoires partagées aux sous-niveaux PVT. Les modèles de composants conçus dans notre travail sont: le processeur, les mémoires caches, les réseaux d'interconnexion (bus, crossbar), les mémoires d'instructions et de données, le contrôleur DMA et l'accélérateur matériel TCD-2D (Transformée en Cosinus Discrète). Ces composants sont partagés principalement en deux classes: actifs et passifs. Un composant est dit actif s'il est décrit à travers un thread, autrement il est dit passif. Notons, qu'un composant actif peut être initiateur ou cible suivant s'il peut initier ou non une transaction. A l'opposé un composant passif est toujours vu comme une cible. Lors de la description de chaque composant, nous allons justifier les choix qui ont été réalisés.

En utilisant la version 2.1 de SystemC et la version 1 de la bibliothèque TLM, chacun des composants a été conçu et implémenté aux 3 sous-niveaux PVT-PA, PVT-TA et PVT-EA. Cette section 1.7 présente les éléments communs aux 3 sous-niveaux pour tous les com-

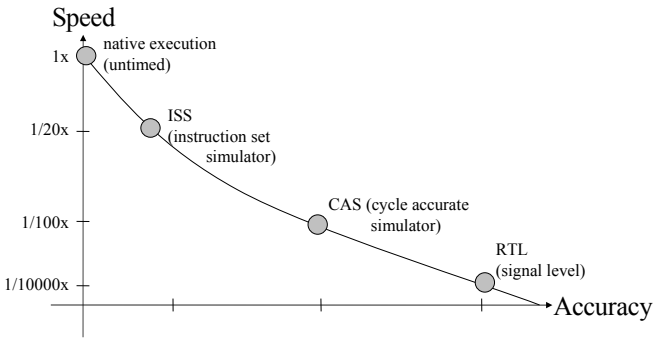


Figure 1.4: *Niveaux de modélisation du processeur [7]*

posants. C'est dans la section 1.8 que les différences entre les sous-niveaux seront soulignées.

1.7.1 Modèle du processeur

Avec l'approche TLM, le comportement d'un processeur peut avoir 3 principales descriptions (figure 1.4). La description la plus précise du processeur est celle où un simulateur au cycle précis est utilisé (CAS dans la figure 1.4). Dans cette description, les détails liés à la micro-architecture sont simulés tels que les étages du pipeline et la prédiction de branchement ce qui permet d'obtenir un modèle de processeur précis. Le processeur MIPS R3000 utilisé dans SoCLib correspond à cette description. Dans la deuxième description, le processeur est modélisé avec un simulateur précis au niveau des instructions (*ISS: Instruction Set Simulator*). Les instructions sont dans ce cas exécutées séquentiellement sans se référer à la micro-architecture du composant. La spécification des délais d'exécution de chaque instruction permet d'estimer le temps d'exécution de l'application. Pour implémenter cette description à l'échelle ISS, nous avons modifié la description du processeur MIPS R3000 au niveau CABA pour qu'elle soit utilisée dans la simulation des deux sous-niveaux PVT-TA et PVT-EA. Dans la troisième description, le processeur réalise l'ensemble de tâches de l'application. Ces tâches seront ensuite exécutées par la machine de simulation. Néanmoins, des annotations de temps peuvent être ajoutées aux tâches pour estimer approximativement le temps d'exécution de l'application. Cette description est utilisée pour modéliser un processeur au niveau PVT-PA.

La figure 1.4 illustre la variation de l'accélération de la simulation en fonction de la précision de la description. L'utilisation des modèles de processeur aux niveaux CABA ou RTL permet d'obtenir des estimations précises avec des temps de simulation importants. A l'opposé, l'exécution native de l'application ou bien l'utilisation des ISS permettent d'accélérer la simulation au prix d'une perte de précision. Pour pallier cet inconvénient, nous avons développé des modèles d'estimation de performance aux sous-niveaux PVT-PA, PVT-TA et PVT-EA qui minimisent l'erreur d'estimation.

Dans chaque sous-niveau, la fonctionnalité du processeur est décrite avec un module SystemC en utilisant le processus SC_THREAD. Par conséquent, un processeur est considéré comme un composant actif. En effet, il y a deux façons possibles de connecter le processeur à d'autres composants de l'architecture.

- La première façon est de faire communiquer le processeur avec des composants passifs donc cibles comme un bus, une mémoire, etc. Dans ce cas les opérations de ces

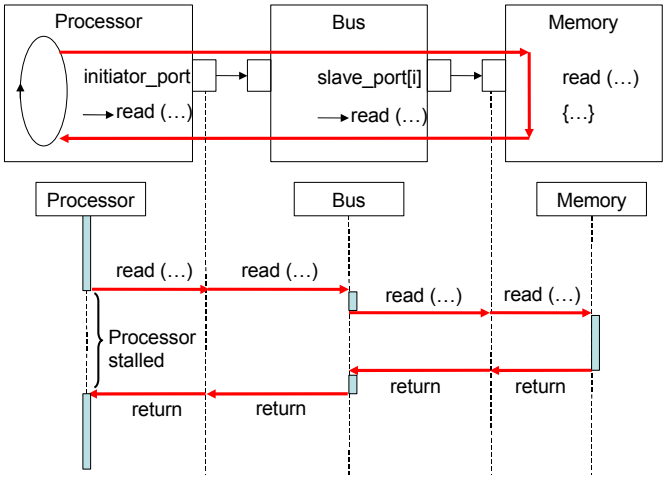


Figure 1.5: *Communication bloquante entre un composant actif (processeur) et un composant passif (mémoire)*

derniers composants sont exécutées avec le contrôle du thread du processeur. La figure 1.5 illustre ce principe. Lorsqu'une nouvelle transaction est réalisée, le processeur envoie une requête *read()* ou *write()* et reste bloqué. Le thread reste actif dans le modèle du bus ou de la mémoire jusqu'a ce qu'il reçoit la réponse de la requête. Ici, nous parlons d'opération de lecture ou écriture bidirectionnelle puisque nous utilisons le même chemin pour la requête et la réponse. Une telle implémentation permet d'achever les gains en accélération de simulation, car il n'y a pas de changement de contexte nécessaire pour l'ordonnanceur SystemC. Néanmoins, avec cette implémentation, nous ne pouvons pas modéliser des composants qui s'exécutent simultanément. A titre d'exemple, pour augmenter les performances (temps d'exécution), le processeur peut continuer à exécuter son code sans attendre la réponse de la mémoire s'il n'a pas besoin de la donnée. De plus, un processeur peut être connecté à un autre composant actif comme le cache de données ou d'instructions, ce qui empêche l'utilisation de ce mécanisme.

- La deuxième technique consiste à faire communiquer le processeur avec des composants actifs, comme le contrôleur DMA ou le cache de données ou d'instructions. Dans ce cas, la communication entre les deux composants n'est pas directe. Elle se fait à travers un composant intermédiaire qui peut être un canal de communication (*Channel*) ou bien un réseau d'interconnexion. La figure 1.5 montre la façon de modéliser une opération de lecture ou d'écriture par un message unidirectionnel pour chacun des deux composants actifs. Dans ce cas, le processeur ou l'initiateur envoie une nouvelle requête et continue à s'exécuter. Au moment où il a besoin de la donnée, celle-ci sera récupérée du canal de communication une fois que la réponse de la cible est reçue. Ce mécanisme permet une modélisation plus exacte de ce qui se passe en bas niveau. Cependant, l'utilisation des threads pour implémenter les composants de type cible ralentit les performances en terme de temps de simulation.

