Optimisation et Génération d'Exécutifs Distribués Temps-Réel pour Algorithmes spécifiés avec les Langages Synchrones

C. Lavarenne, Y. Sorel

INRIA Domaine de Voluceau – Rocquencourt B.P.105
78153 Le Chesnay Cedex – France email : yves.sorel@inria.fr

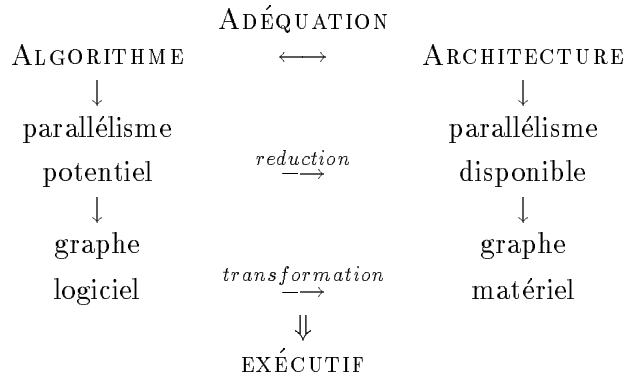
RTS'94 – Paris, 11–14 Janvier 1994

1 Introduction

La complexité grandissante des applications temps-réel nécessite à la fois des outils de spécification de haut niveau et des architectures multi-processeur [1, 2]. Afin de réduire le nombre d'erreurs de spécification des algorithmes et de limiter au maximum les tests matériels, de nouvelles méthodes sont proposées. Elles permettent à l'utilisateur de se concentrer sur les aspects temporels qui sont cruciaux dans le domaine du temps réel (réactivité du programme et temps de réponse contraint), d'étudier les relations entre le parallélisme potentiel au niveau de l'algorithme et celui disponible au niveau de la machine, enfin d'être déchargé de la programmation de bas niveau (exécutif temps-réel) souvent fastidieuse.

Les Langages Synchrones [3] permettent de spécifier un algorithme puis de vérifier si l'ordre des évènements qu'il induit, qui peut être vu comme un temps logique, est cohérent, ceci indépendamment de toute implantation matérielle. On cherche ensuite à réaliser une implantation de l'algorithme sur l'architecture en respectant dans le temps physique, d'une part l'ordre des évènements vérifié lors de la spécification et d'autre part les contraintes d'exécution temps-réel. L'implantation consiste à réduire le parallélisme potentiel de l'algorithme au parallélisme matériel disponible. C'est le rôle de l'exécutif d'allouer les ressources matérielles aux besoins de l'algorithme.

Nous utilisons une méthode basée sur des modèles de graphes pour spécifier l'algorithme et l'architecture, conduisant à formaliser l'implantation en termes de transformations de graphes. Le graphe flot de donnée modélisant l'algorithme est transformé progressivement jusqu'à ce qu'il corresponde au graphe matériel modélisant l'architecture. Ces transformations représentent une distribution (allocation spatiale) et un ordonnancement (allocation temporelle) des calculs sur les processeurs et des communications sur les liaisons physiques inter-processeurs. C'est à partir de cette distribution et de cet ordonnancement qu'un exécutif distribué temps-réel, permettant l'exécution de l'algorithme sur l'architecture, peut



être généré, libérant ainsi l'utilisateur des tâches lourdes de programmation bas niveau et évitant la phase de mise au point temps-réel sur le multi-processeur cible.

Une implantation efficace est obtenue en réalisant une *Adéquation Algorithme/Architecture*. Pour cela, on choisit parmi toutes les transformations possibles celle qui optimise les performances temps-réel. Nous avons développé une heuristique d'optimisation [4] basée sur des calculs dans l'algèbre $(\max, +)$ utilisant les durées d'exécution des sommets du graphe flot de données et des communications inter-processeur. Ces durées sont elles-mêmes calculées grâce au modèle matériel, paramétré par les caractéristiques des composants de l'architecture.

