Méthode d'évaluation des temps d'exécution de programmes cadencés par les données pour processeurs µPD7281 (1)

Computation time estimation of NEC µPD7281 data flow graphs

Patrick MARTIN

LACIS/ITEPEA, UFR Sciences et Techniques de l'Université de Rouen, Place E. Blondel B.P. 118, 76134 Mont Saint Aignan Cedex

Patrick MARTIN est ingénieur à l'Université de Rouen. Il s'intéresse à la mise en œuvre d'architecture de processeurs spécialisés en vue de traitements d'images temps réel. Travaux sur contrats industriels relatifs au contrôle qualité.

Richard LECORDIER

LCIA/ITEPEA INSA de Rouen, Place E. Blondel, B.P. 8, 76131 Mont Saint Aignan Cedex

Richard LECORDIER est maître de conférences à l'IUT de Rouen. Il centre ses travaux sur l'architecture des systèmes de traitement du signal et sur des applications industrielles liées au traitement des images.

Denis de BRUCQ

LACIS/ITEPEA, UFR Sciences et Techniques de l'Université de Rouen, Place E. Blondel B.P. 118, 76134 Mont Saint Aignan Cedex

Denis de BRUCQ est professeur à l'Université de Rouen. Les recherches sous sa direction immédiate, en théorie du signal, permettent l'apprentissage, l'identification et la détection en ligne, de changements de modèles. Les extensions paramétrées, obtenues du modèle de Kalman, s'appliquent à des évolutions aléatoires d'état non-linéaires et donc non gaussiennes.

Roland DEBRIE

LCIA/ITEPEA INSA de Rouen, Place E. Blondel, B.P. 8, 76131 Mont Saint Aignan Cedex

Roland DEBRIE est professeur des universités à l'INSA de Rouen. Il est responsable du groupe visionique. Activités de recherche, systèmes de perception stéréoscopiques multispectrales.

(1) Cette étude a été financée par l'ANVAR sous le contrat n° A 87 07 071P.

RÉSUMÉ

Nous présentons une méthode permettant d'estimer le temps de calcul d'un processeur « data flow » NEC μPD7281 à partir de programmes écrits sous la forme de graphes de flot. Cette méthode présente l'avantage d'éviter à l'utilisateur de transcrire le graphe en langage assembleur et de le simuler ensuite afin d'en évaluer le temps d'exécution.

MOTS CLÉS

Graphe de flot, flot de données, jeton, cycle pipeline, bus en anneau, nœud, arc.

SUMMARY

The method we introduce here enables to estimate the computation time of a NEC μPD7281 data flow processor from written data flow computer programs. This method has the advantage of avoiding transcription into μPD7281 machine language instruction and of simulating it afterwards in order to evaluate the execution time of the object program.

KEY WORDS

Flow graph, data flow, token, pipelined cycle, ring bus, node, arc.

Introduction

Dans un environnement temps réel, le traitement du signal et plus particulièrement le traitement d'images nécessitent des moyens de calcul très rapides. Bien que les processeurs classiques de type von Neumann aient fait des progrès en performances considérables au cours de cette décennie ; ceux-ci demeurent souvent insuffisants pour des applications temps réel telles que le traitement d'images.

Maintes solutions ont vu le jour ; certaines utilisent des processeurs classiques fonctionnant en parallèle. Nous connaissons les difficultés de tels dispositifs pour le traitement du signal. Les traitements sont séparés en processus communicant par boîtes aux lettres et synchronisés par sémaphores. L'ordonnancement des processus est pénalisant en temps. Notons les difficultés, pour ces processeurs, de communiquer entre eux. Un processeur perçoit une donnée ou un résultat partiel à la suite d'un test de présence effectué souvent dans une boucle d'attente ou à l'aide d'interruptions. Ces deux méthodes sont aussi pénalisantes.

D'où la nécessité de rechercher des dispositifs ne présentant pas les inconvénients mentionnés. Les machines cadencées par les données répondent aux problèmes posés.

Dans une machine cadencée par les données le départ d'un processus est déclenché par l'arrivée d'une donnée attendue. Cette propriété entraîne que les processus ne nécessitant pas de communication entre eux peuvent très aisément être exécutés en parallèle.

En outre les processeurs fonctionnant selon ce concept peuvent être interconnectés dans une architecture pipeline pour une exécution en parallèle des processus.

Le passage de résultats partiels entre processeurs ou entre processus peut être exécuté de façon asynchrone sans l'aide de sémaphores ou d'interruptions.

Les processeurs cadencés par les données sont bien adaptés aux architectures construites autour d'une structure en anneau pipeline, ce dernier servant de voie de communication entre les différentes instructions d'un processus en utilisant la même unité de calcul. Une description d'une architecture généralisée est donnée en figure 1 [13]. Un processeur est composé de quatre blocs qui sont : les mémoires, l'unité de calcul, le réseau de routage et le contrôleur d'entrées sorties.

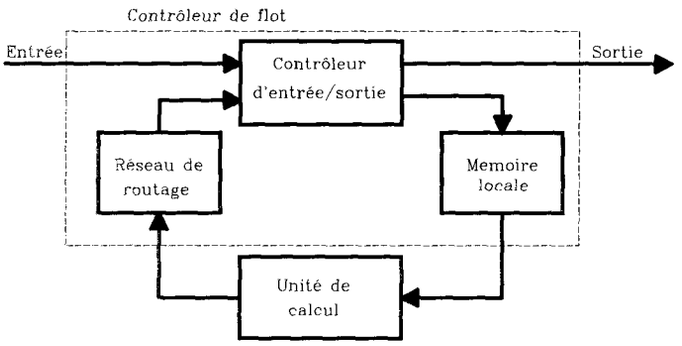


Figure 1. — Organisation d'un processeur cadencé par les données structuré en anneau.

Les mémoires contiennent, en particulier, les codes des opérations que doit réaliser l'unité de calcul. La localisation en mémoire du code correspondant au premier traitement et éventuellement d'autres informations sont associées à une donnée. Cet ensemble constitue un jeton. Après traitement par l'unité de calcul le jeton est modifié par le réseau de routage qui définit l'adresse de la prochaine instruction à effectuer.

Le processeur « data flow » NEC μPD7281 correspond à ce schéma généralisé, bien qu'il ne possède qu'une unité de traitement et que l'ordre des blocs dans l'anneau ne soit pas exactement le même.

En contrepartie des avantages apportés par le processeur « data flow » NEC μPD7281 sur le plan de la vitesse d'exécution et de la simplicité de la mise en œuvre dans une configuration multiprocesseur, il a l'inconvénient

d'avoir une programmation difficile principalement due à l'absence de langage de haut niveau. Afin d'améliorer cette situation, notre laboratoire a développé un semi-interpréteur graphique construit autour d'une base de données qui a pour but d'automatiser la transcription des graphes à flot de données utilisés pour la programmation en langage assembleur. L'addition à ce semi-interpréteur d'un algorithme d'estimation en temps à l'intérêt d'indiquer lors de l'écriture d'un programme la compatibilité avec l'application envisagée.

L'estimation du temps d'exécution d'un programme, proposée par la suite, est faite à partir du langage utilisé (graphes à flots de données dans notre cas) en tenant compte du fonctionnement interne du processeur. Cette méthode s'intègre facilement dans le semi-interpréteur en complétant la base de données. Et elle permet ainsi à l'utilisateur de savoir, lors de la programmation, si les temps d'exécution sont compatibles avec la cadence d'acquisition des données à traiter. Ceci a l'avantage de connaître dans une configuration multiprocesseur à quel niveau la consommation en temps est la plus élevée. Elle peut provenir d'un processeur ayant une tâche trop importante ou bien d'un trop grand nombre d'échanges de données sur un segment du bus. Il est ensuite possible d'optimiser le programme en temps sans avoir recours à l'usage du simulateur assez complexe à mettre en œuvre.

Dans cet article, nous décrivons dans un premier temps l'architecture et le fonctionnement interne d'un processeur μPD7281. Puis nous présentons le language de programmation en faisant le lien avec l'architecture de ce processeur. Enfin nous terminons par la description d'une méthode permettant d'évaluer les temps d'exécution à partir de programmes écrits sous la forme de graphes.