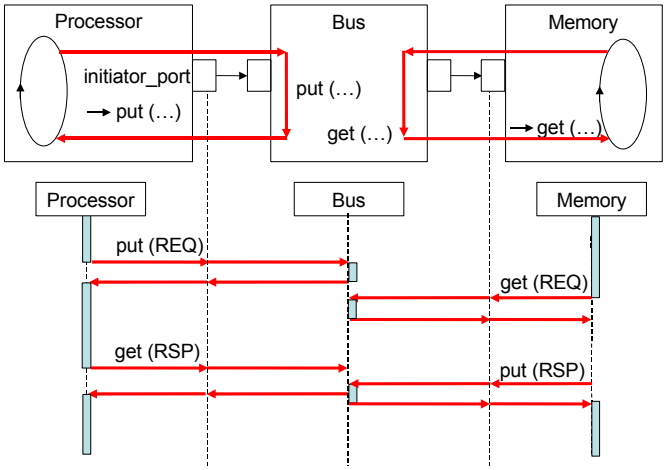


Figure 1.6: *Communication non-bloquante entre composants actifs*

1.7.2 Modèle de la mémoire cache

Le modèle de la mémoire cache est habituellement intégré dans le composant processeur [3, 12]. A l'opposé, dans notre environnement, ce composant est représenté par un modèle séparé du processeur. L'architecture du cache utilisée (taille, associativité, stratégie d'écriture, etc.) influe sur les caractéristiques du système en terme de performance, surface et consommation d'énergie. Par ailleurs, dans un système MPSoC, nous avons besoin de spécifier un mécanisme de cohérence de cache qui dépend du type de réseau d'interconnexion (Bus, Crossbar, etc.) utilisé. Ce mécanisme peut être implémenté soit à travers une solution matérielle, telle que l'espionnage du bus ou le répertoire centralisé [11, 6], ou bien à travers une solution logicielle [10, 9]. Comme on peut le voir la séparation entre le modèle du processeur et celui de la mémoire cache permet d'obtenir une modularité de la conception. Ceci offre la possibilité d'explorer et d'évaluer différentes architectures.

Nous avons conçu un module mémoire cache générique permettant à l'utilisateur de configurer la structure du composant. Les paramètres du cache sont: nombre de blocs, taille du bloc en octets, l'associativité, et la stratégie d'écriture (écriture simultanée ou différée). Ce module est un composant actif et joue le rôle de cible du côté processeur et initiateur du côté réseau d'interconnexion. Cette fonctionnalité est implémentée par un module SystemC possédant deux ports. Le premier port est de type *cible* pour communiquer avec le processeur connecté à la mémoire cache. Le deuxième port, de type *initiateur*, est relié au réseau d'interconnexion pour être utilisé dans le cas de défaut de cache (*cache miss*) ou dans le cas d'opération d'écriture de données en mémoires partagées.

La figure 1.7 montre les communications entre le processeur et le cache d'un côté et le cache et le réseau d'interconnexion de l'autre côté. Dans notre implémentation, nous avons choisi de connecter le processeur et le cache via des canaux de communication bidirectionnels bloquants. Nous pensons que ceci émule ce qui se passe réellement dans un processeur de type scalaire comme le processeur MIPS R3000. En effet, ce processeur permet une exécution dans l'ordre des instructions avec une vitesse d'une instruction/cycle. Le processeur ne peut pas avancer dans l'exécution lors d'un défaut de cache avant d'avoir la donnée manquante. Ceci explique le choix des interfaces bloquantes. Cependant, pour des architectures

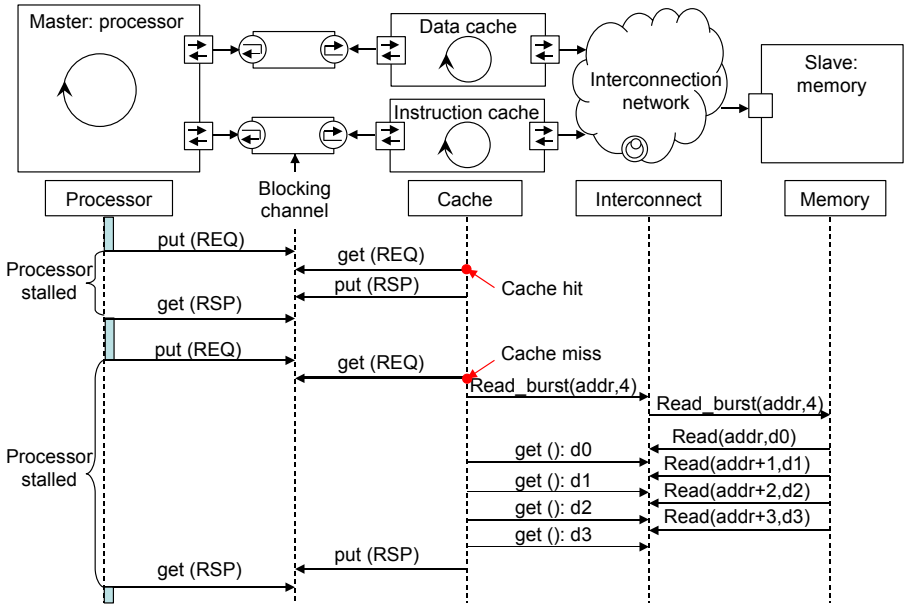


Figure 1.7: *Communication du module mémoire cache avec les autres composants dans le niveau PVT. Taille des blocs=4 mots*

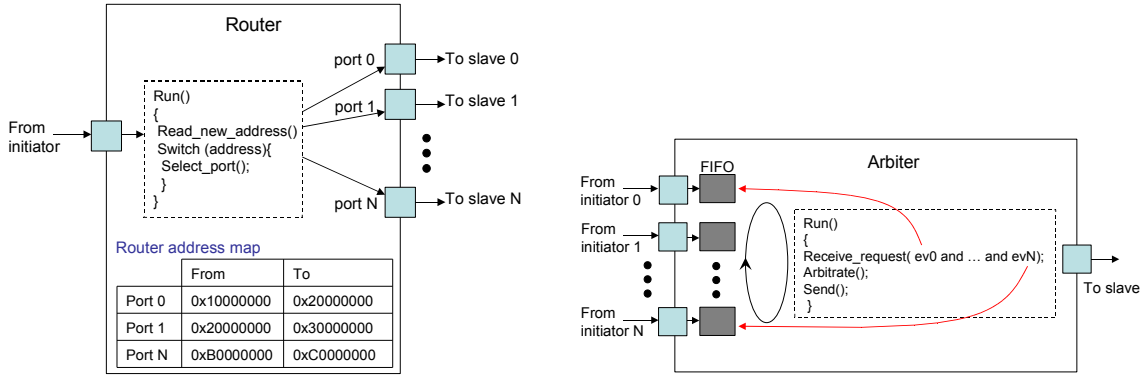
de processeurs plus complexes, comme les processeurs superscalaires, il sera intéressant d'utiliser des interfaces non bloquantes. Ceci va permettre au processeur l'exécution les instructions qui ne dépendent pas de la donnée manquante dans le cache (exécution désordonnée).

Du côté cache/réseau d'interconnexion, les transferts peuvent être réalisés selon deux modes. Le premier correspond au mode simple et consiste à transférer une seule donnée. Le deuxième correspond au mode rafale. Ce dernier est utilisé dans le cas de défaut de cache pour augmenter les performances. La figure 1.7 détaille le mode lecture en rafale dans le cas de défaut de cache. Ainsi, une seule requête commande du cache permet de charger plusieurs données correspondant à des adresses contiguës. Deux spécifications temporelles ont été ajoutées pour estimer le temps d'exécution: le temps d'accès et le temps de cycle à la mémoire cache. Ces deux spécifications ont été obtenues en utilisant l'outil CACTI [1].

1.7.3 Modèle du réseau d'interconnexion

La fonction du réseau d'interconnexion consiste à router les requêtes commandes générées par les initiateurs (processeurs, caches) vers les différentes cibles (mémoires, périphérique d'E/S) et les requêtes réponses dans le sens opposé. Dans cette thèse, nous nous limitons à l'utilisation des composants bus et crossbar. Ce dernier que nous avons développé est basé principalement sur les deux unités fonctionnelles: le routeur et l'arbitre (figure 1.8). Le routeur est un composant générique qui permet de diriger une requête provenant d'un initiateur vers la cible concernée, en utilisant une table de routage spécifiée. Ce module est passif et exécuté sous le contrôle du thread de l'initiateur auquel il est relié. A titre d'exemple dans la figure 1.7, le routeur est exécuté sous le contrôle du cache d'instructions ou de données. Comme le montre la figure 1.8(a), lors d'une nouvelle transaction de l'initiateur, le routeur lit l'adresse correspondante et sélectionne un port de sortie. Pour simuler un comportement

similaire au bas niveau, le routeur développé permet d'implémenter les spécifications des différents types de protocoles de communication (VCI, OCP, etc.). Cette prise en compte des détails du protocole permet, comme il sera présenté par la suite, d'obtenir des estimations proches de celles du niveau CABA.



