Ordonnancement Spatio-Temporel 3D minimisant le coût de communications entre tâches

Quang-Hai KHUAT, Quang-Hoa LE, Daniel CHILLET, Antoine COURTAY, Emmanuel CASSEAU

Université de Rennes 1, IRISA/INRIA, ENSSAT
6 rue de Kerampont, 22305 Lannion Cedex, France

{Quang-Hai.Khuat, Quang-Hoa.Le, Daniel.Chillet, Antoine.Courtay, Emmanuel.Casseau}@irisa.fr

Résumé − L'avènement des architectures 3D offre des perspectives intéressantes en termes de capacité d'intégration. Basées sur l'empilement plusieurs couches de silicium les unes sur les autres, ces architectures modifient la problématique d'ordonnancement et de placement des tâches d'une application sur les ressources d'exécution disponibles. En effet, la prise en compte de la troisième dimension rend le problème plus difficile à résoudre, et cette problème se complexifie encore lorsque l'on considére les communications entre les tâches. Dans cet article, nous proposons une stratégie de placement spatio-temporel dont l'objectif consiste à minimiser le coût global de communication sur une architecture en trois dimensions composée d'une couche multi-processeurs et d'une couche reconfigurable, architecture dite 3D MultiProcesseur Reconfigurable SoC (3DMPRSoC). Nous montrons que notre algorithme permet de mieux exploiter une architecture 3D (en couches) en effectuant un ordonnancement spatio-temporel réduisant le coût des communications d'environ 17% par rapport à un placement naïf des tâches.

Abstract − The advent of 3D architectures offers interesting perspectives in terms of integration capacity. Based on stacking layers of silicon on top of each other, 3D architectures change the problem of task scheduling and task placement of an application on the available execution resources. In fact, the inclusion of the third dimension makes the problem more difficult to solve, and this problem becomes even more complex when we must consider communications between tasks. In this paper, we propose an spatio-temporal scheduling and placement algorithm whose objective is to minimize the overall communication cost of an application on a 3D architecture called 3D multiprocessor architecture reconfigurable System-on-Chip (3DMPRSoC) composed of a multi-processors layer and a reconfigurable layer connected through Through Silicon Vias (TSVs). Our proposal shows up to a 17% reduction in communication cost compared to the naive task placement.

1 Introduction

Les technologies 3D, au travers de l'empilement vertical de couches de silicium, permettent d'envisager des systèmes dans lesquels l'hétérogénéité peut assurer de hautes performances et une bonne efficacité énergétique [1]. De nombreux avantages peuvent être attendus par l'utilisation de ces technologies 3D : la réduction de la longueur des interconnexions et l'adaptation aux systèmes hétérogènes sont de bons exemples des avantages directs offerts [2] [3]. Néanmoins, pour permettre une gestion efficace de ces technologies, des techniques innovantes pour l'ordonnancement spatio-temporel des tâches doivent être développées. C'est en particulier le cas pour la prise en compte les communications à assurer entre les tâches afin que les applications puissent exploiter pleinement les plateformes d'exécution. Pour tenter de répondre à cet objectif, nous avons développé un algorithme réalisant l'ordonnancement spatio-temporel pour une architecture basée sur deux couches de silicium, l'une composée d'un multiprocesseur (CMP), et l'autre composée d'une ressource reconfigurable dynamiquement (eFPGA). Le modèle de communication proposé s'appuie sur des échanges au travers des *Through Silicon Vias* (TSVs) entre la couche reconfigurable et la couche multi-processeurs, et des communications au travers d'un réseau de type grille 2D sur la couche multi-processeurs et d'un réseau 2D similaire sur la couche reconfigurable.

