Réduction de l'influence du placement mémoire par la synthèse de haut niveau

Gwenolé Corre, Nathalie Julien, Eric Senn, Eric Martin

▶ **To cite this version:**

Gwenolé Corre, Nathalie Julien, Eric Senn, Eric Martin. Réduction de l'influence du placement mémoire par la synthèse de haut niveau. FTFC'05 (journées d'études Faible Tension Faible Consommation), 2005, Paris, France. hal-00077388

HAL Id: hal-00077388
https://hal.science/hal-00077388

Submitted on 30 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Réduction de l'influence du placement mémoire par la synthèse de haut niveau

Gwenolé Corre, Nathalie Julien, Eric Senn, Eric Martin
LESTER, Université de Bretagne Sud
Centre de recherche - BP 92116 - 56321 LORIENT Cedex
mailto : prenom.nom@univ-ubs.fr
Tel : +33 (0)2 97 87 45 28 - Fax : +33 (0)2 97 87 45 00

Résumé :**La consommation des circuits électroniques augmente à mesure que les applications de traitement du signal et de l'image manipulent de plus en plus de données. Dans ce contexte, la consommation des mémoires représente une grande proportion de la consommation totale des circuits [1]. Il est alors primordial de concevoir des unités de mémorisation permettant de limiter le coût en consommation en introduisant des techniques de conception et de gestion mémoire à haut niveau. Nous définissons une technique de gestion des accès mémoire par kanban inspirée de la gestion de production. Cette technique nous permet de limiter l'influence du placement des données sur la consommation.**

Mots Clés: Consommation, placement mémoire, synthèse de haut niveau sous contraintes.

I. INTRODUCTION

Dans le domaine du temps réel et des applications utilisant de grande quantité de données, les ressources de calcul et de mémorisation doivent intégrer l'augmentation du flux de données [2]. La conception d'architecture ou de système doit se concentrer sur (1) la réduction de la consommation, (2) la réduction du coût de stockage des données et (3) le respect de contraintes temporelles strictes en terme de cadence et de latence. La conception doit s'appuyer sur une modélisation des contraintes d'E/S et de mémoires de données internes, une analyse des contraintes pour vérifier la faisabilité et l'utilisation de la synthèse de haut niveau pour générer l'architecture du composant.

Dans [3], nous avons introduit une approche prenant en compte l'architecture des mémoires et le placement des données lors de la synthèse de haut niveau. Une table mémoire est extraite à partir du *SFG* ; tous les nœuds de données du *SFG* sont présents dans la table mémoire. Le concepteur peut distribuer et placer les données de l'application et définir un mapping mémoire. Le choix des mémoires s'effectue dans la phase de sélection. A partir de la table mémoire, nous construisons un Graphe de Contraintes Mémoire, *MCG*. Il représente tous les conflits d'accès aux données et les possibilités d'ordonnancement des données placées en mémoire. Le *MCG* est utilisé pour résoudre les conflits d'accès aux données placées en mémoire interne lors de l'ordonnancement. Des informations complémentaires sur les modèles formels et la conception des mémoires peuvent être trouvées dans [3] et [4]. Dans cet article, nous proposons un flot de conception basé sur des modèles formels qui autorise la synthèse de haut niveau d'applications TDSI sous contrainte de mémorisation en introduisant un mécanisme de gestion des accès mémoire basé sur le modèle de gestion par kanban utilisé dans les système de production [5] et [6]. La gestion par kanban permet de gérer simplement l'anticipation des accès aux données et de réduire l'influence du placement des données en mémoire sur la consommation globale.

II. GESTION PAR KANBAN

A. Le modèle de gestion par kanban

La méthode Kanban est applicable à des productions de type " masse " pour lesquelles le nombre de références n'est pas trop élevé et la demande régulière ou à faibles variations. Nous retrouvons ces propriétés dans les applications TDSI et dans les unités de mémorisation qui vont être spécifiées. Cette similitude nous permet de transposer le modèle de gestion kanban à la gestion des accès mémoire dans la synthèse de haut niveau.