(a) Le composant routeur (b) Le composant arbitre

Figure 1.8: *Fonctionnement et composition des 2 composants: routeur et arbitre*

Pour gérer les conflits entre plusieurs demandes d'accès simultanées à une cible, il est nécessaire d'utiliser un composant dont le rôle est de définir la priorité de chaque requête. Pour ce faire, nous avons conçu un composant actif, nommé "arbitre" afin d'ordonnancer les accès aux ressources partagées. Lorsqu'un initiateur a besoin d'accéder à une cible partagée, via le réseau d'interconnexion, il transmet une requête en utilisant le canal de communication correspondant et attend la réponse. Au niveau de l'arbitre, un thread lit les requêtes présentes dans la FIFO de chaque canal de communication et sélectionne la requête prioritaire en se basant sur la stratégie d'arbitrage round-robin (figure 1.8(b)). Après le traitement par la cible, l'arbitre transmet la réponse dans la FIFO du canal de communication correspondante. L'initiateur récupère la réponse et termine la transaction. Cette gestion des communications bloque un routeur pendant le traitement de la requête par la cible, ce qui peut représenter un inconvénient. Elle a néanmoins comme avantage, une simplification du protocole et une réduction du nombre de ports. Ces deux facteurs permettent d'accélérer la simulation. Notre arbitre joue un deuxième rôle, très important, concernant l'estimation des délais dans le réseau d'interconnexion comme nous allons le détailler dans la section 1.8.2.

La figure 1.9 illustre l'implémentation d'un Crossbar 2x2 à partir des deux modules routeur et arbitre que nous venons de présenter. Cette architecture est certes relativement simple, mais suffisante pour atteindre notre objectif de pouvoir observer les contentions et récupérer les informations sur les latences. Par ailleurs à partir de cette architecture, plusieurs topologies de réseaux d'interconnexion, peuvent être conçues comme le réseau multi-étage. Les composants proposés (crossbar et bus) sont génériques et permettent à l'utilisateur de spécifier le nombre d'initiateurs et de cibles à connecter ainsi que la latence entre les différents ports pour l'estimation du temps d'exécution.

1.7.4 Modèle de la mémoire

Le module mémoire que nous avons conçu est un composant passif de type "slave" et comprend deux méthodes: read() et write(). Comme ceci a été expliqué dans la section 1.7.1, cette

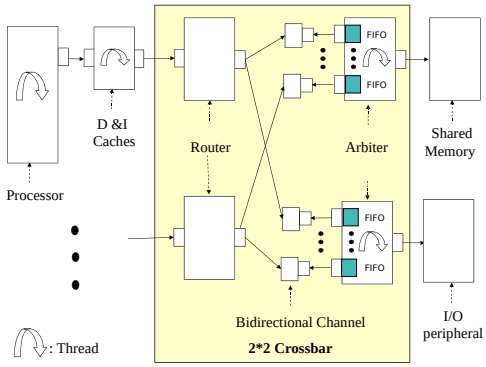


Figure 1.9: *Implémentation d'un crossbar 2x2*

structure nous permet d'accélérer la simulation. En utilisant le mécanisme sc_export introduit dans la version SystemC 2.1, ces deux méthodes sont directement appelées et exécutées dans le thread de l'initiateur connecté au composant mémoire. Ce mécanisme de sc_export est utilisé pour modéliser toutes les cibles de notre système MPSoC tel que les périphériques d'entrées/sorties, etc. Son avantage est qu'il évite l'utilisation des threads pour modéliser les composants de type cible. Par conséquent, il augmente les performances en terme de temps de simulation.

Dans notre environnement, le port de la cible est connecté directement au routeur de l'initiateur si le module est privé à un seul processeur. A l'opposé, si le module est partagé entre plusieurs processeurs, il est connecté à l'arbitre correspondant. Pour améliorer les performances, des mémoires multi-ports peuvent être modélisées au niveau PVT pour simuler des accès simultanés à la même cible. Les paramètres temps d'accès et temps de cycle sont ajoutés à la description du composant pour l'estimation des performances.

1.7.5 Modèle de contrôleur DMA

Pour implémenter le contrôleur DMA au niveau PVT, nous avons simplifié la description du composant par rapport à sa description au niveau CABA. Le modèle contrôleur DMA au niveau PVT est toujours un composant SystemC de type actif, mais la FSM qui contrôle le composant a été supprimée. Ce composant joue le rôle d'un initiateur, pour réaliser le transfert d'un bloc de données d'une adresse source à une adresse de destination, et joue le rôle de cible configurée par un processeur. De ce fait, pour connecter le contrôleur DMA avec le réseau d'interconnexion, nous utilisons trois ports: un de type *cible* et deux de type *initiateur*. Ces deux derniers ports permettent un transfert de données simultané entre une adresse source et une autre destination.

Pour améliorer les performances au cours du transfert de données, le contrôleur DMA utilise aussi les modes de lecture et écriture en rafale comme nous l'avons expliqué dans le composant cache. Par ailleurs, pour imiter le fonctionnement CABA, les délais des événements de la micro-architecture sont mesurés et ajoutés au modèle PVT. Ces délais peuvent être utilisés ou non suivant le sous-niveau PVT dans lequel la simulation est faite.

1.7.6 Modèle de l'accélérateur matériel TCD-2D

La description de l'accélérateur matériel TCD-2D au niveau PVT dérive de notre implémentation au niveau RTL. Ce composant reçoit une séquence de vecteurs de 8 pixels chacun à partir du contrôleur DMA. Après la réception d'un bloc d'images (8x8), le composant opère successivement deux transformations TCD-1D sur les 8 lignes ensuite sur les 8 colonnes. A la fin du traitement, un transfert du résultat vers la mémoire partagée est réalisé par le contrôleur DMA. Un composant actif SystemC a été conçu pour intégrer la fonctionnalité de cet accélérateur. Pour estimer le temps d'exécution de ce composant, les délais de la micro-architecture sont mesurés à partir du niveau RTL et ajoutés au niveau PVT. A titre d'exemple, le temps de chaque transformation TCD-1D vaut 12 cycles.

1.8 Estimation de performance au niveau PVT

Cette section illustre l'utilisation des modèles de composant précédemment décrits pour l'estimation des performances pour les trois sous-niveaux PVT-PA, PVT-TA et PVT-EA. Un modèle de temps pour chaque sous-niveau a été développé. Ces modèles ont été par la suite intégrés dans le simulateur MPSoC en vue d'une exploration des architectures. La méthodologie d'estimation de performance doit satisfaire le critère de flexibilité pour qu'elle soit adaptable aux différentes architectures. La méthodologie proposée doit aussi prendre en considération les problèmes de temps liés à la synchronisation des processeurs, aux contentions dynamiques dans le réseau d'interconnexion et à la spécification du protocole de communication.

1.8.1 Estimation de performance dans PVT-PA

L'estimation de performance au sous-niveau PVT-PA revient à évaluer le temps d'exécution des deux parties calcul et communication. Comme nous l'avons déjà expliqué, dans PVT-PA les tâches sont exécutées nativement sur la machine de simulation. Ceci rend difficile l'estimation du temps d'exécution réel sur un processeur ciblé comme exemple le MIPS R3000. Pour augmenter la précision, nous avons proposé de caractériser ces tâches du point de vue temps d'exécution en utilisant un simulateur de bas niveau. Dans notre cas, pour évaluer les délais de chaque tâche nous avons utilisé le simulateur du processeur MIPS R3000 au niveau CABA. L'estimation des délais d'exécution des tâches dépend bien sûr des paramètres architecturaux, tels que la taille de la mémoire cache d'instructions utilisée. Dans notre cas, ce problème a été simplifié en supposant que cette taille est suffisante pour stocker la totalité du code de la tâche. Pour évaluer le temps d'exécution de la partie calcul, à chaque processeur nous attribuons un compteur de temps local. Sa valeur est incrémentée après l'exécution d'une tâche. Pour respecter les délais d'exécution, des instructions wait() ont été ajoutées après le traitement des tâches (figure 1.10, étape 1).

