TREIZIÈME COLLOQUE GRETSI - JUAN-LES-PINS DU 16 AU 20 SEPTEMBRE 1991

SynDEx
un environnement de programmation pour applications de traitement du signal distribuées

C. Lavarenne, O. Seghrouchni, Y. Sorel, M. Sorine

INRIA, Domaine de Voluceau B.P.105 - 78153 LE CHESNAY CEDEX
email : sorel@seti.inria.fr

RÉSUMÉ

On présente une méthode de programmation pour des applications de traitement du signal en temps-réel sur machines multi-processeur. Cette méthode s'appuie sur le langage synchrone SIGNAL pour spécifier et prouver les algorithmes d'application et sur l'environnement de programmation SynDEx pour générer des exécutifs temps-réel distribués optimisés. Cette méthode est appliquée à un exemple d'égalisation adaptative.

ABSTRACT

We present a programming method for real-time signal processing applications running on multi-processors. This method relies on the synchronous language SIGNAL to specify and prove the application algorithms and on the SynDEx programming environment to generate optimized distributed real-time executives. This method is applied to an example of adaptive equalization.

1 PRINCIPES

1.1 Introduction

La complexité grandissante des applications temps-réel de traitement du signal nécessite à la fois des outils de spécification de haut niveau et des architectures multi-processeur.

Afin de réduire le nombre d'erreurs lors de la 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 paralléllisme possible 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.

SIGNAL[1][2] est un langage temps-réel synchrone, permettant de spécifier les algorithmes puis de vérifier s'ils sont logiquement corrects au niveau temporel (détection d'interblocage).

SynDEx[3][4][5] est un environnement de développement graphique interactif, conçu pour générer des exécutifs temps-réel optimisés pour des programmes d'application spécifiés avec SIGNAL et s'exécutant sur des architectures multi-processeurs.

L'optimisation des exécutifs utilise la description du comportement temps-réel de l'algorithme et de la machine, en prenant en compte les performances des processeurs et de leurs moyens de communication.

1.2 Spécification et preuve des algorithmes d'application

SIGNAL est un langage *flot-de-données synchrone*, qui, après traduction, permet de représenter de nombreux algorithmes par un graphe orienté dans lequel :

- Les Nœuds ou Processus, caractérisés par des ports d'entrée-sortie auxquels sont associés des flots de données, représentent des opérations sur les flots.
- Les Arcs représentent des dépendances entre ports.

Ce modèle flot-de-données permet de décrire facilement le parallélisme, la grosseur des grains (processus) étant laissée au choix de l'utilisateur.

A chaque instant logique où une ou plusieurs entrées externes du graphe sont présentes, certains nœuds sont exécutés. Ces instants définissent un temps logique où les calculs et les résultats qui sont produits sont synchrones avec l'entrée qui les a provoqués. Ceci revient à dire que les durées des calculs et des communications sont nulles dans ce temps logique. On peut ainsi associer à chaque port x un flot de données, encore noté x, et une horloge $h(x)$ (ensemble des instants ou une donnée est présente sur ce port). Un signal X est un couple $(x, h(x))$.

L'exécution d'un programme est déterministe, ce qui simplifie la mise au point, deux exécutions d'un même jeu d'entrées produisant les mêmes sorties.

SIGNAL comporte quatre processus élémentaires que nous décrivons ci-dessous avec le format suivant :

- nom et syntaxe de l'opérateur
  relation sur les flots de données pour tout instant ; équation d'horloges ; dépendance instantannée des signaux.

- fonction immédiate $Y := f(X_1, \ldots X_n)$
  $y = f(x_1, \ldots x_n); h(y) = h(x_1) = \cdots = h(x_n); y$ dépend de x_1 et $\ldots$et x_n.
- retard $ZX := X\$1$
  $zx_n = x_{n-1}$ (zx prend la valeur que x avait à l'instant précédent); $h(zx) = h(x); zx$ ne dépend pas de x.
- sous-échantillonage $Y := X$ when B
  $y = x; h(y) = h(x) \wedge h(b = \text{true}); y$ dépend de x quand b est présent et vrai.
- mélange $X := Y$ default Z
  $x = y$ ou $x = z; h(x) = h(y) \vee h(z); x$ dépend de y quand y présent même si z présent ou x dépend de z quand y non présent (priorité à l'opérande gauche) et z présent.

