PPart : Manuel d'utilisation

L. Lemarchand

January 25, 1999

1 Introduction

Ce manuel d'utilisation fournit les informations nécessaires pour la synthèse en parallèle de circuits FPGA à l'aide du logiciel PPart. La version 1.1 de PPart est disponible en `http://ubolib.univ-brest.fr/~lemarch/PPart`.

Les circuits, combinatoires et séquentiels sont optimisés en utilisant SIS [5], développé à UCB. La parallélisation s'appuie sur un partitionnement des données (les équations logiques du circuit) à l'aide de HMetis [2].

PPart est plus particulièrement dédié à la synthèse logique de circuits pour les FPGA Xilinx des séries xc4000 et xc6200. PPart comporte plusieurs composantes distinctes :

- Un outil de synthèse de circuits en parallèle par partitionnement.
- Un outil de synthèse de circuits hiérarchiques.
- Un convertisseur de liste technologique vers le format XNF des séries xc3000 et xc4000.
- Un synthétiseur-placeur-routeur pour la série xc6200.
- Un convertisseur de circuit non-placés vers le format Edif 2.0.0 pour la série xc6200.

La synthèse de circuits de grande taille nécessite d'importantes ressources CPU et requiert beaucoup de mémoire. La solution proposée dans PPart pour raccourcir les temps de synthèse consiste à décomposer le réseau booléen représentant le circuit à optimiser en sous-réseaux. Chaque sous-réseau contient une partie des équations logiques décrivant le circuit. Les sous-réseaux sont optimisés de façon indépendante à l'aide du logiciel de synthèse logique SIS, suivant les critères d'optimisation choisis par l'utilisateur, puis réassemblés en un circuit final optimisé. Toute latitude est laissée au concepteur pour définir les caractéristiques du partitionnement en fonction de la composition initiale du circuit (nombre d'équations, complexité des équations, nombre d'E/S ...). Le concepteur définit également les scripts SIS à utiliser suivant l'architecture visée (des scripts sont proposés par défaut pour l'optimisation de la surface et la vitesse du circuit). Il peut également réaliser une pré et une post-optimisation globale du circuit afin d'améliorer la qualité du partitionnement et de la solution finale.

La seconde manière de réaliser la synthèse en parallèle d'un circuit consiste à synthétiser, si le circuit est défini hiérarchiquement, les différents blocs le composant de manière indépendante. La synthèse est réalisée en respectant les dépendances fonctionnelles entre blocs, un bloc étant optimisé avant ceux l'utilisant. De la même manière que pour la synthèse par partitionnement, le concepteur choisit les critères d'optimisation et la technologie cible.

Du point de vue pratique, la synthèse est réalisée sur un réseau de stations possèdant un système de partage de fichiers et connectés via un protocole IP. Les données à synthétiser doivent être visibles de toutes les stations utilisées lors de la synthèse. L'utilisateur définit les machines, et le répertoire de travail utilisés, et fournit le nom du fichier descriptif de son circuit. Il définit ensuite la manière de partitionner son circuit (nombre de sous-circuits souhaité) et les scripts d'optimisation SIS à utiliser pour la synthèse des sous-réseaux. Toutes ces informations de synthèse sont regroupées dans un *pscript*. Un exemple simple :

```
# ceci est un commentaire
# fichier pscript
# commande 'pvm'
#  liste des machines a utiliser, avec
#  leur vitesse relative (defaut: 100)
pvm mach1 mach2 120 mach3 mach4 130 mach5 110

# commande 'part'
#  partitionnement en 8 du circuit toto au format BLIF
#  le resultat est dans titi
part 8 toto titi

# commande 'source'
# optimisation en parallele de titi a l'aide du script SIS vite
# le resultat est dans tutu (les fichiers contiennent des descriptions
# au format BLIF)
8 source vite titi tutu

# commande 'merge'
# reassemblage dans final
merge 8 tutu final
```

Ce fichier pscript sera exploité, en lançant dans le répertoire contenant le fichier toto la commande `ppart -f pscript`, qui réalisera la synthèse parallèle de `toto` sur les machines `mach1`, `mach2`, `mach3`, `mach4` et `mach5`. Le résultat de la synthèse sera contenu dans le fichier `final`, au format BLIF.

2 Outil de partitionnement

L'outil de partitionnement de circuits est basé sur un algorithme de partitionnement de graphes Metis [3] ou d'hypergraphes, HMetis [2].

