APPLICATION

Analyse des performances des machines parallèles : approche par simulation à événements discrets

Renaud Goullioud – Nacer Abouchi – Richard Grisel

CPE - LISA - CNRS 0092
31, place Bellecour
69288 Lyon Cedex 02

RÉSUMÉ. *L'analyse des performances des machines multiprocesseurs peut, selon les buts souhaités, s'envisager de plusieures manières : l'approche analytique, l'utilisation de logiciels de simulation spécifiques ou l'utilisation de simulateurs existants et d'application très générale. Dans cet artice, nous avons retenu la simulation à événements discrets avec le simulateur SIMAN. Ainsi, les principaux éléments du système sont modélisés sans atteindre le niveau électronique du composant.*
Afin de valider notre approche, nous avons effectué une étude comparée, entre l'approche analytique et la modélisation par SIMAN, du problème de la cohérence entre les caches. Les résultats obtenus montrent que notre approche est précise et sensible. De plus, nous avons mis en évidence ses apports par rapport à la solution analytique
Afin de montrer l'intérêt de l'approche, un cas pratique est étudié : une carte à trois Transputers. Les résultats comparés entre les simulations d'un programme réel sur le modèle de la carte et l'exécution réelle ont confirmé l'exactitude et la précision de notre d'analyse. Bien que l'étude ait été effectuée sur une machine à base de trois Transputers, elle peut sans difficultés être appliquée à des architectures plus complexes.

ABSTRACT. *Real time data processing, in few topics such as image processing, requires the high processing capability of the machines based on multiple processors. There is a number of ways of approaching the analysis of the performance of computers : the analytical approach, the use of simulation software dedicated to electronics or dedicated to multiprocessors or the use of general purpose simulators. We have selected the last approach by choosing the SIMAN discrete events simulator.*
First, we want to show that SIMAN could be well suited to multiprocessor behavior simulation and that it quickly allows creating reliable models. Second, to confirm the validity of our modelization, we study the concrete problem of a system of caches for multiprocessors. The comparison between the results that we have obtained and the ones obtained by analytical modelization of the system allows us to think about the accuracy of our models.

MOTS CLÉS : *Machines multiprocesseurs à mémoire partagée, Simulation à événements discrets, Cohérence des caches, Transputer.*
KEY WORDS : *Shared Memory Multiprocessors, discrete events simulations, cache coherence, Transputer.*

1. Introduction

Le traitement d'information en temps réel, dans des domaines aussi variés que les télécommunications et le traitement d'images ne peut s'envisager correctement que dans des structures de machines faisant appel à plusieurs processeurs [JAC93]. Les performances de ces machines dépendent "hélas" de plusieurs paramètres : la topologie, la taille des mémoires, la parallélisation des algorithmes, etc. Il n'est donc pas facile en absolu de dire que telle ou telle machine est meilleure. D'autre part, le coût très élevé de conception de ces systèmes reste un gros obstacle et nous oblige à être "sûr" de nos conceptions avant de démarrer la réalisation.

La simulation est et reste un outil très efficace pour évaluer les performances d'une machine face à un ou des problèmes donnés. Plusieurs approches pouront être envisagées. Dans cet article, la simulation à événements discrets, basée sur le langage SIMAN a été retenue. Nous montrerons après avoir présenté les différentes approches classiques de simulation, que SIMAN peut être très bien adapté à la simulation des machines parallèles et permet sans difficultés d'établir très rapidement des modèles comportementaux très fiables.

Afin de confirmer la validité de nos modèles et donc de notre analyse basée sur la simulation à événements discrets, nous avons confronté nos résultats, obtenus sur l'étude d'un **système de mémoires caches pour architectures multiprocesseurs**, à ceux obtenus analytiquement [GOU96].

Ensuite, nous justifierons la nécéssité d'effectuer des simulations à partir de programmes réels. C'est pourquoi, nous détaillerons la modélisation d'une carte existante à base de trois Transputers : 3DTrans. Les résultats de simulation du modèle de la carte face à un programme de convolution seront comparés à l'exécution réelle afin de montrer la fidélité du simulateur SIMAN.

2. Les différentes approches de la simulation

Afin d'évaluer les performances des systèmes concurrents et en particulier des machines multiprocesseurs, différentes approches peuvent être envisagées selon les buts souhaités :

2.1. L'approche analytique

Elle est basée sur une description mathématique par des équations d'états, définissant la structure du système et son évolution dans le temps [WOO94]. La complexité des architectures multiprocesseurs rend leur mise en équation difficile, oblige un certain nombre d'approximations et suppose une homogénéité importante du système et des processus. La plus grosse difficulté est sans aucun doute la mise en équation du problème et sa simplification pour pouvoir en résoudre les formules. La

notion de temps reste difficile à inclure dans le modèle et la simulation de programmes parallèles de test est quasi-impossible.

2.2. L'utilisation de logiciels de simulation spécifiques à l'électronique

On réalise un schéma complet de la structure électronique et on effectue des simulations logiques pour observer les réactions du système. Cependant, il n'existe pas toujours de modèles de tous les composants et la saisie de schéma devient laborieuse, de même que la simulation, lorsque la taille de la structure augmente. La modification d'un paramètre oblige souvent le retour à la saisie du schéma, l'impact résultant de la modification des paramètres de la machine reste très difficile à observer.

2.3. L'utilisation d'un simulateur dédié aux machines multiprocesseurs

De nombreux simulateurs ont été entièrement développés par les concepteurs de machines multiprocesseurs : MINT [VEE94], TANGO [DAV91], etc. Ces simulateurs sont en général très performants, mais leur réalisation demande un important travail de programmation pour recréer le noyau du simulateur. Le principal intérêt de ces simulateurs est la possibilité d'exécuter des programmes parallèles de test (dits "benchmarks") sur le modèle de machine parallèle : le groupe de benchmarks SPLASH de l'université de Stanford [SIN92][WOO95] ou le "perfect club benchmarks" de l'université d'Illinois [BER89]. Par contre, la description du modèle de la machine cible est forcément réalisée par un programme décrivant la réponse de la cible aux diverses sollicitations.

2.4. L'utilisation de simulateurs généraux existants

Certains de ces simulateurs existants que l'on peut qualifier d'universels s'avèrent, moyennant une adaptation, simple à utiliser dans le domaine des machines parallèles. L'établissement du modèle est facilité par des modules pré-programmés [CER88]. Ces simulateurs sont soit de **type continu** : le temps augmente régulièrement à partir d'une horloge, soit de **type discret** : le temps relatif de la simulation augmente de manière discrète de dates en dates. Un calendrier d'événements à exécuter, souvent appelé **échéancier**, se met à jour après chaque événement exécuté. En effet, l'échéancier est une liste dynamique, à la tête de la liste se trouve l'événement en cours de traitement, à la fin se trouve l'événement le plus éloigné dans le temps.

Ce type de simulateur impose donc de connaître, au début d'un traitement, sa durée. Ainsi, il sera possible de placer dans l'échéancier la date de l'événement suivant. Dans un système électronique, les temps d'exécution et de transfert sont fixes ou dépendants, l'échéancier pourra être employé pour gérer les événements.

Ces simulateurs à événements discrets ont fait leur preuve dans de nombreux domaines, comme les chaînes et processus industriels, les services administratifs, la gestion de stock, les réseaux informatiques, etc. En particulier, nous avons montré dans nos travaux antérieurs sur la modélisation des réseaux locaux [ABO90][ABO94], l'efficacité du simulateur à événements discrets SIMAN. Les problèmes rencontrés étant proches de ceux rencontrés dans les machines multiprocesseurs, nous proposons donc de l'utiliser pour l'analyse de ces derniers.

Ainsi, les principales parties du système : mémoires, bus, caches, processeurs..., sont modélisées sans atteindre le niveau électronique des composants. En effet, seul le comportement du module nous intéresse. La création et la modification d'une structure électronique sont fortement simplifiées lorsque les modèles de composants ont été établis. De plus, la notion de temps, les distributions probabilistes et la gestion des statistiques sont à la base du simulateur. Enfin, l'exécution de programmes parallèles de test peut être ajouté au simulateur grâce aux modules d'entrées-sorties disponibles sur celui-ci.

Pour une étude des performances globales d'une machine multiprocesseurs, l'approche par simulateur à événements discrets semble concilier les avantages des autres approches : notion de temps, de statistiques, de prise en compte de paramètres, sans les difficultés de l'établissement d'un modèle mathématique ou d'une saisie de schéma électronique complète, ni la laborieuse réalisation d'un simulateur dédié.

3. SIMAN : un simulateur à événements discrets

Du fait des résultats qu'il nous a permis d'obtenir sur l'étude des réseaux locaux [ABO94], le simulateur à événements discrets SIMAN (SIMulation LANguage for modeling general systems), créé par C.D. PEDGEN et conçu en 1982 par la société Systems Modeling Corporation, a été retenu comme outil d'analyse des machines parallèles pour notre étude. Nous montrerons dans la suite de cet article qu'il est bien adapté à ce type de systèmes.