Pour estimer les latences des communications lors d'une opération de lecture ou écriture des motifs, notre stratégie d'estimation consiste à identifier les activités pertinentes des composants qui interviennent dans les opérations de communication. Ceci concerne les mémoires caches, le réseau, le DMAC, les modules mémoires partagés et enfin l'accélérateur matériel. Ci-après la liste des activités pertinentes qui ont été identifiées:

- Pour le cache: le nombre de succès (*Hit*) et de défauts (*Miss*)

Field	description
`task_time`	Délai de la tâche
`hit_time`	Délai d'un succès de cache
`miss_time`	Délai d'un défaut de cache
`waiting_time`	Délai du réseau d'interconnexion
`read_time`	Délai de l'accès mémoire en lecture
`write_time`	Délai de l'accès mémoire en écriture

Table 1.1: *Description des symboles*

- Pour le réseau d'interconnexion: le nombre de paquets transmis et reçus
- Pour la mémoire partagée: le nombre d'accès en lecture et écriture
- Pour le contrôleur DMA: le nombre de mots transférés
- Pour l'accélérateur matériel TCD-2D: le nombre de blocs traités

L'estimation du temps d'exécution exige la spécification du délai pour chaque type d'activité. Dans notre approche, les délais d'exécution sont mesurés soit à partir d'une caractérisation physique du composant matérielle, soit à partir d'un modèle analytique. Par exemple, les délais d'accès à la mémoire cache sont obtenus en utilisant le modèle analytique proposé par l'outil CACTI [1]. La figure 1.10 montre l'utilisation du compteur de temps local du processeur et les délais des activités pour estimer le temps d'exécution au sous-niveau PVT-PA. Les symboles utilisés dans cette figure sont décrits dans le tableau 1.1.

Lors d'une opération de lecture de donnée de la mémoire cache, le processeur réalise un appel à la fonction (figure 1.10, étape 0). Le paramètre *time* est utilisé pour mesurer l'intervalle de temps entre la transmission de la requête et la réception de la réponse. Il sera récupéré par le processeur pour incrémenter son compteur de temps local. Dans le cas d'un succès de cache, ce temps correspond au temps d'accès pour une opération de lecture du cache (figure 1.10, étape 2). Dans le cas inverse, le cache envoie une nouvelle requête qui sera transmise via le réseau d'interconnexion à la cible concernée (figure 1.10, étape 3). Le temps de l'opération est la somme du temps de transmission, le temps d'accès à la mémoire et le temps d'accès au cache (en cas de lecture) (figure 1.10, étape 4). La difficulté ici est d'estimer le temps de transmission qui n'est pas constant. Ce dernier dépend de la charge dynamique (en terme de transactions) sur le réseau d'interconnexion et des contentions possible pour accéder à une même cible.

Avant de transmettre une requête à la cible, l'arbitre correspondant parcourt l'ensemble des FIFO des différents canaux de communication. L'objectif est de comparer le paramètre *time* des requêtes présentes et de déterminer la plus prioritaire. Ce test permet aussi de localiser les contentions entre les requêtes. Pour évaluer le temps d'attente des requêtes au niveau du réseau d'interconnexion, un compteur est alloué à chaque entrée de FIFO dans le routeur. Lorsqu'une requête est mise en attente, son compteur (*coun_tran_word*) est incrémenté (figure 1.10, étape 5). Plus tard, lorsqu'une FIFO est sélectionnée, la valeur du compteur est lue puis multipliée par l'unité de temps (*t_word*) pour transmettre un seul mot (latence du réseau). Ce compteur est mis ensuite à zéro pour la requête suivante (figure 1.10, étape 6). Nous utilisons la même approche lors de la configuration du contrôleur DMA pour un transfert de données.

Dans notre modèle de temps, nous devons prendre en considération le cas où le processeur attend qu'un de ses motifs d'entrées soit prêt pour reprendre l'exécution. Pour estimer ce temps de repos, l'ordonnanceur implémenté au sous-niveau PVT-PA teste à chaque intervalle de temps si l'une de ses variables de synchronisation est à l'état valide. Si ce n'est pas le cas, alors le compteur local du processeur est incrémenté par un intervalle de temps fixé et nous exécutons une instruction *wait()* pour permettre au simulateur SystemC d'ordonnancer d'autres processeurs. Concernant l'intervalle de temps à choisir, un compromis doit être fait entre la précision et l'accélération de la simulation. En effet, si l'intervalle de temps est trop petit nous pouvons détecter plus rapidement les changements d'état du processeur. Néanmoins, un nombre plus important de changements de contexte entre les processeurs est nécessaire, ce qui va retarder la simulation. A l'opposé, si l'intervalle de temps est trop grand, l'instant de changement d'état du processeur ne sera pas détecté d'une façon précise. Dans notre cas, nous avons choisi un intervalle de temps représentatif du temps de lecture ou d'écriture d'un motif.

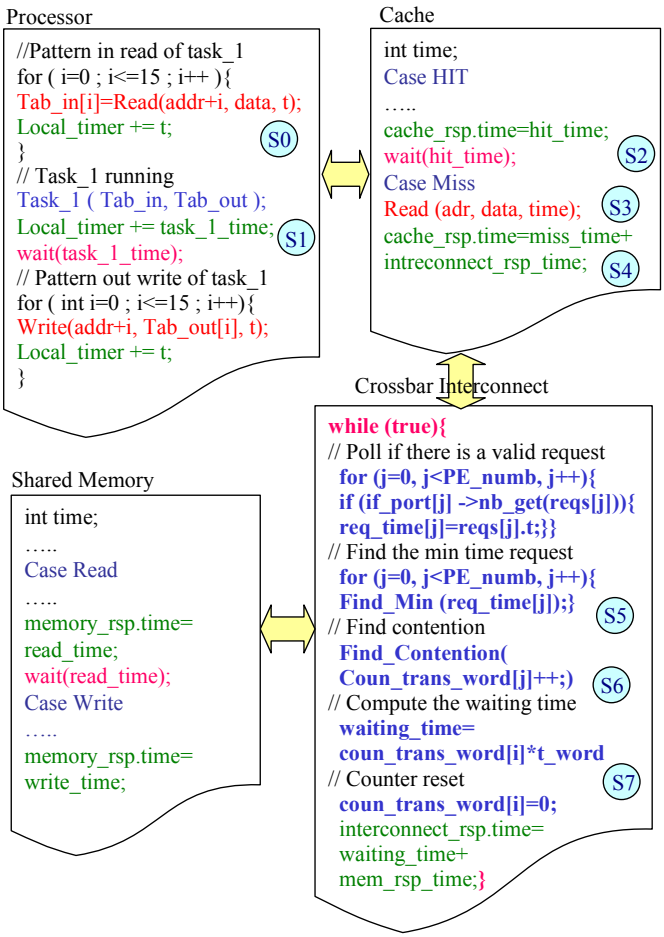


Figure 1.10: *Estimation de performance au sous-niveau PVT-PA*

1.8.2 Estimation de performance dans PVT-TA

La différence entre le modèle du temps au sous-niveau PVT-PA et celui de PVT-TA concerne la partie calcul. En effet, au sous-niveau PVT-TA nous utilisons des ISS pour exécuter l'application. Pour cela nous avons identifié principalement le nombre et le type d'instructions exécutées comme activités pertinentes dans le composant processeur. Dans notre cas les délais d'exécution des instructions du processeur MIPS R3000 sont estimés à partir de la documentation technique donnée par [8]. Le compteur de temps local du processeur est incrémenté après l'exécution de chaque instruction (figure 1.11, étape 0).Ici *Op_add* correspond à l'exécution de l'instruction addition. Le compteur de temps local est incrémenté par le délai d'exécution de l'instruction. Pour estimer le temps d'une opération de lecture de donnée ou d'instruction à partir de la mémoire cache (en cas de succès) ou bien de la mémoire centrale (en cas de défaut de cache), la procédure est la même que celle décrite au sous niveau PVT-PA (figure 1.11, étape 1). Le cas particulier concernant l'estimation du temps lorsque le processeur est mis en attente ne se pose pas dans le cas du sous-niveau PVT-TA. En effet la synchronisation entre les tâches se fait à l'aide de variables déclarées au niveau de l'architecture. A l'état d'attente, le processeur fait des lectures successives sur ces variables et par conséquent le temps de repos est la somme des temps d'accès mémoires avant que celui-ci ne reprenne l'exécution d'une nouvelle tâche.

