Methode chronometrique pour l'optimisation du temps de reponse des executifs syndex

F. Ennesser, Christophe Lavarenne, Yves Sorel

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992

25ème anniversaire

N° 1769

Programme 5
Traitement du Signal,
Automatique et Productique

MÉTHODE CHRONOMÉTRIQUE POUR L'OPTIMISATION DU TEMPS DE RÉPONSE DES EXÉCUTIFS SynDEx

François ENNESSER
Christophe LAVARENNE
Yves SOREL

Octobre 1992

RR-1769

Méthode chronométrique pour l'optimisation du temps de réponse des exécutifs SynDEx

François ENNESSER
Christophe LAVARENNE
Yves SOREL

INRIA - Domaine de Voluceau - Rocquencourt B.P.105
78153 Le Chesnay Cedex - France
Tél: (1) 39 63 55 11 - email: Yves.Sorel@inria.fr

Résumé : La complexité sans cesse croissante et les contraintes temps-réel des applications de traitement du signal et de contrôle de processus conduisent à utiliser des architectures multi-processeurs.

SynDEx est un environnement graphique interactif de développement pour ce type d'application. Il fournit une heuristique paramétrable d'optimisation du temps de réponse permettant de trouver parmi les implantations réalisables, celles satisfaisant les contraintes temps-réel. A partir d'une implantation satisfaisante, il génère automatiquement un exécutif dédié, libérant ainsi l'utilisateur des tâches lourdes de programmation bas niveau.

Des modèles de graphes sont utilisés pour spécifier l'algorithme et le multi-processeur, ce qui permet à l'heuristique de calculer le temps de réponse à partir des durées élémentaires des composants des graphes.

On présente dans ce rapport une méthode chronométrique temps-réel permettant de mesurer ces durées élémentaires et le temps de réponse.

Chronometric method for the response time optimization of SynDEx executives

Abstract: The increasing complexity and the real-time constraints of signal processing and process control applications lead to multi-processor implementations.

SynDEx is an interactive graphical environment for such applications. It provides parameterizable heuristics which optimize the response time in order to find among the feasible implementations those which satisfy the real-time constraints. It then automatically generates, from a consistent implementation, a dedicated executive, thus relieving the programmer from heavy low-level programming.

Graph models are used to specify the algorithm and the multi-processor, so that the heuristics may be used to compute the response time from elementary durations of graph components.

This report presents a real-time chronometric method to measure these elementary durations and the response time.

Table des matières

1 Introduction 5

2 Environnement matériel et son exécutif 9
- 2.1 Transputer T800 . . . 9
- 2.2 Carte multi-Transputers : B008 . . . 9
- 2.3 Assembleur du Transputer : OCCAM . . . 9
- 2.4 Exécutif SynDEx pour le Transputer . . . 11

3 Méthode et résultats de chronométrage des liaisons physiques 13
- 3.1 Principes . . . 13
- 3.2 Résultats . . . 13
  - 3.2.1 Conditions de mesures . . . 13
  - 3.2.2 Durée des communications intra-processeurs . . . 14
  - 3.2.3 Durée des communications inter-processeurs sur un lien direct . . . 14
  - 3.2.4 Durée des communications inter-processeurs à travers le crossbar C004 . . . 15
  - 3.2.5 Parallélisme des communications sur un Transputer . . . 15
  - 3.2.6 Parallélisme des calculs et des communications sur un Transputer . . . 17
  - 3.2.7 Influence du placement des programmes en mémoire interne ou externe . . . 17

4 Méthode de chronométrage de l'exécutif 19
- 4.1 Principes . . . 19
  - 4.1.1 Transparence . . . 19
  - 4.1.2 Etablissement d'un temps global pour tous les Transputers du réseau . . . 20
  - 4.1.3 Collecte des mesures . . . 20
  - 4.1.4 Formes d'implantation étudiées . . . 20
- 4.2 Mise en œuvre du système de chronométrage . . . 21
  - 4.2.1 Forme d'implantation choisie . . . 21
  - 4.2.2 Protocole d'établissement de l'horloge globale . . . 21
  - 4.2.3 Protocole de collecte des mesures . . . 22

5 Résultats de chronométrage d'un réseau de Transputers 25
- 5.1 Conditions de mesures . . . 25
- 5.2 Durée des communications intra-processeurs . . . 25
- 5.3 Durée des communications inter-processeurs point-à-point . . . 25
- 5.4 Durée des communications point-à-point à travers le crossbar C004 . . . 26
- 5.5 Parallélisme des communications sur un Transputer . . . 26
- 5.6 Parallélisme des calculs et des communications sur un Transputer . . . 28

6 Exemple de chronométrage d'un égaliseur adaptatif sur un bi-Transputer 29
- 6.1 Présentation . . . 29
- 6.2 Résultats . . . 29

7 Conclusions et perspectives 33
- 7.1 Ordonnancement des communications . . . 33
- 7.2 Interaction entre calculs et communications . . . 33
- 7.3 Prévisions de performances . . . 33
- 7.4 Placement ordonnancement . . . 34

7.5 Génération de code 34
7.6 Portage multi-processeur 34

Annexes **34**

1 Code généré pour l'exemple égaliseur adaptatif **35**
1.1 Fichier de configuration `ega2t.pgm` 35
1.2 Fichier include `ega2t.inc` 36
1.3 Fichier `ega2t1.occ` pour le processeur `root` 37
1.4 Fichier `ega2t2.occ` pour processeur `p` 39

Chapitre 1

Introduction

La complexité sans cesse croissante et les contraintes temps-réel des applications de traitement du signal et de contrôle de processus conduisent à utiliser des architectures multi-processeurs.

SynDEx (acronyme pour EXécutif Distribué SYNchrone) est un environnement graphique interactif de développement pour ce type d'application [1] [2]. Il fournit une heuristique paramétrable d'optimisation du temps de réponse permettant de trouver parmi les implantations réalisables, celles satisfaisant les contraintes temps-réel. A partir d'une implantation satisfaisante, il génère automatiquement un exécutif fiable, libérant ainsi l'utilisateur des tâches lourdes de programmation bas niveau.

SynDEx offre les services suivants :

- description graphique d'un algorithme d'application sous la forme d'un graphe flot de données
- description graphique d'un multi-processeur sous la forme d'un graphe
- placement et ordonnancement optimisant le temps de réponse de l'algorithme sur le multi-processeur
- calcul et visualisation d'un diagramme temps-réel d'exécution de l'algorithme sur le multi-processeur
- génération de l'exécutif SynDEx pour fonctionnement en temps-réel sur un multi-Transputer

L'algorithme d'application est modélisé par un hypergraphe orienté acyclique (*graphe logiciel*) dont chaque sommet représente un *processus de calcul* flot de données et chaque arc une *communication inter-processus*.

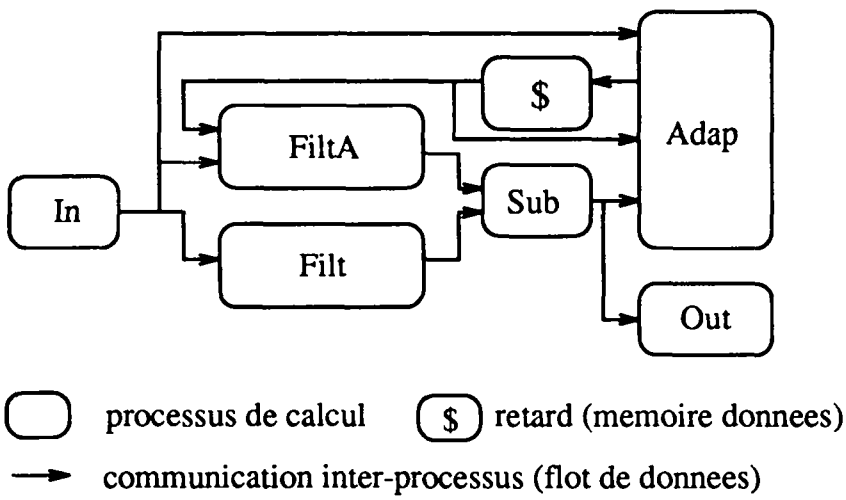

Figure 1.1: Un exemple d'algorithme

Le multi-processeur est modélisé par un hypergraphe (*graphe matériel*) dont chaque sommet représente un *nœud processeur* et chaque hyper-arc une liaison physique de communication connectant plusieurs nœuds processeurs (deux ou plus). Chaque nœud processeur est décrit à partir des éléments suivants :
- processeur de calcul
- processeur de communication (un par hyper-arc adjacent)
- processeur d'entrée sortie
- mémoire de données partagée entre éléments processeurs

Chaque élément processeur inclut sa mémoire de programme qui recevra les instructions de calcul et l'exécutif généré par SynDEx. La coopération entre processeurs de communication d'un même hyper-arc permet le transfert de données entre les mémoires données des nœuds processeurs connectés. Les processeurs d'entrée sortie permettent au multi-processeur programmé de communiquer avec son environnement, c'est-à-dire de réagir (sorties) aux stimuli (entrées) de l'environnement.

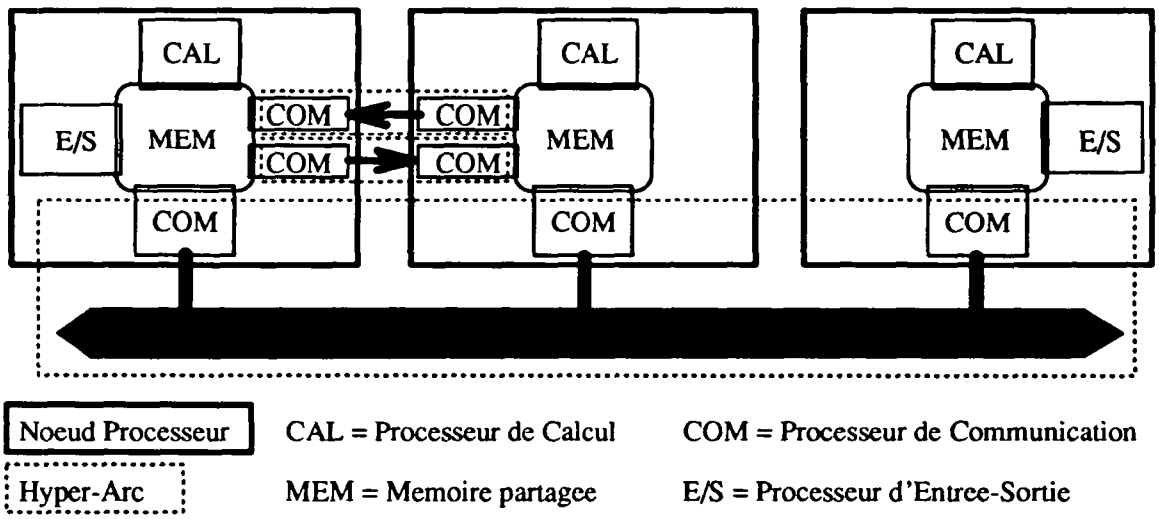

Figure 1.2: Un exemple d'architecture

Le placement ordonnancement est précalculé et statique (pas de migration ni d'ordonnancement de processus à l'exécution). Il consiste à placer (affecter) et ordonnancer (séquentialiser) les processus de calcul (resp. d'entrée sortie) sur les processeurs de calcul (resp. d'entrée sortie). Ce placement produit des communications inter-processeur qu'il faut également placer et ordonnancer sur les processeurs de communication. Pour cela on choisit un chemin (*route*) dans l'hypergraphe matériel, on crée entre les deux processus (de calcul ou d'entrée sortie) communiquant un *processus de communication* pour chaque hyper-arc de la route et on place chacun de ces processus de communication sur l'hyper-arc correspondant, c'est-à-dire sur chacun de ses processeurs de communication. Cette dernière opération consiste à diviser le processus de communication en sous-processus de communication coopérants, placé chacun sur un processeur de communication. Les sous-processus de communication issus de communications inter-processeur différentes sont ordonnancés sur chaque processeur de communication.

L'optimisation du placement ordonnancement actuellement effectuée consiste à minimiser le temps de réponse, c'est-à-dire la longueur du chemin critique du graphe logiciel valué par les durées d'exécution des processus de calcul et de communication. L'algorithme de placement ordonnancement actuellement implanté dans SynDEx est une heuristique (problème NP-complet) dont la complexité est fonction linéaire du produit du nombre de processus et de processeurs.

Le diagramme temporel calculé pendant l'optimisation permet une visualisation graphique (voir figure 6.1 page 30) faisant apparaître les communications inter-processeur et pour chaque processeur l'ordonnancement des processus qui y sont placés.