SIMAN IV est actuellement commercialisé dans le logiciel **ARENA** qui regroupe les logiciels SIMAN et CINEMA pour une interface graphique améliorée [ARE94]. En effet, ARENA permet d'éditer graphiquement le modèle SIMAN par une série de blocs représentant les instructions du simulateur, liés les uns à la suite des autres pour décrire le modèle. Pour la simulation, une animation graphique CINEMA peut être créée pour la visualisation du déplacement des entités dans le modèle et de l'état des ressources, files d'attente, etc. La visualisation graphique de la simulation est utile pour la vérification du fonctionnement du modèle et pour comprendre son évolution au cours du temps.

Un modèle SIMAN est composé d'une série de **modules** et d'**éléments** [SIM94]. Les modules (ou blocs fonctionnels) représentent la succession des instructions du modèle. Les éléments correspondent à la déclaration des composants du modèle (ressources, entités, files d'attente, liaisons). La séparation des déclarations et des

instructions permet de modifier les paramètres du modèle sans modifier le modèle proprement dit.

3.1. Les éléments d'un modèle SIMAN

Les principaux éléments que nous avons utilisés pour notre modélisation sont :

- Les **entités** : ce sont les seuls éléments mobiles du modèle (instructions, données, signaux de contrôle...). Les entités sont créées, déplacées, stockées, bloquées, libérées, dupliquées ou détruites tout au long de la simulation. Pour les caractériser, des attributs, propres à chaque entité, leur sont affectés (nom, type, taille, priorité, catégorie...).
- Les **ressources** : ce sont des éléments fixes du modèle qui sont monopolisés par une entité pour un certain temps, puis libérés (processeur, mémoire, bus...). La ressource possède une capacité (en nombre d'entités), elle est caractérisée par son état : libre ou occupé (lorsqu'une entité utilise la ressource).
- Les **files d'attente** du type FIFO ou LIFO.
- Les **stations :** un modèle SIMAN est composé de plusieurs sous-modèles décrivant les différentes étapes du mécanisme étudié (ex : traitement, décodage, lecture, écriture...). Les stations constituent les points d'entrée de ces différents sous-modèles. Les entités se déplacent alors de stations en stations dans le modèle.

3.2. Les principaux modules d'un modèle SIMAN

Le modèle de simulation est créé en ajoutant les uns à la suite des autres des modules qui correspondent aux instructions du programme décrivant le modèle :

- **Create** & **Dispose :** ce module crée & détruit les entités au cours du temps.
- **Seize** & **Release** *ressource* : l'entité active prend & libère la ressource.
- **Delay** *durée* : l'entité active reste en attente pendant la durée spécifiée.
- **Queue** & **Remove** *file d'attente*: l'entité active entre dans la file d'attente & en est libérée.
- **Choose** : l'entité active va prendre aucun, un ou plusieurs chemins en fonction de boucles conditionnelles : IF THEN, ELSE, ALWAYS.
- **Assign** *attributs* : ce bloc modifie les attributs de l'entité active.
- **Wait** *n* : l'entité est bloquée tant que le signal *n* n'a pas été émis par le module **Signal** *n*.
- **Route** *point d'entrée* : l'entité est transférée au module **Station** *point d'entrée*.
- **Event** : c'est un bloc qui lance un sous programme utilisateur en langage C. Dans ce fichier utilisateur, il est possible d'intervenir sur l'état et les caractéristiques du modèle (changement de capacité, modification des statistiques, des attributs...). Le module est d'une très grande utilité lors de la phase de validation des modèles de machines sur les programmes de test.

3.3. Conclusion

Une rapide analyse d'une machine parallèle, nous montre que cette dernière peut être vue comme l'interconnexion d'un ensemble de modules SIMAN. La structure du système ainsi obtenu doit donc être figé à la compilation : le nombre de ressources, de files d'attente, les chemins et les opérateurs sont fixés à l'édition du modèle.

Les résultats de la simulation seront : les temps moyens de parcours du processus, les temps d'occupation de chaque ressource, les temps d'attente et le nombre d'entités en attente dans les diverses files d'attente, le nombre d'entités passant dans un module, etc. On pourra pour notre application obtenir les temps moyens d'accès aux mémoires, les cycles d'attente des processeurs, les pourcentages de réussite des accès mémoires, etc.

4. Validation des modèles par comparaison avec une approche analytique

Le but de cette étude est de montrer que le simulateur SIMAN répond bien aux besoins des systèmes parallèles et qu'il permet d'obtenir simplement des résultats précis et utiles. C'est pourquoi, nous avons choisi de modéliser le comportement des mémoires caches dans les machines multiprocesseurs à mémoire partagée.

Dans un premier temps nous présenterons le fonctionnement du cache pour multiprocesseurs, puis le principe de sa modélisation mathématique. Nous en détaillerons ensuite la modélisation par modèles SIMAN. Enfin, nous présenterons les résultats de l'étude comparée et nous montrerons que SIMAN peut sans difficultés lever les hypothèses simplificatrices imposées par l'approche analytique.

4.1. Présentation du problème

Un cache est une mémoire rapide placée entre les processeurs et les ressources mémoires. Le cache permet de stocker temporairement les données les plus souvent utilisées. L'intérêt des caches vient de leur temps d'accès beaucoup plus rapide que celui des mémoires conventionnelles, surtout dans un système multiprocesseur où la mémoire partagée est très sollicitée.

Dans le cas des machines multiprocesseurs, plusieurs caches peuvent obtenir une copie de la même donnée. Cependant, si un processeur modifie la donnée contenue dans son cache, cette dernière sera alors différente des autres copies. On a alors "**perte de la cohérence des données"**. De nombreuses techniques ont été mises au point pour conserver la cohérence des données. On pourra consulter l'étude comparative des mécanismes [TOU95]. Dans la suite de l'article, nous nous intéresserons au système de conservation de la cohérence des données utilisant le mécanisme de l'invalidation de blocs (Figure 1) : lorsqu'un processeur veut modifier

une donnée contenue dans son cache, toutes les autres copies du même bloc de données sont invalidées (rendues obsolètes).

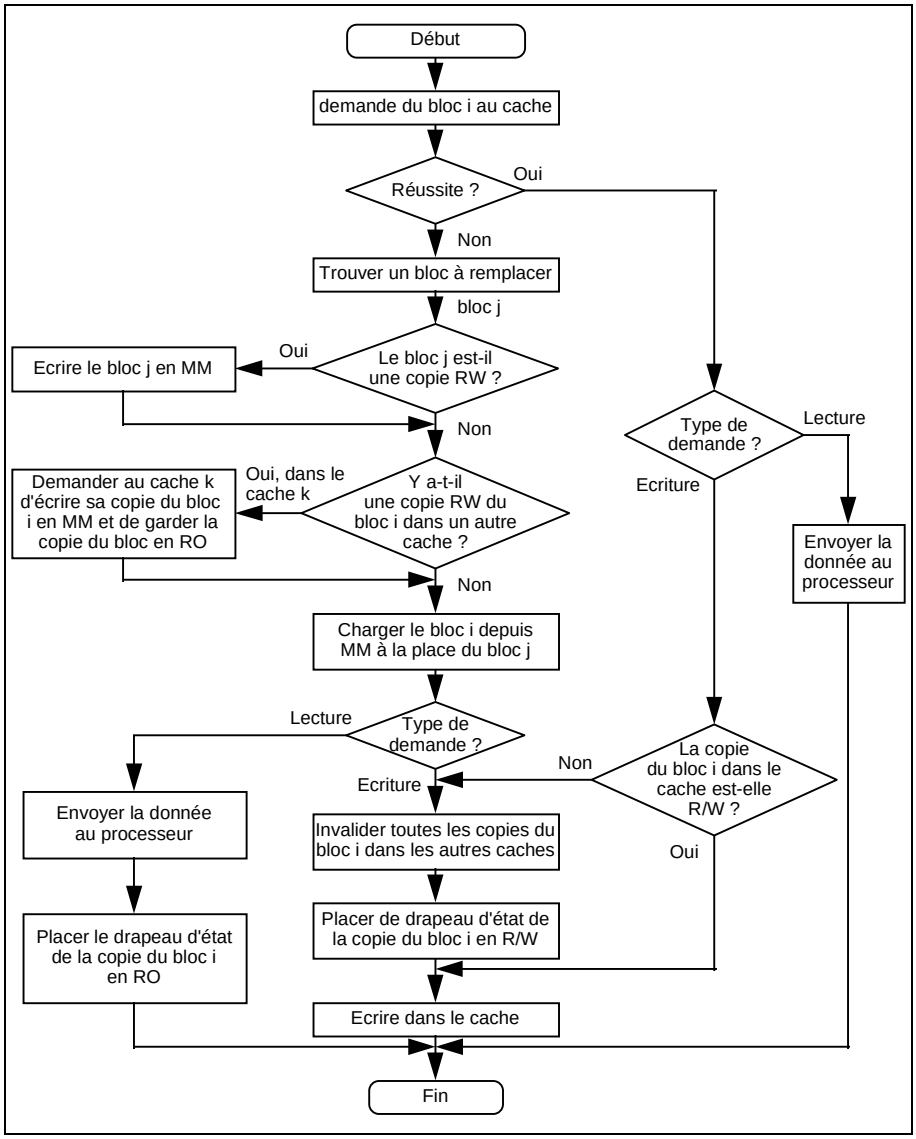

Figure 1. *Diagramme de fonctionnement du cache avec mécanisme d'invalidation*

Nous limiterons l'étude aux systèmes multiprocesseurs, à mémoire partagée, à caches privés et totalement associatifs. Un système est dit à cache privé, si chaque processeur possède son propre cache qui sert d'intermédiaire pour accéder la mémoire partagée par tous les processeurs. Un cache est dit totalement associatif si toutes les cases du cache sont identiques et indépendantes les unes des autres.