L'objectif de notre algorithme consiste à proposer un placement des tâches minimisant les coûts des communications. Nous posons comme hypothèse que ce coût est lié aux temps de transfert entre tâches et est donc lié au nombre de données propagées entre eux lors des communications. Notre proposition vise donc à placer les tâches qui communiquent sur la même ressource lorsque cela est possible ou de réduire la distance entre les tâches en les plaçant au plus près l'une de l'autre si le placement sur la même ressource n'est plus possible. Dans le cas où plusieurs communications doivent être assurées au même moment, nous privilégions le placement des communications mettant en jeu les plus grandes quantités de données. Cette stratégie de placement de tâches permet d'obtenir plusieurs avantages, par exemple : simplification du routage ; réduction du nombre de "points chauds" (ou de congestion) qui peuvent apparaître dans le réseau de communication ; réduction du délai de communication entre tâches ; réduction de l'énergie consommé pour la communication ;

Les auteurs dans [4] ont utilisé les heuristiques *Nearest Neighbor* et *Best Neighbor* pour optimiser le coût de communication et ont montré les avantages en termes de temps d'exécution globale, de charge moyenne du réseau de communication, de puissance consommée en réduisant ce coût. Toutefois, leur plateforme est basée sur une architecture multi-processeur classique en 2 dimensions, et la

stratégie proposée ne prend pas en compte de la quantité de données échangée durant les communications.

Cet article est organisé comme suit. Les deux premières parties de la section 2 présentent l'architecture, le modèle de tâches et la stratégie de notre algorithme. La partie 2.3 détaille notre algorithme. La partie 3 présente des résultats et des comparaisons entre notre solution et la solution produite par un ordonnancement naïf. Enfin, la section 4 conclut cet article et présente quelques perspectives.

2 Ordonnancement Spatio-Temporel

2.1 Architecture 3D et modèle de tâches

Dans l'article [5], la proposition d'architecture 3DMPR-SoC est définie par l'empilement d'une couche reconfigurable eFPGA et d'une couche multi-processeurs (CMP). Chaque cœur de processeur de la couche CMP est supposé relié à un *Partialy Reconfigurable Region (PRR)* de la couche eFPGA, la surface de ce PRR est supposée identique à la surface du processeur associé. La communication entre les deux couches est réalisée via des TSVs (Fig 1.b). L'application est décrite par un graphe de tâches pour lequel chaque tâche T_i est composée de deux parties : matérielle (T_si) et logicielle (T_hi), voir Figure 1.a. La mise en place d'une partie matérielle pour une tâche est généralement liée à un objectif d'accélération de certains traitements de l'application. Aussi, afin d'assurer un fort couplage, les parties T_si et T_hi d'une tâche applicative T_i sont placées en vis-à-vis. Cette hypothèse a été exploitée dans [5], et la technique utilisée pour le placement des tâches suppose que seules les communications entre les parties matérielles des tâches sont considérées. Le placement de T_si est alors une simple projection de T_hi sur la couche multi-processeur. Plus précisément, pour un exemple de graphe tel que celui de la figure 1.a, l'algorithme produit le placement en vis-à-vis présenté dans la figure 1.c. Cet algorithme minimise le coût de communication sur la couche eFPGA mais ignore les communications à assurer sur la couche CMP. Cependant, pour une application complète, les communications sur la couche CMP, ainsi que celles entre la couche eFPGA et CMP peuvent avoir un impact important sur le coût global de communication de l'application. Nous proposons alors d'étendre les travaux précédents et prenant en compte le coût des communications sur la couche CMP et le coût des communications entre les deux couches à travers des TSVs.

2.2 Stratégie

L'objectif de notre algorithme est de trouver la meilleure instanciation possible des tâches (logicielle et matérielle) sur les ressources d'exécution (processeur et PRR) afin de minimiser le coût global de communication d'une application. Nous proposons d'étendre la formulation de la problématique présentée dans [5] en intégrant la minimisation de l'ensemble des types de communications présents