Il a été implanté dans SIS par la commande `part`. La commande s'applique sur le circuit courant (obtenu par exemple par la commande `read_blif`). La syntaxe est la suivante :
`sis> part nb out`
où `nb` est le nombre de parties désirées et `out` est le nom de sortie des fichiers résultants.

La commande partitionne le graphe représentant le circuit *sans toucher à la logique interne* des sommets. Chaque sommet interne (autre que les entrées/sorties) est affecté à l'une des parties `out.part-0`, ... `out.part-<nb-1>`. Des entrées/sorties sont générées automatiquement pour les E/S initiales du circuit et les signaux internes utilisés par des sommets se situant dans des parties distinctes. Les parties sont écrites au format BLIF dans les fichiers résultants. Ceux-ci sont situés dans le répertoire courant.

Le partitionnement est réalisé en essayant d'équilibrer le nombre de sommets (internes) par partie. Par défaut, chaque sommet a le même poids, mais il est possible d'affecter un poids différent aux différent sommets pour refléter la complexité de leur logique interne. Cela permet d'équilibrer la taille des parties non plus en nombre de sommets, mais avec une métrique plus fine reflétant la complexité des parties. La complexité d'une partie est alors mesurée par la somme des complexités des sommets qui y sont inclus. Avoir une mesure, même approximative de la complexité d'une partie, permet d'estimer le temps de synthèse qui sera nécessaire à l'optimisation par un algorithme de synthèse logique de cette partie. Equilibrer la complexité des parties entraîne un équilibrage de la charge de calcul lors de la synthèse simultanée par des processeurs distincts de l'ensemble des parties.

La notion de complexité est intéressante pour évaluer le temps de synthèse correspondant à l'optimisation logique d'un sommet. Or l'optimisation dépend du type d'algorithme employé : il n'est par exemple pas possible d'évaluer *a priori* le nombre de noyaux qui seront extrait d'une équation logique pour sa factorisation.

Une mesure générale de complexité peut être le nombre le litéraux de la forme SOP d'une équation. Certains algorithmes d'optimisation se basent sur le voisinage d'un sommet pour l'optimiser (méthodes de réinjection). Une autre métrique consiste à calculer le nombre d'entrées que comporte une équation.

Lorsque l'on partitionne les sommets dans différentes parties, des informations de dépendances entre sommets sont perdues. Comme il a été dit plus haut, des signaux d'E/S sont générés automatiquement pour des sommets partageant le même signal, mais situés dans des parties différentes. Supposons que un sommet interne A produise en sortie le signal a. Si a est utilisé en entrée par le sommet B, et que A et B sont placés dans des parties P_A et P_B respectivement lors du partitionnement, une sortie primaire a sera générée pour le circuit P_A et une entrée primaire a sera générée pour le circuit P_B. Lors de l'optimisation du circuit P_B, le signal a sera considéré comme un signal d'entrée/sortie, et non pas comme le résultat de la fonction logique A. Aucune optimisation basée sur les propriétés de A ne sera donc possible. Par exemple, si l'équation du sommet A est $a = 1$ (fonction constante) et que celle de B est $b = a + ce$, B peut se simplifier en $b = ce$ en considérant le réseau globalement. Si A et B sont dans des parties séparées, la constante a ne peut être propagée pour simplifier B.

Un réseau optimisé globalement permet d'obtenir de meilleurs résultats en terme de qualité, en exploitant les dépendances entre sommets, comme montré dans l'exemple de réinjection ci-dessus. De nombreux algorithmes exploitent ces informations de dépendance. Le problème réside dans le fait que les algorithmes de synthèse exigent des ressources de calcul et des temps d'exécution exponentiels par rapport au nombre d'équations que comporte le circuit (algorithmes NP-complets).

Une première solution consiste à employer des heuristiques. Celles-ci fournissent des solutions approchées, parfois de bonne qualité. Mais même si ces heuristiques sont moins coûteuses en ressources que des algorithmes exacts, elles atteignent elles-aussi des temps d'exécution prohibitifs

pour des circuits de grande taille.
L'avènement de circuits intégrés ASIC ou FPGA de taille de plus en plus importantes autorise aujourd'hui la réalisation dans le matériel de circuits de taille très importante. Ces progrès technologiques s'accompagnent de l'apparition de logiciels et de langages de spécification de circuits de haut niveau pour maitriser la complexité des circuits réalisables dans ces nouvelles technologies.