Chaque cache possède une pile contenant les numéros de blocs présents dans celui-ci et qui permet premièrement de déterminer si le bloc est présent au moment de la requête ("**Réussite**") et deuxièmement de déterminer le bloc à supprimer du cache en cas d'"**échec**". La pile est du type **LRU** (Least Recently Used Stack), ce qui signifie que le prochain bloc qui sera effacé de la pile est celui pour lequel le dernier accès au bloc est le plus ancien. Le fonctionnement interne du cache que nous avons modélisé est décrit dans la Figure 1.

4.2. Approche analytique

Cette approche mathématique [GOU96] de l'étude des performances est réalisée à partir de graphes de Markov dont les arcs sont pondérés par les probabilités de passage d'un état à un autre (Figure 2). L'étude du système de mémoires caches est tout d'abord réalisée pour le cas plus simple des machines monoprocesseurs. L'étude des dégradations dues aux autres processeurs permet de déduire de l'étude du monoprocesseur les performances du multiprocesseur.

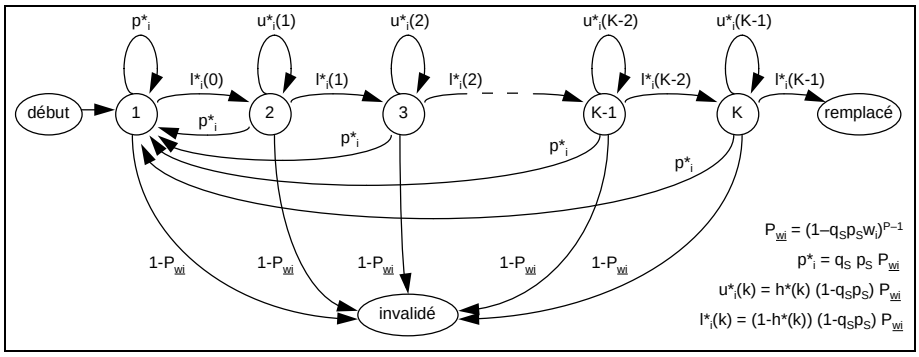

Figure 2. *Déplacement d'un bloc partagé dans la pile du cache*

Pour faciliter le modélisation mathématique du problème, on décomposera le programme à exécuter en P processus homogènes répartis sur P processeurs identiques. On distinguera deux types de blocs-mémoire accédés par les processeurs : les blocs privés et les blocs partagés.

Les lois d'accès aux processeurs sont obtenus par l'analyse d'un programme en mode trace [DUB82]. Le modèle d'appel aux blocs privés est basé sur la localité de l'exécution du programme, une loi empirique étant choisie pour p_{Pi} probabilité d'accès au $i^{ème}$ bloc privé :

$$p_{P1} = 0.599 \quad \text{et} \quad p_{Pi} = \sqrt[3]{\frac{0.008}{i - 1.893}} - \sqrt[3]{\frac{0.008}{i - 0.893}} \qquad \text{pour } i \in [2;1024]$$

Les accès aux blocs partagés sont effectués avec une probabilité fixe p_S :

$$p_S = \frac{1}{N_S}$$

On définit les paramètres suivants pour représenter les accès des processeurs : q_S la probabilité que le prochain appel soit pour un bloc partagé, w_i la probabilité que l'accès au bloc i soit en écriture, N_P et N_S respectivement les nombres de blocs de type privé et de type partagé.

$q_S = 0.1$ $w_i = 0.1$ $N_P = 1024$ $N_S = 16$

On cherche à déterminer h(K) la fonction de probabilité donnant le taux de réussite en fonction de K : taille du cache en nombre de blocs.

Dans un système multiprocesseur asynchrone, la prise en compte, pour la modélisation, des signaux d'invalidation et de mise à jour de blocs venant des autres processeurs est complexe. En effet, il faudrait garder une trace du temps réel pour connaître la position de ces événements par rapport à l'exécution de chaque processus. Il est nécessaire de faire une approximation : entre deux accès locaux, chaque processeur distant effectue une seule et unique référence. Cette hypothèse n'est jamais vérifiée (sinon le système serait totalement synchrone), mais elle reste une bonne approximation dans un programme réparti en processus homogènes.

4.3. Approche par le simulateur SIMAN

L'étude du même problème est reprise cette fois-ci par simulations à l'aide de SIMAN. Un modèle fonctionnel complet de cache pour systèmes multiprocesseurs a été réalisé. Les performances du système sont déterminées par simulations en modifiant les paramètres du système (nombre de processeurs, taille du cache...).

4.3.1. Modélisation

Nous devons créer un modèle SIMAN dont le comportement soit celui du cache défini précédemment. Les composants principaux du système sont les P processeurs, les P mémoires caches et la mémoire principale. Pour rester en accord avec le modèle analytique, on suppose pour la suite que les P processeurs et caches sont identiques.

- Modélisation des données :

Dans notre modèle, les entités représentent tantôt le bloc de données qui se déplace entre le cache et la mémoire, tantôt la requête de demande ou d'invalidation de bloc. Afin de différencier les entités, des attributs leurs sont affectés : le **numéro** du bloc de données correspondant, le **type** de partage du bloc (partagé ou privé), le drapeau **valide** (copie à jour ou copie invalidée), le drapeau du type d'accès **R/W** (R/W pour écriture ou RO pour lecture seule) et le numéro du **processeur** qui a émis la requête.

- Modélisation des processeurs :

Le fonctionnement interne du processeur ne fait pas l'objet de notre étude. Seuls les accès externes nous intéressent. Le processeur est modélisé, au sens SIMAN, par une ressource, nommée **"processeur",** monopolisée par les entités pendant la durée de l'exécution.

- Modélisation de la mémoire principale :

La mémoire principale est considérée comme unique même si, en pratique, elle est constituée d'une série de bancs mémoires juxtaposés. La mémoire principale est modélisée par une ressource unique que l'on nomme "**MP**" et qui est prise par les entités pendant la durée des accès mémoire.

A cette ressource "MP" est associée une file d'attente de type FIFO "**MP_queue**" dans laquelle sont placées les entités qui sollicitent la mémoire principale. Lorsque l'entité active libère la ressource (fin de l'accès mémoire), la première entité de la file d'attente prend automatiquement la ressource et ainsi de suite.

La table contenant l'état de tous les blocs de données contenus dans la mémoire principale est modélisée par une variable utilisateur : "**global_table**" (tableau à une dimension dont les valeurs sont, pour chaque bloc de données, le numéro du processeur qui possède la copie en écriture).

- Modélisation du cache

Le cache est constitué d'une mémoire tampon ayant une capacité de **K** blocs de données, d'une pile interne de type LRU dans laquelle sont placés les numéros de bloc et d'une table dite "locale" pour connaître l'état des blocs stockés dans le cache. Le cache est modélisé uniquement par une file d'attente FIFO "**cache_queue**" qui représente alors à la fois la mémoire tampon, la pile et la table locale.

En effet, la file d'attente, de type FIFO, est mise à jour lors des demandes de blocs ; l'entité en tête de file est la dernière accédée et l'entité se trouvant en fin de file est celle qui est restée le plus longtemps sans être accédée. Ceci permet de modéliser la pile de type LRU.

