HAL
open science

Schemas de communications globales dans les reseaux de processeurs : application a la grille torique

Philippe Michallon

To cite this version:

Philippe Michallon. Schemas de communications globales dans les reseaux de processeurs : application a la grille torique. Réseaux et télécommunications [cs.NI]. Institut National Polytechnique de Grenoble - INPG, 1994. Français. NNT : . tel-00005105

HAL Id: tel-00005105

https://theses.hal.science/tel-00005105

Submitted on 25 Feb 2004

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée par

Philippe MICHALLON

pour obtenir le titre de DOCTEUR

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(Arrêté ministériel du 30 Mars 1992)

Spécialité : **Mathématiques Appliquées**

==========

SCHEMAS DE COMMUNICATIONS GLOBALES DANS LES RESEAUX DE PROCESSEURS : APPLICATION A LA GRILLE TORIQUE

==========

Date de soutenance : 25 frévrier 1994

Composition du jury :

Président	Guy	MAZARE
Rapporteurs	Jean-Claude	BERMOND
	Michel	COSNARD
	Yousef	SAAD
Examinateurs	Denis	TRYSTRAM
	Gilles	VILLARD

Thèse préparée au sein du
Laboratoire de Modélisation et de Calcul

THESE

présentée par

Philippe MICHALLON

pour obtenir le titre de DOCTEUR

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(Arrêté ministériel du 30 Mars 1992)

Spécialité : **Mathématiques Appliquées**

==========

SCHEMAS DE COMMUNICATIONS GLOBALES DANS LES RESEAUX DE PROCESSEURS : APPLICATION A LA GRILLE TORIQUE

==========

Date de soutenance : 25 frévrier 1994

Composition du jury :

Président	Guy	MAZARE
Rapporteurs	Jean-Claude	BERMOND
	Michel	COSNARD
	Yousef	SAAD
Examinateurs	Denis	TRYSTRAM
	Gilles	VILLARD

Thèse préparée au sein du
Laboratoire de Modélisation et de Calcul

A mes parents,
A ma sœur,
Et à tous ceux que j'aime..

Remerciements

Cette thèse n'aurait pas vu le jour sans un certain nombre de personnes que je tiens à remercier ici.

Guy Masare pour m'avoir fait l'honneur de présider le jury.

Jean-Claude Bermond et Michel Cosnard pour avoir accepté de juger ce travail malgré des emplois du temps très chargés. Je tiens à les remercier tous les deux, pour leurs remarques qui m'on permis de corriger ou de reprendre de nombreux points de cette thèse.

Je remercie tout particulièrement Youcef Saad pour avoir accepté d'effectuer un rapport malgré les contraintes temporelles.

Gilles Villard pour ses nombreux conseils au cours de ces trois années, et son esprit de synthèse qui m'on permis de ne pas m'égarer en cours de chemin.

Et surtout celui qui est à l'origine de ce travail, mon directeur de thèse, Denis Trystram, sans qui je n'aurais jamais découvert le parallélisme. Je le remercie pour avoir su guider mes recherches durant ces années. Malheureusement pour lui, il a échoué dans ses tentatives de conversion au yoga et à son fameux régime de "pépins de pomme". Chunty! Chunty! tu réussiras avec d'autres thèsards !!

Je tiens également à remercier toutes les personnes de RUMEUR avec qui j'ai participé à la rédaction du LIVRE (vu le temps consacré, il faut au moins des majuscules !) et à l'école d'été : Anne, Dominique, Marie-Claude, André, Emmanuel, Jean-Claude, Denis, Michel, Pierre. Pour la prochaine école je propose les Bahamas, qu'en pensez-vous?

Je tiens tout particulièrement à remercier Jean-Claude avec qui j'ai travaillé, pour m'avoir appris la rigueur, tout en travaillant dans la bonne humeur de la crèche d'I3S.

Michel avec qui j'ai partagé la rédaction et la gestion du LIVRE et surtout la saturation du réseau national (mode d'emploi : envoyer par *mail* un fichier de plusieurs mégas à plusieurs destinataires dispersés aux quatre coins de la France. Le faire de préférence la veille d'un long *week-end*) et de pas mal de délire autour d'une Guiness.

Je remercie également les autres membres du groupe RUMEUR pour leurs discussions et leur bonne humeur. Mention spéciale à Dominique l'œnologue du groupe.

Je tiens à remercier tous les membres des équipes du laboratoire : Brigitte, Françoise, les Jacques, Jean-Marc, Jean-Louis, Abdel, et j'en oublie... pour leur accueil, leurs conseils et tout ce qui fait la remarquable ambiance de ce groupe.

Je remercie très vivement Nathalie et Christophe pour leurs nombreuses relectures et corrections. Sans eux je n'aurais jamais pu finir avant le départ de mes rapporteurs pour le Chili.

Au cours de cette thèse, j'ai croisé plusieurs générations de thèsards.

Honneur aux plus anciens, Isabelle, Denis (dit DD), Didier (dit la Marmotte), Jean-Yves (capable de vendre un parapluie à un Touhareg au milieu de Sahara), Yvan (dit Nounouille), François (dit Ptigas). Ensembles nous avons passé de nombreuses heures de rigolade autour de boites de *Coca-Cola* et devant les rayons de la *FNAC*. C'était la grande époque des Mickeys.

Puis il y eut ma génération de thèsard. Commençons galanterie oblige par les demoiselles et plus spécialement par ma voisine de bureau. Bravo Nathalie, tu as réussi à me supporter près de quatre ans dans le même bureau !! Merci aussi pour tes délicieuses mousses au chocolat et tes recettes de gâteau.

Cécile avec qui j'ai beaucoup discuté, qui me remontait le moral quand celui-ci était au plus bas et qui chemin faisant à rencontrer Yves le bidouilleur fou du labo.

Sans oublier Laurent (le drômois) le plus cool du labo, Jao Paulo, Paulo, Ricardo (le trio brésilien du labo) pour les nombreuses discussions tant scientifiques que sportives (football et formule 1).

Il y a aussi Frédéric (dit Fred ou le cha..e! non je ne l'ai pas dit), Thierry (dit Titou), les deux sportifs du labo. Si vous tenez à rester entier éviter de les suivre en ski, escalade, etc. même en randonnée pédestre ils sont dangereux. Leur seul but : trouver la ligne de plus grande pente et la suivre. Christophe (dit Xtof), le plus calme de tous et le plus discret, qui est toujours prêt à rendre service, et Yannick (dit Nounours) ; méfiez-vous qu'en même de l'eau qui dort !!

Fréderic et Christophe font parti comme Pascal (surnommé Hypolite par Eveline), Laurent (dit Lolo) des thèsards de Denis. Alors méfiez-vous du Chef, ne vous laissez pas convertir à son régime. Pour Pascal et Laurent je ne me fais pas de souci, mais Fred et Xtof tenaient bon, plus qu'un an !!

Au cours de ces années c'est formé un trio infernal constitué par Pascal, Laurent et moi-même.

Avant de connaître Pascal je détestais la bière, mais il m'a fait découvrir les bières de chez lui, et ce fut le déclic. De plus, il m'a fait découvrir son fameux steak sauce au poivre accompagné de frites et de compote de pommes.

Laurent me fit découvrir la cuisine auvergnate et la passion des jeux de rôles. De plus nous partageons la même passion pour les *verts* de St Etienne. En parlant de vers, c'est quand qu'on va la pêche?

Ensemble nous avons monté des expéditions à Prague, aux Pays-Bas à Namur, en Corse, à Romes qui ont toutes laissé des traces indélébiles dans nos mémoires. A quand la prochaine virée les copains?

Et la dernière génération de thèsard : Evelyne, Ines, Alain, Eric, Gabriel, Michel que je remercie pour son aide et son dévouement (je te dois toujours deux bières!), et Alex qui ma fait piquer des crises de nerfs en installant de nouvelles versions en cours de rédaction mais qui était toujours là pour débloquer la situation.

Enfin comment ne pas dire merci ici à un compagnon de route : Michel le lyonnais que je connais depuis le DEUG (je te supporte depuis si longtemps??). Merci pour les doses de café à la Kafet du LIP et pour tes cotisations à la Kafet du labo.

Merci aussi aux copains de toujours : Marie-Christine et Zeph pour leur soutien.

Table des matières

1 Des circuits VLSI aux machines parallèles **5**
1.1 Introduction . . . 5
1.2 Limites technologiques des VLSI . . . 6
1.2.1 Circuits intégrés . . . 6
1.2.2 Contraintes sur le nombre de liens . . . 7
1.2.3 Densité du réseau et bande passante globale . . . 9
1.3 Nécessité d'une mémoire distribuée . . . 9
1.3.1 Les machines parallèles à mémoire distribuée . . . 10
1.3.2 Modélisation d'un nœud . . . 10
1.3.3 Les différents types de réseaux . . . 12
1.4 Architectures des processeurs . . . 12
1.5 Performances des machines . . . 14
1.6 Classification des différents types de machines . . . 18
1.7 Historique des machines parallèles . . . 21
1.8 Description des composants . . . 23
1.8.1 Les processeurs . . . 23
1.8.2 Les processeurs avec facilités de communication . . . 25
1.9 Les machines parallèles actuelles . . . 28
1.9.1 La dernière génération de machines MIMD . . . 28
1.9.2 La dernière génération de machines SIMD . . . 38
1.9.3 Les grandes lignes actuelles . . . 42
1.10 Perspectives . . . 43

2 Modèles de communications et topologies **49**
2.1 Introduction . . . 49
2.2 Communications dans un nœud . . . 50
2.2.1 Gestion des messages . . . 50
2.2.2 Les contraintes de communication . . . 50
2.3 Différents modes de commutation . . . 52
2.3.1 Commutation de circuit (*circuit-switched*) . . . 52
2.3.2 Commutation de messages (*store-and-forward*) . . . 53
2.3.3 Commutation de paquets (*packet-switched*) . . . 54
2.3.4 Routage *wormhole* . . . 55
2.3.5 *Virtual-cut-through* . . . 56
2.3.6 Le mode transparence . . . 57
2.4 Modélisation du temps de communication . . . 57
2.4.1 Modèle temps constant . . . 57
2.4.2 Modèle temps linéaire . . . 58
2.4.3 Les autres modèles . . . 59
2.5 Comparaison des principaux modes de routage . . . 60
2.5.1 La technique du *pipeline* . . . 61

2.5.2 La technique des chemins disjoints . . . 62
2.6 Description des principales topologies . . . 62
2.6.1 Les hypercubes . . . 63
2.6.2 Les grilles . . . 64
2.6.3 Les grilles toriques . . . 64
2.6.4 Les de Bruijn . . . 65
2.7 Conclusion . . . 66

3 Communications globales **69**
3.1 Introduction . . . 69
3.2 Les Bibliothèques de calculs et de communications . . . 70
3.2.1 Les bibliothèques d'algèbre linéaire . . . 70
3.2.2 Les bibliothèques de communications . . . 71
3.3 Les schémas de communications globales . . . 72
3.4 Utilisation des schémas de communications globales . . . 72
3.4.1 L'élimination de Gauss . . . 73
3.4.2 Le gradient conjugué . . . 74
3.5 Conclusion . . . 78

4 Diffusion dans les grilles toriques **81**
4.1 Notations et modèle de communication . . . 82
4.2 Les outils de la diffusion . . . 82
4.3 Une première solution . . . 84
4.4 Des arbres de recouvrement de profondeur minimale . . . 85
4.4.1 Le principe de base . . . 86
4.4.2 Cas 1:4k+1 . . . 88
4.4.3 Cas 2:4k+3 . . . 93
4.4.4 Cas 3:4k . . . 95
4.4.5 Cas 4:4k+2 . . . 96
4.4.6 Etude théorique des arbres . . . 98
4.4.7 Algorithme de diffusion . . . 103
4.5 Généralisation . . . 104
4.5.1 Les solutions existantes . . . 106
4.5.2 Quelques améliorations . . . 108
4.5.3 Construction d'arbres de profondeur minimale . . . 109
4.6 Conclusion . . . 113

5 Communications globales dans les grilles toriques **115**
5.1 L'échange total . . . 115
5.1.1 Décomposition de la grille torique . . . 116
5.1.2 Les chemins hamiltoniens . . . 120
5.1.3 Stratégie gloutonne . . . 121
5.2 La distribution . . . 124
5.2.1 Application aux grilles toriques . . . 126
5.3 Le regroupement . . . 127
5.4 La multidistribution . . . 131
5.4.1 Application aux grilles toriques . . . 132
5.5 Synthèse des résultats . . . 133
5.6 Conclusion . . . 134

6 Expérimentations sur machine à base de Transputers **137**
6.1 Introduction . . . 137
6.2 Présentation du MégaNode . . . 138
6.2.1 Les machines SuperNode . . . 138
6.2.2 Le MégaNode . . . 138
6.2.3 L'environnement logiciel . . . 139
6.3 Modélisation des communications . . . 140
6.4 Description des algorithmes de diffusion . . . 145
6.5 Expérimentations . . . 148
6.6 Conclusion . . . 151

7 Recouvrements calculs/communications **153**
7.1 Introduction . . . 153
7.2 Pourquoi faire du recouvrement? . . . 153
7.3 Problème du choix de la taille des paquets . . . 156
7.3.1 Analyse du produit matrice-vecteur . . . 157
7.3.2 Description des algorithmes . . . 159
7.3.3 Calcul du temps d'exécution . . . 166
7.3.4 Adaptation au lien *full-duplex* . . . 166
7.3.5 Simulations . . . 169
7.4 Conclusion . . . 171

8 Communications globales en commutation de circuit **173**
8.1 Résultat préliminaire . . . 173
8.2 La diffusion . . . 175
8.2.1 Diffusion dans l'anneau . . . 175
8.2.2 Diffusion dans la grille torique . . . 177
8.3 L'échange total . . . 179
8.3.1 Coût minimal suivant chaque paramètre . . . 179
8.3.2 Différents algorithmes . . . 179
8.4 Comparaison des différents algorithmes . . . 182
8.5 Conclusion . . . 185

9 Conclusion **187**

A A **189**

Présentation de la thèse

Le parallélisme est né d'une idée très simple : si diverses parties d'un algorithme sont indépendantes et si on dispose de plusieurs unités de traitement, alors ces parties peuvent être exécutées simultanément sur les diverses unités.

Dès le début de l'informatique, les concepteurs d'ordinateurs ont eu cette idée, mais les contraintes matérielles de l'époque ne leur permettaient pas de l'exploiter. Dans les années 80, le développement de nouvelles technologies ont permis de construire les premières machines multiprocesseurs vraiment utilisables. Deux types de machines sont apparus, que l'on distingue suivant l'organisation de la mémoire : partagée ou distribuée. L'alternative mémoire distribuée est sans aucun doute le meilleur compromis performance/prix et permet d'envisager de très grands nombres de processeurs. Sur ce type d'architecture, chaque processeur dispose de son propre code et de ses propres données. Les échanges de données entre les processeurs se font par envoi de messages sur le réseau d'interconnexion qui relie les processeurs.

Les algorithmes parallèles peuvent être alors décomposés en une succession de phases de calcul et de phases de communication. Mais les phases de communication sont généralement très coûteuses, et empêchent les utilisateurs d'obtenir les puissances de crête des machines. Afin de réduire les temps d'exécutions, il est donc important de disposer de procédures de communications efficaces. Toutefois, la réalisation de telles bibliothèques est très liée à l'architecture du réseau d'interconnexion et aux modes de communication utilisés pour réaliser ces échanges.

Dans le même temps, de nombreuses bibliothèques de calcul se sont développées augmentant encore les performances des phases de calculs. Afin de ne pas réduire à néant les performances obtenues par les noyaux de calcul, en gardant des temps d'attente de données trop prohibitifs, une bonne solution est d'anticiper les mouvements de données afin de recouvrir les communications par du calcul.

Le but de cette thèse est donc d'abord d'étudier les schémas de communication les plus utilisés, et plus particulièrement la diffusion, pour en proposer des implémentations efficaces sur des machines parallèles dont le réseau d'interconnexion est une grille torique. Nous nous intéressons ensuite au problème du recouvrement calcul/communication sur l'exemple du produit matrice-vecteur.

Enfin nous abordons l'extension à d'autres modèles de communication correspondant aux machines actuelles.

Les deux premiers chapitres sont des synthèses sur les machines parallèles actuelles, et sur les protocoles de communication utilisés dans ces machines. Ils sont largement inspirés des textes écrits en collaboration avec Michel Syska, pour l'école d'été RUMEUR (1992) et complété pour le livre [3].

Plus précisément, le premier chapitre a pour but de présenter les orientations prises en matière d'architecture parallèle, et de les justifier. Les limites technologiques des circuits VLSI, les contraintes de fabrication et de coût ont obligé les constructeurs de machines parallèles à faire des choix entre mémoire partagée et mémoire distribuée, ou encore dans leurs réseaux d'interconnexion. Nous faisons ressortir ces choix dans la présentation des processeurs et des machines parallèles qui sont commercialisés à ce jour. Il en ressort que les machines à mémoire distribuée de type MIMD semblent être la solution offrant le meilleur compromis coût/puissance et que les réseaux d'interconnexion multi-étages semblent être une solution d'avenir. Désormais les grands constructeurs ont pour objectif de construire des machines capables d'effectuer mille milliards d'opérations par seconde (le TeraFlops).

Dans le deuxième chapitre, nous présentons les différents protocoles de communication utilisés dans les machines parallèles ainsi que leur modélisation. Si il y a encore deux ans la commutation de message *store and forward* était utilisée dans la majorité des machines, de nos jours ce sont la commutation de circuit et le routage *wormhole* qui se retrouvent le plus fréquemment. En *wormhole* les messages traversent les processeurs intermédiaires sans y être stockés comme en commutation de message, d'où la nécessité de modéliser différemment le temps d'une communication. Lorsqu'on réalise des communications entre deux ou plusieurs processeurs il faut aussi tenir compte de la topologie du réseau d'interconnexion. En effet, chaque réseau possède ses propres paramètres comme le degré, le diamètre etc.

Dans le chapitre trois nous présentons les principaux schémas de communications globales et nous montrons sur deux exemples la nécessité de tels schémas. En effet, pour paralléliser une application numérique, une bonne démarche consiste à répartir ses données initiales, identifier les procédures de calculs locaux et effectuer les rencontres entre les données en utilisant des schémas de communication.

Le chapitre quatre est consacré à l'étude de la diffusion dans les grilles toriques dans le mode commutation de message. Pour cela on utilise une modélisation *temps linéaire* du coût des communications. La première partie de ce chapitre traite le cas de la diffusion sous l'hypothèse de liens *full-duplex*. Ce problème est résolu en construisant des arbres de recouvrement à arcs disjoints. Pour les

grilles toriques carrées, nous présentons une nouvelle famille de quatre arbres de recouvrement à arcs disjoints dont la profondeur est minimale. Ce résultat a fait l'objet d'un article [6]. Ces arbres étant arcs disjoints, ils permettent d'obtenir une diffusion pipeline en temps optimal. Nous généralisons ensuite ce résultat au cas des grilles toriques rectangulaires. Dans la deuxième partie de ce chapitre nous étendons notre étude au cas des grilles torique possédant des liens *half-duplex*. Nous montrons que pour ce type de grilles il existe deux arbres de recouvrement disjoints de profondeur minimale qui pourront être utilisés pour effectuer des diffusions. Ceci fait l'objet de l'article [1].

Dans le chapitre cinq, nous présentons l'ensemble des schémas de communication globales dans la grille torique. Cette étude permet de présenter les meilleurs algorithmes connus pour chaque schéma. Les communications étant très coûteuses par rapport aux calculs, il est nécessaire de disposer de schémas de communications efficaces. Pour chaque schéma nous présentons l'algorithme avant d'en calculer le temps d'exécution. Ces résultats de synthèse ont été présentés au cours des « Rencontres sur le Parallélisme ». [5]

Lors des études théoriques nous supposons que les processeurs peuvent communiquer simultanément avec plusieurs voisins, ce qui est vrai sur la plupart des machines parallèles. Mais la théorie ne tient pas compte du fait que le coût des communications dépend du nombre de communications réalisées simultanément. Dans le chapitre six nous mettons en évidence ce phénomène et nous proposons une nouvelle modélisation des communications à partir d'une machine MegaNode. Nous appliquons ce nouveau modèle à la diffusion sur la grille torique et nous montrons que les résultats expérimentaux confirment les résultats théoriques. De plus comme cette machine est configurable, nous comparons les diffusions sur différentes topologies et nous en déduisons la meilleure topologie en fonction de la taille du message. Ces résultats font l'objet d'un article [4].

Le chapitre sept est consacré à l'étude du recouvrement des communications par du calcul. En effet on constate les processeurs restent souvent inoccupés à attendre des données, d'où l'idée lorsque cela est possible, d'anticiper les communications, en pipelinant l'envoi de petits messages. Toutefois la taille des messages à envoyer n'est pas facile à calculer. Elle dépend de la taille du problème, du nombre de processeurs dans le réseau, du coût d'une opération arithmétique et du coût d'une communication. Ceci est illustré par l'étude du produit matrice-vecteur sur un anneau de processeurs. L'article correspondant à ce résultat est soumis à publication [2].

Avant de conclure et de dégager les perspectives de ce travail, nous proposons, dans le chapitre huit, une étude du schéma d'échange total sur la grille torique en mode commutation de circuit. L'algorithme est inspiré du découpage récursif de la grille torique. Il permet d'obtenir un nombre d'étapes plus faible que les algorithmes traditionnels en *store and forward*, ce qui est intéressant lorsque

le *start-up* est prépondérant dans l'envoi d'un message. De plus cet algorithme fournit de bon résultat avec des messages de petites tailles.

Bibliographie

[1] J. C. Bermond, P. Michallon, and D. Trystram. Broadcasting in wraparound meshes with parallel monodirectionnal links. *Parallel Computing*, 18:639–648, 1992.

[2] L. Colombet, P. Michallon, and D. Trystram. Parallel matrix-vector product on a ring with a minimum of communications. *soumis à Parallel Computing.*

[3] Jean de Rumeur. *Communications dans les réseaux de processeurs.* collection ERI. Masson, 1994.

[4] P. Michallon and D. Trystram. Practical experiments of broadcasting algorithms on a configurable parallel computer. *à paraître dans Discrete Applied Mathematics, issue spéciale sur le broadcasting et le gossiping.*

[5] P. Michallon and D. Trystram. Résolution de schémas de communications globales sur réseau de processeurs en grille torique. In *5ème rencontres sur le parallélisme (Brest)*, 1993.

[6] P. Michallon, D. Trystram, and G. Villard. Optimal broadcasting algorithms on torus. *soumis à Journal of Distributed and Parallel Computing.*

Chapitre 1

Des circuits VLSI aux machines parallèles

1.1 Introduction

Le but de ce chapitre est de présenter de manière systématique la technologie et les techniques mises en œuvre dans les machines parallèles. Mais aussi de montrer combien il est nécessaire d'étudier les échanges de données et fournir des algorithmes de communication et de routage efficaces, vue la complexité de ces machines et le coût élevé de celles-ci par rapport au coût des calculs. En effet, l'augmentation constante de la demande de puissance de calcul a donné lieu lors de la dernière décennie au développement des ordinateurs parallèles. Dans ces machines parallèles, les constructeurs utilisent des processeurs qui sont de plus en plus puissants (c'est-à-dire qui calculent de plus en plus rapidement) et en plus grand nombre. Or la puissance de ces processeurs a augmenté plus vite que le débit des liens de communication, ce qui nécessite des algorithmes de communication efficaces afin de réduire les temps d'attente des processeurs. Ce chapitre est extrait du l'ouvrage collectif *communications dans les réseaux de processeurs* [24].

Après un rappel sur les limites technologiques de la conception de circuits intégrés, ainsi qu'une description des paramètres permettant de comparer et de classifier les machines parallèles, nous présenterons les principaux processeurs du moment, ainsi que les principales machines existantes et leurs différents réseaux d'interconnexion. Malgré la difficulté de prédire à long terme les réseaux et les technologies qui seront utilisés dans les futures machines parallèles, nous conclurons ce chapitre par les grandes lignes qui se dessinent pour les prochaines années.

1.2 Limites technologiques des VLSI

Les machines parallèles sont construites dans l'espace à trois dimensions, ce qui implique, pour les réseaux multidimensionnels comme les hypercubes, d'être plongés dans ces trois dimensions physiques. On distingue également plusieurs niveaux d'intégration des composants, ou pour certains niveaux, comme les circuits intégrés, seulement deux dimensions sont possibles (avec éventuellement plusieurs couches pour relâcher cette contrainte). Ainsi, la meilleure façon d'interconnecter entre eux des processeurs, dépend non seulement de lois mathématiques, mais aussi de contraintes dues aux procédés de fabrication disponibles.

1.2.1 Circuits intégrés

Les nœuds d'un réseau (processeurs ou organes de routage) sont réalisés en technologie VLSI (*Very Large Scale Integration* : technologie de construction de circuits intégrés [36, 61, 72]), et doivent respecter les contraintes de l'intégration en deux dimensions (planaire).

Un circuit intégré est en fait un graphe de connexions entre des composants élémentaires (transistors, amplificateurs opérationnels, diodes, résistances, etc.) gravés dans une plaquette de silicium. Certains types de circuit (circuits logiques) permettent de réaliser des fonctions booléennes. Un circuit intégré est appelé communément puce. Ces puces sont plus ou moins complexes suivant l'usage qu'il en est fait, ainsi on les retrouve aussi bien dans les cartes téléphoniques qu'au cœur même des ordinateurs.

Pour fonctionner, une puce a besoin d'être alimentée en courant et en données. De plus, s'il s'agit d'un processeur, il faut le connecter à une mémoire, ce qui nécessite beaucoup de liens depuis la puce. Or, celle-ci mesure de l'ordre d'un centimètre carré, d'où une impossibilité technique de la connecter sur une carte (circuit imprimé sur lequel on met des circuits intégrés) qui mesure de l'ordre de la centaine de centimètres carrés. Pour réaliser la connexion entre ces deux éléments d'échelles différentes, on fixe la puce sur un module (boîtier) en céramique disposant de pattes qui relient le module en céramique à la carte.

La figure 1.1 montre les trois niveaux « d'encapsulage » : la puce, le module et la carte [6].