Tant l'évolution de la capacité des circuits que celle des possibilités de spécification de fonctionnalités complexes induisent des besoins très importants en capacité de traitement lors de l'étape intermédiaire entre la spécification et la réalisation physique que constitue la synthèse logique. La méthode de synthèse par partitionnement offre l'avantage de réduire la taille des circuits traités. Par contre, elle induit une perte au niveau de la qualité finale du circuit, reconstitué, après optimisations de ses différentes parties.

Pour minimiser cette perte, il faut minimiser le nombre de signaux partagés par différentes parties, tout en préservant l'équilibre en terme de complexité de celles-ci. Ce problème revient à réaliser un partitonnement multi-partie de graphe (*k-way partitioning problem*). Le graphe traité est celui associé au réseau booléen définissant le circuit, en affectant aux différents sommets un poids reflétant leur complexité. Le but de l'algorithme est de partitionner le graphe en équilibrant les poids entre parties et en minimisant la coupe. La coupe est définie comme le nombre d'arêtes du graphe connectant des sommets situés dans des parties distinctes. C'est ce que réalise les algorithmes Metis et HMetis interfacés par la commande `part` dans SIS. Sa syntaxe complète est la suivante :

```
part [options] [-f out]
```

les options sont les suivantes :

- `-p np` : `np` est le nombre de blocs désirés (défaut 4).
- `-f out` : `out` est le nom des fichiers de sortie pour l'écriture des parties au format BLIF (défaut `partition`).
- `-u` : donner un poids unitaire à chaque sommet pour le calcul du poids de chaque partie (défaut).
- `-l` : le poids de chaque sommet correspond au nombre de litéraux dans la forme SOP de l'équation associée au sommet.
- `-h` : utiliser HMetis pour le partitionnement (défaut : utiliser Metis).
- `-b` : utiliser un algorithme de *Bin-Packing* à la place des algorithmes de partitionnement Metis et HMetis. Plus rapide, mais ne minimise pas les interconnexions coupées par le partitionnement.
- `-n` : modifie pour l'option `-h` la manière dont les déconnexions sont comptabilisées.

Le réassemblage, après optimisation par un logiciel de synthèse des différentes parties est réalisé par la commande `merge`, elle aussi intégrée dans SIS. Sa syntaxe est la suivante :

```
merge [-f in] [-p nb]
```

`in` est le nom des `nb` fichiers à réassembler, décrivant les sous-circuits au format BLIF (défauts : `partition` et 4).

3 Synthèse parallèle par partitionnement

Comme vu précédemment, la synthèse parallèle est réalisée en partitionnant un circuit à l'aide de la commande `part` puis en appliquant des scripts SIS sur chacune des parties avant de réassembler le circuit. Comme chaque partie constitue un circuit indépendant (si l'on fait abstraction des dépendances entre sous-circuits modélisées par l'ajout d'E/S primaires fictives), chaque circuit peut être optimisé en parallèle sur un réseau de processeurs. Cela permet des gains en temps pour la synthèse très appréciables. De plus, n'importe quel algorithme de synthèse peut être utilisé, et les améliorations apportées à un algorithme peuvent être directement exploitées. Enfin, le mécanisme de distribution des tâches de synthèse est complètement paramétrable par l'utilisateur, qui peut, en ajoutant des phases de pré- ou post-optimisation globales, garantir la qualité de son circuit final (même si cela peut parfois être fait au détriment du temps d'exécution de la synthèse).

Le logiciel en charge de l'exécution des tâches de synthèse est PPart. Il regroupe :

- Une interface vers SIS pour l'exécution de tâches de synthèse sur des circuits entiers (un seul processeur) ou des circuits partitionnés (plusieurs processus SIS exécutés sur des processeurs différents)
- Un analyseur de commandes, permettant de spécifier les processeurs à utiliser pour la synthèse parallèle, les fichiers d'entrée et de résultat pour les circuits à synthétiser, et les scripts à utiliser lors de la synthèse. L'ensemble des commandes permettant la synthèse d'un circuit donné sont regroupées dans un *pscript*. L'analyseur de PPart lit et exécute les commandes séquentiellement, lançant au fur et à mesure les tâches de synthèse, d'affichage de statistiques ou de conversion de fichiers spécifiées par l'utilisateur.