Les trois principales actions de l'unité de gestion du cache sont modélisées de la manière suivante :

- En cas d'échec, il faut supprimer le dernier bloc de la file d'attente : l'entité qui traverse le module SIGNAL[1] envoie le signal *signal_1* pour débloquer la dernière entité de la file d'attente qui était bloquée par le module WAIT *signal_1* (Figure 4). Cette dernière entité se dirige ensuite vers la station *Suppression* pour être détruite.
- En cas de réussite, le bloc est déjà dans le cache, cependant il faut le retirer de la file pour pouvoir l'utiliser : le module REMOVE *cache_F* est utilisé pour retirer une entité à une position donnée dans une file d'attente. La valeur du paramètre position dans notre application du cache est associée à une fonction utilisateur qui parcourt la file d'attente pour repérer la position de l'entité correspondante.
- Dans les deux cas, le bloc accédé doit être placé au sommet de la file d'attente.

[1] les modules prédéfinies SIGNAL, WAIT, REMOVE et QUEUE ont été définis dans le paragraphe 3.2.

L'entité qui entre dans le module QUEUE *cache_F* est placée en entrée de la file d'attente. L'entité restera dans la file d'attente jusqu'à ce qu'un des deux événements ci-dessus se présente (elle traverse la file puis se bloque au niveau du module WAIT ou elle est retiré de la file par le module REMOVE).

4.3.2. Fonctionnement du modèle de machine au sens "SIMAN"

Le système de caches multiprocesseurs peut se diviser en six étapes associées aux six stations ci-dessous, représentées dans l'organigramme de la Figure 3 et détaillées dans le modèle complet de la machine (Figure 4) :

- Station **Départ** : mise en forme de la requête de bloc de données.
- Station **Recherche** : recherche dans le cache et prise de décision.
- Station **Lecture** : lecture du bloc depuis la mémoire principale.
- Station **Passage_RO** : recopie du bloc en mémoire principale pour sa mise à jour et passage de la copie en écriture de l'état R/W à l'état RO.
- Station **Exécution** : mise à jour de la file d'attente du cache et envoi du bloc au processeur.
- Station **Suppression** : suppression du bloc à remplacer.

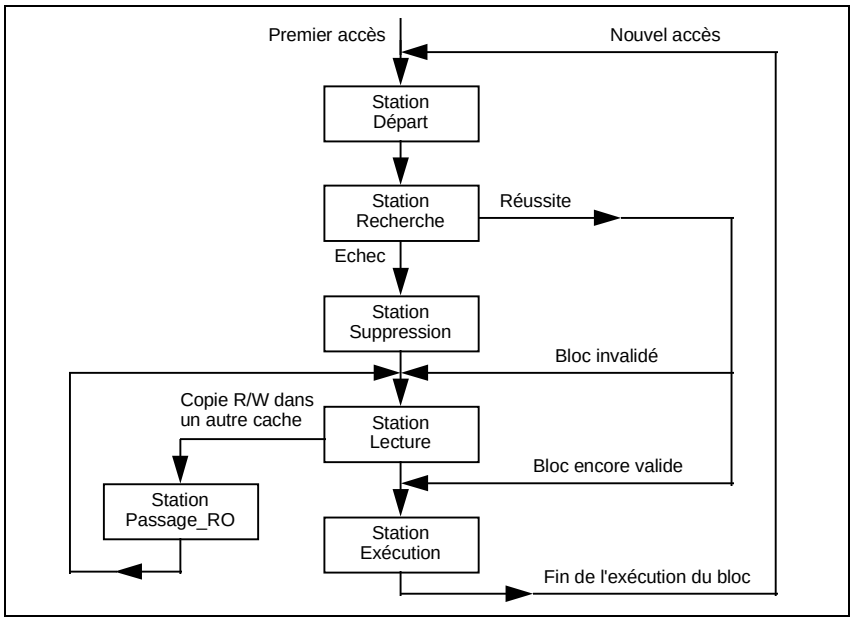

Figure 3. *Organigramme du cheminement des entités dans le modèle de cache*

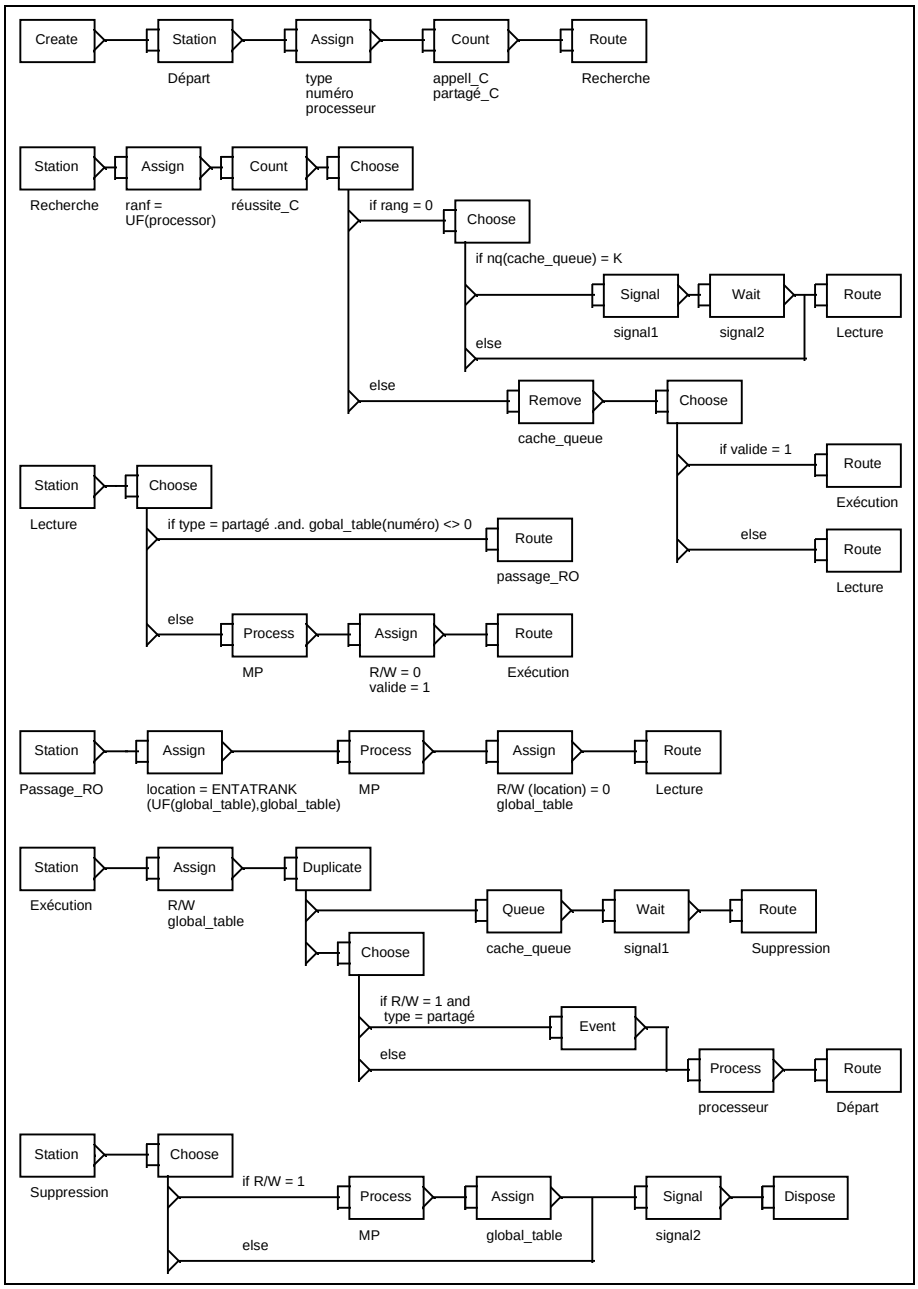

Figure 4. *Modélisation du cache par SIMAN*

4.3.3. Contexte des simulations

Afin de rester cohérent avec les hypothèses simplificatrices de l'approche analytique, on ne tient pas compte, pour les simulations, des temps de chargement, de copie, de mise à jour de blocs, ni d'exécution de blocs par les processeurs. Cependant, nous signalons que SIMAN permet sans difficultés, non seulement d'éliminer ces hypothèses en tenant compte des temps définis ci-dessus, mais aussi d'obtenir des statistiques sur ces temps de transfert.

Pour les simulations, nous avons volontairement retenu les mêmes lois d'accès aux blocs que pour la modélisation analytique du problème. Les résultats sont récupérés par les différents compteurs en utilisant le module COUNT.On obtient également, pour les blocs de type partagé, le taux d'invalidation, les taux de mises à jour de la mémoire, etc.

On effectue des simulations en faisant varier les paramètres suivants : N_S nombre de blocs partagés, q_S proportions d'accès aux blocs partagés, K taille du cache et P nombre de processeurs. Pour obtenir une bonne moyenne, on comptabilisera les accès de tous les processeurs pour obtenir un maximum d'échantillons.