Cette méthode est assez générale pour prendre en compte tous les types d'exécutifs, depuis le tout dynamique jusqu'au tout statique. On présente deux exemples d'exécutif générés, le premier avec distribution statique et ordonnancement dynamique des calculs et des communications le second avec distribution et ordonnancement statiques. Pour chaque type d'exécutif, on utilise un noyau générique d'exécutif demandant des modifications minimales lorsqu'on introduit une nouvelle architecture. On présente enfin SynDEx, un prototype d'environnement graphique interactif supportant cette méthode d'Adéquation Algorithme/Architecture et générant automatiquement les exécutifs.

2 Modèles Algorithme et Architecture

L'algorithme est modélisé par un *Graphe Flot de Données Conditionné* (hypergraphe orienté), dont chaque sommet représente une *action*, de calcul, d'entrée-sortie, de mémorisation ou de conditionnement, et dont chaque arc représente un transfert itératif de données (*flot de données*), établissant une précédence entre deux actions. Ce modèle met en évidence le parallélisme potentiel de l'algorithme (ordre partiel induit par les précédences du graphe) et la mémoire d'état. Les sommets sans prédécesseur représentent les interfaces d'entrée gérant les stimuli produits par l'environnement. Les sommets sans successeur représentent les interfaces de sortie générant les réactions aux stimuli.

L'architecture est modélisée par un hypergraphe non orienté représentant un réseau de processeurs MIMD ou SPMD, dont chaque sommet est un processeur et chaque hyper-arc

est une liaison physique de communication bidirectionnelle qui permet des transferts de données entre les mémoires des processeurs, au besoin par l'intermédiaire d'une mémoire commune.

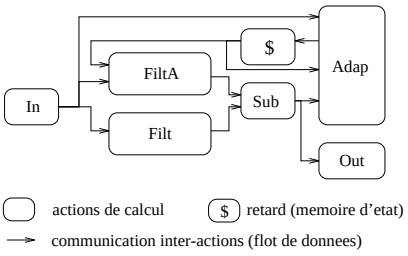


Figure 1: Exemple de graphe flot de données

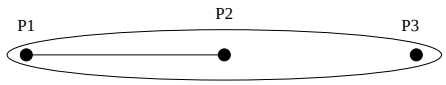


Figure 2: Exemple d'architecture

3 Implantation

L'exécutif a pour rôle d'allouer les ressources de l'architecture matérielle au programme d'application qui code l'algorithme. Les exécutifs peuvent être classés en fonction de leur manière d'arbitrer l'allocation des ressources. Les ressources sont de trois types, les unités de calcul, de mémoire et de communication de chaque processeur. L'allocation des ressources par l'exécutif est à la fois spatiale (distribution) et temporelle (ordonancement). Si les optimisations et les décisions, ou arbitrages, que doit prendre l'exécutif sont effectuées pendant l'exécution, avec l'aide de l'horloge temps réel, on dit que l'allocation est dynamique. Si cela est fait avant l'exécution, à la compilation, en ayant connaissance a priori des durées d'exécution, l'allocation est dite statique. On peut ainsi classifier les exécutifs temps réel distibués disponibles actuellement en fonction des quatre types d'allocation pour chacun des trois types de ressources (c.f. tableau ci-dessous).

Exemples d'exécutifs	calcul		mémoire		communic.	
	dist	ord	dist	ord	dist	ord
Meiko	dyn	dyn	dyn	dyn	dyn	dyn
CHORUS	sta	dyn	sta	dyn	dyn	dyn
SynDEx v1	sta	dyn	sta	dyn	sta	dyn
SynDEx v2	sta	sta	sta	dyn	sta	dyn
minimum	sta	sta	sta	sta	sta	sta

Une partie des fonctionnalités de l'exécutif peut être supportée au niveau matériel. Par exemple, dans le cas du Transputer T9000, toutes les fonctionnalités, l'ordonnancement des actions de calcul et de communication et même le routage des communications (VCP et C104) sont supportées au niveau matériel, et donc de manière dynamique puisque indépendamment de l'application. Par contre, dans le cas de la plupart des processeurs comme par exemple le TMS320C40, toutes les fonctionnalités de l'exécutif doivent être supportées au niveau logiciel.