Les processus sont construits par composition en parallèle d'autres processus ou de processus élémentaires au moyen de l'opérateur "|", les connexions entre ports se faisant par identité de leurs noms.

Un modèle synchrone permet de décrire et vérifier le comportement temporel d'un algorithme. Pour cela, le compilateur SIGNAL tente de résoudre le système d'équations d'horloges des instructions du programme SIGNAL. S'il y arrive, le programme est correct temporellement, sinon, soit il y a trop de contraintes sur les horloges, ce qui entraînerait un interblocage à l'exécution, soit il reste des horloges non déterminées et il faut ajouter des contraintes au moyen de l'opérateur *synchro* qui force l'égalité de plusieurs horloges (dont une et une seule doit être déterminée par ailleurs).

L'exécution d'un programme SIGNAL revient à exécuter un graphe flot-de-données conditionné où chaque nœud et chaque arc est étiquetté par une horloge indiquant les instants auxquels il doit s'exécuter.

1.3 Placement et ordonnancement

Le problème général consiste à placer les processus du graphe flot-de-données conditionné (*graphe logiciel*) sur le réseau de processeurs décrit par un graphe non orienté (*graphe matériel*), à ordonnancer les processus placés sur chaque processeur et à ordonnancer les communications inter-processeurs sur chaque liaison physique de communication.

Le placement et l'ordonnancement sont réalisés simultanément par un algorithme basé sur une heuristique qui vise à minimiser le temps de réponse de l'application, c'est-à-dire le temps d'exécution du graphe placé. Celui-ci est obtenu par simulation du comportement temps-réel de l'application et de la machine cible. Les processeurs ayant la même vitesse de calcul, on donne à chaque processus une durée d'exécution (mesurée isolément sur un processeur). Les liaisons physiques ayant la même vitesse de communication, les durées de communication inter-processeur sont fonction du type de données et du nombre de liaisons traversées (routage). Les durées de communication intra-processeur sont nulles. Ces informations permettent de calculer les dates de début et de fin des processus et celles des communications inter-processeur.

L'algorithme de placement-ordonnancement consiste à faire progresser au long du graphe une coupe séparant les processus déjà placés sur des processeurs de ceux qui ne le sont pas encore. La progression se fait en respectant les précédences du graphe de processus [6]. De tous les processus à placer sur la coupe et de tous les processeurs, on choisit la paire qui optimise une fonction de coût locale prenant en compte :

- la différence entre les dates d'exécution au plus tôt et au plus tard (schedule flexibility [7])
- l'allongement du temps de traversée du graphe (temps de réponse)

L'ordonnancement des processus placés sur un même processeur est directement déduit de l'ordre dans lequel ils ont été placés. Le placement d'un processus sur un processeur entraîne l'ordonnancement de ses communications inter-processeurs.

L'utilisateur a la possibilité de contraindre tout processus à être placé sur un processeur particulier. Ceci permet de répondre à des nécessités matérielles (entrée-sortie) ou bien de guider itérativement l'heuristique afin d'obtenir éventuellement de meilleurs résultats.

1.4 Génération des exécutifs

Un exécutif distribué temps-réel offre des services tels que la gestion des ressources de calcul, de communication et de mémoire et la gestion du temps.

La gestion des ressources de calcul et de communication est faite statiquement par un placement et un ordonnancement des processus et des communications inter-processeur, précalculé à l'aide d'une étude des graphes logiciel et matériel (cf. §1.3). SynDEx utilise un mécanisme de routage statique qui permet d'établir des communications entre processeurs non directement connectés.

La gestion de la mémoire est assurée par SynDEx tant au niveau de l'application (retards de SIGNAL) qu'au niveau de l'exécutif (tampons de communications intra- et inter-processeur).