1. Description des processeurs NEC μPD7281 [6], [7]

Les processeurs « data flow » μPD7281 permettent de réaliser une machine spécialisée dans le traitement du signal et plus particulièrement dans le traitement d'images. Ces processeurs constituent des modules pouvant être liés sur un bus en anneau pour un fonctionnement en parallèle. Grâce au circuit d'interface spécialisé μPD9305 ils peuvent constituer une machine esclave d'un processeur classique du type 680 × 0 ou 80 × 86. Le même circuit μPD9305 peut réaliser l'interface avec une mémoire alimentée en données par un capteur ou par une caméra dans le cas de traitement d'images. La figure 2 représente une machine pour traitement d'images issues de caméra linéaire.

A l'initialisation de la machine, les processus que doivent effectuer les μPD7281 sont chargés par le processeur hôte. Dès lors chaque module attend les données, leurs arrivées cadencent la séquence des opérations. Ces données peuvent provenir :

— du processeur hôte,
— du capteur,
— d'un autre module de l'anneau.

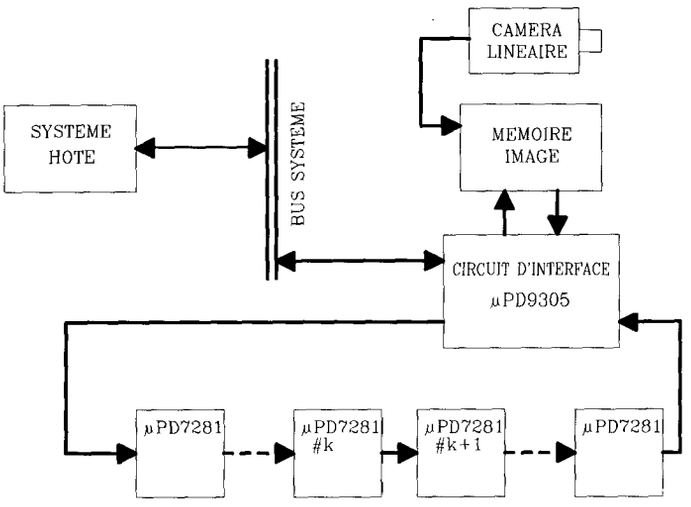
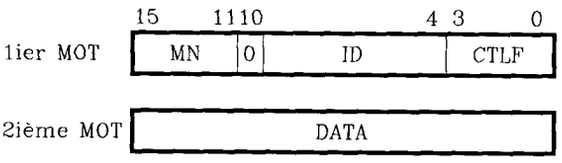


Figure 2. — Configuration multi-processeurs.

Dans ce dernier cas la donnée peut être un résultat partiel ou une demande de nouveau processus.

Chaque donnée est incluse dans un jeton composé de deux mots de 16 bits (*fig.* 3). Le deuxième mot est la donnée elle-même ; le premier comprend les identificateurs suivants :

— MN numéro du module devant recevoir le jeton,
— ID identificateur servant d'adresse interne dans le μPD7281,
— CTLF champ de contrôle précisant la nature du 2e mot, en particulier donnée de chargement du programme ou donnée à traiter.



MN Module de destination, sélecte un des processeurs
ID Identificateur servant d'adresse d'accés à LT
CTLF Champ de contrôle qui détermine la nature du jeton

Figure 3. — Format des jetons d'entrées/sorties.

Les jetons émis par le circuit d'interface ou par un processeur se déplacent sur l'anneau de module en module jusqu'à trouver le processeur concerné par le numéro MN.

Un processeur comprend un seul chemin de communication, l'anneau pipeline interne, de taille variable allant de 28 à 61 bits. Il est pourvu d'une unité arithmétique et logique 16 bits (opérateurs arithmétiques en point fixe, logiques, décalages et duplications de jetons) et d'un multiplieur rapide 16 bits, point fixe. Ces deux éléments constituent le bloc PU. Toutes les opérations effectuées par ce bloc sont câblées.

A l'intérieur du processeur le jeton progresse de bloc en bloc, en un seul mot et en un cycle pipeline ; au cours de ce voyage il subit quelques mutations (*fig.* 4 *a* et *fig.* 4 *b*).

Lorsqu'il parvient dans un processeur il est accepté ou renvoyé vers le module suivant par le contrôleur d'entrée IC, selon le numéro du module. Dans le cas d'une acceptation l'adresse de lien ID sert de pointeur pour extraire dans la table de lien LT à la fois la nouvelle adresse de lien ID' utilisée après un tour complet du pipeline et l'adresse FTA de l'instruction qui doit être exécutée dans le même tour.

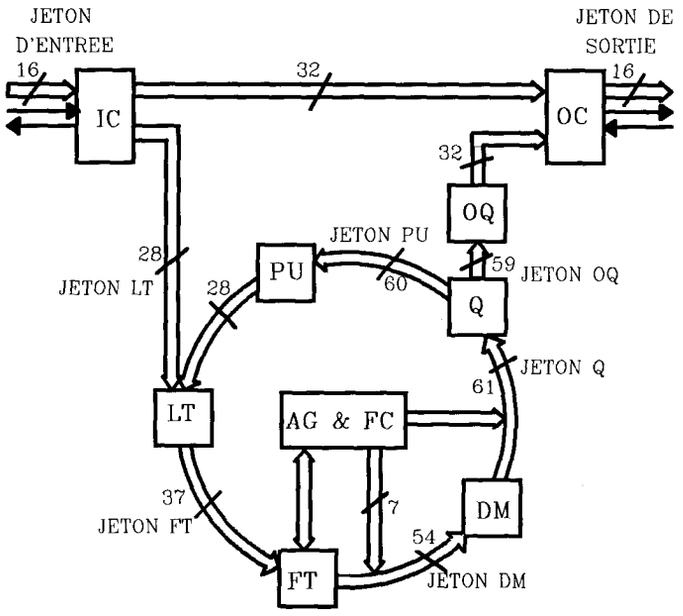
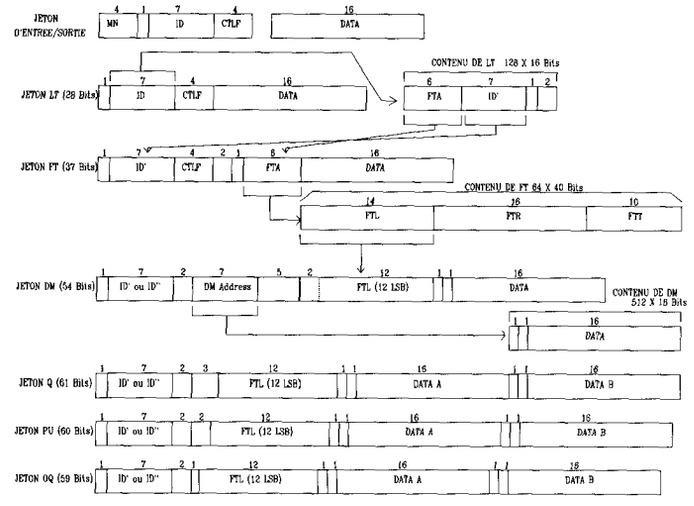


Figure 4 *a*. — Structure interne d'un processeur et format des jetons.



Jeton LT	jeton accédant à la table des liens	FTA	adresse de la table des *fonctions*
Jeton FT	jeton accédant à la tabledes fonctions	FTL	contient les codes opérations des instructions PU, GE et OUT
Jeton DM	jeton accédant à *la mémoire des données*	FTR	contient les codes opérations des instructions AG & FC
Jeton Q	jeton accédant à la file d'attente	FTT	contient les registres et les compteurs des instructions AG & FC
Jeton PU	jeton accédant à l'unité de calcul		
Jeton OQ	jeton accédant à la file d'attente de sortie		
ID	identificateur pour un jeton d'entrée ou de sortie	MN	*numéro de module*
ID'	identificateur pour un jeton quittant LT la 1ère fois	CTLF	champ de contrôle
ID"	*identificateur pour un jeton* quittant LT la 2ème fois	DATA	champ de données 16 bits

Figure 4 *b*. — Modification des jetons entre chaque bloc *du processeur*.

Chaque fois qu'un jeton est accepté par le contrôleur d'entrée, l'unité PU est bloquée pour libérer l'accès à la table LT.

Toutes les instructions ont été mémorisées lors de l'initialisation dans la table de fonction FT. Cette table est une mémoire RAM constituée de 64 mots de 40 bits pointés par le champ FTA.