Dans les deux chapitres suivants, on montre à travers deux exemples comment l'exécutif peut être supporté au niveau logiciel dans le cas d'une distribution statique et d'un ordonnancement soit dynamique (de type "SynDEx v1", ligne 3 du tableau), soit statique (de type "minimum", ligne 5 du tableau). A travers ces exemples, on montre comment la distribution et l'ordonnancement se formalisent en termes de transformations de graphes et on décrit les composants du noyau générique d'exécutif logiciel utilisé pour combler les fonctionnalités qui ne sont supportées ni au niveau matériel ni au niveau d'un éventuel exécutif résidant sur chaque processeur.

4 Exemple d'Exécutif Dynamique

Dans le cas d'un exécutif avec distribution statique et ordonnancement dynamique des calculs et des communications, on détermine la distribution avant l'exécution et on laisse le soin à l'exécutif de décider de l'ordonnacement à l'exécution.

Avec les ensembles $\mathcal{A}$ des actions, $\mathcal{D}$ des dépendances inter-actions, $\mathcal{P}$ des processeurs et $\mathcal{L}$ des liaisons physiques de communication inter-processeurs, la distribution consiste à partitionner $\mathcal{A}$ en autant d'éléments de partition $\mathcal{A}_p$ qu'il y a de processeurs. Les arcs inter-partitions nécessitent des communications inter-processeurs. Bien que le graphe matériel soit nécessairement connexe, chaque processeur n'est pas forcément connecté directement à tous les autres. En construisant l'ensemble $\mathcal{R}$ de tous les chemins (routes) du graphe matériel $(\mathcal{P}, \mathcal{L})$ on peut distribuer les communications inter-processeurs sur les routes :

$$(\mathcal{P}, \mathcal{L}) \stackrel{routage}{\longrightarrow} (\mathcal{P}, \mathcal{R})$$

$$((\mathcal{A}, \mathcal{D}), (\mathcal{P}, \mathcal{R})) \stackrel{distrib_R}{\longrightarrow} \left(\bigcup_{p \in \mathcal{P}} (\mathcal{A}_p, \mathcal{D}_p), \bigcup_{r \in \mathcal{R}} \mathcal{D}_r \right)$$

$\mathcal{A} \supset \mathcal{A}_p$ = actions exécutées par le processeur p
$\mathcal{D} \supset \mathcal{D}_p$ = dépendances entre actions exécutées par p
$\mathcal{D} \supset \mathcal{D}_r$ = dépendances inter-processeur routées sur r

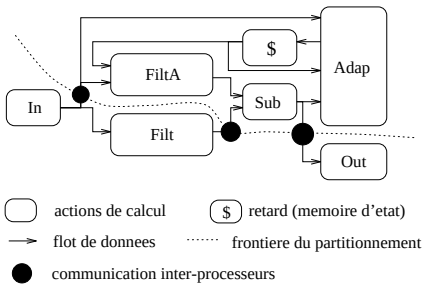


Figure 3: Exemple de distribution

Dans les cas où le routage des communications n'est pas supporté, ni au niveau matériel comme par exemple dans le cas des Transputers T800, ni au niveau d'un exécutif résidant sur chaque processeur comme dans le cas de la version actuelle de C3L pour Transputer

ou TMS320C40, les fonctionnalités supplémentaires à ajouter à l'exécutif pour assurer le routage peuvent se décomposer de la manière suivante :

- PE (Porte d'Emission) pour créer, à partir des données à transmettre, un message avec informations de routage
- PR (Porte de Réception) pour extraire d'un message reçu les données transmises
- BL (Bus Logiciel) pour router les messages entre les différentes portes
- PES (Porte d'Entrée-Sortie) pour tamponner et envoyer et recevoir les messages à travers une liaison physique de communication

La figure 4 présente un exemple de ce qu'il faut ajouter (en grisé) à l'exécutif résidant pour supporter le routage des communications. Cette couche supplémentaire est construite à partir des éléments du noyau générique décrit ci-dessus.

