Évaluation d'un procédé de parallélisation de SystemC

AUDÉOUD Henry-Joseph
Stagiaire à VERIMAG
Étudiant à l'Université Joseph Fourrier, Grenoble

Tuteurs :
Matthieu MOY, Claire MAIZA

Juillet 2011

Abstract

Face au manque de performance des moyens de simulations des systèmes sur puces, Samuel Jones a créé une version parallélisée de SystemC au cours de son master de recherche. Comme tout travail de ce genre, afin de prouver son fonctionnement, il a fallu faire des tests. Mon travail durant ce stage a consisté à effectuer un test démontrant le fonctionnement de sa production.

Table des matières

Chapitre 1

Introduction

SystemC est une librairie écrite en C++, visant à décrire le matériel présent sur une puce électronique. Elle permet de créer des prototypes virtuels, qui aident à la conception de la puce, tant lors de sa création que lors de sa mise au point (correction de bugs, mise au point des logiciels...).
Cette librairie est souvent assimilée à un langage en lui-même, mais ce n'est qu'un regroupement de classes. Elle permet entre autres de créer virtuellement les composants que l'on veut mettre dans la puce.

SystemC n'est prévu pour fonctionner qu'en séquentiel : un seul processeur est capable de travailler sur la simulation. Celle-ci étant souvent très longue, il serait utile de permettre à la simulation d'utiliser les nombreux cœurs dont sont pourvus de plus en plus de machines, ce qui permettrait d'accélérer la simulation, jusqu'à gagner un ordre de grandeur dans le temps de simulation
Il s'agit ici de séparer les différents modules présents dans la puce. En effet, ceux-ci étant relativement indépendants, il n'est pas nécessaire qu'un seul processeur en assume l'entière charge. Toutefois, nous ne nous intéresserons pas au problème de la parallélisation d'un module en lui-même.

Lors de son master de recherche, Samuel Jones [1] a modifié SystemC, afin que celui-ci puisse travailler avec plusieurs cœurs. Durant mon stage, j'ai évalué ce procédé, pour voir les différences qu'il y a, principalement de vitesse de simulation, entre cette version modifiée et la version originale.

Chapitre 2

Principe de parallélisation

2.1 SystemC original, son problème

SystemC, au même titre que Verilog ou VHDL mais avec un niveau d'abstraction plus élevé, permet la description de matériel informatique pour la simulation.
Lors de la création du prototype virtuel, chaque composant est représenté par un objet de la classe *module*. Dans ces modules, des fonctions (appelées *process* sont utilisées afin de faire réagir le composant avec le reste de la puce. Ainsi, les SC_THREAD sont des process qui démarrent automatiquement au début de la simulation, et les SC_METHOD sont des fonctions qui sont réveillées à chaque fois qu'une variable présent dans leur liste de sensibilité est modifiée.
La simulation est gérée grâce à un scheduleur, qui donne la main à chacun des modules selon leurs besoin. Quand un module a la main, il exécute sa tâche, puis rends la main au scheduleur, grâce à la fonction *wait()*.

Toutefois, la simulation de gros systèmes sur puces est très lente, et n'exploite pas toutes les ressources qu'ont maintenant nos ordinateurs, savoir les multiples processeurs présents sur les machines. En effet, le scheduleur de SystemC n'est prévu pour fonctionner qu'en séquentiel, sur un seul processeur. En permettant la parallélisation de SystemC, nous pourrions gagner 1, voir presque 2 puissances de 10 sur les plus grosses machines, ce qui n'est clairement pas négligeable.

2.2 SystemC modifié : son principe

Lors de son master de recherche, Samuel Jones [1] a implémenté une solution pour permettre la parallélisation de SystemC.
Le principe est le suivant :

- On implémente la puce, de la même manière qu'avec la version originale.
- On partitionne virtuellement la puce (grâce à la macro SC_AFFINITY, qui détermine sur quelle partition se trouve chaque module). On peut ainsi regrouper les différents modules en fonction de leur consommation en ressources.
- Lors de la simulation, un processeur est affecté à chaque partition. Chacune d'elle travaille indépendamment des autres partitions, avec leur propre scheduleur et composants, comme s'il s'agissait de plusieurs simulations indépendantes.