L'introduction de kanban doit, à partir d'un modèle matériel de l'unité de traitement, proposer une gestion de l'ordonnanceur avec anticipation des accès aux données.

Le modèle matériel doit répondre à deux problèmes. D'un coté, il doit permettre de stocker toutes les données dont les accès ont été anticipés. Il faut augmenter le nombre de registres internes de l'unité de traitement. De l'autre, le coût en surface et en consommation de l'unité de traitement doit être maîtrisé. Il faut limiter le nombre de registres de l'unité de traitement. Le modèle matériel de l'unité de

traitement permettant de stocker les données anticipées contient exclusivement des registres. La taille définie pour les différentes files et listes de gestion de l'ordonnanceur permettra de déterminer le nombre maximum de registres alloués pour stocker les données dont les accès ont été anticipés.

La gestion de l'ordonnanceur doit permettre de réguler le nombre de données présentes dans l'unité de traitement dont les accès à la mémoire ont été anticipés. Pour ce faire nous introduisons une gestion des lectures et des écritures en mémoire. Les files et listes qui seront définies dans la section C servent uniquement à la gestion de l'ordonnanceur et ne sont pas implanter matériellement dans l'unité de traitement.

B. *Condition de mise en oeuvre*

L'anticipation des lectures et les accès en écriture doivent être cohérents vis à vis de l'ordonnancement des calculs (algorithme list-scheduling). Cette cohérence est garantie par le tri des données à lire et à écrire. Le tri est réalisé statiquement en utilisant les dates ASAP puis ALAP après ordonnancement préalable des nœuds du graphe flot de signaux. Nous fixons statiquement l'ordre des accès en lecture et en écriture en fonction :

- de la distribution des données dans les différentes mémoires allouées.
- de la distribution des données sur les différents types d'opération sélectionnées pour réaliser les calculs de l'unité de traitement.

Nous proposons un formalisme permettant de définir l'ordre des lectures et des écritures de toutes les données placées en mémoire.

Pour cela, nous introduisons de files et des listes permettant de trier les données et fixer l'ordre et de déterminer le nombre de données dont les transferts seront anticipées. Elles servent à la gestion des transfert mémoire en lecture et en écriture.

C. *Définition des files*

La mise en œuvre de la gestion de type kanban dans l'ordonnancement des opérations amène à la définition de files ou de listes, Figure 1. Elles servent uniquement à la gestion de l'ordonnanceur ; elles ne sont pas implantées dans l'architecture de l'unité de traitement. Les différentes files et listes utilisées sont :

- Les files permettant de gérer l'ordre des accès en lecture à la mémoire sont appelées files *AL_M*, nous reviendrons sur le classement des données dans la file *AL_* Les files, notées *AL_S*, contiennent les données à lire en fonction du type d'opération dont elles sont les opérandes files.
- Les listes *L_S* contiennent les données qui ont été lues en mémoire suivant le type d'opération à exécuter et dont les opérations n'ont pas été ordonnancées.
- Les listes *AE_M* contiennent tous les résultats fournis par les opérations ordonnancées et qui doivent être placés.
- Les files *E_M* contiennent les données qui ont été produites par l'unité de traitement, les données contenues dans cette file vont être réinjectées dans les files *AL_M* lorsque les données doivent à nouveau être lues dans l'itération courante de l'algorithme