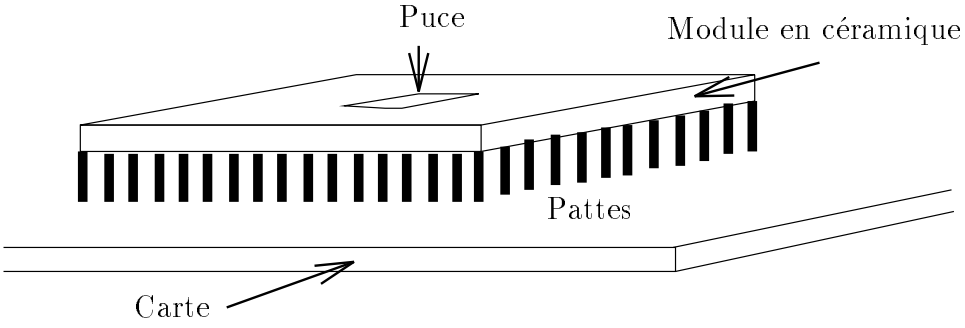


FIG. *1.1 : Différents niveaux d'encapsulage en VLSI*

Un nœud peut être constitué de plusieurs modules assemblés sur une même carte. On doit noter qu'il peut y avoir (rarement) plusieurs nœuds par puce ; on parle alors de *Multi-Chip Module*. Certaines technologies, comme la *Wafer Scale Integration*, rentrent aussi dans ce concept (sur une même tranche de silicium, on imprime un grand nombre de circuits identiques, qui peuvent communiquer avec leurs voisins. Ce nombre est limité par l'épaisseur des substrats réalisables).

Le réseau des nœuds doit lui aussi être conçu sur une surface à deux dimensions au niveau des cartes sur lesquelles sont posés les modules.

De nombreux articles ont été publiés sur le problème de la conception des circuits intégrés (graphe des connexions entre les cellules élémentaires [67]) sur une surface à deux dimensions la plus petite possible. Plus le dessin du circuit est compact, plus le délai écoulé entre l'entrée et la sortie des données (temps de calcul du circuit) est court. Ainsi, un graphe très complexe, qui nécessite une surface plus grande, peut s'avérer moins efficace qu'un graphe de calcul plus simple que l'on sait concevoir sur une surface plus petite. Un circuit de taille réduite consomme moins de puissance électrique, accepte une fréquence d'horloge plus élevée, est plus fiable, occupe moins de volume et pèse moins lourd. De plus, le coût de fabrication diminue avec la taille ! Ces considérations s'appliquent aux réseaux de processeurs des machines que nous présentons dans ce chapitre.

On peut consulter les articles de Valiant [78], Karchmer [52], Lengauer [57] pour une introduction à ces problèmes et pour des références complètes sur la théorie des VLSI.

1.2.2 Contraintes sur le nombre de liens

Il se pose également le problème de la connexion de ces puces entre elles. En effet, pour construire une machine parallèle on ne peut pas relier les processeurs comme on veut. Des contraintes physiques et matérielles nous empêchent de relier autant de processeurs que l'on veut à un seul, comme nous allons le voir maintenant.

Les puces comportent aujourd'hui environ trois millions de transistors disposés sur un peu plus d'un centimètre carré, alors que le nombre de pattes issues du module est beaucoup plus faible (par exemple, égal à 208 sur le T9000 d'Inmos). La plupart des processeurs sont destinés à équiper des stations de travail et les ordinateurs personnels, et n'offrent pas de gestion interne (*on chip*) des communications, le coût étant prohibitif. C'est le nombre de liens (de pattes) que l'on peut tirer d'un module et non les terminaisons (liens) de la puce qui limite l'interconnexion des VLSI, à cause de la différence d'échelle entre ces deux niveaux [75, p. 141]. Comme la surface des circuits doit être minimisée pour les raisons évoquées ci-dessus, la portion réservée aux communications externes ne

peut être augmentée sans entraîner des pertes de performances globales du circuit. Un transistor de technologie MOS (Metal Oxyde Semiconductor) peut changer d'état en $0,10$ ns, mais si l'on ajoute quelques centaines de microns sur sa ligne de connexion, ce délai est augmenté de plusieurs nanosecondes [69] !

De plus, la différence d'échelle entre puce et carte impose aussi l'amplification des signaux qui doivent sortir de la puce : le délai dû à cette amplification est du même ordre qu'une période de l'horloge interne du circuit [76].

Parmi ces pattes en nombre limité, une grande partie est réservée à la connexion avec la mémoire externe du processeur et une autre partie à la transmission des signaux utiles au système et à la synchronisation des processeurs entre eux.

Il reste donc un nombre très limité de pattes pour réaliser les liens de communication entre processeurs [19]. Ceci explique en partie la préférence pour les liens « série », où l'on transmet une suite de bits sur une patte, aux liens parallèles où l'on transmet des mots, mais sur un nombre de pattes égal à la taille du mot (sans compter les signaux de contrôle de la transmission : parité, accusé de réception, ...).

Il faut aussi trouver un équilibre entre le nombre de liens par nœud et la largeur (bande passante unitaire) de ces liens. La bande passante d'un lien correspond au nombre maximum d'octets transitant sur ce lien par unité de temps, ou par cycle d'horloge.

De façon plus concrète, vaut-il mieux construire une machine avec un réseau en hypercube et des liens d'un bit de large ou bien une machine avec un réseau en grille, et donc un diamètre (distance maximale entre deux nœuds quelconques du réseau) plus grand, mais avec des liens de 16 bits de large ?

Des études menées par Dally [22] montrent que sous certaines hypothèses, les réseaux en grille de faibles dimensions sont plus performants que les hypercubes pour un même nombre de processeurs. Pour cette comparaison, Dally normalise la taille des canaux de communication par rapport à la bissection (nombre minimum de liens qu'il faut couper pour partager le réseau en deux parties ayant le même nombre de nœuds) du réseau considéré. Il montre que, pour 256 processeurs, la grille à deux dimensions est optimale. Cependant, cette étude porte sur les grilles multidimensionnelles (*k-ary-n-cubes*) et ne prend pas en compte des réseaux moins courants comme les réseaux de De Bruijn ou les réseaux *cube-connected-cycles*, qui offrent pourtant un faible diamètre pour un degré (nombre de liens) constant.

Si l'on augmente considérablement le nombre de processeurs, le diamètre résultant va augmenter de façon défavorable dans le cas des grilles. Par exemple, pour un peu plus d'un million de processeurs (2^{20}) et un nombre de liens par processeur égal à 4, il faut traverser au plus 20 nœuds dans un De Bruijn $B(2,20)$, et 2048 dans une grille carrée 1024×1024 pour joindre deux nœuds quelconques !

Certains travaux récents ([15, 14, 73]) ont montré que la réalisation d'un réseau de processeurs avec une topologie de De Bruijn était possible. Un décodeur de codes de *Viterbi* a été réalisé dans le cadre de la mission Galileo pour la NASA.

Le réseau utilisé est un graphe de De Bruijn $B(2, 13)$ dont chaque nœud est un circuit arithmétique élémentaire du décodeur. Dans une réalisation plus générale, un nœud pourrait être un vrai processeur de calcul. D'ailleurs, on a affaire ici plus à une véritable machine MIMD qu'à un simple décodeur, car chaque circuit possède sa propre horloge et sa propre mémoire. La construction est modulaire, et est composée de 16 cartes. Chaque carte comporte 16 puces, une puce implantant 32 « papillons » (opérations élémentaires). Il suffirait de dupliquer les cartes pour obtenir un réseau plus grand. Pour ce qui est des connexions, 56 % sont internes (entre « papillons » de la même puce), 17 % sont sur une même carte (entre des puces reposant sur la même carte) et 27 % sont entre des cartes différentes. Si le réseau était une grille, on aurait de façon équivalente des valeurs de 82 %, 14 % et 4 %, qui sont meilleures dans l'absolu, mais pour décoder des données sur une grille, il y aurait beaucoup plus d'échanges de données. Dans les deux cas, on a le même degré pour les nœuds (nombre de liens).

Quoi qu'il en soit, les réseaux de faible degré semblent être les meilleurs candidats pour interconnecter un très grand nombre de processeurs dans une machine parallèle.

1.2.3 Densité du réseau et bande passante globale

Pour comparer l'efficacité des réseaux, il faut également tenir compte de la longueur des liens requise par leur implantation physique. Cette longueur intervient dans le délai de transmission d'un message entre deux nœuds voisins du réseau.

Pour des liens « très courts » on considère que le délai est constant ; si le lien est simplement « court », le délai est une fonction logarithmique de la longueur ; s'il est « long », le délai devient une fonction linéaire de cette longueur. Les notions de « très court », « court » et « long » dépendent de la technologie [21].

Les coûts de communication dépendent aussi de la densité des liens du réseau. Une mesure de cette densité des liens est donnée par une analyse à bissection constante [75, p. 143].

1.3 Nécessité d'une mémoire distribuée

Plus la mémoire est grande, plus le délai moyen d'accès aux données est important. La technique employée pour diminuer ce temps d'accès est basée sur l'utilisation de *caches* (un *cache* est une mémoire intermédiaire à accès très rapide et de faible taille, qui est placée entre le processeur et la mémoire. On parle aussi d'antémémoire) encapsulés avec le CPU (unité de traitement ou *Central Processing Unit*) dans la même puce. Cela est rendu possible par l'intégration d'un plus grand nombre de transistors dans les puces. Parfois, on utilise plusieurs niveaux de caches entre le processeur et la mémoire [44]. Cependant, la

taille adressable directement par un processeur reste faible si l'on veut garder un temps de cycle réduit. Cela implique l'utilisation d'une mémoire distribuée.

1.3.1 Les machines parallèles à mémoire distribuée

L'utilisation ou non de ce concept dans les machines parallèles a coupé les machines parallèles en deux familles : les machines à mémoire distribuée et les machines à mémoire partagée. Sur les machines à mémoire partagée, les processeurs accèdent tous à une mémoire commune, par l'intermédiaire du réseau d'interconnexion qui gère les accès concurrents à la mémoire. Au contraire, sur les machines à mémoire distribuée, chaque processeur possède sa propre mémoire, les échanges de données entre processeurs s'effectuant par échange de messages sur le réseau d'interconnexion.

Aujourd'hui, en raison des coûts trop élevés et de la complexité de réalisation des mémoires partagées, (il est très difficile de gérer une mémoire partagée à laquelle pourraient accéder plusieurs dizaines de processeurs), de nombreux constructeurs ont opté pour le « distribué ». Notons que ce choix pourrait être remis en cause en raison d'évolutions technologiques, mais semble le plus à même, pour l'heure, de répondre au défi que représente une vitesse de calcul du *TeraFlop*.

1.3.2 Modélisation d'un nœud

Les ordinateurs parallèles sont composés de plusieurs unités de calcul dupliquées pouvant participer à plusieurs applications différentes. De fait, les unités de calcul ne sont pas les seules à être dupliquées, toutes les autres unités nécessaires au fonctionnement d'un processeur peuvent l'être également. Nous appellerons *nœud*, tout ensemble d'unités (processeurs, registres, horloges, mémoire locale, etc.) qui est dupliqué au sein de la machine.

Les échanges de données entre les nœuds s'effectuent en utilisant les liens d'un réseau d'interconnexion, cela nécessite de trouver sur chacun des nœuds un ensemble d'unités nécessaires aux communications.

Un nœud pourra donc être constitué des éléments suivants :

- unité de calcul scalaire ;
- unité de calcul flottant (parfois vectorielles) ;
- unité de contrôle (séquenceur, etc.) ;
- mémoire locale ;
- horloge interne ;
- liens de communication ;
- unités de gestion des messages, etc.

Aujourd'hui il existe des nœuds encore plus complexes, intégrant des mémoires caches et un analyseur de performances. Celui-ci enregistre certains événements qui se déroulent sur le nœud (accès mémoire, communications, etc.). Ceci permet de visualiser sur un écran le déroulement du programme en chaque nœud du réseau. Les mémoires caches de données et d'instructions sont des organes de mémoire à accès très rapide, mais de faible capacité, dans lesquels sont stockées des informations très souvent utilisées ou que l'on va utiliser immédiatement. Ils servent de tampons entre la mémoire principale d'accès plus lent et le processeur plus rapide. Sans ces caches il faudrait attendre plusieurs cycles pour charger les données dans les registres du processeur.

Voici un exemple de type de nœud que l'on peut trouver sur les machines parallèles à mémoire distribuée.

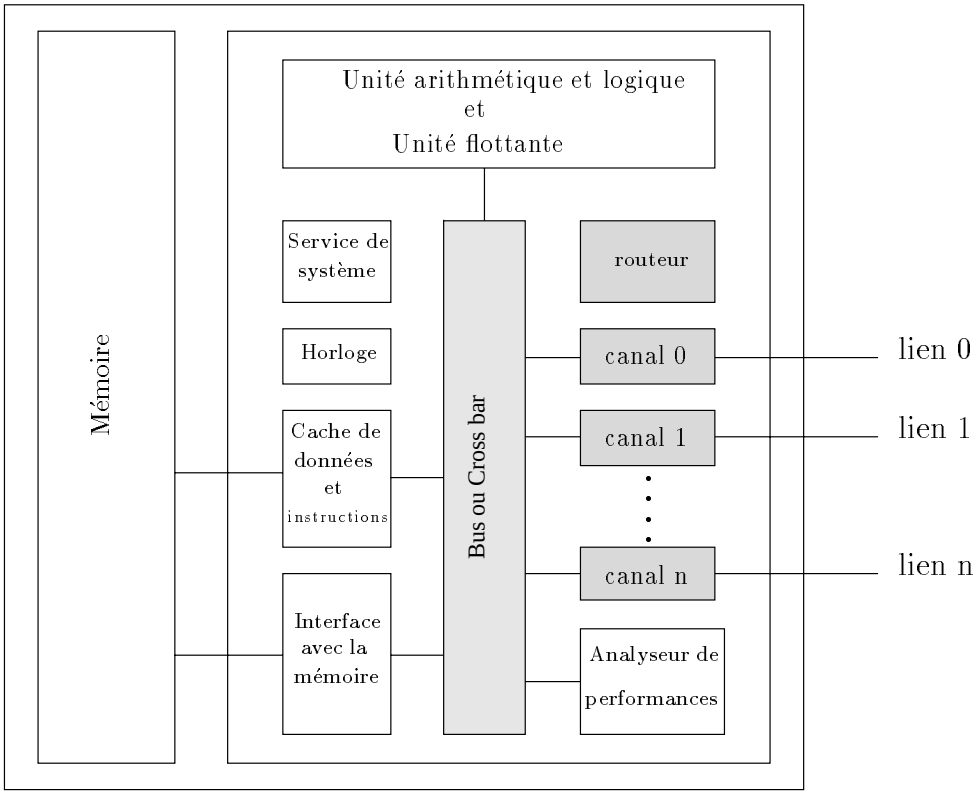


FIG. *1.2 : Synopsis de nœud.*

Par abus de langage, on assimile un nœud à un processeur, surtout lors de la description des algorithmes sur ces machines. Cependant un nœud peut comporter plusieurs processeurs (ces processeurs communiquant via leur mémoire commune, par exemple la Concerto et le Cray T3D), ou encore en simuler un grand nombre, on parle alors de processeurs virtuels.

1.3.3 Les différents types de réseaux

Pour réaliser les communications entre les nœuds, il existe deux types de réseaux :

- **Point-à-point :** chaque nœud peut communiquer directement avec ses voisins (qui sont les nœuds avec lesquels il est physiquement relié). Pour communiquer avec des nœuds à une distance supérieure, le message doit passer par une série de nœuds intermédiaires.

- **Multi-étages :** ces réseaux sont constitués de nœuds tels que ceux décrits ci-dessus, et d'unités dédiées aux communications. Ces dernières sont généralement des commutateurs (*switches*). Elles servent soit à acheminer les messages entre les unités de calculs dans les machines à mémoire distribués, soit à acheminer les messages entre les unités de calculs et la mémoire dans les machines à mémoire partagée.

Il existe d'autres types de réseaux. En particulier, les réseaux *configurables*, où la topologie n'est pas figée et peut être définie en fonction de l'application par l'utilisateur. Si de plus, cette topologie est modifiable en cours d'exécution du programme, elle est dite *reconfigurable*. Dans les deux cas, une fois la machine configurée, elle se comporte comme une architecture point-à-point. Les temps de configuration restant encore élevés, ce type de machine n'a pas été retenu actuellement pour la conception de machines massivement parallèles.

1.4 Architectures des processeurs

Le temps moyen d'exécution T d'un programme sur un processeur est donné par : $NI \times NC \times TC$ [37], où NI est le nombre d'instructions machine qui constituent le programme, NC le nombre de cycles d'horloge nécessaires en moyenne pour exécuter une instruction, et TC le temps de cycle de l'horloge du processeur.

Lors de la conception d'une architecture, on vise à minimiser cette fonction. Les types d'architectures les plus employées dans les processeurs actuels sont : RISC, *pipeline*, *superpipeline*, *superscalaire* et VLIW. Nous donnons dans la suite les caractéristiques principales de ces divers types d'architectures.

- **RISC** : les années 80 ont vu l'apparition des architectures RISC (*Reduced Instruction Set Computer*) [37]. L'objectif de ces architectures est de pouvoir exécuter une instruction par cycle ($NC = 1$). Dans les architectures de la génération précédente (CISC : *Complex Instruction Set Computer*), le jeu d'instructions proposé était très étendu, et ceci avait au moins deux conséquences :

  - les compilateurs avaient des problèmes pour exploiter au mieux ce jeu d'instructions (90 % du temps était consacré à l'utilisation d'un en-

semble d'instructions représentant seulement 10 % du jeu total d'instructions).

- la partie contrôle du processeur était microprogrammée, et utilisait une surface importante du composant (60 % dans le cas du 68000 de Motorola).

Au contraire, le RISC a un jeu d'instructions plus réduit, ce qui permet un contrôle câblé plus rapide (cela consiste en un automate à nombre fini d'états), et la surface ainsi libérée permet d'implanter des caches plus grands. De plus, les modes d'adressage plus simples (du type registre à registre au lieu de registre à mémoire) autorisent l'implantation de *pipelines* d'instructions.

- ***Pipeline*** : le principe consiste à décomposer les instructions en n phases de longueur égale à un cycle [53]. Ainsi, on peut exécuter n instructions simultanément. Pendant un cycle d'horloge, la phase k de l'instruction i est exécutée en même temps que la phase $k-1$ de l'instruction $i+1$, ... Par exemple, dans le cas du processeur MicroSparc de Sun, ces phases sont, pour une instruction simple (en un cycle) :

  1. lecture de l'instruction pointée par le compteur de programme ;
  2. décodage, lecture des opérandes et mise à jour du compteur de programme ;
  3. exécution de l'instruction par l'unité arithmétique et logique ;
  4. rangement du résultat.

  Chaque circuit qui exécute une phase d'une instruction est appelé étage du *pipeline*. Dans l'exemple décrit ci-dessus, le *pipeline* possède quatre étages. On parlera aussi d'un *pipeline* de profondeur quatre.

  Le bon fonctionnement du *pipeline* peut être perturbé par des interruptions, mais surtout par des conflits d'accès aux données, des dépendances de données ou des sauts dans le programme (résolus en partie par anticipation).

- ***Superpipeline*** : en subdivisant à l'extrême les phases élémentaires dans un *pipeline*, on obtient un *pipeline* plus profond dont chaque phase est plus fine, ce qui permet d'augmenter la vitesse d'horloge du processeur (par exemple l'Alpha de DEC).

- ***Superscalaire*** : ce type d'architecture se trouve sur les processeurs possédant plusieurs unités de traitement pouvant opérer simultanément. Typiquement un processeur *superscalaire* possède une unité entière et une unité flottante.

Une architecture superscalaire nécessite de fournir les instructions aux unités de traitement avec un débit suffisant pour exploiter pleinement le parallélisme. Ceci est automatique si chaque unité est lente, mais si les unités sont rapides (1 cycle par opération) il faut lire et décoder plusieurs instructions par cycle (sans compter la recherche en mémoire et le chargement des opérandes). Ceci peut être réalisé de différentes façons :

- 1) **VLIW** : dans le VLIW (*Very Long Instruction Word*) chaque mot d'instruction est divisé en champs, chaque champ codant l'instruction d'une unité. Cette technique est très en vogue dans les processeurs destinés au traitement du signal (par exemple les DSP). Ce type de technique correspond à une programmation très proche du matériel.
- 2) multiplication des caches et des ports internes : en utilisant un cache d'instructions, un bus interne élargi et des caches de données à accès multiples, il est possible de lire et de coder plusieurs instructions avec leurs arguments dans un seul cycle d'horloge. Par exemple, le T9000 lit sur 32 bits quatre instructions de 8 bits.

1.5 Performances des machines

Les performances de crête d'une machine sont obtenues lors de l'utilisation optimale de celle-ci, c'est-à-dire en utilisant au mieux les mémoires caches, les registres, les unités de calculs flottants, et en n'utilisant pas ou peu de communications. Les performances de crête sont impossibles à atteindre en pratique, la compilation d'applications réelles ne permettant pas l'utilisation optimale des ressources. Les compilateurs de langages de haut niveau (comme C, Fortran) donnent en général des performances bien en-dessous de celles annoncées par les constructeurs (c'est un phénomène bien connu pour le i860 ou l'Alpha), et de plus, la plupart des algorithmes parallèles nécessitent de nombreuses communications qui limitent les performances.

Mots clés pour l'étude des performances

- MFlops (*Million of Floating point operations per second*) : million d'opérations flottantes effectuées en une seconde (notons que M est aussi l'initiale de Mega). Actuellement, on mesure plutôt le nombre d'opérations en GFlops, ce qui correspond au milliard d'opérations flottantes par seconde (G pour Giga). La prochaine étape sera le TFlops qui correspondra au millier de milliards (ou billion) d'opérations flottantes par seconde (T pour Tera).

- MIPS (*Million of Instructions Per Second*) : million d'instructions réalisées en une seconde. De même, il existe le GIPS et le TIPS.

- Bande passante d'un lien : taux de transfert en octets par seconde. La bande passante d'un nœud est la somme des transferts réalisables de façon concurrente par tous ses liens. La bande passante d'un réseau est la somme de la bande passante de tous les liens. Ce débit se mesure en Megaoctets par seconde que l'on note Mo/s. On parlera aussi de Gigaoctets par seconde que l'on notera Go/s.

- MHz : unité de mesure de la fréquence de l'horloge interne du processeur ; son inverse, la période, est égale à la durée d'un cycle. Le coût d'une instruction est un nombre entier de cycles, sauf en *superscalaire* où l'on peut exécuter des instructions en parallèle. Par exemple, une fréquence d'horloge de 40 MHz correspond à un cycle de 25 nanosecondes.

- *Benchmark* : programme standard dont le but est de comparer les performances de différentes machines ou processeurs. Les principaux *benchmarks* présentés ci-dessous sont : *Whestone*, *Dhrystone*, les boucles de *Livermore*, *LINPACK*, *NAS kernel*, *SPEC*.

  - Le *benchmark Whestone* a été le premier, élaboré en 1976 au *National Physical Laboratory* (Grande-Bretagne). Il est constitué de plusieurs modules écrits en FORTRAN, chacun contenant des instructions de différents types (arithmétique sur des entiers, arithmétique sur des flottants, des instructions conditionnelles, des appels de fonctions, etc.) [81]. Ces modules sont pondérés, et exécutés plusieurs millions de fois, ce qui donne des résultats en MWIPS (*MegaWhestone Instructions Per Second*).

  - Le *benchmark Dhrystone* est une amélioration du Whestone. Il est apparu en 1984. Les résultats sont donnés en MegaDhrystone par seconde [81].

  - Les boucles de *Livermore* consistent en 24 boucles FORTRAN qui ont été réalisées au *Lawrence Livermore National Laboratory* (USA). Ces boucles sont constituées par des noyaux d'exécutions que l'on retrouve dans la majeure partie des applications numériques. Cela va des noyaux d'algèbre linéaire (produit de matrices, gradient conjugué, etc.) aux noyaux de tri ou de recherche [39]. Ces boucles sont utilisées pour évaluer les performances en MFlops des processeurs et des machines parallèles.

  - Le *benchmark LINPACK* (*LINear PACKage*) est une bibliothèque de procédures écrite en FORTRAN, facilement transportable, contenant les principales procédures d'algèbre linéaire [12] [27]. Cette bibliothèque a été réalisée à *Argonne National Laboratory* à partir de 1977 et deux procédures en ont été extraites afin de constituer une série

de tests [32]. Les performances sont alors exprimées en MFlops LINPACK, ce qui permet de comparer les performances des différents ordinateurs (Jack Dongarra a réalisé une telle étude, qu'il tient périodiquement à jour [31]).

- Le *NAS kernel benchmark* a été développé lors du programme *NAS* (*Numerical Aerodynamic Simulation*) au centre de recherche d'Ames de la NASA. Ce programme est constitué de sept noyaux de calcul (écrits en FORTRAN) utilisés par la NASA. Il y a des codes d'algèbre linéaire (comme le produit de matrices, la factorisation de Cholesky, l'élimination de Gauss, etc.), du traitement de signal (FFT bidimensionnelle) et des codes de turbulence (éléments finis). De plus, la NASA a défini quatre niveaux de test. Le niveau 0 où le code est porté sur la machine cible sans optimisation, le niveau 20 où on peut optimiser le code, sans modifier ou ajouter plus de 20 lignes de code, le niveau 50 où l'on peut mieux optimiser le code (modification ou ajoût d'au plus 50 lignes de code), et le niveau 1000 où l'on optimise l'application pour une machine (modification ou ajout d'un nombre illimité de lignes). En fait, ces différents niveaux permettent de différencier les performances du compilateur de celles du matériel. Si le compilateur est efficace, la différence entre le niveau 0 et le niveau 1000 sera faible. Les résultats de ce *benchmark* sont donnés en MFlops et en temps CPU [5].
- Le *benchmark SPECmark* a été créé par l'organisme SPEC (*System Performance Evaluation Consortium*) et les fabricants IBM et Hewlett-Packard en 1989. Puis en 1992, SPECint et SPECfp lui ont succédé. Le SPECmark est basé sur un ensemble standard d'applications réelles. Il est constitué de dix programmes de tests (six en FORTRAN et quatre en C), comme la simulation d'un réacteur nucléaire ou de circuits analogiques, ou des programmes évaluant les performances du système, comme la compilation d'une application donnée avec GNU C [42]. Les programmes sont exécutés sur la machine cible, avec mesure des temps d'exécution. Le rapport du temps d'exécution sur la machine cible à celui d'un DEC VAX 11/780 donne une mesure que l'on exprime en SPECmark (ou SPECmark 89). Mais le SPECmark a été souvent critiqué pour son manque de précision entre les différents types de mesures et pour le phénomène d'inflation qu'il subissait. En effet, les constructeurs se sont lancés dans la course au SPECmark en améliorant le compilateur de leurs machines afin que ces dernières réussissent parfaitement les tests [43]. C'est pour cela que SPEC a créé en 1992 le SPECint et le SPECfp. SPECint est composé de six *benchmarks* écrits en C représentant des applications de simulation de circuits logiques, de compression de données ou d'interpréteur Lisp. Il teste essentiellement les performances en calcul entier de l'unité centrale, alors que SPECfp mesure les capacités de calcul flottant. SPECfp

est constitué de quatorze *benchmarks* scientifiques ou techniques dont deux sont écrits en C et le reste en FORTRAN [43]. Les performances ainsi obtenues s'expriment en SPECint92 et SPECfp92.

