T9000 et C104, la nouvelle génération de transputers

Frédéric Desprez, Eric Fleury, Michel Loi

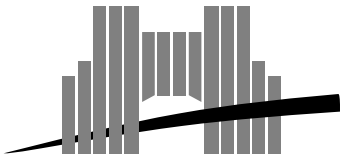
Laboratoire de l'Informatique du Parallélisme
Ecole Normale Supérieure de Lyon
Institut IMAG
Unité de recherche associée au CNRS n°1398

T9000 et C104, La Nouvelle Génération De Transputers

Frédéric Desprez
Eric Fleury
Michel Loi

Février 1993

Technical Report N^{O} 93-01



Ecole Normale Supérieure de Lyon
46, Allée d'Italie, 69364 Lyon Cedex 07, France,
Téléphone : + 33 72 72 80 00; Télécopieur : + 33 72 72 80 80;
Adresses électroniques :
lip@frensl61.bitnet; lip@lip.ens−lyon.fr (uucp).

T9000 et C104, La Nouvelle Génération De Transputers

Frédéric Desprez
Eric Fleury
Michel Loi

Février 1993

Abstract

This document describes the INMOS IMS T9000, the latest member in the transputer family and the IMS C104, a complete routing switch. This transputer combines a general purpose processor with a *grouping of instructions*, a *super scalar pipeline* unit for floating point operations and a *virtual processor channel* to provide hardware communication between two transputers. The C104 is based around the T9000. It is a 32 by 32 way non blocking crossbar switch and allows non local communications between T9000 that are not directly connected.

Keywords: T9000, C104, grouping of instructions, routing

Résumé

Ce document décrit le nouveau processeur T9000 et son processeur de routage $C104$. Le T9000 est un transputer INMOS qui intègre un processeur général avec un *groupeur d'intructions*, une partie dédiée au calcul numérique avec un *pipeline super-scalaire* et un processeur de communication permettant la gestion *hardware* des communications inter-transputers. Le $C104$ est un processeur de communications INMOS permettant de router des messages à travers un réseau et d'opérer des communications non locales.

Mots-clés: T9000, C104, groupeur d'instructions, routage

Introduction

Depuis quelques années, de nombreuses applications nécessitant une très grande puissance de calcul ont émergé. Les machines séquentielles se sont vite avérées insuffisamment puissantes pour résoudre des problèmes dont le temps de calcul doit rester raisonnable pour avoir un quelconque intérêt.

Les processeurs ont, de plus, commencé à atteindre les limites physiques de vitesse. Une alternative réside dans l'utilisation de plusieurs processeurs travaillant ensemble à la résolution d'un même problème.

Le parallélisme apparaît alors comme une solution d'avenir permettant d'aller plus loin dans la simulation de problèmes physiques sur des ordinateurs.

L'utilisation de processeurs adaptés à ce nouveau défi s'avère nécessaire, de même que la résolution du problème du partage et de l'échange des données entre les processeurs. Les futures machines auront plusieurs centaines, voire plusieurs milliers de processeurs. Il conviendra donc de réduire au maximum la distance séparant les processeurs tout en assurant une bande passante importante, afin d'avoir la notion de localité virtuelle entre tout couple de processeurs.

Dans ce rapport, nous présentons les nouveaux produits d'INMOS tout en les situant dans l'espace des recherches actuelles sur les micro-processeurs de calcul et de communication. Dans un premier chapitre nous rappelons les principales caractéristiques des processeurs actuels, puis nous présentons le processeur T9000 en détaillant les caractéristiques spécifiques de ce nouveau processeur d'INMOS. Dans un second chapitre nous présentons les propriétés du processeur de routage C104 associé au T9000.

Chapitre 1

Le processeur T9000

1.1 Introduction

Le processeur T9000 est le dernier né des processeurs INMOS. Il est dédié aux applications temps-réel embarquées, et il constitue surtouune brique pouvant donner naissance à des architectures parallèles à mémoire distribuée. Ceci ne le différencie pas des précédentes versions de transputers. La principale nouveauté est le changement de modèle de communication et de nombreuses innovations à l'intérieur du processeur lui-même.

1.2 Les nouveaux processeurs

La course aux performances est toujours passée par l'augmentation de la puissance des CPU (Central Processing Unit). Cette règle, qui est bien entendu vraie pour les machines séquentielles, ne l'est plus pour les machines parallèles où les échanges entre les processeurs sont un point crucial de l'efficacité.

Dans cette section, nous présentons quelques principes fondamentaux d'architecture des CPU, afin de comprendre ce qui a amené les constructeurs à produire les architectures d'aujourd'hui dont fait partie le T9000.

1.2.1 RISC contre CISC

Dans les processeurs, deux stratégies concernant les ensembles d'instructions dominent. Ces deux stratégies sont les CISC (Complex Instruction Set Computer) et les RISC (Reduced Instruction Set Computer). Un bon exemple de processeur CISC est le VAX de DEC. Ce processeur possède jusqu'à trois opérandes. Ceci a pour effet de simplifier le travail des compilateurs et d'éviter le gâchis de registres temporaires. Par contre, la tâche de la partie contrôle du processeur s'en trouve compliquée, d'où une inévitable perte de performances.

Les architectures des CPU sont passées par plusieurs principes fondamentaux, gouvernés par les possibilités de la technologie. Après être passés du RISC au CISC, les constructeurs sont revenus aux idées de base du RISC pour enfin obtenir un mélange de RISC et de CISC : les CRISC.

Les RISC sont nés de la recherche de performances et des contraintes technologiques. Dans les années 60, un des composants les plus coûteux était la mémoire [2] et tout ce qui pouvait limiter les accès à la mémoire et sa trop grande utilisation relevait du bon principe de fabrication. De nombreux principes du RISC et même du CISC relèvent de cette contrainte.

Les CISC sont nés du principe, chez IBM, de compatibilité entre les différentes versions de processeurs et du désir de réduire la taille du code. Les jeux d'instructions étendus (parfois jusqu'à 200 instructions) conduisaient à une utilisation du micro codage des instructions et ainsi à leur ralentissement.

Les premières idées sur le RISC ont été développées à Berkeley et à Stanford au début des années 80. Historiquement, pour éviter le surcoût dû aux instructions complexes, les constructeurs sont revenus à des jeux d'instructions réduits et à des instructions ayant uniquement des registres comme opérandes. Ceci simplifie le travail de la partie contrôle [1].

Une citation qui résume bien les idées qui ont conduit les concepteurs à produire des architectures RISC provient d'un des architectes du prototype 801 d'IBM [2] :

> "*Si l'existence d'un jeu d'instructions plus complexe ajoute un niveau de logique à un cycle de la machine qui en comporte 10, le processeur est ralenti de* 10 *%. Les améliorations des performances qu'apportent les fonctions complexes doivent d'abord compenser cette dégradation de* 10 *%, puis elle doivent justifier le coût supplémentaire de la machine*"

Les bases du RISC sont les suivantes [2, 6] :

- **Un jeu d'instructions réduit** : Après études, il s'est avéré que dans l'exécution des programmes, 80% du temps CPU était passé dans 20% des instructions existantes (VAX). C'est ce principe qui a donné son nom aux processeurs RISC. Par contre, un code développé pour une unité RISC aura une taille de 25 à 45 % plus grosse que celle d'un code développé pour une unité CISC [19].

- **Moins de formats d'instructions** : Toutes les instructions doivent avoir le même format pour faciliter leur chargement. Généralement, il s'agit d'instructions sur un mot.

- **Registre référence** : Ce registre est toujours égal à 0, quelque soit la valeur que l'on y charge.

- **Architecture "load-store"** : Les seules instructions permettant d'accéder à la mémoire externe sont les *load* et les *store*. Les autres instructions accèdent à des registres.

- **Modes d'adressage limités** : Le seul adressage possible est indirect par un registre. Malgré tout, de nombreux processeurs comme le SPARC, possèdent d'autres modes d'adressage. Ceux-ci sont toutefois moins nombreux que sur les processeurs de type CISC.

- **Pas de micro-code** : Grâce au jeu d'instructions réduits et à leur nombre limité, la partie contrôle du processeur n'a pas besoin d'être microcodée. Ceci a bien sûr pour effet de l'accélérer.

- **Instructions sur un cycle** : Il s'agit là de la règle d'or du concept RISC. En effet, la plupart des instructions étant exécutées sur un seul cycle d'horloge, la vitesse du processeur s'en trouve accrue. Certaines instructions plus compliquées nécessitent toutefois plusieurs cycles pour s'exécuter.

- **Beaucoup de registres** : Le nombre de registres important favorise les accès aux données car plus rapides qu'avec la mémoire et réduit également le nombre d'accès à cette mémoire. Par exemple, le processeur SPARC possède 119 registres 32 bits.

- **Du travail pour les compilateurs** : Il faut laisser aux compilateurs le soin d'optimiser le code en fonction de l'architecture cible. Ce ne sont plus les instructions puissantes qui tirent le meilleur parti des architectures, mais les compilateurs qui allouent au maximum les registres pour réduire les accès à la mémoire et accélérer les accès aux variables.

Il est évident que tous les processeurs RISC ne suivent pas à la lettre ces principes. Cependant, certaines idées de base comme le nombre d'instructions réduit et les instructions effectuées sur un cycle sont généralement suivies sur la majorité des processeurs de ce type. On est toutefois loin des 39 instructions du premier prototype : le RISC-I [18].

Un des moyens d'accélérer les architectures RISC consiste à transformer les différents étages des pipelines en pipelines eux mêmes. Cette technique s'appelle le superpipelining [17]. Grâce à ce pipeline de pipelines, des instructions nécessitant plusieurs cycles pour s'exécuter, peuvent être envoyées les unes à la suite des autres à chaque cycle. Bien entendu, la fréquence d'horloge doit être augmentée pour rendre efficace ce principe, ce qui n'est malheureusement pas possible indéfiniment.

D'importantes recherches ayant été effectuées dans les années précédentes concernant les processeurs CISC, il émerge également une tendance consistant à intégrer dans des processeurs CISC des idées RISC. Intel a d'ailleurs baptisé cette tendance CRISC (Complex Reduced Instruction Set Computer) [11].

Les processeurs dits "RISC" actuels ne sont en fait qu'un mélange de toutes les bonnes propriétés qui ont fait les processeurs du passé, commercialisés ou pas. De gros progrès ont été faits pour obtenir les meilleures performances de la technologie actuelle. Le problème maintenant est surtout d'avoir des compilateurs capables de tirer parti des possibilités de ces processeurs. Les performances en crête sont trop souvent le double des performances obtenues avec des programmes en assembleur et au mieux le quadruple des performances obtenues avec des compilateurs C ou FORTRAN. Les processeurs actuels sont encore trop souvent d'accès difficile. Les constructeurs doivent entre autre faire attention à la demande des utilisateurs et ne pas uniquement se focaliser sur la puissance en crête de leurs processeurs.

1.2.2 Les processeurs vectoriels

Les processeurs vectoriels sont nés du constat que, dans la plupart des programmes scientifiques, de nombreuses sections de code faisaient référence à des vecteurs. Ils ont été introduits au début des années 80. Leurs registres vectoriels et unités de calcul associées, garantissaient de bonnes performances.

Un des principes des processeurs vectoriels est directement issu du RISC : il s'agit d'avoir une instruction vectorielle par cycle.

Ces processeurs concurrencent les minisupercomputers comme le CRAY-I grâce à leur aptitude à effectuer un grand nombre d'opérations flottantes par seconde.

1.2.3 Les processeurs super-scalaires et super-pipelines

Il est maintenant admis qu'une des solutions pour accélérer une machine est de pouvoir effectuer plusieurs opérations en parallèle. Cette idée peut être utilisée également à l'intérieur même du processeur, en employant une architecture super-scalaire, où plusieurs instructions seront effectuées à chaque cycle d'horloge. Dans ce cas, il n'est plus obligatoire de chercher à augmenter coûte que coûte la vitesse d'horloge du processeur, avec les problèmes que cela pose [17].

De cette façon, la course aux performances ne se résume plus à une augmentation des performances de la technologie.