A ce niveau le jeton reçoit un nouveau champ contenant le code opération de l'instruction qui sera exécutée par l'unité arithmétique et logique PU.

Si une instruction AG ou FC est spécifiée, le bloc générateur d'adresses et contrôleur de flot AG&FC déterminent une adresse de la mémoire DM ou modifient la valeur du champ ID' du jeton, voire les deux.

La mémoire DM, constituée de 512 mots de 18 bits peut servir de mémoire locale ou permet d'associer au jeton un second opérande précédemment mémorisé.

Les jetons se déplaçant sur l'anneau interne subissent une opération à chaque passage dans l'unité arithmétique et logique PU. Suivant la nature de l'opération, le résultat peut modifier dans le jeton traité : le champ de données, le champ ID ou bien les deux à la fois. Dans le cas d'une instruction de type PU le champ de données est modifié par le résultat et le jeton peut avoir deux destinations différentes, c'est-à-dire deux sorties. Le deuxième jeton est alors une copie du premier avec un champ ID' incrémenté par rapport à ID. Il existe aussi des instructions spécialisées de duplication, instructions GE, pouvant générer un jeton vers seize destinations différentes ou seize jetons vers la même destination, chaque destination correspond à une valeur du champ ID.

Le temps d'exécution d'une instruction par l'unité PU diffère en fonction du nombre de jetons qu'elle doit générer, un cycle pipeline étant nécessaire par jeton émis. La file d'attente Q est un FIFO de 48 mots qui contrôle l'entrée des jetons dans l'unité arithmétique PU. D'autre part, ce FIFO comporte un mécanisme de régulation « throttling mechanism » [13] consistant à favoriser la génération de jetons (instructions de type GE). Pour cela, deux suites chronologiques de jetons à traiter sont construites en classifiant ces jetons suivant leur nature GE ou PU et en les gérant de la façon suivante :

— lorsque le pipeline est sous-employé, les jetons des deux suites chronologiques sont dirigés alternativement vers l'unité arithmétique, en favorisant ainsi l'exécution des instructions de type GE, afin d'augmenter le taux d'occupation de l'anneau interne ;

— pour éviter la saturation de la file d'attente Q, l'exécution des instructions de type GE est bloquée lorsque le nombre de jetons en attente dans cette file dvient trop important. Ceci permet un contrôle fin du flot de données évitant ainsi une expansion trop importante des données à traiter.

Après avoir traversé l'unité PU, le jeton arrive à nouveau au bloc LT mais cette fois avec l'identificateur ID' pour effectuer un nouveau tour de pipeline, être détruit ou être

envoyé vers le contrôleur de sortie OC. Il faut sept cycles pipeline à un jeton pour parcourir l'anneau interne, un cycle par bloc, excepté Q et PU qui en nécessitent chacun deux. Si tous les blocs sont actifs à la fois, l'unité de calcul fait une opération à chaque cycle de pipeline réduisant ainsi le temps apparent de traitement d'une donnée.

Si l'opération est dyadique, une instruction particulière (QUEUE) doit être utilisée pour placer temporairement le premier opérande dans la mémoire DM en attendant que le second arrive. A cet instant l'opération est validée et est dirigée vers le PU. La profondeur maximale de cette file d'attente est de 16 niveaux, certaines précautions doivent être prises lors de la programmation pour ne pas dépasser cette limite. Dans le cas d'une opération monoadique, le jeton se dirige directement vers le PU. Un seul jeton peut être traité à la fois par le PU et 48 autres peuvent être en attente de traitement dans la file Q (16 de type GE et 32 de type PU).

Un buffer de sortie OQ d'une capacité de 8 mots permet de gérer les conflits lors des accès vers l'anneau externe, les jetons venant du contrôleur d'entrée sont prioritaires.

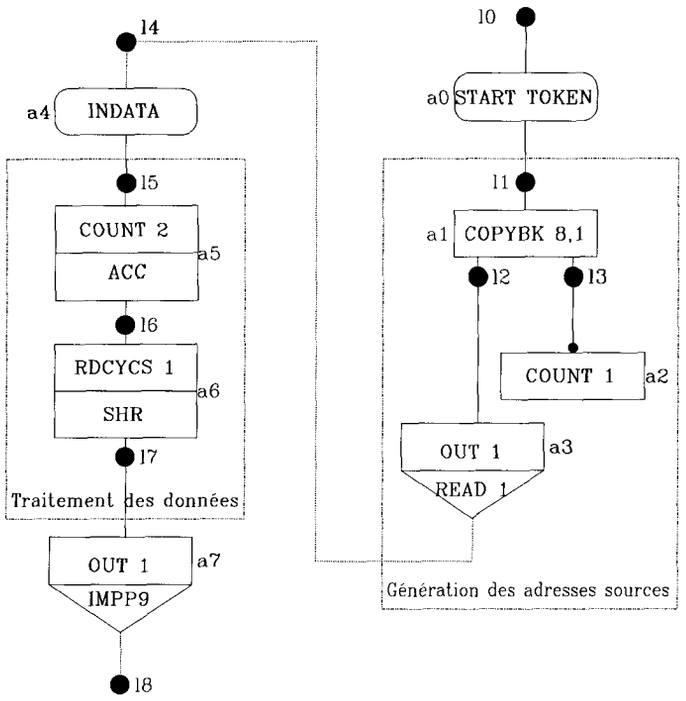


Figure 5. — Graphe de flot pour µPD7281.

2. Programmation des processeurs [8]

La programmation des processeurs cadencés par les données est facilitée par l'utilisation de graphes, nommés graphes de flot. Les processeurs NEC ont un graphisme particulier, tenant compte de leur jeu d'instruction. Un assembleur, spécifique aux processeurs NEC, permet d'établir du code à partir d'un langage source déduit des graphes manuellement. La mise au point des programmes est aidée par l'utilisation d'un simulateur permettant de visualiser la circulation des jetons à l'intérieur des processeurs et donnant ainsi la possibilité d'évaluer les temps d'exécution après les phases d'assemblage et de simulation. Ce simulateur est un modèle logiciel du processeur µPD7281. Il utilise le code objet généré par l'assembleur et il simule l'évolution de l'état interne du processeur cycle par cycle.

Les graphes sont composés d'arcs et de nœuds (*fig.* 5). Les arcs sont les voies de communication que doit emprunter le flot de données tandis que les nœuds spécifient le type d'opération que devra effectuer le µPD7281. Les arcs représentent les identificateurs ID adressant la table de liens alors que les nœuds sont les entrées dans la table FT.

Après avoir défini l'algorithme, la principale difficulté lors de la programmation est l'établissement du graphe de flot de données. En plus de l'algorithmique, certaines instructions demandent à être synchronisées entre elles pour contrôler le flot de données. Ce problème se présente le plus souvent dans les deux cas suivants.

Dans le premier cas, le graphe est composé de plusieurs sous-graphes, par exemple deux sous-graphes appelés *f* et *g* (*fig.* 6 *a* ou 6 *b*). Si *f* alimente *g* en données plus rapidement que *g* ne peut les consommer (*fig.* 6 *a*), alors une accumulation de jetons va apparaître à l'entrée de *g* qui risque de produire un dépassement de la capacité des files d'attentes. La génération de données par le sous-

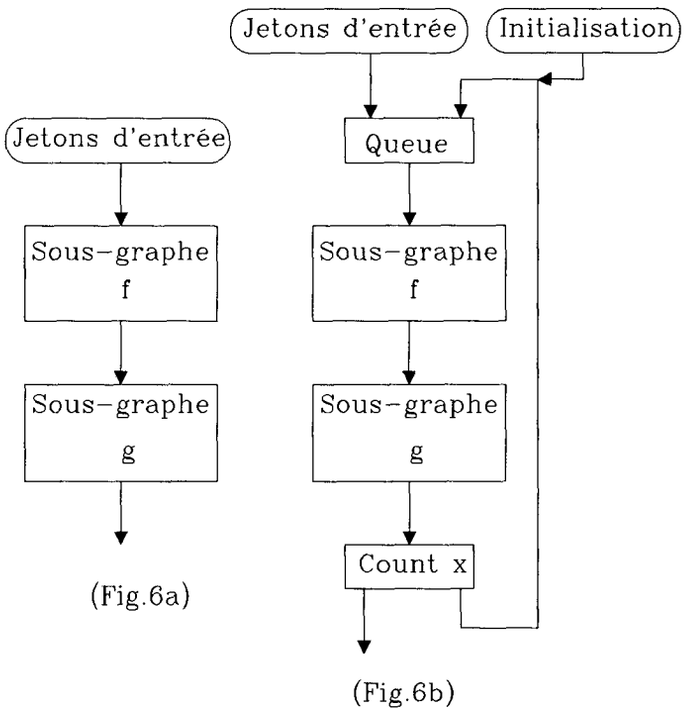


Figure 6. — Problèmes de synchronisation.

graphe *f* doit être contrôlée par *g* (*fig.* 6 *b*) ; d'où la nécessité de la part de *g* de créer des jetons de synchronisations qui accorderont la production du sous-graphe *f* avec la capacité de traitement de *g*.