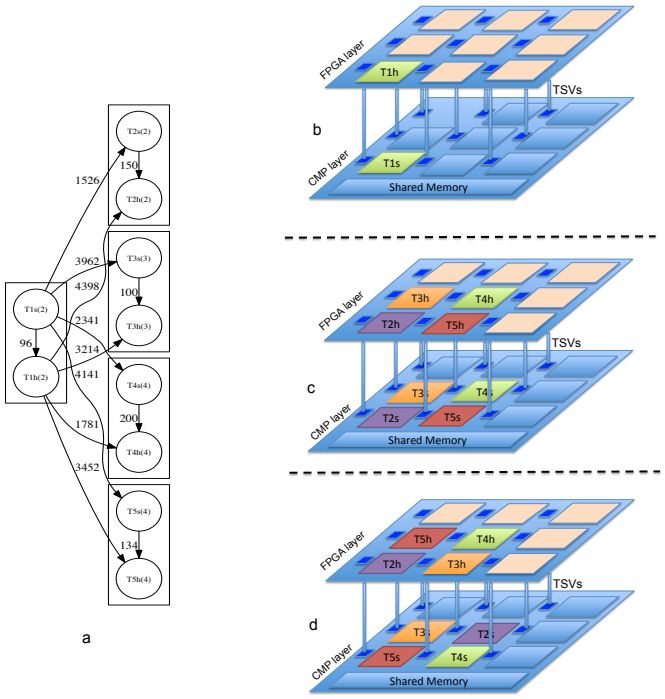

FIGURE 1 – -a- Exemple d'une partie de graphe de tâches pour lequel il existe des dépendances entre les tâches (lorsque T1 termine son exécution, elle envoie des données aux tâches T2, T3, T4 et T5). -b- Placement initial de T1. -c- Placement de tâches généré par l'algorithme $SpatioTemporalScheduling$ dans [5] en respectant le vis-à-vis vertical entre la partie logicielle et la partie matérielle d'une même tâche. -d- Placement de tâches généré par notre algorithme dit $3DTasksPlacement$.

dans le système : **1**) Communication entre deux parties logicielles de deux tâches différentes sur la couche CMP ($T_si \Rightarrow T_sj$). **2**) Communication entre deux parties matérielles de deux tâches différentes sur la couche eFPGA ($T_hi \Rightarrow T_hj$). **3**) Communication entre la partie logicielle et la partie matérielle de la même tâche à travers des TSVs ($T_sj \Rightarrow T_hj$ ou $T_hj \Rightarrow T_sj$). Pour chaque type de communication, le coût de communication est estimé comme le produit de la distance par le nombre de données échangées entre les deux parties. Le coût de communication entre la tâche T_i et la tâche T_j est la somme des trois types de communication et ce coût est calculé par l'équation suivant :

$$Com_{T_i,T_j} = \sum_{p\in\{s,h\}} \sum_{q\in\{s,h\}} Dist_{T_pi,T_qj} * Nbdata_{T_pi,T_qj} \quad (1)$$

La stratégie utilisée consiste alors à évaluer les communications les plus coûteuses en premier afin de placer les tâches impliquées dans l'échange le plus proche l'une de l'autre. Notamment, nous privilégions le placement d'une tâche réceptrice d'une communication au même emplacement que la tâche émettrice afin de réduire le transfert de données. S'il s'agit d'une communication de type 1 (respectivement de type 2), la tâche réceptrice sera placée sur le même processeur (respectivement sur le même PRR) que la tâche émettrice. Dans ce cas, les données sont stockées dans une mémoire locale et il n'y a pas de communication à travers un réseau (sur le eFPGA et/ou sur le

CMP). Nous considérons donc que le coût de communication dans ce cas là est égal à zéro.

Pour une communication de type 3, les parties logicielle et matérielle de la tâche seront placées soit en vis-à-vis (verticalement alignée sur la couche reconfigurable et la couche processeur), ou verticalement décalée. Le choix de l'alignement vertical n'est plus implicite comme cela était le cas dans [5], mais sera pris par notre algorithme lorsque cela le justifie. Notre algorithme peut alors produire des solutions pour lesquelles certaines couples de tâches (logicielle et matérielle) sont alignées et d'autres ne le sont pas. La figure 1.d montre un exemple de placement pour lequel notre algorithme trouve une solution où chaque couple de tâches est verticalement décalé.