Ceci enfreint l'un des premiers principes des RISC consistant à effectuer une instruction par cycle. Avec les processeurs super-scalaires, on effectue plus d'une instruction par cycle. C'est d'ailleurs là l'origine du terme "super-scalaire".

Le premier processeur super-scalaire était le processeur ACS (Advanced Computer System) d'IBM qui permettait de lancer à chaque cycle 7 instructions en parallèle [2].

Les processeurs super-scalaires ont également engendré ce que l'on appelle les machines VLIW (Very Long Instruction Word) [3, 16] où les instructions destinées à plusieurs unités de traitement sont toutes codées sur un seul mot d'instruction allongé. Le travail du compilateur s'en trouve encore compliqué du fait de la difficulté à gérer les flots de contrôle.

Ces architectures posent deux problèmes : l'accès à la mémoire doit être possible depuis plusieurs unités et la bande passante entre les unités et la mémoire doit être considérablement augmentée.

Le compilateur associé à un tel processeur est plus complexe. Ainsi les performances obtenues avec un compilateur C ou FORTRAN pour un i860 ne dépassent même pas le huitième des performances en crête du processeur.

1.2.4 Les transputers

Le terme *transputer* a été inventé dans les années 70 par Iain Barron, un des fondateurs d'INMOS.

Cette dénomination définit un type de processeur possédant à la fois une unité de calcul et une unité de communication avec l'extérieur, et ceci sur le même composant.

Un tel composant est, de par sa conception, une brique de base toute trouvée pour la conception de machines parallèles à mémoire distribuée.

Les seuls processeurs précédemment commercialisés correspondant à cette définition sont la série des T4XX et T8XX d'INMOS. Il y a à présent un nouveau processeur correspondant à cette dénomination : le TMS 320C40 de Texas Instrument.

En 1985, INMOS a lancé sur le marché le premier microprocesseur créé spécialement pour bâtir des machines parallèles : le T414. Il était parmi les plus rapides du marché.

Le concept de son architecture était simple mais révolutionnaire : mettre dans un même composant un CPU, de la mémoire et 4 liens lui permettant d'être la brique

de base de nouvelles architectures de machines parallèles MIMD à mémoire distribuée (DMPC : Distributed Memory Parallel Computer).

Les versions suivantes T8XX, ont été à la base de nombreuses machines et ont contribué à développer les machines parallèles à mémoire distribuée en Europe. Elles sont de plus très utilisées sur des cartes d'extension pour des ordinateurs personnels, des stations de travail, et même pour des périphériques tels que des imprimantes laser.

Les versions de T800 à 30 MHz peuvent délivrer jusqu'à 15 MIPS et 2.25 Mflops en crête pour les opérations en simple précision.

Grâce à son concept "lego-bloc", et à son crossbar associé le C004, n'importe quel étudiant en électronique est capable de concevoir une machine parallèle à mémoire distribuée à partir de T800.

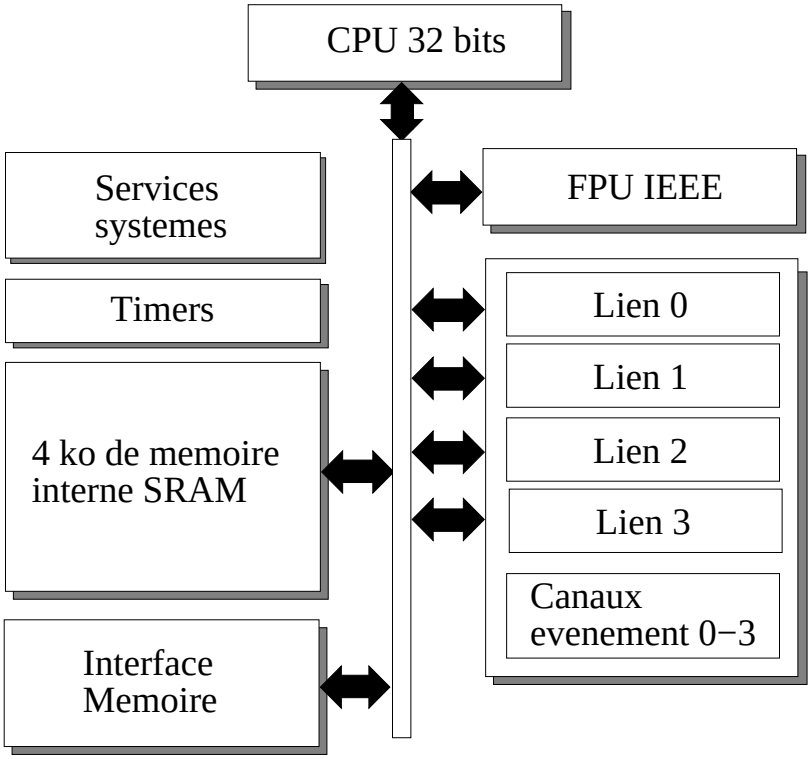


FIG. 1.1 - Architecture d'un T800

Ce processeur possède sur le même circuit, une unité scalaire 32 bits, une unité flottante IEEE, 4 liens bidirectionnels avec l'extérieur (gérés par 8 DMA), une mémoire interne de 4 ko, une interface mémoire externe, 2 timers et une unité de gestion de processus.

Avec leur mémoire interne, les transputers d'INMOS peuvent être complètement autonomes : on peut y placer le code et les données. Notons de plus que les accès à cette mémoire sont plus rapides que ceux à la mémoire externe, ce qui permet de faire des optimisations très intéressantes. En effet un code chargé dans cette mémoire sera exécuté environ 40% plus vite qu'un code en mémoire externe.

Les unités scalaires et flottantes peuvent travailler de manière concurrente et ceci en parallèle avec des communications (grâce aux DMA).

Le modèle de communication est le *store-and-forward 4-port full-duplex*. Les performances des transputers ont été rapportées dans plusieurs articles et rapports [4, 5, 7].

Les transputers ont été conçus en vue d'être programmés à l'aide du langage OCCAM [14]. Ce langage applique le modèle CSP de Hoare [8].

Grâce aux DMA et à la gestion multitâche, des recouvrements calculs/communications sont possibles, ce qui améliore l'efficacité des programmes parallèles, mais pas leur simplicité.

Sa gestion très rapides des changements de contextes (inférieurs à la μs) en font un composant très utile pour le temps réel.

Les performances du transputer T800 sont modestes, comparées à celles d'un processeur comme l'i860 d'Intel, mais ses concepts sont intéressants et ont permis la création de nombreuses applications parallèles dans la recherche et le monde industriel.

Le T800 a été à la base de nombreuses machines parallèles comme le Tnode de Telmat, la Meiko, le FPS T40, les machines Volvox d'Archipel (où il est parfois couplé avec des i860), ainsi que d'une myriade de cartes d'extension pour stations de travail et ordinateurs personnels.

Grâce à ses qualités pour le temps réel, il a été également à la base de nombreux systèmes embarqués.

Le transputer a sans aucun doute contribué au développement du parallélisme en Europe et dans le monde.

1.3 Nouveautés par rapport aux T8XX

Comme ses prédécesseurs, le T9000 a pour principe de mettre sur un même circuit un grand nombre d'unités fonctionnant en parallèle ainsi que des liens de communications avec l'extérieur.

Malgré tout, on peut noter un certain nombre de différences avec les précédentes versions de transputers qui vont toutes dans le sens de l'amélioration des performances et du confort d'utilisation.

Les deux améliorations les plus importantes sont le pipeline super-scalaire et son groupeur d'instructions d'une part et concernant les communications, la gestion hardware des canaux virtuels et l'utilisation du modèle *wormhole* d'autre part. Malgré tout, une connexion entre des transputers de générations différentes est possible grâce au circuit C100 [9].

D'autres améliorations, moins visibles, sont également significatives. La gestion des processus a été améliorée pour une plus grande facilité de programmation.

Le nombre de cycles de certaines instructions a été considérablement réduit, comme par exemple ceux de la multiplication.

L'échange de messages de taille nulle est maintenant possible. Ces messages pourront être utilisés pour des synchronisations de processeurs par échange de messages.

On peut aussi avoir des tailles différentes entre l'émission et la réception, sans blocage des processus de communication.

1.4 Le T9000

Sur un seul circuit, INMOS a souhaité intégrer sur le T9000, deux unités de calcul (une sur les entiers de 32 bits et une sur les flottants de 64 bits), 16 ko de mémoire cache, un processeur de communications, 4 liens de communications ainsi que plusieurs unités système et des *timers*. Ce circuit sera réalisé avec une technologie CMOS.

Son architecture est donnée par la figure 1.2.

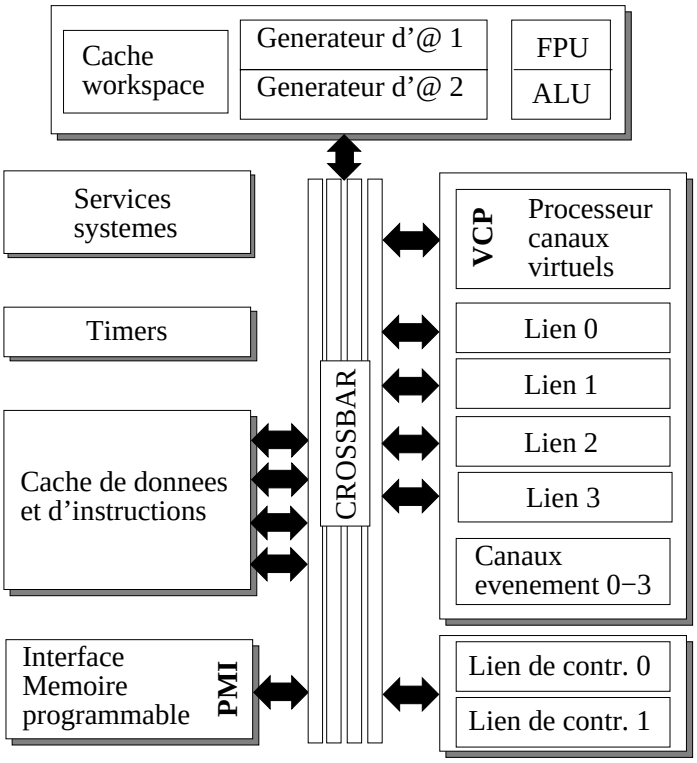


FIG. 1.2 - Architecture du T9000

1.4.1 Ses performances

Les performances du processeur sont de **200 MIPS** et de **25 Mflops** en crête.

Ses réponses aux interruptions ainsi que le changement de contexte se font en moins d'une micro-seconde. Ces performances en font un composant intéressant pour les applications temps réel.

De plus, comme nous l'avons signalé dans la section 1.2.4, le nombre de cycles de la multiplication a été considérablement réduit, passant de 38 à entre 2 et 5 pour des multiplications sur des entiers 32 bits et de 27 à 3 pour des multiplications sur des flottants 64 bits. Le trop grand nombre de cycles de la multiplication sur les anciennes versions pénalisait ces composants pour les applications nécessitant beaucoup de calculs.

La bande passante totale est de **80 Mo/s** pour 4 liens bidirectionnels. Nous reviendrons plus en détail sur les performances des communications de ce processeur dans la section 1.4.6.

1.4.2 Le pipeline et son groupeur d'instructions

Comme nous l'avons décrit dans la section sur les architectures super-pipelines et super-scalaires (1.2.3), une des solutions pour accélérer les processeurs est d'effectuer plusieurs opérations en même temps et avoir ainsi ce que l'on appelle du micro-parallélisme ou parallélisme interne au processeur.

Le T9000 fait partie des processeurs de nouvelle génération qui permettent d'effectuer plusieurs opérations par cycle.

Les instructions sont les mêmes que le T805 mais s'exécutent à une vitesse plus importante.