L'exécutif généré pour chaque processeur par SynDEx supporte les processus de calcul qui y ont été placés (tirés d'une bibliothèque fournie par l'utilisateur). Il est constitué par un assemblage d'éléments

du noyau d'exécutif (tirés d'une bibliothèque système) qui gèrent les communications inter-processus. Les communications inter-processus *intra*-processeur sont réalisées par l'intermédiaire de la mémoire. Les communications inter-processus *inter*-processeur sont réalisées par des processus de communication assurant le transfert des données entre processeurs par passage de messages. Ces processus réalisent les fonctions suivantes :
- formatage et déformatage des messages
- routage des messages à l'intérieur des processeurs
- transfert des messages à travers les liaisons physiques de communication

Ces fonctions réalisent les protocoles de communication des principaux niveaux de la norme ISO [3].

L'efficacité de l'exécution en temps réel de l'algorithme sur le multi-processeur dépend essentiellement de l'optimisation du placement ordonnancement basé comme on l'a dit précédemment sur le calcul de la durée du chemin critique du graphe valué par les durées d'exécution des processus de calcul et de communication [4]. Si le multi-processeur n'est pas disponible, ces durées doivent être estimées, sinon elles peuvent être mesurées en temps réel.

Pour chronométrer un processus de calcul, on l'exécute isolément sur un processeur et on calcule la différence entre ses dates de fin et de début d'exécution prises sur l'horloge temps-réel du processeur. La durée d'une communication inter-processus *intra*-processeur est négligeable face à celle d'une communication inter-processeur. Une communication inter-processus *inter*-processeur se décompose en transferts de messages à travers les liaisons physiques qu'elle traverse (routage). La durée de transfert à travers chaque liaison physique est la somme d'une éventuelle durée d'attente (de fin d'un autre transfert en cours) et de la durée de la transmission proprement dite du message. Pour mesurer précisément la durée du transfert d'un message à travers une liaison physique, il faut mesurer isolément à l'aide de l'horloge temps-réel du processeur la durée d'un aller-retour de ce message puis la diviser par deux.

A partir de ces données chronométriques élémentaires, on peut calculer le temps de réponse. Pour valider ce dernier, il faut pouvoir également le mesurer en temps réel.

Le chronométrage est réalisé en enregistrant les dates de début et de fin de tous les processus de calcul et de tous les processus de communication en utilisant comme chronomètres les horloges temps-réel des processeurs. Ceci permet d'une part d'enregister le diagramme temporel réel pour le comparer au diagramme temporel calculé par l'algorithme de placement ordonnancement et d'autre part de répartir équitablement les perturbations dûes aux chronométrages. Avant de lancer l'application, les horloges temps-réel locales des processeurs sont comparées de proche en proche afin d'installer l'horloge globale implicite du diagramme temporel. Lorsque l'application est terminée, les mesures sont collectées de proche en proche et stockées dans un fichier pour analyse ultérieure.

L'objet de ce rapport est de détailler ces méthodes chronométriques dans le cas de l'exécutif SynDEx pour une architecture multi-Transputer.

On décrit d'abord l'architecture et l'exécutif associé. Puis on donne une méthode de chronométrage des transferts de données à travers les liaisons physiques et on donne les résultats dans le cas des liens des Transputers. On décrit ensuite une méthode de chronométrage de l'exécutif SynDEx. On utilise ensuite cette méthode associée à la génération automatique de l'exécutif faite par SynDEx pour chronométrer l'exécutif SynDEx dans le cas des réseaux multi-Transputers. Enfin, cette dernière méthode est appliquée à un exemple classique en traitement du signal, un égaliseur adaptatif, s'exécutant sur deux Transputers.

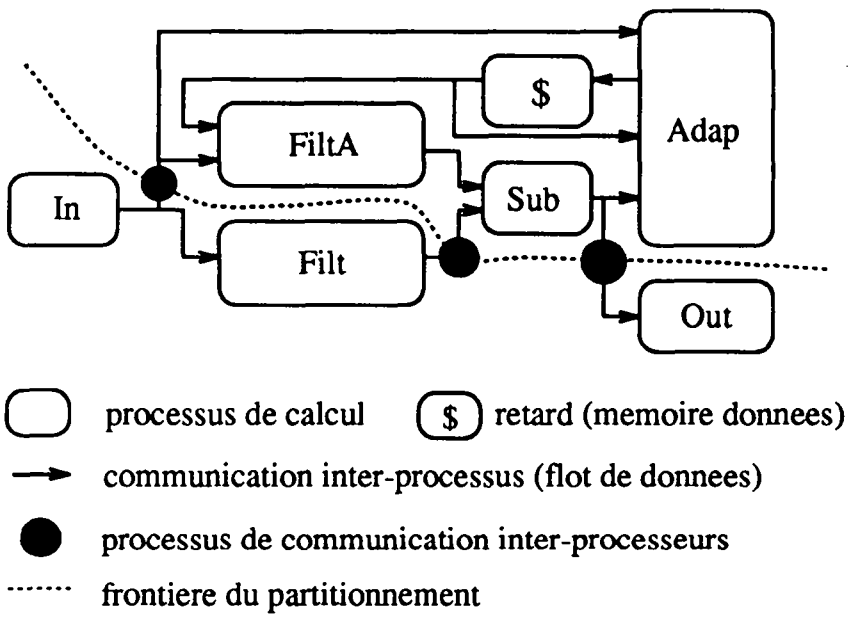

Figure 1.3: Graphe logiciel après placement

Chapitre 2

Environnement matériel et son exécutif

2.1 Transputer T800

Le composant de base des machines parallèles sur lesquelles nous avons fait des mesures est le Transputer [5]. C'est un processeur produit par INMOS, dont la particularité est d'être doté de 4 liens (liaisons séries bidirectionnelles), chacun constitué de 2 canaux monodirectionnels (un pour chaque sens) associés à un contrôleur de DMA. La communication sur ces canaux est règlementée par un protocole simple demandant la transmission de 3 bits supplémentaires par octet transmis plus 2 bits de récépissé pour assurer le contrôle de flux. Comparées aux liaisons multipoints par bus parallèle, les liaisons par liens présentent les avantages etles inconvénients suivants :
- un lien ne nécessite que deux fils, ce qui simplifie la connectique et rend possible la réalisation de brasseurs de liens (circuit C004) simplifiant la configuration topologique du réseau
- un lien ne pose pas de problème de charge capacitive ni de problème de partage de ressources
- le débit d'un lien est inférieur à celui d'un bus parallèle, mais le nombre de liens, donc le débit total, augmente avec le nombre de processeurs

Le modèle utilisé précisément est le T800 cadencé par une horloge à 20 MHz. C'est un processeur d'architecture 32 bits comportant une unité flottante 64 bits fonctionnant en parallèle avec un CPU capable d'effectuer des calculs sur des entiers. Il permet la programmation concurrente de processus (2 niveaux de priorité, Time Slicing toutes les 2 ms en basse priorité). Ses principales caractéristiques sont les suivantes :

- Temps de cycle 33 ns ; 30 MIPS
- 4 Koctets de RAM statique"on chip"à 120 Moctets/s
- 4 Goctets de memoire externe adressable, à 40 Moctets/s

Ses liaisons séries ont été programmées dans notre cas à 20 Mbits/s.

2.2 Carte multi-Transputers : B008

C'est un circuit imprimé sur lequel on peut insérer jusqu'à dix Transputers avec leur mémoire, et qui comporte un brasseur de liens (crossbar C004) configurable par un des Transputers. La carte se présente comme sur la figure 2.1. On remarque que chaque Transputer est relié par des liens directs à ses deux voisins, alors que ses deux autres liens sont connectés au C004. La programmation adéquate de ce dernier circuit permet donc de connecter les Transputers selon une architecture quelconque. Dans notre cas, la carte B008 était équipée de 5 Transputers *IMS T805 b* cadencés à 20 MHz, disposant chacun de 4 Moctets de RAM externe. La carte est supportée par un ordinateur hôte de type compatible IBM PC.

2.3 Assembleur du Transputer : OCCAM

OCCAM [6] est un langage de programmation pour lequel le Transputer a été spécialement adapté. Il permet de décrire un système comme un ensemble de processus concurrents communicant par des canaux

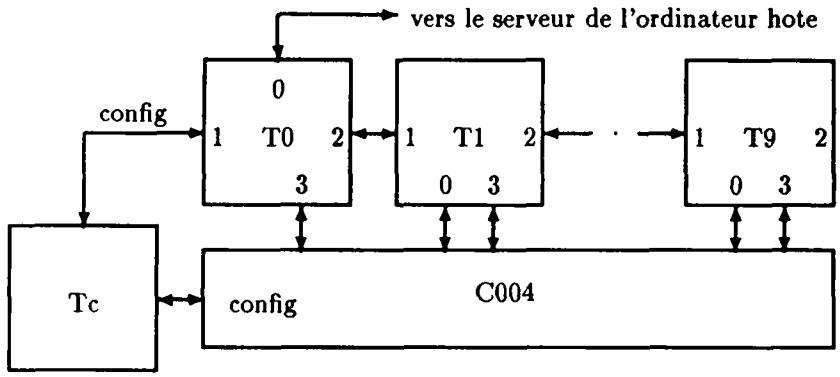

Figure 2.1: La carte B008

point-à-point typés. Les communications inter-processus s'effectuent par rendez-vous, c'est-à-dire lorsque les deux processus émetteur et récepteur sont prêts à communiquer. Il y a trois principaux processus élémentaires :

- une assignation : **variable := expression**
- une entrée : **canal ? variable**
- une sortie : **canal ! expression**

Un processus de sortie est toujours associé à un seul processus d'entrée (et réciproquement) par l'intermédiaire d'un canal dont le type doit être le même que celui de la **variable** et de l'**expression**. Au moment du rendez-vous, le résultat de l'expression est transféré dans la variable.

Un processus composé est constitué en groupant d'autres processus (élémentaires ou composés) à l'aide d'un des 4 constructeurs suivants :

- **SEQ** : exécution séquentielle des processus constituants (les processus sont exécutés en séquence, le **SEQ** se termine avec son dernier processus constituant)
- **PAR** : exécution parallèle des processus constituants (les processus sont exécutés en concurrence, le **PAR** se termine lorsque tous ses constituants sont terminés)
- **IF** : exécution alternative déterministe (les conditions précédant chaque processus sont évaluées en séquence; le premier processus dont la condition est vérifiée est exécuté)
- **ALT** : exécution alternative indéterministe (chaque processus est précédé d'un processus élémentaire d'entrée, optionnellement précédée d'une condition ; parmi les processus dont la condition est vérifiée et dont l'entrée est prête à être exécutée, un est choisi au hasard et son entrée et lui-même sont exécutés)

Tous les composants d'un processus composé doivent être indentés de deux espaces par rapport au constructeur. Les variables locales à un processus doivent être déclarées juste avant celui-ci avec la même indentation. Par exemple :

```
INT x, y, z:
SEQ
  x := 2
  CHAN OF INT c:
  PAR
    c ? y
    c ! x+1
  z := y-x
```

Un processus composé peut également être construit par réplication d'un seul processus (élémentaire ou composé) à l'aide d'un des quatres constructeurs précédent associé au mot clé **FOR** qui permet de déclarer une variable entière avec une valeur initiale et un nombre d'incréments. Par exemple :

```
SEQ i=5 FOR 10 -- ce double tiret marque un commentaire jusqu'en fin de ligne
  processus(i) -- processus est appele pour i variant de 5 a 14 inclus
```

OCCAM dispose aussi d'une répétition conditionnelle :

```
WHILE condition
  processus
```

Des processus concurrents peuvent de plus être placés à 2 niveaux de priorité à l'aide du constructeur **PRI PAR**, qui place son premier processus constituant en haute priorité et les suivants en basse priorité. Les processus de basse priorité s'exécutent en temps partagé (en tranches de temps de 2ms sur les Transputers) tant qu'aucun processus de haute priorité ne peut s'exécuter. Dès qu'un des processus de haute priorité peut s'exécuter, il interrompt le processus de basse priorité actif, et ne peut que se terminer ou s'interrompre sur une entrée ou une sortie.

A chaque niveau de priorité, les processus peuvent utiliser l'horloge du processeur par l'intermédiaire de variables "timer" grâce aux 2 instructions suivantes :

- Lecture horloge : `timer ? variable`
- Mise en attente : `timer ? AFTER expression`

Les timers déclarés en haute priorité ont une résolution de 1 μs, alors que ceux de basse priorité s'incrémentent toutes les 64 μs.