4.4. Résultats et critiques

Les simulations, réalisées à partir du modèle SIMAN que nous venons de présenter, ont été confrontées au modèle mathématique de cache. Nous présentons dans ce paragraphe les résultats qui ont été obtenus. L'étude comparative a été réalisée sur deux points :

- L'étude des performances du système multiprocesseur pour vérifier l'aptitude de SIMAN et également du modèle analytique à prendre en compte les dégradations apportées par les invalidations de blocs.
- Suite à un écart croissant entre les résultats analytiques et simulés, une analyse par simulation des conséquences du mode de remplacement des blocs dans le cache a été réalisée. Elle a permis de montrer les limites de la modélisation analytique.

4.4.1. Influence du nombre de processeurs sur le taux de réussite

Pour remplir la Table 1, on calcule, pour chaque valeur du nombre de processeurs, le taux d'invalidation et le taux de remplacement de blocs puis on en déduit le taux de réussite qui tient compte des signaux d'invalidation émis par les processeurs.

Nombre de processeurs	P = 1	P = 2	P = 4	P = 8	P = 16	P = 32
Taille du cache	**Taux de réussite calculé**					
2	54,2%	54,2%	54,2%	54,2%	54,2%	54,2%
4	71,2%	71,2%	71,2%	71,2%	71,2%	71,1%
8	78,6%	78,6%	78,6%	78,5%	78,4%	78,3%
16	83,6%	83,5%	83,4%	83,1%	82,7%	82,1%
32	89,7%	89,4%	88,8%	87,8%	86,6%	85,3%
64	95,1%	94,2%	92,9%	91,1%	89,3%	87,7%
128	97,1%	96,2%	94,8%	93,0%	91,1%	89,5%

Table 1. *Performances calculées pour un cache multiprocesseur*

La Table 2 a été établie en récupérant les résultats de simulation du modèle SIMAN de cache multiprocesseur. On a effectué des simulations en faisant varier, d'une simulation à une autre, le nombre de processeurs et la capacité du cache.

Nombre de processeurs	P = 1	P = 2	P = 4	P = 8	P = 16	P = 32
Taille du cache	**Taux de réussite simulé par SIMAN**					
2	55,0%	55,0%	54,9%	54,9%	54,9%	54,9%
4	71,9%	72,0%	71,8%	71,9%	71,9%	71,9%
8	79,0%	79,1%	79,1%	79,1%	78,8%	78,7%
16	83,9%	84,0%	83,7%	83,4%	83,2%	82,5%
32	90,2%	89,7%	89,2%	88,2%	86,7%	85,4%
64	95,2%	94,3%	93,1%	91,2%	89,4%	87,8%
128	97,1%	96,0%	94,9%	93,2%	91,2%	89,7%

Table 2. *Performances simulées pour un cache multiprocesseur*

On constate que les écarts entre la simulation et le calcul proviennent plus du calcul du taux de réussite du cache monoprocesseur que pour le multiprocesseur : l'erreur pour les caches de 2, 4, 8 et 16 blocs est la même quel que soit le nombre de processeurs.

Lorsque le nombre de processeurs augmente, le taux de réussite diminue pour deux raisons : d'une part les accès aux blocs partagés peuvent aboutir à des échecs si les copies ont été invalidées, d'autre part, pour un nombre important de copies invalidées inutilisables, tout se passe comme si le cache était de taille inférieure.

4.4.2. Influence du mode de remplacement sur le taux de réussite

Le choix du bloc à remplacer en cas d'échec peut modifier le nombre de copies invalidées présentes dans le cache et donc sur le taux de réussite. Lors d'un appel à un bloc partagé dont la copie invalidée est encore présente dans le cache, deux possibilités sont envisageables pour le remplacement d'un bloc :

- On considère que l'on a un échec, on choisit donc de remplacer, le dernier bloc de la pile du cache comme l'impose l'algorithme de remplacement LRU.

- On considère que l'on a un "semi-échec", on recharge la nouvelle copie du bloc par dessus l'ancienne version invalidée du même bloc.

Si on applique la première solution, le taux de réussite du cache sera plus faible (Table 3). En effet, un nombre important d'emplacements dans le cache (contenant plusieurs fois des copies invalidées des mêmes blocs) seront utilisés par des copies obsolètes qui resteront dans le cache tant qu'elles n'auront pas traversé toute la pile. Par contre, l'algorithme de remplacement est plus simple car il ne faut pas tenir compte du cas particulier des copies invalidées déjà présentes dans le cache.

Si on applique la seconde solution, le taux de réussite est meilleur (Table 3) car il n'y aura jamais plus d'emplacements contenants des copies invalidées que de blocs partagés accessibles (N_S).

Taux de réussite des caches multiprocesseurs						
taille du cache	**4 processeurs**			**32 processeurs**		
	Remplacement de la copie invalidée		Modèle analytique	Remplacement de la copie invalidée		Modèle analytique
	Non	Oui		Non	Oui	
2	54,8%	54,9%	54,2%	54.9%	54,9%	54,2%
8	79,0%	79,1%	78,6%	78.6%	78,7%	78,3%
32	89,0%	89,2%	88,8%	84.9%	85,4%	85,3%
128	94,4%	94,9%	94,8%	88.3%	89,7%	89,5%

Table 3. *Influence du choix du bloc à remplacer en cas d'échec pour les accès aux blocs partagés invalidés déjà présents dans la cache : remplacement de la copie obsolète invalidée ou remplacement du dernier bloc de la pile LRU*

Dans le modèle analytique, l'accès aux blocs privés a été considéré indépendant du nombre de processeurs afin de simplifier la modélisation. Ceci n'est valable que si on néglige la baisse du taux de réussite due au nombre important d'emplacement du cache contenant des copies invalidées. Ce cas correspond donc à la deuxième solution présentée, la modélisation de la première étant un peu difficile à effectuer.

4.5. Convergence des résultats

Comme pour toute autre simulation, les résultats obtenus ne peuvent être considérés comme fiables que si le système est stable. Aussi, il nous a paru nécessaire de mener une étude de convergence ayant pour but d'évaluer le nombre d'accès nécessaire pour que notre système se stabilise et que les statistiques soient fiables.

Lorsque le nombre de processeurs augmente, le nombre total d'accès augmente d'autant. On pourra donc réduire le nombre d'accès par processeur pour conserver une durée de simulation raisonnable, tout en gardant un nombre d'échantillons correct. On a tracé la courbe de remplissage des files d'attentes "cache_queue" et la courbe d'évolution du taux de réussite du système à 32 processeurs (Figure 5). Une

brève analyse de ces courbes nous permet de dire que le cache peut être considéré comme plein à partir de 4000 accès et que les statistiques telles que les taux de réussite, invalidations... sont stables à partir de 50 000 accès processeurs.

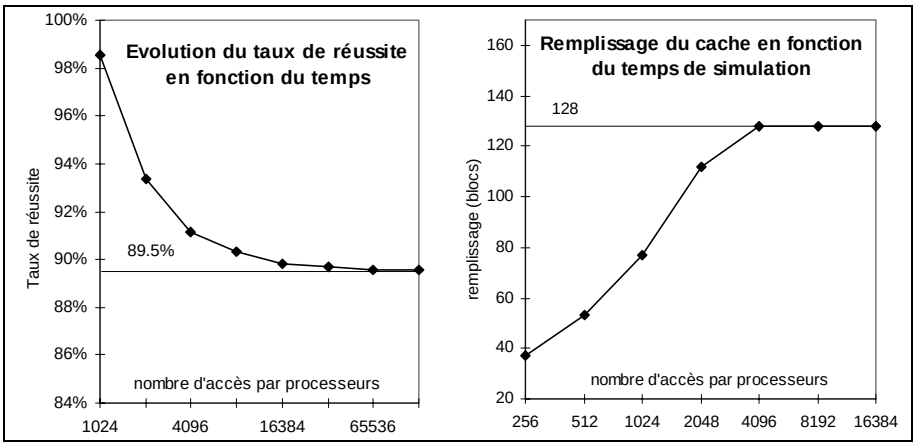

Figure 5. *Evolution du remplissage du cache et du taux de réussite au cours de la simulation pour un système de 32 caches de 128 blocs*

5. La validation des modèles par programmes réels

L'estimation des performances des machines multiprocesseurs pose de nombreux problèmes liés à la concurrence et aux communications dans les algorithmes parallèles. Pour prévoir les performances d'une machine multipro-cesseur, il est nécessaire de faire des simulations sur des programmes parallèles bien réels et non sur une simple modélisation des accès. Un intérêt principal des simulateurs est donc la possibilité d'exécuter des programmes réels. De nombreux concepteurs de machines proposent un certain nombre de programmes parallèles de test pour des applications aussi variées que possible [BER89][WOO95].