Le deuxième cas correspond aux graphes cycliques (*fig.* 7) (réentrance, construction itérative, boucle), ce type de graphe peut conduire à un comportement indéterminé si certaines précautions ne sont pas prises. Des instructions de verrouillage devront être utilisées car on ne connaît pas a priori l'ordre exact dans lequel les instructions seront effectuées contrairement à un processeur de type séquentiel. Cete méthode est simple mais elle a pour inconvénient de vider le pipeline et de dégrader les performances du processeur. Un exemple fréquemment rencontré est décrit dans [10].

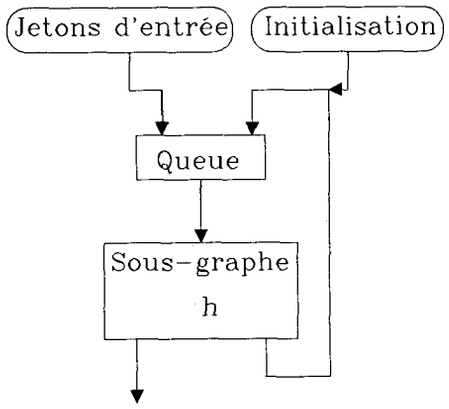


Figure 7. — Construction d'une boucle suivant la méthode de verrouillage.

CLASSIFICATION DES INSTRUCTIONS

Il existe quatre classes d'instructions pour la programmation des μPD7281 :

— instructions AG&FC, elles assurent le contrôle du cheminement des données et la génération d'adresses pour la mémoire DM ;

— instructions GE, elles génèrent plusieurs jetons à partir d'un seul, duplication de jetons ou génération d'adresses pour la mémoire externe ;

— instructions OUT, elles dirigent les données traitées par le processeur vers le bus externe ;

— instructions PU, elles réalisent les instructions arithmétiques et logiques.

Les graphes à flot de données destinés aux μPD7281 sont une présentation exacte des programmes en langage assembleur mis sous une forme plus explicite à l'utilisateur, chaque nœud est une instruction du processeur. Ils mettent en évidence d'une part la dépendance des données et d'autre part la concurrence des opérations à l'intérieur du processeur. La notation graphique utilisée est spécifique au μPD7281 [6]. Chaque nœud est associé à un des mnémoniques d'instructions, accompagné de paramètres qui définissent complètement la nature de l'instruction. Les entrées et les sorties externes au processeur sont indiquées par un graphisme particulier. Les caractéristiques des entrées et des sorties de nœuds se distinguent par un symbole spécifique.

Certains nœuds associent deux instructions ; l'une d'entre elles, étant obligatoirement de type AG&FC, modifie la fonction de la deuxième instruction. Par la suite nous considérons que ces nœuds auront la classe de l'instruction associée PU ou GE. Di une instruction AG&FC est utilisée seule, elle sera prise en compte comme une instruction PU en supposant qu'elle soit associée à une instruction NOP (no operation).

Actuellement les graphes de flots doivent être traduit en langage source manuellement et transformé en code machine par l'assembleur. Rappellons qu'un interpréteur graphique est en cours de développement dans notre laboratoire pour automatiser cette tâche. Le language source comprend quatre sections de déclarations (tableau 1), ces sections sont :

— la définition des liens d'entrée et de sortie externes du graphe,

— la table des liens associant à chaque nœud les adresses des liens d'entrée ID et les adresses des liens de sortie ID',

— la table des fonctions associant à chaque nœud une instruction accompagnée de ses paramètres,

— le contenu de la mémoire interne pouvant être des constantes ou des zones réservées.

L'assembleur transforme ces déclarations sous forme de jetons qui sont utilisés pour charger les tables LT, FT et la mémoire DM du processeur avant l'arrivée des données.

```
/*  ENTREES-SORTIES    */

INPUT L0 ,L4                                          ;
OUTPUT L8                                             ;

/*    TABLE LT      */

LINK L2, L3              = A1  (L0)                   ;
LINK                     = A2  ( , L3)                ;
LINK L4                  = A3  (L2)                   ;
LINK L6                  = A5  (L4)                   ;
LINK L7                  = A6  (L6)                   ;
LINK L8                  = A7  (L7)                   ;

/*    TABLE FT      */

FUNC A1                  = COPYBK (16, 1)             ;
FUNC A2                  = COUNT (1)                  ;
FUNC A3                  = OUT1 (READ1, 01H)          ;
FUNC A5                  = ACC, COUNT (2)             ;
FUNC A6                  = SHR, RDCYCS (ONE, 1)       ;
FUNC A7                  = OUT1 (IMPP9,00H)           ;

/*    MEMOIRE DM    */

MEMORY ONE               = 0001H                      ;
```

Tableau 1. — Représentation en langage source du programme de la figure 5. On trouve trois sections de déclaration LT, FT et DM qui correspondent aux trois blocs où le programme est chargé. On peut remarquer que la notation graphique est équivalente au langage source.

Nous utilisons le lien étroit existant entre l'organisation interne du processeur et les graphes pour évaluer le temps d'exécution d'un programme en nombre de cycles machine, pour cela il faut prendre en compte le fonctionnement interne du processeur qui est cadencé par une horloge. Bien que le fonctionnement du processeur soit synchrone, les programmes sont par contre de nature asynchrone.

3. Répartition des tâches entre les processeurs

Deux approches sont possibles lorsqu'on développe des programmes pour les μPD7281 dans une configuration multiprocesseur :

— soit partager les données à traiter entre plusieurs processeurs chargés avec le même programme (traitement simultané de plusieurs données à la fois),

— soit scinder le programme entre plusieurs processeurs (traitement pipeline d'une donnée).

De fait, nous serons amenés à un panachage des deux approches.

Les performances de cette architecture dépendent de la répartition des tâche entre les processeurs. Si la répartition est inégale, des processeurs sont en attente de données et la notion de parallélisme tend à disparaître. Ainsi il est nécessaire que la distribution des tâches entre les processeurs soit équitable pour que les performances optimales d'une configuration multiprocesseur soient atteintes.

La répartition des tâches n'est pas seulement liée à des problèmes de temps mais aussi à la capacité des mémoires internes LT et FT du processeur.

Cette contrainte ne laisse guère de possibilité quand la taille d'un programme est trop importante pour être contenue dans un seul processeur. Il faut le partager entre plusieurs processeurs de façon à ce qu'un étage ne sature pas le suivant plus lent par un envoi trop rapide de données. Sinon il faudra synchroniser ces étages entre eux pour éviter la saturation d'un processeur.

Une répartition judicieuse ne peut être obtenue que si on est capable d'estimer les temps. Actuellement la méthode consiste à simuler le programme et en fonction des résultats, réorganiser la distribution des tâches ou synchroniser les échanges de données pour éviter les dépassements de capacité mémoire.

4. Estimation du temps d'exécution d'un graphe à flot de données pour μPD7281

Nous allons d'abord rappeler le formalisme des graphes à flot de données introduit par Kavi [4], [5] issu du modèle original présenté par Dennis [11] en l'adoptant à l'architecture du processeur μPD7281.

Un graphe à flot de données (*fig.* 5) est un graphe bipartite étiqueté composé de deux types de nœuds appelés les acteurs et les liens.

(1) $$G = \langle A \cup L, E \rangle$$

$A = \{a_1, a_2, ..., a_n\}$ est l'ensemble des acteurs représentant les opérations à effectuer.

$L = \{\ell_1, \ell_2, ..., \ell_m\}$ est l'ensemble des liens considérés comme des places porteuses de données.