Enfin, dans le cas ou un programme doit s'exécuter sur un réseau multi-Transputers, les instructions **PLACED PAR** et **PLACE** permettent le placement respectif des processus sur les processeurs et des canaux sur les liens physiques. **PLACE** permet également le placement de données en mémoire interne ou externe.

2.4 Exécutif SynDEx pour le Transputer

L'exécutif SynDEx supporte les communications intra-processeur par l'intermédiaire de la mémoire et les communications inter-processeur par des processus de communication assemblés à partir d'un noyau d'exécutif assurant le transfert des données entre processeurs par passage de messages. Ce noyau est constitué des éléments suivants :

- **PE** (porte d'émission) et **PR** (porte de réception) assurent respectivement le formatage et le déformatage des messages
- **BL** (bus logiciel), un seul dans chaque nœud processeur, route les messages entre les portes (PE, PR, PPL)
- **PPL** (porte d'entrée sortie entre processeur et lien), une par processeur de communication, transfère les messages à travers les liaisons physiques (lien des Transputers)

Les processus de communication sont placés sur chaque processeur en haute priorité, tandis que les processus de calculs sont placés en basse priorité. L'utilisateur a la possibilité de générer sur chaque processeur du code séquentiel ou pseudo-parallèle (le premier étant plus avantageux puisque moins coûteux en temps : on évite des changements de contexte et des transferts de messages entre processus de cacul).

Le code généré a la forme suivante :

```
PRI PAR
  Partie_Communications -- haute priorite, BL et portes PE, PR et PPL
  Partie_Calculs -- basse priorite
:
```

où les processus de `Partie_Calculs` sont de la forme :

- en code parallèle :

```
PAR
  -- premier processus de calcul
  ... -- declarations des tampons de communication du processus
  SEQ i=0 FOR NbIterations
    PAR -- entrees
```

```
      canal_entree ? tampon_entree -- connecte a PR ou autre processus calcul
      ... -- autres entrees
    ... -- Calcul utilisant les tampons de communication
    PAR -- sorties
      canal_sortie ! tampon_sortie -- connecte a PE ou autre processus calcul
      ... -- autres sorties
  -- autres processus de calcul du PAR
  ...
```

- en séquentiel :

```
... -- declaration des tampons de communication de tous les processus
SEQ i=0 FOR NbIterations
  SEQ
    ...
    canal_entree ? tampon_entree -- connecte a PR
    ... -- calcul utilisant des tampons de communication
    ... -- les communications intra-processeur se font par tampons partages
    canal_sortie ! tampon_sortie -- connecte a PE
    ...
```

Chapitre 3

Méthode et résultats de chronométrage des liaisons physiques

3.1 Principes

Une communication inter-processeur se décompose en transferts de messages à travers les liaisons physiques qu'elle traverse (routage). La durée de transfert à travers chaque liaison physique est la somme d'une éventuelle durée d'attente (de fin d'un autre transfert en cours) et de la durée de la transmission proprement dite du message. On ne s'intéresse ici qu'à cette dernière.

Le schéma de base utilisé consiste à connecter **2 processus "ping-pong"** se renvoyant l'un l'autre un message un nombre défini de fois n. En chronométrant n aller-retours du message, on en déduit avec une bonne précision le temps passé pour transférer des données d'un processus à l'autre à travers la liaison physique. Ces mesures sont faites pour toutes les configurations de placement des processus ping-pong et toutes les configurations possibles d'utilisation des liaisons physiques.

On étudie, à l'aide des processus ping-pong, la durée des communications en **fonction du nombre d'octets**. On caractérise une liaison physique par son *débit* (pente de la droite représentant les durées en fonction du nombre d'octets) et par son *start-up* (offset de cette droite). Le start-up correspond au coût des activités d'établissement de la communication qui ne se répètent pas pour tout octet transmis. Une technique de **régression linéaire** est employée pour obtenir le débit et le start-up.

Dans le cas du Transputer, on veut vérifier que les unités physiquement distinctes de calcul (CPU et FPU) et de communication (un DMA pour chacun des 4 liens) fonctionnent effectivement en parallèle. Pour cela, on définit un **processus "mange-temps"** qui s'exécutera sur les unités de calcul pendant un temps connu réglable par l'utilisateur, en parallèle avec des processus de communication.

3.2 Résultats

3.2.1 Conditions de mesures

Les mesures ont été effectuées en se plaçant dans le cas le plus général, c'est-à-dire pour des processus placés systématiquement en mémoire externe (nous avons forcé un tel placement en remplissant la mémoire interne par un tableau tampon inutilisé). Au dernier paragraphe, nous analysons les variations obtenues dans le cas où ces programmes se trouvent en mémoire interne.

Lorsqu'on moyenne les mesures sur un grand nombre de communications –ce qui se fait en prenant les mesures autour d'une boucle de répétition– il convient de retrancher au start-up obtenu le coût parasite d'une boucle vide, évalué à 0,65 μs/boucle en mémoire interne, et à 0,8 μs/boucle en mémoire externe. Ces valeurs restent faibles par rapport aux start-ups chronométrés. D'autre part, si d'autres instructions que celles de communications interviennent dans le code chronométré, il convient également d'en soustraire le coût au start-up. Le coût des instructions utilisées le plus fréquemment par la suite sont extrait de [5].

Dans tous les cas, nous n'avons pas considéré, pour effectuer le calcul des débits de communications, les messages contenant peu d'octets : en effet les courbes ne sont linéaires qu'au delà d'un nombre suffisant d'octets, selon l'allure donnée figure 3.1.

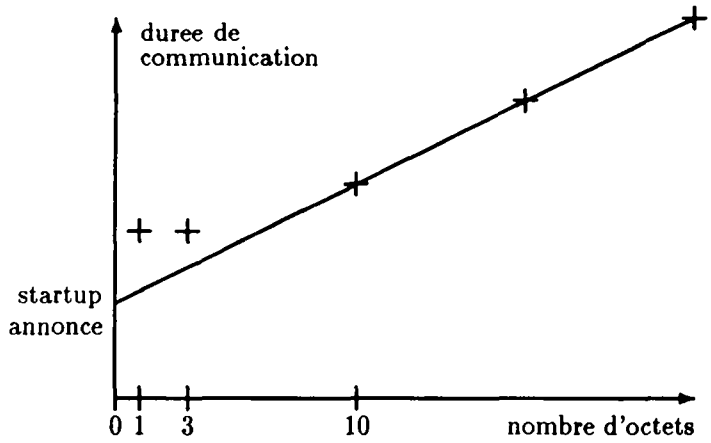

Figure 3.1: Durée relative de transmission des petits messages

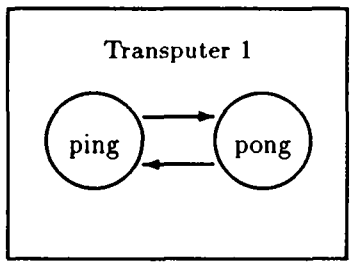

Figure 3.2: Communication intra-processeur

Les temps de start-up indiqués par le programme correspondent à l'offset théorique que l'on obtiendrait par prolongation de la partie vraiment linéaire de la courbe (calculée à partir du temps de transmission de messages suffisamment longs). Ils sont inférieurs aux start-ups que l'on obtiendrait en extrapolant la courbe réelle, ces derniers correspondant à peu près au temps mis pour transmettre les messages de moins de 5 octets. Il faut donc être délicat dans leur interprétation.

3.2.2 Durée des communications intra-processeurs

Il s'agit ici de communications inter-processus qui s'effectuent par l'intermédiaire de la mémoire. La méthode de mesure employée dans ce cas consiste simplement à utiliser les principes de mesure décrits ci-dessus en plaçant les processus *ping* et *pong* en parallèle sur un même processeur, ce qui donne le schéma de la figure 3.2.

En effectuant une régression linéaire sur des nombres d'octets transmis significatifs (jusqu'à 1 Moctet), on obtient avec une erreur très faible un débit de 13,15 Moctets/s et un start-up de 10 μs lorsqu'on exclut de la mesure le coût du constructeur PAR (qui est de 50 cycles processeurs, soit 1,65 μs). Le débit obtenu est conforme aux résultats publiés dans [7].

3.2.3 Durée des communications inter-processeurs sur un lien direct

Le schéma effectué est cette fois celui de la figure 3.3. Le start-up tombe aux alentours de 6 μs, et le débit mesuré est de l'ordre de 1,77 Moctets/s (avec une très faible erreur lorsqu'on considère des nombres d'octets transmis supérieurs à 10). Le débit théorique devrait être 1,817 Moctets/s (liens à 20 Mbits/s avec 11 bits transmis par octet, 3 de protocole plus 8 de données). La différence relative est assez faible (moins de 1%) : elle s'explique probablement par un temps de latence entre la transmission des différents octets. Quand à la diminution du temps de start-up, elle s'explique par le fait que le "rendez-vous" matériel établi ici se fait par l'intermédiaire des contrôleurs de DMA des liens des Transputers, ce qui n'était pas le cas du "rendez-vous" logiciel précédent. L'intérêt de ces contrôleurs de DMA apparaît donc

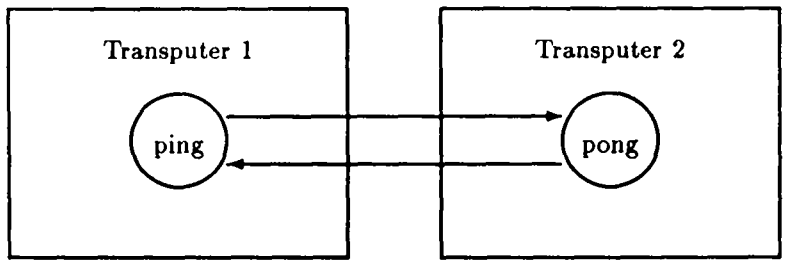

Figure 3.3: Communications inter-processeur par lien direct

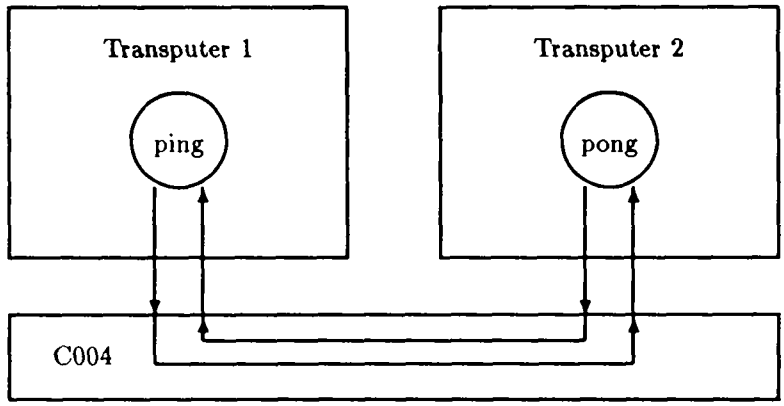

Figure 3.4: Communication inter-processeur par C004

concrètement ici.

3.2.4 Durée des communications inter-processeurs à travers le crossbar C004

Le schéma effectué est le même que précédemment, mais les liaisons passent par le crossbar C004 comme sur la figure 3.4

Le start-up change peu, mais le débit mesuré varie entre 1,33 et 1,40 Moctets/s (avec une faible erreur dans les conditions de mesures précédentes). Les quelques 5% de variations qu'on peut enregistrer d'une mesure sur l'autre s'expliquent par le comportement thermique du C004 : plus on l'utilise, moins il transmet vite.

Le débit à travers le C004 est donc de 30% inférieur au débit à travers un lien direct. Le constructeur annonce pour sa part un délai de 1,75 bit par octet, ce qui correspond à une variation un peu inférieure, de l'ordre de 20%.

3.2.5 Parallélisme des communications sur un Transputer

Par un placement convenable des processus *ping-pong*, nous avons effectué en parallèle des entrées-sorties sur les différents liens d'un Transputer : la carte B008 nous permettait d'effectuer les schémas de la figure 3.5 qui peuvent faire apparaître des interactions supplémentaires et même des schémas plus condensés comme sur la figure 3.6.

Dans chaque cas, nous avons mesuré globalement et individuellement les temps de transfert : que les communications passent ou non par le C004, les résultats obtenus coïncident presque exactement avec les précédents, si l'on retranche au temps de start-up le coût du PAR (2,3 μs pour 3 processus, d'après la formule du constructeur). Après ces corrections, l'augmentation restante du start-up est négligeable. La durée d'une communication est donc indépendante du nombre de communications concurrentes sur un même Transputer.