Le pipeline

Le T9000 possède un pipeline à 7 étages (2+5) (voir figure 1.3) constitué de :

- Les étages de *fetch* et de décodage/groupage : Ceux-ci récupèrent, décodent et groupent les instructions qui seront exécutées dans les 5 étages suivants.
- Le cache espace de travail : récupération de 2 variables locales ou bien chargement de deux constantes. Grâce aux trois ports du cache workspace (2 en lecture et 1 en écriture), deux lectures et une écriture peuvent être exécutées dans le même cycle.
- La génération des adresses non locales : deux calculs d'adresse à trois opérandes peuvent être exécutés en même temps. Cet étage est utilisé aussi bien pour la lecture que l'écriture de données non locales.
- Le cache principal : récupération de deux variables non locales. Si les deux données sont contenues dans des bancs différents, les deux lectures sont effectuées dans le même cycle.
- Les unités entières et flottantes (ALU et FPU).
- L'écriture mémoire et la gestion des sauts : dans le cas d'une écriture, l'adresse aura été calculée dans l'étage de calcul d'adresse et la donnée dans l'étage ALU/FPU. Si l'adresse d'écriture correspond à une adresse présente dans le cache workspace, celui-ci est mis à jour pour garder la cohérence avec la mémoire externe.

Le pipeline du T9000 est d'utilisation transparente pour l'utilisateur, grâce au groupeur d'instructions décrit dans la section ci-dessous.

Le groupeur d'instruction

L'utilisation du groupeur d'instructions et d'un compilateur capable d'ordonnancer les instructions dans un ordre tel que le groupeur les reconnaîtra comme un seul groupe, permettra certainement d'obtenir des performances proches des performances de crête.

Le groupeur tient compte du haut degré de concurrence possible à l'intérieur du pipeline et notamment du fait que les caches ont plusieurs ports.

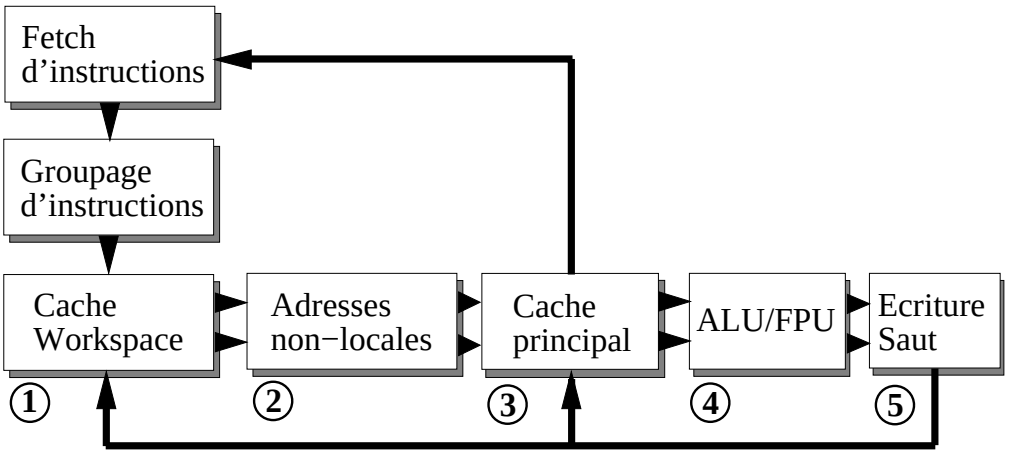


FIG. 1.3 - Pipeline du T9000

Instr	Traduction	Etage du pipeline
ldl j	chargt variable locale j	1
ldl b	chargt adresse de base tableau b	1
wsub	calcul adresse de b[j]	2
ldnl 15	chargt valeur élément b[j+15]	2, 3
ldl k		1
ldl c		1
wsub		2
ldnl 7	chargt valeur élément c[k+7]	2,3
add	addition des deux valeurs en haut de la pile	4
ldl i		1
ldl a		1
wsub		2
stnl 1	rangt dans a[i+1]	2, 5

TAB. 1.1 - Exemple de groupage d'instructions.

Un groupe d'instructions peut contenir jusqu'à 8 instructions et de ce fait, des instructions nécessitant plusieurs cycles pour s'exécuter ne ralentiront pas le pipeline (si toutefois elles sont combinées avec des instructions adéquates).

Les 200 MIPS, calculés à partir du taux d'exécution maximum de 4 instructions par cycle, ne sont que des performances de crête. Il sera assez difficile d'avoir un code adéquat, qui permette un groupage optimal des instructions pour atteindre ces performances.

La table 1.1 donne un exemple de calcul sur des tableaux en assembleur. Grâce au groupeur d'instructions, ce code peut être découpé en 3 blocs qui seront exécutés les uns à la suite des autres.

Grâce au pipeline et à son groupeur d'instruction, un T9000 à 50 MHz pourra exécuter (en théorie) le code d'un transputer de la génération précédente (T805 à 20 MHz) 10 fois plus vite. De plus, la compatibilité binaire devrait être assurée pour la plupart des instructions, ce qui permettrait de pouvoir exécuter directement et plus

rapidement les codes déjà développés sur les anciennes versions.

Sans être la solution à tous les problèmes d'ordonnancement des instructions, le groupeur peut s'avérer être un outil très intéressant d'aide au chargement maximum du pipeline. Il restera toutefois quand même au compilateur la tâche ardue de mettre les instructions dans le bon ordre afin qu'elles puissent être groupées.

Un autre exemple plus complet de groupage d'instructions (multiplication de deux nombres complexes) peut être trouvé dans [13].

1.4.3 Le scheduleur

Comme pour les précédentes versions de transputers, le T9000 possède son scheduleur interne permettant une gestion hardware des processus. Cette fonctionnalité très intéressante permet d'avoir des performances raisonnables pour la gestion des processus, ce qui est rarement le cas avec une gestion logicielle du multitâche.

Le coût de changement de contexte est inférieur à la microseconde.

Les mécanismes de scheduling du transputer sont accessibles aux programmeurs pour le développement de noyaux temps réel.

Les états possibles d'un processus sont :

actif : - s'exécutant
- en attente d'exécution dans une liste d'attente

inactif : - prêt à envoyer
- prêt à recevoir
- en attente pendant un temps donné

Les mécanismes précis de gestions des processus sont décrits dans la documentation INMOS [9].

Le T9000 possède également un système de gestion des interruptions. Un gestionnaire d'interruptions est traité exactement comme un autre processus pouvant émettre et recevoir.

Le T9000 possède par ailleurs 4 ensembles de canaux "événements". Ces canaux sont destinés à des opérations de synchronisation avec des événements externes et de contrôle.

Comme sur les générations précédentes, le T9000 possède deux timers, un avec une période de $1\mu s$, l'autre avec une période de $64\mu s$.

1.4.4 La mémoire hiérarchique

Dans cette section, nous décrivons le système de mémoire hiérarchique du T9000.

Il se compose de trois éléments principaux de mémoire : deux caches (principal et d'espace de travail) et une interface mémoire programmable (voir figure 1.4). Le tout est relié par l'intermédiaire d'un crossbar permettant des accès concurrents entre les différents éléments (voir figure 1.5).

Le cache espace de travail

Une des caractéristiques importantes des processeurs RISC est l'accès très rapide aux variables locales. Sur la plupart des autres processeurs, ceci est réalisé grâce à un grand nombre de registres. Sur le T9000, il n'y a que six registres. Pour palier à ce

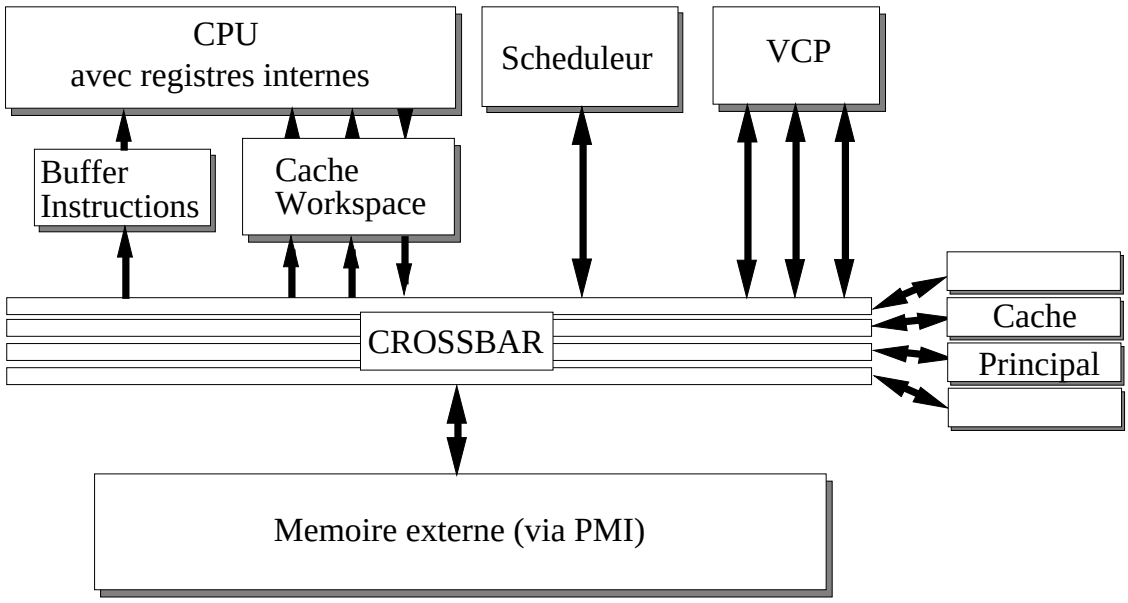


FIG. 1.4 - Système de mémoire hiérarchique du T9000

manque, les variables locales sont stockées dans un cache inclu dans le pipeline, appelé "cache espace de travail" (Workspace cache) [13]. Ce qui donne au T9000 deux niveaux de cache.

Ce cache de 32 mots possède trois ports (un d'écriture et deux de lecture). Ces trois opérations peuvent s'effectuer sur ce cache à chaque cycle. Celui-ci peut contenir une copie des 32 premiers mots de la pile de processus et d'espace de travail. Sa bande passante est de 150 Mmots/s. Cette vitesse lui permet d'être pratiquement aussi rapide en accès que des registres classiques.

Son mode de fonctionnement est de type *write-through*, c'est à dire que dès qu'une variable est mise à jour dans le cache d'espace de travail, elle également est mise à jour dans le cache principal. De ce fait, la mémoire possède toujours les dernières valeurs des variables.

Son fonctionnement précis est décrit dans la documentation INMOS [9].

Le cache principal

Les 16 ko du cache principal sont découpés en mémoire en 4 bancs indépendants de 256 lignes de 4 mots consécutifs (cette mémoire est complètement associative). Celui-ci est de type *write-back* : l'allocation se fait sur faute d'écriture (noter la différence avec le cache workspace). Les bancs sont entrelacés sur les bits d'adresse 4 et 5.

A l'initialisation, ce cache se comporte comme une mémoire interne de 4 ko. Cette mémoire très rapide peut être utilisée dans le cas de systèmes embarqués qui ne nécessitent pas de mémoire externe. Dans le cas de systèmes plus classiques, cette mémoire peut être utilisée pour ranger des variables souvent utilisées, afin d'accélérer leur temps d'accès. Cette méthode était également utilisée dans la précédente version de transputer. On peut également y ranger du code qui s'exécute alors plus rapidement. On peut y placer par exemple un gestionnaire d'interruptions.

Le temps d'accès à cette mémoire est d'un cycle pour chacun des bancs. La bande passante est, quant à elle, égale à 200 Mmots/s.

A l'initialisation, cette mémoire peut être configurée pour se comporter comme 16 ko de cache, 8 ko de cache et 8 ko de mémoire ou 16 ko de mémoire.

La partie cache du cache principal contient du code et des données. Chacun des bancs gère un quart de l'espace mémoire adressable du T9000.

Grâce au cache principal et au cache espace de travail, le CPU peut faire quatre accès mémoire en lecture à chaque cycle, deux pour chaque cache, si bien entendu, il n'y a pas de fautes de cache.

Si une adresse demandée n'est pas présente dans le cache, l'adresse est donnée au système de remplissage de cache (*cache refill engine*). Si cette adresse est cachable, celui-ci calcule toutes les adresses nécessaires au remplissage de la ligne. L'interface mémoire programmable récupère alors la ligne depuis la mémoire externe. Si l'adresse est marquée incachable, alors seule l'adresse demandée est récupérée.