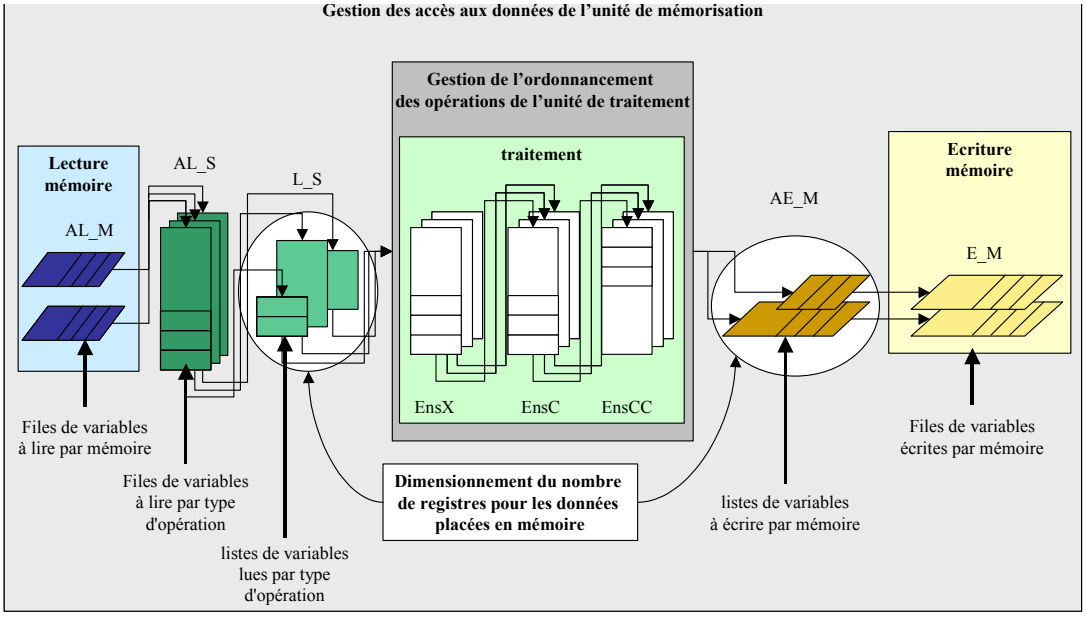

Figure 1 : Introduction de files pour gestion kanban.

Ces nouvelles files et listes devront permettre d'anticiper les accès mémoire dans le temps et de réduire la complexité de l'unité de mémorisation tout en conservant une complexité de l'unité de traitement raisonnable. La complexité de l'unité de traitement dépend du nombre de registres nécessaire à stocker les données dont les lectures ont été anticipées. Il faut garantir les contraintes temporelles (augmenter le nombre de registres) et limiter la complexité de l'architecture de l'unité de traitement (réduire le nombre de registres).

La gestion des files *L_S* et *AE_M* est une adaptation de la gestion par kanban. L'originalité de l'approche est la gestion de l'orthogonalité entre les lectures en mémoire (parallélisme d'accès) et leurs consommation par les opérations de l'unité de traitement (parallélisme de calcul). L'introduction des files *AL_M* et *AL_S* permet de répondre à ce problème lors de l'ordonnancement.

I. GESTION DE L'ORDONNANCEMENT

A. Gestion des lectures

La méthode de gestion employée pour gérer les accès en lecture par anticipation va permettre de limiter la complexité. L'ordre des accès à la mémoire va être défini statiquement. Les données à transférer vont être classées statiquement à partir des dates ASAP et ALAP du *SFG*. Ce tri réalisé statiquement permet de définir l'ordre dans lequel les données vont être lues. Ceci permet de réduire la complexité de l'ordonnancement puisque le choix des transferts de données ne sont plus réalisées dynamiquement. Seules la gestion des variables disponibles dans l'unité de traitement devra être gérer dynamiquement. Les différentes files et listes utilisées pour gérer l'anticipation des lectures vont être détaillées ci dessous.

1) Les files AL_M.

Les données sont distribuées en mémoires et pour chaque mémoire allouée, nous définissons une file contenant toutes les données qui doivent y être lues. Dans le cas des lectures, nous utilisons les dates *ALAP* des nœuds de données qui doivent être lues pour déterminer leurs positions dans la file. Pour déterminer les dates *ALAP* des nœuds de données accédant à la mémoire, il faut connaître la date *ALAP* de tous ses nœuds successeurs (Figure 2).

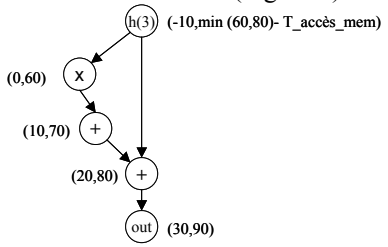

Figure 2 : Détermination des dates ALAP des nœuds accédés en lecture

La date *ALAP* du nœud de données est le minimum des dates *ALAP* de ses successeurs moins le temps d'accès à la mémoire. Lors de l'ordonnancement, à chaque cycle d'accès à la mémoire, un nombre de données dépendant de la mémoire sélectionnée (simple ou multiports …), sont défilées et considérées disponibles pour l'ordonnancement des opérations de l'unité de traitement.