Ces échelles permettent de comparer les différents processeurs et systèmes du marché. Ce *benchmark* est le plus utilisé actuellement, car il permet de tester des machines sur des applications concrètes.

Le numéro spécial *Benchmarking of high performance supercomputers* de *Parallel Computing* [34], fait un tour d'horizon de toutes les méthodes utilisées pour évaluer les performances des ordinateurs.

Un exemple pratique de calcul de performances

- Soit un processeur doté d'une fréquence d'horloge de 40 Mhz. Ce processeur dispose d'un additionneur et d'un multiplieur. Il faut trois cycles pour calculer une addition ou une multiplication.

- Soit deux matrices A et B de taille 100 par 100, dont on veut calculer le produit $A \times B$, noté C. L'algorithme est le suivant :

Algorithme 1.1 *Produit matriciel*

pour $i \leftarrow 1$ *à* 100 *faire*
    pour $j \leftarrow 1$ *à* 100 *faire*
        $C[i,j] \leftarrow 0;$
        pour $k \leftarrow 1$ *à* 100 *faire*
            $C[i,j] \leftarrow C[i,j] \ + \ A[i,k] \ \times \ B[k,j];$

Ce produit nécessite 10^6 multiplications et 10^6 additions.

Un cycle prend 25 nanosecondes, ce qui donne 75 nanosecondes pour une addition ou une multiplication.

Donc le temps total est de : $(75 \times 10^6 + 75 \times 10^6) \times 10^{-9} = 0,15$ seconde pour réaliser 2×10^6 opérations, ce qui fait 13 millions d'opérations par seconde soit 13 MFlops.

Notons que si l'on dispose d'un multiplieur et d'un additionneur *pipeline* et qu'on les utilise de façon conjointe, on peut avoir une multiplication plus une addition par cycle, ce qui permet d'obtenir de l'ordre de 80 millions d'opérations par seconde, soit 80 MFlops.

On notera que, dans les deux cas, on ne tient pas compte du temps d'accès à la mémoire, et dans le cas des architectures *pipelines*, du temps d'amorçage (temps de traversée du pipeline par le premier élément entré) des pipelines, ce qui est d'autant plus négligeable que la dimension du problème est grande.

1.6 Classification des différents types de machines

La grande diversité des machines parallèles [2] a donné lieu, de la part de Flynn, à une taxinomie [40] à laquelle on fait toujours référence même si aujourd'hui elle semble un peu dépassée. En effet cette classification ne prend en compte que deux critères : le type du flot d'instructions et le type du flot des données traitées par les processeurs élémentaires. Les flots sont soit simples (*single*) soit multiples (*multiple*). De ce fait on a du mal à classer certaines machines (par exemple celles à flot de données) et à distinguer les machines à mémoire distribuée des machines à mémoire globale.

Les ordinateurs parallèles qui nous concernent ici sont ceux à mémoire distribuée type SIMD (*Single Instruction Multiple Data flow*) et type MIMD (*Multiple Instruction Multiple Data flow*).

Notons que les machines de type SISD (*Single Instruction Single Data flow*) correspondent aux machines séquentielles (modèle de Von Neuman) et que les machines de type MISD (*Multiple Instruction Single Data flow*) peuvent être assimilées aux unités *pipeline*, en effet dans une telle unité plusieurs instructions sont effectuées simultanément sur un même flot de données.

- **SIMD** : sur ce type de machine, le programme s'exécute sur un frontal et les instructions traitant des données parallèles sont diffusées à tous les processeurs pour être appliquées aux données réparties dans les mémoires locales de ces processeurs. Ainsi, tous les processeurs effectuent la même instruction, mais sur des données différentes. Le fonctionnement de ce type de machines est synchrone.

  L'avantage des machines SIMD tient à ce que seules les unités de calcul doivent être dupliquées. En effet, une seule unité de contrôle suffit à décoder les instructions pour tous les processeurs (voir figure 1.3).

  Dans la pratique, de tels ordinateurs sont utilisés pour des problèmes réguliers, c'est-à-dire manipulation de vecteurs, de matrices, etc. La Connection Machine 2 de Thinking Machines Corporation, la MP-1 de MasPar ou les DAP (Digital Array Processors) de AMT et la Zephir de Wavetracer font partie de ce type de machines. Elles ont un point commun : assembler un grand nombre de processeurs simples (64 K processeurs 1 bit sur la CM-2, et 8K processeurs 4 bits pour la MasPar). C'est ce que l'on appelle un parallélisme à grain fin (le ratio nombre de données traitées / nombre de processeurs utilisés est proche de 1).

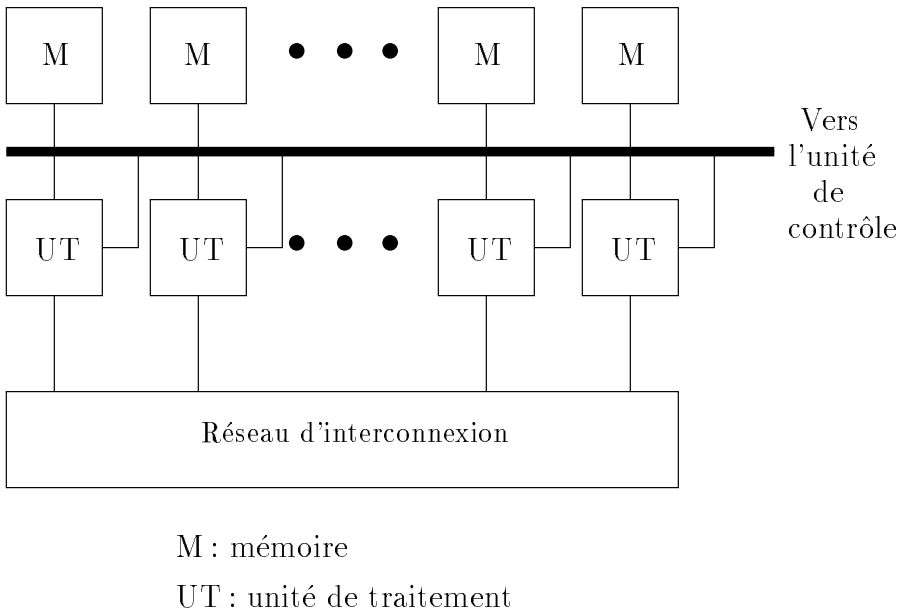
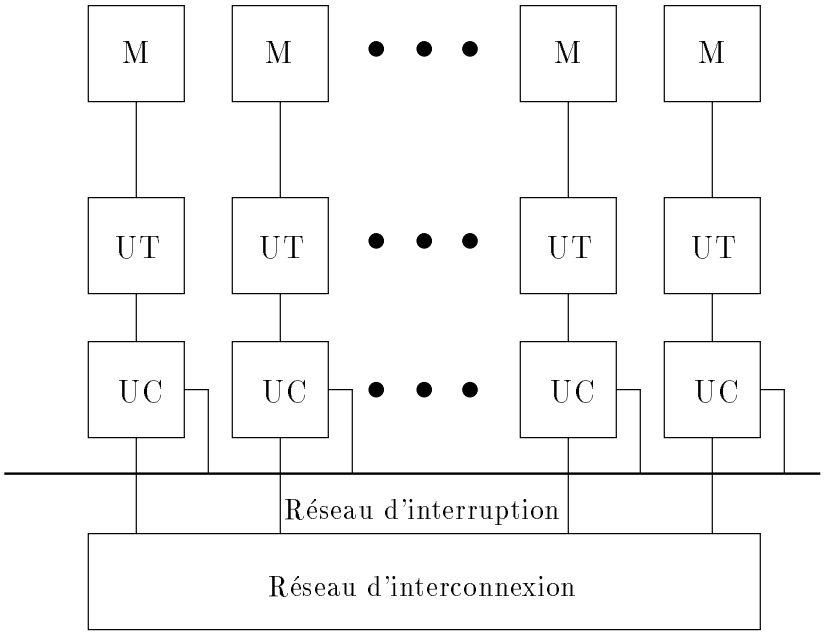


FIG. *1.3 : Architecture SIMD*

- **MIMD** : ce type de machines permet non seulement aux différents processeurs élémentaires de travailler sur des données différentes, mais, de plus, chaque processeur élémentaire peut exécuter des instructions différentes. Ces machines n'ont pas d'horloge globale et opèrent donc en mode asynchrone.

  L'architecture se compose de plusieurs unités de contrôle et d'autant de processeurs, d'une mémoire (qui peut être commune ou distribuée sur chaque processeur), d'un réseau d'interconnexion permettant les échanges entre unités de contrôle ou entre mémoires dans le cas où celles-ci seraient distribuées, et d'un réseau d'interruption (voir figure 1.4).

  Les iPSC/2, iPSC/860, la Paragon d'Intel, les machines à base de Transputers Inmos (T-Node et MegaNode de Telmat, Computing Surface de Meiko, ...) sont des machines MIMD. Ces machines ont un nombre plus faible de processeurs que dans le cas des SIMD (parallélisme à grain moyen et gros), mais ceux-ci sont beaucoup plus performants. Ce nombre varie de 8 processeurs superscalaires 64 bits sur le Cray 2 à 2048 nœuds (avec 2 processeurs 64 bits par nœud) sur la machine Paragon d'Intel.



M : mémoire

UT : unité de traitement

UC : unité de contrôle

FIG. *1.4 : Architecture MIMD*

Les iPSC/2, iPSC/860, la Paragon d'Intel, les machines à base de Transputers Inmos (T-Node et MegaNode de Telmat, Computing Surface de Meiko, ...) sont des machines MIMD. Ces machines ont un nombre plus faible de processeurs que dans le cas des SIMD (parallélisme à grain moyen et gros), mais ceux-ci sont beaucoup plus performants. Ce nombre varie de 8 processeurs superscalaires 64 bits sur le Cray 2 à 2048 nœuds (avec 2 processeurs 64 bits par nœud) sur la machine Paragon d'Intel.

Le mode de programmation SPMD (Simple Program Multiple Data) consiste à exécuter, sur une machine MIMD, un même programme sur tous les processeurs, mais chaque processeur travaille sur ses propres données. La distinction entre les différents processeurs ne se fait qu'à l'aide d'une variable locale contenant le numéro du processeur. En fait ce type de programmation est une extension du mode de programmation sur une machine SIMD. Par ailleurs les communications sont vues, en mode SPMD, comme un réarrangement de données. Il n'existe plus d'instruction du type *send* ou *receive*. Les programmes se décomposent en une succession de phases de calcul et de communication bien distinctes, ce qui nécessite des outils de synchronisation sur les machines MIMD.

Ce type de programmation se développe du fait de sa facilité de réalisation et des langages très performants existent comme le FORTRAN 90, le C*, qui permettent de réaliser du parallélisme sur les données (*data parallelism*).

1.7 Historique des machines parallèles

Les projets de construction des premières machines parallèles remontent aux années 70. C'est à l'université de Carnegie-Mellon (USA) qu'a débuté un de ces projets (C.mmp *Computer with multiple mini-processors*). Ce projet a donné naissance à l'Illiac IV [77]. Devant les problèmes rencontrés (fiabilité, techniques mal maîtrisées, absence de langage de programmation et d'outils de développement, etc.) il y eut peu d'autres projets durant cette décennie. Par contre, les idées développées au cours de ces années se sont retrouvées par la suite dans la conception des processeurs.

Dans les années 80, avec l'évolution des technologies et une meilleure connaissance des problèmes, sont apparues les premières machines parallèles. En Grande-Bretagne, ICL a produit le DAP (*Distributed Array Processor*) contenant 4096 processeurs 1 bit. Mais la machine qui a fait date est le Cosmic Cube [70], conçue sous la direction de Seitz à Caltech en 1981. Le Cosmic Cube était constitué de 64 processeurs Intel 8086 organisés en hypercube de dimension 6. Elle a été à l'origine de la machine iPSC/1 d'Intel, dans laquelle les communications étaient assurées par des interfaces « Ethernet ». Rapidement, on a vu l'émergence d'autres machines en hypercube comme celles produites par les sociétés Ametek [68] et nCUBE [63]. La taille moyenne de ces machines était, en 1985, d'une centaine de processeurs.

D'autre part, certains constructeurs comme FPS misaient sur la construction de machines parallèles à base de processeurs possédant des facilités de communication (quatre liens bidirectionnels). FPS proposait les séries T dont chaque nœud était constitué de *Transputers* T414 d'Inmos couplés avec des co-processeurs vectoriels. Les quatre liens des *Transputers* permettaient de construire des hypercubes de dimension 4 et, pour les hypercubes de taille supérieure, les liens étaient multiplexés, d'où une baisse de performance des communications entre processeurs. Du fait que les calculs vectoriels étaient très rapides, l'équilibre entre les communications et les calculs a été rompu.

La caractéristique essentielle de ces machines était l'échange de données entre processeurs par communication le long de liens physiques. Le routage était de type *store-and-forward* (voir le chapitre 2).

Vers la fin des années 80 sont apparues des machines issues des mêmes technologies, mais avec des processeurs plus performants (80386 et i860 à la place des 8086 pour Intel et les T800 à la place des T414 pour Inmos). Les communications étaient plus performantes mais restaient toujours pénalisantes par rapport au potentiel de calcul. C'était le cas des iPSC/2 et iPSC/860 d'Intel. Un certain nombre de machines telles que la série T-Node de Telmat [50], Computing Sur-

face de Meiko, ou Volvox d'Archipel sont construites autour du *Transputer* T800, ou parfois de processeurs i860 pour le calcul et d'un ou plusieurs T800 pour les communications.

Les machines à base de *Transputers* utilisent des *crossbars* pour interconnecter les processeurs. Si l'on veut relier un grand nombre de processeurs comme dans le cas du MegaNode (ensemble de T-Node), il faut un deuxième niveau de *crossbars* [62].

Grâce à un *crossbar* spécialisé (le *SN-switch*) qui équipe les machines de la série T-Node, est apparue la notion de reconfiguration dynamique (en cours d'exécution d'un programme). En effet, au cours d'un algorithme, il peut être intéressant d'exécuter une partie du programme sur une grille, puis une autre sur un hypercube, sans avoir à sauvegarder les résultats intermédiaires, reconfigurer, et recharger le programme sur la nouvelle topologie. Mais pour des raisons de difficulté et de coût lorsque le nombre de processeurs devient trop élevé, cette technique est devenue rapidement caduque.

A la fin des années 80, le routage restait encore du type *store-and-forward* (logiciel) sauf pour Intel qui a introduit le *direct-connect* (dérivé de la commutation de circuit), qui est réalisé de façon matérielle. Ce mode, présenté plus en détail dans le chapitre 2, permet de diminuer le temps de transfert d'un message entre deux nœuds éloignés. A cette fin, deux techniques sont exploitées : le contrôle est indépendant du processeur de calcul et est de type matériel (routeur implanté avec des mémoires programmables : *gate array CMOS*), de plus, il n'y a pas de stockage dans les nœuds intermédiaires comme en *store-and-forward*.

Les constructeurs de machines ont adopté ce type de routage, comme les versions iPSC/2 et iPSC/860 de l'hypercube d'Intel, la Simult (ex Ametek) 2010, en collaboration avec Caltech [75, chap. 1] ou encore le *Torus Routing Chip* (*Message Driven Processor* du MIT [20]) qui autorisait un routage *wormhole* (voir chapitre 2) sans interblocage sur un *k-ary-n-cube*, de préférence de dimension $n = 2$. On y exploitait des techniques mises au point par Dally et Seitz [23].

A peu près à la même époque est apparue la machine Butterfly TC2000 de BBN (Bolt, Beranek and Newman Inc.). Cette machine était constituée de processeurs 68020 de Motorola interconnectés par un réseau multi-étages en forme de *butterfly*. Les communications interprocesseurs se faisaient par commutation de paquets (voir chapitre 2), ce qui permettait d'avoir de bonnes performances [4].

Dès le début des années 90, une nouvelle génération de processeurs est apparue sur le marché, et la généralisation des communications par commutation de circuit a donné naissance à une nouvelle « gamme » de machines parallèles. Ces machines se retrouvent communément appelées MPP (*Massively Parallel Processing*).

Mais avant de décrire brièvement ces nouvelles machines, nous allons nous intéresser aux composants qui les constituent.

1.8 Description des composants

1.8.1 Les processeurs

A l'heure actuelle, les processeurs intègrent une grande puissance de calcul (de l'ordre de plusieurs dizaines de MFlops). Généralement, ces processeurs performants ne possèdent pas de gestion interne des liens de communication. En effet, l'intérêt économique des machines parallèles reste faible, et ces processeurs sont d'abord conçus pour équiper les stations de travail et les micro-ordinateurs. Pour construire des machines parallèles avec de tels processeurs, les constructeurs vont associer, au sein du même nœud, un deuxième processeur qui ne s'occupera que des communications (comme par exemple sur la Paragon d'Intel, la Computing Surface de Meiko, la Volvox d'Archipel, etc.). Voici une présentation des processeurs les plus utilisés :

- **Alpha**

  Le processeur Alpha est la nouvelle architecture RISC 64 bits proposée par DEC [28][29]. Comme Sun avec le Sparc, DEC propose une architecture ouverte à diverses implantations, en particulier aux implantations parallèles. L'architecture de ce processeur devrait évoluer au cours des vingt prochaines années, et les performances devraient être améliorées d'un facteur 1000 [46]. Dans un premier temps, les performances de ce processeur sont de l'ordre de 200 SPECmark, 400 MIPS, et 200 MFlops. DEC annonce jusqu'à 200 MHz mais l'horloge est à une phase (tic), au lieu de deux (tic-tac). De plus, on ne sait pas encore fabriquer des caches externes qui supportent 200 MHz.

  Fin 1992, DEC a présenté sa nouvelle gamme de stations de travail à base d'Alpha. La plus puissante de ces stations est équipée d'un processeur développant 125 SPECmark, et ayant une fréquence d'horloge de 150 MHz.

  Cray, qui est un partenaire industriel de DEC dans ce projet, utilise l'Alpha pour construire une machine massivement parallèle : le T3D.

- **i860 XR/XP**

  Ce processeur est sorti vers la fin des années 80. Il possède une architecture RISC 32 bits ayant une fréquence d'horloge de 40 MHz. Il existe une version améliorée, l'i860 XP, avec cohérence de cache ; il possède une horloge à 50 MHz et peut fournir une puissance de crête de 75 MFlops et 42 MIPS. Le i860 possède, en plus des composants courants d'un processeur élémentaire, une unité de contrôle des *pipelines* qui gère une unité graphique, et une unité de calcul flottant, le tout sur le même boîtier. L'originalité du processeur vient du fait que les unités *pipelines* peuvent travailler en parallèle avec le reste du processeur [54].

  Sa conception ne prévoyant pas de liens de communication, les cartes utilisées sur les machines Meiko ou Archipel sont dotées de *Transputers* pour

pallier cette lacune. Les communications entre le i860 et les *Transputers* se font via la mémoire. Pour l'iPSC/860, Intel a mis au point un composant de communication spécial, le DCRM (*Direct Connect Routing Module*), qui est connecté au i860 via la mémoire.

On notera que ce processeur est le seul qui n'équipe pas de stations de travail.

- **PA-Risc**

  Hewlett-Packard développe des composants PA-Risc (PA pour Precision Architecture). Ces composants forment une série de processeurs RISC 64 bits. Le haut de gamme a une fréquence d'horloge de 99 MHz, et développe 124 MIPS pour une puissance de 40 MFlops. Ce processeur équipe les stations de travail de la gamme HP9000 série 700 [26].

  Convex utilise ce processeur dans la fabrication d'une machine parallèle. De plus HP propose des *clusters* (groupe de stations connectées par exemple par réseau Ethernet) composés de huit stations de travail.

- **R4000**

  Ce processeur est développé par la société Mips ; il a une architecture RISC 64 bits. Sa fréquence d'horloge est de 100 MHz, pour une puissance de 70 SPECmark. Le R5000 devrait succéder au R4000. Le R5000 devrait avoir une fréquence de 150 MHz et développer 140 SPECmark.

  On retrouve ce processeur dans les machines de Silicon Graphics, où ces processeurs sont connectés jusqu'au nombre de huit.

- **RS/6000**

  Ce processeur, développé par IBM, équipe les stations de travail RS/6000. Il a une fréquence d'horloge de $62,5$ MHz, et développe 125 MFlops. Ce processeur 32/64 bits a été choisi par IBM pour constituer les nœuds de la nouvelle machine parallèle SP1.

- **MicroSparc**

  Sun a développé le concept de processeur à architecture extensible, avec la gamme de ses processeurs de la série Sparc (en fait, Sparc est l'acronyme de *Scalable Processor Architecture*). Le MicroSparc est un processeur qui suit les spécifications Sparc. C'est un processeur RISC 32 bits [48], utilisé par Sun dans la construction de ses stations de travail (Sun 4).

  Actuellement, ce processeur a une fréquence d'horloge de 40 Mhz, et peut fournir 40 MIPS, avec une puissance de crête de l'ordre de 5 MFlops. Grâce à sa MMU (Memory Management Unit) très performante, le processeur MicroSparc est capable de changer très rapidement de contexte entre différents processus, ce qui lui permet de gérer les applications de plusieurs utilisateurs.

– **SuperSparc ou Viking**

Sun s'est associé avec Texas Instruments, pour construire un processeur à architecture Sparc. Le nom de ce processeur est le Viking ou SuperSparc (c'est une évolution du MicroSparc). C'est un processeur RISC 64 bits, ayant une fréquence d'horloge de 40 MHz, et ayant une puissance de 100 MIPS. Ce processeur devrait évoluer pour atteindre, en 1994, une puissance de 150 SPECint92 et 200 SPECfp92 pour une fréquence d'horloge de 90 MHz. C'est le seul processeur présenté ici qui soit construit en technologie BICMOS (bipolaire CMOS).

Le Viking est utilisé par le consortium PCI, regroupant Meiko, Parsys et Telmat, pour la construction de la nouvelle machine CS-2. De plus, ce processeur est utilisé par Sun pour sa nouvelle gamme de stations de travail : SparcStation 10. La plus grosse de ces stations regroupe 4 Viking connectés par un bus.

Pour 1995, Sun prévoit une nouvelle gamme de processeurs appelés UltraSparc. Ces processeurs auront des fréquences d'horloge variant de 140 à 275 MHz, pour une puissance maximale de 475 SPECint92 et 750 SPECfp92. Pour réaliser de tels processeurs, Sun va devoir augmenter considérablement le nombre de transistors de ses puces. Ce qui va nécessiter l'utilisation de technologies d'intégration plus performantes.

1.8.2 Les processeurs avec facilités de communication

– **Texas Instrument TMS**320**C**40

Ce processeur 32 bits est construit pour le traitement parallèle du signal (*Digital Signal Processor*). Il possède une unité centrale de calcul (CPU) développant 25 MIPS et 50 MFlops avec une fréquence d'horloge de 25 MHz [49]. Il comporte des instructions de transfert de blocs avec l'adressage *bit-reverse* utile dans le calcul de la FFT (transformée de Fourier rapide). Il possède six co-processeurs DMA (*Direct Memory Access*) et six ports de communication bidirectionnels parallèles, ce qui lui permet d'avoir une bande passante totale de 320 Mo/s [41]. Rappelons qu'un DMA est une unité dédiée aux transferts de données entre les diverses mémoires et les ports de communications, les transferts s'effectuant sans intervention du CPU. Les DMA permettent donc un traitement concurrentiel des entrées/sorties et des opérations du CPU.

Texas Instruments propose aussi un *Digital crossbar switch*, qui offre 16 connexions de liens. Son délai de reconfiguration est de 20 ns.

– **T**800

Ce processeur RISC 32 bits, réalisé par Inmos, développe 1,5 MFlops, 30 MIPS et a une fréquence de 25 MHz. Il dispose de 4 liens bidirectionnels,

ayant chacun un débit de $2,5$ Mo/s. Ces liens fonctionnent de manière autonome par l'intermédiaire d'un DMA par lien et par direction. Ces processeurs peuvent être interconnectés par un *crossbar* (C004) [51]. Le T800 déjà ancien est peu performant, et sert surtout dans les systèmes embarqués, ou il est couplé à des processeurs plus performants. Dans ce dernier cas, le T800 ne s'occupe alors plus que des communications. C'est le plus utilisé parmi les processeurs possédant des facilités de communication.

- **T**9000 **et C**104

  - **T**9000

    Le T9000 est le processeur qui succède au T800 dans la gamme Inmos. Tout comme le T800, il sera destiné à la construction de machines MIMD, chacun de ses quatre liens bidirectionnels pouvant être connecté à un composant de routage : le C104. Le T9000 est un processeur 32 bits, doté d'une unité flottante sur 64 bits et dont les performances annoncées sont 200 MIPS et 25 MFlops. L'architecture est du type *superscalaire* et comporte un *pipeline*. Un groupeur d'instructions est chargé d'ordonnancer au mieux les instructions en attente [65], afin d'en faire exécuter certaines en parallèle, mais sans modifier l'ordre dans le flot d'instructions.

    Comme dans le T800, il y a un ordonnanceur intégré qui permet un temps de changement des processus actifs très faible (de l'ordre de la microseconde). De plus, les processus inactifs ne consomment pas de temps CPU. Ceci permet d'écrire des programmes parallèles performants (en Occam ou en C) [13] [58].

    Les principales évolutions par rapport au T800 sont :

    - la fréquence d'horloge qui passe de 25 à 50 MHz ;
    - le débit des liens de communication qui passe de 2.5 Mo/s à $12,5$ Mo/s.
    - la gestion de canaux virtuels qui va grandement faciliter la programmation. Une unité interne gère le multiplexage des liens physiques, c'est le *Virtual Channel Processor*. Il gère l'envoi des messages entre processus, de façon locale ou à travers le C104. Les envois sont mis dans une file d'attente pour ne pas bloquer le CPU. Ce VCP gère les quatre liens du T9000 et leurs contrôleurs DMA associés.

    On peut construire de grands systèmes à base de T9000, par des liaisons point à point, ou bien les interconnecter avec des C104.

  - **C**104

    Le C104 est un composant de routage incluant un *crossbar* 32×32 non bloquant, qui permet d'interconnecter des liens de T9000 ou des liens

de C104. Les messages transmis sur ces liens sont découpés en paquets de 32 octets et sont routés en mode *wormhole* dans le C104 [59]. Ce routage de paquets permet la gestion des canaux virtuels autorisés par le T9000. Ainsi, les messages peuvent être multiplexés sur les liens par le *Virtual Channel Processor*.