$E \subseteq (A \times L) \cup (L \times A)$ est l'ensemble des arcs (ou frontières), ils sont les voies de communications entre les nœuds.

On définit aussi deux sous-ensembles S et T inclus dans L représentant respectivement les entrées et les sorties externes du graphe de flot.

Dans l'ensemble A, deux sous-ensembles A_{GE} et A_{PU} sont définis suivant la classe d'instruction auquel appartient leurs éléments a_j, $j = 1, ..., n$:

$a_j \in A_{GE}$ si l'acteur a_j est une instruction de type GE .

$a_j \in A_{PU}$ si l'acteur a_j n'est pas une instruction de type GE

Un troisième sous-ensemble A_{IN} représente l'entrée des données dans le processeur. les acteurs appartenant à A_{IN} ne sont pas des instructions du μPD7281 mais ils agissent sur le PU pour pouvoir introduire des données dans l'anneau interne. En effet pour introduire une donnée, le PU doit être bloqué pour libérer l'accès à LT.

Les instructions de la classe $OUT(A_{OUT})$ ne sont pas prises en compte car elles ne sont pas traitées par l'unité du processeur, elles indiquent la destination des données quittant l'anneau interne.

L'ensemble des liens d'entrée et l'ensemble des liens de sortie d'un acteur a_j sont représentés par $I(a_j)$ et $O(a_j)$ tel que :

(2) $$I(a_j) := \{\ell \in L ;\ (\ell, a_j) \in E\} ,$$

(3) $$O(a_j) := \{\ell \in L ;\ (a_j; \ell) \in E\} .$$

Le nombre de liens d'entrée est noté par $|I(a_j)|$ et celui de sortie par $|O(a_j)|$.

De la même façon l'ensemble des acteurs d'entrée et l'ensemble des acteurs de sortie d'un lien ℓ_k, $k = 1, ..., m$ sont représentés par $I(\ell_k)$ et $O(\ell_k)$ tel que :

(4) $$I(\ell_k) := \{a \in A ;\ (a, \ell_k) \in E\} ,$$

(5) $$O(\ell_k) := \{a \in A ;\ (\ell_k, a) \in E\} .$$

Le nombre d'acteurs d'entrée est noté par $|I(\ell_k)|$ et celui de sortie par $|O(\ell_k)|$.

L'architecture du processeur et le format des jetons imposent des limites aux valeurs que peuvent prendre $|I(a_j)|$, $|O(a_j)|$, $|I(\ell_k)|$ et $|O(\ell_k)|$.

Pour tous les acteurs $|I(a)|$ a seulement deux valeurs possibles car la mémoire DM permet d'associer deux jetons au plus :

— $|I(a)| = 1$, cas d'une instruction monoadique (champ DATA A ou DATA B (*fig.* 4),

— $|I(a)| = 2$, cas d'une instruction dyadique (champs DATA A et DATA B).

Un acteur peut avoir jusqu'à seize destinations différentes (16 jetons ayant chacun un champ ID différent) alors

$$16 \geqslant |O(a)| \geqslant 0 .$$

Le cas particulier $|O(a)| = 0$ correspond à une instruction de type AG&FC utilisée seule, par exemple pour écrire une donnée dans la mémoire interne DM, le jeton entrant est alors détruit systématiquement.

Dans le cas des liens, $|I(\ell)| \geqslant 0$

— $|I(\ell)| = 0$ si $\ell \in S$ (entrée externe du graphe).

— $I(\ell) = 1$ dans le cas général, un lien n'a qu'un seul acteur d'entrée.

— $|I(\ell)| > 1$ le lien a plusieurs acteurs d'entrée. Le processeur ne possèdant qu'une seule unité de traitement implique que seulement un des acteurs du graphe est actif et il ne peut donc pas se produire de conflit à l'entrée d'un lien quand $|I(\ell)| > 1$ puisqu'au plus un jeton est généré à chaque cycle pipeline.

— $|O(\ell)| = 0$ si $\ell \in T$ sortie externe du graphe,

— $|O(\ell)| = 1$ dans les autres cas.

Cette limitation oblige, si on désire diriger la même donnée vers plusieurs acteurs, de la dupliquer par un opérateur.

Le marquage d'un graphe de flot spécifie l'absence ou la présence de jetons dans les liens, il est représenté par la fonction :

$$(6) \qquad M_i : L \rightarrow \{0, 1, ..., h\} .$$

A chaque lien ℓ_k, est associé un entier $M_i(\ell_k)$ qui représente le nombre de jetons en attente tel que :

$$\sum_{k = 0, ..., m} M_i(\ell_k) \leqslant h ,$$

h étant la limite imposée par la taille des files d'attente internes aux processeurs. M_i est utilisé au sens de marque pour l'ensemble des places $[M_i(\ell_1), M_i(\ell_2), ..., M_i(\ell_m)]$.

On distinguera deux marquages particuliers :

— M_0 marquage initial défini par l'utilisateur

$$M_0(\ell) \geqslant 0 \text{ si } \ell \in S \text{ et } M_0(\ell) = 0 \text{ si } \ell \notin S .$$

— M_t marquage terminal

$$M(\ell) \geqslant 0 \text{ si } \ell \in T \text{ et } M(\ell) = 0 \text{ si } \ell \notin T .$$

Ce marquage est obtenu quand tous les jetons générés à partir du marquage initial ont été consommé. Si le marquage du graphe n'évolue plus et que tous les jetons n'ont pas atteint le marquage final, alors le graphe contient des interblocages qui conduisent à un mauvais déroulement du programme, ce cas n'est pas pris en considération ici.

L'indice i représente le nombre de cycles pipeline entre le marquage M_0 et le marquage M_i.

On associe à chaque acteur deux ensembles de marques minimales, un en entrée et l'autre en sortie, de même dimension que M_i :

$$F(a_j) \quad \text{et} \quad H(a_j) ,$$

le premier définit le marquage pouvant activer l'acteur a_j et le deuxième désigne le lien recevant un jeton.

$$(7) \qquad F(a_j) : I(a_j) \rightarrow \{0, 1\} \text{ et } L - I(a_j) \rightarrow \{0\} .$$

Suivant le type de l'acteur a_j, unaire ou binaire, un ou deux éléments du marquage $F(a_j)$ prennent la valeur 1, ces derniers représentent les données consommées par l'acteur a_j.

$$(8) \qquad H(a_j) : O(a_j) \rightarrow \{0, 1\} \text{ et } L - O(a_j) \rightarrow \{0\} .$$

Cette marque n'a qu'un élément égal à 1 correspondant au lien qui reçoit la donnée au cycle pipeline i. On associe une marque par jeton généré, l'ordre des marques est défini par l'instruction.

Le graphe de flot de données contient les informations nécessaires qui déterminent les éléments de $F(a_j)$ et $H(a_j)$ pour chaque instruction a_j. L'ensemble des marques $F(a_j)$ et $H(a_j)$ sont déduits des instructions définies par le constructeur [6].

Il existe aussi des instructions conditionnelles, l'utilisateur pourra intervenir pour spécifier la fréquence de sélection des sorties, quand elle est connue à l'avance, afin que ces instructions soient traitées comme les autres. Dans le cas où ces instructions dépendent des données à traiter et que la fréquence de sélection des sorties est inconnue, il faut faire appel à des méthodes probabilistes [5] qui n'ont pas été mises en œuvre ici. Cette restriction reste cependant peu contraignante, car l'emploi de ce type d'instruction apparaît le plus souvent dans les traitements de niveau supérieur, lors de la prise de décision par exemple, où le nombre de données à traiter a considérablement diminué.

Le sens des opérations et le contenu des jetons n'ont pas d'intérêt ici car la connaissance des $F(a_j)$ et des $H(a_j)$ est suffisante pour déterminer la propagation des données à travers le graphe.