En classant les données suivant l'ordre croissant de leur date *ALAP*, nous garantissons la cohérence entre les transferts de données et l'ordonnancement des opérations.

2) Les files AL_S

Les données distribuées en mémoire ne sont pas forcément les opérandes d'une seule opération. Nous définissons des files contenant les opérandes placés en mémoire pour chaque type d'opération *f* sélectionnée. Les données seront là aussi classées suivant une date *ALAP* (Figure 3) qui diffère en fonction des opérations.

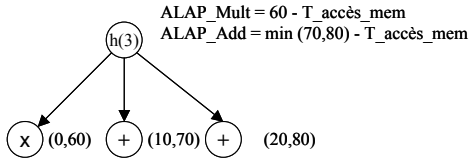

Figure 3 : Détermination des dates *ALAP* de lecture en fonction des opérations

Nous fixons des dates *ALAP* pour chaque type d'opération utilisée dans le graphe. Ce classement permet de définir une priorité d'accès aux données suivant le type d'opération à réaliser.

3) Les listes L_S

Ces listes sont gérées dynamiquement au cours de l'ordonnancement ; elles vont permettre de gérer le flux de données entre l'unité de mémorisation et l'unité de traitement (Figure 4).

Les données lues en mémoire à chaque cycle d'accès sont rangées dans les listes *L_S* en fonction du type d'opération dont elles sont les opérandes. Elles sont classées dans ces listes en fonction des dates *ALAP* déterminées pour chaque type de fonction.

Si les données placées dans une liste *L_S* ne sont pas consommées par le traitement, elles restent dans la liste ; une fois consommées par le traitement elles sont retirées de la liste.

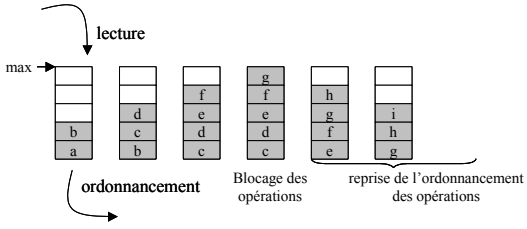

Figure 4 : Gestion des listes L_S

Le flux entre l'unité de mémorisation et l'unité de traitement est géré par la taille de la liste. Les lectures

en mémoire sont gelées lorsqu'une liste *L_S* est pleine. Dans ce cas il faut ordonnancer les opérations dont les opérandes sont contenus dans la liste pour pouvoir relancer le processus de lecture de données en mémoire. Ceci revient à dimensionner le nombre de kanban qui peuvent être utilisés pour chaque type d'opération. Nous pouvons déterminer les bornes maximales des listes *L_S* en fonction du nombre d'opérandes qui sont potentiellement accessibles en parallèle lors de l'ordonnancement.

B. Gestion du traitement

Pour qu'une opération soit exécutable, il faut qu'elle respecte les deux critères suivants. Toutes les opérations dont elle dépend doivent être exécutées (critère de précédence), c'est à dire que le temps de fin de l'ensemble de ses prédécesseurs doit être inférieur au temps courant. De plus les opérandes doivent être disponibles (critère de disponibilité), c'est à dire que l'ensemble des données nécessaires à l'opération est présent soit dans la liste *L_S*, soit en interne dans l'unité de traitement. Toute opération ne respectant pas ces deux critères ne peut être ordonnancée.