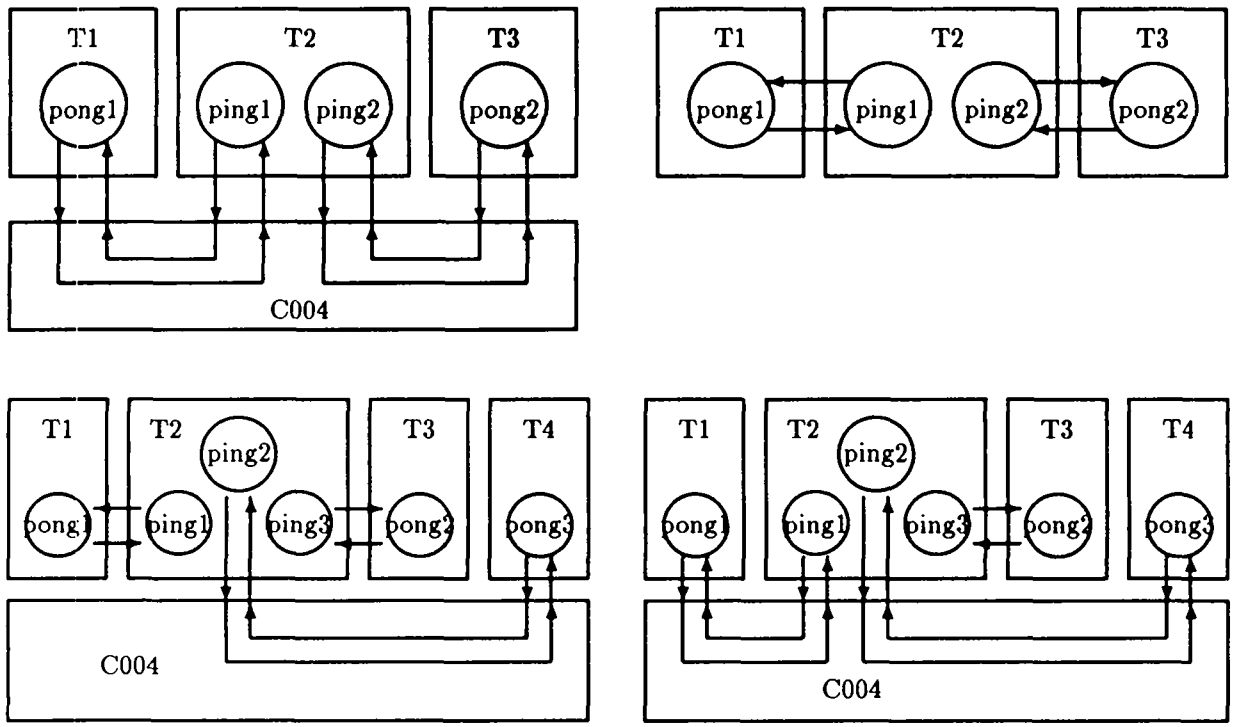

Figure 3.5: Communications concurrentes

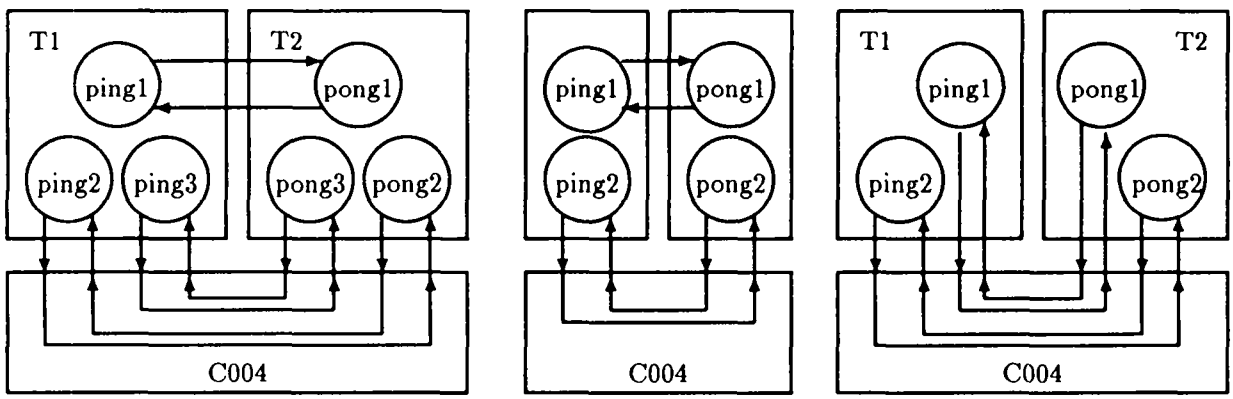

Figure 3.6: Communications concurrentes

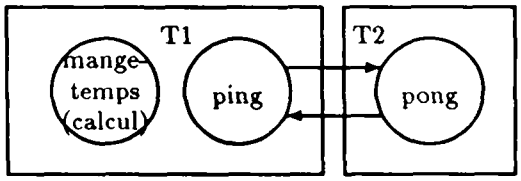

Figure 3.7: Communications et calculs concurrents

3.2.6 Parallélisme des calculs et des communications sur un Transputer

Pour cela, nous avons construit un *processus mange-temps* utilisant le FPU et le CPU du Transputer. Nous l'avons placé en séquentiel puis en parallèle avec un processus *ping-pong*, selon le schéma de la figure 3.7.

Nous avons constaté que le placement en parallèle n'affectait pas la durée de ces processus, ce qui montre que les vols de cycles des DMA des processeurs de communications n'affectent pas les durées de calcul.

3.2.7 Influence du placement des programmes en mémoire interne ou externe

En effectuant les mesures précédentes pour des programmes placés en mémoire interne, on constate que les débits et les durées de start-up des communications ne sont pas altérés. Par contre les durées des communications intra-processeur sont diminuées.

Chapitre 4

Méthode de chronométrage de l'exécutif

4.1 Principes

Le système de chronométrage à mettre en place pour les exécutifs SynDEx doit avoir les caractéristiques suivantes :

- Il doit permettre des comparaisons faciles avec les diagrammes temporels calculés par SynDEx. Pour cela, on produira à partir des mesures effectuées sur l' application lors de l'exécution temps-réel, une vue temporelle similaire à celle calculée par SynDEx.
- Comme tout bon système de mesure, celui-ci doit être le plus transparent possible, afin de perturber au minimum et de la maière la plus homogène possible le déroulement de l'application.
- Les deux points ci-dessus nous conduisent à opter pour des mesures systématiques de tous les processus de calcul et de communication.
- Enfin, on utilisera un code de mesure unique et simple, afin de permettre une génération automatique.

4.1.1 Transparence

Pour minimiser la perturbation dûe à l'introduction de mécanismes de mesures dans le code, il faut que les instructions de chronométrage aient un coût très faible. Deux techniques ont été étudiées :

- La plus simple consiste à introduire directement les instructions de mesure dans le code sous la forme :

  ```
  ...
  Timer[local] ? mesures[numero_mesure]
  ...
  ```

  La mesure se réduit alors à une seule instruction, on ne peut pas trouver moins coûteux (moins de 1μs). Par contre cette méthode présente des difficultés de mise en œuvre car les processus de calcul sont en basse priorité et les processus de communication en haute priorité. Or les instructions **timer** ont non seulement une résolution différente en haute et basse priorité (1 μs et 64 μs respectivement) mais on a pu constater qu'il n'existe aucune relation entre l'instant d'incrémentation du timer basse priorité et la valeur du timer haute priorité, sur un même processeur, ce qui rend difficile les comparaison des mesures effectuées. Par ailleurs, la précision des mesures risque d'être insuffisante en basse priorité.

- L'autre méthode permet d'effectuer toutes les mesures, y compris celles de l'application, en haute priorité donc avec une résolution de 1 μs, en envoyant une synchronisation — par exemple un entier permettant par la suite d'identifier la mesure — depuis l'instruction à dater jusqu'à un processus haute priorité chargé d'effectuer toutes les mesures, selon le principe suivant :

- APPLICATION (basse ou haute priorité) :

```
...
canalSynchro[local] ! numero_mesure
...
```

- PROCESSUS DE MESURES (haute priorité) :

```
PAR count=0 FOR nbVoieSynchro
  INT numero_mesure:
  WHILE TRUE
    SEQ
      canalSynchro[count] ? numero_mesure
      Timer_local ? mesures[numero_mesure]
```

Cette méthode est un peu plus coûteuse, elle dure 6 à 7 μs en mémoire externe, et 5 μs en mémoire interne.

4.1.2 Etablissement d'un temps global pour tous les Transputers du réseau

Pour obtenir un diagramme temporel équivalent à celui calculé par SynDEx, où les évènements sur tous les Transputers sont représentés sur une même échelle de temps, il est nécessaire d'établir une horloge globale à partir des horloges temps-réel locales à chaque Transputer. En effet ces dernières ne sont pas a priori initialisées au même instant. Ceci nécessite de choisir par rapport à l'une des horloges un instant de référence et de propager dans le réseau le délai qui s'est écoulé depuis cet instant de référence. Cette propagation ne pourra se faire que grâce à un protocole permettant de connaître le décalage des horloges internes des processeurs les unes par rapport aux autres. Ce protocole demande au minimum l'aller-retour d'un message de synchronisation, plus un envoi d'information. La date de départ de l'application doit être calculée avec une marge suffisante pour ne pas être antérieure à la fin de la propagation des messages de synchronisation.

4.1.3 Collecte des mesures

Une fois les mesures recueillies au niveau de chaque processeur, il convient de les propager à travers le réseau afin de les collecter pour permettre un traitement ultérieur. Cette propagation peut se faire selon un ordre préétabli, ou bien les mesures doivent être accompagnées d'informations permettant d'identifier leur provenance. D'autre part, l'envoi des mesures ne doit pas perturber l'application. Comme les processus placés sur les processeur ne se terminent probablement pas tous au même moment, plutôt que d'envoyer des signaux de terminaison à travers le réseau au risque d'interagir avec les dernières communications, on se fixe un délai de sûreté après la fin des calculs sur un processeur, délai calculé par l'heuristique d'optimisation du placement ordonnancement.

4.1.4 Formes d'implantation étudiées

Nous avons d'abord implanté des mesures directes en basse priorité selon le shéma suivant :

```
PRI PAR
  Partie_Communications -- haute priorit\'e
  -- basse priorit\'e
  [nombre_mesures]INT mesures:
  SEQ
    Init_mesures(mesures, routageInit,...)
    SEQ i=0 FOR NbIterations
      Calculs -- avec instructions: timer ? mesures[numero_mesure]
    Collect_mesures(mesures,routageFin,...)
```

Mais il est vite apparu que la résolution du timer basse priorité, de 64 μs, était insuffisante car la durée de beaucoup de processus entrant en jeu dans les applications testées sous Syndex (papillon FFT, filtre ...) est bien inférieure à cette résolution. Nous avons donc préféré la seconde méthode avec synchronisation entre processus de calcul (en basse priorité) ou de communication (en haute priorité) et processus de mesure en haute priorité. De plus dans ce cas, dans chaque processeur, le processus initial d'établissement de l'horloge globale, le processus de chronométrage et le processus final de collecte des mesures peuvent

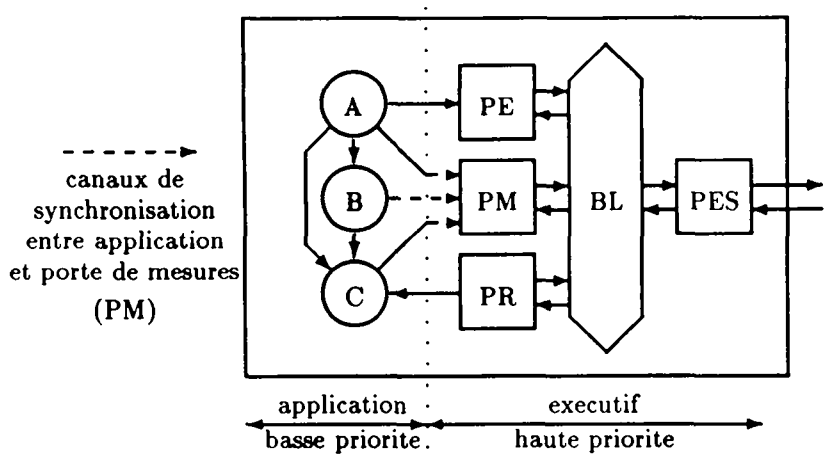

Figure 4.1: Porte de mesures

être regroupés sous la forme d'une *porte de mesure* semblable à celles déjà utilisées pour la génération de l'exécutif SynDEx.

```
PRI PAR
  PAR -- haute priorit\'e
    Partie_Communication -- BL et portes PE, PR et PPL
    Porte_Mesures(canalSynchro, mesures, routage...)
  SEQ -- basse priorit\'e
    ... -- initialisations
    SEQ i=0 FOR NbIterations
      Partie_Calculs -- avec instructions: canalSynchro ! numero_mesure
```

4.2 Mise en œuvre du système de chronométrage

4.2.1 Forme d'implantation choisie

Pour la phase initiale d'établissement de l'horloge globale ainsi que pour la phase finale de collecte des mesures, les portes de mesures doivent communiquer entre elles. Pour cela, on extrait du graphe des processeurs un arbre. Ainsi, chaque processeur, c'est-à-dire chaque porte de mesure, peut avoir un nombre de fils quelconque (nul ou non) et a un père unique. Dans le processeur racine de l'arbre, la porte de mesure a pour père un processus placé sur le même processeur, responsable du stockage sur disque des mesures collectées.