3.1 L'algorithme parallèle

Le processus PPart fonctionne en mode maître/esclave. L'analyseur constitue le maître, et les tâches de synthèse sont réalisées par des processus esclaves exécutant SIS. La syntaxe de PPart est la suivante :

```
ppart -f pscript
```

`pscript` contient la liste de commandes à exécuter. La distribution des tâches est basée sur PVM [1], un ensemble de routines permettant l'exécution sur réseau hétérogène de stations de travail de processus distribués. L'utilisateur spécifie au départ le nombre et le nom des processeurs disponibles pour réaliser la synthèse des circuits, en y incluant la machine exécutant le maître PPart. Chacune des machines exécutera un processus esclave SIS. L'algorithme fonctionne de la façon suivante.

En plus du maître, un esclave (démon) SIS est lancé en début de synthèse sur chacune des machines, et attend des commandes de synthèse à exécuter. Le maître PPart exécute un algorithme de répartition de charge tournant sur une liste de tâches à exécuter qu'il stocke. A chaque fois qu'un esclave SIS est libre, il effectue une demande de travail au maître, qui lui envoie une tâche constituée d'une commande SIS à réaliser. L'algorithme de distribution termine lorsque tous les travaux à exécuter en parallèle ont été réalisés. Le fait de lancer un démon SIS par machine permet d'éviter de relancer SIS sur chaque machine à chaque fois que un travail doit être réalisé par le processeur associé. L'algorithme tournant, associé à une mesure de la rapidité des processeurs (soit définie de manière externe, soit mesurée par un test, *cf.* commande `speed`) permet d'équilibrer la charge de manière raisonnable, si les tâches de synthèse à réaliser sont assez fines, ou si le nombre de tâches est élevé par rapport au nombre de processeurs, les plus rapides d'entre eux exécutant le plus de tâches. Par contre si la puissance des machines est très hétérogène, et que le nombre de tâches à réaliser avoisinne celui des processeurs, les machines les plus rapides resteront inoccupées le temps que les machines les plus lentes aient terminé leur exécution. Il n'existe pas de mécanisme de migration de tâches pour transférer une tâche d'un processeur peu performant à une autre machine plus efficace.

3.2 Les commandes pscript

Les commandes sont réparties en 3 classes :

- Partitionnement et fusion de circuit (`part` & `merge`, voir ci-dessus).
- L'exécution de scripts SIS en parallèle par les esclaves sur des sous-circuits.
- Des commandes diverses pour l'initialisation (définition des machines utilisées), l'affichage de statistiques, la conversion de fichiers dans des formats divers ...

Le lancement de la machine virtuelle Les machines physiques utilisées pour réaliser la synthèse d'un circuit forment, dans la terminologie PVM, une machine virtuelle [1]. Avant de réaliser une tâche de synthèse, un pscript doit contenir l'ordre de lancement de la machine virtuelle correspondant à l'exécution d'une commande `pvm` de la forme :

```
pvm machine [speed] [machine [speed]]
```

La vitesse `speed` associée à chaque machine permet de classer celles-ci par ordre de performances. La vitesse par défaut est de 100. Toutes les machines citées dans la liste doivent avoir accès au répertoire courant (d'où est lancé PPart) si des commandes avec entrées/sorties de fichiers sont utilisées dans la suite du pscript. En particulier, la machine définie comme étant la plus puissante sera utilisée pour l'exécution de toutes les commandes SIS séquentielles (entre autres le partitionnement du circuit). Si aucune vitesse n'est spécifiée, c'est la dernière machine de la liste qui sera utilisée.

Les machines, et l'environnement de l'utilisateur sur chacune d'entre-elles, doivent permettre l'exécution de PVM, et du logiciel SIS. Notamment dans le cas de réseau hétérogènes en matière de systèmes d'exploitation, il faut vérifier que SIS est bien exécutable sur chacune des architectures, et que les librairies utilisées (script par défaut par exemple pour l'optimisation d'un circuit) sont bien cohérentes entre les différentes installations.