Toutefois, le temps simulé doit correspondre entre les différentes partitions. S'il est trop décalé de l'une à l'autre, la simulation sera faussée, à cause d'un décalage temporel. À un intervalle près, tous les temps doivent donc être les mêmes.
Pour cela, deux solutions ont été élaborées :

- *Global control*
  Dans cette approche, on fixe un delta de temps, qui représente l'écart maximal que peuvent avoir les différents temps de simulation. Chaque scheduleur avance ainsi aussi vite qu'il veut, mais ne peut pas distancer de trop les autres scheduleurs.

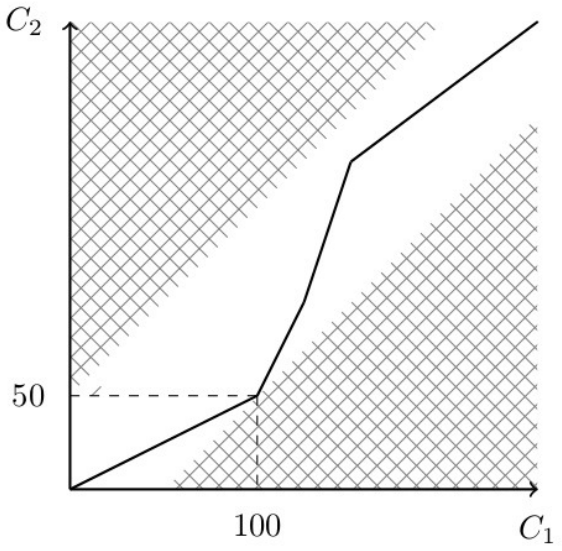

FIGURE 2.1 – Principe de la solution de global control

Par exemple, si l'on a deux scheduleurs. Avec un delta de temps de 50, si l'on met le temps simulé du premier sur l'axe x et celui du deuxième sur l'axe y, le tracé peut avoir cette forme-là (voir figure 2.1). La zone

hachée montre la zone interdite, car les deux scheduleurs aurait plus que 50 de décalage.

– *Local Control*
Ici, chaque scheduleur travaille de manière désynchronisée envers les autres scheduleurs, et ils ne se resynchronisent qu'une fois par intervalle de temps déterminé.

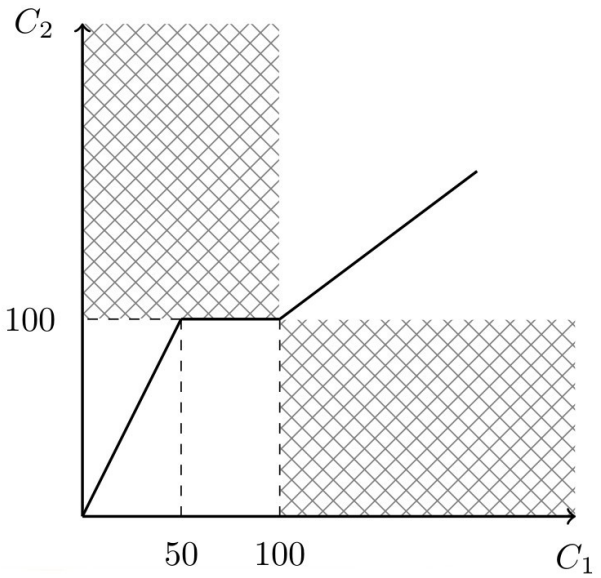

FIGURE 2.2 – Principe de la solution de local control

Si l'on reprends le même exemple, mais avec le local control, on obtiendrait une courbe différente, comme le montre la figure 2.2. Le tracé serait obligé de passer par le point (100,100)

Ainsi, les scheduleurs peuvent travailler en parallèle avec une certaine liberté, mais sans toutefois pouvoir trop se décaler, pour éviter le problème des décalages temporels.

Durant mon stage, j'ai mis en place la solution du global control dans un exemple, afin de pouvoir le comparer le temps pris par la simulation avec la version original ou modifiée de SystemC.

Chapitre 3

Évaluation de ce procédé

3.1 Présentation du test

Pour tester la version parallélisée de SystemC, j'ai utilisé un modèle de système sur puce prévu pour fonctionner avec le SystemC original, et je l'ai adapté à le version prévue pour fonctionner en parallèle. Ainsi, je pouvais comparer exactement la différence de performance entre les deux versions.
Le principe du test est le suivant :

- Un programme écrit en C exécute le jeu de la vie. On a choisi ce programme, car il demande une bonne quantité de calcul, donc donne beaucoup de travail aux processeurs.

- Un système sur puce FPGA Xilinx, où se trouvent principalement un timer, une mémoire RAM, ainsi qu'un processeur de type Microblaze, exécute ce programme. Il gère les calculs, les données, et l'affiche.