L'entrelacement de l'exécution du programme de test avec la simulation de la machine cible peut s'effectuer à deux niveaux : Il en résulte donc deux principales catégories de simulateur :

5.1. Simulation à partir de la trace d'exécution

Un fichier contenant la trace de l'exécution du programme est placé en entrée du simulateur pour prédire les performances d'un modèle de la machine.

Ces simulations sont relativement simple, efficaces et reproductibles mais présentent plusieurs inconvénients. Le principal est l'absence de rétroaction entre la simulation et l'exécution du programme qui a été réalisée à priori. Ainsi, les durées des échanges de données dans l'environnement parallèles ne seront pas pris en

compte dans la simulation du programme alors que ceux-ci peuvent influencer la suite de l'exécution du programme. On peut dire que cette simulation se limite aux machines dont le comportement est proche de celui de la machine qui a fourni l'exécution initiale. Un autre inconvénient est l'important volume disque nécessaire pour garder les traces des exécutions de tous les processeurs.

On pourra citer à titre d'exemple le simulateur TRAPEDS [STU89] qui simule le passage de messages à partir des traces du calculateur intel PSC/2. Cependant, il ne permet malheureusement pas la modélisation des mémoires partagées.

5.2. Simulation d'exécution de programmes

Ce type de simulation résout les problèmes précédents en permettant la simulation au fur et à mesure de l'exécution du programme. Ainsi, le simulateur contrôle l'exécution du programme comme si elle était réalisée sur la machine modélisée. On peut décomposer le noyau de ce type de simulateur en deux parties : le **générateur de références mémoire** qui gère l'exécution des différents processus et s'arrête sur chaque référence mémoire et le **simulateur de machine** qui permet la modélisation de la réaction des hiérarchies mémoires et des réseaux de communication de la machine face aux événements (Figure 6).

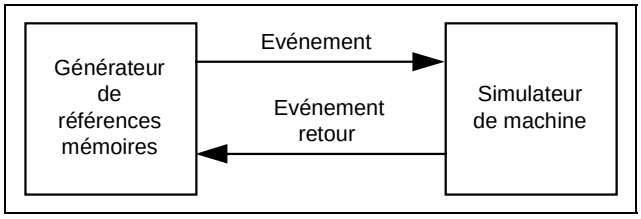

Figure 6. *Principe d'un simulateur sur programme modifié*

Cependant, ce type de simulateur nécessite l'ajout d'instructions d'appels dans le programme à simuler, ces instructions étant reconnues par le générateur de références. Cette technique nécessite donc la recompilation du programme pour y introduire les appels du générateur de références.

On peut citer à titre d'exemple les simulateurs suivants :

- le simulateur PROTEUS permettant de simuler des machines multiprocesseurs à mémoire partagée, pour des applications en C à recompiler pour placer les annotations avant chaque référence partagée [BRE91].
- le simulateur TANGO permettant de simuler les machines multiprocesseurs à mémoire partagée, il utilise les processus UNIX pour représenter les différents processeurs, ce qui simplifie la modélisation du parallélisme mais complique la gestion des références [DAV90].
- le simulateur MINT qui permet la simulation des machines multiprocesseurs à mémoire partagée. Il exécute directement le code machine du processeurs MIPS

R3000 sans recompilation et gère dans un espace mémoire unique la répartition des mémoires et la commutation des taches [VEE94].

Néanmoins, l'inconvénient principal de tous ces simulateurs est qu'ils nécessitent une modélisation de la machine cible sous forme d'un programme décrivant le fonctionnement et la réponse de la cible aux diverses sollicitations.

Il serait donc intéressant que le simulateur SIMAN ne soit pas limité à la simulation de modèles d'exécution et qu'il puisse estimer le comportement d'un modèle de machine vis à vis d'un programme de test. Cette adaptation est possible au travers du module EVENT, nous allons la détailler dans le paragraphe suivant.

6. Analyse des performances de la carte 3DTrans

Dans cette partie, nous allons étudier les performances du simulateur SIMAN pour la modélisation de l'exécution d'un programme de test. Cette étude est basée sur une petite architecture multiprocesseur : la carte 3DTrans développée par le laboratoire LISA [TIX93] que nous allons présenter. Nous présenterons ensuite les résultats que nous avons obtenus lors de la simulation sur le modèle SIMAN d'un programme de convolution.

6.1. Description de la carte

Cette carte contient trois transputers INMOS T805 [INM88]. Chaque transputer est connecté aux autres via les liens bidirectionnels disponibles sur les transputers (Figure 7). Le Transputer Maître T0 est également connecté à l'ordinateur hôte par son troisième lien. Les deux Transputers Esclaves T1 et T2 peuvent accéder trois types de mémoires : la mémoire interne rapide, la Mémoire Locale et la Mémoire Partagée. En fait, cette dernière, de type RAM dynamique, rafraîchie périodiquement, est partagée par les T1 et T2 et par l'ordinateur hôte.

Dans toute la suite on choisira le cycle-processeur comme unité de temps. La durée des accès-mémoire dépend de la localisation de la donnée dans la hiérarchie mémoire :

- Mémoire interne : accès rapide 2 cycles.
- Mémoire Locale : accès assez rapide 4 cycles (mémoire statique).
- Mémoire Partagée : accès lent 8 cycles (mémoire dynamique).

Lors des accès en mémoire partagée, les contrôleurs-mémoire génèrent des cycles d'attente (4 cycles minimum) pour stopper le transputer pendant le temps nécessaire à l'obtention du droit d'accès à la mémoire partagée et la mise en forme des adresses lignes puis colonnes pour l'accès en dynamique. Cette durée n'est pas constante puisqu'elle dépend de l'état de la mémoire lors de la requête.

Les conflits d'accès à la mémoire partagée sont résolus par un mécanisme de priorités fixes :

- Les cycles de Rafraîchissement ont la plus haute priorité (priorité 0).
- Les accès depuis l'ordinateur hôte possèdent la priorité 1.

- Les accès depuis le transputer T1 possèdent la priorité 2.
- Les accès depuis le transputer T2 ont la plus faible priorité (priorité 3).

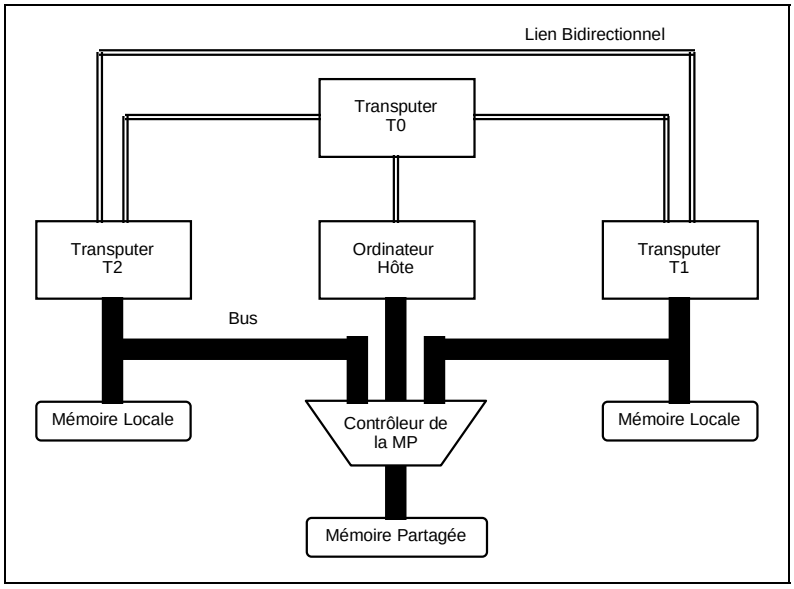

Figure 7. *Synoptique de la carte 3DTrans*

Chaque transputer possède une pile de tâches pour permettre une exécution multitâches au niveau du transputer. Le langage OCCAM est utilisé pour implanter des programmes parallèles sur les transputers.

6.2. Modélisation du réseau de transputers

Les transputers sont modélisés par des ressources SIMAN : **T0**, **T1** et **T2**.

La mémoire partagée est modélisée par une ressource unique **MP** et une file d'attente **MP_F** en entrée. Cependant, l'organisation de la file d'attente n'est pas du type FIFO classique, les entités y sont classées par ordre décroissant de l'attribut **priorité**.

Les liens bidirectionnels sont modélisés par deux ressources (une par sens) appelées **L1_2**, **L2_1**, **L1_3**, etc. Les signaux (au sens de SIMAN) sont utilisés pour bloquer et débloquer les transputers qui reçoivent ou émettent des données au travers des liens (Figure 8).

Le module EVENT[2] est utilisé pour appeler une routine en langage C qui parcourt un fichier dans lequel sont stockés, pour chaque transputer, les types d'accès et le nombre de cycles-processeur entre ces accès. Ainsi, de petits algorithmes

[2] module SIMAN, voir présentation de SIMAN.

peuvent être exécutés sur notre modèle pour prévoir les temps d'exécution, les taux de partage et d'encombrement de la mémoire partagée, l'encombrement des liens, etc.