Sélection et allocation;
Initialisation de l'ordonnancement;
    Ordonnancement ASAP-ALAP
    Création des séquences d'accès en lecture par mémoire
Ordonnancement
Tant qu'il reste des opérations à exécuter *faire*
    Création éventuelle d'opérateur pour traiter le parallélisme dur;
    Chercher à assigner toutes les opérations exécutables
    Progression du temps;
    Rechercher les opérations en cours d'exécution terminant ou relachant leurs opérateurs;
    Libérer les opérateurs aux opérations se terminant;
    Gérer les séquences d'accès en écriture par mémoire
    Déterminer les nouvelles opérations exécutables;
Fin tant que;

Figure 5 : ordonnancement des caculs

La mise en œuvre ne change pas fondamentalement la gestion de l'ordonnancement des opérations de l'unité de traitement. Le critère de disponibilité (présence dans la liste L_S) des données a changé par rapport à un ordonnancement sans anticipation d'accès aux données [3]. La seconde modification vis à vis de l'ordonnancement est qu'il peut être interrompu si le nombre d'écritures en mémoire atteint un seuil critique. Ce point sera détaillé dans le paragraphe suivant.

C. Gestions des écritures

En ce qui concerne les écritures en mémoire le poste amont est l'unité de traitement et le poste aval, l'unité de mémorisation. Dans ce cas, nous ne sommes pas obligés de tenir compte du type d'opération qui produit une donnée pour gérer les écritures en mémoire. Nous définissons une file contenant les données produites par tout type d'opération de l'unité de traitement pour chaque mémoire. Ce sont les files de données à écrire dans une mémoire donnée *(liste AE_M)*. Les *files E_M* contiennent les données qui ont été écrites en mémoire.

1) Les listes AE_M

Ces files sont gérées dynamiquement ; elles vont permettre de gérer le flux de données entre l'unité de traitement et l'unité de mémorisation.

Lorsqu'une opération se termine (les opérations sont défilées des ensembles *EnsC* pour les opérations réalisées sur un opérateur non-pipeline et de *EnsCC* pour les opérations réalisées sur un opérateur pipeline), les variables produites peuvent être mises en mémoire. Les données qui doivent être écrites en mémoire sont placées dans les listes *AE_M* en fonction de leur mémoire destination. Elles sont classées dans ces files en fonction des dates *ASAP* des opérations qui les ont produites et des temps de traversée des opérations. Le graphe flot de signaux partiel de la Figure 6 illustre le calcul des dates *ASAP* des nœuds de données devant être écrites en mémoire.

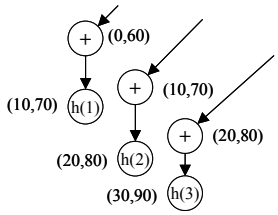

Figure 6 : détermination des dates *ASAP* des nœuds de données à écrire en mémoire

Il faut vérifier que les mémoires associées aux files sont accessibles. Si les mémoires sont accessibles, les variables sont défilées des listes *AE_M* et enfilées dans les files *E_M*. Dans le cas contraire, elles restent dans les listes *AE_M*. Pour gérer le flux de données entre unité de traitement et unité de mémorisation, les listes *AE_M* sont bornées. Ceci revient à dimensionner le kanban pour les écritures vers les différentes mémoires. Nous pouvons déterminer les bornes maximales des listes *AE_M* en fonction du nombre d'opérations fournissant une donnée à placer en mémoire qui sont potentiellement réalisables en parallèle lors de l'ordonnancement. Les bornes minimales sont égales à 1.

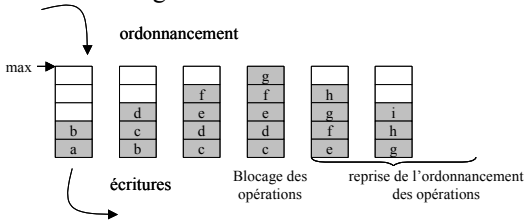

Figure 7 : Gestion des écritures

Pour un nombre de kanban donnée, lorsque la liste *AE_M* est remplie, il faut impérativement écrire en mémoire, les opérations de l'unité de traitement qui

produisent des données à écrire en mémoire sont bloquées tant que la liste *AE_M* est pleine. Ce mécanisme permet de réguler les accès mémoire en écriture.