Le schéma de la figure 4.1 représente une porte de mesures connectée au bus logiciel du processeur.

4.2.2 Protocole d'établissement de l'horloge globale

La transmission à tous les processeurs de la date à laquelle ils doivent lancer leur application se fait de proche en proche, afin que tout nœud se comporte de la même façon vis-à-vis de ses fils que son père se comporte vis-à-vis de lui-même. La seule exception est celle du nœud racine, pour qui la date à laquelle il doit lancer l'application est déduite d'un délai suffisant pour que la propagation de l'horloge globale ait eu le temps de se terminer sur tous les processeurs. Tous les processeurs peuvent ensuite lancer leurs processus de calcul à la même date de l'horloge globale.

Le protocole de synchronisation est le suivant : chaque père, une fois qu'il a reçu son propre délai de départ, envoi à son fils, à une date qu'il note, un message de synchronisation qui lui est renvoyé immédiatement. Le fils prend soin au passage de noter sa date de réception à son horloge locale. Le père note également la date de retour du message. Comme la longueur du message transmis à travers la liaison physique a été la même dans les deux sens et comme il n'y a pas d'autre activité ni sur la liaison physique ni sur les processeurs père et fils, les durées d'aller et de retour du message sont égales. Le père peut donc calculer le délai qu'il doit communiquer à son fils simplement en soustrayant à sa propre date de départ

la date à laquelle, pour sa propre horloge, son fils a reçu la synchro, c'est-à-dire la moyenne entre la date d'envoi et la date de retour du message. Le père envoie donc ce délai à son fils, qui l'ajoute à sa date de réception du message de synchronisation pour connaître sa date locale de départ. Chaque père procède ainsi séquentiellement avec tous ses fils.

Sur une échelle temporelle, ce protocole a l'allure de la figure 4.2.

4.2.3 Protocole de collecte des mesures

Comme pour le protocole d'établissement de l'horloge globale, chaque processeur se comporte avec ses fils comme son père le fait vis-à-vis de lui-même. Par contre, alors que le protocole d'établissement de l'horloge globale permet une exécution parallèle, le protocole de collecte des mesures, utilisant une ressource unique, le disque, doit être séquentiel.

Sur le processeur racine, le processus de stockage sur disque, lancé après la fin du dernier processus de calcul, attend tout d'abord un délai suffisant pour laisser le temps à tous les processeurs de terminer leurs derniers calculs, puis il envoie un message de crédit à son fils (la porte de mesures du processeur racine de l'arbre).

Chaque porte de mesures se met en attente d'un crédit venant de son père, signifiant qu'elle peut envoyer un résultat de mesure (résultat calculé par rapport à la date de départ de l'application, auquel est joint le numéro de mesure). Chaque fois que le processus de stockage sur le processeur racine reçoit un message, il le stocke sur disque et renvoie un nouveau message de crédit. Lorsqu'une porte de mesure n'a plus de données locales à transmettre, elle transmet systématiquement les crédits qu'elle reçoit de son père à l'un de ses fils et retransmet à son père les données qu'elle reçoit de ce fils.

Lorsqu'une porte de mesure n'a plus de données à transmettre, elle répond au prochain crédit par un message vide. La porte de mesure du processeur père comprend ce signal comme signifiant la fin de la transmission, et passe à la porte de mesure du processeur fils suivant s'il en a d'autres. Sinon, il renvoie lui-même un message vide à la porte de mesure du processeur père. C'est ainsi que, quand toutes les portes de mesures ont transmis leurs données, le protocole se termine.

L'ordre de parcours d'un arbre est représenté schématiquement sur la figure 4.2.

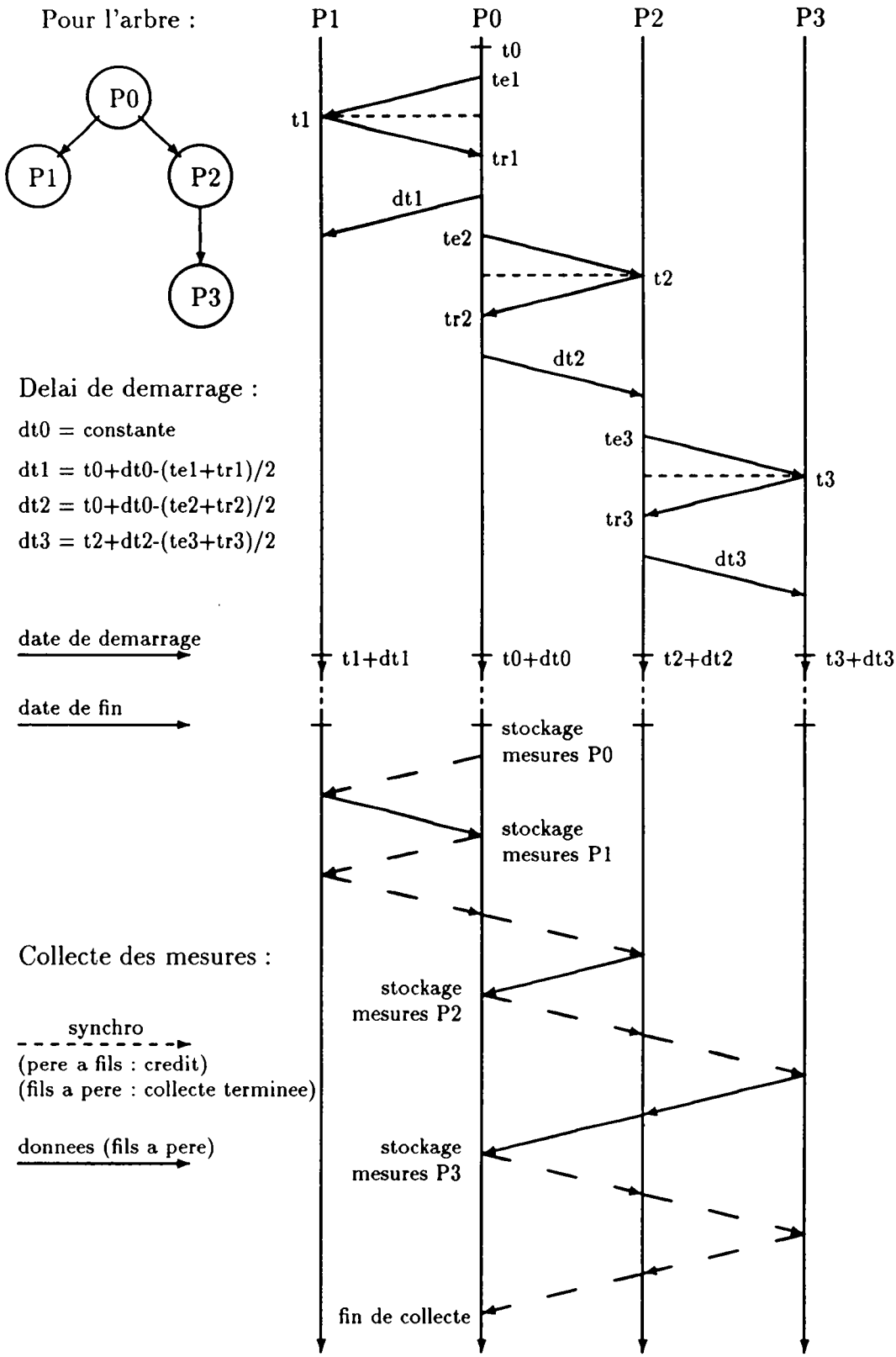

Figure 4.2: Protocoles d'établissement de l'horloge globale et de collecte des mesures

Chapitre 5

Résultats de chronométrage d'un réseau de Transputers

Dans le chapitre 3.1, nous avons chronométré les communications à travers les liaisons physiques, correspondant au plus bas niveau de la norme ISO.

Dans ce chapitre, nous reprenons la même campagne de mesures pour les deux niveaux suivants de la norme ISO, c'est-à-dire pour les communications supportées par l'exécutif SynDEx à travers le réseau, point-à-point ou routées. Ceci permet de mesurer le surcoût de l'exécutif SynDEx.

Pour effectuer ces mesures, nous avons utilisé la méthode chronométrique présentée au chapitre précédent.

5.1 Conditions de mesures

Toutes les mesures ont été effectuées en plaçant les programmes en mémoire externe. Nous avons pris soin de corriger les résultats lorsqu'une instruction de mesure, de coût 7μs, était incluse dans le code mesuré.

Dans tous les cas, nous n'avons pas considérés, pour effectuer le calcul des débits de communications, les messages contenant peu d'octets, pour la raison exposée en 3.2. On peut faire les mêmes remarques concernant l'interprétation de ces durées.

Nous avons remarqué des variations importantes des durées de transmission d'un message en fonction du nombre de portes connectées au bus logiciel d'un processeur. Cette variation est essentiellement dûe à l'instruction non déterministe **ALT** du Transputer utilisée dans le code du bus logiciel. Afin de s'affranchir de ces variations, nous avons effectué les mesures à nombre constant de portes connectées au bus logiciel. Des méthodes permettant d'éviter l'utilisation du **ALT** qui n'est pas souhaitable dans une approche déterministe comme la nôtre, seront étudiées ultérieurement.

5.2 Durée des communications intra-processeurs

Il s'agit toujours ici de communications inter-processus qui s'effectuent par l'intermédiaire de la mémoire. Le schéma utilisé figure 5.1 est le même qu'au chapitre 3.1. Tous les transferts ont ici été moyennés sur 1000 itérations. On obtient un débit de 13,3 Moctets/s et un start-up de 12 μs, très proches des résultats obtenus au chapitre 3.1.

5.3 Durée des communications inter-processeurs point-à-point

Le schéma effectué est cette fois celui de la figure 5.2. Les messages passent par :

1. une porte d'émission PE attachée au processus émetteur
2. le bus logiciel BL du processeur émetteur
3. une porte d'accès au lien inter-processeur ou PPL
4. le lien physique

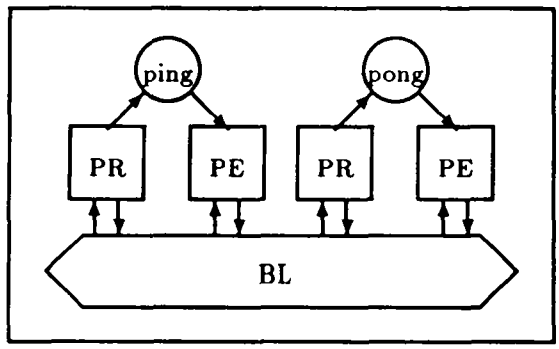

Figure 5.1: Durée des communications intra-processeur

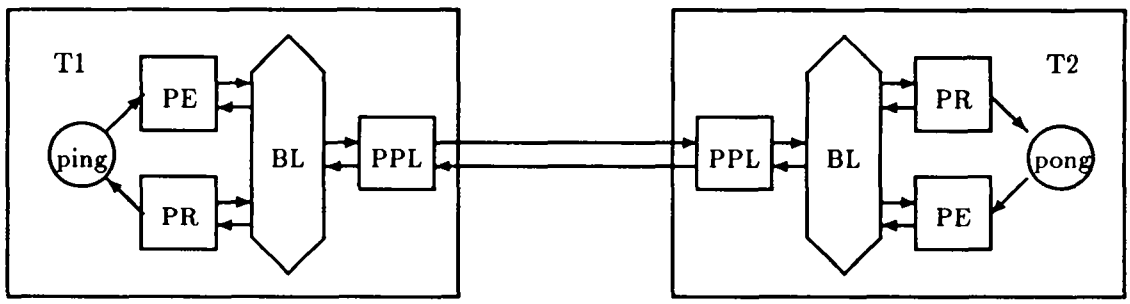

Figure 5.2: Communications inter-processeur point-à-point

5. la PPL du processeur récepteur

6. le BL du processeur récepteur

7. une porte de réception PR attachée au processus récepteur

Pour bien séparer les problèmes, nous avons mesuré séparément les durées des séquences PE-BL-PPL et Lien-PPL-BL-PR.