2.3 3DTasksPlacement Algorithme

Cette partie représente notre algorithme dite **3DTasksPlacement**. Cet algorithme est exécuté à chaque instant t où une (ou plusieurs) tâche(s) T_i (s) atteint sa (ses) fin d'exécution et commence à envoyer les données à d'autres tâches. À cet instant, une liste des communications produites par la tâche T_i (T_si et T_hi) vers toutes ses tâches réceptrices est établie, nous appelons cette liste L. À l'initialisation, la liste L ne contient que les communications de type 1($T_si \Rightarrow T_sj$) et type 2 ($T_hi \Rightarrow T_hj$), les communications de type 3 ($T_sj \Rightarrow T_hj$ ou $T_hj \Rightarrow T_sj$) seront ajoutées ultérieurement si seulement si les communications de type 1 ou/et de type 2 correspondants ont déjà eu lieu, c-a-d si le placement de la tâche émettrice (T_sj ou T_hj) pour les communications de type 3 a été fixé.

L est ensuite à nouveau triée suivant le nombre de données échangées dans les communications pour créer une nouvelle liste L_+. Tant que la liste L_+ n'est pas vide, nous extrairons la première communication dans la liste et nous appliquons la stratégie présentée afin de trouver le placement pour la tâche réceptrice de cette communication.

La première communication de L_+ peut être de type 1, 2 ou 3. Notre algorithme prend en compte tous ces types et pour chaque type, le placement est le suivant :

- Ligne [9-19], type 1 ($Ts_i \Rightarrow Ts_j$) : si le processeur P sur lequel Ts_i a été exécutée est libre, Ts_j est placée sur P et P devient occupé. Si le processeur P n'est pas libre, nous recherchons le processeur disponible le plus proche par rapport à P pour placer Ts_j. Une fois que la décision de placement pour Ts_j a été prise, la communication verticale entre Ts_j et Th_j (type 3) est ajoutée dans $L+$ et la liste est à nouveau triée. Parce que Ts_j a été décidément placée, tous les communications depuis les autres tâches émettrices logicielles Ts_* vers Ts_j doivent être pris en compte pour le calcul de coût de communication et puis être supprimées de la liste $L+$.

- Ligne [21-31], type 2 ($Th_i \Rightarrow Th_j$) : la même stratégie que le type 1 est appliquée pour ce type de communication, sauf que Th_i et Th_j sont exécutées sur des PRRs et non pas des des processeurs.

Algorithm 1 Cette fonction est exécutée à chaque fois qu'une (ou plusieurs) tâche(s) finit(s) son(ses) exécution(s)

```
1:  function 3DTasksPlacement {
2:  L est la liste des communications : {(Tsi ⇒ Tsj), (Thi ⇒ Thj)}
3:  L+ est la liste des communications de L triées dans l'ordre décrois-
    sant du nombre des données
4:
5:  while L+ ≠ ∅ do
6:      (T*i ⇒ T*j) = {L+}.ExtrairePremierElement
7:      L+.Supprimer (T*i ⇒ T*j)
8:      Switch (T*i ⇒ T*j) {
9:        Case (Tsi ⇒ Tsj) :
10:         if (Proc[Assigner[Tsi]] = 0) then
11:             Assigner[Tsj] = Assigner[Tsi]
12:             Proc[Assigner[Tsj]] = 1
13:         else
14:             Assigner[Tsj] = TrouverLeProcesseurLePlusProche[Tsi]
15:             Proc[Assigner[Tsj]] = 1
16:             ComCost += data (Tsi ⇒ Tsj) * distance (Tsi ⇒ Tsj))
17:         end if
18:         L+.AjouterPuisTrier (Tsj ⇒ Thj)
19:         L+.Supprimer (Ts* ⇒ Tsj)
20:
21:   Case (Thi ⇒ Thj) :
22:       if (PRR[Assigner[Thi]] = 0) then
23:           Assigner[Thj] = Assigner[Thi]
24:           PRR[Assigner[Thj]] = 1
25:       else
26:           Assigner[Thj] = TrouverLePRRLePlusProche[Thi]
27:           PRR[Assigner[Thj]] = 1
28:           ComCost += data (Thi ⇒ Thj) * distance (Thi ⇒ Thj))
29:       end if
30:       L+.AjouterPuisTrier (Thj ⇒ Tsj)
31:       L+.Supprimer (Th* ⇒ Thj)
32:
33:   Case (Tsj ⇒ Thj) :
34:       if  (PRR[[Assigner[Tsj]] = 0) then
35:           Assigner[Thj] = Assigner[Tsj]
36:           PRR[[Assigner[Thj]] = 1)
37:       else
38:           Assigner[Thj] = TrouverLePRRLePlusProche[Tsi]
39:           PRR[[Assigner[Thj]] = 1)
40:           ComCost -= (data (Tsj ⇒ Thj) * distance (Tsj ⇒ Thj))
41:       end if
42:       L+.Supprimer (Th* ⇒ Thj)
43:
44:   Case (Thj ⇒ Tsj) :
45:       if  (Proc[[Assigner[Thj]] = 0)  then
46:           Assigner[Tsj] = Assigner[Thj]
47:           Proc[[Assigner[Tsj]] = 1
48:       else
49:           Assigner[Tsj] = TrouverLeProcesseurLePlusProche[Thi]
50:           Proc[[Assigner[Tsj]] = 1
51:           ComCost -= data (Thj ⇒ Tsj) * distance (Thj ⇒ Tsj)
52:       end if
53:       L+.Supprimer (Ts* ⇒ Tsj)
54:     }
55: end while
56: }
```