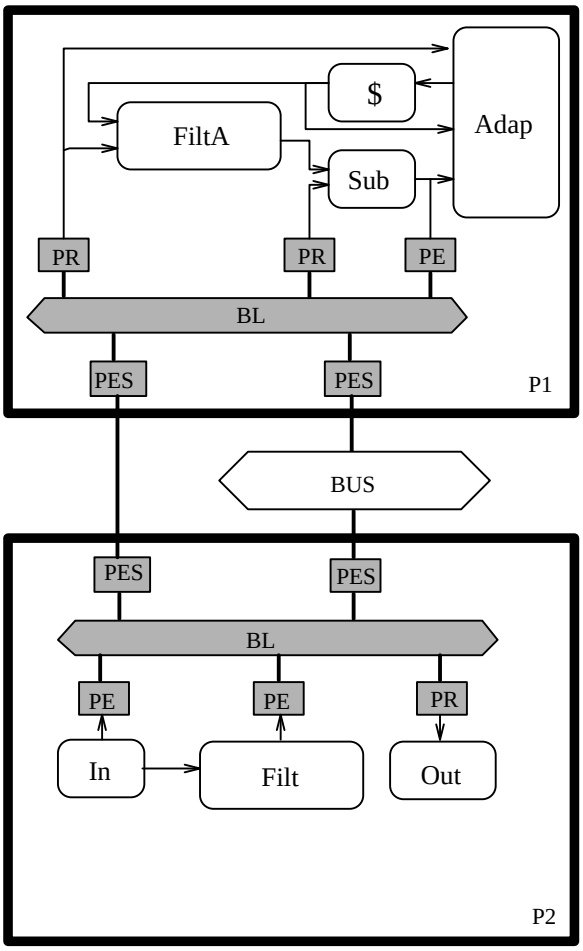


Figure 4: Exécutif dynamique

5 Exemple d'Exécutif Statique

On considère maintenant le cas minimum où la seule fonctionnalité de l'exécutif est l'ordonnancement statique supporté par le séquenceur du processeur. Toutes les actions, de calcul, de mémorisation, de conditionnement, de communication et d'entrée-sortie, sont distribuées et ordonnancées statiquement. Le routage est supporté en décomposant les communications inter-processeurs en fonction des processeurs traversés sur la route.

Chaque arc inter-partition est transformé en une chaîne (graphe linéaire) comportant un sommet pour chaque liaison de la route :

$$\forall r \in \mathcal{R} \quad \forall d_r \in \mathcal{D}_r \quad d_r \stackrel{com}{\longrightarrow} (c_p, a_l, c_{p'}, \ldots a_{l''}, c_{p''})$$

Chaque nouveau sommet a_l est une *action de communication* qui correspond à un transfert de données entre les mémoires de deux processeurs directement connectés par $l \in \mathcal{L}$. En fait toutes les unités de communication connectées à l coopèrent (d'une manière spécifique au matériel) pour exécuter le transfert, donc on peut considérer que chaque action de communication est distribuée sur les unités de communication partageant la liaison physique de communication. Donc les nouveaux arcs c_p sont intra-processeurs mais inter-unités (calcul-communication ou communication-communication ou communication-calcul). En regroupant les a_l d'un même $l \in \mathcal{L}$, et les c_p d'un même $p \in \mathcal{P}$, on obtient les ensembles $\mathcal{A}_l$ et $\mathcal{C}_p$ et la transformation $distrib_R$ devient :

$$((\mathcal{A}, \mathcal{D}), (\mathcal{P}, \mathcal{L})) \stackrel{distrib}{\longrightarrow} \left(\bigcup_{p \in \mathcal{P}} (\mathcal{A}_p, \mathcal{D}_p \cup \mathcal{C}_p), \bigcup_{l \in \mathcal{L}} \mathcal{A}_l \right)$$

La distribution et l'ajout des actions de communication a_l ne modifient pas l'ordre partiel $\mathcal{D}$ du graphe logiciel. Sur chaque processeur, un ordonnancement des actions (autres que de communication) est un ordre total $\bar{\mathcal{D}}_p$ qui inclut l'ordre partiel restreint à $\mathcal{A}_p$ c'est-à-dire $\mathcal{D}_p$. Par ailleurs sur chaque liaison de communication, un ordonnancement des actions de communication est un ordre total $\bar{\mathcal{D}}_l$ qui inclut l'ordre partiel restreint à $\mathcal{A}_l$. Après ordonnancement, la transformation *distrib* devient :

$$((\mathcal{A}, \mathcal{D}), (\mathcal{P}, \mathcal{L})) \stackrel{dist/ord}{\longrightarrow} \left(\bigcup_{p \in \mathcal{P}} (\mathcal{A}_p, \bar{\mathcal{D}}_p \cup \ \mathcal{C}_p), \bigcup_{l \in \mathcal{L}} (\mathcal{A}_l, \bar{\mathcal{D}}_l) \right)$$