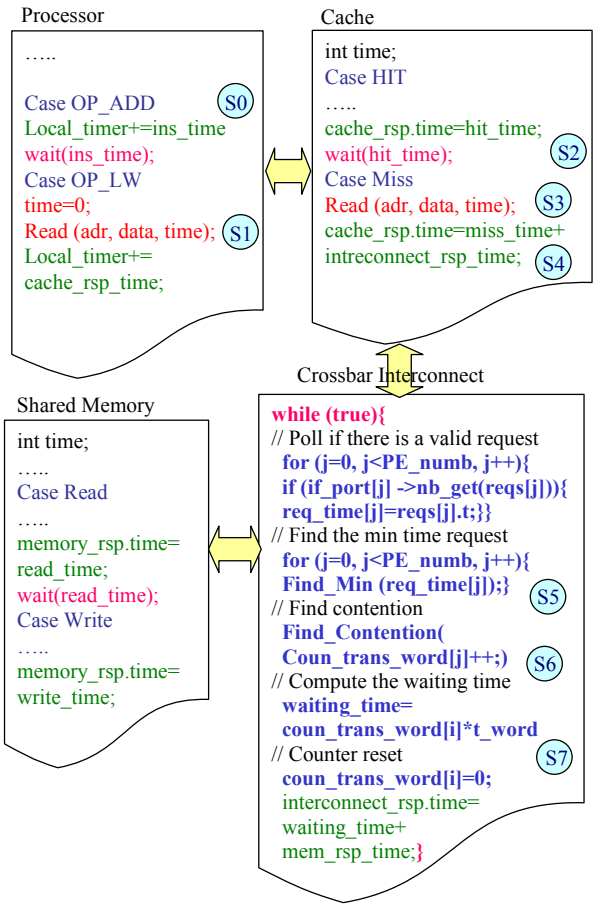


Figure 1.11: *Estimation de performance au sous-niveau PVT-TA*

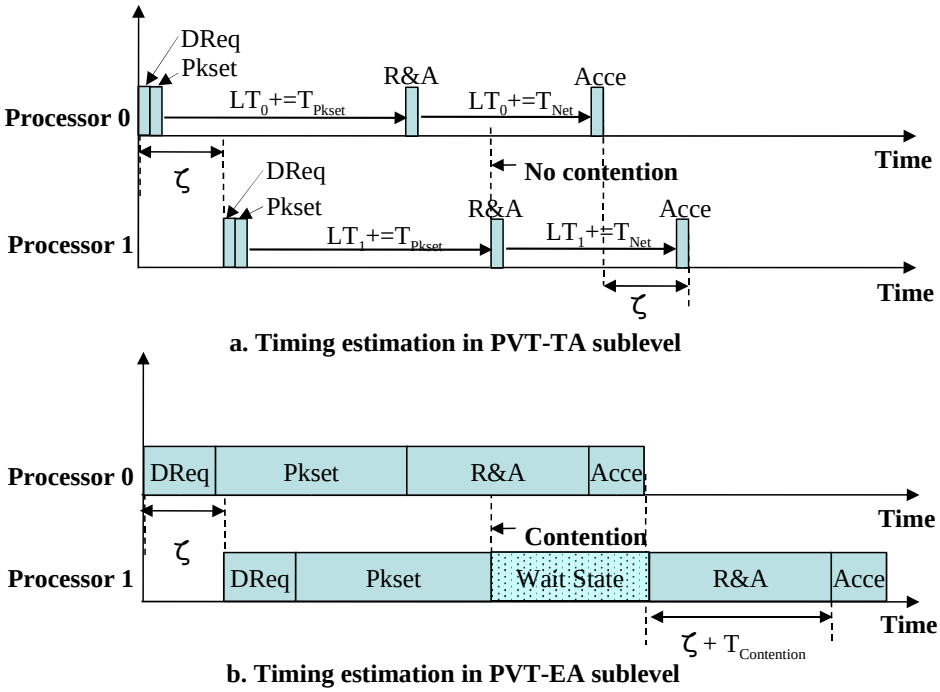
Activités	Délais
Préparation d'une requête de commande VCI	4 cycles
Préparation d'une requête de réponse VCI	4 cycles
Exécution d'une instruction	1 cycle
Succès de cache	1 cycle
Défaut de cache	12 cycles + cycles due aux contentions sur le réseau
accès mémoire en lecture	2 cycles
accès mémoire en écriture	2 cycles
accès au DMAC	2 cycles
TCD 1-D appliquée à un bloc 8x8	12 cycles

Table 1.2: *Délais des activités utilisés dans les expérimentations*

1.8.3 Estimation de performance dans PVT-EA

L'inconvénient du sous niveau PVT-TA c'est qu'il ne tient pas compte des délais temporels entre les événements. Dans le cas de défauts de cache pour deux processeurs différents, l'instant où un processeur effectue son accès mémoire correspondant peut influer sur l'instant d'accès de l'autre processeur en cas de collision. La figure 1.12.a montre un exemple d'erreur de détection de contention dans le réseau d'interconnexion à cause du non-respect des délais des événements (*Packet setup, Routing and Arbitration*). Lorsque le paquet transmis issu du processeur 1 arrive au niveau du routeur (R&A dans la figure 1.12.a), il n'y a aucune possibilité de détecter l'occupation du routeur par le traitement du paquet issu du processeur 0. En effet, les événements dans le sous-niveau PVT-TA sont exécutés instantanément (*zero delay*). Cette abstraction change le comportement de la partie communication dans le système, ce qui diminue la précision d'estimation de performance. Pour résoudre ce problème, nous avons amélioré le sous-niveau PVT-TA par l'introduction d'instructions de synchronisation. Elles prennent en considération les délais des activités des composants, les délais de transmission des paquets et finalement le protocole de communication. Ce sont les caractéristiques du sous-niveau PVT-EA. Les événements à ce sous-niveau se produisent de la même façon qu'au niveau CABA (figure 1.12.b).

Pour pouvoir comparer l'erreur d'estimation entre le sous-niveau PVT-EA et le niveau CABA pour la plateforme SoCLib, il nous a fallu implémenter les spécifications du protocole VCI. Néanmoins, notre modèle d'estimation de performance est flexible et nous permet d'implémenter une variété de protocoles de communication. Pour émuler le même comportement du protocole VCI au sous-niveau PVT-EA et respecter le comportement des composants, des instructions *wait()* ont été ajoutées dans la description des composants ainsi qu'avant la transmission des requêtes commande et réponse. Les instructions *wait()* à ajouter nécessitent des arguments exprimés en unité de temps tels que nano seconde (ns) ou en nombre de cycles. Dans nos expérimentations, ces arguments sont mesurés depuis la plateforme CABA. Le tableau 1.2 présente les délais apportés au sous-niveau PVT-EA.



a. Timing estimation in PVT-TA sublevel

b. Timing estimation in PVT-EA sublevel

LT0: Local Timer 0; **LT1**: Local Timer 1; **DReq**: Data Request

Pkset: Packet Setup; **R&A**: Routing and Arbitration; **Acce**: Memory Access

$\mathbf{T_{Pkset}}$: Packet Setup time; $\mathbf{T_{Net}}$: Network time

Figure 1.12: *Estimation de temps dans les sous-niveaux PVT-TA et PVT-EA*

1.9 Simulation et résultats

1.9.1 L'environnement de simulation

Dans cette section nous allons évaluer la pertinence de notre environnement PVT proposé avec ces 3 sous-niveaux en vue d'une exploration d'architecture pour les systèmes MPSoC. Plusieurs expérimentations ont été menées. L'objectif était de mesurer pour chaque sous-niveau décrit précédemment l'accélération de la simulation, la précision de prédiction de performance et l'effort de développement nécessaire pour modéliser les composants du système MPSoC. Ces métriques sont rapportées en comparaison avec le niveau CABA et en faisant varier la taille du cache de données et d'instructions et le nombre de processeurs. Il est intéressant de noter que cet environnement peut être utilisé pour accomplir une exploration architecturale en fonction d'autres paramètres (exemple: type de processeur, type de réseau d'interconnexion, etc.) dans un intervalle de temps raisonnable et une précision acceptable. La figure 1.13 montre l'architecture du système MPSoC utilisée dans les expérimentations. Nous avons parallélisé et testé plusieurs applications de traitement de données intensif: la multiplication de matrices, le downscaler [5], une version logicielle de la TCD et le codeur H.263 [2]. Ces applications sont parallélisées en partageant le calcul d'une façon équitable entre les différents processeurs. Par exemple, la TCD est parallélisée en attribuant

différents segments d'image pour chaque processeur.