Le start-up obtenu est dans tous les cas supérieur à 110 μs pour la séquence PE-BL-PPL et à 200 μs pour la séquence Lien-PPL-BL-PR. Ces résultats sont largement supérieurs à ceux obtenus au chapitre 3.1 et mettent en évidence le surcoût de l'exécutif.

Le débit obtenu pour la séquence PE-BL-PPL est de l'ordre de 4 Moctets/s (soit environ le tiers du débit d'une recopie interne, dû effectivement aux 3 transferts mémoire entre processus émetteur, PE, BL et PPL) et celui mesuré pour la séquence Lien-PPL-BL-PR est de l'ordre de 1,5 Moctets/s.

5.4 Durée des communications point-à-point à travers le crossbar C004

Le schéma effectué figure 5.3 est le même que précédemment, mais les liaisons passent par le crossbar C004.

Le start-up de la communication est très peu supérieur au précédent, mais le débit mesuré (séquence Lien-PPL-BL-PR) reste du même ordre de grandeur (1,3 Moctets/s). La diminution de débit à travers le crossbar constatée au chapitre 3.1 est négligeable devant le surcoût apporté par l'exécutif.

5.5 Parallélisme des communications sur un Transputer

Par un placement convenable des processus *ping-pong*, nous avons effectué en parallèle des entrées-sorties sur les différents liens d'un Transputer. Nous avons utilisé ici les schémas de la figure 5.4.

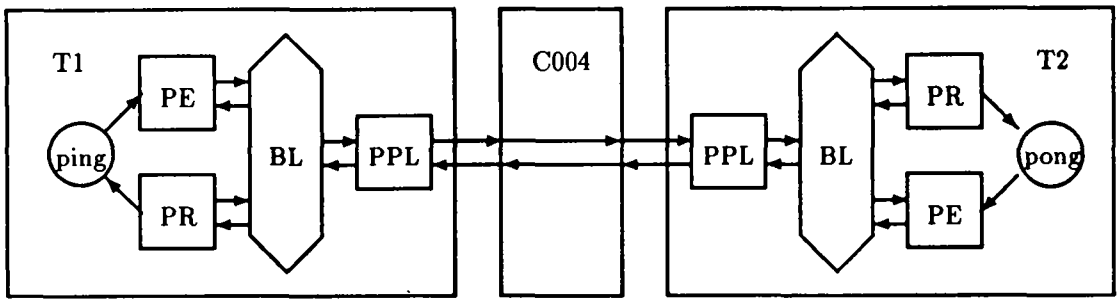

Figure 5.3: Communications point-à-point à travers le crossbar C004

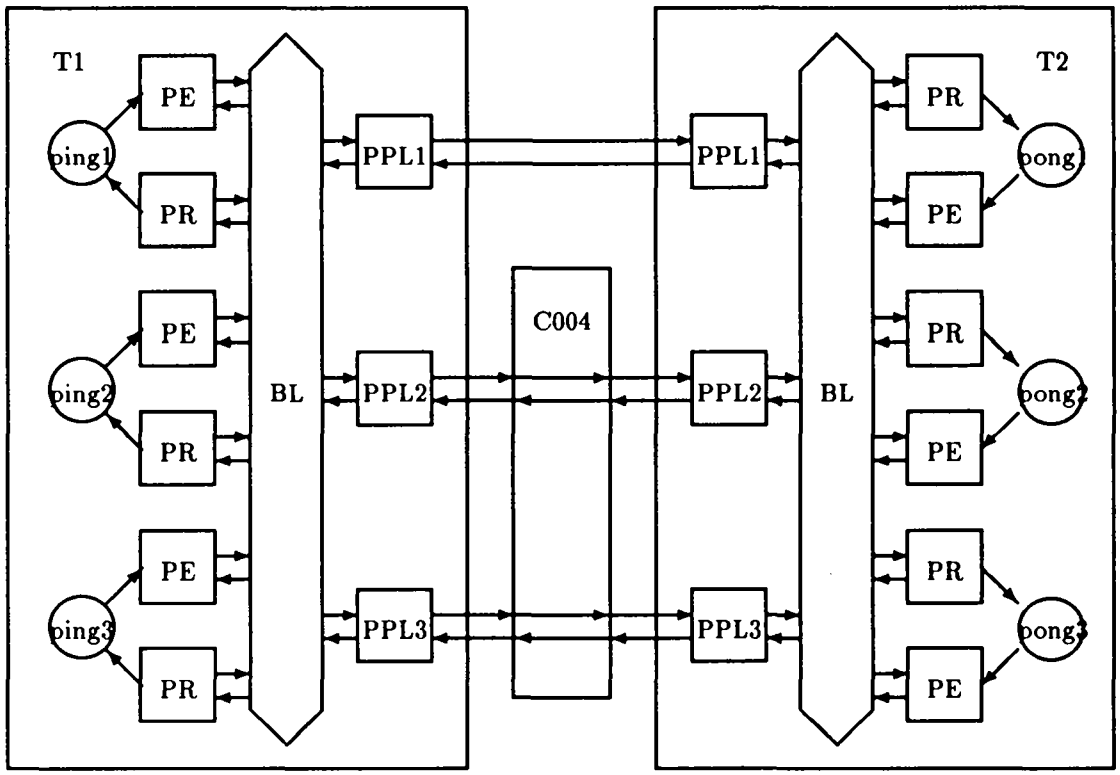

Figure 5.4: Communications concurrentes

Certains processeurs exécutent en parallèle plusieurs séquences PE-BL-PPL ou PPL-BL-PR. Les résultats obtenus indiquent clairement que les séquences PE-BL-PPL et PPL-BL-PR ne sont pas indivisibles, contrairement à l'hypothèse faite pour les calculs du diagramme temporel faits par SynDEx.

Il semble qu'à l'émission et même en général à la réception, les séquences s'entrelacent : les temps de start-up mesurés peuvent s'expliquer par le fait qu'un processeur supportant 2 processus 1 et 2 émettant en parallèle peut par exemple effectuer PE(1)-PE(2)-BL(1)-BL(2)-PPL(1)-PPL(2). A la réception, le phénomène peut être plus complexe car les communications arrivent en général déjà décalées. Cependant des entrelacements peuvent là aussi apparaître.

Ces phénomènes ont pour conséquence des réductions de débit apparent sur les séquences internes et des augmentation importantes des temps de start-up, qui peuvent être près de n fois plus élevés que dans le cas simple, lorsque n liens d'un Transputer sont utilisés en parallèle.

L'entrelacement des séquences n'est pas facilement prévisible. Lorsque plusieurs communication parallèles se répètent, on peut constater des dérives croissantes entre l'émetteur et le récepteur, qui peuvent résulter en des changements dans l'entrelacement des séquences sur l'un des processeurs.

5.6 Parallélisme des calculs et des communications sur un Transputer

Comme au chapitre 3.1, nous avons utilisé le *processus mange-temps* s'exécutant sur le FPU et le CPU du Transputer. Nous l'avons interrompu par l'arrivée d'une communication afin de mesurer pendant combien de temps l'exécution d'une séquence PPL-BL-PR interrompt les calculs. Cette durée se révèle à peu près égale à la durée d'exécution d'une séquence PE-BL-PPL. Le modèle utilisé par SynDEx pour calculer le diagramme temporel, où le calcul n'est pas perturbé par l'arrivée d'une communication, doit donc être revu.

Nous avons aussi mesuré, par le même procédé, la durée qu'il faut à un Transputer pour router une communication. Ceci correspond au déroulement d'une séquence PPL1-BL-PPL2. On obtient là encore des durées supérieures à 150 μs, que le modèle de calcul de SynDEx ne prend pas en compte.

Chapitre 6

Exemple de chronométrage d'un égaliseur adaptatif sur un bi-Transputer

6.1 Présentation

L'égaliseur adaptatif est une application classique de traitement du signal, où l'on cherche à identifier les coefficients d'un filtre inconnu en adaptant les coefficients d'un filtre adaptatif à l'aide d'un algorithme de gradient stochastique. La figure 6.1 montre le graphe flot de données représentant l'égaliseur dans l'environnement graphique SynDEx [8].

6.2 Résultats

On voit dans la partie droite de la figure 6.1 le diagramme temporel calculé par SynDEx pour l'implantation de l'algorithme de l'égaliseur sur deux Transputers `root` et `P` reliés par un lien. L'annexe 1 présente le code séquentiel généré par SynDEx pour cette implantation, avec instructions et portes de mesures.

Le tableau 6.2 présente les dates du diagramme temporel calculé par SynDEx d'une part, et les mesures en temps réel sur le code séquentiel et sur le code parallèle d'autre part. Le format utilisé pour chaque élément du tableau est le suivant : "date de début : nom du processus (durée du processus)". On y a représenté par des flèches les envois et les réceptions de messages. Notez que les mesures temps-réel ont été corrigées en retranchant d'une part les durées d'ouverture et de fermeture de fichier et d'autre part les durées des instructions de chronométrage.

Les conclusions que l'on peut tirer de l'observation de ce tableau sont les suivantes :

- Tout d'abord, on constate que l'ordre partiel du graphe logiciel est respecté à l'exécution autant pour le code séquentiel que pour le code pseudo-parallèle, ce qui montre que l'implantation des communications inter-processus est correcte, aussi bien dans leur forme intra-processeur qu'inter-processeur.

  Remarque : les processus de calcul R1 et R2 qui représentent les retards en traitement du signal mémorisent des données d'une exécution sur l'autre du graphe logiciel et peuvent donc être vus autant comme processus initiaux (cas du diagramme temporel calculé) que comme processus terminaux (cas du code généré).

- On constate ensuite que le temps de réponse calculé (et toutes les dates intermédiaires) est en-dessous de la réalité. En effet, le calcul du diagramme temporel ne prend pas en compte le fait que la séquence PE-BL-PPL consomme du temps du processeur de calcul. Ce décalage se voit sur le tableau 6.2 après la fin du processus D1 qui précède l'émission d'un message. Le modèle de calcul doit absolument être amélioré dans ce sens.

- On constate enfin que le code parallèle est nettement moins efficace que le code séquentiel. Ceci est dû principalement à l'utilisation de l'instruction `PAR` d'OCCAM qui impose l'implantation des communications intra-processeur par mouvements mémoires (canaux OCCAM) et introduit des surcoûts de changement de contexte.