Cette notion de *wormhole* est quelque peu différente de celle utilisée à propos des réseaux point à point. En effet, il n'y a pas de liens dans un C104, ni de nœuds intermédiaires à traverser. Le principe du *wormhole* est expliqué en détail dans le chapitre 2, mais on peut préciser ici qu'il diffère de la commutation de circuit du fait que l'on n'attend pas d'accusé de réception dans ce mode de commutation. On transmet les données en même temps que l'on établit le chemin.

Le protocole de communication choisi permet d'éviter les interblocages dans les C104. Il consiste à attribuer à chaque lien de sortie d'un C104 un intervalle qui permet de sélectionner un lien en fonction de l'adresse de destination du message. L'algorithme d'étiquetage ne permet pas toujours de donner un routage optimal, mais il fonctionne bien sur les topologies courantes.

On peut aussi utiliser un mode de routage universel [79] qui permet d'éviter les goulots d'étranglement dans le réseau. Ces deux fonctions sont assurées par des composants internes du C104 : *l'interval selector* et le *random header generator*.

On peut connecter ces *crossbars* programmables soit à des T9000, soit à d'autres C104 pour construire de plus grands réseaux.

Les simulations faites par Inmos donnent des délais moyens de transmission entre deux nœuds de 27 à 64 μs, selon la taille du réseau (hypercubes de 64 à 16384 nœuds).

Notons qu'il existe plusieurs composants spécialisés dans le routage, développés par les constructeurs pour équiper leurs machines. Mais chaque constructeur garde secret ses composants.

Voici un tableau récapitulatif des différents types de processeurs commercialisés, qui permet de comparer les performances et les optiques de chaque constructeur.

Processeurs	Alpha	i860 XP	PA-7100	R4000	RS/6000	MicroSparc	Viking
Architecture							
RISC	OUI	OUI	OUI	OUI	OUI	OUI	OUI
superscalaire	OUI	OUI	OUI	NON	OUI	NON	OUI
pipeline	OUI	OUI	OUI	OUI	OUI	OUI	OUI
superpipeline	OUI	NON	NON	OUI	NON	NON	NON
Performances							
MHz	200	50	99	100	62	50	40
MIPS	300	42	124	-	-	40	109
MFlops	150	75	80	-	125	5	50
SPECint92	107	-	80	57	59	26	52
SPECfp92	196	-	150	61	128	21	65

Tableau récapitulatif des processeurs sans facilités de communication

Processeurs	TI C40	T800	T9000
Architecture			
RISC	OUI	OUI	OUI
superscalaire	OUI	OUI	OUI
pipeline	OUI	OUI	OUI
superpipeline	NON	NON	NON
Performances			
MHz	25	25	50
MIPS	25	30	200
MFlops	50	1,5	25
SPECint92	-	-	-
SPECfp92	-	-	-
nombre de liens de communication	6	4	4

Tableau récapitulatif des processeurs intégrant des facilités de communication

1.9 Les machines parallèles actuelles

1.9.1 La dernière génération de machines MIMD

Dans ce paragraphe, nous présentons les principales machines parallèles qui sont apparues sur le marché ces quatre dernières années.

- **Campus FX**800 [1]. Cette machine interconnecte jusqu'à 800 processeurs via un réseau de *clusters*. Elle est d'une puissance totale de 32 GFlops. C'est une machine MIMD, à mémoire distribuée entre *clusters*, et à mémoire partagée à l'intérieur d'un même *cluster*. La Campus FX800 est constituée de 32 *clusters*. Chaque *cluster* comporte 25 processeurs élémentaires i860 et 4 Go de mémoire partagée. Les *clusters* sont interconnectés par un HMI (*High-speed Memory Interconnect*) d'un débit de $2,56$ Go/s. A l'intérieur d'un *cluster*, l'accès à la mémoire partagée se fait par un *crossbar* avec un débit de $1,28$ Go/s.

– **CM-5**

C'est la dernière-née de Thinking Machines Corporation (TMC) [17]. Elle est prévue pour interconnecter de 32 à 2048 processeurs. Le but de TMC est d'augmenter le nombre de processeurs élémentaires jusqu'à 16384, afin d'obtenir 2 TFlops de performance de crête. Cette machine est du type MIMD à mémoire distribuée. Dans sa configuration actuelle, elle a une puissance maximale de 262 GFlops.

Les processeurs élémentaires sont constitués d'un processeur Sparc, d'une mémoire de 32 Mo, et de quatre unités de calcul flottant de 32 MFlops chacune.

Une des principales originalités de la CM-5 réside dans l'existence de trois réseaux. Un réseau de données (*Data Network*) gère les communications point à point, un réseau de contrôle (*Control Network*) permet des opérations globales (telles que les diffusions, réductions, synchronisations, etc.) par *pipeline* des messages, et enfin un réseau de diagnostic (*Diagnostic Network*) transmet les messages d'erreur.

La topologie adoptée pour le réseau de données est un *fat-tree* (voir figure 1.5) défini par Leiserson dans [55]. Les feuilles sont les processeurs élémentaires, et les nœuds intermédiaires sont des contrôleurs-routeurs (représentés en noir sur la figure 1.5). Une telle topologie offre à la machine une bonne résistance aux pannes, en permettant de doubler chaque lien et chaque contrôleur-routeur.

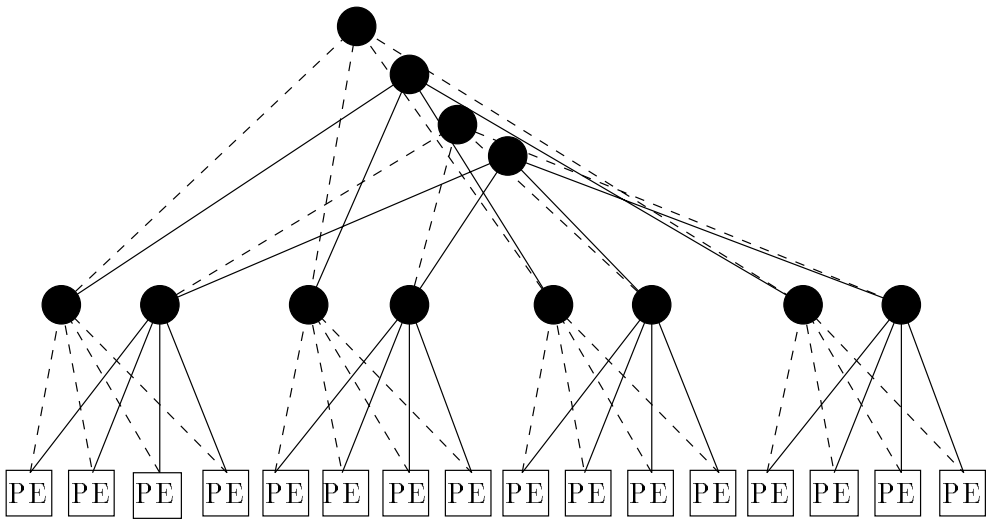


FIG. *1.5 : Fat-tree du réseau de données de la CM-5 à 16 processeurs*

La figure 1.6 montre comment est réalisé un des contrôleurs-routeurs du *fat-tree* [56]. Chaque cellule de routage a quatre fils, et deux ou quatre parents

suivant le nombre d'étages du réseau (ou plus exactement suivant le nombre de processeurs). Un lien du contrôleur a un débit de 20 Mo/s et les 4 liens peuvent fonctionner en parallèle.

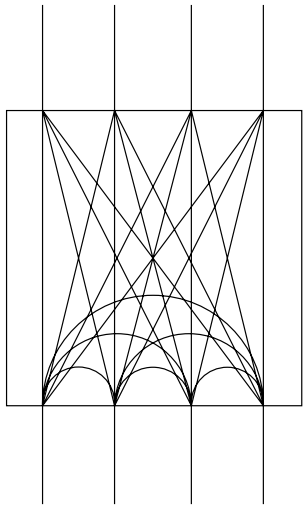


FIG. *1.6 : Un contrôleur du fat-tree*

Le débit des liens du réseau de données dépend de la proximité des processeurs élémentaires, c'est-à-dire du nombre de contrôleurs que le message doit traverser. Pour un contrôleur, le débit des liens est de 20 Mo/s ; pour deux contrôleurs, le débit est de 10 Mo/s ; pour plus de deux contrôleurs, le débit tombe à 5 Mo/s. Lors d'une communication point-à-point, le message utilise la technique suivante : remonter de façon aléatoire dans le réseau de contrôleurs-routeurs jusqu'à trouver celui qui permettra la redescente vers le destinataire, de façon déterministe.
La bande passante globale est de $0,5$ Go/s.

La topologie du réseau de contrôle est beaucoup plus simple. En effet, il s'agit d'un arbre binaire complet [56] dont les feuilles sont les processeurs, et les autres nœuds des contrôleurs. Ce réseau permet de réaliser toutes les opérations globales (diffusion, synchronisation, ou logique, ou exclusif, etc.). Les algorithmes utilisés sont des plus simples, comme pour la diffusion par exemple : le message à diffuser remonte jusqu'au contrôleur racine, avant de redescendre le long de toutes les branches.

- **Concerto**

  Cette machine est le fruit de l'association de trois constructeurs européens Meiko, Parsys et Telmat, qui forment le consortium PCI (Performant Computing Industry) [60]. La Concerto est une machine MIMD à mémoire distribuée possédant de 2 à 76 processeurs élémentaires, ce qui lui permet d'avoir une puissance de crête de l'ordre de 5 GFlops. Chaque processeur élémentaire est constitué d'un processeur de calcul qui peut être soit un i860 soit un Sparc, et de deux T800 qui assurent les communications entre processeurs élémentaires. Ces processeurs élémentaires sont reliés entre eux par un réseau d'interconnexion appelé CSN (*Computing Surface Network*), dont les constructeurs ne veulent rien dévoiler. Chaque processeur élémentaire a une bande passante de 16 Mo/s.

  La particularité de cette machine est d'être reconfigurable. En effet, on peut créer sur cette machine n'importe quelle topologie de degré 8 avec les huit liens des deux *Transputers*. Notons que la société française Archipel propose aussi des machines mixtes en mélangeant des i860 et des T800 qui échangent leurs données via la mémoire locale du nœud.

- **Cray MPP: T3D**

  Le dernier supercalculateur de Cray Research [18, 30] se nomme le T3D. Il a une architecture MIMD, avec une mémoire distribuée, mais accessible globalement par tous les processeurs. Chaque nœud du réseau est constitué de deux processeurs Alpha. Le réseau de communication est une grille tridimensionnelle. Elle devrait être capable d'atteindre 100 GFlops dans le courant 1994 et 1 TFlops en 1995.

- **CS-2**

  La CS-2 est la dernière machine née du consortium PCI. Elle permet d'interconnecter 1024 processeurs. Tout comme TMC, le but de PCI est d'augmenter le nombre de processeurs jusqu'à 4096 afin d'obtenir une machine capable de développer 2 TFlops de puissance de crête.

  C'est une machine MIMD, à mémoire distribuée. Le réseau de communication (voir figure 1.7) adopté est un réseau oméga. Les commutateurs qui constituent le réseau sont des *crossbars* 8×8, développés par PCI (appelés Elite). En fait, ce composant ressemble au C104 dans son fonctionnement.

  Un Elite est couplé avec quatre nœuds. Chaque nœud est constitué par quatre processeurs : deux processeurs vectoriels Fujitsu développant 200 MFlops chacun, un processeur Viking développant 150 SPECmarks et 30 MFlops LINPACK et le dernier processeur est un composant de routage, se nommant Elan, et construit par PCI. Ce processeur a une architecture RISC, et possède le même jeu d'instructions que le Viking.

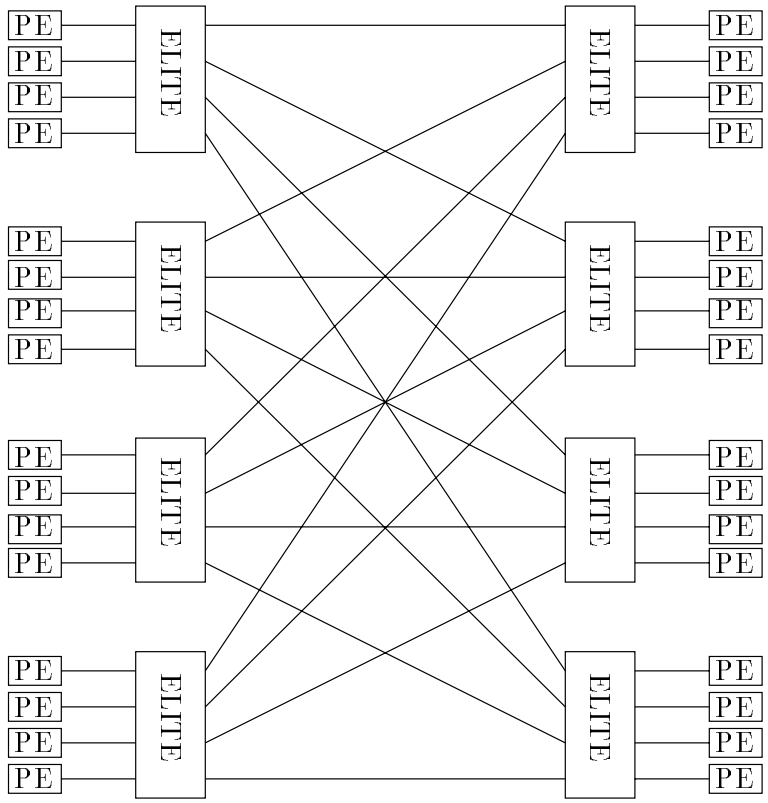


FIG. *1.7 : Réseau d'interconnexion de la CS-2 à 32 processeurs*

Les communications se font en mode *wormhole*. Comme sur la CM-5, la diffusion se fait en remontant au niveau d'Elite le plus haut avant de s'étendre à tout le réseau. Entre l'Elan et l'Elite le débit est de 50 Mo/s.

– **Fujitsu AP1000**

Cette machine MIMD comporte jusqu'à 1024 nœuds. Un programme est composé d'une partie qui se déroule sur l'hôte (station Sun qui gère toutes les entrées/sorties), et de tâches chargées sur les nœuds. Chaque nœud est doté d'un processeur MicroSparc et est relié à trois réseaux de communication distincts :

- Le *T-net* est le tore qui assemble les nœuds. Il offre des communications à 25 Mo/s et des liens larges de 16 bits. Le routage des messages entre nœuds est de type *wormhole*.
- le *B-net* est un réseau dédié à la diffusion d'un message d'un nœud à tous les autres (*broadcast*). Il connecte les nœuds, par paquets de 32, à un anneau. Chaque paquet a la structure d'un arbre dont la racine est un nœud de l'anneau. Le *B-net* offre une bande passante de 50 Mo/s

qui permet de le considérer comme un bus partagé malgré sa structure multicouches.

- Le *S-net* permet d'établir des points de synchronisation entre les nœuds du réseau. C'est un arbre dont les arêtes sont des lignes porteuses de signaux et les nœuds sont des portes logiques ET. Chaque nœud écrit son signal sur le *S-Net* et reçoit en retour le ET de tous les nœuds.

Cette machine exploite efficacement les caches des processeurs RISC en permettant d'échanger directement leurs données avec le routeur, sans utiliser d'accès DMA [71].

- **iWarp d'Intel** (de l'expression *warp speed* utilisée dans *Star Trek*)

  C'est une machine destinée aux applications temps réel embarquées (traitement du signal, traitement d'images). Elle est constituée de 8 à 1024 cellules iWarp connectées en tore bidimensionnel. Chaque cellule est un processeur de type VLIW (20 MIPS et 20 MFlops en simple précision) qui peut exécuter jusqu'à neuf opérations en parallèle. Un module de communication doté de quatre liens de communication parallèles bidirectionnels confère à la cellule une bande passante totale de 320 Mo/s ($4 \times 2 \times 40$ Mo/s) [7]. Vingt canaux virtuels, multiplexés sur les ports physiques, peuvent être gérés en parallèle. Le routage est de type *wormhole*.

  Les liens de type parallèle (où l'on transmet des octets) sont difficiles à utiliser pour des systèmes embarqués à la différence des liens de type série (où l'on transmet des bits) des DSP (Texas Instruments) ou des T800 (Inmos). Il existe un mode communication systolique qui est très efficace quand l'algorithme s'y prête et que les messages sont bien ordonnés.

- **KSR**

  La KSR de Kendall Square Research est une machine pouvant posséder de 32 à 1088 processeurs, et développant une puissance totale de 43 GFlops. Le réseau de processeurs est un anneau d'anneaux (voir figure 1.8). Sur la machine à 1088 processeurs, il existe un premier niveau constitué d'un anneau de 32 processeurs, et un deuxième niveau constitué de 34 contrôleurs eux-mêmes connectés en anneau. Sur chaque contrôleur vient se connecter un anneau de premier niveau. C'est une machine MIMD à mémoire dite *all-cache*, c'est-à-dire distribuée sur le réseau, mais virtuellement partagée par l'ensemble des processeurs. En fait chaque processeur dispose d'une mémoire cache locale notée LC sur la figure 1.8. Les processeurs, construits par Sharp, possèdent une architecture RISC 64 bits et chacun est couplé à trois co-processeurs, une unité de calcul flottant (FPU), une IPU (Integer and logical Operation Unit) et un co-processeur gérant les entrées/sorties. Ce processeur développe une puissance de 40 MFlops et 20 MIPS. Les communications se font par accès à des sous-pages mémoire de taille 128

Ko [66]. Lorsqu'un processeur veut accéder à une donnée qu'il ne possède pas, il lance une requête sur son anneau de niveau 1. Si celle-ci n'aboutit pas, elle passe au niveau 2. En retour, le processeur demandeur obtient sa page.

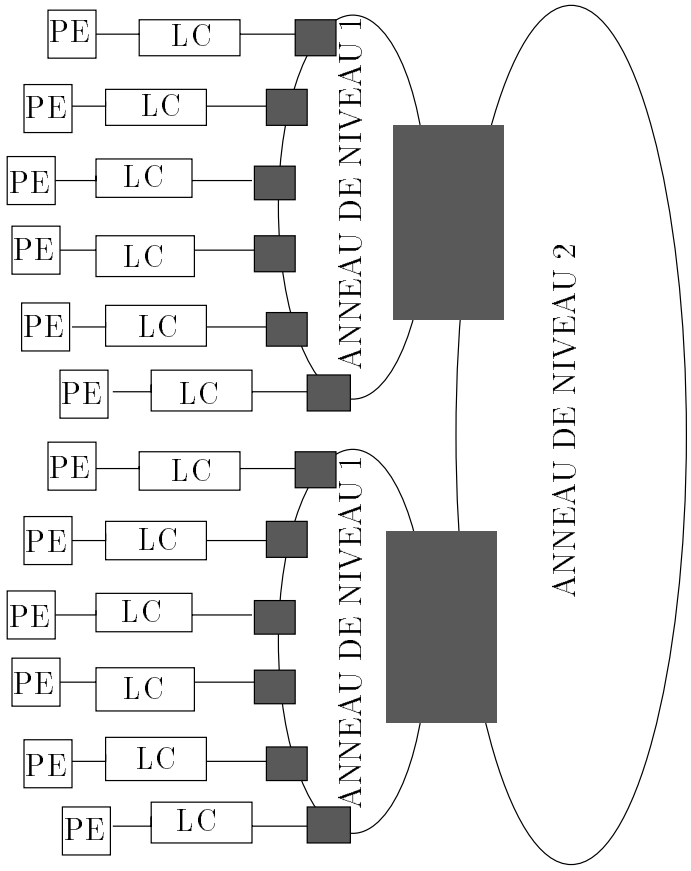


FIG. *1.8 : Anneau d'anneaux de la KSR*

– **nCUBE 2**

Cette machine permet d'interconnecter jusqu'à 8192 processeurs assemblés en hypercube (13-cube) et développe une puissance maximale de 27 GFlops. C'est une machine MIMD à mémoire distribuée. Les processeurs élémentaires sont réalisés par nCUBE et ont une puissance de crête de $2,4$ MFlops pour une fréquence d'horloge de 20 Mhz. Chaque processeur élémentaire peut disposer jusqu'à 32 Mo de mémoire. Le débit de chaque lien est de $2,22$ Mo/s [63].

Le nCUBE 2 présente deux particularités. La première est que chaque processeur élémentaire possède un lien directement connecté avec l'extérieur, ce qui permet de réaliser très facilement des entrées/sorties. La deuxième est

l'existence d'une horloge globale. En effet, peu de machines MIMD ont cette particularité et on peut ainsi introduire de vrais points de synchronisation dans les programmes.

Le routage est réalisé par le matériel, selon le mode *wormhole*.

En attendant sa future machine, nCUBE a annoncé, en septembre 1992, le nCUBE 2S qui offre un gain de performance de 50% par rapport au nCUBE 2.

- **nCUBE 3**

  Le nCUBE 3 (connu sous le nom de code *Whopper*) est le futur ordinateur parallèle de nCUBE, et sa commercialisation est prévue pour 1994. Cette machine sera MIMD et permettra d'interconnecter jusqu'à 65536 processeurs 64 bits. Comme sur son prédécesseur, les processeurs seront connectés en hypercube (16-cube), mais cette fois-ci nCUBE utilisera des processeurs construits par Hewlett-Packard. Jusqu'à ce jour, nCUBE utilisait des processeurs « maison ». Ces processeurs seront cadencés à 50 Mhz et développeront de l'ordre de 100 MFlops et 3 GIPS. Chaque processeur disposera de 18 canaux de communication et de 1 Go de mémoire locale. La machine dans sa version complète aurait une puissance de crête de $6,5$ TFlops.

- **Paragon**

  Cette machine est la version commerciale du projet *Delta-Touchstone* mené par Intel. Elle dispose de 56 à 4096 nœuds connectés physiquement en grille bidimensionnelle (voir figure 1.9) et développe une puissance de crête de 4 à 300 GFlops. C'est une machine MIMD à mémoire distribuée [16]. Chaque nœud est constitué d'un processeur de calcul i860 XP, d'une mémoire pouvant aller jusqu'à une capacité de 128 Mo, et d'un deuxième i860 XP chargé de la préparation des messages lors des réceptions ou des émissions. Ce deuxième i860 permet de libérer le processeur de calcul de cette tâche. Chaque nœud contient aussi un PMRC (*Paragon Mesh Routing Chip*), l'ensemble des PMRC constituant les sommets de la grille. Ces PMRC ont pour rôle de router les informations dans le réseau. Ce routage est effectué en *wormhole*. Chaque PMRC possède 4 liens bidirectionnels avec ses 4 voisins et chaque lien a un débit de 200 Mo/s en *full duplex*.

- **SP1**

  Après avoir développé des réseaux de stations de travail RS/6000, reliées entre elles par des interconnexions en fibre optique, IBM produit une machine parallèle : le SP1 (*Scalable Parallel*) [47]. Cette machine peut avoir de 8 à 64 processeurs, pour une puissance maximale de 8 GFlops. Chaque processeur est un RS/6000 ayant une fréquence d'horloge de $62,5$ MHz et développant 125 MFlops de puissance de crête. Il existe sur le SP1 deux

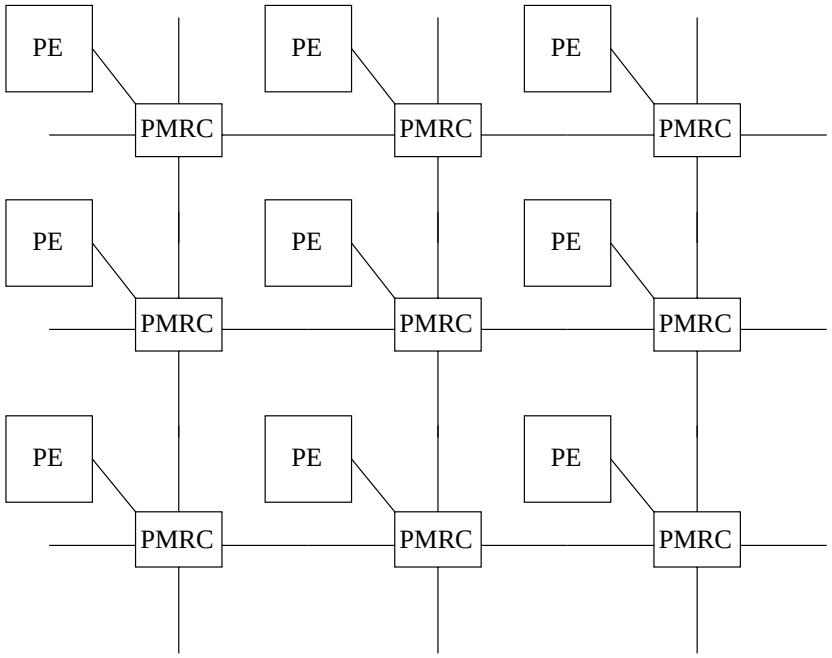


FIG. *1.9 : Réseau de la Paragon*

réseaux de communication. Le premier permet de connecter les processeurs via Ethernet, avec un débit de 10 Mo/s, ou via FDDI (*Fiber Distributed Data Interface*), avec un débit de 100 Mo/s. Le second est un réseau multi-étages en oméga. Les liens de ce réseau ont un débit de 40 Mo/s (*full-duplex*). Le SP1 à 64 processeurs est constitué de quatre *clusters* contenant chacun 16 processeurs RS/6000 (rectangles grisés sur la figure 1.10) et deux étages de commutateurs (petits rectangles blancs sur la figure 1.10). De chaque *cluster* sortent seize liens, douze vers les trois autres *clusters*, et les quatre restant servent à des connexions externes (disques par exemple).