*2) Les files E_*M

Les files *E_M* servent à conserver toutes les données qui ont été écrites en mémoire. Certaines des données placées dans ces files vont être recopiées dans les files *AL_M* et *AL_S* car elles sont à nouveaux lues.

Pour valider le nouvel ordonnancement et définir son influence sur l'architecture de l'unité de traitement, nous avons effectué plusieurs synthèses avec GAUT pour une application de FFT sur 32 points

II. EXPERIENCES

Nous comparons les résultats de synthèse en fonction de différents mapping mémoire. Les synthèse sont réalisées sous contraintes de mémoire simples puis en utilisant la gestion par kanban (Figure 8). Les échantillons sont placés dans deux bancs. Les 16 premiers dans le premier banc, les 16 derniers dans le banc2 pour map1, puis 8 par 8 pour map2, 4 par 4 pour map3, 2 par 2 pour map4 et enfin les échantillons pairs dans un banc, les échantillons impairs dans le second pour map5. Le nombre de ressources, la surface et le temps sont définis par l'outil GAUT. La consommation en puissance est déterminée par l'outil Xpower après synthèse sur un FPGA virtexE xcv400e_8bg432.

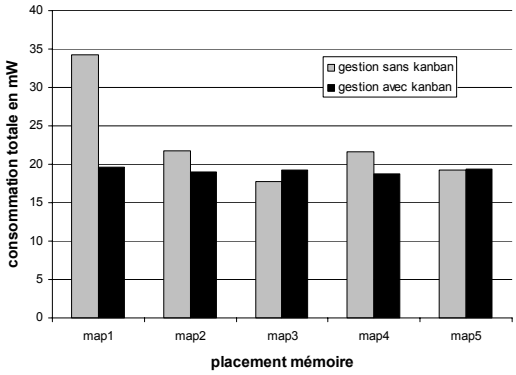

Figure 8 : comparaison de la consommation totale

Seule la puissance dynamique est considérée car la puissance statique dépend uniquement du composant Virtex utilisé. Le nombre de ressources arithmétiques est le même pour toutes les solutions, à savoir un multiplieur, un additionneur et un soustracteur. Si une gestion classique des accès permet d'obtenir une consommation plus faible pour certains placements (map 3), la gestion par kanban permet réguler la consommation quel que soit le placement (variation de 4% contre 50%) tout en garantissant une consommation relativement faible (+5% par rapport à la meilleure solution sans kanban).

III. CONCLUSIONS

Dans cet article, nous montrons l'efficacité d'une gestion des accès mémoire par kanban lors de la phase d'ordonnancement de la synthèse de haut niveau. Cette approche permet de gérer de manière simple efficace l'anticipation des transferts de données entre mémoire et unité de traitement. Les effets du placement mémoire sur la consommation en puissance sont lissés par rapport à un ordonnancement sous contrainte mémoire classique.

BIBLIOGRAPHIE

[1] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, A. Vandecappelle, "*Custom Memory Management Methodology : Exploration of Memory Organisation for Embedded Multimedia System Design*," ISBN 0-7923-8288-9, Kluwer Acad. Publ., Boston,1998.

[2] P. R. Panda, N. D. Dutt, and A. Nicolau, "*Memory Issues in Embedded Systems-On-Chip: Optimizations and Exploration*," Kluwer Academic Publishers, Norwell, MA, 1999

[3] Gwenolé Corre, Eric Senn, Nathalie Julien, Eric Martin ," A Memory Aware behavioral Synthesis Tool for Real-Time VLSI Circuits,"in *proc of GLSVLSI*, pp 82-85, Apr 2004.

[4] Gwenolé Corre, Eric Senn, Nathalie Julien, Eric Martin ," Memory Accesses management during High Level Synthesis," *in proc of CODES+ISSS 04*, sep 2004.

[5] J.Souty, *"L'ingénierie de production 50 fiches pour mieux produire,"* ISBN : 2225823057, Edition MASSON, Collection Organisation Industrielle, 1991

[6] G. Javel, *"Pratique de la gestion industrielle : Organisation, méthodes et outils,"* ISBN 2100053868, Edition Dunod, Collection Technique et ingénierie, 2003.