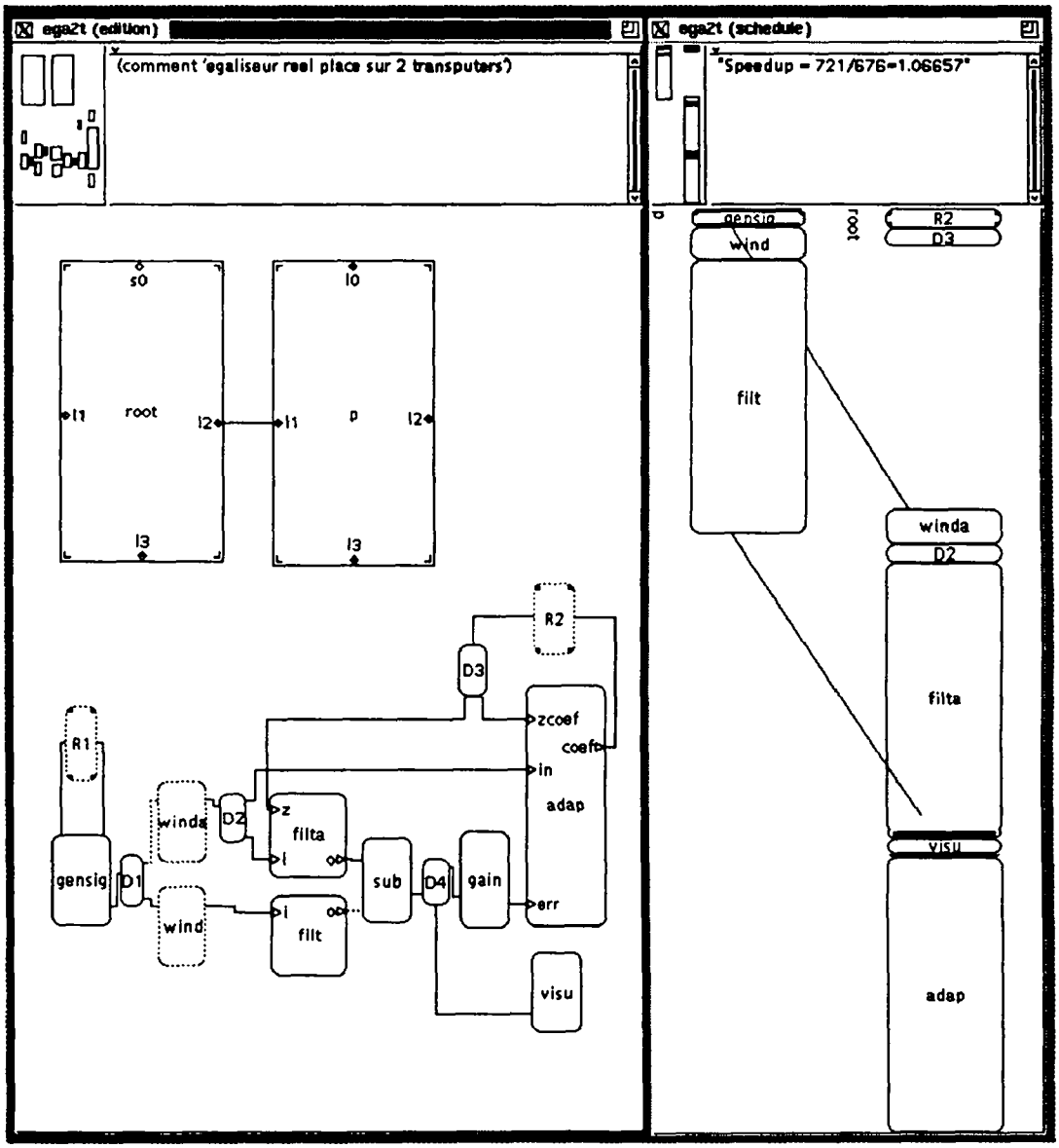

Figure 6.1: Environnement graphique SynDEx

Calculs faits par **SynDEx**		Mesures sur le code séquentiel		Mesures sur le code parallèle	
Proc. R	Proc. P	Proc. R	Proc. P	Proc. R	Proc. P
0: R2(6)	0: ...(242)	0: ...(17)	0: ...(276)	0: ...(73)	0:...(439)
6: D3(15)		17: D3(16)			
21: R1(1)					
22:gensig(10)					
32:D1(3)→		33: gensig(10)			
35:winda(25)		43: D1(3)			
60: D2(15)		46:→(93)		73: R1	
				81:gensig(10)	
75:filta(36)				101: D3(16)	
				R1:127	
111: ...(399)		139:winda(28)		129: R2	
				142: D1(3)	
		167:D2(16)		→	
	→	183:filta(37)	→		
	242:wind(25)	220: ...(376)	276:wind(34)	262:winda(29)	
	267:filt(36)				
	303:←		310:filt(35)	305:D2(16)	
			345:←(80)	346:filta	→
			:425	filta	439: wind(27)
←				filta:489	477: filt(36)
510: sub(3)				489:...(267)	
513: D4(3)					513:←(93)
516: visu(1)					
517:gain(4)					
521:adap(41)		←			
:562		596:sub(3)			:606
		599:D4(2)			
		601:visu(1)			
		602:gain(4)			
		606:adap(39)			
		645:R2(14)			
		669:R1(9)			
		:678		←	
				756:sub(3)	
				759:D4(3)	
				762:visu(1)	
				763:gain(3)	
				766:adap(41)	
				R2:807	

Figure 6.2: Traces d'exécution de l'égaliseur adaptatif

Chapitre 7

Conclusions et perspectives

7.1 Ordonnancement des communications

L'exécutif généré par SynDEx dans sa version actuelle est une implantation directe de l'exécutif décrit en [9]. L'exécutif, dans sa partie "communications inter-processus inter-processeur", est composé de processus élémentaires du noyau (PE, PR, BL et PPL) mis en parallèle. Cela entraîne des surcoûts d'ordonnancement dynamique de ces processus et de mouvements mémoire pour les faire communiquer. Par ailleurs, l'ordonnancement dynamique introduit un indéterminisme sur l'ordonnancement de ces processus, en contradiction avec le modèle déterministe de calcul du diagramme temporel fait par SynDEx.

Comme on l'a vu, l'implantation séquentielle de la partie calculs (séquentialisation des processus de calculs et donc l'implantation des communications intra-processeur par mémoire partagée) amène déjà un gain de performances significatif. On peut en attendre de même d'une implantation séquentielle de la partie communications inter-processeur de l'exécutif.

Sur chaque processeur de communication, la séquence des processus de communication qui y ont été placés s'exécutera en parallèle avec la séquence des processus de calcul et les séquences des autres processeurs de communication. Pour chaque communication inter-processeur, les séquences se synchroniseront directement entre elles, les données étant simplement communiquées par l'intermédiaire de la mémoire partagée entre les processeurs (de calcul et de communication).

Ainsi le noyau de l'exécutif (à adapter à chaque type de processeur) se réduira à quatre éléments, deux de synchronisation intra- nœud processeur et deux de communication inter- nœud processeur.

La séquentialisation des communications apportera un gain annexe non négligeable : l'ordre des communications étant prédéterminé, leur routage le sera également, aussi il ne sera plus nécessaire de transmettre ni de décoder des informations de routage (suppression du bus logiciel et de ses inconvénients).

Une première évaluation de cette méthode a permis de chiffrer sur un multi-Transputer le start-up d'une communication à environ 17 μs, à comparer aux centaines de μs mesurées avec l'exécutif actuel.

7.2 Interaction entre calculs et communications

Les calculs de diagramme temporel faits actuellement par SynDEx se basent sur l'hypothèse que les calculs et les communications s'effectuent en parallèle. Or cette étude a montré que les processus de calcul sont interrompus par les processus élémentaires du noyau pendant la durée d'établissement d'une communication. Ces durées d'interruption ne peuvent pas être négligées face à celles des processus de calcul.

Un nouveau modèle d'exécutif prenant en compte de manière plus précise les interactions entre processeurs de calcul et de communication est en cours d'étude.

7.3 Prévisions de performances

Si l'on dispose des composants d'un multi-processeur (processeurs et liaisons physiques inter-processeurs), en appliquant la méthode chronométrique décrite dans ce document, on peut chronométrer d'une part les processus de calcul sur un seul processeur et d'autre part les processus de communication inter-processeur générés par SynDEx sur un multi-processeur minimum. A partir de là, on peut avec SynDEx prévoir les performances de l'implantation d'un algorithme sur tout multi-processeur construit avec ces composants.

Même dans le cas où on ne dispose pas de ces composants matériels, on peut obtenir les durées des processus de calcul et de communication soit par estimation, soit par calcul à l'aide de la documentation ou d'outils fournis par le constructeur des composants matériels.

7.4 Placement ordonnancement

C'est à partir des durées des processus de calculs et de communications que SynDEx optimise le placement-ordonnancement du graphe logiciel sur le graphe matériel et calcule le temps de réponse de l'application et donc son "speedup" (rapport entre les temps de réponse mono- et multi- processeur).

L'heuristique de placement ordonnancement est un algorithme glouton prenant en compte des contraintes de placement. Son temps d'exécution est une fonction linéaire du nombre de processeurs et du nombre de nœuds et d'arcs du graphe logiciel (quelques secondes en général à quelques minutes pour de grosses applications).

C'est grâce à cette heuristique que SynDEx peut générer, automatiquement, un exécutif optimisé pour un alogrithme et un multi-processeur donnés.

7.5 Génération de code

Contrairement aux systèmes d'exploitation distribués standards (du genre de MACH et CHORUS) qui fournissent des services à travers une copie d'un noyau résidant sur chaque processeur, SynDEx produit un exécutif taillé sur mesure pour l'application, construit à la compilation à partir d'une bibliothèque formant le noyau générique de l'exécutif SynDEx.

L'utilisateur n'a pas d'autre code à écrire que celui de ses processus de calcul. Tout le reste (séquencement et appel des processus de calcul, allocation de la mémoire nécessaire aux communications inter-processus -intra- ou inter- processeur- et code des processus de communication) est généré automatiquement par SynDEx à partir des graphes logiciel et matériel, des résultats du placement ordonnancement et du noyau générique de l'exécutif SynDEx. La simplicité du noyau facilite son portage sur de nouveaux composants matériels.

Le temps de génération de code est de l'ordre de quelques secondes.

La méthode de génération de code garantit que le comportement entrées-sorties de l'application sera le même sur une architecture multiprocesseur que sur l'architecture monoprocesseur sur laquelle l'application aura été préalablement déboguée.

7.6 Portage multi-processeur

Alors que dans un environnement classique, le portage sur un multi-processeur d'une application déboguée sur un mono-processeur demande plusieurs jours (voire plusieurs semaines) de codage et de déboguage, avec l'environnement SynDEx, ce portage est l'affaire de quelques minutes, au plus de quelques heures si le graphe représentant l'algorithme est de taille importante et que plusieurs cycles de placement ordonnancement sont nécessaires pour obtenir un placement satisfaisant.

Annexe 1

Code généré pour l'exemple égaliseur adaptatif

Nous présentons ici le code séquentiel généré par SynDEx dans quatre fichiers :

- un fichier include définissant les constantes nécessaires au routage
- un fichier pour chaque processeur (deux dans le cas de l'égaliseur)
- un fichier de configuration décrivant les processeurs, leurs inter-connexions et le placement des programmes et des canaux de communication

Nous n'avons pas inclus ici le source des fonctions de calcul normalement fourni par l'utilisateur ni le makefile généré par SynDEx.

1.1 Fichier de configuration ega2t.pgm

```
------------------------------------------------------------
-- machine ega2t (16 July 1991 2:27:56 pm ) SynDEx v1.1 INRIA
-- egaliseur reel place sur 2 transputers
------------------------------------------------------------
#INCLUDE "hostio.inc" -- constains SP protocol
#INCLUDE "linkaddr.inc" -- link address constants
PROTOCOL MESSAGE IS BYTE::[]BYTE: -- 0:routID 1:destID 2:data...
#USE "ega2t1.c8h" -- processor root
#USE "ega2t2.c8h" -- processor p
CHAN OF SP fs, ts: -- server channels
CHAN OF MESSAGE root.12.p.11:
CHAN OF MESSAGE p.11.root.12:

PLACED PAR
  PROCESSOR 0 T8
    [1000]INT bouchon:
    PLACE bouchon IN WORKSPACE:
    PLACE ts AT link0.out:
    PLACE fs AT link0.in:
    PLACE root.12.p.11 AT link2.out:
    PLACE p.11.root.12 AT link2.in:
    root ( fs, ts, p.11.root.12, root.12.p.11 )

  PROCESSOR 1 T8
    [1000]INT bouchon:
    PLACE bouchon IN WORKSPACE:
    PLACE p.11.root.12 AT link1.out:
    PLACE root.12.p.11 AT link1.in:
    p ( root.12.p.11, p.11.root.12 )

-- That's all folks !! --
```

1.2 Fichier include ega2t.inc

```
------------------------------------------------------------
-- machine ega2t (16 July 1991 2:27:52 pm ) SynDEx v1.1 INRIA
-- egaliseur reel place sur 2 transputers
------------------------------------------------------------
#INCLUDE "hostio.inc" -- server protocol
#USE "hostio.lib" -- server functions
PROTOCOL MESSAGE IS BYTE::[]BYTE: -- 0:routID 1:destID 2:data...
#USE "syndex.lib" -- PES, BL, PE and PR for primitive protocols
-- (protocol INT "is" INT) -- primitive protocol
-- (protocol REAL32 "is" REAL32) -- primitive protocol
-- (protocol TABREAL32 "is" :INT<10:REAL32)
PROTOCOL TABREAL32 IS INT::[]REAL32 :
#USE "ega2t.lib" -- user protocols and functions

VAL NbIterations IS 1:
VAL local IS 255(BYTE):

-- destIDs in Procr root -- R2 D3 R1 gensig D1 winda D2 filta sub D4 visu gain adap --
VAL root.s0 IS 0(BYTE):-- ps1 >< connected to server
VAL root.l2 IS 1(BYTE):-- pp1 >< p.l1
VAL D1.o1 IS 2(BYTE):-- PE >D> wind.i/p  >S> PE
VAL sub.i2 IS 3(BYTE):-- PR < filt.o/p
VAL visu.s IS 4(BYTE):-- PS >D> root.s0/root  >S> PS
VAL root.moni IS 5(BYTE): -- monitoring-gate
VAL root.disp IS 6(BYTE): -- displaying-gate
VAL root.sizeBL IS 7:
-- routeIDs through Procr root --
VAL root.l2.p.dir IS 0(BYTE):
VAL root.l2.p.rev.p IS local:

-- destIDs in Procr p -- wind filt --
VAL p.l1 IS 0(BYTE):-- pp1 >< root.l2
VAL wind.i IS 1(BYTE):-- PR < D1.o1/root
VAL filt.o IS 2(BYTE):-- PE >D> sub.i2/root  >S> PE
VAL p.moni IS 3(BYTE): -- monitoring-gate
VAL p.sizeBL IS 4:
-- routeIDs through Procr p --
VAL root.l2.p.dir.root IS local:
VAL root.l2.p.rev IS 0(BYTE):

-- routes cross-references --
-- root.l2.p.dir -- route root -> p = ( root.l2 p.l1 ) --
-- root.l2.p.rev -- route root <- p, reverse route --
-- D1.o1 <S< wind.i
-- D1.o1 >D> wind.i
-- sub.i2 <D< filt.o
-- sub.i2 >S> filt.o
-- That's all folks !! --
```