Les fonctionnalités d'un exécutif minimum avec distribution et ordonnancement statiques s'expriment en langage C de la manière suivante :

- allocation statique mémoire (déclaration de variables)
- exécution des actions (appels de fonctions) : les actions de calcul et d'entrée-sortie sont fournies par le concepteur applicatif, les actions de communication (`ComPut`, `ComGet`) dont le codage dépend de l'architecture cible, sont fournies par le concepteur du noyau générique d'exécutif
- séquencement des actions : direct (`;`), conditionnel (`if`, `else`), itératif (`while`)

La figure 5 présente un exemple d'exécutif statique.

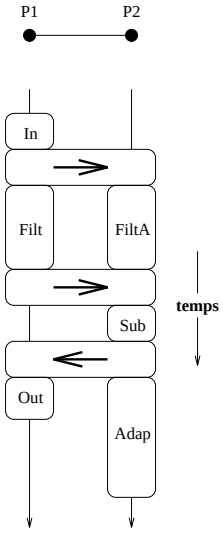


Figure 5: Exécutif Statique

6 Environnement SynDEx

SynDEx [5] est un environnement graphique interactif de développement pour applications temps-réel de traitement du signal et des images et de contrôle de processus, supportant la méthode d'Adéquation Algorithme/Architecture. Il permet la saisie des graphes de l'algorithme et de l'architecture. Le graphe flot de données de l'algorithme peut être également produit par le compilateur du Langage Synchrone SIGNAL et SynDEx sera bientôt compatible avec le *format commun GC* (Graph Code) généré par les Langages Synchrones.

Après exécution de l'heuristique d'optimisation du temps de réponse, un diagramme prévisionnel d'exécution temps-réel de l'algorithme sur le multi-processeur est visualisé. Enfin, un exécutif sans interblocage est généré automatiquement en langage C à partir d'une part des actions de calcul et d'entrée-sortie fournies par l'utilisateur et d'autre part d'un noyau générique des actions de conditionnement, de mémorisation et de communication. Deux implantations du noyau générique ont été développées pour des architectures à base de processeurs de type Transputer ou TMS320C40.

L'exécutif peut être généré optionnellement avec des instructions de chronométrage permettant la mesure des durées d'exécution de toutes les actions [6]. Ces durées mesurées sont nécessaires pour l'optimisation et l'évaluation des performances temps réel. Ces mesures ne sont nécessaires que lorsqu'on introduit un nouveau type de composant matériel (processeur ou liaison de communication) [7].
L'environnement SynDEx est disponible dans sa version 3.1 sur plateformes Sun/X11 et IBM-PC/Windows.