- Ligne [33-42], type 3 ($Ts_j \Rightarrow Th_j$) : pour ce type de communication entre la partie logicielle et matérielle à travers des TSVs, l'idéal sera de pouvoir placer Th_j en face de Ts_j si le PRR en face du processeur P où Ts_j a été exécutée est encore libre. Si ce PRR n'est pas disponible, nous recherchons le PRR disponible le plus proche par rapport à P. Dans ce cas, le coût de communication doit aussi être calculé.

- Ligne [44-53], type 3 ($Th_j \Rightarrow Ts_j$) : la même stratégie que le type 3 ($Ts_j \Rightarrow Th_j$) est appliquée pour ce type de communication.

Pour chaque communication, si le placement de la tâche réceptrice est fixé, cette communication est supprimée de

$L+$ et l'algorithme continue jusqu'à ce que $L+$ soit vide.

La figure 1.d montre la solution générée par notre algorithme pour une partie de l'application présentée dans 1.a. À l'initialisation, T1 est exécutée toute seule avec ses deux parties (logicielle et matérielle) placées face-à-face (figure 1.b). Une fois que T1 achève son exécution, elle envoie des données à T2, T3, T4 et T5 et puis elle est supprimée de l'architecture. Au temps suivant, les placements de T2, T3, T4 et T5 produits par notre algorithme $3DTasksPlacement$ sont présentés dans la figure 1.d où deux parties d'une tâche sont verticalement décalées.

3 Résultat

Cette section présente quelques résultats obtenus en appliquant notre stratégie. Afin d'évaluer la qualité de ces résultats, nous comparons le coût de communication global de notre solution avec la technique développée dans [5]. Ces résultats sont obtenus pour une plateforme 3D simplifiée contenant 4 PRRs et 4 processeurs. Pour produire ces résultats, nous avons généré des graphes de tâches orientés acycliques, c-a-d des graphes orientés qui ne possèdent pas de cycle. Chaque partie d'une tâche dispose d'au moins un prédécesseur sauf pour les tâches racines.

On appelle R la proportion de données entre la communication horizontale (type 1 or 2) et la communication verticale (type 3). Dans notre contexte, les applications qui contiennent une faible valeur de R ou R proche de 1 ne sont pas très réalistes car cela signifierait qu'un grand nombre de données transiteraient au travers des TSVs alors que peu de données seraient échangées sur la couche CMP et/ou eFPGA. Bien que notre architecture puisse supporter ce type d'échanges, nous supposons que la communication entre les couches au travers des TSV est réservée au contrôle entre la partie logicielle et la partie matérielle d'une tâche, ce contrôle ne nécessitant qu'un faible volume de données à échanger. C'est la raison pour laquelle nous nous intéressons uniquement aux graphes avec une grande valeur de R. Dans ce travail, nous avons généré différents exemples de graphes ayant un nombre de tâches variant entre 8 et 20 et avec une valeur de R proche de 100. Le nombre de données de type 1 et 2 est généré dans l'intervalle [10000, 50000] alors que le nombre de données de type 3 est compris dans l'intervalle [100, 500]. Le temps d'exécution pour chaque tâche varie de 2 à 6 unités de temps.