Le facteur d'accélération de la simulation est défini comme suit:

$$Speed\_fact = PVT\ Simul.time/CABA\ Simul.time$$

Dans cette expression, PVT correspond au sous-niveau PVT-PA, PVT-TA ou PVT-EA. Dans toutes les expérimentations, une machine équipée d'un processeur Pentium M (1.6 GHz) et 1GB de mémoire est utilisée. Nous notons que les tests au niveau CABA exigent plusieurs heures de simulation et dépendent de la configuration du système simulé. L'application H.263 est choisie pour illustrer nos expérimentations, elle sera détaillée dans la sous-section suivante.

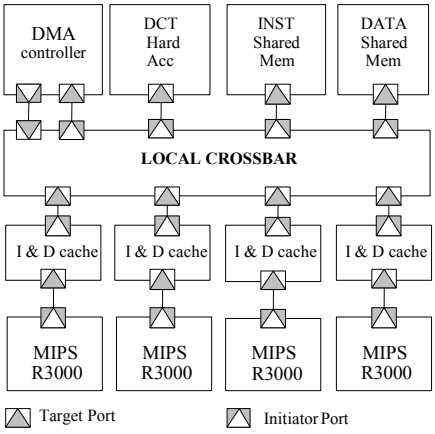


Figure 1.13: *Exemple de structure MPSoC*

1.9.2 Le codeur H.263

Le standard H.263 de codage a été développé pour la transmission de la vidéo sur des lignes à très bas débits pour les applications de visiophonie, les systèmes de surveillance sans fil, etc. Dans nos expérimentations, nous nous intéressons à l'implémentation du codeur H.263. Cette application est composée de trois tâches; la Transformée en Cosinus Discrète (TCD), la quantification (Quant) et le codage (VLC). La TCD permet d'éliminer la redondance de données et de transformer les données du domaine spatial en une représentation fréquentielle. La tâche de quantification consiste à diviser chaque coefficient de la TCD par un pas de quantification et mettre les coefficients non significatifs à zéro. La tâche de codage permet d'encoder les coefficients TCD quantifiés en attribuant aux différents symboles (luminance, chrominance) un mot binaire. La méthode de codage utilisée est celle de Huffman [4].

Les données manipulées sont structurées sous forme de macroblocs qui représentent un espace de 16x16 pixels d'une image vidéo. Le format de données du macrobloc est le YCbCr qui contient trois composantes: luminance (Y), chrominance bleu (Cb) et chrominance rouge (Cr). Les blocs de luminance décrivent l'intensité des pixels tant que les blocs de chrominance contiennent des informations sur les couleurs des pixels. Un macrobloc contient 6 blocs de 8x8 : 4 blocs contiennent les valeurs de la luminance, un bloc contient les valeurs de chrominance bleu et un bloc contient les valeurs de chrominance rouge. Cette application est ici parallélisée pour qu'elle puisse s'exécuter en utilisant un système MPSoC formé de 4 à

16 processeurs. La tâche TCD est affectée à l'accélérateur matériel TCD-2D. Les deux tâches quantification et codage sont partagées équitablement entre les processeurs en allouant différents macroblocs à chaque processeur. Au cours des simulations, le codeur H.263 est appliqué à la séquence vidéo Foreman.qcif (figure 1.14).

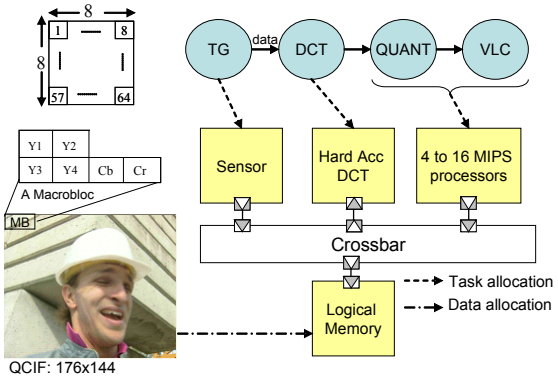


Figure 1.14: *L'application codeur H.263*

1.9.3 Résultats de simulation au sous-niveau PVT-PA

Au sous niveau PVT-PA, nous avons identifié les différentes tâches de l'application H.263, la spécification du domaine d'itération ainsi que les synchronisations nécessaires entre les différentes tâches. Nous avons exécuté notre application en faisant varier le nombre de processeurs entre 4 et 16 pour évaluer le facteur d'accélération et la précision d'estimation. Comme premier résultat de simulation, nous avons vérifié notre hypothèse concernant le flux d'accès à la mémoire d'instructions qui est négligeable devant le flux d'accès à la mémoire de données. En CABA nous observons que les instructions ne représentent que 0.7% du nombre d'accès total.

La figure 1.15 nous montre les résultats de simulation au sous-niveau PVT-PA en faisant varier le nombre de processeurs. Nous observons que l'augmentation du nombre de processeurs entraîne une amélioration du facteur d'accélération qui peut atteindre la valeur 32 avec un système de 16 processeurs. Une analyse précise de la trace produite par le simulateur SystemC nous montre que 25% du temps de simulation concerne l'exécution de la fonction du réseau d'interconnexion alors que le temps de simulation de la partie calcul est presque nul. En particulier, l'utilisation d'un nombre assez important de processeurs crée plus de contentions sur le réseau d'interconnexion lors des accès simultanés aux ressources partagées. PVT-PA est efficace pour la modélisation d'applications nécessitant beaucoup de communications. Malgré sa performance en terme d'accélération, le sous-niveau PVT-PA comporte une erreur d'estimation de performance raisonnable qui varie en fonction du nombre de processeurs et qui ne dépasse pas 27% (figure 1.15). Cette erreur est due principalement à deux causes. La première est liée à notre supposition de négliger les accès mémoires qui correspondent aux variables locales des fonctions et des variables de synchronisation. Ce type d'accès peut présenter jusqu'à 10% de l'accès total aux ressources partagées. La deuxième cause correspond à l'erreur de détection des contentions dans le

réseau d'interconnexion. En effet, les transferts de motifs au sous-niveau PVT-PA ne sont qu'une approximation des transferts réels observables au niveau CABA.

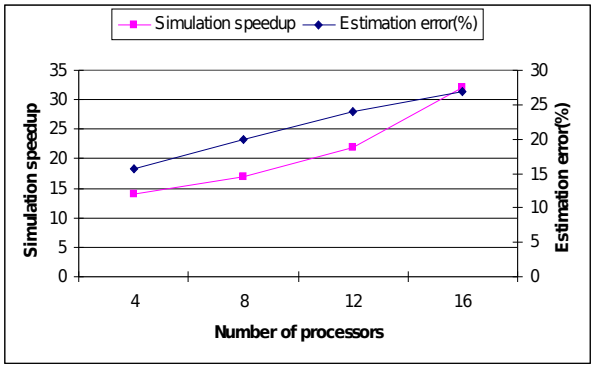


Figure 1.15: *Estimation de performance et de l'erreur d'estimation au sous-niveau PVT-PA*

Jusqu'à ce point, nous nous sommes focalisés sur l'utilité du sous-niveau PVT-PA pour accomplir une validation fonctionnelle du système MPSoC avec des facteurs intéressants d'accélération de la simulation. Ce sous-niveau peut être utilisé aussi pour l'observation des contentions au cours de l'exécution de l'application. Il est possible de détecter les goulots d'étranglement dans le réseau d'interconnexion. La figure 1.16 présente le nombre de contentions cumulées chaque 1000 cycles au cours de l'exécution. Ce résultat est obtenu avec un système de 4 processeurs. Les allures des deux courbes PVT-PA et CABA sont proches l'une de l'autre. Les transferts de données au sous-niveau PVT-PA se font de façon régulière avant et après l'exécution de la tâche contrairement à ce qui se passe réellement en bas niveau où les transferts de données se font au cours de l'exécution de la tâche. Pour cela le nombre de contentions au sous-niveau PVT-PA est présenté sous forme d'une valeur moyenne par rapport aux contentions au niveau CABA. Ce qui est intéressant à partir de cette figure 1.16 c'est que nous pouvons détecter aussi facilement les goulots d'étranglement dans le réseau d'interconnexion au niveau PVT-PA qu'au niveau CABA.