Dans un environnement d'utilisation partagé des ressources (réseau de stations non utilisées exclusivement pour la synthèse par exemple), la vitesse des machines peut être déterminée à un instant donné en exécutant la commande `speed`, afin de réaliser un test en grandeur réelle de la vitesse d'exécution de SIS sur les diverses machines utilisées. La syntaxe de la commande est la suivante :

```
speed file script
```

La commande exécute le script SIS `script` sur le fichier `file` au format de description BLIF. Le temps d'exécution sur les différents processeurs tient lieu d'évaluation pour la vitesse de la machine associée. Le script utilisé et le fichier de test ne doivent pas être trop complexes afin de ne pas pénaliser de manière trop importante les temps d'exécution utiles lors de la synthèse. Cependant, si par exemple, les effets de swap (utilisation de la mémoire virtuelle sur le disque dur) veulent être pris en compte, la tâche à exécuter doit être assez importante. Si la tâche ne nécessite que très peu de ressources de calcul, le temps mesuré correspondra plus aux effets de distribution des données (lancement de SIS ou envoi de messages d'exécution suivant l'algorithme de répartie, partage NFS, entrées/sorties dans SIS) que aux temps CPU d'exécution de la synthèse elle-même. Or, dans les applications réelles du logiciel de synthèse sur des circuits de grande taille, le temps de synthèse est très nettement prépondérant par rapport au coût de la parallélisation. Le test effectué doit donc prendre en compte ces aspects contradictoires : ne pas être lui-même trop coûteux en temps de calcul, mais rester significatif par rapport à la mesure à effectuer.

Partitionnement et fusion Les commandes `part` et `merge` sont utilisées pour partitionner et fusionner un circuit. L'idée est que, une fois le circuit partitionné, chaque sous-circuit, stocké dans le répertoire courant (répertoire d'exécution de PPart), sera ensuite optimisé par un processus SIS distinct en parallèle. Les commandes d'optimisation répartissent les tâches d'exécution en fonction de la vitesse des machines en considérant que chaque tâche a une complexité d'exécution équivalente, et en les considérant d'après le nom de fichier d'entrée dans l'ordre des noms, les données des fichiers d'entrées sont classées par ordre de taille décroissante. Cela favorise l'affectation des tâches de synthèse ayant les données d'entrées les plus volumineuses aux processeurs les plus rapides.

Le nombre de parties est laissé à l'appréciation de l'utilisateur. C'est à lui de définir une taille de partition assez élevée pour générer des circuits de taille assez restreinte pour être synthétisés en des temps raisonnables. La notion de *temps raisonnables* dépend du concepteur et de la qualité d'optimisation qu'il attend. *a priori* plus le nombre de parties est élevé, plus la synthèse sera rapide en terme de temps CPU utilisé, mais plus la qualité du circuit final risque de se dégrader (par rapport au même type d'optimisation effectué sans partitionner le circuit). Par contre, si le nombre de processeurs disponibles est élevé, l'utilisation du potentiel de chacun d'eux nécessite un partitionnement en au moins un nombre égal au nombre de processeurs que de parties. Enfin, l'outil de partitionnement ne touchant pas à la description fonctionnelle du circuit, celle-ci doit être assez éclatée pour permettre un partitionnement efficace : par exemple, si un circuit est décrit en logique 2-niveaux (format PLA), le nombre d'équations logiques sera égal au nombre de sorties du circuit. Cela signifie que le partitionnement est alors limité par ce nombre de sorties du circuit. Il est dans ce cas possible de réaliser une transformation globale (*i.e.,* sur un seul processeur) du circuit avant de réaliser son partitionnement. Cette transformation doit avoir pour but de décomposer les sommets (équations) complexes en sommets moins denses, ce qui augmente le nombre de sommets total décrivant le circuit et facilite son partitionnement.

Les options du partitionnement par défaut peuvent être changées en utilisant la commande

```
partopt options
```

Les options correspondent à celles de la commande `part` incluse dans SIS et décrites ci-dessus : `bmluh`.

synthèse simultanée de sous-circuits Le principe de spécification de commandes de synthèse à exécuter en parallèle est le suivant : si la commande est précédée d'un nombre `nb`, elle donnera lieu à la génération de `nb` tâches distribuées aux processeurs disponibles. Les noms des fichiers donnés comme arguments d'entrée et/ou de sortie des commandes pscript seront post-fixées par `.part-`i, où i est un nombre compris entre 0 et $nb - 1$. Par exemple, la commande pscript `xnf`, qui génère à partir d'un fichier d'entrée représentant une liste technologique une description de circuit au format XNF 5.2 compatible avec les outils Xilinx pour les FPGA de la série 4000 pourra être utilisée pour convertir plusieurs fichiers simultanément par la commande pscript :

```
3 xnf toto titi
```

Les fichiers `toto.part-0`, `toto.part-1` et `toto.part-2` (décrivant des circuits au format BLIF) seront respectivement convertis en `titi.part-0`, `titi.part-1` et `titi.part-2` au format XNF 5.2 .