L'activation d'un acteur a_j a pour conséquence de modifier le marquage du graphe noté par :

$$M_i \xrightarrow{a_j} M_{i'} , \quad \text{avec} \quad i' > i .$$

Ceci peut être généralisé à une séquence

$$M_i \xrightarrow{a_1} M_{i_1} \xrightarrow{a_2} M_{i_2} \xrightarrow{a_3} \cdots \xrightarrow{a_p} M_{i_p}$$

notée par :

$$(9) \qquad M_i \xrightarrow{a} M_{i_p} \quad \text{où} \quad a = a_1 a_2 a_3 ... a_p$$

a est la séquence des acteurs dans l'ordre dans lequel ils ont été validé et M^a est l'ensemble des marquages obtenus durant la séquence a

$$M^a = M_i M_{i_1} M_{i_2} ... M_{i_p} .$$

Les instructions du μPD7281 peuvent générer un ou plusieurs jetons vers un ou plusieurs liens à partir d'un seul jeton entrant. Le temps d'exécution de l'instruction sera proportionnel au nombre de jetons émis (un cycle pipeline

par jeton). Nous allons considérer que chaque acteur est une séquence élémentaire et indivisible définie par le constructeur qui modifie le marquage à chaque cycle pipeline. L'évolution du marquage provoquée par l'acteur a_j :

$$M_i \xrightarrow{a_j} M_{i'}$$

peut se noter en tenant compte de chaque cycle de la machine :

$$(10) \qquad M_i \xrightarrow{cp_1} M_{i+1} \xrightarrow{cp_2} M_{i+2} \xrightarrow{cp_3} \cdots \xrightarrow{cp_q} M_{i+q}$$

où q représente le nombre de cycles nécessaires à l'instruction et

$$M^{a_j} = M_1 M_2 \ldots M_q$$

les différents marquages obtenus pendant le déroulement de a_j dont le nouvel état se traduit par :

$$(11) \quad M_{i'} = M_i - F_1(a_j) + H_1(a_j) + H_2(a_j) + \cdots + H_q(a_j)$$

avec $$i' = i + q\,.$$

Un acteur a_j n'appartenant pas à A_{IN} est valide dans un marquage M_i si tous les marquages compris entre M_i et $M_{i'}$ respectent la condition suivante :

$$M \geqslant F(a_j) \text{ pour chaque } \ell \in F(a_j)$$

avec $i - i' \geqslant tp$, tp représentant le nombre nécessaire de cycles pour qu'un jeton parcours l'anneau pipeline interne.

En effet il y a deux conditions pour qu'un acteur soit valide : il faut que les liens d'entrée possèdent au moins chacun un jeton. De plus il faut que ce jeton soit présent depuis un nombre de cycles égal au nombre d'étages de l'anneau pipeline interne du processeur (nombre de cycles pour qu'une donnée parvienne d'un acteur à un autre). Dans notre cas, il faut sept cycles pipeline notés par tp pour remplir cette dernière condition. Si aucun acteur n'est valide dans un marquage M_i, il faut attendre que la condition sur le nombre de cycles pipeline soit remplie. Pendant ces cycles d'attente, le marquage du graphe n'évolue pas.

La séquence $M_i \xrightarrow{a} M_{ip}$ devient en tenant compte des cycles pipeline d'attente :

$$(12) \qquad M_{i_0} \xrightarrow{a_1} M_{i'_1} \rightarrow M_{i_1} \xrightarrow{a_2} M_{i'_2} \rightarrow M_{i_2} \xrightarrow{a_3} \cdots \xrightarrow{a_p} M_{i_p}$$

avec $$M_{i_k} = M_{i'_k}\,,$$

$$tp \geqslant (i'_k - i_k) \geqslant 0$$

et $$i_{(k+1)'} - i_k = q_k\,.$$

q_k étant le nombre de jetons engendré par l'acteur a_k.

Quant aux acteurs appartenant à A_{IN}, ils recopient les jetons présents sur le lien d'entrée vers le lien de sortie un à un tous les deux cycles pipelines.

Lorsque le processeur a besoin de données situées dans la mémoire commune, il doit sortir une adresse sur le bus externe accompagnée d'un identificateur pour pouvoir récupérer la donnée lue en mémoire. Ceci est réalisé grâce à un jeton qui est placé sous la forme de deux mots de seize bits sur le bus externe. Il faut sept cycles pour que le jeton accède au bus externe, un cycle par processeur rencontré et deux cycles pour l'accès à la mémoire. Le temps que met le processeur pour émettre une donnée vers l'extérieur et que le résultat soit disponible peut aussi être évalué en nombre de cycles pipeline. Au niveau du graphe, le numéro de module recevant le jeton est spécifié par l'instruction de type OUT alors que sa destination est représentée par un arc en trait pointillé joignant le lien de sortie.

C'est l'étude de la séquence $M_0 \xrightarrow{a} M_t$ qui va nous permettre d'évaluer le temps de calcul correspondant à un graphe de flot. Nous allons maintenant établir les règles qui régissent l'ordre des acteurs de cette séquence en fonction de l'architecture du processeur μPD7281.

Règle 1 : Un acteur ne peut pas être interrompu.

Règle 2 : Les acteurs appartenant à l'ensemble A_{IN} sont prioritaires par rapport aux autres.

Règle 3 : Si plusieurs acteurs sont valides simultanément, on alterne un acteur a_{GE} puis un acteur a_{PU}.

Règle 4 : Si plusieurs acteurs d'un même type sont valides simultanément alors on choisit celui qui était valide le premier par ordre chronologique.

5. Exemple

Nous allons estimer le temps d'exécution d'un programme (*fig. 5*) qui lit huit octets en mémoire et réduit le nombre de ces octets par deux en faisant une moyenne puis qui transmet le résultat au processeur suivant.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$F_1(a_1)$	0	1	0	0	0	0	0	0
$H_1(a_1)$	0	0	1	0	0	0	0	0
$H_2(a_1)$	0	0	1	0	0	0	0	0
:	:	:	:	:	:	:	:	:
$H_8(a_1)$	0	0	1	0	0	0	0	0
$H_9(a_1)$	0	0	0	1	0	0	0	0
$F_1(a_5)$	0	0	0	0	0	1	0	0
$H_1(a_5)$	0	0	0	0	0	0	0	0
$H_{1'}(a_5)$	0	0	0	0	0	0	1	0
$F_1(a_6)$	0	0	0	0	0	0	1	0
$H_1(a_6)$	0	0	0	0	0	0	0	1

Tableau 2. — Marquage minimal des acteurs de la figure 5.

Explication du rôle des acteurs :

a_0 : entrée de l'adresse source dans le processeur.
a_1 : génère huit adresses incrémentées de un vers ℓ_2 et duplique la dernière dans ℓ_3.
a_2 : détruit l'adresse contenue dans ℓ_3.
a_3 : dirige les adresses vers la mémoire.
a_4 : entrée des données lues en mémoire dans le processeur.
a_5 : somme cumulative de deux données.
a_6 : décalage à droite de un bit.
a_7 : envoi du résultat vers le processeur suivant.

$A := \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$

$A_{GE} := \{a_1\}$
$A_{PU} := \{a_5, a_6\}$
$A_{IN} := \{a_0, a_4\}$
$A_{OUT} := \{a_3, a_7\}$

$L := \{\ell_0, \ell_1, \ell_2, \ell_3, ..., \ell_8\}$
$S := \{\ell_0, \ell_4\}$
$T := \{\ell_7\}$

$I(a_0) := \{\ell_0\}$ $\quad I(a_1) := \{\ell_1\}$
$I(a_4) := \{\ell_4\}$ $\quad I(a_5) := \{\ell_5\}$
$O(a_0) := \{\ell_1\}$ $\quad O(a_1) := \{\ell_2, \ell_3\}$
$O(a_4) := \{\ell_5\}$ $\quad O(a_5) := \{\ell_6\}$

$I(a_2) := \{\ell_3\}$ $\quad I(a_3) := \{\ell_2\}$
$I(a_6) := \{\ell_6\}$ $\quad I(a_7) := \{\ell_7\}$
$O(a_2) := \{\varnothing\}$ $\quad O(a_3) := \{\ell_4\}$
$O(a_6) := \{\ell_7\}$ $\quad O(a_7) := \{\ell_8\}$

$I(\ell_0) := \{\varnothing\}$ $\quad I(\ell_1) := \{a_0\}$
$I(\ell_4) := \{a_3\}$ $\quad I(\ell_5) := \{a_4\}$
$O(\ell_0) := \{a_0\}$ $\quad O(\ell_1) := \{a_1\}$
$O(\ell_4) := \{a_4\}$ $\quad O(\ell_5) := \{a_5\}$