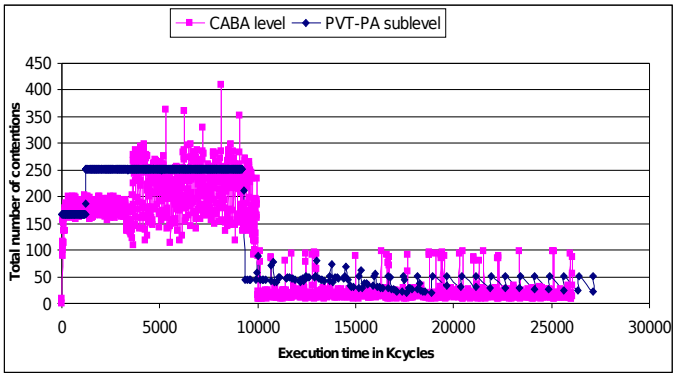


Figure 1.16: *Contentions dans le réseau d'interconnexion dans PVT-PA and CABA*

1.9.4 Résultats de simulation au sous-niveau PVT-TA

Pour réduire l'erreur d'estimation de performance au sous-niveau PVT-PA, nous allons utiliser des ISS pour exécuter l'application. Les tâches de l'application H.263 sont compilées vers notre processeur MIPS R3000. Notre étude à ce sous-niveau va être plus détaillée qu'au paragraphe précédent puisque nous envisageons une estimation assez précise de performance et ensuite de la consommation d'énergie (chapitre suivant). Une analyse par trace de l'exécution de l'application montre que l'introduction des ISS des processeurs dans le système MPSoC produit un ralentissement de la simulation de la partie calcul par un facteur de 30. Néanmoins, cette partie ne présente que 6% de la charge totale de la simulation et la partie communication reste prédominante pour le temps de simulation.

Pour évaluer l'impact de la taille des caches d'instructions et de données sur le facteur d'accélération de la simulation, nous avons exécuté plusieurs applications au sous-niveau PVT-TA en utilisant un système MPSoC de 4 processeurs. La taille des caches varie de 1 Koctets à 32 Koctets (Ko) en adoptant la configuration suivante: l'associativité=1, la taille du bloc= 32octets et la stratégie d'écriture est "Write-Through". La figure 1.17 montre que la réduction de la taille des caches diminue les temps de simulation. En effet, les caches de taille réduite amplifient le nombre de défauts et par conséquent le trafic sur le réseau d'interconnexion. Nous notons aussi un écart important dans l'accélération lorsque le noyau de l'application et les données correspondantes ne peuvent pas être totalement stockés dans les caches. Dans notre exemple, un écart important dans l'accélération est obtenu à partir des tailles de caches inférieures à 16 Ko pour l'application H.263. Pour la version software de la TCD, cet écart important est détecté à partir des tailles de caches inférieures à 2 Ko.

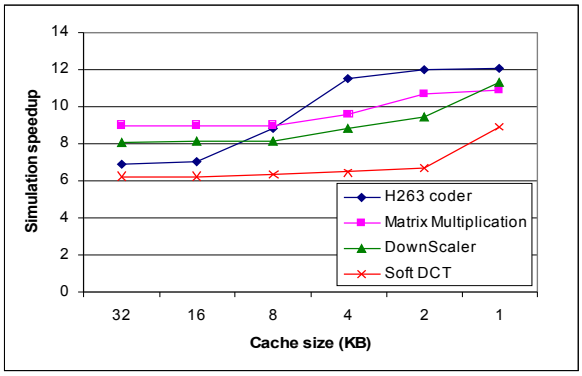


Figure 1.17: *Accélération de la simulation pour différentes applications*

Pour évaluer l'impact du nombre de processeurs sur le facteur d'accélération de la simulation, nous avons exécuté l'application H.263 au sous-niveau PVT-TA avec un système MPSoC formé de 4 à 16 processeurs. La figure 1.18 montre qu'avec le sous-niveau PVT-TA, il est possible d'accélérer la simulation avec un facteur qui peut atteindre 18. L'addition de nouveaux processeurs au système augmente le facteur d'accélération, cela s'explique par l'amplification de la communication entre les processeurs et les modules mémoires partagés.

Le sous-niveau PVT-TA comporte une faible erreur d'estimation de performance. Comme le montre la figure 1.20, l'augmentation du nombre de processeurs ou l'utilisation des caches de petite taille accentue les communications et l'erreur d'estimation. Cette erreur peut atteindre au maximum 7.2% avec un système de 16 processeurs et 1Ko de taille de

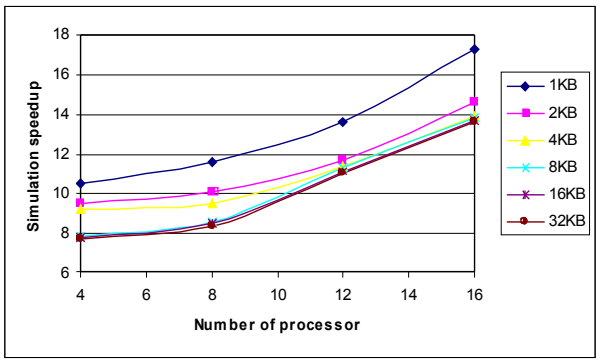


Figure 1.18: *Accélération de la simulation dans le sous-niveau PVT-TA*

cache. Toujours pareil, plus il y a de communication, plus il y a d'erreurs de détection des contentions.

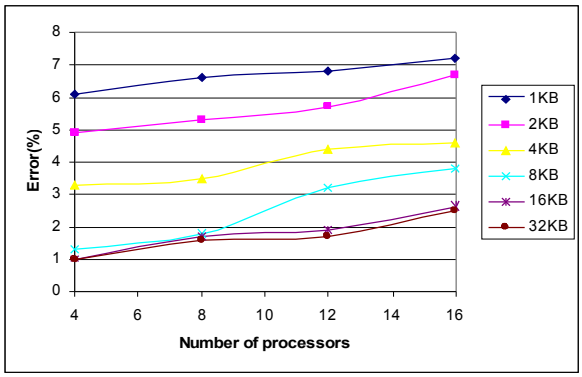


Figure 1.19: *Erreur d'estimation dans le sous-niveau PVT-TA*

1.9.5 Résultats de simulation au sous-niveau PVT-EA

Plusieurs expérimentations ont été menées en utilisant la même application et les configurations du système MPSoC pour évaluer le sous-niveau PVT-EA. Dans le modèle de temps, nous intégrons les spécifications du protocole VCI ainsi que les délais entre les événements. L'erreur de précision avec PVT-EA est réduite presque à zéro pour toutes les configurations testées. Nous pensons que l'utilisation d'un processeur d'architecture simple avec une exécution dans l'ordre (*in-order*) est parfaitement adaptée à notre niveau de simulation et permet ainsi de minimiser l'erreur d'estimation. Par rapport au sous-niveau PVT-TA, le sous-niveau PVT-EA ralentit la simulation de 30%(Fig 1.20).

1.9.6 Effort de modélisation

Jusqu'à maintenant nous avons montré l'utilité de notre approche en terme d'accélération de la simulation et en terme d'estimation de performance. Néanmoins, cette approche a prouvé son efficacité aussi en terme d'effort de modélisation. Elle permet aux concepteurs le

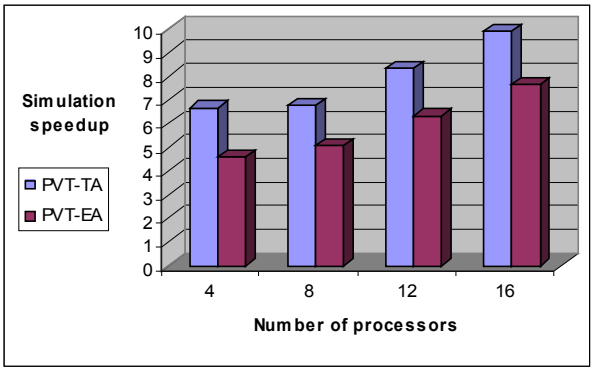


Figure 1.20: *Comparaison de l'accélération entre PVT-TA et PVT-EA*

Abstract level	Modeling effort(LOC)							Reduction (%)
	Processor	Cache	Interconnect	Memory	DMA	TCD	Total	
CABA	1578	553	399	312	650	300	3792	
PVT-EA	1486	244	177	183	115	175	2380	37.23
PVT-TA	1259	238	170	180	105	170	2122	44.04
PVT-PA	330	238	170	180	105	170	1193	68.53

Table 1.3: *Comparaison de l'effort de modélisation*

développement et la validation des systèmes MPSoC dans un temps réduit. Le tableau 1.3 présente l'effort de modélisation exprimé en terme de lignes de code (*LOC:lines of code*) nécessaires pour la conception d'un système MPSoC dans les niveaux CABA et PVT (PVT-PA, PVT-TA et PVT-EA). Selon les résultats, l'effort de modélisation avec PVT-PA, PVT-TA et PVT-EA est réduit respectivement d'un facteur de 68.53%, 44.04% et 37.23%. L'utilisation d'une stratégie de simulation multi-niveaux (par objectifs) permet rapidement de focaliser sur un sous-ensemble de systèmes MPSoC sans avoir à multiplier les efforts de modélisation pour chaque niveau d'abstraction.