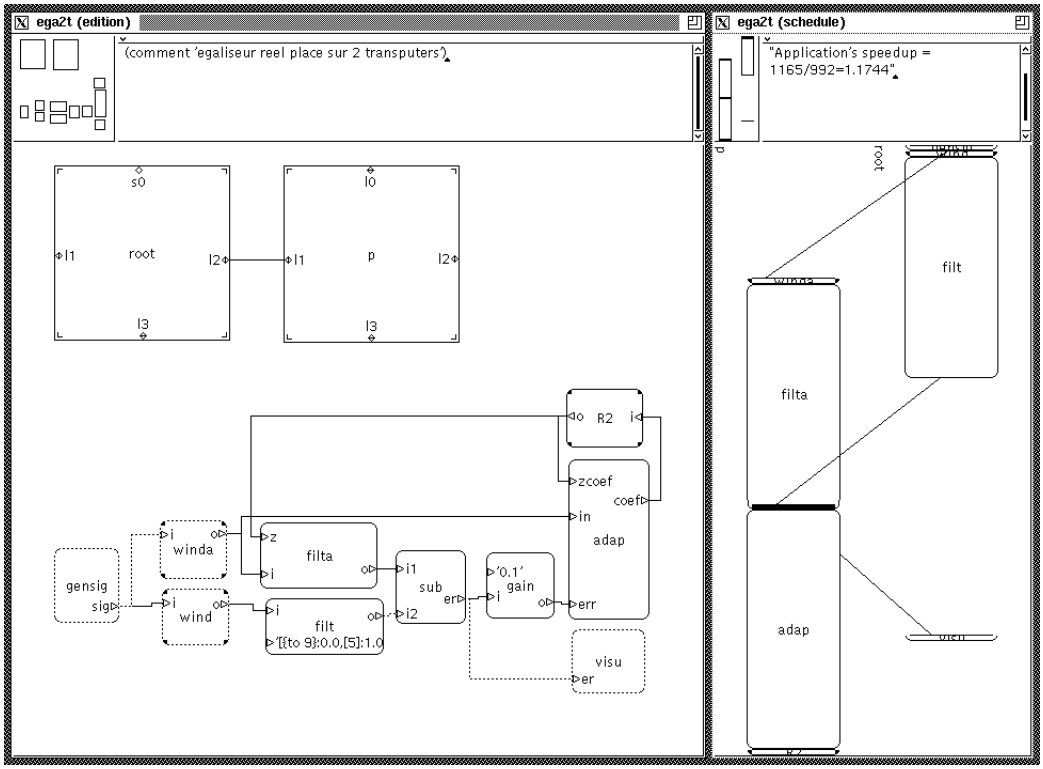


Figure 6: Environnement SynDEx

7 Conclusion

L'originalité de la méthode *Adéquation Algorithme/Architecture* présentée dans cet article réside dans son approche globale utilisant le même formalisme, celui des graphes, pour spécifier l'algorithme, l'architecture multi-processeur et l'implantation.

Cette méthode permet de prendre en compte tous les types d'exécutifs, du tout dynamique au tout statique, et de voir un exécutif comme le résultat d'une transformation de graphes. Cette transformation respecte l'ordre des évènements (respect de la causalité, pas d'interblocage) vérifié lors de la spécification de l'algorithme faite hors contraintes matérielles avec les Langages Synchrones. De plus, la prise en compte des durées d'exécution des actions, utilisées par l'heuristique d'optimisation du temps de réponse, permet de vérifier si les contraintes temps réel sont respectées.

L'environnement logiciel SynDEx supporte cette méthode et permet de générer automatiquement des exécutifs distribués optimisés. Tout cela a pour conséquence une réduction considérable des temps de développement (conception et mise au point) des applications temps réel.

Bibliographie

[1] C. Lavarenne, C. Milan, M. Paindavoine, G. Richard, Y. Sorel
Implantation d'algorithmes de traitement d'images sur une architecture multi-DSP avec l'environnement d'aide à l'implantation SynDEx.
Quatorzième Colloque GRETSI, Juan-Les-Pins, Septembre 1993.

[2] C. Lavarenne, R. Reynaud, Y. Sorel
Spécification et validation à l'aide d'un langage synchrone d'un protocole d'appariement de données asynchrones.
Quartozième Colloque GRETSI, Juan-Les-Pins, Septembre 1993.

[3] A. Benveniste, G. Berry:
The Synchronous Approach to Reactive and Real-Time Systems.
Proc. of the IEEE vol79, n.9, pp.1270–1282, 1991.

[4] C. Lavarenne, Y. Sorel
Performance Optimization of Multiprocessor Real-Time Applications by Graph Transformations.
Parallel Computing 93, Grenoble, Septembre 1993.

[5] C. Lavarenne, O. Seghrouchni, Y. Sorel, M. Sorine:
The SynDEx software environment for real-time distributed systems design and implementation.
Proc. of the European Control Conference, 1991.

[6] F. Ennesser, C. Lavarenne, Y. Sorel:
Méthode chronométrique pour l'optimisation du temps de réponse des exécutifs SynDEx.
INRIA Research Report n°1769, 1992.

[7] C. Lavarenne, Y. Sorel:
Specification, Performance Optimization and Executive Generation for Real-Time Embedded Multiprocessor Applications with SynDEx.
Proc. of Real-Time Embedded Processing for Space Applications, CNES International Symposium, 1992.