L'interface mémoire programmable (PMI)

Cette interface mémoire gère l'accès à un système de mémoire externe. Elle est programmable, c'est à dire qu'il est possible de construire des systèmes complexes de mémoire externe, contenant des éléments de types différents. On peut avoir jusqu'à 8Mo (4×2 Mo) de mémoire externe DRAM sans rien ajouter comme logique. Le taux de transfert maximum est de 50 Mmots/s.

Cette interface possède 4 ensembles de signaux de contrôle mémoire différents. L'espace d'adressage est divisé en 4 bancs.

Elle peut interfacer des éléments de largeur de bus 8, 16, 32 ou 64 bits.

La programmation de cette interface se fait par le biais de registres spécialisés (un ensemble de registres pour chacun des bancs).

Une autre fonctionnalité intéressante de la PMI est de pouvoir gérer des signaux externes d'accès à la mémoire. Cette fonctionnalité permet de gérer une mémoire partagée entre les processeurs. Des instructions spéciales de vidage des caches ainsi que l'invalidation de données dans les caches existent pour permettre une bonne gestion de la mémoire partagée. De même, certains bancs peuvent être déclarés "non-cachables" de telle sorte qu'un processeur essayant d'accéder à un tel bloc mémoire, sera certain d'accéder à des données valides et ceci plus rapidement que si un vidage du cache avait été nécessaire.

Le crossbar

Le crossbar est au centre du T9000. C'est lui qui connecte entre eux tous les éléments du processeurs (unités de mémoire et d'exécution) (voir figure 1.5). Il contrôle 4 chemins de données de largeur 32 bits (adresse et données).

Il relie les CPU, VCP, PMI et Scheduleur aux 4 bancs mémoires du cache principal. 9 ports sont disponibles : 1 pour la PMI, 4 pour le CPU, 3 pour le VCP et 1 partagé entre le scheduleur et l'unité de contrôle (voir figure 1.5).

Il arbitre les accès concurrents entre les éléments et les bancs du cache principal de telle sorte que chacun des 4 bancs puisse être accédé à chaque cycle.

Quand une adresse est présentée au système de mémoire, celle-ci est routée vers le bon banc mémoire grâce à l'arbitre.

La bande passante totale maximum est de 800 Mo/s (200 Mo/s/banc). Cette bande passante permet de ne pas ralentir (en théorie) le CPU et le VCP. Le CPU a besoin d'une bande passante de 600 Mo/s tandis que le VCP a besoin d'une bande passante de 120 Mo/s.

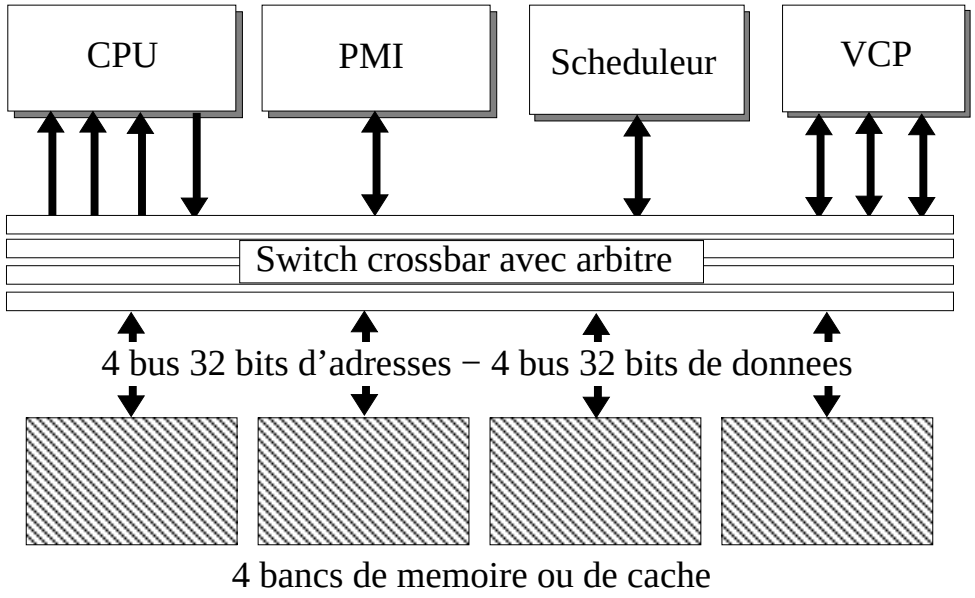


FIG. 1.5 - Liaisons entre le crossbar et les autres éléments du T9000

1.4.5 La gestion de la mémoire

Une des améliorations du T9000 par rapport à ses prédécesseurs est la possibilité d'avoir une protection mémoire empêchant certains processus d'avoir accès à n'importe quelle région.

Les processus invoquant la gestion mémoire sont de type P-Process. Un P-Process s'exécute sous le contrôle de son processus père (le superviseur). Les instructions exécutables par le P-Process sont un sous-ensemble des instructions du T9000.

Les adresses accédées sont logiques, la conversion vers des adresses physiques étant faites par hardware. Dans le cas d'un accès mémoire illégal ou de l'exécution d'une instruction privilégiée, le contrôle est redonné au processus père.

La description précise de la gestion mémoire d'un P-Process peut être trouvée dans la documentation INMOS [9] pages 78 à 81.

1.4.6 Les communications

Nous avons vu dans la section 1.4 que le T9000 possède 4 liens de communication bidirectionnels. Il y a un contrôleur de DMA (Direct Memory Access) pour chaque canal d'entrée et chaque canal de sortie. Des données peuvent donc être transmises sans que le processeur ne soit perturbé.

Le T9000 possède également un scheduleur performant (Cf. section 1.4.3). Plusieurs processus peuvent s'exécuter de façon concurrente sur un même processeur. Afin de mieux exploiter ces deux possibilités, le processeur peut suspendre un processus qui est en attente de communication au profit d'un autre. Quand la communication est achevée, le processus est repris, ce qui permet d'effectuer une synchronisation automatique avec le transfert de donnée.

Les communications entre deux processus résidant sur un même processeur se font via la mémoire du processeur. Les communications point à point entre deux processus distants (placé sur deux processeurs différents) se font via les liens de communication. Un programme constitué d'un ensemble de processus peut donc être exécuté soit sur un seul transputer, soit sur un réseau de transputer comme le montre la figure 1.6.

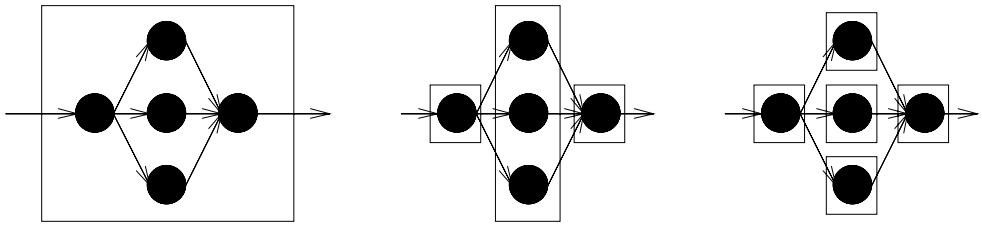


FIG. 1.6 - Différents modes d'exécution d'un même ensemble de processus sur 1, 3 et 5 transputers.

Beaucoup d'algorithmes parallèles nécessitent un nombre de canaux de communication supérieur au nombre de liens physiques entre processeurs. De plus, des processus peuvent vouloir communiquer avec d'autres processus ne se trouvant pas sur un processeur voisin. La gestion des communications du T9000 va apporter une solution nouvelle par rapport aux T8XX :

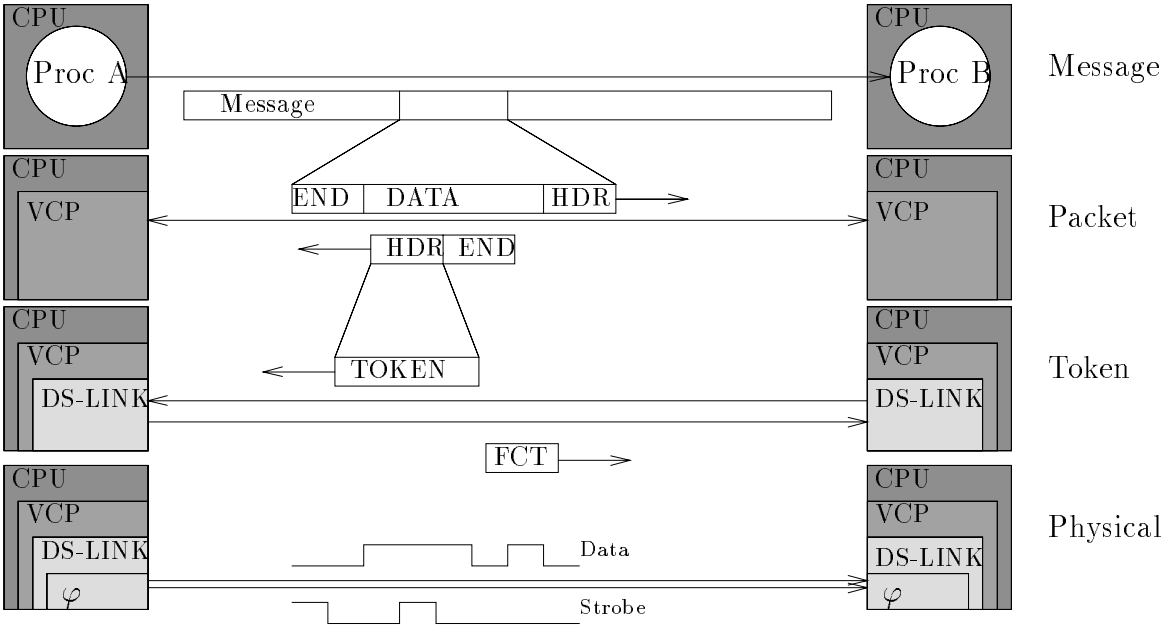
- Le T9000 inclut une gestion hardware des communications en multiplexant plusieurs canaux virtuels sur un lien physique.
- Un routeur nommé C104 permet de construire des réseaux de transputers et de gérer des communications entre processeurs non voisins (Cf. chapitre 2).
- Le T9000 implémente un routage hardware de type *wormhole* (Cf. section 2.2.1) qui diffère du mode de routage software *store and forward* des réseaux à base de T8XX.

Les canaux virtuels

Le T9000 permet d'effectuer un nombre arbitraire de communications point à point via un seul lien physique grâce à son processeur de communication nommé VCP (*Virtual Channel Processor*) capable de multiplexer un nombre quelconque de liens virtuels sur un seul lien physique.

Le protocole de communication est basé sur un concept à quatre couches comme le montre la figure 1.7. Les *messages* sont échangés entre *processus* et peuvent être de taille quelconque. Ces messages sont découpés en *paquets* lors de la transmission. Les paquets sont échangés entre *processeurs* et sont transmis sous forme de flot de *tokens*.

Les tokens sont échangés entre *devices* (nommés DS-link) et transportent des données ou des informations de contrôle de flot. Un DS-link est en fait composé de quatre liens, 2 dans chaque direction (un lien *data* et un lien *strobe*). Le protocole entre deux DS-link garantit que les deux signaux ne changent pas d'état durant le même top d'horloge. Le signal de strobe change d'état chaque fois que le signal de data reste inchangé. Chaque fois que l'on détecte un changement d'état (strobe ou data) on lit le bit sur le signal data. Les tokens sont codés lors de la transmission. Il reste la couche *physique* qui permet le transport des tokens le long des fils de cuivre... La figure 1.8 donne un aperçu d'une communication uni et bidirectionnelle via un lien virtuel.



Un processus du processeur A envoie un message à un autre processus du processeur B. C'est le VCP qui se charge de découper le message en paquets. Chaque paquet contient un entête, des donneés et une fin de paquet. Un paquet est lui-même composé de tokens. La gestion de ces token est effectuée par le DS-link. Reste la couche physique (φ). Les deux signaux *data* et *strobe* d'un seul lien sont représentés sur la figure entre les couches physiques.