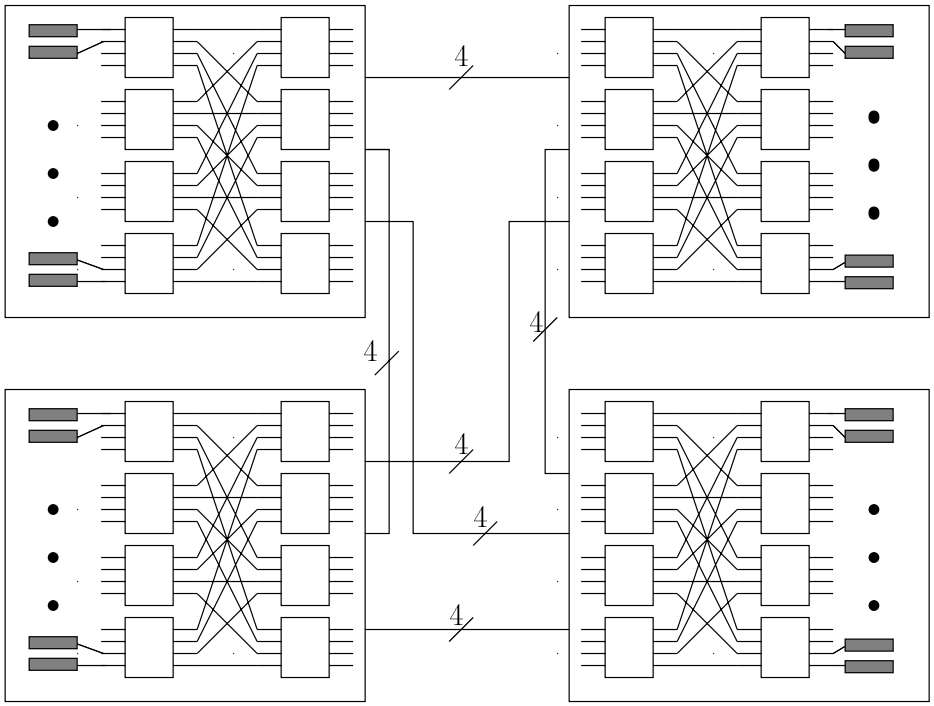


FIG. *1.10 : Réseau d'interconnexion du SP1 à 64 processeurs*

– **SUN**

Sun ne propose pas encore proprement dit une machine parallèle, mais plutôt un réseau de stations très performant. Sun groupe un ensemble de stations SuperSparc modèle 40 ayant chacune une puissance de 23 MFlops. Ces stations sont reliées entre elles par un réseau ATM (*Asynchronous Transport Mode*) à 155 Mbits/s. Ce réseau a une structure globale en étoile. Afin de pouvoir utiliser ce réseau comme une véritable machine MIMD, il est nécessaire d'utiliser une sur-couche logicielle (style PVM).

– **Projets autour du T9000**

Les *Transputers* T9000 d'Inmos et leurs composants d'interconnexion (C104), bien que prometteurs au vu des annonces, tardent à entrer en phase de production. Ils devraient permettre la construction de grands calculateurs MIMD européens.

Les programmes européens GPMIMD et GENESIS II regroupent les constructeurs Meiko, Parsys et Telmat, et ont pour but la construction d'une machine parallèle à base de T9000 et de C104.

Un autre projet de machine réalisé cette fois par Parsytec, associe le T9000 et le C104 [64]. Cette machine aurait pour nom CG et aurait jusqu'à 16384 processeurs T9000, ce qui permettrait de développer une puissance de crête

de 400 GFlops. Son architecture serait une grille tridimensionnelle formée de $8 \times 8 \times 16$ *clusters*. Chaque *cluster* formerait un nœud de la grille tridimensionnelle, contenant 16 T9000 et 44 C104, où chaque T9000 aurait un lien connecté à chaque C104.

Enfin, le projet LHPC (Laboratoire de Hautes Performances en Calcul) réalisé par la société Archipel et des universitaires, a pour objet la construction d'une machine modulaire à base de T9000 et de C104 pour les communications et de processeurs spécifiques pour les calculs.

1.9.2 La dernière génération de machines SIMD

Il est à noter toutefois que ce type de machine tend à disparaître au profit des machines MIMD. En effet, les machines SIMD sont constituées d'un grand nombre de processeurs peu performants, souvent incapables de réaliser à eux seuls une opération sur des flottants. De plus ce type de machine était étudié uniquement pour du parallélisme à grain fin.

- **CM-2**

  La Connection Machine 2 [45] est une machine du constructeur Thinking Machines Corporation (TMC). Elle est constituée de cartes identiques, regroupant 16 modules de 32 processeurs élémentaires 1 bit à 10 MHz. Dans sa configuration maximale elle contient 65536 processeurs élémentaires et a une puissance de crête de 40 GFlops. Un module contient deux circuits constitués de 16 processeurs élémentaires plus une unité de communication, auxquels s'ajoutent une mémoire locale et une unité de calcul flottant. L'ensemble des modules est relié à l'ordinateur frontal à travers un séquenceur par un bus d'instruction et un bus d'adresse. Les circuits sont reliés entre eux par un réseau de type hypercube (de degré 12 pour la configuration maximale). Les liaisons internes d'un groupe de 16 processeurs élémentaires sont réalisées par un réseau d'interconnexion. Il s'agit d'un *butterfly* à quatre étages [25]. L'essentiel des applications testées sur la CM-2 étant de type scientifique, une unité de calcul flottant a été ajoutée afin d'accélérer les calculs sur les nombres réels. Sur les dernières versions de la machine (CM-200), il est possible d'utiliser des processeurs travaillant en format double précision (64 bits) pour améliorer la précision. Le gain de temps obtenu grâce à l'apport de l'unité flottante est de l'ordre de 20 par rapport au calcul série (addition bit à bit) sur les processeurs élémentaires.

  Les communications possibles sont de trois types :

  - les communications générales qui servent à envoyer un message d'un processeur à n'importe quel autre. Elles utilisent une unité de communication interne (un routeur à la CM-2), ce qui entraîne une perte sensible au niveau des performances.

  - les communications NEWS (North, East, West, South) qui permettent d'envoyer un message sur un processeur voisin dans la topologie choisie. Les processeurs exécutant la même instruction, il n'existe pas de conflit de liens. Ces communications sont donc très rapides.
  - les communications avec recombinaison. Elles permettent de communiquer entre les processeurs d'une même dimension de la topologie choisie. Elles associent à la fois communications et opérations de recombinaison binaire des messages. Ces communications utilisent les liens de l'hypercube reliant tous les circuits (groupe de 16 processeurs). Par exemple, elles permettent de réaliser un produit scalaire sur une ligne de matrice.

- **DAP (*Distributed Array of Processors*)**

  La DAP est une machine SIMD commercialisée par le constructeur anglais AMT. Elle est constituée de processeurs 1 bit connectés en grille 64×64, soit 4096 processeurs au total. Ces processeurs ont une fréquence d'horloge de 10 MHz. La DAP à 4096 processeurs a une puissance de 560 MFlops. Les communications se font selon les lignes et les colonnes de la grille. La bande passante totale de la machine est de $5,2$ Go/s. De plus, AMT a développé des bibliothèques de communications et de calculs qui permettent à la DAP d'être particulièrement bien adaptée au traitement du signal et d'images.

- **MasPar MP-1 et MP-2**

  La MasPar MP-1 est une machine de type SIMD dont les processeurs constituent une grille torique octogonale [10], chaque processeur étant relié à huit autres processeurs suivant les directions de la rose des vents. Les processeurs élémentaires sont des processeurs 4 bits ; ils fonctionnent avec une fréquence d'horloge de base de $12,5$ Mhz, et possèdent chacun 16 Ko de mémoire. La MP-1 interconnecte de 1024 à 16384 processeurs. La MP-1216, c'est-à-dire la MasPar à 16384 processeurs, a une puissance de $1,5$ GFlops. Chaque processeur de la grille est connecté à un séquenceur ou ACU (*Array Control Unit*). Ce séquenceur est un processeur 4 bits possédant ses propres registres, sa propre mémoire pour les données et les instructions. C'est lui qui reçoit le code exécutable, et qui est chargé, lors de l'exécution, d'envoyer le code, ainsi que les données, aux processeurs de la grille.

  Il y a trois types de communication sur la MP-1 :

  - Les communications *X-Net* permettent à tous les processeurs élémentaires actifs d'envoyer un message à tous les processeurs se trouvant à une distance donnée dans une des huit directions définies par les liens.
  - Les communications via un routeur permettent de faire communiquer entre eux n'importe quels processeurs. Le routage est réalisé grâce à trois niveaux de *crossbars*.

  - Les communications de type « proc » qui permettent d'établir des communications entre les processeurs élémentaires et l'ACU (*Array Control Unit*).

  Digital vient d'annoncer la MasPar MP-2. Cette machine est une évolution de la MP-1 et possède entre 1024 et 16384 processeurs 32 bits capables chacun de réaliser 133 MIPS et $12,5$ MFlops suivant les mêmes principes.

- **Zephir**

  C'est la première machine de Wavetracer [80]. Zephir est du type SIMD et possède jusqu'à 8192 processeurs. La topologie adoptée par Wavetracer est la grille tridimensionnelle, ce qui permet de traiter plus facilement les problèmes dans l'espace, tels que la diffusion de la chaleur ou encore le traitement d'images tridimensionnelles. Un des avantages de cette machine réside dans le déploiement de la grille tridimensionnelle, afin d'obtenir une grille bidimensionnelle. Avec 8192 processeurs on peut donc avoir soit une grille tridimensionnelle ($16 \times 32 \times 16$) soit une grille bidimensionnelle (64×128). Les processeurs de cette machine sont des processeurs élémentaires 1 bit ayant une fréquence d'horloge de 10 Mhz. Cette machine développe 700 MIPS pour 120 MFlops. Le but de Wavetracer est de construire des machines massivement parallèles bas de gamme, et ne prenant pas de place. En effet cette machine tient sur un bureau (elle n'est guère plus grosse qu'une station de travail).

Récapitulatif

Les tableaux suivants résument les principales caractéristiques des machines parallèles énumérées précédemment ainsi qu'un tableau comparatif des performances réelles de ces machines (ces performances ont été obtenues avec un *benchmark* LINPACK par J. Dongarra [33]).

Nom de la machine	Puissance de crête	Nombre de processeurs	Mode de contrôle	Topologie du réseau de communication
Campus FX800	32 GFlops	800	MIMD	« réseau complet »
CM-5	262 GFlops	1024	MIMD	*fat-tree* et arbre binaire
Concerto	5 GFlops	76	MIMD	reconfigurable
CS-2	51 GFlops	256	MIMD	oméga
Fujitsu AP1000	4 GFlops	1024	MIMD	tore
KSR	43 GFlops	1088	MIMD	anneau d'anneaux
nCUBE2	27 GFlops	8192	MIMD	hypercube
Paragon	300 GFlops	4096	MIMD	grille 2D
SP1	8 GFlops	64	MIMD	oméga
T3D	4.8 GFlops	32	MIMD	grille 3D torique
CM-2	40 GFlops	65536	SIMD	hypercube
MasPar	1,5 GFlops	16384	SIMD	tore 2D
Zephir	120 MFlops	16384	SIMD	grille 3D

Récapitulatif des principales machines parallèles existantes

Type de machine	nombre de processeurs	R_{max} GFlops	N_{max}	$N_{1/2}$	R_{peak} GFlops
Thinking Machines CM-5	1024	59,7	52224	24064	131
Intel Delta (40 MHz)	512	13,9	25000	7500	20
Thinking Machines CM-200 (10 MHz)	2048	9,0	28672	11264	20
Alliant Campus FX800 (40 MHz)	192	4,8	17024	5768	7,7
SP1	128	3,8	26500	8000	16
KSR	256	3,4	10240	1824	10,2
Paragon	206	3,3	10000	-	10,4
Fujitsu AP1000	512	2,3	25600	2500	2,8
nCUBE 2 (20 MHz)	1024	1,9	21376	3193	2,4
MasPar MP-2216	16384	1,6	11264	1920	2,4
Intel iPSC/860 (40 MHz)	72	1,4	9000	3500	2,9
Meiko Computing Surface (40 MHz)	62	1,3	8500	3500	2,5
IBM RS/6000 Cluster (62,5 MHz)	8	0,694	10000	1500	1,0
MasPar MP-1216	16384	0,473	11264	1280	0,55

Performances des principales machines parallèles existantes

Les colonnes du tableau précédent donnent les quantités suivantes :

- R_{max} est la performance en GFlops pour le plus grand problème pouvant être exécuté sur la machine.
- N_{max} est la taille en octets (du nombre de données) du plus grand problème pouvant être exécuté sur la machine.
- $N_{1/2}$ est la taille en octets pour laquelle les performances obtenues sont moitié de R_{max}.

– R_{peak} est la performance de crête de la machine.

1.9.3 Les grandes lignes actuelles

Les tendances qui se dégagent des nouvelles machines présentées ci-dessus sont la généralisation du routage de type *wormhole* ou commutation de circuit (avec l'apparition de routeurs matériels performants), et la disparition des ordinateurs frontaux (comme pour la CM-5 ou la Paragon où les utilisateurs se connectent directement sur un processeur de la machine, vu qu'un certain nombre de nœuds ont un noyau système). Une autre tendance est la disparition des réseaux « classiques » (hypercube, anneau, grille, etc.) au profit de réseaux multi-étages comme sur la CS-2 de PCI, la SP1 d'IBM. L'avantage de ces réseaux est de garantir le choix entre de nombreux chemins pour faire communiquer deux processeurs quelconques. De plus, vues les méthodes de routage utilisées dans ces machines, le coût de communication entre deux processeurs reste sensiblement le même, quel que soit l'emplacement de ces processeurs dans la machine. On voit également apparaître la notion de machines extensibles (*scalable*) : en ajoutant des cartes on peut augmenter le nombre de processeurs de la machine, sans que les utilisateurs aient à modifier leurs applications pour travailler sur la machine complète.

De plus, il émerge deux stratégies : la première consiste à chercher la plus grande puissance de calcul : c'est celle adoptée par les grands constructeurs tels que Cray, Intel, Thinking Machines Corporation, etc. ; la deuxième consiste à maximiser la puissance à prix constant : c'est le cas des machines à base de *Transputer*, des constructeurs Wavetracer et Alliant, etc.

En marge de ces machines développées par des industriels (souvent en collaboration étroite avec des universités), de nombreux projets universitaires ont pour objet la construction de machines parallèles dédiées à la programmation fonctionnelle, au traitement d'images, etc.

Outre ces machines parallèles « traditionnelles », on voit apparaître une utilisation des réseaux de stations de travail comme machines parallèles, grâce à des logiciels tels que PVM (*Parallel Virtual Machines)* [8], Express, P4 [11]. Les stations de travail sont reliées entre elles par un réseau Ethernet ou FDDI (*Fiber Distributed Data Interface*) ou par un *Token-ring* [74].

Un problème subsiste : le refroidissement

De tous temps, les constructeurs ont multiplié la puissance des processeurs par le nombre de processeurs de leur machine parallèle, pour annoncer sa puissance maximale. Cependant, il ne suffit pas d'assembler indéfiniment des processeurs pour avoir des systèmes toujours plus puissants. Un premier problème est la longueur des connexions (c'est la raison principale de la limite à 16384 nœuds de la CM-5). Un autre problème est le refroidissement du système. On ne peut pas regrouper un grand nombre de processeurs dans un espace réduit sans connaître des problèmes de surchauffe. Ces problèmes n'étaient à présent connus que dans

le cadre des supercalculateurs (par exemple, Cray). Les techniques de refroidissement liquide employées par Cray ne sont sûrement pas applicables directement sur des machines de la taille d'un gymnase (taille de la CM-5 à 16384 processeurs), et il faudra donc reconsidérer certains paramètres des réseaux. Par l'augmentation du nombre de composants dans les processeurs, plus de courant circule et donc plus de chaleur se dégage. Sur son dernier processeur, le Pentium (destiné pour le moment à succéder au 80486), Intel a dû installer un ventilateur sur le processeur afin d'éviter que celui-ci ne fonde après un quart d'heure d'utilisation. DEC a rencontré ce type de problème avec l'Alpha.

1.10 Perspectives

Certains constructeurs annoncent pour leurs machines des performances de l'ordre d'une centaine de GFlops, mais, sur des applications réelles, les machines que nous avons présentées ne dépassent guère quelques dizaines de GFlops. Ceci est dû aux communications entre processeurs, et à la non-optimisation du code pour chaque processeur. En effet, la puissance des processeurs a augmenté plus vite que les performances des liens de communication, actuellement les processeurs calculent à des vitesses de l'ordre de 100 MFlops alors que les débits des liens sont de l'ordre 15 Mo/s ! ce qui fait que les processeurs passent une grande partie de leur temps à attendre des données. De plus, les applications doivent rester portables pour éviter de réécrire le code chaque fois que l'on change de machine. Ceci implique qu'il est difficile de tirer parti des spécificités des processeurs sur lesquels on travaille. Comme, les applications sont écrites en langage de haut niveau, il faut donc fournir rapidement des compilateurs adéquats avec chaque nouvelle machine.

Ainsi, pour répondre à la demande croissante de puissance de calcul, un certain nombre de constructeurs tels que Cray, Intel ou Thinking Machines Corporation envisagent de construire prochainement des machines atteignant les 3T (TeraFlops, Teraoctets, Teraoctets/seconde). Pour atteindre les 3T, les constructeurs prévoient d'une part de mélanger des techniques SIMD et MIMD, et d'autre part de rendre les processeurs plus performants. Le CMOS et le BICMOS n'ont pas encore atteint leurs limites physiques (on parle moins de l'arséniure de gallium GaAs), et le « tout optique » [38] ne semble pas recueillir beaucoup de suffrages. On compte sur les supraconducteurs, la fibre optique, ou encore l'augmentation des possibilités d'intégration pour augmenter les performances. Par exemple, Intel a réussi à intégrer quatre processeurs élémentaires sur un seul substrat en laboratoire (Micro 2000), mais seulement deux en pratique (Pentium, où il y a deux processeurs 80486 sur le même substrat). Ces technologies restent encore expérimentales et sont très onéreuses. Cependant, NEC a réussi à produire, avec ces techniques, des processeurs développant $0,2$ GFlops [82]. Une production à grande échelle devrait pouvoir réduire les coûts de fabrication.

Les processeurs RISC comportent de plus en plus d'instructions (par exemple

168 pour le Alpha, 155 pour le HP, 118 pour le Viking, etc.), ce qui les éloigne radicalement du concept de jeu d'instructions réduit. Par ailleurs, les processeurs CISC utilisent des instructions de plus en plus simples. L'idée est donc de fusionner ces deux principes. Par exemple, Intel est arrivé à exécuter plusieurs instructions par cycle d'horloge sur ses processeurs 80486 à architecture CISC. Les derniers processeurs d'Intel comme le i960 et le Pentium combinent les techniques RISC et CISC : ce sont des processeurs de type *superscalaire*.

Les super-machines parallèles ne remplaceront pas des calculateurs moins ambitieux, mais plus efficaces, pour les applications courantes. Ces super-machines seront dédiés à une classe d'applications qui nécessitent les 3T : les prévisions météorologiques sur de longues périodes, la modélisation du génome humain, ou encore la dynamique des battements du cœur (actuellement un jour de CPU Cray modélise un battement).

Des machines plus communes pourront connecter de 4 à 20 processeurs performants sur un bus, limite probable due aux problèmes posés par la cohérence des caches. Un gain en performance évident devrait être obtenu en mode multi-utilisateurs par répartition de la charge des utilisateurs sur les processeurs. Enfin, des réseaux locaux à haut débit permettront d'accéder à ces différentes machines depuis les stations individuelles.

Bibliographie

[1] Alliant. *The Campus FX800 massively parallel Supercomputer*, 1991.

[2] G.S. Almasi. Overview of parallel processing. *Parallel Computing*, 2:191–203, 1985.

[3] W.C. Athas and C.L. Seitz. Multicomputers : message-passing concurrent computers. *IEEE Computers*, 21:9–24, 1988.

[4] R. G. Babb. *Programming parallel processors.* Addison-Wesley Publishing Company Inc., 1988.

[5] D. H. Bailey and J. T. Barton. *The NAS kernel benchmark program.* NASA Ames Research Center (USA), June 1986.

[6] H.B. Bakoglu. *Circuits, interconnections and packaging for VLSI.* Addison-Wesley, 1990.

[7] B. Baxter and B. Greer. Apply : a parallel compiler on iWarp for image-processing applications. In IEEE Computer Society Press, editor, *The Sixth Distributed Memory Computing Conference proceedings*, pages 186–193, 1991.

[8] A. Beguelin, J.J. Dongarra, A. Geist, R. Manchek, and V. Sunderam. A users' guide to PVM parallel virtual machine. Technical Report TM-11826, Oak Ridge National Laboratory, (USA), July 1991.

[9] G. Bell. The future of high performance computers in science and engineering. *Communication of the ACM*, 32(9):1091–1101, 1989.

[10] T. Blank. The MasPar MP-1. *IEEE Computers*, pages 20–24, 1991.

[11] R. Bulter and E. Lusk. User's Guide to the P4 Programming System. Technical Report ANL-92/17, Argonne National Laboratory, (USA), 1992.

[12] J. R. Bunch, J. J. Dongarra, C. B. Moler, and G. W. Stewart. *LINPACK user's guide.* SIAM, 1979.

[13] A. Burns. *Programming in OCCAM-2.* Addison Wesley, 1988.

[14] O. Collins, S. Dolinar, R. Eliece, and F. Pollara. A VLSI decomposition of the De Bruijn graphs. Technical report, Jet Propulsion Laboratory, CIT, Pasadena (USA), 1989.

[15] O. Collins, F. Pollara, S. Dolinar, and J. Statman. Wiring Viterbi decoder (splitting De Bruijn graphs). Technical Report TDA progress 42-96, Jet Propulsion Laboratory, Pasadena (USA), 1988.

[16] Intel Corporation. *PARAGON XP/S, product overview*, 1991.

[17] Thinking Machines Corporation. *CM-5*, 1992.

[18] Inc Cray Research. *Cray T3D technical Summary*, 1993.

[19] R. Cypher. Theoretical aspects of VLSI pin limitations. Technical Report 89-02-01, Department of Computing Science, University of Washington (USA), 1989.

[20] W. Dally, L. Chao, A. Chien, S. Hassoun, W. Horwat, P. Song J. Kaplan, B. Totty, and S. Wills. *Architecture of a message driven processor.* Journal of the ACM, 1987.

[21] W.J. Dally. *A VLSI architecture for concurrent data structures.* Kluwer Academic Publishers, 1987.

[22] W.J. Dally. Performance analysis of $k-$ary $n-$cube interconnection networks. *IEEE Transactions on Computers*, C-39(6):775–785, 1990.

[23] W.J. Dally and C.L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[24] Jean de Rumeur. *Communications dans les réseaux de processeurs.* collection ERI. Masson, 1994.

[25] D. Delesalle, D. Trystram, and D. Wenzek. *Tout ce que vous voulez savoir sur la Connection Machine.* Laboratoire de Modélisation et de Calcul, Grenoble (France), 1991.

[26] E. Delsol. HP: le RISC plus puissant et moins cher. *01 Informatique*, page 4, November 1992.

[27] J. Demmel, J. J. Dongarra, J. Ducroz, A. Greenbaum, S. J. Haammarling, and D. C. Sorensen. A project for developing a linear algebra library for high-performance computers. In M. H. Wright, editor, *IFIP*, pages 87–92. Elsevier Science Publishers, 1989.

[28] Digital. *21st century 64-bit RISC architecture*, 1992.

[29] Digital. *Digital 21064-AA microprocessor*, 1992.

[30] T. Mc Donald. *The Cray research MPP FORTRAN programming model.* Cray Research Inc., 1992.

[31] J. J. Dongarra. Performances comparées de 80 ordinateurs sur des programmes FORTRAN. *Technique et Science Informatique*, pages 355–360, 1984.

[32] J. J. Dongarra. The LINPACK benchmark: an explanation. In Supercomputing, editor, *First International Conference*, pages 456–474, 1988.

[33] J. J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, Mathematical Science Section, Oak Ridge National Laboratory (USA), April 1993.

[34] J. J. Dongarra and W. Gentzsch. Special issue : Benchmarking of high performance supercomputers. *Parallel Computing*, 17(10), December 1991.

[35] R. Duncan. A survey of parallel computer architectures. *IEEE Computers*, 5:5–16, 1990.

[36] K. Eshraghian and D. A. Pucknell. *Basic VLSI design, systems and circuits.* Prentice-Hall, 1988.

[37] D. Etiemble. *Architecture des processeurs RISC.* 2ai, Armand Colin, 1991.

[38] D. G. Feitelson. *Optical Computing.* The MIT Press, 1988.

[39] J. T. Feo. An analysis of the computational and parallel complexity of the Livermore loops. *Parallel computing*, 7(2):163–185, June 1988.

[40] M.J. Flynn. Very high speed computing systems. *Proc. IEEE*, 54:1901–1909, 1966.

[41] H. Frydlender. *Implantation de réseaux de neurones artificiels sur multi-processeurs à mémoire distribuée.* PhD thesis, Institut National Polytechnique de Grenoble, November 1992.

[42] J. D. Gee, M. D. Hill, D. N. Pnevmatikatos, and A. J. Smith. Cache performance of the SPEC benchmark. Technical Report 1049, University of California at Berkeley Computer Science Division (USA), 1991.

[43] D. Géneau. Le SPECmark est mort. *01 Informatique*, January 1992.

[44] J.L. Hennessy and N.P. Jouppi. Computer technology and architecture : an evolving interaction. *IEEE Computers*, 24(9):18–29, 1991.

[45] W.D. Hillis. *The Connection Machine.* MIT Press, 1985.

[46] E. Hostachy. L'architecture Alpha. *01 Informatique*, pages 1–3, January 1992.

[47] IBM. *IBM scalable POWERparallel ststem SP1 - User guide*, 1993.

[48] Sun Microsystem Inc. *The Sparc architecture manual*, 1987.

[49] Texas Instruments Inc. Texas instruments parallel-processing floating-point DSP. Documentation, 1990.

[50] Telmat informatique. *T-Node user manual.* 1990.

[51] Inmos. *Transputer databook.* Inmos databook series. Inmos limited, 1988.

[52] M. Karchmer. *Communication Complexity, A New Approach to Circuit Depth.* The MIT Press, 1989.

[53] P.M. Kogge. *The Architecture of Pipelined Computers.* Hemisphere Publishing Corporation, 1981.

[54] L. Kohn and N. Margulis. Introducing the Intel i860 64-bit microprocessor. *IEEE Micro*, 1989.

[55] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transaction on Computers*, C-34(10):892–901, October 1985.

[56] C. E. Leiserson. The network of the CM-5. In ACM, editor, *SPAA 92*, 1992.

[57] T. Lengauer. VLSI theory. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 836–868, 1990.

[58] Inmos Limited. *OCCAM2: Manuel de référence.* Masson, 1989.

[59] Inmos Limited. Special T9000. *La Lettre du Transputer (hors série)*, 1991.

[60] Meiko Limited, Parsys Limited, and Telmat Informatique. *A technical overview of the Concerto system : software tools and implementation*, 1992.

[61] C. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley, 1980.

[62] T. Muntean and P. Waille. L'architecture des machines SuperNode. *La Lettre du Transputer*, (7), 1991.

[63] nCUBE. *Technical overview system - nCUBE 2*, 1990.