$I(\ell_2) := \{a_1\}$ $\quad I(\ell_3) := \{a_1\}$
$I(\ell_6) := \{a_5\}$ $\quad I(\ell_7) := \{a_6\}$
$O(\ell_2) := \{a_3\}$ $\quad O(\ell_3) := \{a_2\}$
$O(\ell_6) := \{a_6\}$ $\quad O(\ell_7) := \{a_7\}$

Le tableau 3 représente l'évolution des marquages du graphe de flot de la figure 5, les marques minimales associées aux acteurs figurent dans le tableau 2. Les marquages manquant correspondant à des cycles d'attentes du processeur. Ici il faut 53 cycles pour traiter les huit données soit 10,6 µs (un cycle pipeline cp = 200 ns), alors

		l_0	l_1	l_2	l_3	l_4	l_5	l_6	l_7
	M_0	1	0	0	0	0	0	0	0
a_0	M_2	0	1	0	0	0	0	0	0
a_1	M_9	0	0	1	0	0	0	0	0
	M_{10}	0	0	2	0	0	0	0	0
	:	:	:	:	:	:	:	:	:
	M_{16}	0	0	8	0	0	0	0	0
	M_{17}	0	0	8	1	0	0	0	0
	M_{23}	0	0	0	0	8	0	0	0
a_4	M_{25}	0	0	0	0	7	1	0	0
a_4	M_{27}	0	0	0	0	6	2	0	0
a_4	M_{29}	0	0	0	0	5	3	0	0
a_4	M_{31}	0	0	0	0	4	4	0	0
a_5	M_{32}	0	0	0	0	4	3	0	0
a_4	M_{33}	0	0	0	0	3	4	0	0
a_5	M_{34}	0	0	0	0	3	3	1	0
a_4	M_{35}	0	0	0	0	2	4	1	0
a_5	M_{36}	0	0	0	0	2	3	1	0
a_4	M_{37}	0	0	0	0	1	4	1	0
a_5	M_{38}	0	0	0	0	1	3	2	0
a_4	M_{39}	0	0	0	0	0	4	2	0
a_5	M_{40}	0	0	0	0	0	3	2	0
a_6	M_{41}	0	0	0	0	0	3	1	1
a_5	M_{42}	0	0	0	0	0	2	2	1
a_5	M_{44}	0	0	0	0	0	1	2	1
a_6	M_{45}	0	0	0	0	0	1	1	2
a_5	M_{46}	0	0	0	0	0	0	2	2
a_6	M_{51}	0	0	0	0	0	0	1	3
a_6	M_{53}	0	0	0	0	0	0	0	4

Tableau 3. — Évolution des marquages du graphe de flot de données de la figure 5. Les marquages d'attente ne sont pas représentés.

que le simulateur calcule 55 cycles, il prend en compte deux cycles supplémentaires pour introduire le premier jeton du marquage initial dans le bus externe.

Le marquage M_{23} correspond à l'arrivée des données lues en mémoire dans le processeur 14 cycles après la génération de la première adresse. Bien que les données arrivent séquentiellement, nous les introduisons par bloc pour simplifier l'écriture des marquages. Cette simplification n'a pas d'incidence sur l'évaluation du temps, elle est compensée par les conditions de fonctionnement des acteurs de A_{IN}.

A partir de la séquence

$$a = a_0\, a_1\, a_4\, a_4\, a_4\, a_4\, a_5 \ldots a_5\, a_6\, a_6\,,$$

il est possible de déduire le nombre de jetons traités par le processeur en faisant la somme du nombre de cycles utilisés par les acteurs de la séquence a :

$$N = q_0 + q_1 + q_4 + q_4 + \cdots + q_5 + q_5 + q_6$$
$$N = 1 + 9 + 1 + 1 + \cdots + 1 + 1 + 1$$
$$N = 30 \text{ jetons}\,.$$

In évalue le taux d'occupation de l'anneau interne représentant l'activité du processeur par le rapport :
nombre de cycles générant un jeton divisé par le nombre de cycles total soit :

$$30/53 = 0{,}57 .$$

Maintenant ce programme est modifié (*fig.* 8) de façon à réduire une ligne image de 512 à 256 pixels. Pour cela la génération d'adresses est changée de façon à créer 32 blocs de 16 adresses adjacentes. On obtient ainsi 2 756 cycles (0,55 ms) avec un taux d'occupation du pipeline de 0,86. La valeur de ce rapport augmente par rapport au précédent parce que de nouvelles adresses sont générées alors que le traitement des données précédentes n'est pas terminé si bien que l'anneau pipeline est mieux occupé.

6. Évaluation des temps de transfert de données sur le bus en anneau externe

Le temps d'exécution d'un programme est lié à deux facteurs dont un devient souvent prépondérant ; ce sont :
— le nombre d'accès total à la mémoire commune et le nombre de transferts de données sur l'anneau externe,
— le temps de calcul des processeurs.

Nous allons donc évaluer le nombre d'accès à la mémoire et le nombre d'échanges entre les processeurs afin d'estimer le temps d'exécution minimal d'un programme dans une configuration multiprocesseur.

Un réseau d'interconnexion peut être décrit par un ensemble R de fonctions d'interconnexion [12]. Chaque fonction d'interconnexion est une bijection (permutation) de l'ensemble $S:=\{0,1,2, ..., N-1\}$ des adresses des processeurs. Les fonctions d'interconnexions représentent les transferts de données interprocesseurs. Quand une fonction d'interconnexion notée par r est réalisée, PE_k, $k = 0, ..., N-1$ envoie une donnée à $PE_{r(k)}$; PE_k représentant l'élément processeur ayant pour adresse k. Pour transmettre une donnée entre deux processeurs qui ne sont pas directement connectés, la donnée doit transiter par des processeurs intermédiaires créant ainsi une séquence de transfert à travers le réseau.

Chaque processeur PE_k possède un contrôleur d'entrée IC_k et un contrôleur de sortie OC_k. Lorsqu'une donnée est introduite, elle est recopiée de IC_k vers OC_k si elle n'est pas destinée à PE_k. Les jetons internes devant accéder au bus externe sont dirigés également vers OC_k. Dans le cas d'un bus unidirectionnel en anneau un processeur PE_k est seulement autorisé à transmettre une donnée vers l'élément PE_{k+1}.

Le temps de transfert pris en compte est celui correspondant au plus grand nombre d'échange de données entre deux processeurs adjacents. Ce temps est évalué à partir du nombre de jetons, pour chaque processeur noté PE_k, qui traversent les liens d'entrées des opérateurs de type OUT :

$$(13)\ N_{PEk \to PEr(k)}(\ell) = \sum_{\substack{i=0,\ldots,t-1 \\ M_{i+1}(\ell) - M_i(\ell) = 1}} (M_{i+1}(\ell) - M_i(\ell)) ;$$

$$\ell \in I(a_{OUT}(k))$$

PE_k : processeur émetteur de données
$PE_{r(k)}$: processeur destinataire
$a_{OUT}(k)$: instruction de sortie du processeur k spécifiant l'adresse $r(k)$.

De la même façon que pour le marquage des graphes de flot de données, une fonction Q représentant le nombre de jetons échangés entre deux processeurs est définie par :

$$Q : S \to \{0, N_{PEk \to PEr(k)}(\ell)\}$$
$$Q_1(s) := N_{PEk \to PEr(k)}(\ell) \quad \text{si} \quad s \in [k, r(k)[$$
$$Q_1(s) := 0 \text{ autrement .}$$
$$Q_T(s) := \sum_{k \le s < r(k)} \sum_{\ell \in (a_{out}(k))} (N_{PEk \to PEr(k)}(\ell))$$
$$\text{si} \quad s \in \{k, k+1, ..., r\ (k)\} .$$

L'élément ayant la plus grande valeur de Q_T, $Q_T(s)$ max $\{Q_T\}$ représente la portion du bus où il y a le plus grand nombre d'échanges de données. Le temps total du transfert de données vaut : $2[Q(s) \times cp]$.

DEUXIÈME EXEMPLE (*fig.* 8)

Si nous revenons à l'exemple précédent dans le cas où nous traitons une ligne image de 512 pixels avec un seul processeur, le nombre d'accès vers le bus externe comprend :

— 512 accès pour lire la ligne dans la mémoire image, $N_{PE0 \to PE0}(\ell_2) = 512$ jetons
— 256 accès pour envoyer le résultat vers le processeur suivant qui effectue un autre traitement, $N_{PE0 \to PE1}(\ell_7) = 256$ jetons.