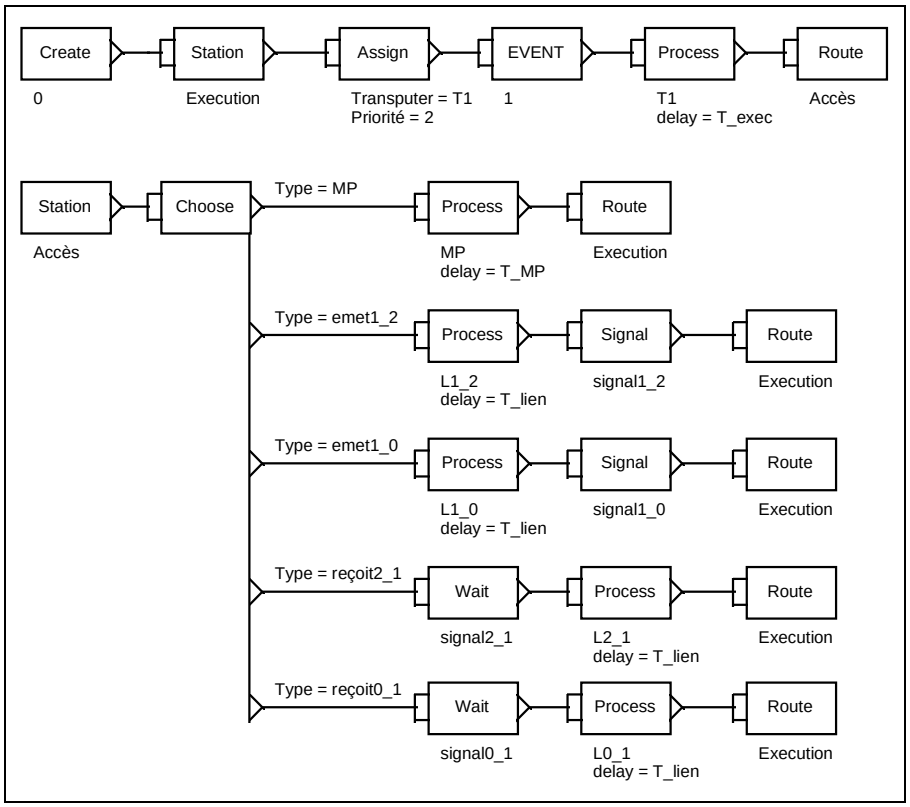

Figure 8. *Modèle SIMAN du comportement du transputer T1*

Pour obtenir ce fichier de trace, il faut utiliser un analyseur de code-machine pour Transputer capable de lire le fichier exécutable créé par le compilateur OCCAM et adapter notre modèle SIMAN au moyen du noeud EVENT.

Le transputer étant un processeur de type RISC, son code-machine est relativement intelligible du fait du nombre limité d'instructions et de leur codage sur un seul octet.Un programme a donc été développé en langage C pour parcourir le fichier exécutable dans le but de repérer les accès en mémoire partagée (instructions 'store', 'load' et 'move') et de compter le nombre de cycles-processeurs associé à chaque instruction [TIS92]. Ces informations sont stockés dans chaque fichier de trace d'exécution associé à chaque transputer.

Les boucles itératives et les boucles conditionnels sont difficiles à analyser car on ne peut pas connaître la destination du pointeur d'instructions sans calculer toutes les variables et le contenu des registres du transputer. C'est pourquoi, la version actuelle du programme permet uniquement l'emploi des boucles dont le nombre d'itérations

est fixé à la compilation et interdit l'emploi de boucles conditionnelles. Cette restriction limite le champ d'application du simulateur aux algorithmes réguliers.

6.3. Exploitation et résultats

Dans ce paragraphe, nous allons présenter les résultats obtenus d'une part par simple exécution sur la carte 3DTrans et d'autre part par simulation avec SIMAN. Au travers de ces résultats, nous chercherons tout d'abord à vérifier le fonctionnement de notre analyseur et de son interface avec le module EVENT puis à mettre en évidence les conflits d'accès au niveau de la mémoire partagée. Nous montrerons ensuite le type d'informations que SIMAN peut apporter sur le comportement de la machine.

6.3.1. Vérification de la chaîne de simulation d'un programme de test

Afin de vérifier l'exactitude de notre analyse, on procède à une série de simulation dont le but est de comparer les temps de simulation avec ceux obtenus lors de l'exécution réelle. Cette analyse est réalisée à partir d'un programme de convolution d'images. L'image à traiter (tableau 64*64 pixels) est placée soit en Mémoire Partagée, soit en Mémoire Locale. La Table 4 permet donc de mesurer l'impact des instructions de transfert de blocs entre la Mémoire Partagée et la Mémoire Locale.

Routines	**Simulation sur SIMAN**	**Exécution sur 3DTrans**
T1 ou T2 : Programme de Convolution (Image à traiter placée en Mémoire Locale)	245400	245400
T1 : Programme de Convolution (Image à traiter placée en Mémoire Partagée)	255600	255900
T2 : Programme de Convolution (Image à traiter placée en Mémoire Partagée)	253600	254000

Table 4. *Résultats comparés de simulation de la carte 3DTrans (cycles-processeur)*

Une simple analyse de ce tableau montre que l'augmentation du temps d'exécution est due aux cycles d'attente liés à l'accès à la Mémoire Partagée. Cette attente est d'autant plus importante pour le processeur T1 qui met un cycle de plus par accès, c'est à dire 2048 cycles pour une image 64*64 pixels.

6.3.2. Influence du partage de la Mémoire

Pour faire apparaître les effets du partage de la mémoire, on exécute un autre traitement sur l'autre Transputer Esclave. On choisit un programme gourmand en accès mémoire (accès répétés en lecture ou en écriture).

La Table 5 montre l'impact des instructions de transfert en Mémoire Partagée sur le programme de convolution sur un tableau 64*64 placé en Mémoire Partagée.

Routines	Simulation sur SIMAN	Exécution sur 3DTrans
T1 : Programme de Convolution T2 : Pas d'accès en MP	255600	255900
T1 : Programme de Convolution T2 : Accès répétés en MP (lecture)	257500	257500
T1 : Programme de Convolution T2 : Accès répétés en MP (écriture)	256800	257000

Table 5. *Résultats de simulation de la carte 3DTrans, influence du partage de la mémoire(cycles-processeur)*

L'analyse du tableau montre que le partage de la mémoire principale peut augmenter le temps d'exécution jusqu'à 5000 cycles supplémentaires (soit 2 % du temps initial du traitement). Cette valeur est cependant assez faible car le temps de calcul est majoritaire dans cette application 90 % du temps d'exécution.

6.3.3. Impact des accès-mémoire sur les performances de la machine

L'exécution de ce programme de convolution est assez périodique avec 400 cycles d'accès mémoire en moyenne puis environ 3600 cycles de calcul pour chaque ligne. Si le programme est placé sur les deux Transputers, le temps d'exécution pourra varier suivant si les deux Transputers effectuent leurs accès en même temps ou de manière différée.

L'environnement graphique de l'outil ARENA, permet de suivre graphiquement l'évolution de la simulation grâce à l'animation CINEMA associée au simulateur SIMAN. En effet, CINEMA nous permet de visualiser le mouvement des entités dans le modèle et l'état des ressources. En particulier, lors de la simulation du programme de convolution, on observe que les deux processeurs effectuent leurs transferts de blocs aux mêmes instants, cette synchronisation étant conservée pendant la totalité de l'exécution.

Pour éviter les conflits d'accès, il faut donc, dès le début de l'exécution, retarder de 2000 cycles l'un des Transputers afin d'éviter le chevauchement des transferts. En effet, la Table 6 montre qu'en retardant le plus rapide des deux Transputers (T2), on obtient des performances meilleures.

Routines		Simulation sur SIMAN	Exécution sur 3DTrans
T1 seul : Programme de Convolution		255600	255900
T2 seul : Programme de Convolution		253600	254000
T1 et T2 : Programme de Convolution	T1 :	257300	257500
(Synchronisation au départ)	T2 :	257400	257400
T1 et T2 : Programme de Convolution	T1 :	255700	256000
(T2 retardé de 2000 cycles)	T2 :	255900	256000

Table 6. *Résultats de simulation de la carte 3DTrans, impact des accès-mémoire sur les performances de la machine (en cycles-processeur)*

Une analyse approfondie des accès-mémoire pour cet algorithme permet de justifier ces résultats. En effet les transferts Mémoire Partagée - Mémoire Locale se font en : 8 + 3 = 11 cycles pour T1 et 7 + 3 = 10 cycles pour T2.