Les différentes tâches de conversion seront réalisées en parallèle sur les machines disponibles. Si aucun nombre n'est spécifié avant la commande pscript, une seule tâche est générée et exécutée, sans post-fixage des arguments d'entrée-sortie.

Les fichiers fournis en argument (fichiers d'entrée/sortie, fichiers script ...) peuvent être spécifiés à partir du répertoire de travail (répertoire de lancement de PPart), en utilisant la notation ~ ou en donnant le chemin absolu du fichier. Tous les fichiers utilisés (à part ceux de la librairie SIS) doivent être accessibles via NFS sur toutes les machines utilisées pour la synthèse.

Les commandes pouvant donner lieu à exécution simultanée de plusieurs tâches sont les suivantes :

- `[np] xnf in out` (voir ci-dessus). Les circuits décrits dans le(s) fichier(s) `in` doivent correspondre à des listes technologiques xc4000, *i.e.,* ne pas contenir de sommets ayant plus de 4 entrées distinctes. Les bascules des circuits séquentiels sont converties en flip-flops asservies à une horloge globale commune (circuits considérés comme synchrones). L'algorithme de conversion définit des cellules HMAP et FMAP, mais sans forcer leur placement dans des CLB communs. Pour cette commande et les suivantes, le format des fichiers est toujours BLIF. Des conversions vers ce format sont possibles à l'aide de SIS, à partir des formats : blif, kiss, slif, pla, eqn [1].

- `[np] ps in` exécute la commande SIS `print_stats` qui affiche les caractéristiques d'un circuit. Le mode `verbose` (voir plus bas) doit être activé pour que l'affichage des informations soit effectif. Par exemple, la commande `ps essai1.blif` donnera :

```
[t40003] BEGIN
[t40002] essai1.blif    pi= 8   po= 2   nodes=  7       latches= 2
[t40002] lits(sop)=  187
[t40002] EOF
```

[1]edif est aussi accepté dans notre distribution. L'outil de conversion de BLIF vers EDIF 2.0.0 fourni par le site ISCAS'93 a été interfacé dans SIS par les commandes read_edif et write_edif

Le circuit `essai1.blif`, au format BLIF, contient 8 entrées, 2 sorties, possède 8 sommets internes (*i.e.,* est décrit par 8 équations logiques dans cette description) pour un total de 187 litéraux si les équations sont représentées sous forme normale disjonctive. Deux bascules sont égalements incluses dans le circuit.

- `[np] part parts in out` réalise le partitionnement en `parts` parties du (des) circuits contenus dans les fichiers `in`. Les parties sont stockées au format BLIF dans les fichiers `out.part-`*i*. Les options par défaut de la commande `part` sont utilisées pour le partitionnement du (des) circuit(s).

- `[np] merge parts in out` fusionne les `parts` circuits contenus dans les fichiers `in`. Le circuit résultant est stocké dans le fichier `out` au format BLIF.

- `[np] source sis_com in out` applique la suite de commandes SIS contenue dans le fichier `sis_com` au fichier `in`. Le résultat de la synthèse est stocké dans le fichier `out`. Le fichier de commandes SIS est dans la terminologie de SIS un *script*. La personnalisation des commandes incluses dans le script SIS permet de cibler une technologie de réalisation du circuit, et de choisir des critères d'optimisation spécifiques. Les entrées/sorties de fichiers sont à utiliser avec précaution dans ces scripts, car le script est appliqué en parallèle aux différents circuits si `np` est supérieur à 1. Si par exemple la commande d'écriture `write_blif result.blif` est utilisée dans le script `sis_com` et que `np` vaut 8, le même fichier `result.blif` sera ouvert et écrit simultanément par les 8 processus SIS, ce que engendrera bien sûr un résultat invalide. L'application de la commande `source` prend en compte directement les entrées/sorties correspondant aux argements `in` et `out`, qu'il n'est pas nécessaire d'inclure dans le script. De plus, les appels à librairies correspondent à ceux réalisables en lançant l'exécutable SIS en ligne. Si l'on désire par exemple appliquer le script SIS `script` de la librairie SIS (qui est le script standard d'optimisation de MisII), il suffit d'appeler dans le pscript :

```
8 source script initial result
```

De plus, les fichiers script peuvent, comme les fichiers d'entrés/sorties être spécifiés à partir du répertoire courant ou en absolu.