1.3 Fichier ega2t1.occ pour le processeur root

```
------------------------------------------------------------
-- machine ega2t (16 July 1991 2:27:53 pm ) SynDEx v1.1 INRIA
-- egaliseur reel place sur 2 transputers
------------------------------------------------------------
#INCLUDE "ega2t.inc"
-- (processor root "i/o" PSL s0=0, PPL l1=0 l2=9 l3=0)
-- (placeact "on" root "schedule" R2 D3 R1 gensig D1 winda D2 filta sub D4
--   visu gain adap)
PROC root ( CHAN OF SP is0, os0, CHAN OF MESSAGE il2, ol2 )
  -- system connections : BL to root.s0 root.l2 D1.o1 sub.i2 visu.s
  [root.sizeBL]CHAN OF MESSAGE iBL, oBL:
  -- application process connections :
  CHAN OF REAL32 D1.o1.wind.i.D, filt.o.sub.i2.D:
  CHAN OF SP root.s0.visu.s, visu.s.root.s0:
  [2]CHAN OF INT strobes: -- monitor strobes
  PRI PAR -------- body of Procr root --

    PAR -- system
      [512]BYTE fifoIn, fifoOut:
      PSL ( oBL[INT root.s0], iBL[INT root.s0], is0, os0, fifoIn, fifoOut )
      [512]BYTE fifoIn, fifoOut:
      PPL ( oBL[INT root.l2], iBL[INT root.l2], il2, ol2, fifoIn, fifoOut )
      PEREAL32 ( D1.o1.wind.i.D, oBL[INT D1.o1], iBL[INT D1.o1],
          [root.l2.p.dir, wind.i] )
      PRREAL32 ( filt.o.sub.i2.D, oBL[INT sub.i2], iBL[INT sub.i2],
          [root.l2.p.dir, filt.o] )
      PSP ( visu.s.root.s0, root.s0.visu.s, oBL[INT visu.s], iBL[INT visu.s],
          [local, root.s0, local, visu.s] )
      BLrr ( iBL, oBL, [ [root.l2, root.l2.p.dir.root]] )
      [18]INT measures:
      monitor ( oBL[INT root.moni], iBL[INT root.moni], strobes, measures,
          [ [local,root.disp], [root.l2.p.dir,p.moni] ],
          "- 0:root:1 2:D3 3:gensig 4:D1 5:D1.o1 6:winda 7:D2 8:filta 9:sub.i2
          10:sub 11:D4 12:visu 13:gain 14:adap 15:R2 16:R1" )
      display(oBL[INT root.disp], iBL[INT root.disp],
          root.moni, root.disp, root.s0)

    -- application
    INT R2.o.D3.i.l: -- buffer length used
    [10]REAL32 R2.o.D3.i:
    INT D3.o2.adap.zcoef.l: -- buffer length used
    [10]REAL32 D3.o2.adap.zcoef:
    INT D3.o1.filta.z.l: -- buffer length used
    [10]REAL32 D3.o1.filta.z:
    INT R1.o.gensig.zi, gensig.zo.R1.i:
    REAL32 gensig.sig.D1.i, D1.o2.winda.i, D1.o1.wind.i:
    INT winda.o.D2.i.l: -- buffer length used
    [10]REAL32 winda.o.D2.i:
    INT D2.o2.adap.in.l: -- buffer length used
    [10]REAL32 D2.o2.adap.in:
    INT D2.o1.filta.i.l: -- buffer length used
    [10]REAL32 D2.o1.filta.i:
    REAL32 filta.o.sub.i1, filt.o.sub.i2:
    REAL32 sub.er.D4.i, D4.o2.visu.er, D4.o1.gain.i:
    INT32 visu.s.sid:
    REAL32 gain.o.adap.err:
    INT adap.coef.R2.i.l: -- buffer length used
    [10]REAL32 adap.coef.R2.i:
    BYTE result: -- for so.open and so.close
    SEQ
      -- initializations
      strobes[0] ! 0 -- wait startDate
```

```
      CHAN OF TABREAL32 init:
      PAR
        init ! 9::[0.0(REAL32),0.0(REAL32),0.0(REAL32),0.0(REAL32),0.0(REAL32),
                  0.0(REAL32),0.0(REAL32),0.0(REAL32),0.0(REAL32)]
        init ?  R2.o.D3.i.l::R2.o.D3.i
      CHAN OF INT init:
      PAR
        init ! 0
        init ?  R1.o.gensig.zi
      -- so.open( root.s0.visu.s, visu.s.root.s0, "erreur",
      --     spt.text, spm.output, visu.s.sid, result)
      SEQ clock=0 FOR NbIterations
        SEQ
          strobes[1] ! 2
          diffuseTabReel ( [R2.o.D3.i FROM 0 FOR R2.o.D3.i.l],
              D3.o2.adap.zcoef.l, D3.o2.adap.zcoef,
              D3.o1.filta.z.l, D3.o1.filta.z )
          strobes[1] ! 3
          gensig ( R1.o.gensig.zi, gensig.zo.R1.i, gensig.sig.D1.i )
          strobes[1] ! 4
          diffuseReel ( gensig.sig.D1.i, D1.o2.winda.i, D1.o1.wind.i )
          strobes[1] ! 5
          D1.o1.wind.i.D ! D1.o1.wind.i
          strobes[1] ! 6
          fenetre ( D1.o2.winda.i, winda.o.D2.i.l, winda.o.D2.i, 9,clock )
          strobes[1] ! 7
          diffuseTabReel ( [winda.o.D2.i FROM 0 FOR winda.o.D2.i.l],
              D2.o2.adap.in.l, D2.o2.adap.in, D2.o1.filta.i.l, D2.o1.filta.i )
          strobes[1] ! 8
          filtreReel ( [D3.o1.filta.z FROM 0 FOR D3.o1.filta.z.l],
              [D2.o1.filta.i FROM 0 FOR D2.o1.filta.i.l], filta.o.sub.i1 )
          strobes[1] ! 9
          filt.o.sub.i2.D ? filt.o.sub.i2
          strobes[1] ! 10
          moins ( filta.o.sub.i1, filt.o.sub.i2, sub.er.D4.i )
          strobes[1] ! 11
          diffuseReel ( sub.er.D4.i, D4.o2.visu.er, D4.o1.gain.i )
          strobes[1] ! 12
          -- FLROUTREAL32 ( D4.o2.visu.er, root.s0.visu.s, visu.s.root.s0,
          --     visu.s.sid )
          strobes[1] ! 13
          multp ( 0.1(REAL32), D4.o1.gain.i, gain.o.adap.err )
          strobes[1] ! 14
          egaAdapteReel ( gain.o.adap.err,
              [D2.o2.adap.in FROM 0 FOR D2.o2.adap.in.l],
              [D3.o2.adap.zcoef FROM 0 FOR D3.o2.adap.zcoef.l],
              adap.coef.R2.i.l, adap.coef.R2.i )
          strobes[1] ! 15
          CHAN OF TABREAL32 tmp:
          PAR
            tmp !  adap.coef.R2.i.l::adap.coef.R2.i
            tmp ?  R2.o.D3.i.l::R2.o.D3.i
          strobes[1] ! 16
          CHAN OF INT tmp:
          PAR
            tmp !  gensig.zo.R1.i
            tmp ?  R1.o.gensig.zi
          strobes[1] ! 17
        -- finalizations
        -- so.close( root.s0.visu.s, visu.s.root.s0, visu.s.sid, result)
        strobes[0] ! 1 -- record endDate
: -- end of processor root
-- That's all folks !! --
```

1.4 Fichier ega2t2.occ pour processeur p

```
------------------------------------------------------------
-- machine ega2t (16 July 1991 2:27:55 pm ) SynDEx v1.1 INRIA
-- egaliseur reel place sur 2 transputers
------------------------------------------------------------
#INCLUDE "ega2t.inc"
-- (processor p "i/o" PPL 10=0 11=9 12=0 13=0)
-- (placeact "on" p "schedule" wind filt)
PROC p ( CHAN OF MESSAGE il1, ol1 )
  -- system connections : BL to p.l1 wind.i filt.o
  [p.sizeBL]CHAN OF MESSAGE iBL, oBL:
  -- application process connections :
  CHAN OF REAL32 D1.o1.wind.i.D:
  CHAN OF REAL32 filt.o.sub.i2.D:
  [2]CHAN OF INT strobes: -- monitor strobes
  PRI PAR -------- body of Procr p --

    PAR -- system
      [512]BYTE fifoIn, fifoOut:
      PPL ( oBL[INT p.l1], iBL[INT p.l1], il1, ol1, fifoIn, fifoOut )
      PRREAL32 ( D1.o1.wind.i.D, oBL[INT wind.i], iBL[INT wind.i],
          [root.l2.p.rev, D1.o1] )
      PEREAL32 ( filt.o.sub.i2.D, oBL[INT filt.o], iBL[INT filt.o],
          [root.l2.p.rev, sub.i2] )
      BLrr ( iBL, oBL, [ [p.l1, root.l2.p.rev.p]] )
      [6]INT measures:
      monitor ( oBL[INT p.moni], iBL[INT p.moni], strobes, measures,
          [ [root.l2.p.rev,root.moni] ],
          "- 0:p:1 2:wind.i 3:wind 4:filt 5:filt.o" )

    -- application
    REAL32 D1.o1.wind.i:
    INT wind.o.filt.i.l: -- buffer length used
    [10]REAL32 wind.o.filt.i:
    REAL32 filt.o.sub.i2:
    BYTE result: -- for so.open and so.close
    SEQ
      -- initializations
      strobes[0] ! 0 -- wait startDate
      SEQ clock=0 FOR NbIterations
        SEQ
          strobes[1] ! 2
          D1.o1.wind.i.D ? D1.o1.wind.i
          strobes[1] ! 3
          fenetre ( D1.o1.wind.i, wind.o.filt.i.l, wind.o.filt.i, 9,clock )
          strobes[1] ! 4
          filtreReel ( [0.0(REAL32),0.0(REAL32),0.0(REAL32),0.0(REAL32),
              1.0(REAL32),0.0(REAL32),0.0(REAL32),0.0(REAL32),0.0(REAL32)],
              [wind.o.filt.i FROM 0 FOR wind.o.filt.i.l], filt.o.sub.i2 )
          strobes[1] ! 5
          filt.o.sub.i2.D ! filt.o.sub.i2
      -- finalizations
      strobes[0] ! 1 -- record endDate
: -- end of processor p
-- That's all folks !! --
```

Bibliographie

[1] C. Lavarenne, O. Seghrouchni, Y. Sorel, M. Sorine
SynDEx, un environnement de programmation pour applications de traitement du signal distribuées.
Treizième Colloque GRETSI (Septembre 1991).

[2] 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 (Juillet 1991).

[3] P. Rolin
Réseaux locaux, normes et protocoles
Hermes (1988)

[4] N. Ghezal
Quelques méthodes de répartition et d'ordonnancement de processus de traitement du signal sur un réseau de Transputer
Thèse; Université de Paris XI- ORSAY (1988)

[5] INMOS
The Transputer Databook
Prentice Hall (1989)

[6] INMOS
OCCAM 2 Reference Manual
Prentice Hall (1988)

[7] F. Guidec
Performances des communications sur le T-Node
La Lettre du Transputer No.7
Laboratoire d'Informatique de Besançon (1990)

[8] 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)

[9] 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)

ISSN 0249 - 6399