FIG. 1.7 - Différentes couches du protocole pour les communications

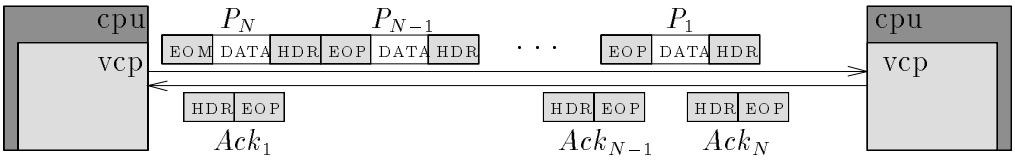
La figure 1.9 donne le détail de la structure des paquets qui forment le message et dont la taille est restreinte à 32 octets. Chaque paquet a un entête (*header*), dont la taille est de 1 ou 2 octets, utilisé pour :

– le routage du paquet vers le processus destinataire d'un processeur éloigné.

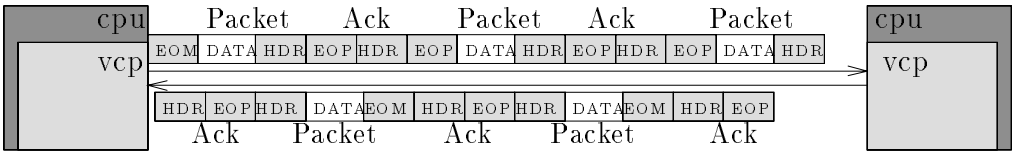
– identifier le bloc de contrôle du lien virtuel utilisé par le processus destinataire.

Un lien virtuel est initialisé par la création d'un bloc de contrôle dans chacun des deux transputers et l'information contenue dans un entête de message peut donc se limiter à l'adresse de sa destination (l'information concernant la source n'est pas utile). Chaque paquet d'un message est transmis directement :

– du processus envoyeur vers le lien physique.



Communication unidirectionnelle : Le message à envoyer est découpé en $P_1 \ldots P_N$ paquets. Ces paquets sont acheminés sur un canal d'un lien virtuel. Les acquitements corespondant $Ack_1 \ldots Ack_N$ sont acheminés sur l'autre canal du même lien virtuel.



Communication bidirectionnelle : Les paquets sur le canal sortant vont être mélangés aux acquitements des paquets du canal entrant, et réciproquement.

FIG. 1.8 - Communication uni et bidirectionnelle via un lien virtuelle

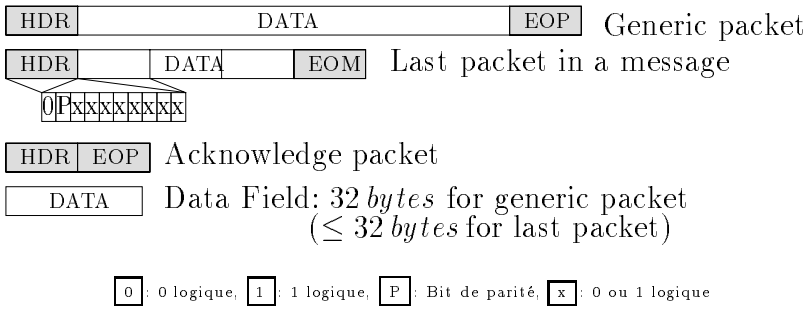


FIG. 1.9 - Structure d'un paquet.

– du lien physique vers le processus récepteur (dans l'hypothèse où un processus attend l'arrivée du paquet).

Avec l'hypothèse précédente, un acquittement est envoyé vers le processus envoyeur sur le lien virtuel pour chaque paquet reçu par le processus receveur. De cette façon, la transmission de l'acquittement peut être recouverte par la transmission des messages.

Si le premier paquet d'un message arrive sur un lien virtuel, il est possible qu'aucun processus ne soit en attente de message. Dans ce cas, le paquet doit être stocké temporairement dans des buffers associés au bloc de contrôle. Ceci sous-entend que les communications via des liens virtuels partageant le même lien physique ne sont pas pénalisées. Le stockage des paquets se fait au niveau du DS-link qui doit avoir un buffer pouvant stocker au minimum 8 tokens. Le DS-link effectue la gestion des tokens entre processeurs par un contrôle de flot. Un token de contrôle de flot est envoyé chaque fois qu'il y un buffer libre pouvant contenir 8 tokens. La figure 1.11 donne un exemple de circulation de tokens de flot de contrôle entre deux DS-LINK.

Bande passante utilisable

La taille des différents tokens (Cf. Figure 1.10) est de 10 bits (8 de données, 1 bit de parité et 1 bit de flag indiquant la présence de données) pour les tokens de donnés,

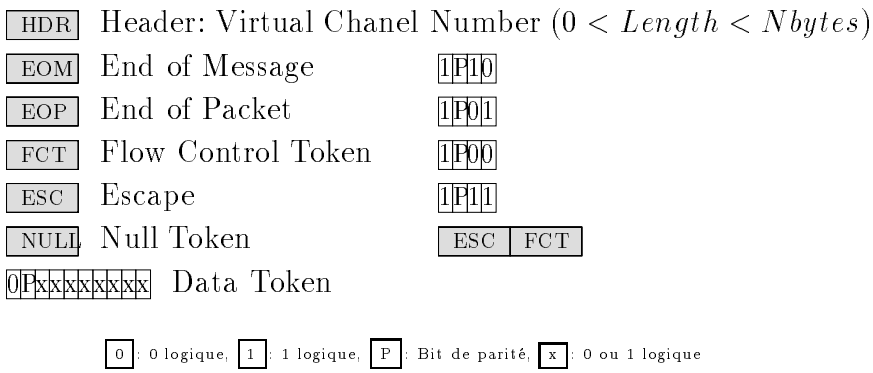


FIG. 1.10 - Structure des différents token.

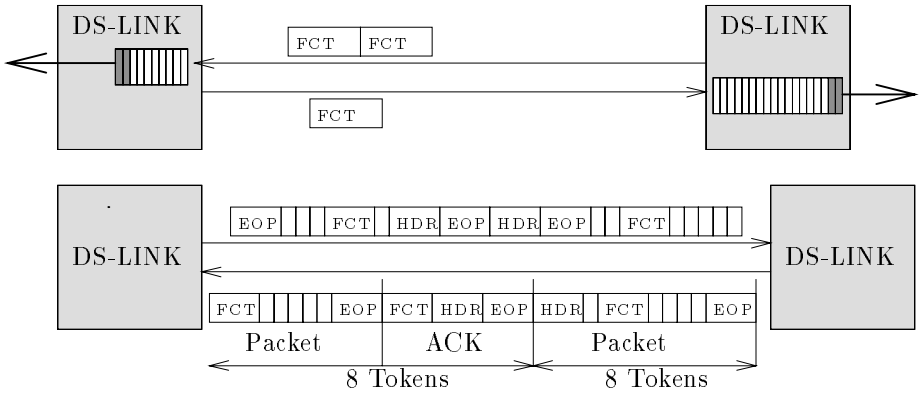
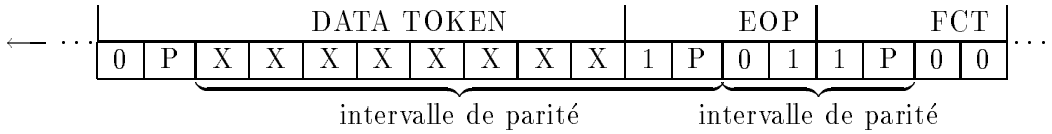


FIG. 1.11 - Gestion des buffers de token par contrôle de flot

4 bits (avec un bit de parité) pour les tokens de flot de contrôle, fin de paquet, fin de message. La parité couvre les tokens précédents :



Il semble évident que le débit de 100 Mbits/s ne pourra pas être pleinement atteint par l'utilisateur pour les raisons suivantes :

- chaque octet d'information est codé sur 10 bits.
- un token de flot de contrôle est envoyé tous les 8 tokens de données reçu.
- chaque paquet a un entête et un token de fin.
- pour chaque paquet transmit, un token d'acquittement est envoyé.

Considérons le cas d'une communication unidirectionnelle. Soit m la taille du message en octets, il sera transmis en n_p paquets où $n_p = \lceil \frac{m}{32} \rceil$. Soit s la taille de l'entête du message. Le nombre de bits transmis est : $b_d = 10m + (10s + 4)n_p$. Les n_p acquittements

vont utiliser l'autre canal physique du lien. Cependant, ces acquittements vont générer des tokens de flot de contrôle puisqu'ils vont avoir une taille de $n_{dt} = (s+1)n_p$ tokens. Tous les 8 tokens, un FCT (Flot Control Token) est envoyé. Donc le nombre de FCT est $n_{ft} = \lceil \frac{n_{dt}}{8} \rceil$. La taille d'un FCT étant de 4 bits, le nombre total de bits transmis est $B = b_d + 4n_{ft}$. Le nombre de bits transférés propres au message est $8m$ et requière B bits. Le taux de transfert est $D = \frac{8m}{B} \times 100$ Mbits/s sur un lien à 100 Mbits.

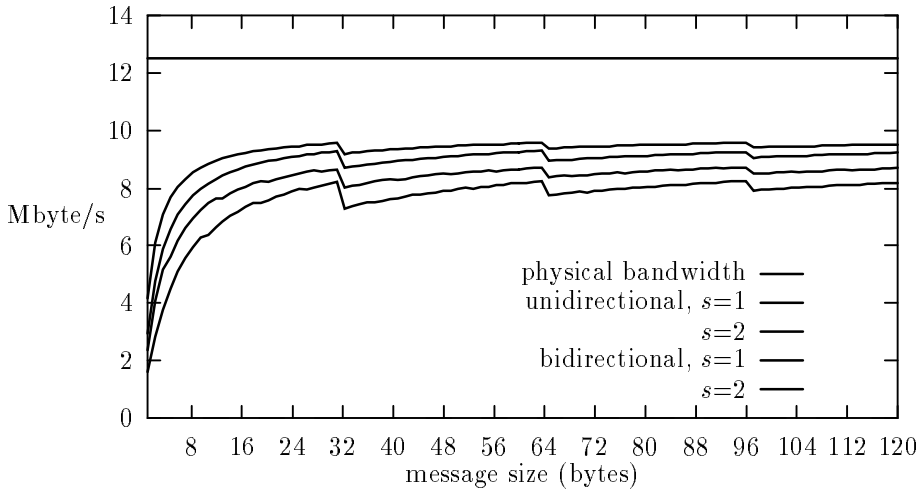


FIG. 1.12 - Débit sur un lien pour des messages courts

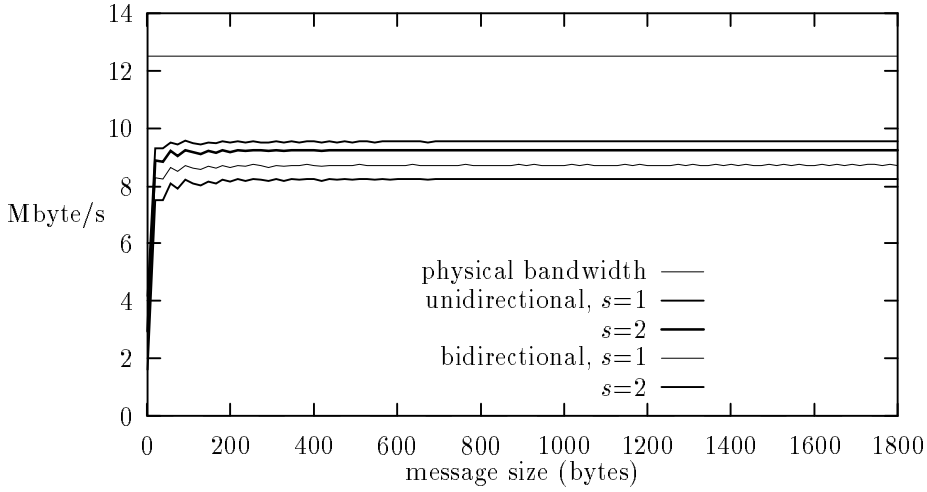


FIG. 1.13 - Débit sur un lien pour des messages longs

Prenons maintenant le cas d'une communication bidirectionnelle. On suppose que le message est sur le canal sortant et que la même quantité d'information circule sur le canal entrant. Le nombre de paquets ne varie pas : $n_p = \lceil \frac{m}{32} \rceil$. Le nombre de bits transmis pour le message sortant et pour les acquittements sortant est : $b_d = (10m + (10s+4)n_p) + (10s+4)n_p)$. Le lien de sortie va transporter les FCT des données du lien entrant. Le nombre de tokens sur le lien entrant est $n_{dt} = (m + (s+1)n_p) + (s+1)n_p$. Donc le nombre de FCT est $n_{ft} = \lceil \frac{n_{dt}}{8} \rceil$. Le nombre total de bits transférés est donc $B = b_d + 4n_{ft}$ et le taux de transfert est $D = \frac{8m}{B} \times 100$ Mbits/s sur un lien à 100 Mbits.