- Un ordinateur simule la puce des deux manières présentées plus tôt : une fois avec la version originale de SystemC, une autre fois avec la solution du global quantum de la version modifiée (la solution du local quantum n'a pas été testée).

Une plate-forme FPGA contient normalement un seul processeur. Tous les calculs sont donc concentrés en un seul point (les autres composants ne demandant que peu de performances). Ceci ne nous convient pas pour faire des tests, car nous ne pouvons paralléliser un unique processeur, ce qui revient à effectuer l'essentiel des calculs sur en séquentiel dans les deux cas.
J'ai donc créé un deuxième processeur microblaze, semblable au précédent. Les deux processeurs s'occupent chacun d'une moitié de la surface du jeu de la vie, ce qui permet de diviser les calculs.
Il y a donc une double parallélisation. La première se fait au niveau de l'exé-

cution du jeu de la vie : deux processeurs de type Microblaze rafraîchissent l'écran. Cette parallélisation est présente dans les deux tests. La seconde parallélisation est au niveau de la simulation. Dans le premier test, tout les composant son simulés par un seul processeur physique. Dans le deuxième test, chaque Microblaze est simulé par un processeur, et tous les autres composants sont simulés par un troisième.

3.2 Résultats des tests

Il ne reste donc plus qu'à chronométrer la vitesse d'exécution des différents tests. Pour cela, j'ai mesuré deux valeurs de temps différents : le temps réel d'exécution, ainsi que le temps de processeur consommé.
En théorie, le temps de processeur consommé est sensé être approximativement le même, car les deux tests doivent effectuer la même quantité de calculs. Par contre, on peut supposer que le temps d'exécution sera de moitié moindre, car deux processeurs exécutent de manière simultanée ces calculs.
Les résultats ne sont pas ceux attendus...
Pour une seconde de temps simulé, le test avec la version originale de SystemC a pris 52.42 secondes de temps CPU pour 52.68 secondes de temps réel. Cette différence de temps s'explique facilement si l'on sait que le processeur a sûrement eu d'autres programmes à gérer, indépendants de la simulation.
Le test avec la version modifiée de SystemC a pris 102.82 secondes de CPU et 56.72 secondes de temps réel. Le temps de CPU consommé est donc bien le double du temps réel, ce qui montre que deux processeurs travaillent sur le programme en même temps.
Toutefois, en comparant les deux résultats, on s'aperçoit que le temps de CPU consommé a doublé entre les deux tests. Au lieu des 52 secondes de la version originale, le calcul a pris 102 secondes de temps processeur pour le deuxième test.

3.3 Explication des résultats

Pour comprendre d'où vient l'erreur qui nous a donné ces valeurs, nous avons fait un troisième test. Ce test utilise la version modifiée de SystemC, toujours avec la solution du global quantum, mais nous y avons mis tous les composants sur une seule partition. Donc un seul processeur physique simule la plate-forme FPGA.
Ce test devrait en théorie être quasiment semblable au premier test. C'est ce qui s'est révélé être exact, car il a pris 61.43 secondes de temps CPU pour 61.59 secondes de temps réel. Compte tenu du fait que le troisième test doit gérer le global quantum, contrairement au premier test, on peut considérer

que les deux tests on prit le même temps.
Pour expliquer le surplus de temps consommé par les CPUs dans le deuxième test, on peut suggérer l'hypothèse que les mutex génèrent de l'attente active. S'il y a beaucoup de verrous, les CPU consomment du temps de calculs, mais celui-ci n'avance pas pour autant.

Chapitre 4

Conclusion

Au cours de mon travail, j'ai donc testé les modifications apportés par Samuel Jones [1] à la librairie SystemC. Ces tests ont montré que la simulation se fait effectivement en parallèle, mais il a aussi mis en évidence le problème des mutex, qui pouvaient gêner les processeurs en les faisant faire de l'attente active, et les faire consommer beaucoup plus de temps de calcul que s'il n'y avait pas du tout de parallélisation.
Nous avons pu accélérer la simulation en multipliant le nombre de processeurs sur le travail à faire, mais nous avons ainsi généré un autre problème : celui des mutex. Il serait utile de faire d'autres tests afin de savoir si ce problème se reproduit sur beaucoup de plate-forme, et si c'est le cas, essayer de trouver une solution afin de libérer au maximum les mutex.

Bibliographie

[1] Samuel Jones. Optimistic parallelisation of systemc. Master's thesis, Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UJF, 2011.