[64] Parsytec. *Parsytec GC*, 1991.

[65] D. Pountain. Virtual channels: the next generation of Transputers. *Byte*, April 1990.

[66] Kendall Square Research. *Technical summary*, 1992.

[67] Carl Sechen. *VLSI Placement and Global Routing using Simulated Annealing*. Kluwer Academic Publishers, 1988.

[68] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W-K. Su. The architecture and programming of the Ametek serie 2010 multicomputer. In ACM Press, editor, *Third Conference on Hypercube Concurrent Computers and Applications proceedings*, pages 33–36, 1988.

[69] C.L. Seitz. Concurrent VLSI architectures. *IEEE Transactions on Computers*, C-33(12):1247–1265, 1984.

[70] C.L. Seitz. The Cosmic Cube. *Communications of the ACM*, 28(1):22–33, 1985.

[71] T. Shimizu, T. Horie, and H. Ishihata. Low-latency message communication support for the AP1000. In ACM SIGARCH, editor, *The 19th International Symposium on Computer Architecture*, pages 288–297, 1992.

[72] Mazakazu Shoji. *CMOS Digital Circuit Technology*. Prentice Hall International Editions, 1988.

[73] J. Statman, G. Zimmerman, F. Pollara, and O. Collins. A long constraint length VLSI Viterbi decoder for the DSN. Technical Report TDA progress 42-95, Jet Propulsion Laboratory, Pasadena (USA), 1988.

[74] W. R. Stevens. *UNIX network programming*. Prentice Hall, software series, 1990.

[75] R. Suaya and G. Birtwist. *VLSI and parallel computation*. Morgan Kaufmann, 1990.

[76] D.W. Terry, D.E. Stivers, K.W. Pennington, M.W. Riley, and H.C. Nguyen. Packaging for high performance. Technical Report SA23-2619, IBM RISC System/6000 Technology, IBM Corporation (USA), 1990.

[77] A. Trew and G. Wilson. *Past, present, parallel: a survey of available parallel computing systems*. Springer-Verlag, 1991.

[78] L.G. Valiant. Universality considerations in VLSI circuits. *IEEE Transaction on Computers*, C-30(2):135–140, 1981.

[79] L.G. Valiant. General purpose parallel architectures. Technical Report 07-89, Harvard University, (USA), 1989.

[80] Wavetracer. *A deskside massively parallel computer*, 1990.

[81] R. P. Weicker. An overview of common benchmarks. *IEEE Computer*, pages 65–75, December 1990.

[82] G. Zorpette. Special issue: supercomputers. *IEEE Spectrum*, pages 27–76, September 1992.

Chapitre 2

Modèles de communications et topologies

2.1 Introduction

Lorsque l'on exécute un programme sur une machine parallèle à mémoire distribuée, le temps d'exécution de ce programme dépend du temps de calcul et de la durée des communications entre les processeurs. Pour minimiser le temps d'exécution d'un processeur il est donc en particulier nécessaire de toujours « alimenter » les unités de calcul, c'est-à-dire de réduire les temps d'attente de données provenant d'autres processeurs. Il faut donc d'abord minimiser le volume global de données à échanger entre les processeurs, puis minimiser le coût de ces échanges en développant des stratégies, c'est-à-dire des algorithmes de communication efficaces (voir livre livre *communications dans les réseaux de processeurs* [7]).

Il existe de nombreux articles sur le sujet des communications. Le groupe « RUMEUR » (opération de C^3) dispose d'une bibliographie de plus de deux cents articles ! Cependant l'étude de ces articles montre que les hypothèses utilisées diffèrent souvent, ce qui rend difficile une quelconque comparaison de l'efficacité des solutions proposées. De plus, de nombreuses études utilisent des modèles abstraits et les résultats sont parfois peu exploitables sur des machines distribuées. Afin de réaliser par la suite une étude comparative des différents algorithmes de communication, nous prendrons le modèle qui correspond le mieux aux communications dans les machines parallèles à mémoire distribuée de ces dernières années.

Le but de ce chapitre est de présenter les différents modes de routage utilisés dans les machines à mémoire distribuée actuelles, ainsi que les modélisations des coûts de communication qui leur sont associées. En outre, la topologie du réseau sous-jacent est un paramètre important pour les communications. La dernière partie de ce chapitre sera consacrée à la présentation des topologies les plus utilisées à l'heure actuelle.

2.2 Communications dans un nœud

Dans le chapitre 1 nous avons présenté la modélisation d'un nœud, sans en expliquer la gestion ni les contraintes associées. C'est ce que nous allons exposer maintenant.

2.2.1 Gestion des messages

Dans ce paragraphe, nous présentons sommairement la gestion de la reception ou de l'emission de messages entre deux processeurs: lorsqu'un nœud veut communiquer avec un autre, il utilise un de ses liens connectés au réseau d'interconnexion. Il est alors important pour les performances des algorithmes que les unités de traitement ne soient pas interrompues. Pour cela chaque lien est couplé à une interface du type DMA (*Direct Memory Access*). Cette interface peut accéder à la mémoire ou aux caches de données sans interrompre les autres unités du nœud.

Du fait que les nœuds deviennent de plus en plus puissants, et qu'ils disposent de systèmes multi-tâches, il n'est pas rare que plusieurs processus s'exécutent sur un même nœud. Le nombre de processus peut donc être très largement supérieur au nombre de liens du nœud, ce qui pose un problème lorsque les processus veulent communiquer avec d'autres processus se trouvant sur d'autres nœuds du réseau. Pour résoudre ce problème, on considère que les nœuds communiquent avec leurs voisins en échangeant des messages sur des *canaux*. Un canal est une liaison point à point, uni-directionnelle, entre deux nœuds reliés par un lien physique. Plusieurs canaux peuvent partager un même lien physique, on parle alors de *multiplexage*.

Sur chaque nœud peut être installé un routeur. Le rôle de ce routeur est d'acheminer les messages entre nœuds non voisins. Les routeurs réalisent de façon distribuée un algorithme de routage qui spécifie le chemin à suivre dans le réseau pour se rendre de tout nœud x vers tout nœud y. Cet algorithme est décrit par une fonction de routage. Localement, le routeur choisit, pour chaque message qui arrive sur un des canaux d'entrée du nœud, le canal de sortie qui convient. Pour cela, il peut utiliser des tables. Ce genre de méthode n'est cependant pas couramment utilisé dans les machines multiprocesseurs (au contraire des grands réseaux planétaires comme par exemple le routage du courrier électronique sur Internet), et les multiprocesseurs utilisent le plus souvent une méthode de routage basée sur une arithmétique simple qui peut être implantée avec une mémoire programmable (en technologie CMOS) sur le nœud.

2.2.2 Les contraintes de communication

Malheureusement, pour les utilisateurs on ne peut pas réaliser des communications aussi facilement sur une machine parallèle à mémoire distribuée. En effet, suivant les technologies et les techniques employées un certain nombre de

restrictions existent sur les façons et les possibilités de faire communiquer deux nœuds ensemble.

Une première restriction peut provenir du type même des liens utilisés dans le réseau. Considérons deux processeurs voisins p_1 et p_2 reliés entre eux par un lien.

- Si les communications entre ces deux processeurs ne peuvent se faire dans un sens, c'est-à-dire que p_1 peut envoyer un message à p_2 mais p_2 ne pourra jamais envoyer de message à p_1 par ce lien alors ce type de lien est dit *monodirectionnel*.

- Si les communications entre ces deux processeurs peuvent se faire dans les deux sens c'est-à-dire que p_1 peut envoyer un message à p_2 et que p_2 peut aussi envoyer un message à p_1 par ce lien alors ce type de lien est dit *bidirectionnel*.

Une deuxième restriction peut avoir trait à l'échange simultané de données entre deux nœuds voisins. Considérons deux processeurs p_1 et p_2 reliés par un lien de communication bi-directionnel.

- Si un seul message à la fois peut circuler entre p_1 et p_2, soit de p_1 vers p_2, soit de p_2 vers p_1, le lien est dit *half-duplex*.

- Si deux messages peuvent circuler au même instant sur le même lien, l'un transitant de p_1 vers p_2 et l'autre de p_2 vers p_1, le lien est dit *full-duplex*. C'est le cas le plus fréquent en pratique.

Une autre restriction pourra être dûe aux limitations des possibilités de communication entre l'interface mémoire et les liens de communication pour chaque nœud.

- Si, lors d'une communication, chaque nœud ne peut envoyer ou recevoir un message que sur un seul lien à la fois, les communications sont dites *1-port*. Cette contrainte qui limite les échanges est donc aussi appelée *processor-bound* ou *whispering* (littéralement *en murmurant*). C'était le cas pour l'iPSC/1 d'Intel.

- Au contraire, si chaque nœud peut utiliser simultanément tous ses liens, alors les communications sont dites Δ-*port*, où Δ fait référence au nombre de liens maximum des nœuds du réseau. C'est maintenant le nombre de liens qui limite les communications. Ce schéma est aussi appelé *link-bound* ou *shouting* (littéralement *en criant*). C'est le cas des systèmes à base de *Transputers* (T800 ou T9000), l'iWARP, CM2 ...

- Entre ces deux cas les plus fréquents, il y a celui où seulement k liens de communication parmi les Δ disponibles sont simultanément utilisables. On

se trouve alors dans un schéma *k-port*. Les communications sont limitées par le nombre maximum de DMA gérant les transferts de données entre la mémoire des processeurs et leurs liens de communication. On trouve donc aussi l'appellation *DMA-bound*. Exemples de machine : les systèmes à base de *Transputers* avec multiplexage comme les FPS série T.

2.3 Différents modes de commutation

La transmission d'un message entre deux nœuds non directement liés s'effectue grâce aux routeurs des nœuds intermédiaires. Chaque routeur reçoit une adresse de destination, qu'il décode pour déterminer le canal de sortie, et envoie cette adresse sur ce canal. C'est ce qu'on appelle la commutation du routeur. Le temps mis par le routeur pour réaliser sa commutation est appelé la *latence*.

Il existe différentes techniques de commutation, voici la description des plus utilisées.

2.3.1 Commutation de circuit (*circuit-switched*)

La commutation de circuit consiste à établir un chemin entre les deux nœuds qui veulent communiquer, avant d'envoyer le message. Le fonctionnement est le suivant : l'émetteur envoie un en-tête (*header*) en direction du récepteur. Dans chaque nœud traversé l'en-tête réserve un canal afin de construire un circuit. Une fois l'en-tête arrivée à destination, un accusé de réception (*acknowledgement*) est émis par le récepteur en direction de l'émetteur. Cet accusé de réception emprunte le circuit précédemment établi. A la réception de cet accusé, l'émetteur envoie le message en une seul fois [12]. La figure 2.1 décrit ce fonctionnement. Sur cette figure l'en-tête est noté h et l'accusé de réception a.

Il est impossible à d'autres messages d'emprunter une partie d'un circuit déjà établi tant que le circuit n'est pas libéré. Ce mode de fonctionnement correspond au principe du téléphone.

Exemple. Le *direct connect* qui est proposé par *Intel* sur les hypercubes de la série iPSC/2 et iPSC/860 est un routage de type commutation de circuit.

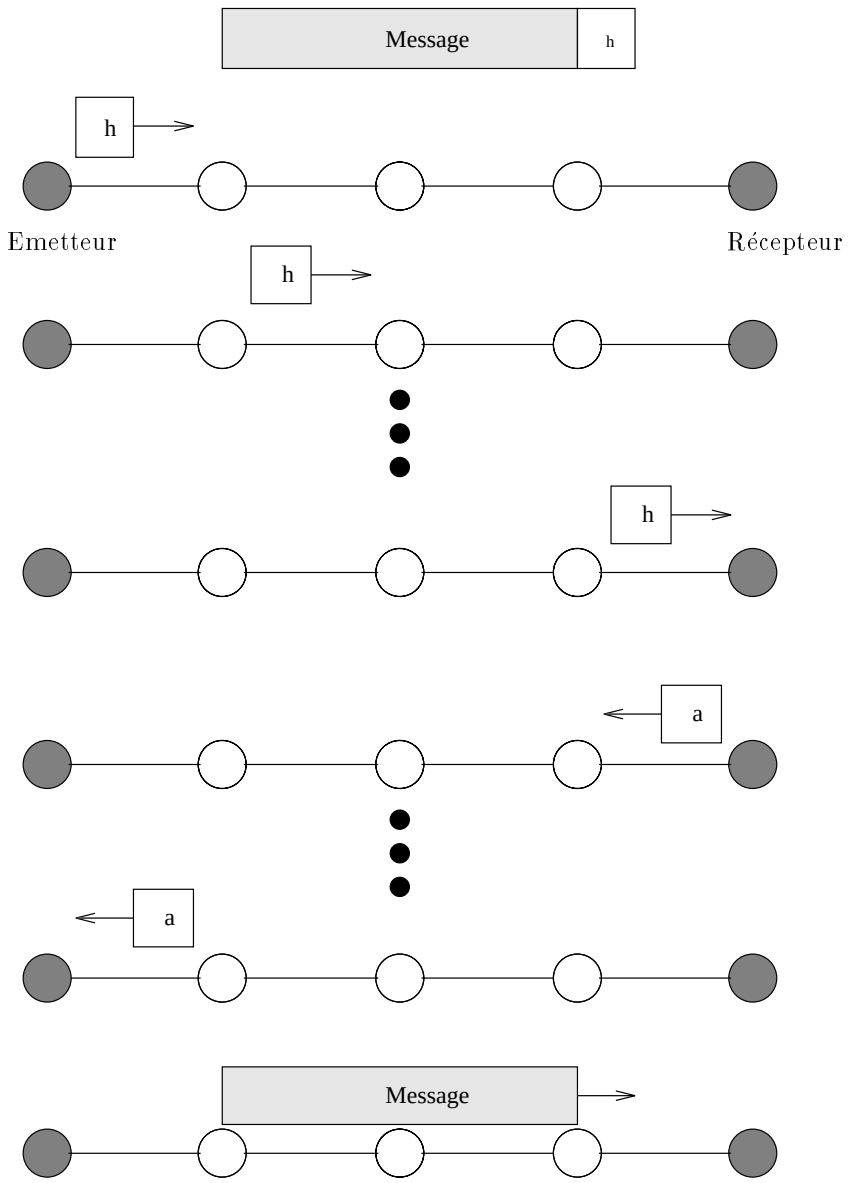


FIG. 2.1 : *Commutation de circuit.*

2.3.2 Commutation de messages (*store-and-forward*)

Dans ce mode de commutation, l'émetteur envoie en une seule fois le message et l'en-tête. Le message avance alors vers sa destination en étant stocké dans chaque nœud intermédiaire. A chaque étape, le canal emprunté est aussitôt libéré. Cette technique est connue sous le nom de *store-and-forward* [12].

Le défaut de cette technique est de nécessiter une taille de registre importante pour stocker le message sur les processeurs intermédiaires. En fait, ces messages sont généralement stockés en mémoire, or les accès mémoire sont proportionnels à la taille des messages d'où un ralentissement des communications. Par contre

l'avantage de cette technique est que tous les processeurs intermédiaires ont eu connaissance du message, ce qui est fort utile dans les schémas de communication globaux comme la diffusion (voir chapitre 5).

Exemple. La Computing Surface de *Meiko*, les T-Node et Mega-Node de *Telmat*, ou les Volvox d'*Archipel* utilisent un routage par commutation de messages.

2.3.3 Commutation de paquets (*packet-switched*)

Ce mode est une adaptation du mode *store-and-forward* : en effet le message initial est découpé en paquets de taille fixe, chaque paquet possède un en-tête et est sont envoyés les uns à la suite des autres, ce qui permet de bénéficier d'un effet *pipeline* (voir description précise à la section 2.5.1).

Ce mode de commutation présente deux avantages. Le premier est dû à la taille fixe des paquets et à leur petite taille. En effet, les paquets vont pouvoir être stockés dans les registres du routeur au lieu que le message complet soit stocké en mémoire. La capacité de stockage nécessaire sur un nœud pour un canal se limite donc à la taille d'un paquet. Comme chaque paquet possède un en-tête, le deuxième avantage est de pouvoir router les paquets indépendamment les uns des autres, les paquets peuvent même emprunter des chemins différents.

Notons que ceci implique un surplus de communications engendré par le supplément d'information ajouté à chaque paquet, notament pour la recomposition du message à partir des différents paquets.

Ceci est schématisé par la figure 2.2.

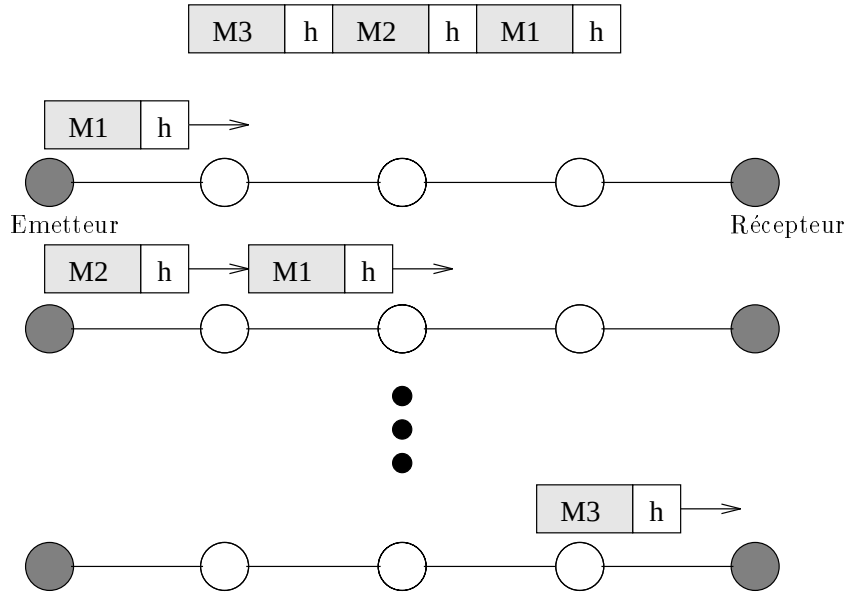


FIG. 2.2 : *Commutation de paquets.*

2.3.4 Routage *wormhole*

Dans ce mode de routage, le message est découpé en paquets de petite taille que l'on appelle *flits* (*flow control digit*) [6, 8] (égals à la taille de la queue d'un canal). Le premier *flit* contient l'adresse de destination, et les autres des morceaux du message. Le premier *flit* avance dans le réseau et établit le chemin, il est directement suivi par les autres *flits* (voir figure 2.3). Une fois que le dernier *flit* a traversé un canal, ce dernier est alors disponible pour un nouveau message.

Si le premier *flit* ne peut plus avancer car le canal qu'il doit emprunter est occupé, il attend que ce canal se libère. Les *flits* qui le suivaient sont stockés dans les queues des canaux qu'ils occupaient, bloquant ainsi le chemin. On peut comparer le cheminement d'un message dans ce mode à celui d'un ver qui progresse sous la terre. Notons qu'une fois que le *flit* de tête a été affecté à un canal, ce canal ne peut transmettre aucun *flit* d'un autre message tant que le message originel n'est pas passé : « on ne peut pas couper le ver ».

Remarque.
Cette technique de routage permet au récepteur de recevoir des morceaux du message avant que l'émission du message ne soit terminée, ce qui se produit pour des messages longs. Mais elle permet aussi, dans le cas de messages très courts de libérer les canaux proches de l'émetteur avant le début de la réception du premier *flit* par le destinataire. Un problème se pose si la distance entre les deux nœuds est grande, en effet, des erreurs de transmission peuvent se produire, d'où la nécessité d'introduire un protocole d'accusé de réception des *flits* dans ce cas-là.

Il est important de bien distinguer ce mode de routage des routages par commutation de paquets et des routages par commutation de circuit (ou *circuit-switched*). En effet, dans le routage par commutation de paquets, chaque paquet contient en en-tête l'adresse de destination et peut donc être routé indépendamment des autres paquets. Au contraire, dans le cas du *wormhole*, seul le premier *flit* contient l'adresse de destination et tous les *flits* suivent le même chemin. Dans le routage par commutation de circuit, il faut établir un circuit physique entre les correspondants. Dans le cas du *wormhole*, il n'y a pas construction de circuit physique : le circuit n'est que virtuel et n'existe qu'entre le premier et le dernier *flit*.

Exemple. Le *wormhole* a été choisi par *Inmos* pour la nouvelle série des *Transputers* T9000 et des *crossbars* C104. La CM-5 de *Thinking Machine Corp.*, la Paragon d'Intel, la CS-2 de PCI utilisent aussi ce type de routage.

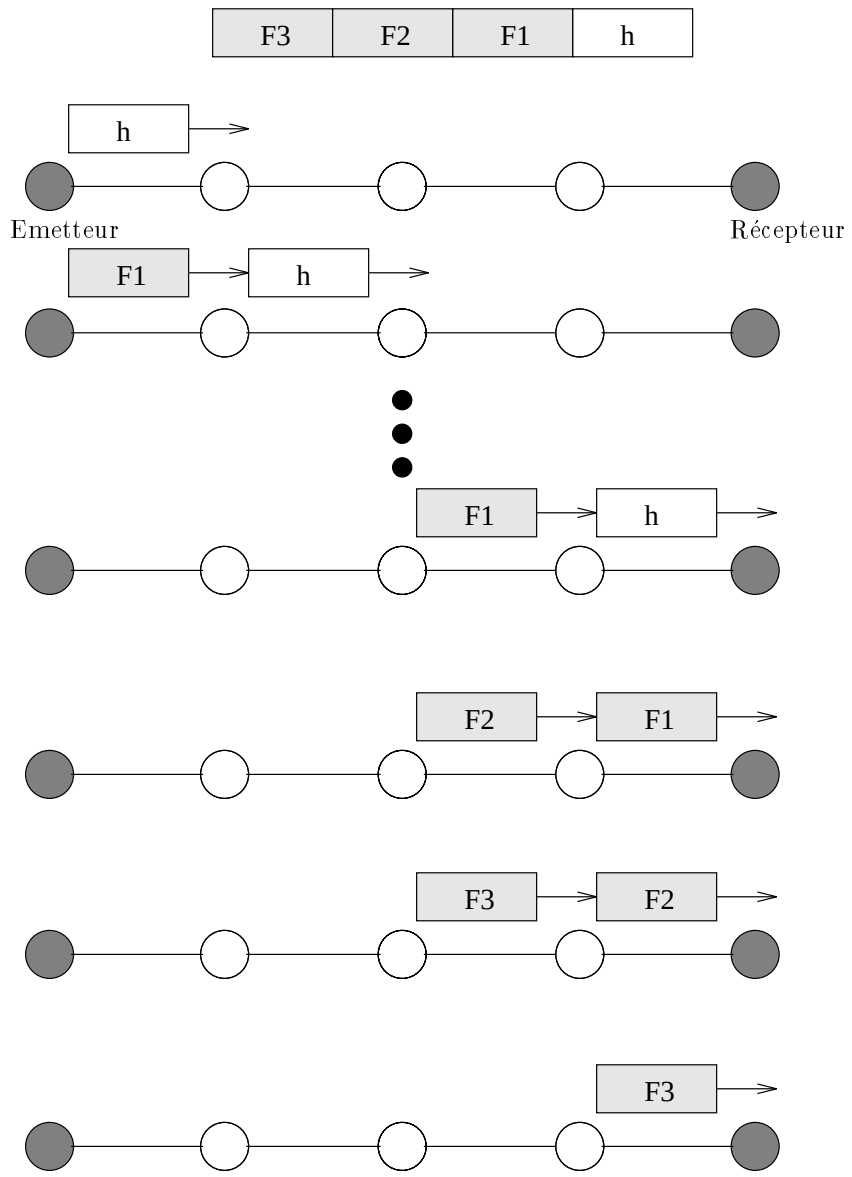


FIG. 2.3 : Routage wormhole

2.3.5 *Virtual-cut-through*

Le *virtual-cut-through* a été introduit par Kermani et Kleinrock [12]. Ce mode de routage est identique au *wormhole* sauf quand la propagation du premier *flit* d'un message est impossible. Cette fois-ci les *flits* contenant le corps du message continuent à avancer et sont tous stockés sur le nœud où le premier *flit* s'est trouvé bloqué, ce qui présente l'avantage de libérer des canaux qui peuvent être ainsi utilisés par d'autres messages. Le routeur doit donc disposer de queues de tailles arbitrairement grandes, ce qui ne permet pas de l'intégrer facilement sur un nœud.

2.3.6 Le mode transparence

Le problème commun à la commutation de circuit et à la commutation de paquets, du *wormhole* et du *virtual-cut-through* est que les nœuds intermédiaires ne peuvent pas lire les *flits* qui transitent par leur routeur. En effet, en mode *wormhole* par exemple, lorsqu'un message est émis par un nœud x à destination de z, et que ce message transite par un nœud y lors du routage, le message n'a pas été stocké dans la mémoire de y, mais seulement dans les registres du routeur. Ceci pose des problème lors de schémas de communication tels que la diffusion. C'est pourquoi le mode transparence fait l'objet d'études de la part de C. T. Ho, dans ce mode les nœuds intermédiaires pourraient lire les *flits* qui transitent par leur routeur.

2.4 Modélisation du temps de communication

Pour évaluer le coût d'un algorithme sur une machine à mémoire distribuée il est nécessaire de connaître le coût des communications. Afin, de le déterminer différents modèles, des plus simples aux plus proches des machines, du coût d'une communication ont été proposés.

Notons que les modèles cités dans cette section ne tiennent pas compte des problèmes de congestion, c'est-à-dire que l'on considère que tous les canaux sont libres.

2.4.1 Modèle temps constant

Dans ce modèle, on considère qu'une communication entre deux nœuds voisins coûte une unité de temps et ne dépend pas de la taille du message communiqué.

$$T_1 = 1. \tag{2.1}$$

De même la communication entre deux nœuds distants de d va coûter d unités de temps.

$$T_d = d. \tag{2.2}$$

Dans ce modèle, les messages peuvent être découpés et recombinés sans affecter le temps de transfert. Comme le coût de la communication ne dépend pas de la taille du message, un nœud qui a reçu précédemment deux messages a intérêt à combiner les deux messages au lieu de faire deux envois séparés. A l'inverse, le découpage d'un message en petits paquets coûtera plus cher, car l'envoi de chaque paquet coûtera 1 unité. Ce modèle est peu réaliste et n'a guère qu'un intérêt théorique.