Il faut un seul jeton (deux mots de 16 bits) pour que les processeurs communiquent entre eux ou pour qu'ils lisent un octet en mémoire. Par contre deux jetons sont utilisés pour écrire un octet en mémoire, un contient l'adresse et l'autre la donnée.

Le nombre d'accès sur le bus externe n'est pas identique sur chaque segment du bus. Ici, le processeur générateur d'adresses PE_0 reçoit les données, alors les jetons parcourent entièrement le bus externe en anneau. Mais les données traitées sont dirigées vers un autre processeur PE_1 et dans ce cas uniquement la partie du bus comprise entre ces deux processeurs assure l'échange de données. Donc une partie du bus $PE_1 \to PE_2$ doit assurer le transfert de $Q_T(s_1) = 512$ jetons et l'autre partie du bus $PE_0 \to PE_1$: $Q_T(s_0) = (512 + 256)$ jetons.

Sur le bus externe, l'échange d'un mot se fait en un cycle pipeline et les jetons en deux cycles. Par la suite, on suppose que le temps de cycle des mémoires est inférieur à deux cycles pipeline (400 ns, ceci étant très raisonnable étant donné le temps d'accès des mémoires dynamiques

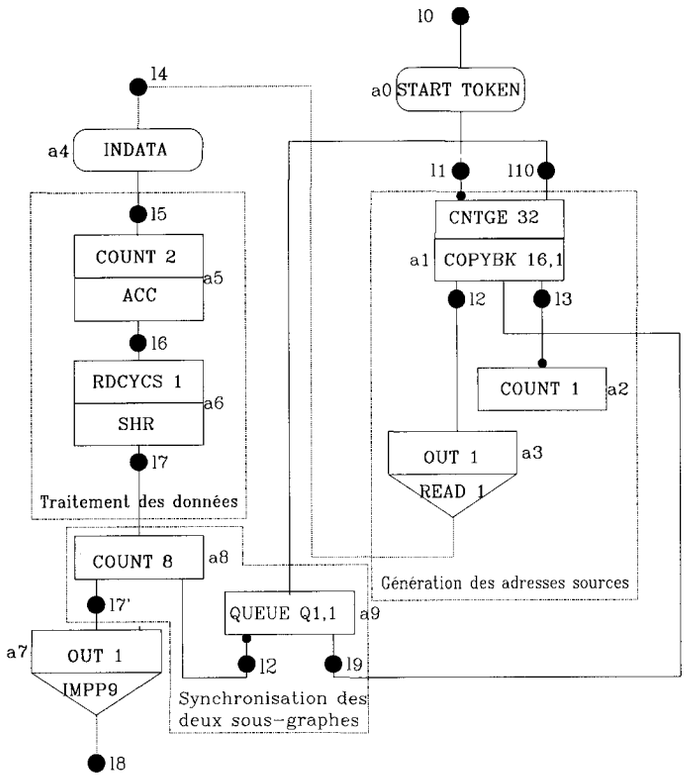


Figure 8. — Programme permettant de réduire une ligne de 512 pixels à 256.

actuelles ($\simeq 100$ ns). Le temps total de transfert dans la partie la plus occupée du bus est

$$(512 + 256) \times 0{,}4 \simeq 0{,}31 \text{ ms}.$$

Ce résultat ne peut pas être amélioré car les accès au bus sont de nature séquentielle. Comparé au temps de calcul du processeur (0,57 ms), il ne pénalise pas le temps global du traitement.

Si nous répartissons maintenant la tâche du premier processeur, sur deux processeurs PE_0 et PE_1, la ligne image est scindée en deux blocs de 256 pixels et chaque processeur traite une moitié de ligne. Le temps de calcul de chaque processeur devient approximativement

$$0{,}57/2 \simeq 0{,}29 \text{ ms}$$

alors que le nombre d'accès est resté inchangé, le temps total est limité par le bus à 0,31 ms. Il est donc inutile de répartir ce programme sur plus de deux proceseurs. Dans ce cas, le temps évalué par le simulateur est de 0,31 ms.

7. Conclusion

Nous avons décrit dans cet article une méthode d'aide à l'utilisateur facilitant une répartition optimale des charges entre les processeurs. Elle répond à l'aspect temps réel des applications envisagées et à la structure des traitements utilisés en inspection automatique. Cette méthode s'intègre dans un semi-interpréteur graphique développé dans notre laboratoire pour faciliter la programmation peu aisée des processeurs μPD7281. L'estimation des temps introduite dans le semi-interpréteur a l'avantage d'indiquer à l'utilisateur si le temps d'exécution est compatible avec l'application envisagée.

En traitement d'images beaucoup de problèmes se divisent naturellement en un petit nombre de tâches successives et indépendantes, par exemple des prétraitements, suivi d'extraction de caractéristiques et puis de reconnaissance de formes. Cette particularité justifie la configuration en bus en anneau des processeurs choisie par le constructeur NEC. La principale difficulté réside dans une distribution équitable des tâches, surtout celles de bas niveau, en tenant compte de ce parallélisme naturel pour obtenir les performances optimales.

L'emploi de processeurs cadencés par les données présente deux avantages par rapport aux architectures traditionnelles. D'une part les échanges entre processeurs se font par la transmission directe des données, réduisant ainsi le goulet d'étranglement d'accès à la mémoire. Et d'autre part au niveau de chaque processeur, le parallélisme intrinsèque à chaque tâche est exploité de façon implicite par l'exécution concurrente des instructions n'ayant pas de dépendances de données. Ces caractéristiques rendent ce concept attrayant en traitement d'images et dans bien d'autres domaines où le nombre d'informations à traiter est important.

En conclusion, la réalisation de machines construites à partir d'architectures basées sur le concept du flot de données est récente, malgré que ce concept soit déjà ancien. Ainsi nous avons réalisé un opérateur multiprocesseur à faible coût et les résultats que nous avons obtenus sont très encourageants. Bien entendu l'absence de langage évolué est un sérieux frein au développement de tels processeurs. Mais l'intérêt que porte ce sujet laisse présager l'avénement de nouveaux langages et avec un peu plus d'expérience l'apparition de machines multiprocesseurs très puissantes.

Manuscrit reçu le 5 avril 1990.

BIBLIOGRAPHIE

[1] T. Jeffery, *The* μPD7281 *processor*. Byte, Nov. 1985, pp. 237-246.

[2] M. Chase, *A pipelined data flow architecture for digital processing, the* NEC μPD7281. IEEE workshop on VLSI signal processing, NEC Electronics inc., Nov. 1984.

[3] A. Zoicas, *L'architecture du* μPD7281, *processeur à flots de données pour le traitement des images*. Minis & micros n° 241, Oct. 1985, pp. 35-40.

[4] K. M. Kavi, B. P. Buckles, U. N. Bhat, *A formal definition of data flow graph models*. IEEE Transactions on computer. vol. C-35. n° 11, Nov. 1986, pp. 940-948.

[5] K. M. KAVI, B. P. BUCKLES, U. N. BHAT, *Isomorphisms between Petri nets and dataflow graphs.* IEEE Transactions on software engineering, vol. SE-13, n° 10, Ovt. 1987, pp. 1127-1134.

[6] NEC, *User's manual* μPD7281.

[7] NEC, *Product description* μPD9305 *for image pipelined processor* 7281.

[8] NEC, μPD7281 *Application library for image pipelined processor.*

[9] R. LECORDIER, P. MARTIN, M. DESHAYES, I. GUIGUENO, *Image processing for automated visual inspection.* EUSIPCO, Sept. 1988, pp. 319-322.

[10] P. MARTIN, R. LECORDIER, L. PIEDFORT, I. GUIGUENO, *Avantages apportés par une architecture à flot de données dans un système de vision artificielle.* Gretsi (Juan les Pins), juin 1989, pp. 833-836.

[11] J. B. DENNIS, *First version of a data flow procedure language ; lecture notes in computer sciences,* vol. 19, Springer Verlag, 1974, pp. 362-376.

[12] K. HWANG, F. A. BRIGGS, *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, 1984.

[13] V. M. MILUTINOVIC, *Computer Architecture Concepts and Systems.* North-Holland, New York, 1988.