La gestion du temps est prise en compte à travers le cadre synchrone. Du point de vue de l'exécutif, cela consiste à maintenir les propriétés temporelles de l'algorithme spécifié avec SIGNAL lorsqu'il est implanté sur une machine multi-processeur. Actuellement, SynDEx se limite aux applications mono-horloge, c'est-à-dire dont le graphe flot-de-données n'est pas conditionné (pas de sous-échantillonnage ni de mélange de signaux), dans ce cas tous les signaux ont la même horloge et le graphe est le même pour chaque instant logique de SIGNAL.

SynDEx génère l'exécutif, c'est-à-dire le code système qui doit être compilé avec le code application fourni par l'utilisateur.

Pour chaque processeur, le code système est construit à partir d'un noyau d'exécutif modulaire indépendant de la machine dont les objets élémentaires sont :

- **PE** et **PR** : pour chaque communication inter-processeur, l'objet PE (Porte d'Emission) associé au processus émetteur construit un message contenant les données à transmettre et des informations de routage. Les autres objets de l'exécutif utilisent ces informations pour faire parvenir le message à l'objet PR (Porte de Réception) associé au processus récepteur. Afin d'assurer la synchronisation entre les processus émetteur et récepteur,

la PR renvoie à la PE un message d'acquittement après réception du message de données.

- **PES** : à chaque port d'entrée-sortie d'un processeur est associé un objet PES (Porte d'Entrée-Sortie). Il effectue les transferts de messages entre le processeur auquel il appartient et les autres processeurs. Dans les deux directions, les messages sont tamponnés en mode FIFO, permettant la séquentialisation de plusieurs messages concurrents. La taille de ces tampons, qui forment la mémoire système, est calculée de manière à éviter les interblocages de communication qui pourraient provenir du partage de ces tampons par plusieurs messages concurrents. Le protocole de transfert des messages dépend du type du port d'entrée-sortie. SynDEx supporte actuellement trois types de ports, pour des liaisons biprivées (liens point-à-point), des liaisons partagées (bus) et des liaisons avec des serveurs d'entrée-sorties (gestionnaires de fichiers, capteurs ...).
- **RT** : sur chaque processeur, toutes les portes (PE, PR et PES) sont connectées à un objet RT (routeur) qui oriente les message entre les portes (d'une PE vers une PES ou d'une PES vers une autre PES ou d'une PES vers une PR), grâce à une table de routage et aux informations de routage contenues dans les messages.
- **ARB** : dans chaque processeur, un objet ARB (arbitre) décide de l'ordonnancement (cf. §1.3) des calculs et des communications inter-processeurs.

Actuellement, SynDEx vise des machines cibles multi-transputers. Les processus de calcul sont écrits en OCCAM par l'utilisateur. SynDEx génère le code de l'exécutif en OCCAM à partir d'une bibliothèque système contenant les objets définis ci-dessus. Ces codes OCCAM sont compilés, chargés et exécutés dans l'environnement Transputer Development Toolset [8].

2 EXEMPLE D'APPLICATION

2.1 Spécification de l'application

L'application traitée consiste à identifier un filtre transversal avec un égaliseur adaptatif transversal. Pour cela, un processus génère un signal pseudo-aléatoire qui est envoyé d'une part sur le filtre à identifier et d'autre part sur le filtre adaptatif, on calcule l'erreur en faisant la différence entre la sortie estimée par le filtre adaptatif et le signal sortant du filtre à identifier. Cette erreur sert à calculer à l'aide d'un algorithme de gradient stochastique les coefficients du filtre adaptatif. On visualise en temps réel l'erreur afin de vérifier la convergence.

On décrit brièvement les calculs à effectuer afin de détailler cet algorithme dont le programme SIGNAL est donné fig.1.

`gensig` est un processus qui génère un signal pseudo-aléatoire par multiplication de nombres premiers entre eux combinés à la valeur trouvée précédemment.