2.4.2 Modèle temps linéaire

Dans ce modèle le temps de transmission d'un message de taille L entre deux processeurs voisins est modélisé par la somme :

- d'un temps d'initialisation (*start-up*) β représentant le temps des initialisations de registres mémoire, ou le temps de procédures d'envoi/accusé de réception ;
- et d'un temps de propagation $L\tau$ proportionnel à la taille L du message communiqué. τ représente le temps de propagation d'une unité de message. La bande passante d'un lien est donc l'inverse de τ.

$$T_1 = \beta + L\tau. \tag{2.3}$$

Ce modèle [16, 18] se rapproche plus de la réalité des machines à mémoire distribuée [3, 17] que le modèle temps constant. Mais comme on le verra au chapitre 6, il peut être encore amélioré.

Voici pour quelques machines les valeurs de β et de τ :

	β μsec	τ $\mu sec/octet$
Ametek 2010	0.08	0.035
CM-2	-	0.100
CM-5	73	0.1
Fujitsu AP 1000	0.04	0.040
iPSC/1	1000	1.000
iPSC/2	136	0.384
iPSC/860	350	0.2
MegaNode	4.85	1.125
Meiko CS-1	250	1.000
nCUBE	446	2.500
FPS T40	830	1.430

Il convient de faire plusieurs remarques relatives à ce tableau :

- il faut se garder de comparer de façon brutale les valeurs du tableau pour en déduire une hiérarchie entre les machines. En effet, certaines valeurs de *start-up* par exemple ne sont pas directement comparables car elles proviennent de mesures effectuées dans des environnements logiciels différents. C'est le cas pour les iPSC et le MegaNode.
- sur les iPSC [4, 3], le temps de propagation est plus important pour les messages longs que pour les courts (limites respectives à 1Ko dans le cas de l'iPSC/1, et à 100 octets dans le cas de l'iPSC/2). Ceci est dû à un processus de vérification de l'espace mémoire disponible chez le destinataire

avant l'émission de gros messages. La modélisation temps linéaire ne s'applique pas directement, il faut modifier le modèle afin de considérer cette discontinuité des paramètres.

Dans ce modèle, la communication entre deux nœuds distants de d va dépendre du mode de routage.

Commutation de messages

Dans le mode *store-and-forward* simple, le message est transmis de voisins en voisins, ce qui revient à faire d communications de voisin à voisin, d'où un temps de communication égal à :

$$T_d = d(\beta + L\tau). \tag{2.4}$$

Commutation de circuit et *wormhole*

Pour ces deux modes de routage, le paramètre β se décompose en un *start-up* α dû au protocole de communication dans le nœud et à la somme des temps δ nécessaires pour commuter chaque routeur des nœuds intermédiaires [14], d'où un temps de communication égal à :

$$T_d = \alpha + d\delta + L\tau. \tag{2.5}$$

On peut remarquer que si l'on fait une communication entre deux voisins suivant ces modes de routage on trouve que $\beta = \delta + \alpha$.

2.4.3 Les autres modèles

Les modèles exposés jusqu'à présent sont les plus utilisés, mais il existe dans la littérature d'autres modèles dont voici les principaux :

- Une étude menée par Bertsekas et al. [2] sur les communications dans les hypercubes utilise les hypothèses suivantes : le transfert d'informations sur les liens se fait par paquets de taille fixe. Il n'est pas possible de découper, ni de combiner les paquets entre eux. Envoyer un paquet prend un temps 1, et envoyer p paquets prend un temps p. Plus généralement, on peut modéliser le temps voisin à voisin par :

  $$T_1 = \left\lceil \frac{L}{m} \right\rceil \tau$$

  L étant la longueur du message à envoyer, m la taille des paquets, et $\frac{1}{\tau}$ la bande passante des liens.

- Touzene et Plateau [19] ont étudié les communications dans le tore en supposant que le temps d'initialisation des communications était nul. Les messages peuvent ainsi être découpés et recombinés si nécessaire. Cela correspond au modèle :

$$T_1 = L\tau$$

où L est la longueur du message à envoyer, et $\frac{1}{\tau}$ est la bande passante des liens. Ce modèle est utilisable sur des messages de grande taille, c'est-à-dire $L\tau \gg \beta$.

- Il est possible d'affiner le modèle temps linéaire. En effet ce modèle ne tient pas compte du nombre de liens utilisés à la fois sur chaque nœud. Des expérimentations [9] montrent que le temps d'initialisation et le temps de propagation peuvent tous deux dépendre du nombre de liens utilisés par un processeur à un instant donné. En effet, il faut payer le prix de la gestion logicielle du parallélisme des liens. Le modèle est de la forme :

$$T_1 = \beta_k + L\tau_k$$

où L est la longueur du message à envoyer, et β_k et τ_k sont respectivement les temps d'initialisation et de propagation lorsque le processeur communique en utilisant k liens. Faire varier τ est aussi la méthode que l'on peut utilisée pour modéliser les communications dans les machines à communications hiérarchiques.

2.5 Comparaison des principaux modes de routage

Comme nous l'avons vu au chapitre 1 les trois modes de routage les plus répandus dans les machines à mémoire distribuée sont le *store-and-forward*, la commutation de circuit et le *wormhole*, c'est pourquoi il est intéressant de comparer rapidement ces différents modes de routage.

Comme le montrent les équations 2.4 et 7.2, le routage *store-and-forward* s'effectue approximativement d fois plus lentement que dans le routage *wormhole* ou par commutation de circuit.

Voici un tableau récapitulatif, rappelant les principales différences entre le mode *store-and-forward* et les modes *wormhole* et commutation de circuit. Ce tableau indique aussi des valeurs expérimentales des paramètres caractéristiques

des communications.

	Store-and-forward	*Wormhole* ou Circuit
Un nœud stocke	tout le message en mémoire	1 *flit* sur un registre du routeur
Blocage possible	sur un nœud	sur un chemin
Communication à distance d	$d(\beta + L\tau)$	$\alpha + d\delta + L\tau$
Exemples de machines	T-node, iPSC/1, nCUBE, Cosmic-cube	iPSC/2, iPSC/860, T9000 nCUBE-2, MDP (MIT)
Valeurs	iPSC/1 : $\beta = 1000\ \mu s$ $\tau = 1\ \mu s/octet$	iPSC/860 : $\alpha = 350\ \mu s$ $\delta = 10\ \mu s, \tau = 0.2\ \mu s/octet$

Il existe cependant des techniques *logicielles* qui peuvent être utilisées pour accélérer les communications dans le mode *store-and-forward*, en particulier la technique du *pipeline* ou encore la technique du *pipeline* couplée à l'utilisation de chemins disjoints.

2.5.1 La technique du *pipeline*

Pour réduire le temps de transmission dans le mode de routage *store-and-forward*, on peut découper le message de longueur L en paquets, et les transmettre les uns à la suite des autres. C'est la technique du *pipeline*. Pour pouvoir utiliser cette technique il est nécessaire de travailler en mode 2-port, car chaque nœud intermédiaire doit pouvoir émettre et recevoir sur deux canaux différents.

Soit d la distance entre la source et la destination. On découpe le message de longueur L en paquets de taille q. Supposons pour simplifier l'exposé, que L est divisible par q. Les paquets sont alors transmis les uns après les autres sur le chemin de routage entre la source et la destination. Le premier paquet arrive après un temps $d(\beta + q\tau)$. Puis il arrive un paquet à chaque étape, et comme il reste $\frac{L}{q} - 1$ paquets de taille q, il est nécessaire d'attendre encore $(\frac{L}{q} - 1)(\beta + q\tau)$ pour que la destination ait reçue la totalité du message. Le temps de transmission est alors :

$$\begin{aligned} T_d &= d(\beta + q\tau) + (\frac{L}{q} - 1)(\beta + q\tau) \\ &= (d + \frac{L}{q} - 1)(\beta + q\tau). \end{aligned}$$

Le temps T_d est une fonction de q. La taille de paquet q qui minimise $T_d(q)$ est $q_{min} = \sqrt{\frac{L\beta}{(d-1)\tau}}$. On obtient, pour cette taille de paquet, un coût de

$$\left(\sqrt{(d-1)\beta} + \sqrt{L\tau}\right)^2.$$

Ainsi, pour un L très grand, on obtient un temps de communication dans le mode *store-and-forward* en $L\tau + o(L)$, ce qui masque l'influence de la distance. En cas de messages longs, ce temps devient comparable à celui de l'équation 2.5 obtenue dans le mode de routage *wormhole* et de même, le chemin complet devient inutilisable pour le transfert d'autres messages.

2.5.2 La technique des chemins disjoints

Parmi tous les chemins reliant une source et une destination, il s'agit de trouver un certain nombre de chemins disjoints et de router certains paquets sur ces chemins disjoints [11].

Soit θ le nombre de ces chemins disjoints de longueur au plus $d' \geq d$, on peut découper le message initial de longueur L en θ blocs et router chaque bloc de longueur $\frac{L}{\theta}$ (supposée ici aussi entière pour simplifier la présentation) indépendamment sur chacun des chemins. Le temps de transmission devient :

$$T_d = d'(\beta + \frac{L}{\theta}\tau)$$

dans le mode *store-and-forward*, et

$$T_d = \alpha + d'\delta + \frac{L}{\theta}\tau$$

dans le mode *wormhole*.

De plus, dans le mode de routage *store-and-forward*, on peut *pipeliner* indépendamment les θ blocs sur chacun des chemins pour obtenir un temps de :

$$\left(\sqrt{(d'-1)\beta} + \sqrt{\frac{L}{\theta}\tau}\right)^2,$$

ce qui reste comparable au mode *wormhole* pour des messages assez grands.

Pour conclure, l'utilisation simultanée de θ chemins sur θ parties différentes du message sous l'hypothèse multi-ports, revient à augmenter la bande passante.

2.6 Description des principales topologies

Pour minimiser le temps de communication entre deux nœuds, un des premiers paramètres à minimiser est la longueur du chemin que le message va parcourir

entre les deux nœuds. Une bonne modélisation des architectures à mémoire distribuée est alors donnée par un graphe $G = (V, E)$, où l'ensemble V des sommets du graphe représente les nœuds, et l'ensemble E des arêtes représente les liens de communication entre les nœuds. La théorie des graphes [1] est très utile dans ce cadre-là.

Comme nous l'avons vu au chapitre 1 et dans le chapitre 2, il existe deux types de réseaux : les réseaux point à point et les réseaux multi-étages. Du point de vue des communications les réseaux multi-étages posent moins de problèmes du fait que chaque nœud est relié à tous les autres (le réseau peut être vu comme un réseau complet). Mais il reste des problèmes de congestion lorsque l'on veut réaliser des schémas de communications globales. De plus, dans ces réseaux l'utilisateur n'a pas accès aux commutateurs du réseau, c'est-à-dire qu'il ne peut pas agir sur les chemins empruntés par ses messages. C'est pour cela que nous nous sommes plus particulièrement intéressés aux réseaux point à point.

Parmi les réseaux existant, les hypercubes, les grilles, les grilles toriques et les De Bruijn ont été les plus utilisés et les plus étudiés. La description de ces réseaux va permettre de mettre en valeur leur degré, leur diamètre, leur nombre de nœuds et leur nombre de liens, des paramètres qui leurs sont caractéristiques.

2.6.1 Les hypercubes

L'hypercube est sans doute le réseau qui a d'abord été le plus étudié [11, 18, 16] et le plus utilisé dans des machines parallèles. Ce type de réseau a été employé par Intel dans la série des iPSC, par nCUBE, par FPS et dans les Connection Machines 1 et 2.

On appelle hypercube de dimension n et on note $H(n)$, le graphe dont les sommets sont les mots de longueur n sur un alphabet à deux lettres 0 et 1, et dont deux sommets sont adjacents si et seulement si ils diffèrent en une seule coordonnée. Un sommet, noté $x_1x_2 \cdots x_i \cdots x_n$, est donc relié aux sommets $x_1x_2 \cdots \overline{x_i} \cdots x_n$, avec $i = 1, 2, \ldots, n$ [15].

$H(n)$ est de degré n, de diamètre n, il possède 2^n sommets et $n2^{n-1}$ arêtes.

$H(n)$ peut être défini récursivement à partir de deux copies de $H(n-1)$. Il suffit de relier entre eux les sommets de même numéro. Les sommets du nouvel hypercube sont obtenus en préfixant les sommets des deux copies respectivement par 0 et par 1. Cette propriété est très intéressante du point de vue algorithmique [13].

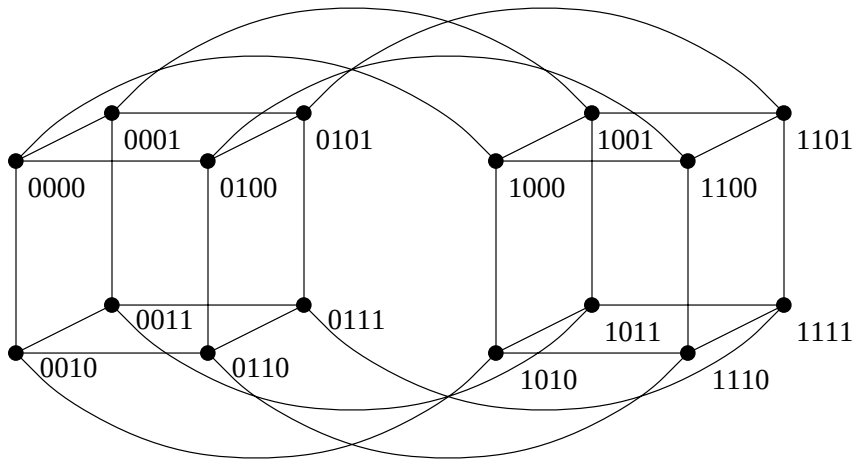


FIG. *2.4 : Hypercube de dimension* 4

2.6.2 Les grilles

Ce réseau a lui aussi été très étudié. Grâce à ces études Intel a construit la Paragon.

La grille de dimension 2, notée $M(p_1, p_2)$, est la somme cartésienne de 2 chaînes ayant p_i sommets, avec $i = 1, 2$.

$M(p_1, p_2)$ est de degré 4 (sauf sur les bords) de diamètre $(p_1 - 1) + (p_2 - 1)$ et possède $p_1 \times p_2$ sommets, et $2p_1p_2 - p_1 - p_2$ arêtes.

Du fait de son diamètre élevé, il faut des liens de communication très rapides pour compenser l'éloignement des nœuds.

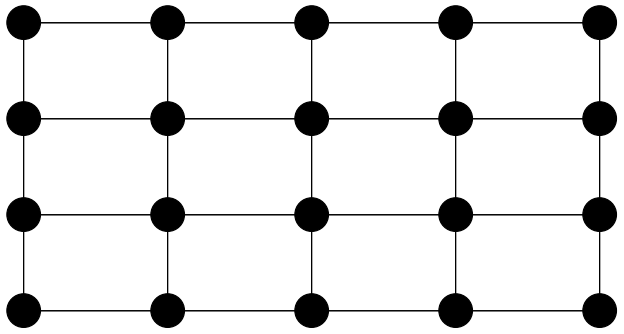


FIG. *2.5 : Grille M(5,4)*

2.6.3 Les grilles toriques

Intel a aussi construit une machine parallèle ayant un réseau en grille torique : il s'agit du iWarp. Des études sur ce réseau ont été menées par Y. Saad et M.H. Schultz [16].

La grille torique de dimension n, notée $TM(p_1, p_2, \ldots, p_n)$, est la somme cartésienne de n cycles C_i, avec $i = 1, 2, \ldots, n$.

$TM(p_1, p_2)$ est de degré 4, de diamètre $\lfloor \frac{p_1}{2} \rfloor + \lfloor \frac{p_2}{2} \rfloor$ et possède $p_1 \times p_2$ sommets, et $2p_1p_2$ arêtes.

L'avantage de ce réseau par rapport à la grille est de diviser le diamètre par un facteur 2 et d'être un graphe plus régulier.

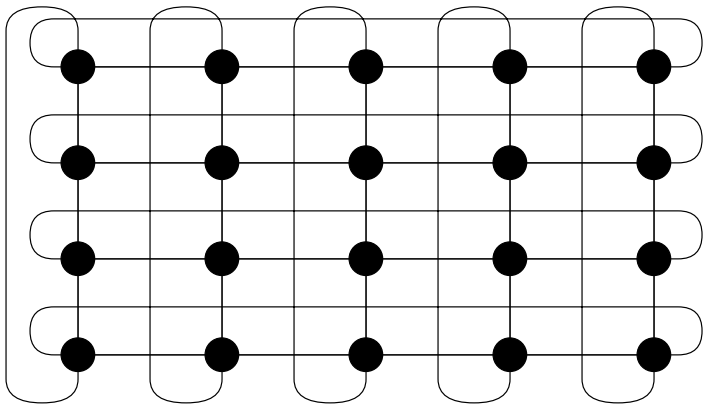


FIG. *2.6 : Grille torique TM(5,4)*

2.6.4 Les de Bruijn

Le graphe de De Bruijn orienté, noté $B(d, D)$, est le graphe dont les sommets sont tous les mots de longueur D sur un alphabet de d lettres.
Pour tout sommet $x_1x_2 \cdots x_D$, il existe un arc de $x_1x_2 \cdots x_D$ vers chaque sommet $x_2x_3 \cdots x_D\lambda$, λ étant une lettre quelconque de l'alphabet. En pratique, on se limite souvent à $d = 2$.

$B(d, D)$ est de degré d, de diamètre D et possède d^D sommets, et d^{D+1} arêtes [7, 10].

L'avantage de ce réseau est de posséder un grand nombre de nœuds pour un diamètre très faible. Pour le moment, ce réseau a été utilisé uniquement par la NASA en traitement du signal [5].

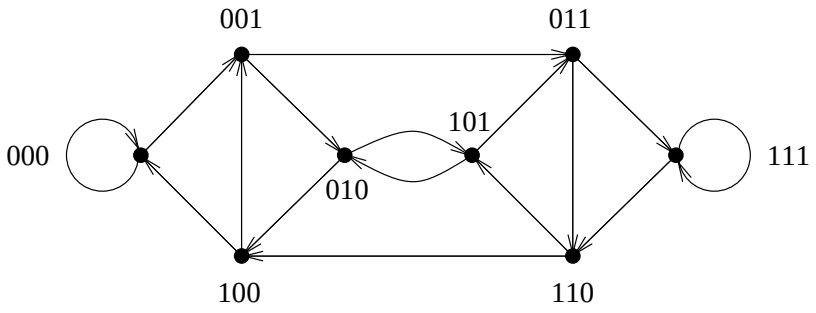


FIG. *2.7 : De Bruijn B(2,3)*

Afin d'obtenir la version non orientée du de Bruijn, il faut supprimer les boucles des nœuds (000...0), (111...1) et ne plus tenir compte de l'orientation des arcs.

2.7 Conclusion

Sur les machines parallèles les communications sont très coûteuses, car la vitesse de transfert sur les liens de communications est nettement inférieure à la vitesse de calcul des processeurs (vitesse de transfert de l'ordre de 10 Mo/s alors que la vitesse de calcul est de l'ordre de 80 MFlops cf. chapitre 1). Le fait de connaître le mode communication, sa modélisation, le type de liens et la topologie du réseau que l'on utilise va nous permettre de développer des stratégies de communications les moins onéreuses possibles.

Bibliographie

[1] C. Berge. *Graphes*. Gauthier-Villars, 1987.

[2] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing*, 11:263–275, 1991.

[3] L. Bomans and D. Roose. Benchmarking the iPSC/2 hypercube multiprocessor. *Concurrency: Practice and Experience*, 1(1):3–18, 1989.

[4] L. Bomans and D. Roose. Communication benchmarks for the iPSC/2. In North-Holland, editor, *Hypercube and Distributed Computers*, pages 93–103, 1989.

[5] O. Collins, S. Dolinar, R. McEliece, and F. Pollara. A V.L.S.I. decomposition of the de Bruijn graphs. *Journal of the ACM*, 39:931–949, 1992.

[6] W.J. Dally and C.L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[7] Jean de Rumeur. *Communications dans les réseaux de processeurs*. collection ERI. Masson, 1994.

[8] C. Seitz et al. Wormhole chip project report. 1985.

[9] P. Fraigniaud. Performance analysis of broadcasting in hypercubes with restricted communication capabilities. *Journal of Parallel and Distributed Computing*, 16(1):15–26, 1992.

[10] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. to appear in Discrete Applied Mathematics, special issue on broadcasting and gossiping.

[11] S.L. Johnsson and C.T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transaction on Computers*, 38(9):1249–1268, 1989.

[12] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. *Computers Networks*, 3:267–286, 1979.

[13] F. T. Leighton. *Introduction to parallel algorithms and architectures*. Morgan Kaufmann, 1991.

[14] J.G. Peters and M. Syska. Circuit-switched broadcasting in torus networks. Cmpt tr 93-04, Simon Fraser University, 1993.

[15] Y. Saad and M. H. Schultz. Topological properties of hypercubes. *IEEE Transaction on Computers*, 37(7):867–871, 1988.

[16] Y. Saad and M.H. Schultz. Data comunication in parallel architectures. *Journal of Parallel Computing*, 11:131–150, 1989.

[17] S. R. Seidel, M. H. Lee, and S. Fotedar. Concurrent bidirectional communication on the Intel iPSC860 and iPSC2. Cs-tr90-06, Université Technique du Michigan (USA), 1990.

[18] Q.F. Stout and B. Wagar. Intensive hypercube communication, prearranged communication in link-bound machines. *Journal of Parallel and Distributed Computing*, 10:167–181, 1990.

[19] A. Touzene and B. Plateau. Optimal multinode broadcast on a mesh connected graph with reduced bufferization. In A. Bode, editor, *Distributed Memory Computing, Lecture Notes in Computer Science 487*, pages 143–152. Springer-Verlag, 1991.

Chapitre 3

Communications globales

3.1 Introduction

Le passage de l'algorithmique séquentielle ou parallèle sur architecture à mémoire partagée, à l'algorithmique parallèle distribuée ne se fait pas sans quelques problèmes. Dans le cas des machines à mémoire partagée, le programmeur ne s'occupe que du découpage de son problème en tâches indépendantes, tandis que sur les machines à mémoire distribuée le programmeur doit en plus s'occuper des échanges de données entre les processeurs. Ces échanges peuvent être réalisés avec des communications point à point, comme nous l'avons vu dans le chapitre précédent. Cependant en étudiant des algorithmes numériques (d'algèbre linéaire, traitement d'images etc.), il apparaît des mouvements de données impliquant l'ensemble des processeurs, ce qui nécessite d'autres schémas de communication. On appellera ces types de mouvements de données des communications globales.

Face au manque d'outils de programmation parallèle (logiciels de débogage, d'analyse de performances, et a fortiori de compilateurs parallèles !) une démarche raisonnable pour paralléliser des applications - en particulier dans le domaine numérique - est de choisir une répartition initiale des données, d'identifier des procédures de calculs locaux et de mettre en œuvre de schémas de communication pour organiser les rencontres entre ces données. Afin de faciliter la tâche du programmeur, les constructeurs ont créé des bibliothèques de calculs et de communications.

Après une présentation des bibliothèques de calculs et de communications disponibles sur les machines à mémoire distribuée, nous décrirons plus précisément les schémas de communications globales, et nous montrerons sur deux exemples, l'algorithme d'élimination de Gauss et le gradient conjugué, la nécessité de tels schémas de communications.

3.2 Les Bibliothèques de calculs et de communications

Pour un programmeur non spécialiste des architectures et de l'algorithmique parallèles, il s'agit de trouver un bon compromis entre l'investissement à fournir pour écrire ses programmes et l'obtention de bonnes performances. Les bibliothèques parallèles permettent de jouer ce rôle. Elles doivent être assez simples d'utilisation, et sont portées sur la plupart des machines parallèles, ce qui permet d'avoir un code portable d'une machine à une autre, et d'être optimisées pour chaque machine parallèle.

3.2.1 Les bibliothèques d'algèbre linéaire

Il existe toutes sortes de bibliothèques de calcul très spécialisées (par exemple pour le traitement du signal, traitement de l'image, etc.), mais au cours de cette thèse, nous nous sommes plus particulièrement intéressés aux bibliothèques générales d'algèbre linéaire. En voici les principales :

- les BLAS (*Basic Linear Algebra Subroutines*) [11, 18] sont des noyaux d'exécution fondamentaux d'algèbre linéaire. L'ensemble de ces procédures permet d'écrire de façon simple les algorithmes, à condition de bien identifier les noyaux de calculs. Il existe trois niveaux de BLAS. Le niveau 1 correspond aux opérations vecteur-vecteur (mise à jour de vecteurs : AXPY, somme avec accumulation :DOT). Le niveau 2 permet de faire des opérations de type matrice-vecteur, et le niveau 3 permet de réaliser des opérations de type matrice-matrice. Toutefois, il faut noter que les BLAS ne fonctionnent pas encore totalement sur les machines à mémoire distribuée. Le projet PUMMA [3] en cours de réalisation a pour but de réaliser des BLAS distribuées de niveau 3.

- LAPACK (*Linear Algebra PACKage*) [9] est une bibliothèque de haut niveau qui comprend la résolution de systèmes linéaires, le calcul des valeurs propres d'une matrice, des factorisations de matrices type LU, Cholesky etc. Cette bibliothèque est le résultat de la fusion de deux autres bibliothèques EISPACK (spécialisée dans la résolution des problèmes de valeurs propres) et LINPACK (bibliothèque de procédures d'algèbre linéaire), et est spécialement conçue pour les machines vectorielles et parallèles à mémoire partagée. Cette bibliothèque est écrite à partir de BLAS de niveau 3.

- ScaLAPACK (*Scalable Linear Algebra PACKage*) [10] est la version LAPACK pour machines parallèles à mémoire distribuée. ScaLAPACK utilise les BLAS distribuées de niveau 3, la bibliothèque de communication BLACS (voir paragraphe 3.2.2), ainsi que des BLAS séquentielles.

- chaque constructeur possède, à des variantes près, des implémentations de ces bibliothèques.

3.2.2 Les bibliothèques de communications

De nombreuses bibliothèques de communications existent, en effet chaque constructeur de machines parallèles développe des bibliothèques efficaces pour ses machines, par exemple NX pour Intel ou CMMD pour Thinking Machines Corporation. Malheureusement les applications écrites avec de telles bibliothèques ne sont pas portables d'une machine à une autre (les modèles de communication étant en général différents). C'est pourquoi des universitaires ont développé des bibliothèques de communication au plus haut niveau d'abstraction, que l'on peut utiliser sur une grande majorité de machines parallèles. Devant le succès de ces bibliothèques, les constructeurs ont décidé de les optimiser pour leurs machines, ce qui permet d'obtenir d'assez bonnes performances avec ces outils. Du fait du grand nombre de bibliothèques existantes, il est impossible de toutes les présenter, voici les plus « populaires ».

- BLACS (*Basic Linear Algebra Communication Subroutines*) [1] est une bibliothèque de communication spécialisée, utilisée pour la parallélisation des BLAS de niveau 3 et de la bibliothèque LAPACK. En fait, c'est une bibliothèque efficace, appliquée aux calculs matriciels.