La valeur asymptotique en fonction de la taille de l'entête est $D = \frac{51200}{649+21s}$ pour le cas unidirectionnel et $D = \frac{25600}{345+21s}$ pour le cas bidirectionnel. Nous avons représenté sur les figures 1.12 et 1.13 la bande passante (Moctets/s) obtenue en fonction de la taille des messages (octets).

1.5 Conclusion

Les performances du T9000 peuvent paraître modestes vis à vis de la concurrence. En effet, si l'on compare les puissances en crête du T9000 par rapport à celle du processeur $i860$ d'Intel, du processeur $R4000$ de chez MIPS ou le $DEC21064$ (plus connu sous le nom d'Alpha) elles sont largement inférieure (en nombre de Mflops et de Mips). Cependant, comme nous l'avons vu dans la section 1.4.6 le T9000 innove en intégrant un processeur de communication virtuelle (VCP) qui permet d'utiliser un protocole de communication wormhole entre 2 processeurs voisins et de multiplexer un nombre quelconque de canaux virtuels sur un lien physique. Un concurrent plus sévère pour le T9000 peut être le $C40$ de Texas Instrument qui se positionne sur le même créneau.

L'atout majeur du T9000 va être son routeur associé, le C104 que nous allons décrire dans la chapitre 2 et qui va le distinguer du $C40$ mais aussi de tous les autres processeurs n'intégrant pas de processeur de communication.

Chapitre 2

Le circuit de routage C104

2.1 Introduction

Nous avons vu que le T9000 était capable d'exécuter des communications entre deux processus sur un même processeur via la mémoire et entre deux processeurs voisins via un lien point à point. Nous allons voir comment opérer des communications entre des processeurs distants grâce à un *routeur* de paquets C104. Ce routeur va permettre d'interconnecter plusieurs T9000 entre eux pour réaliser divers topologies de réseaux d'interconnection.

2.2 Description du C104 et de son mode de routage

Le protocole de communication entre transputers a été décrit dans la section 1.4.6. Chaque *message* envoyé le long d'un canal virtuel est divisé en *paquets*, et pour que des paquets puissent être mélangés sur un même *lien physique*, chacun contient un *en-tête* dont la taille est de 1 ou 2 octets. Cet en-tête identifie le *lien virtuel* d'entrée du processeur destinataire. Chaque paquet est acquitté, et la procédure d'acquittement s'effectue par l'envoi de paquets ne contenant pas de données. Ces acquittements sont acheminés par le second canal du lien virtuel. Si l'on effectue une communication bidirectionnelle, ils sont mélangés aux paquets de données sur un lien physique.

Le C104 est utile pour pouvoir communiquer entre des transputers non voisins. Le C104 est un routeur de paquets. Sa tâche est donc de router des paquets qui arrivent sur un lien, vers un autre lien.

Le C104 a 32 liens (Fig. 2.1). Il inclut un *crossbar* 32×32 non bloquant capable de router un paquet de n'importe quel lien vers n'importe quel autre. Il faut noter que les liens sont bidirectionnels c'est-à-dire, si un lien est connecté avec un autre lien cela signifie que son lien entrant est connecté au lien sortant et vice versa. En plus des 32 liens, il possède deux liens de contrôle qui seront détaillés dans la section 2.3.

Le C104 possède également une unité de commande qui lui permet de sélectionner le lien de sortie en fonction de l'en-tête d'un message. Tout ce qui suit un en-tête est considéré comme étant le corps du paquet jusqu'au *token* de fin de paquet. Ceci va permettre au C104 de transmettre des messages de taille arbitraire.

Les performances des communications dépendent du choix du mode de routage notamment par le *temps de latence* que l'on peut définir comme étant le temps minimum

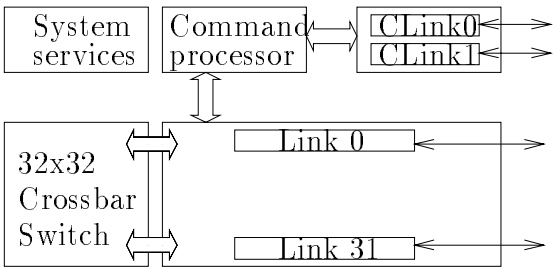


FIG. 2.1 - Architecture simplifiée du C104

nécessaire pour transmettre un message de longueur nulle (composé seulement d'un en-tête) entre deux processeurs, et par la *bande passante* utilisée qui est le nombre d'octets transmis durant le temps pris par la communication par rapport à la bande passante disponible sur l'ensemble des liens utilisés lors de cette communication.

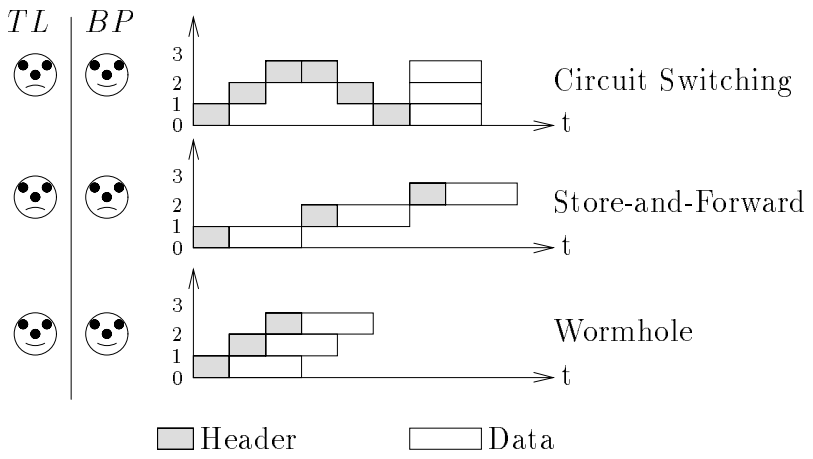
Dans les versions précédentes du transputer (T8XX), le mode de routage s'effectuait dans un mode *Store and Forward*. Un message progressait dans un réseau en étant entièrement stocké dans les noeuds intermédiaires. Ce mode a comme inconvénient majeur un temps de latence assez important dû au fait qu'à chaque étape, chaque paquet est stocké, décodé et envoyé vers son processeur destinataire. De plus, la bande passante n'est pas pleinement utilisée puisque, pour un message, on utilise un seul lien à la fois sur l'ensemble des liens nécessaires à la communication (Fig. 2.2).

Un second mode de routage, le *circuit switching* est utilisé par exemple sur les hypercubes de type *iPSC* et sur la machine Paragon d'Intel. Cette technique est comparable au principe du téléphone. Un en-tête est envoyé afin de créer un circuit entre les deux processeurs voulant communiquer et, une fois que ce circuit est réalisé, le message est envoyé, en une seule fois. Le temps de latence reste important (temps qu'il faut pour établir un circuit) mais la bande passante est pleinement utilisée puisqu'une fois le circuit établi on utilise la bande passante de tous les liens.

Pour essayer de pallier aux inconvénients de ces deux modes, le C104 va implanter un mode de routage différent.

2.2.1 Mode de routage du C104

Le C104 utilise un mode de routage *wormhole* qui permet de router un paquet dès que l'en-tête est arrivé sur le C104. La figure 2.3 donne une représentation schématique de ce *ver de terre* progressant à travers le réseau en ouvrant un chemin devant lui au fur et à mesure et le refermant automatiquement après son passage. Si le lien de sortie que l'on doit emprunter est libre, l'en-tête est directement transmis et le reste du message suit. Dans le cas contraire, l'en-tête est bufferisé. Ainsi, l'en-tête qui passe à travers un réseau de C104 crée un circuit temporaire nécessaire à l'acheminement du message. Plusieurs messages peuvent transiter simultanément par un routeur en un temps donné. Le mode de routage implique que l'en-tête puisse être reçu par le destinataire avant que la totalité du message n'ait été envoyée. Ceci permet de minimiser le temps de latence et de bien utiliser la bande passante (Fig 2.2).



TL : Temps de Latence. BP : Bande Passante.

Les différents graphiques schématisent le temps de communication entre deux processeurs à distance 3.

FIG. 2.2 - Différents modes de routage.

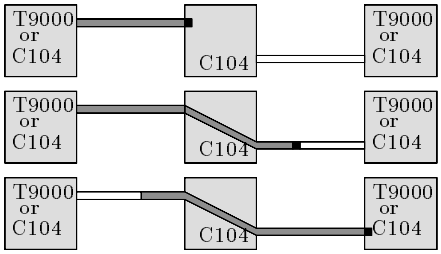


FIG. 2.3 - Mode de routage wormhole

2.2.2 Algorithme de routage du C104

Le *mode* de routage wormhole du C104 présenté ci-dessus nécessite une *stratégie* de routage afin de déterminer sur quel lien doit être envoyé un paquet qui vient d'arriver. Cet algorithme doit être sûr (tout paquet envoyé doit arriver !), sans interblocage, et simple afin de ne pas prendre trop de temps, de façon à minimiser le temps de latence. De plus, la partie automate de contrôle du C104 ne doit pas prendre une surface trop importante lors de l'implantation matérielle. Le C104 utilise un algorithme nommé *intervalle labeling*.

Supposons que l'on ait N transputers, on assigne à chaque transputer un numéro de 0 à $N-1$. Pour tout routeur dans le réseau, à chaque lien de sortie de chaque C104 correspond un intervalle de numéros consécutifs. On note $[x, y)$ l'intervalle défini par $\{n | x \leq n < y\}$. Cet intervalle contient les numéros des transputers qui sont accessibles par le lien comme le montre la figure 2.4. Les intervalles associés aux liens d'un routeur sont non-recouvrants et tout label doit appartenir à un intervalle.

Lorsqu'un paquet arrive dans un routeur, ce dernier compare son en-tête (i.e. sa destination) avec l'ensemble des intervalles, et le dirige vers le lien de sortie correspondant à l'intervalle contenant l'adresse du processeur destinataire. L'information nécessaire à

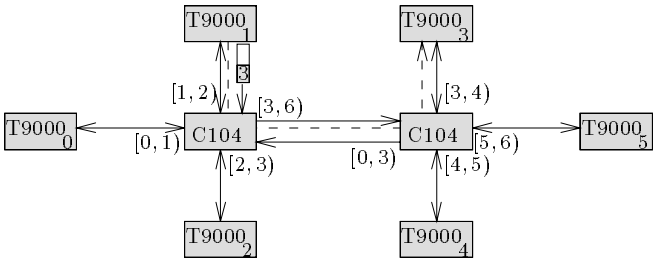


FIG. 2.4 - Exemple de routage par numérotation d'intervalle.

cet algorithme est restreinte et est stocké dans des registres spéciaux de la partie sélection d'intervalle (Fig. 2.7). L'initialisation de ces intervalles est faite lors du démarrage du système de manière logicielle via les liens de contrôles (Cf. section 2.3) . Un exemple de routage d'un message est donné dans la figure 2.4. Le processeur 1 veut envoyé un paquet au processeur 3. L'en-tête du paquet contient donc 3. Quand le paquet arrive sur le premier C104, il compare son en-tête avec les intervalles et est dirigé vers le lien $[3, 6)$, puis de la même manière sur le lien $[3, 4)$ du second C104.