Dans les deux cas, la Mémoire Partagée est monopolisée pendant 6 cycles par transferts. Donc, si les deux Transputers effectuent leurs transferts en même temps, ils vont prendre la ressource chacun à leur tour, ce qui imposera une durée fixe pour les transferts : 6 + 6 = 12 cycles pour un transfert de T1 et un transfert de T2.
On retrouve donc le résultat ci-dessus (3ème ligne) : le Transputer T1 met 2000 cycles supplémentaires ((12–11) cycle par accès * 2000 accès) tandis que le Transputer T2 met 4000 cycles supplémentaires ((12–10) cycles par accès * 2000 accès). Les temps d'exécution sont alors identiques.

Par contre, si l'exécution sur T2 est retardée de 2000 cycles, il n'y a plus de conflits (4ème ligne du tableau), T1 met donc environ 255600 cycles tandis que T2 a besoin de 253600 + 2000 = 255600 cycles pour le traitement. Il semble donc plus performant, pour le mécanisme de transfert de données vers la Mémoire Partagée, de mettre en place une priorité par blocs de transfert au lieu d'une priorité pour chaque accès, ce qui est implanté actuellement.

7. Conclusion

Dans cet article, nous avons brièvement exposé une synthèse des méthodes d'évaluation des performances des machines parallèles afin de montrer l'intérêt et la portée de la simulation à événements discrets. En exploitant les possibilités offertes par le langage de simulation SIMAN, nous avons montré combien ce dernier pouvait très bien être adapté à la simulation des machines parallèles, en surmontant les difficultés de la modélisation mathématique, de la laborieuse saisie de schémas électroniques ou du développement d'un simulateur spécifique.

Dans un souci de validation de notre approche de l'étude des machines parallèles, un problème concret a été étudié : un système de mémoires caches pour les architectures multiprocesseurs. L'utilisation de SIMAN nous a permis de bâtir très

rapidement un modèle de machine multiprocesseur afin d'étudier le problème énoncé ci-dessus.

La comparaison des résultats obtenus par simulation avec ceux obtenus à partir de la modélisation analytique du même problème nous a montré l'exactitude et la fiabilité des modèles établis. Dans un deuxième temps, nous avons montré que nous pouvons sans difficulté, lever certaines hypothèses simplificatrices sans lesquelles l'approche analytique resterait très difficile. En effet, il est tout à fait possible de réaliser des simulations à partir de modèles de processeurs et de programmes hétérogènes en tenant compte de tous les temps d'exécution et de communication sans modifier le modèle développé.

D'autre part, afin de valider une architecture machine sur un programme précis ou même d'estimer les performances d'un algorithme, la simulation de programmes réels peut facilement être ajoutée au simulateur grâce au module EVENT.

Pour vérifier cette affirmation, nous l'avons appliquée sur une carte réelle à base de trois Transputers : la carte 3DTrans. Un analyseur de code pour Transputers a été développé pour analyser un programme parallèle et fournir au simulateur SIMAN les traces d'exécution. Pour simplifier sa mise au point, le champ d'applic

ation de l'analyseur a été limité aux programmes réguliers n'utilisant pas de sauts conditionnels.

Les résultats comparés entre les simulations d'un programme de test sur le modèle de la carte et l'exécution réelle ont confirmé l'exactitude et la précision de cette approche. L'animation graphique a permis de mieux comprendre comment se déroulait l'exécution et donc d'optimiser celle-ci pour réduire les conflits d'accès aux ressources.

Bien que la validation de la chaîne d'analyse ait été effectuée sur une machine à base de trois Transputers, nous pensons, vus les modèles établis, qu'elle peut sans difficultés être appliquée sur des architectures plus complexes.

La simulation étant limitée à l'état fonctionnel des principaux éléments de la machine, des résultats peuvent être obtenus assez rapidement en comparaison avec d'autres approches plus fidèles.

8. Bibliographie

[ABO 90] N. ABOUCHI, Analyse et mesure de performance des réseaux de communication par simulation, thèse de doctorat, INSA Lyon, France, n° 90ISAL0057, 1990.

[ABO 94] N. ABOUCHI, R. GRISEL, « SLAM based modelization tool for communications networks », *Proceedings of the IEEE Internationnal Conference of Computer Technology*, Singapour, p 353-356, Aug. 1994.

[ARE 94] *Guide de références - ARENA / Variables*, Systems Modeling Corporation, Sewickley PA, 1994.

[BER 89] M. BERRY, D. CHEN, P. KOSS, D. KUCK, S. LO, *The perfect club benchmarks : Effective performance evaluation of supercomputers*, Univ of Illinois, Urbana IL, CSRD rep. 827, May 1989.

[BRE 91] E. A. BREWER, C. N. DELLAROCAS, A. COLBROOK, W. E. WEIHL, *PROTEUS : a high-performance parallel-architecture simulator*, Technical Report MIT/LCS/TR-516, MIT, Sept. 1991.

[CER 88] A. CERNAULT, *La simulation des systèmes de production*, Cepadues Editions, Toulouse, 1988.

[DAV 90] H. DAVIS, S. R. GOLDSCHMIDT, J. HENNESSY, *TANGO : A Multiprocessor simulation and tracing system*, Technical Report CSL-TR-90-439, Stanford University, 1990.

[DAV 91] H. DAVIS, S. R. GOLDSCHMIDT, J. HENNESSY, « Multiprocessor simulation and tracing using TANGO », *Proceedings of the 1991 Conference on Parallel Processing*, vol. II, p. 99-107, Aug. 1991.

[DUB 82] M. DUBOIS, F. A. BRIGGS, « Effects of cache coherency in multiprocessors », *IEEE Transactions on Computers*, vol. C-31, num. 11, Nov. 1982.

[GOU 96] R. GOULLIOUD, N. ABOUCHI, R. GRISEL, « SIMAN : A discrete events simulator for performance analysis of parallel computers », *Proceedings of Telecommunication Distribution Parallelism 96*, p. 383-397, June 1996.

[INM 88] *The Transputer Databook*, INMOS Limited, Bristol UK., 1988.

[JAC 93] J. L. JACQUEMIN, *Informatique parallèle et systèmes multiprocesseurs*, Editions Hermès, Paris, 1993.

[SIM 94] *Guide de references - SIMAN V*, Systems Modeling Corporation, Sewickley PA, 1994.

[SIN 92] J. P. SINGH, W. D. WEBER, A. GUPTA, « SPLASH : Stanford parallel applications for shared memory », *Computer architecture news*, vol. 20, p. 5-44, Mar. 1992.

[STU 89] C. B. STUNKEL, W. KENT FUCHS, « TRAPEDS : producing traces for multicomputers via execution driven simulation », *performance evaluation review*, vol. 17, p. 70-78, May 1989.

[TIS 92] *Transputer Instruction Set*, INMOS Limited, 1992.

[TIX93] T. TIXIER, *Machine de traitement et d'analyse d'image 3D - propositions de choix*, rapport de Mastère Image et Matériaux, ICPI Lyon - ENTPE, 1993.

[TOU 95] F. M. TOUSSI, D. J. LILJA, « The potential of compile-time analysis to adapt the cache coherence enforcement startegy to the data sharing caracteristics », *IEEE transactions on parallel and distributed systems*, vol. 6, p. 470-481, May 1995

[VEE 94] J. E. VEENSTRA, R. J. FOWLER, « MINT : A front end for efficient simulation of shared-memory multiprocessors. », *Proceedings of MASCOTS'94*, 1994.

[WOO 94] M. E. WOODWARD, *Communication and computer networks - Modelling with discrete-time queues*, IEEE Computer Society Press, Los Alamitos, California, 1994.

[WOO 95] S. C. WOO, M. OHARA, E. TORRIE, J. P. SINGH, A. GUPTA, « The SPLASH-2 programs : Characterization and methodological considerations », *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, p. 24-36, June 1995.

Renaud Goullioud *est diplômé de l'Institut de Chimie Physique Industrielles de Lyon en 1994. Il prépare actuellement une thèse sur la modélisation des échanges de données dans les machines parallèles à l'Ecole Supérieure de Chimie Physique Electronique de Lyon (CPE-Lyon).*

Nacer Abouchi *a soutenu en 1990 à l'INSA-Lyon une thèse sur la modélisation des réseaux locaux. Depuis 1990, il enseigne l'électronique à l'Ecole Supérieure de Chimie Physique Electronique de Lyon. Ses travaux actuels portent sur les systèmes électroniques pour le traitement d'images.*

Richard Grisel *a soutenu en 1987 à l'Université de Rouen une thèse en électronique et en 1994 une habilitation à diriger des recherches à l'Université de Saint-Etienne. Il est actuellement à la tête du groupe de recherche sur les systèmes électroniques et microélectroniques pour le traitement d'images au sein du laboratoire LISA (EP CNRS 0092) de l'Ecole Supérieure de Chimie Physique Electronique de Lyon (CPE)*