Une seconde façon de spécifier les scripts SIS à utiliser consiste à les définir directement dans le fichier *pscript*, en utilisant le mot-clé `online` à la place du nom de fichier correspondant au script. Les instructions suivantes, jusqu'au mot-clé `end` seront alors interprétées comme des commandes SIS. Cette possibilité est particulièrement utile dans le cas où le script SIS utilisé est court. Exemple :

```
8 source online
decomp -q
sweep; simplify
end initial result
```

est équivalent à `8 source scriptCourt initial result` , avec les commandes SIS `decomp -q; sweep; simplify` dans le fichier `scriptCourt`.

Autres commandes Des commandes de calcul de statistiques, de conversion et d'affichage sont également disponibles :

- `verbose` active le mode explicite d'exécution : toutes les commandes pscript sont affichées au début de leur exécution, et les temps d'exécution pour les différentes machines participant à l'exécution d'une commande sont détaillées. Le mode `verbose` doit être activé avant le lancement de la machine virtuelle si l'on désire des informations détaillées, et notamment dans le cas de l'utilisation de la commande `ps`. Les informations fournies par chacun des processeurs sont alors affichées sur la console de lancement de PPart. Par exemple, le script suivant

```
verbose
pvm mach1 mach2 120
part 4 initial partie
4 source script partie res
merge 4 res final
```

produira en sortie :

```
Machines : (mach2, 120, 0) (mach1, 100, 0)
[t40002] BEGIN
[t80001] BEGIN
Phase 2 : read_blif initialPART; metis 4 partiePART 2 0; pvm_sync

Machines exec time (sec):
  0   mach2 :   0 :  1.4
Phase exec time (sec): 1.7
Phase 3 : read_blif partiePART; source script; write_blif resPART; pvm_sync

Machines exec time (sec):
  0   mach2 :   0   2 :  3.9
  1   mach1 :   1   3 :  4.5
Phase exec time (sec): 4.8
Phase 4 : metis_part 4 resPART; write_blif finalPART; pvm_sync

Machines exec time (sec):
  0   mach2 :   0 :  0.4
Phase exec time (sec): 0.5
Terminated
```

Les phases 2 et 4 sont réalisées par la machine la plus rapide (mach2). Elles correspondent au partitionnement et à la fusion. La phase 1 est celle du lancement de la machine virtuelle. La phase 3 correspond à la synthèse parallèle des 4 sous-circuits sur 2 processeurs. Le processeur 0 (mach2) a par exemple exécuté les tâches 0 et 2 (synthèse des parties 0 et 2), pour un total de temps CPU de 3.9 secondes. La durée totale d'exécution de la synthèse des différentes parties lors de la phase 3 est de 4.8 secondes. Le surcoût par rapport au temps d'exécution du plus lent des processeurs (mach1, 4.5 secs) est dû aux temps de transit des messages de terminaison des différentes tâches. Il est négligeable pour des circuits de grande taille nécessitant d'importants temps de synthèse.

- `silent` déactive le mode `verbose`. C'est le mode par défaut. Aucune information concernant le déroulement de l'algorithme de synthèse parallèle n'est affichée.
- `time` <`counter`> affiche sur la sortie standard de PPart le temps total écoulé depuis le début de l'exécution du programme, le temps passé depuis le dernier appel à `time` et accumule ce temps dans le compteur `counter`. Les compteurs sont numérotés à partir de 0. Le compteur de temps 0 est démarré automatiquement lors du lancement de PPart.

3.3 Exemples

Les fichiers correspondant aux exemples de ce paragraphe se trouvent dans le répertoire `examples` de la distribution PPart.

3.3.1 Exemple 1 : STD

Le circuit synthétisé est `cc.blif` du jeu de tests ISCAS'89 [6].