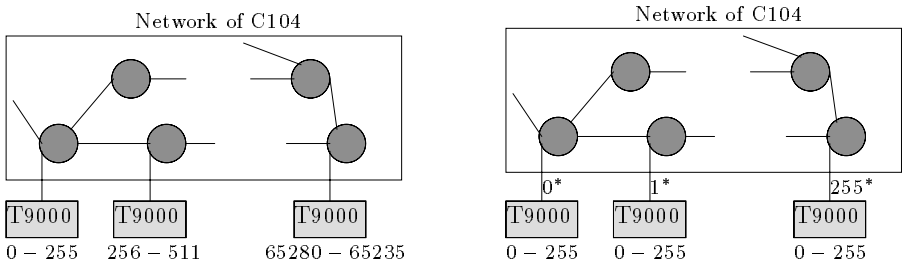
Tout réseau peut être numéroté de manière à obtenir un routage par intervalle sans interblocage [10, 12]. Il suffit de construire un arbre de recouvrement. Cependant, le routage obtenu n'est pas forcément optimal. Il est possible de trouver une meilleure numérotation d'intervalle pour la plupart des réseaux usuels (hypercube, grille, arbre, réseau multi-étage) telle que le routage soit optimal et sans interblocage [12].

2.2.3 Suppression d'en-tête

Un inconvénient majeur du mode de routage par intervalle est qu'il ne permet pas à un message d'être routé à travers une hiérarchie de réseaux. Une solution à ce problème est le *grignotage d'en-têtes*. Chaque lien de sortie d'un C104 peut être programmé (Cf. section 2.3) pour supprimer l'en-tête d'un paquet. Les données qui suivent l'en-tête deviennent le nouvel en-tête. L'en-tête global d'un paquet est composé de plusieurs sous en-têtes (Fig. 2.6 (b)). Ceci permet de :

- simplifier la numérotation du système en séparant la numérotation du réseau de celle des liens virtuels sur les T9000 (Fig. 2.5).
- dépasser la limite de 64K canaux virtuels par système (réseau de C104 et de T9000) dû à la limitation de 2 octets maximum pour un en-tête.
- construire des réseaux hiérarchiques (Fig. 2.6 (a)).
- minimiser le temps de latence.

La figure 2.5 donne un exemple de l'utilisation de ce mode de grignotage d'en-tête. L'utilisation de 2 sous en-têtes permet de faire décroître le temps de latence. En effet, le nombre de transputers étant généralement inférieur au nombre de canaux virtuels, le premier sous en-tête peut être de taille réduite (un seul octet). Ainsi, tant que le message est dans le réseau de C104, il est considéré comme n'ayant qu'un seul en-tête de 1 octet ce qui minimise le délai entre chaque routeur. De plus, le nombre de



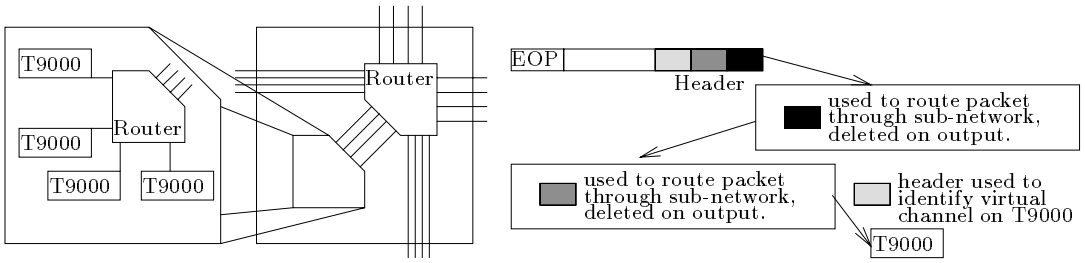
(a) : Tous les liens du système sont programmés pour recevoir un en-tête de 2 octets. La table de routage des intervalles des C104 et les en-têtes des T9000 sont programmés pour supporter 256 canaux virtuels reliant chaque T9000. La valeur des en-têtes est indiquée en dessous de chaque T9000.

(b) : Tous les liens du système sont programmés pour recevoir un en-tête de 1 octet, et les liens terminaux (numérotés de 0^* ... 255^*) du réseau de C104 sont programmés pour grignoter un en-tête. Un paquet est maintenant transmis avec 2 sous en-têtes de 1 octet. Le premier sert à router le paquet à travers le réseau jusqu'au C104 terminal qui le grignote. Le second sert alors à identifier le canal virtuel.

FIG. 2.5 - Exemple de grignotage d'en-tête afin de faciliter la numérotation d'un système en faisant la distinction entre le réseau et les canaux virtuels.

liens virtuels peut alors être augmenté en programmant les transputers pour accepter 2 octets d'en-tête, tout en ne modifiant pas la taille (1 octet) des en-têtes du réseau de C104 (Fig. 2.5 (b)).

Une autre utilisation du grignotage d'en-tête est la construction de réseaux hiérarchiques. On peut imaginer que dans une grille 2D, chaque transputer puisse être remplacé par un réseau local de transputer comme le montre la figure 2.6.



(a) : Les en-têtes sont supprimés lors de leur entrée ou sortie du réseau local

(b) : Exemple de routage d'un paquet à travers 2 réseaux avec comme destination finale un canal virtuel d'un T9000. L'en-tête est constitué de 3 sous en-têtes.

FIG. 2.6 - Réseau hiérarchique utilisant le grignotage d'en-tête.

2.2.4 Routage universel

Nous avons vu qu'il existe une stratégie de routage *wormhole* par intervalle sans interblocage. Cependant, la vitesse de transmission des messages est dépendante du nombre de collisions entre paquets sur un lien. Il peut arriver, par exemple lors de communications dispersées sur un réseau, qu'un nombre important de messages passent par le même lien, créant un *goulot d'étranglement* dans le réseau et augmentant considérablement le temps de communication du fait de la mise en attente des messages. Pour éviter ce cas, il est possible d'utiliser un algorithme de *routage universel* [15] qui se déroule en deux étapes :

1. routage du paquet vers un routeur intermédiaire choisi aléatoirement.

2. routage du paquet vers sa destination finale.

Dans les deux cas, le routage est wormhole et l'algorithme de routage est par intervalle. Cet algorithme peut être implanté facilement sur le C104 en le programmant pour appliquer un routage universel.

La première étape se fait en programmant les liens d'entrée d'un routeur afin d'ajouter un en-tête aléatoire à chaque paquet qui arrive dans ce routeur afin d'indiquer la destination intermédiaire du paquet. Cet en-tête est choisi aléatoirement dans un intervalle de valeurs programmable (Cf. section 2.3). Ce générateur est connecté à chaque lien comme le montre la figure 2.7. Le paquet est alors routé vers sa nouvelle destination.

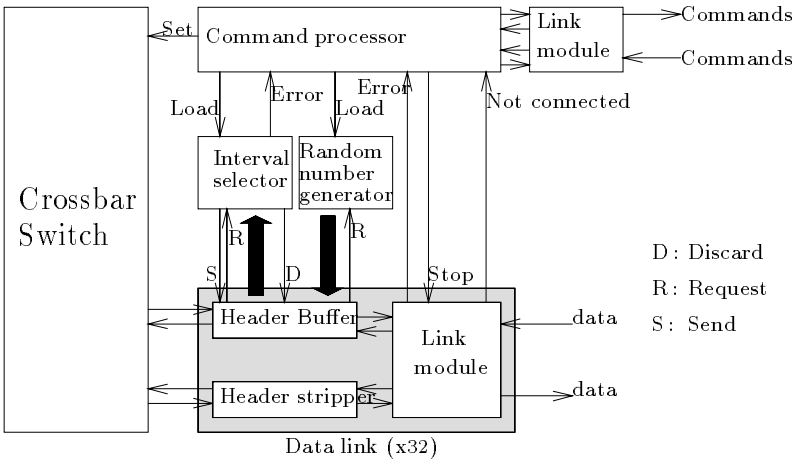


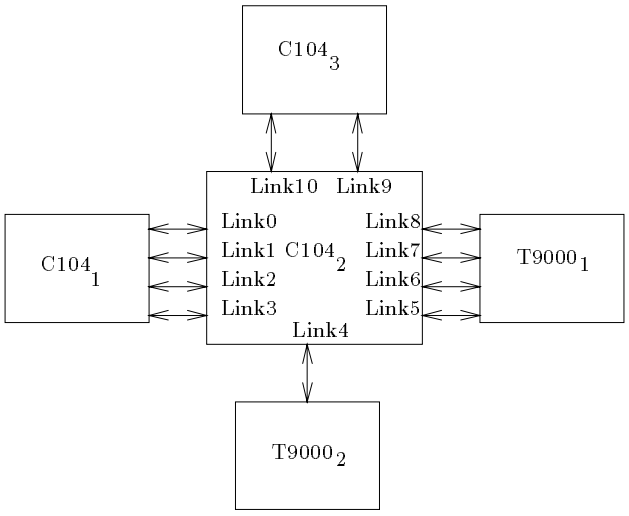
FIG. 2.7 - C104 détaillé

La seconde phase consiste a supprimer l'en-tête aléatoire quand un paquet atteint sa destination intermédiaire. Chaque C104 est capable de reconnaître un intervalle de *valeurs seuils*. Quand un en-tête appartient à un tel intervalle seuil, il est supprimé. Le paquet est alors routé vers sa destination originelle.

Afin que cette technique ne génère pas à son tour des points chauds, il est préférable que les deux phases utilisent des canaux différents. Tous les liens du réseau doivent être considérés comme étant soit *"destinations"* soit *"aléatoires"*. Ce schéma va être équivalent à deux réseaux séparés, un pour la phase aléatoire et un pour la phase de routage. La conjugaison des deux est sans interblocage si les deux réseaux sont sans blocage.

2.2.5 Routage adaptatif groupé

On peut aussi implanter un *routage adaptatif groupé*. Plusieurs liens consécutifs d'un même C104 sont regroupés en sous-groupes de sorte que, lorsqu'un paquet arrive, il est routé vers le premier lien libre du sous-groupe auquel il appartient. La figure 2.8 donne un exemple de routage avec cette méthode. Ce mode de routage permet d'améliorer les performances d'un réseau en diminuant le temps de latence.



Soit un paquet routé du $C104_1$ via le $C104_2$ à destination du $T9000_1$. Supposons qu'arrivant sur le $C104_2$, l'en-tête du paquet spécifie que ce dernier doit être dirigé en sortie sur le lien 6. Si ce lien est utilisé, le paquet sera automatiquement routé vers les liens 5, 7 ou 8 selon l'ordre de leur disponibilité.

0	Start	Grouped
1	Continue	
2	Continue	
3	Continue	
4	Start	
5	Start	Grouped
6	Continue	
7	Continue	
8	Continue	
9	Start	Grouped
10	Continue	
11	Start	
⋮	⋮	
31	Start	

Les liens sont configurés par le registre spécial **Group**. Chaque bit correspond à un lien et peut être initialisé à "Start" pour commencer un groupe ou "Continue" pour être inclus dans un groupe.

FIG. 2.8 - Routage adaptatif groupé

2.3 Configuration du C104

2.3.1 Les différents paramètres programmables

Comme on a déjà pu le remarquer dans les sections précédentes, le C104 est "configurable" puisque plusieurs paramètres (spécifiques pour chaque lien) peuvent être modifiés par l'intermédiaire de registres :

- valeur des bornes de chaque intervalle (**Interval**$_0$ ...**Interval**$_{34}$).
- numéro du lien associé à chaque intervalle (**SelectLink**$_0$ ...**SelectLink**$_{34}$).
- positionnement de la valeur seuil pour le routage universel (**Discard**$_0$ ... **Discard**$_{34}$).
- positionnement du drapeau pour le mode de génération aléatoire pour un lien d'entrée (**Randomize**).
- valeur des bornes du générateur aléatoire (**RandomBase** et **RandomRange**).
- positionnement du drapeau pour le mode de grignotage des en-têtes sur les liens de sortie (**DeleteHeader**).
- longueur (1 ou 2 octets) des en-têtes de chaque lien (**HeaderLength**).
- positionnement des groupes pour le routage adaptatif groupé (**Group**).