`wind` et `winda` sont des fenêtres glissantes, pour chaque échantillon d'entrée elles produisent un vecteur dont les composantes sont décalé de une à k unités dans le temps logique :

$$wx_n = (x_n, x_{n-1}, \cdots, x_{n-k})$$

```
process EGALISEUR = (integer K ; real P) {}
(| GENSIG ?ZI !ZO, SIG
 | ZI := ZO $ 1
 | WIND ?SIG !OW
 | WINDA ?SIG !OWA
 | FILT ?OW !OFILT
 | FILTA(K) ?OWA, ZCOEF !OFILTA
 | SUB ?OFILT, OFILTA !ER
 | GAIN(P) ?ER !ERP
 | ADAP ?OWA, ZCOEF, ERP !COEF
 | ZCOEF := COEF $ 1
 | VISU ?ER
 |)
where
 ...
end
```

Figure 1: programme SIGNAL de l'égaliseur adaptatif

`filt` et `filta` sont des filtres transversaux dont les k coefficients sont respectivement fixes et variables (coefficients fournis par le processus d'adaptation) :

$$y_n = \sum_{i=1}^{k} h_{n,i} x_{n-i}$$

On a choisi de ne pas se limiter à une seule fenêtre diffusée sur les deux filtres fixe et adaptatif, afin de permettre des longueurs k différentes pour ces filtres.

`sub` est le processus qui calcule l'erreur par différence entre le signal après filtrage adaptatif et le signal après filtrage par le filtre à identifier :

$$e = ya - y$$

`gain` est le processus qui calcule l'erreur pondérée erp, p étant une constante de gain de valeur inférieure à 1 :

$$erp = e.p$$

`adap` est le processus d'adaptation des coefficients selon un algorithme de gradient stochastique :

$$h_{n,i} = h_{n-1,i} + erp.x_{n-i} \qquad \forall i \in [1, k]$$

2.2 Utilisation de l'environnement SynDEx

L'utilisateur manipule un environnement graphique (souris et menus "pop-up" contextuels) pour décrire les graphes logiciel et matériel, activer l'algorithme de placement-ordonnancement et la génération du code OCCAM pour une machine multi-transputer.

La figure 2 présente le "browser SynDEx" tel qu'il apparaît à l'écran, divisé en cinq vues : une vue d'édition topologique des graphes et sa vue zoom à gauche, une vue temporelle et sa vue zoom à droite et une vue texte en haut. La vue texte permet la saisie et l'affichage d'informations qui n'apparaissent pas sous forme graphique.

Le programme SIGNAL de l'égaliseur adaptatif (§2.1) a servi à spécifier l'algorithme et sa compilation a permis de vérifier qu'il est temporellement correct. Cette spécification,

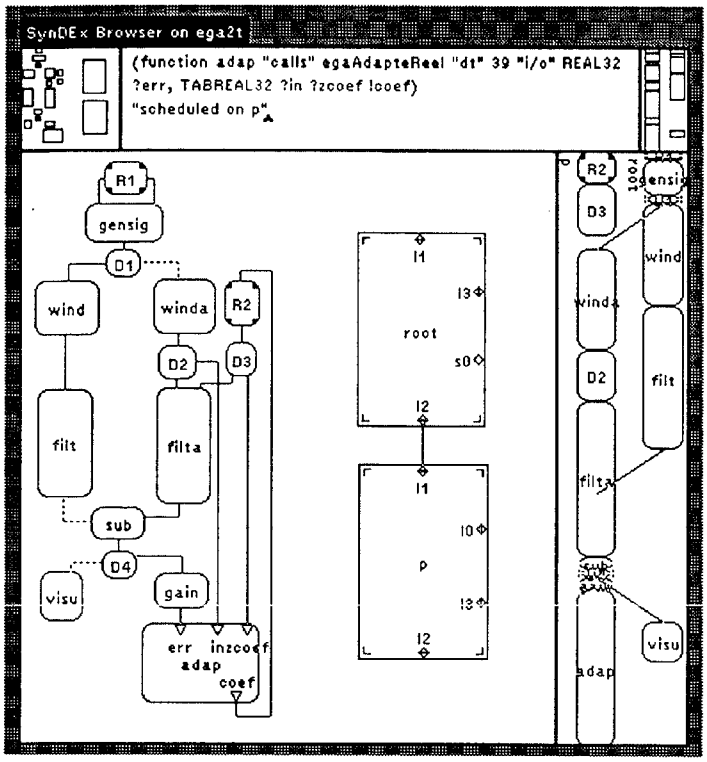

Figure 2: SynDEx Browser sur l'égaliseur adaptatif

reprise à l'aide de l'environnement SynDEx, correspond au graphe logiciel orienté qui apparaît sur la gauche de la fig.2.