Le pscript définissant les différentes phases de la synthèse est stocké dans le fichier `8P`. La synthèse va être effectuée sur un ensemble de 4 processeurs, en le découpant en 8 parties. Le but est d'optimiser le circuit puis de réaliser sa conversion technologique vers les FPGA xc4000 de Xilinx. La liste technologique résultante de la synthèse logique sera convertie ensuite au format XNF 5.2 pour être placée et routée sur le composant Xilinx par les outils adéquats (PPR).

```
#########################
# Pscript file STD/8P : #
#########################

# Phase 0 : Demarrage de la machine virtuelle, en assignant des
# vitesses a certains processeurs
pvm mach1 160 mach2 mach3 mach4 300

silent
# Phase 1 : Partitionner cc.blif
part 8 cc.blif cc.blif

# Phase 2 : Optimiser chaque partie en parallele
# 'script' est un script de la librairie SIS recommande
# pour ses bons resultats en optimisation logique
8 source script cc.blif cc1.blif

# Phase 3 : Re-evaluer la vitesse des machines sur site, avant une
# tache de synthese difficile
speed little.blif little_script

# Phase 4 : Conversion technologique des parties avec le script SIS
```

```
# script.xilinx_4k, qui realise le partitionnement des equations
# logiques en tables logiques d'au plus 4 entrees.
8 source script.xilinx_4k cc1.blif cc2.blif

# Phase 5 : Fusion des resultats dans un circuit unique
merge 8 cc2.blif mapped.blif

# Phase 6 : Conversion vers XNF et affichage de statistiques
time 0
verbose
ps mapped.blif
xnf mapped.blif mapped.xnf

# La machine virtuelle est stoppee automatiquement
```

La synthèse est lancée en exécutant la commande

```
% ppart -f 8P
```

3.3.2 Exemple 2: PLA

Le circuit à synthétiser est `num6add.raw.pla`, un additionneur 6 bits (6+6 bits vers 6 bits) avec un codage spécial des opérandes [4]. Le circuit est décrit en logique 2-niveaux au format PLA. Cela correspond à une équation logique unique, très complexe (le fichier fait 82 Ko), par sortie du circuit. Là encore, la cible technologique est la série FPGA xc4000. 8 processeurs sont disponibles pour la synthèse.

L'application de la méthode de synthèse de l'exemple STD ne donnera pas de bons résultats sur ce circuit pour deux raisons :

- Le circuit ne possède que 6 équations, donc il ne pourra pas être partitionné en plus de 6 parties non vides, et seuls 6 processeurs sur 8 pourront être exploités.
- Chaque équation est très complexe, ce qui peut mener à des temps d'exécution de la synthèse prohibitifs, même si les équations sont considérées individuellement.

La représentation du circuit va donc être décomposée (éclatée) avant le partitionnement pour la synthèse en parallèle, afin de permettre la répartition des équations sur les 8 processeurs disponibles. De plus, en employant un nombre de parties important, on va limiter la complexité de chaque partie dans un premier temps. Ensuite, pour obtenir des résultats acceptables en terme de qualité, des parties de taille plus importante seront utilisées dans une deuxième passe d'optimisation.

```
#########################
# Pscript file PLA/8P : #
#########################

# Demarrage de la machine virtuelle
pvm mach1 mach2 mach3 mach4 mach5 machine6 machine7 machine8
```

```
# Repartition des equations dans un sous-circuit chacune, puis
# eclatement de la repr\'esentation des equations en sommets de
# petite taille a l'aide du script SIS 'pla'
part 6 num6add.raw.pla pla
6 source pla num6add.raw.pla num6add.comb
merge 6 num6add.comb num6add.comb.blif

# Le circuit obtenu est de tres grande taille. Partition en 16
# petites parties et synthese
part 16 num6add.comb.blif
16 source 1P num6add.comb.blif num6add.map4.blif
merge 16 num6add.map4.blif num6add.map4.blif

# Le nombre de parties utilise dans la phase precedente etant
# tres important (16), le partage de logique commune n'a pas ete
# realise efficacement. Une deuxieme phase de synthese, sur le
# circuit deja optimise est faite, avec un nombre de parties moins
# important, pour ameliorer la qualite du circuit
part 8 num6add.map4.blif
8 source 1P num6add.map4.blif num6add.map4.blif
merge 8 num6add.map4.blif num6add.map4.blif
```

References

[1] G. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Network Parallel Computing.* MIT Press, 1994.

[2] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning : Application in VLSI domain. In *Proc. ACM/IEEE Design Automation Conf.*, June 1997.

[3] G. Karypis and V. Kumar. Multilevel k-way partitionning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[4] B. Pottier. Resiviting smalltalk-80 blocks. In *IEEE Workshop on* FPGA*s for Custom Computing Machines*, Napa, CA, 1996.

[5] E. Sentovich and al. SIS: a system for sequential circuit synthesis. memorandum UCB/ERL M92/41, University of California at Berkeley, mai 1992.

[6] S. Yang. Logic synthesis and optimization benchmarks user guide. Technical report, Stanford University, 1991.