1.10 Synthèse et généralisation

Au cours de ce chapitre, nous avons présenté un environnement décrit au niveau PVT pour simuler des systèmes MPSoC. Cet environnement se décline en 3 sous-niveaux PVT-PA, PVT-TA, et PVT-EA. A chaque sous-niveau d'abstraction, un modèle d'estimation de performance a été développé pour évaluer le temps d'exécution d'une application donnée. En utilisant ces trois sous-niveaux, plusieurs objectifs ont été atteints:

- Les trois sous-niveaux permettent une vérification fonctionnelle du système MPSoC. Néanmoins, c'est PVT-PA qui offre la vérification la plus rapide en comparaison avec le niveau CABA.

- Les trois sous-niveaux intègrent un modèle d'estimation de performance et offrent différents compromis accélération/précision. PVT-TA présente le meilleur compromis.

- Avec le sous-niveau PVT-PA, nous avons pu visualiser les contentions sur le réseau d'interconnexion. Ceci constituera un outil pour vérifier l'adéquation du support de communication avec le système et pour détecter les goulots d'étranglement vers une cible particulière.
- Avec le sous-niveau PVT-EA, les spécifications du protocole de communication ont été prises en considération au plus tôt dans le processus de développement.
- En utilisant les 3 sous-niveaux, nous avons pu réduire considérablement l'effort de développement par rapport au niveau CABA.

Pour faciliter aux concepteurs la simulation et l'évaluation de performance des systèmes MPSoC en utilisation les 3 sous-niveaux développés, nous allons intégrer les différents composants matériels conçus dans notre environnement de conception Gaspard. En plus, nous allons définir les concepts nécessaires permettant de prendre en considération le sous-niveau choisi et les informations temporelles correspondantes comme il sera décrit dans le chapitre **??**.

Les composants matériels seront utilisés depuis notre bibliothèque au niveau PVT. Pour intégrer d'autres composants décrits au niveau transactionnel dans notre bibliothèque, le concepteur doit adapter les méthodes de communication à notre définition (*read(adr, data, time)* et *write(adr, data, time)*). Dans notre proposition, nos modèles d'estimation de performance sont fortement liés à la description des composants. Néanmoins, nous pouvons généraliser notre approche. Pour les initiateurs qui exécutent les tâches, nous définissons un compteur de temps local pour chacun afin d'estimer le temps d'exécution de la tâche correspondante. Pour les autres composants qui jouent le rôle d'une cible ou d'un intermédiaire de communication, la requête est récupérée et le paramètre *time* est lu. Après l'exécution de sa fonctionnalité, chaque composant ajoute les délais appropriés au paramètre *time* de la requête. Ainsi le temps de transmission totale est calculé. Ce paramètre sera récupéré par l'initiateur pour incrémenter son compteur local.

Comme ceci a été décrit au niveau CABA, nous visons avec notre environnement Gaspard la génération automatique du code SystemC au niveau PVT à partir d'une description de haut niveau du système MPSoC. Le même ensemble de transformations décrit au niveau CABA sera réutilisé pour les sous-niveaux PVT-PA, PVT-TA et PVT-EA. Ceci est considéré parmi les avantages de la méthodologie de conception dirigée par les modèles. La distinction entre les spécifications de chaque sous-niveau fera l'objet de la phase de déploiement qui sera détaillée dans le chapitre **??**.

1.11 Conclusion

Pour réduire la complexité de développement et accélérer l'exploration de l'espace de solution architectural, il s'avère indispensable de modéliser les systèmes MPSoC dans des phases avancées du flot de conception. Notre contribution dans ce chapitre consiste à décrire les systèmes au niveau transactionnel PVT. A ce niveau, nous avons proposé un environnement décrit avec 3 sous-niveaux PVT-PA, PVT-TA et PVT-EA permettant l'accélération de la simulation des systèmes MPSoC. Différents composants matériels ont été conçus pour implémenter les trois sous-niveaux. Afin d'obtenir une prédiction précise du temps d'exécution dans notre environnement, nous avons enrichi les trois sous-niveaux par des modèles de

temps permettant une erreur d'estimation relative à la précision de description du système. Les résultats de simulation montrent une complémentarité entre les trois sous-niveaux PVT-PA, PVT-TA et PVT-EA qui offrent différents compromis accélération/précision.

Dans notre travail, nous nous sommes limité à des systèmes MPSoC à mémoires partagées ainsi des architectures de processeurs simples tels que le MIPS R3000. Il sera intéressant d'étudier notre approche d'accélération pour des systèmes MPSoC plus complexes tels que des architectures à mémoires distribuées. Dans les perspectives nous proposons d'étudier le comportement de processeurs plus complexes tels que des topologies superscalaire ou VLIW (*Very Large Instruction Word*). Nous nous sommes aussi intéressés dans ce travail aux applications de traitement de données intensif qui ne nécessitent pas généralement l'utilisation d'un système d'exploitation (SE). La prise en considération du comportement du SE lors de la simulation logicielle/matérielle des systèmes MPSoC ainsi que les aspects de temps correspondants reste des problèmes difficiles qui dépassent les objectifs de cette thèse.

Certes, l'estimation de performance est un critère efficace pour comparer les différentes alternatives d'un espace de solution architectural. Aujourd'hui dans les systèmes embarqués, il est indispensable de prendre en compte la consommation d'énergie comme un critère de développement au même titre que la vitesse et la surface. Une exploration fiable des systèmes MPSoC nécessite de définir des outils d'estimation de consommation d'énergie à différents niveaux d'abstraction. Ce défi fait l'objectif de notre prochain chapitre.

Bibliography

[1] CACTI Home Page. `http://research.compaq.com/wrl/people/jouppi/CACTI`.

[2] G. Cote, B. Erol, M. Gallant, and F. Kossentini. H.263+: video coding at low bit rates. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):849–866, November 1998.

[3] F. Ghenassia. *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer, 2006.

[4] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.

[5] H. Lee, B. Lee, Y. Lee, and B. Kang. Optimized vlsi design for enhanced image downscaler. In *Second IEEE Asia Pacific Conference on ASIC*, 2000.

[6] M. Loghi and M. Poncino. Exploring energy/performance tradeoffs in shared memory mpsocs: Snoop-based cache coherence vs. software solutions. In *Design, Automation and Test in Europe Conference and Exhibition*, 2005.

[7] T. Meyerowitz. Transaction level modeling definitions and approximations. `http://www.eecs.berkeley.edu/~alanmi/courses/2005_290A/reports/290a_modeling.pdf`.

[8] MIPS Technologies Consortium. `http://www.mips.com`.

[9] F. Petrot, A. Greiner, and P. Gomez. On cache coherency and memory consistency issues in noc based shared memory multiprocessor soc architectures. In *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, 2006.

[10] I. Tartalja and V. **Milutinović**. Classifying software-based cache coherence solutions. *IEEE Softw.*, 14(3):90–101, 1997.

[11] M. Tomasevic and V. Milutinovic. Hardware approaches to cache coherence in shared-memory multiprocessors part 2. *IEEE Micro*, 14(6):61–66, 1994.

[12] E. Viaud, F. PÃłcheux, and A. Greiner. An efficient TLM/T modeling and simulation environment based on parallel discrete event principles. In *Design, Automation and Test in Europe Conference and Exhibition*, 2006.