La figure 2 compare les résultats produits par notre algorithme avec ceux produit par une solution forcant le placement en vis-à-vis des tâches. Sur cette figure, nous constatons que notre algorithme produit des solutions dont le coût de communication est toujours plus faible que pour les solutions de placement en vis-à-vis.

Ces résultats montrent que le coût des 3 types de communication jouent un rôle important et que la prise en compte du problème de communication dans sa globalité permet une réduction non négligeable, de l'ordre de 17%.

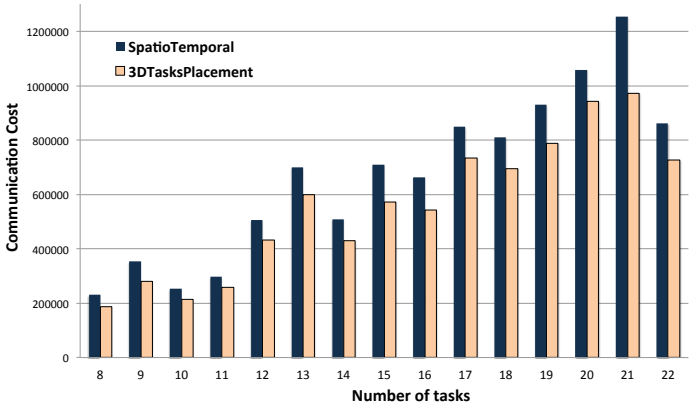

FIGURE 2 – Comparaison des coûts de communication générés par la solution de placement en Vis-à-Vis et notre algorithme.

Ce chiffre est cohérent à ce que nous attendons car notre algorithme permet plusieurs choix possibles pour le placement alors que le placement des tâches en vis-à-vis réduit la flexibilité en imposant un placement des parties logicielles et matérielle des tâches face-à-face.

4 Conclusion

Dans cet article, nous avons adressé la problématique de l'ordonnancement spatio-temporel de tâches pour une architecture 3DMPRSoC. L'objectif a consisté à développer un algorithme qui minimise le coût global des communications entre les tâches exécutées sur cette plateforme. Les résultats montrent que notre algorithme permet de réduire le coût moyen de communication de l'ordre de 17% sur des graphes générés aléatoirement. Ces travaux seront prolongés en proposant de gérer des zones reconfigurables de tailles variables. Dans ce contexte, la taille d'une tâche matérielle n'est donc plus limitée par la taille d'un PRR mais peut être plus petite ou plus grande en fonction des besoins applicatifs. Pour répondre à cette nouvelle problématique, nous étudions actuellement les méthodes de gestion d'espaces libres (de type Vertex List ou KAMER) en tenant compte à la fois des communications mais aussi de l'aspect 3D de l'architecture.

Références

[1] *Cong.J, Guruaj.K, Huang.M, Li.S, Xiao.B and Zou.Y*, "Domain-specific processor with 3d integration for medical image processing". In : Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on, pp. 247 –250.

[2] *Xie (Y.) and Ma (Y.)*. "Design space exploration for 3d integrated circuits". In : Solid-State and Integrated-Circuit Technology, 2008. ICSICT 2008. 9th International Conference on, pp. 2317 –2320.

[3] *Xue, Licheng, and al.* "3D floorplanning of low-power and area-efficient Network-on-Chip architecture". In : Microprocessors and Microsystems 35.5 (2011) : 484-495

[4] *Singh.A. K, Jigang W, and al.* "Run-time mapping of multiple communicating tasks on MPSoC platforms". In : Procedia Computer Science 1.1 (2010) : 1019-1026.

[5] *Q.H Khuat, Q.H Le, D.Chillet and S.Pillement.* "Spatio-Temporal Scheduling for 3D Reconfigurable & Multiprocessor Architecture". In : 7th International Design and Test Symposium, IDT2012, Doha, Quatar - December 2012.