La programmation de ces registres se fait via les liens de contrôle comme nous le verrons dans la section 2.3.2. Le choix du lien de sortie est effectué par le *sélecteur d'intervalle* (Fig. 2.7). Les différents registres de ce sélecteur sont représentés dans la

figure 2.9. Chaque comparateur est connecté à une paire de registre **Interval**$_n$ sauf le premier dont la base est fixé à zéro. Chaque registre **Interval**$_n$ est donc la base inférieure d'un comparateur et la limite supérieure d'un autre sauf pour le dernier qui sert juste de limite supérieure. Les registres **SelectLink**$_n$ contiennent le numéro du lien associé à l'intervalle correspondant [**Interval**$_{n-1}$, **Interval**$_n$).

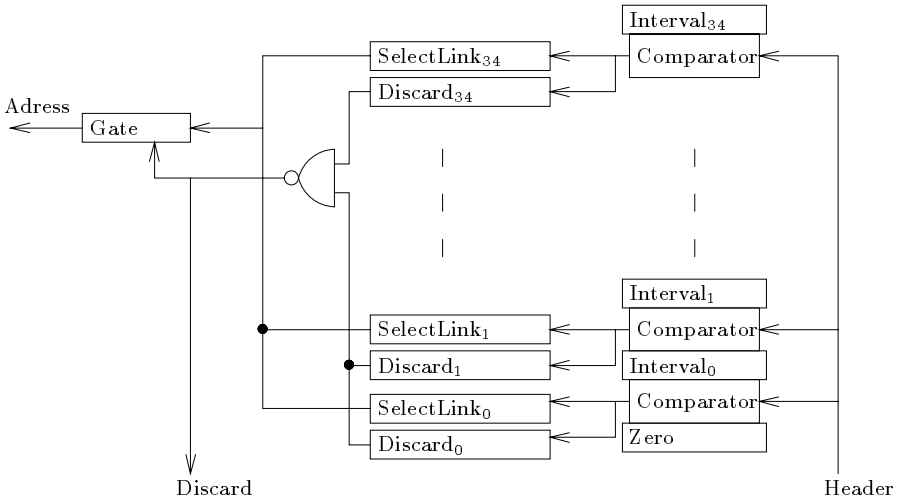


FIG. 2.9 - Registre du sélecteur d'intervalle

Un drapeau **Discard**$_n$ est associé à chaque intervalle permettant d'indiquer quel intervalle joue le rôle de *seuil* pour le routage universel. Si un en-tête appartient à un intervalle dont le drapeau est positionné, un signal *Discard* est envoyé au buffer d'en-tête (Fig. 2.7) afin que ce dernier le supprime. Si aucun drapeau n'est positionné, le mécanisme de seuil (routage universel) n'est pas disponible.

L'implantation d'un *routage adaptatif groupé* s'effectue en regroupant les liens du C104 en sous-groupes via un registre **Group** de 32 bits comme le montre la figure 2.8. Chaque bit de ce registre correspond à un lien.

2.3.2 Les liens de contrôle

Les deux liens *CLink0* et *CLink1* (Cf. figure 2.1) du C104 permettent de créer un réseau de contrôle pouvant fonctionner indépendamment du réseau de communication. Ces liens vont permettre de configurer les différents registres décrits dans la section précédente, de récupérer des erreurs pouvant se produire sur un routeur, de communiquer avec le *command processor* (Fig. 2.7). Ces deux liens sont bidirectionnels, et utilisent le même protocole de communication que le *DS-LINK*. On peut donc connecter un C104 par son lien de contrôle à un lien de communication d'un T9000 comme le montre la figure 2.10. Toutes les communications avec le processeur de contrôle d'un C104 se font par son lien CLink0, le lien CLink1 servant de *daisy-chain* (si une commande ne concerne pas le C104, ce dernier transmet la commande sur le lien CLink1). Il est donc nécessaire que chaque noeud possède une adresse. On peut donc connecter simplement (par une chaîne ou un arbre) le réseau de contrôle. Les liens de contrôle permettent,

via des messages de commande, de :

- démarrer le C104.
- initialiser l'identificateur d'un C104
- connaître l'identificateur d'un C104
- lire et écrire dans les registres de configuration (*CPeek* et *CPoke*).

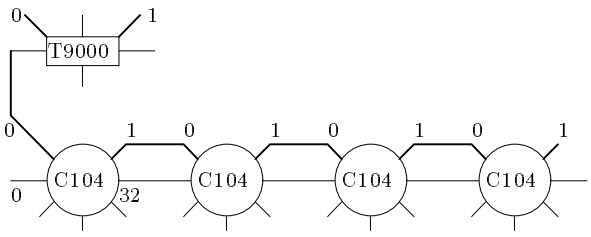


FIG. 2.10 - Exemple de réseau de contrôle en chaîne (*daisy-chain*) avec les liens de contrôle du C104.

2.3.3 Outil de configuration et initialisation

La configuration et l'initialisation du réseau (C104 et T9000) va donc se faire de manière logicielle. Par initialisation il faut entendre positionnement des différents registre des C104 et des T9000 d'un réseau. Les outils de configuration permettent de spécifier le *harware* et le *software*. A partir de ces descriptions, le "configureur" va générer un fichier de configuration qui sera utilisé pour initialiser et charger le réseau de T9000 et de C104 via les liens de contrôle. En fin d'initialisation, les T9000 seront dans un mode prêt à recevoir des données sur leurs liens de communication.

La description du harware se fait par un *Network Description Language* (NDL) qui permet :

- la déclaration de processeurs, de routeurs, et leurs interconnexions.
- la spécification des attributs pour les C104 (bornes des intervalles, mode de grignotage,...).
- la vérification de non-interblocage du système.

Le fichier NDL pour un système donné sera probablement fourni par le constructeur mais, il peut être utile de pouvoir modifier sa configuration de base.

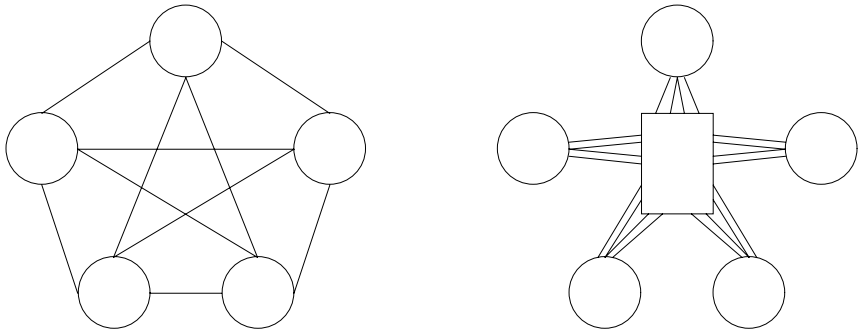
La description du software consiste à spécifier le code objet pour chaque processus et la façon de charger ces processus sur les processeurs (*mapping*).

Ces outils vont donc permettre d'initialiser le réseau. Une fois cette initialisation faite, les exécutables pourront être chargés d'une façon similaire à celle utilisée dans un réseau de T8XX.

2.4 Conclusion

L'intérêt du routeur C104 est qu'il permet de minimiser le coût des communications non locales qui requièrent des algorithmes de communication complexes, gaspillent de la bande passante et du temps CPU nécessaire par le *software*. Le C104 permet d'effectuer la scission entre le *software* et le *harware* :

- Un programme arbitraire peut tourner sur une machine quelconque.
- Un programme tournant sur un nombre quelconque de transputers peut s'exécuter sur un seul.
- Un programme tournant sur plusieurs transputers (via un ou plusieurs routeur) peut tourner sur un nombre réduit de transputers.
- Un programme tournant sur des transputers (via un réseau de routers) peut tourner sur le même nombre de transputers mais avec un topologie différentes.



(a) : 5 transputeurs totalement interconnectés pour une application spécifique.

(b) : La même topologie avec un seul C104

FIG. 2.11 - Exemples de petits systèmes

Même si on imagine un petit système avec seulement 5 transputers totalement connecté l'utilisation d'un C104 accroît considérablement la bande passante puisque tout couple de processeurs est connecté par quatre chemins disjoints comme le montre la figure 2.11. De plus, des entrées/sorties avec l'extérieur peuvent être effectuées et le le debuggage facilité.

Conclusion

Nous avons essayé de faire dans ce rapport un tour d'horizon des futures capacités du processeur T9000 et du processeur de routage C104.

Les différentes innovations qu'intègre le nouveau transputer T9000 d'INMOS en font un processeur général, mais pouvant être dédié aux calculs numériques et étant capable de gérer de façon *hardware* des communications via des canaux virtuels grâce à son processeur de communication intégré.

La compilation destinée à des processeurs généraux est assez bien maîtrisée. Cependant, la gestion du parallélisme de différentes unités ainsi que l'utilisation optimale des différents pipelines sont beaucoup plus ardues et encore assez mal connues. Le T9000 va certainement susciter de nombreux travaux à ce sujet de part son module de groupage d'instruction. Les compilateurs vont devoir essayer d'utiliser au maximum les différentes possibilités de parallélisme qu'offre ce nouveau transputer.

Le processeur de routage C104 qui est associé au T9000 semble lui aussi très prometteur. Son mode de routage *wormhole*, son faible temps de latence ainsi que la possibilité de faire du routage universel ou adaptatif vont en faire un routeur idéal pour la construction de machines massivement parallèles.

Bibliographie

[1] C. G. Bell, *RISC: Back To The Future*, Datamation, (1987).

[2] J. C. Berthillier, *Machines RISC: Des Prototypes Aux Superscalaires*, Publications Scientifiques et Techniques d'IBM France, 3 (1992), pp. 9–37.

[3] H. P. Charles, *Le Processeur i860*, Tech. Rep. 90-02, LIP-IMAG, 1990.

[4] F. Desprez, *Algèbre Linéaire sur Tnode*, Master's thesis, LIP/ENS-Lyon, June 1990.

[5] F. Desprez and B. Tourancheau, *Modélisation des Performances des Communications sur le Tnode avec le Logical System Transputer Toolset*, La lettre du Transputer et des calculateurs distribués, (1990), pp. 65–72.

[6] B. Glass, *SPARC Revealed*, Byte, (1991).

[7] F. Guidec, *Performances des communications sur le tnode*, La lettre du Transputer et des calculateurs distribués - Septembre, (1990).

[8] C. A. R. Hoare, *C.S.P.: Communicating Sequential Processes*, Prentice Hall, 1985.

[9] Inmos, *The T9000 Products - Overview Manual*, SGS-THOMSON - INMOS, 1991.

[10] J. v. Leeuwen and R. B. Tan, *Interval routing*, The Computer Journal, 30 (1987), pp. 298–306.

[11] B. Ryan, *Built For Speed*, Byte, (1992).

[12] N. Santoro and R. Khatib, *Labelling and implicit routing in networks*, The Computer Journal, 28 (1985), pp. 5–8.

[13] R. Sheperd, *Next Generation of Transputers and Beyond - 2: T9000: superscalar transputer*, in 1991 CERN School of Computing, C. E. O. for Nuclear Research, ed., vol. 92-02, 1992, pp. 309–322.

[14] R. M. Stein, *T800 and Counting*, Byte, (1988).

[15] L. G. Valiant, *A scheme for fast parallel communication*, SIAM J. COMPUT., 11 (1982), pp. 350–361.

[16] P. Wayner, *VLIW: Heir to RISC?*, Byte, (1989).

[17] R. Weiss, *Third-generation RISC Processors*, EDN, (1992).

[18] D. Weyand, *Architecture RISC: Une Programmation un peu Spéciale*, Minis et Micros, (1990).

[19] R. Wilson, *Applications Determine the choice of RISC or CISC.*, Computer Design, (1989).

Table des matières

Table des figures