Les processus diffusion D_i servent à matérialiser la diffusion qui est native en SIGNAL. Ils effectuent simplement une série d'affectations de leur entrée sur toutes leurs sorties.

Les processus R_i sont les "$" de SIGNAL, retards de une unité. Ils représentent la ressource mémoire application allouée aux processeurs. Ils s'exécutent différemment des autres processus, ils produisent tout d'abord la sortie puis consomment l'entrée.

Les autres processus sont des fonctions immédiates de SIGNAL dont l'exécution suit la règle d'activation flot-de-données qui correspond à la boucle suivante :

1) Lectures de toutes les entrées
2) Appel d'un processus de calcul
3) Ecritures de toutes les sorties

La machine cible, construite dans cet exemple à l'aide de deux transputers, correspond au graphe matériel non orienté qui apparaît au centre de la figure 2. Les nœuds du graphe matériel représentent des processeurs. Les arcs représentent des liaisons de communications bidirectionnelles associées à des ports d'entrée-sortie.

Le diagramme temporel qui apparaît après placement-ordonnancement sur la droite de la figure 2, représente la simulation de l'exécution en temps-réel de l'application sur la machine cible. Le temps se déroule de haut en bas avec une colonne par processeur. Chaque processus est représenté par une boîte dont la hauteur est proportionnelle à la durée d'exécution du processus (§1.3). Chaque communication inter-processeur est représentée par un segment de droite dont l'origine se situe dans la colonne du processeur émetteur à la date de début de communication et dont l'extrémité se situe dans la colonne du processeur récepteur à la date de fin de communication. Entre cette date et la date de début du processus récepteur, la donnée est mémorisée dans un tampon. Comme on le constate sur la figure, le modèle choisi autorise la concurrence des calculs et des communications.

Lorsque la machine ainsi progammée fonctionne en temps réel, le signal d'erreur est visualisé sur un moniteur graphique. La figure 3 montre la convergence de l'algorithme de gradient pour 200 itérations. On observe une erreur résiduelle très faible après une centaine d'itérations.

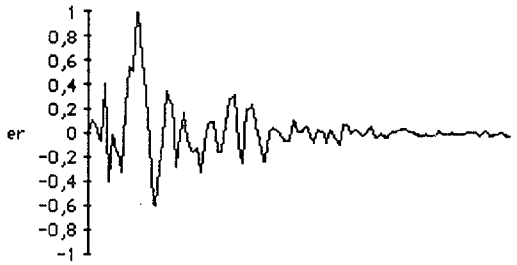

Figure 3: Erreur résiduelle pour 200 itérations

References

[1] A. Benveniste, B. Le Goff, P. Le Guernic : *Hybrid dynamical systems theory and the language SIGNAL.* Rapports de Recherche INRIA n°838 (1988)

[2] A. Benveniste, P. Le Guernic, Y. Sorel, M. Sorine : *A denotational theory of synchronous reactive systems.* Information and Computation (1990)

[3] 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 july 91.

[4] N. Ghezal, S. Matiatos, P. Piovesan, Y. Sorel, M. Sorine : *SYNDEX, un environnement de programmation pour multi-processeur de traitement du signal – Mécanismes de communication.* Rapports de Recherche INRIA n°1236 (1990)

[5] C. Lavarenne, Y. Sorel : *SYNDEX, un environnement de programmation pour multi-processeur de traitement du signal – Manuel de l'utilisateur version v.0.* Rapports Techniques INRIA n°113 (1989)

[6] N. Ghezal : *Quelques méthodes de répartition et d'ordonnancement de processus de traitement du signal sur un réseau de transputers.* Thèse de doctorat, Université de Paris-Sud Orsay (1989)

[7] Z. Liu, J. Labetoulle : *A heuristic method for loading and scheduling flexible manufaturing systems.* Proc. of the Int. Conf. Control 88, London, IEE Conference Publication n°285 pp195–200, (1988)

[8] *OCCAM 2 Toolset User Manual.* INMOS (1989)