- MPI (*Message Passing Interface*) [13] n'est pas encore une bibliothèque de communication, mais plutôt une liste de spécifications pour l'échange des messages sur les machines à mémoire distribuée. Les gens qui ont travaillé à ces spécifications sont des universitaires, des industriels (essentiellement des constructeurs de machines), et des spécialistes du domaine des applications. Cela a permi de définir au mieux les besoins et les outils que la future bibliothèque devra posséder. De plus, comme tous les grands constructeurs de machines parallèles participent à MPI, cette bibliothèque sera portée et optimisé'e sur les machines courante.

- PICL (*Portable Instrumented Communication Library*) [15] est basée sur trois niveaux de primitives, un niveau bas, un niveau haut incluant la synchronisation, la diffusion, et un troisième niveau qui permet de générer des traces utilisables par la suite avec *ParaGraph* (logiciel de visualisation de traces).

- PVM (*Parallel Virtual Machine*) [12] est un environnement de gestion de processus plus une bibliothèque de communication utilisable sur réseau de stations de travail. L'avantage de cette bibliothèque est de pouvoir utiliser des machines hétérogènes se trouvant sur des sites différents. De plus, PVM permet d'inclure des machines parallèles dans un réseau. Nous avons vu au chapitre 1 que la nouvelle génération de machines parallèles était construite

avec les mêmes processeurs que ceux des stations de travail. Comme les nœuds des machines parallèles fonctionnent avec des noyaux UNIX, il est facile d'utiliser PVM sur ces machines. Avec PVM, on peut mettre au point une application sur un réseau de stations, et la transférer directement sur une machine parallèle afin d'obtenir de la puissance de calcul sans avoir à changer une seule ligne de code.

3.3 Les schémas de communications globales

Dans les bibliothèques de communications, on trouve un certain nombre de primitives qui permettent de réaliser des communications globales [7].

- **La diffusion** *(one to all, broadcasting)* : un processeur envoie un message à tous les autres processeurs.
- **L'échange total** *(all to all, gossiping, total exchange)* : diffusion simultanée à partir de chacun des processeurs.
- **La distribution** *(personalized one to all, scattering)* : un processeur envoie un message différent à chacun des autres processeurs.
- **La multidistribution** *(personalized all to all, complete exchange, multiscattering)* : distribution simultanée à partir de chacun des processeurs.
- **Le regroupement** *(all to one, gathering)* : tous les processeurs envoient leur message à un processeur donné (c'est l'inverse de la distribution). On parle de combinaison lorsqu'il y a des calculs à chaque étape intermédiaire.

Toutefois, d'autres schémas de communications, ne mettant pas en jeu l'ensemble des processeurs existent comme, par exemple, le *one-to-many* où un processeur envoie un même message à un groupe particulier de processeurs. On peut également considérer toutes les extensions du type *many-to-many*. Mais dans cette thèse nous nous limiterons à l'étude des schémas classiques, c'est-à-dire : diffusion, échange total, distribution, multidistribution et regroupement, en insistant sur la diffusion.

3.4 Utilisation des schémas de communications globales

Dans cette section, nous allons montrer comment peuvent être parallélisés l'élimination de Gauss et le gradient conjugué sur une architecture parallèle à mémoire distribuée et surtout comment les schémas de communications globales interviennent dans cette parallélisation.

Pour cela nous allons considérer un ensemble de p processeurs $P_0, P_1, ..., P_{p-1}$. Ces processeurs sont reliés entre eux par un réseau physique (par exemple arbre binaire, grille, hypercube, De Bruijn etc.). Avant toute chose, lorsqu'on traite un problème sur une machine parallèle à mémoire distribuée il faut se poser la question du placement des données. En algèbre linéaire, trois types de placements de matrices sont employés : répartition par lignes, par colonnes ou par blocs [14]. Dans les trois types d'allocation, il est fait une répartition équitable des données entre les processeurs. Quant aux vecteurs, eux aussi sont répartis équitablement.

3.4.1 L'élimination de Gauss

L'algorithme d'élimination de Gauss permet de résoudre des systèmes linéaires denses. Le principe consiste à transformer la matrice de départ en une matrice triangulaire supérieure en effectuant des combinaisons linéaires entre les lignes. On résout le système triangulaire par substitution arrière [16]. Dans l'étude qui va suivre nous nous limiterons à un placement par colonnes de la matrice. On trouvera dans [6, 8, 20] des études similaires avec les autres types de placement.

Soit un système linéaire $Ax = b$, où A est une matrice dense $n \times n$ et x et b deux vecteurs, l'algorithme de l'élimination de Gauss est le suivant :

Algorithme 3.1 *Elimination de Gauss*

Pour $k \leftarrow 1$ *jusqu'à* $n-1$ *faire*
  $pivot \leftarrow \frac{1}{A_{k,k}}$;
  Pour $i \leftarrow k+1$ *jusqu'à* n *faire*
    $A_{i,k} \leftarrow pivot \times A_{i,k}$;
  Pour $j \leftarrow k+1$ *jusqu'à* n *faire*
    Pour $i \leftarrow k+1$ *jusqu'à* n *faire*
      $A_{i,j} \leftarrow A_{i,j} - A_{i,k} \times A_{k,j}$;

On constate qu'il y a un seul noyau de calcul qui correspond à un AXPY (BLAS 2). La parallélisation de cet algorithme a été très étudiée [5, 6, 19, 22]. Le processeur qui possède à l'étape k l'élément $A_{k,k}$ doit calculer le pivot et mettre à jour sa colonne k, avant de la diffuser à tous les processeurs. L'algorithme parallèle est le suivant :

Algorithme 3.2 *Elimination de Gauss en parallèle*

Pour $k \leftarrow 1$ *jusqu'à* $n-1$ *faire*
  Si le processeur est en charge de l'élément $A_{k,k}$ *alors*
    pivot$\leftarrow \frac{1}{A_{k,k}}$;
    Pour $i \leftarrow k+1$ *jusqu'à* n *faire*
      $A_{i,k} \leftarrow pivot \times A_{i,k}$;
    Diffuser la colonne k;
  Sinon
    Recevoir la colonne k;
    Mettre à jour ses colonnes;

Remarque.
On notera que lors de la parallélisation de cet algorithme, le placement des données initiales de la matrice est important. En effet, si on affecte les $\frac{n}{p}$ premières colonnes au premier processeur, les $\frac{n}{p}$ suivantes au second, etc., il est clair qu'au bout de la $\frac{n}{p}$ étape le premier processeur n'aura plus rien à faire. La solution consiste à répartir les colonnes de façon circulaire : 1, $\frac{n}{p}$, $2\frac{n}{p}$, etc. sur le premier processeur, 2, $\frac{n}{p}+1$, $2\frac{n}{p}+1$, etc. sur le deuxième...

3.4.2 Le gradient conjugué

L'algorithme du gradient conjugué [17] est une méthode itérative pour résoudre des systèmes linéaires. Cette méthode a été très étudiée pour résoudre des grands systèmes creux à matrice définie positive symétrique [2, 16]. De plus, couplé à un préconditionnement, cette méthode converge très rapidement. Dans l'étude qui va suivre nous nous limiterons à un placement par lignes de la matrice.

Pour un système linéaire $Ax = b$, où A est une matrice symétrique définie positive $n \times n$ et x et b deux vecteurs, l'algorithme du gradient conjugué avec préconditionnement est le suivant :

Algorithme 3.3 *Gradient conjugué*

$Initialisation(x, d, r, z, M);$
$old =< r, z >;$
$new = 0;$
Faire
$\lambda = \frac{old}{<Ad,d}$;
$x = x + \lambda d;$
$r = r + \lambda Ad;$
Résoudre $Mz = r;$
$old = new;$
$new =< r, z >;$
$\beta = \frac{new}{old};$
$d = -z + \beta d;$
Tant que $< r, r > \leq$ *tolérance;*

où $<,>$ correspond au produit scalaire, et M correspond à la matrice de préconditionnement. Il existe plusieurs choix possibles pour M, soit M est la diagonale de A, soit un polynôme de faible degré en A, soit une décomposition triangulaire de A.

Du point de vue séquentiel, si on identifie les noyaux de calcul, on obtient l'algorithme suivant :

Algorithme 3.4 *Noyaux de calcul du gradient conjugué*

H(1) calcul du produit matrice-vecteur $v = Ad;$
(2) calcul du produit scalaire $< v, d >;$
(3) calcul du scalaire $\lambda;$
(4) calcul de 2 AXPY $x = x + \lambda d$ *et* $r = r + \lambda v;$
(5) résoudre $Mz = r;$
(6) calculer le produit scalaire $< r, z >;$
(7) calcul du scalaire $\beta;$
(8) calcul d'un AXPY $d = -z + \beta d;$

Cette présentation permet d'identifier huit noyaux de calculs. Le noyau (1) est le plus coûteux $O(n^2)$ et correspond à des BLAS de niveau 2 [11]. Les noyaux (2), (4), (6) et (8) correspondent à des BLAS de niveau 1 (DOT ou AXPY) de complexité $O(n)$. Les noyaux (3) et (7) correspondent à deux divisions. La complexité du noyau (5) dépend du choix fait pour M au départ, mais dans le pire des cas ce noyau de calcul coûte $O(n^2)$.

parallélisation de l'étape (1)

Cette étape correspond à un produit matrice-vecteur. Afin que chaque processeur puisse calculer les composantes du vecteur $v = Ad$, il faut que tous les éléments $a_{i,j}$ rencontrent les éléments d_j. Comme nous avons réalisé un placement par lignes de la matrice, tous les processeurs possèdent des lignes complètes. Malheureusement, il n'en n'est pas de même pour les composantes du vecteur d. Il est donc nécessaire d'effectuer un échange total, afin que les processeurs possèdent les composantes du vecteur d.

parallélisation de l'étape (2)

Cette étape consiste à calculer un produit scalaire distribué. Dans un premier temps chaque processeur va calculer un produit scalaire partiel avec les composantes dont il dispose, puis chaque processeur va envoyer le résultat de son produit scalaire à un processeur particulier (P_0 par exemple) par un schéma de communication de regroupement. Ce processeur effectue la somme des produits scalaires partiels, puis diffuse le résultat à tous les processeurs.

Remarque.
Il est possible de paralléliser cette étape d'une autre façon. En effet, chaque processeur peut calculer son produit scalaire partiel, mais au lieu d'effectuer un regroupement puis une diffusion, l'ensemble des processeurs effectue un échange total. Ainsi, chaque processeur effectue la somme des produits scalaires partiels afin d'obtenir le résultat. Cette deuxième solution est d'autant meilleure, qu'elle est réalisable en un nombre d'étapes de communication égal au diamètre de la topologie du réseau, alors que la première solution est en deux fois le diamètre.

parallélisation de l'étape (5)

En fait la parallélisation du préconditionnement dépend du choix de celui-ci.

- si M est la diagonale de A, le système de préconditionnement peut être résolu en effectuant une division sur chaque composante. En parallèle, cela peut se faire uniquement par des calculs locaux sur les morceaux de vecteurs que possède chaque processeur : AXPY locaux.

- si M est un polynôme de faible degré en A, il faut calculer successivement Ar, $A(Ar)$, $A(A^2r)$, etc. En stockant les calculs précédents, à chaque itération il n'y a qu'un produit matrice-vecteur à effectuer. Du point de vue de la parallélisation, cette étape est identique à l'étape (1).

- si M est une matrice triangulaire supérieure, ou inférieure, ou le produit de deux matrices triangulaires $M = LL^t$, le système de préconditionnement met en œuvre la résolution d'un ou de deux systèmes triangulaires. Dans

le pire des cas, il faut effectuer deux résolutions successives $Ly = r$ et $L^t z = y$ où y est un vecteur intermédiaire. Nous ne détaillerons pas cette phase de parallélisation (pour cela se référer aux articles de Coleman et Li [4] et Ortega et Romine [21] sur hypercube), mais nous allons juste donner le principe de la résolution d'un système triangulaire supérieur. Comme dans l'élimination de Gauss, nous considérons, une allocation circulaire des lignes par processeur afin que ceux-ci travaillent le plus longtemps possible. A l'étape k de la résolution triangulaire, on calcule l'élément x_{n-k}. Afin que l'ensemble des processeurs puissent effectuer la mise à jour de leur solution partielle, il est nécessaire que le processeur qui vient de calculer x_{n-k} le diffuse à l'ensemble des processeurs.

Les noyaux (3), (4), (7) et (8) sont des calculs locaux que tous les processeurs effectuent sur leurs données. Le noyau (6) est identique au noyau (2), donc pour sa parallélisation se référer à la parallélisation du noyau (2). L'algorithme parallèle est le suivant :

Algorithme 3.5 *Algorithme parallèle du gradient conjugué*

Echange total de taille $(\frac{n}{p})$ *avec les composantes de* d*;*
Calculer $\frac{n}{p}$ *produits scalaires de taille* n*;*

Calculer un produit scalaire de taille $\frac{n}{p}$*;*
Regroupement de taille 1 vers le processeur P_0
Si processeur P_0 *alors calculer la somme de tous les morceaux reçus;*
Diffusion de taille 1 depuis le processeur P_0*;*

Calculer le scalaire λ*;*

Calculer $x = x + \lambda d$*;*
Calculer $r = r + \lambda v$*;*

Calculer un produit scalaire de taille $\frac{n}{p}$*;*
Regroupement de taille 1 vers le processeur P_0
Si processeur P_0 *alors calculer la somme de tous les morceaux reçus;*
Diffusion de taille 1 depuis le processeur P_0*;*

Calculer le scalaire β*;*

calculer $d = -z + \beta d$*;*

Afin de simplifier l'algorithme nous avons omis la phase de préconditionnement. De plus, pour cet algorithme nous avons considéré la matrice A comme

dense, ce qui n'est pas toujours vrai. Dans le cas où la matrice A est creuse, il n'est pas nécessaire d'effectuer un échange total, mais seulement un échange total partiel (*many-to-many*). Il en va de même pour le regroupement et la diffusion.

Récapitulatif

Afin de résumer l'importance des schémas de communications globales, voici un tableau récapitulatif, donnant une liste (non exhaustive) d'applications numériques utilisant ces schémas.

diffusion	résolution de systèmes linéaires, élimination de Gauss, etc.
échange total	produit matrice-vecteur (placement par lignes), algorithme de Durand-Kerner, etc.
distribution	chargement de code, chargement de données
multidistribution	produit matrice-vecteur (placement par colonnes), transposition, etc.
regroupement	produit scalaire distribué, calcul de test d'arrêt

3.5 Conclusion

Nous venons de montrer que les schémas de communications globales sont nécessaires à la parallélisation des algorithmes numériques tels que l'élimination de Gauss, le gradient conjugué et le produit matrice-vecteur. C'est pour cela que l'étude et l'optimisation de tels schémas de communications sont importantes. En effet c'est en utilisant des noyaux de calculs optimisés (BLAS,LAPACK) et des bibliothèques de communications optimisées que les utilisateurs de machines parallèles à mémoire distribuée obtiendront de bonnes performances. Mais du fait du grand nombre de topologies des machines parallèles, il est nécessaire de créer des bibliothèques de communications efficaces pour toutes ces topologies. Actuellement ces bibliothèques mettent en œuvre des mécanismes de gestion logicielle des messages, qui sont encore fort pénalisant, afin de réduire ces coûts de gestion, une solution serait des mécanismes de gestion matérielle.

Bibliographie

[1] E. Anderson, A. Benzoni, J. Dongara, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. Van de Geijn. BLACS: Basic linear algebra communication subroutines. In IEEE Computer Society Press, editor, *Sixth Distributed Memory Computing Conference - Portland (USA)*, pages 287–290, 1991.

[2] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. A taxonomy for conjugate gradient methods. *SIAM J. Numer. Anal.*, Vol . 27, 1990.

[3] J. Choi, J. J. Dongarra, and D. W. Walker. PUMMA: parallel universal matrix multiplication algorithms on distributed concurrent computers. Technical Report ORNL/TM-12252, Oak Ridge National Laboratory (USA), April 1993.

[4] T. F. Coleman and G. Li. A parallel triangular solver for hypercube multiprocessor. Technical Report TR 86-787, Department of Computer Science, Cornell University, October 1986.

[5] M. Cosnard, M. Marrakchi, Y. Robert, and D. Trystram. Gauss elimination for MIMD computers. *Parallel Computing*, 1987.

[6] M. Cosnard and D. Trystram. *Algorithmes et architectures parallèles.* InterEditions, 1993.

[7] Jean de Rumeur. *Communications dans les réseaux de processeurs.* Collection ERI. Masson, 1994.

[8] Jean de Rumeur. *Communications dans les réseaux de processeurs.* Collection ERI. Masson, 1994.

[9] J. Demmel, J. J. Dongara, and W. Kahan. LAPACK working note. LAPACK: on designing portable high-performance numerical libraries. Technical Report 39, Department of Computer Science - University of Tennessee (USA), 1991.

[10] J. Demmel, J. J. Dongarra, R. Van De Geijn, and D. W. Walker. LAPACK for distributed memory architectures: the next generation. In SIAM, editor, *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 323–329, 1993.

[11] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of FORTRAN basic linear algebra subroutines. *ACM Transaction on Mathematical Software*, 14(1):1–17, March 1988.

[12] J. J. Dongarra, A. Geist, R. Manchek, and W. Jiang. Using PVM 3.0 to run grand challenge applications on a heterogenous network of parallel computers. In SIAM, editor, *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 873–877, 1993.

[13] J. J. Dongarra, R. Hempel, A. J. G. Hey, and D. W. Walker. A proposal for a user-level message passing interface in a distributed memory environment. Technical report, Oak Ridge National Laboratory (USA), February 1993.

[14] J. J. Dongarra and D. Walker. LAPACK working note: the design of linear algebra libraries for high performance computer. Technical Report 58, Department of Computer science - University of Tennessee, 1993.

[15] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. PICL: a portable instrumented communication library. Technical Report ORNL/TM-11130, Oak Ridge National Laboratory (USA), July 1990.

[16] G. H. Golub and C. F. Van Loan. *Matrix Computations; second edition.* The John Hopkins University Press, 1989.

[17] M. R. Hestenes and D. E. Stiefel. Methods of conjugate gradient for solving linear systems. *Journal Res. Nat. Bur. Stan.*, Vol . 49, 1952.

[18] C. Lawson, R. Hanson, D. Kincaind, and F. Krogh. Basic linear subprograms for FORTRAN usage. *ACM Transaction on Mathematical Software*, 5:308–323, 1979.

[19] P. Quinton and Y. Robert. *Algorithmes et architectures systoliques.* collection ERI. Masson, 1989.

[20] Y. Robert, B. Tourancheau, and G. Villard. Data allocation strategies for the Gauss and Jordan algorithms on a Ring of Processors. *Information Processing Letters*, 31:21–29, 1989.

[21] C. H. Romine and J. M. Ortega. Parallel solution of triangular system of equations. Technical Report RM-86-05, Department of Applied Mathematics, University of Virginia, 1986.

[22] Y. Saad. Gaussian elimination on hypercubes. In M. Cosnard and al., editors, *Parallel Algorithms and Architectures*, pages 5–17. Elsevier Science Publishers, 1986.

Chapitre 4

Diffusion dans les grilles toriques

Au cours de la dernière décennie, beaucoup d'études ont été réalisées sur les communications dans les hypercubes. Ceci était dû au fait qu'un grand nombre de machines parallèles utilisaient ce type de réseau d'interconnexion. Mais l'augmentation de la demande en puissance de calcul a conduit les constructeurs de machines parallèles à assembler plusieurs centaines de nœuds performants, ce qui posait de gros problèmes lors de la conception de machines ayant une topologie en hypercube. En effet, comme nous l'avons vu au chapitre 1 le nombre de liens sortants d'un processeur n'est pas infini, ce qui limite le nombre de connexions et par là même, le nombre de nœuds de la machine. De plus, ces dernières années est apparue la notion de machine extensible (*scalable*), c'est-à-dire qu'à partir d'une machine de taille donnée, on peut augmenter le nombre de nœuds de celle-ci, en rajoutant simplement des cartes. Ceci n'est réalisable matériellement que sur des topologies dont le degré de chaque nœud est indépendant du nombre de nœuds du réseau, ce qui n'est pas le cas de l'hypercube. C'est pourquoi les réseaux d'interconnexion à degré faible et fixé (comme les grilles et les grilles toriques) se sont développés. Ce type de réseau d'interconnexion est la base de machines telles que la Paragon et le iWarp d'Intel, l'AP1000 de Fujitsu et plus récemment le Cray T3D (voir chapitre 1).

Mais en contrepartie ces réseaux ont l'inconvénient d'avoir des diamètres élevés, c'est pourquoi l'étude de schémas de communications globales est importante. Dans ce chapitre, nous allons nous intéresser au problème de la diffusion dans les grilles toriques. Dans un premier temps nous décrirons la construction d'arbres de recouvrement deux à deux disjoints de profondeur minimale dans la grille torique orientée. Nous pouvons utiliser ces arbres pour réaliser des diffusions pipeline avec des communications *store and forward full-duplex*. Enfin une étude similaire est réalisée pour la grille torique non-orientée.

4.1 Notations et modèle de communication

Dans ce paragraphe nous considérons des grilles toriques ayant p processeurs, où p est un carré parfait. Chaque nœud est repéré par ses coordonnées cartésiennes (i, j) telles que :

$$|i| \leq \lfloor \frac{\sqrt{p}}{2} \rfloor \text{ et } |j| \leq \lfloor \frac{\sqrt{p}}{2} \rfloor \text{ si } \sqrt{p} \text{ est impair ;}$$

$$-\lfloor \frac{\sqrt{p}}{2} \rfloor + 1 \leq i \leq \lfloor \frac{\sqrt{p}}{2} \rfloor \text{ et } -\lfloor \frac{\sqrt{p}}{2} \rfloor \leq j \leq \lfloor \frac{\sqrt{p}}{2} \rfloor - 1 \text{ si } \sqrt{p} \text{ est pair ;}$$

L'initiateur de la diffusion est en position $(0, 0)$. Il sera représenté en grisé et sera positionné au centre des différentes figures.

Pour simplifier les notations, on notera x_{min}, x_{max}, y_{min}, y_{max}, les bornes supérieures et inférieures atteintes par i et j.

Rappelons que le diamètre D d'une grille torique $\sqrt{p} \times \sqrt{p}$ est égal à $2\lfloor \frac{\sqrt{p}}{2} \rfloor$ c'est-à-dire $\sqrt{p}$ si $\sqrt{p}$ est pair, ou $\sqrt{p} - 1$ si $\sqrt{p}$ est impair.

Pour que cette étude soit utilisable sur des machines parallèles à mémoire distribuée, nous avons choisi le modèle de communication le plus répandu sur ces machines, c'est-à-dire un modèle Δ-port (*link-bounded*) [5] avec des liens de communication *full-duplex*, et une transmission de messages entre nœuds par commutation de messages (*store and forward*) [6, 8, 10]. Le temps de communication d'un message de taille L entre deux nœuds voisins est : $\beta + L\tau$ où β est le *start-up* et τ le taux de transmission (voir chapitre 2).

Par abus, on dira qu'un nœud (i, j) appartient au bord de la grille torique si $i \leq x_{min} - 1$ ou $x_{max} + 1 \leq i$ ou $j \leq y_{min} - 1$ ou $y_{max} + 1 \leq j$.

4.2 Les outils de la diffusion

Les techniques utilisées pour l'étude des communications dans un réseau d'interconnexion proviennent de la théorie des graphes [2]. En effet, le réseau peut être modélisé par un graphe $G = (V, E)$, où l'ensemble V des sommets du graphe représente les nœuds, et l'ensemble E des arêtes représente les liens de communication entre les processeurs.

Afin de bien comprendre les autres outils nécessaires à la diffusion dans les grilles toriques, examinons ceux utilisés lors de la communication entre deux nœuds (OTO). Lors du chapitre 2 on a vu que l'utilisation de la technique du pipeline couplée à l'utilisation de chemins disjoints permettait de réduire considérablement le temps de communication entre deux nœuds. Pour les grilles toriques, une étude réalisée par Saad et Schultz aboutit à la proposition suivante :

Proposition 4.2.1 *Il existe quatre chemins disjoints deux à deux reliant tout couple de nœuds A et B dans une grille torique. De plus si A et B ne sont pas alignés, on peut choisir ces chemins de longueur égale à d(A,B)+2 ; si A et B sont alignés mais non voisins, alors on peut choisir ces chemins de longueur distance égale à d(A,B)+4 ; et si A et B sont voisins, alors on peut choisir ces chemins de longueur 7.*

La figure 4.1 illustre cette proposition.

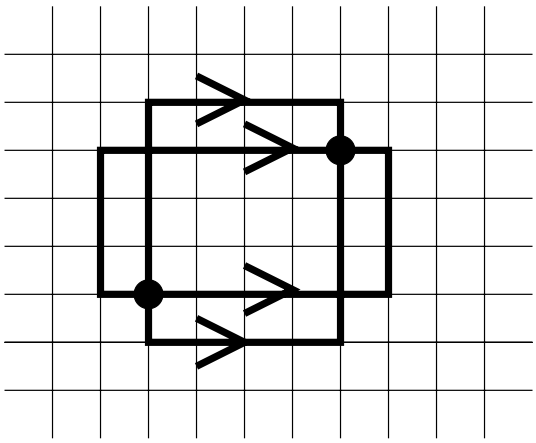


FIG. *4.1 : Envoi en utilisant quatre chemins de longueur distance(A,B)+2 .*

De plus, dans le modèle *store and forward*, les nœuds intermédiaires gardent trace en mémoire de tous les messages qui les traversent, d'où l'idée, pour diffuser dans la grille torique, de construire à partir de la racine R (initiatrice de la diffusion) le plus grand nombre possible d'arbres de recouvrement (*spanning trees*) arcs disjoints.

Rappels :

- Etant donnés un graphe $G = (V, E)$ et un arbre orienté T de racine x ($x \in V$), on dit que T est un arbre de recouvrement ou couvrant s'il contient tous les nœuds du graphe.

- Deux arbres de recouvrement de G sont dit arcs disjoints s'ils n'ont pas d'arc en commun.

Le nombre d'arbres de recouvrement peut être donné par le théorème d'Edmonds [3]. Dans son théorème Edmonds utilise l'arc-connexité, cette notion est définie de la façon suivante :

Définition. *L'arc-connexité d'un graphe connexe G, noté $\lambda(G)$, est le nombre minimum d'arcs qu'il faut supprimer pour que G ne soit plus connexe.*

Théorème 4.2.2 *Soit $G = (V, E)$ un graphe orienté d'arc-connexité λ et soit x un sommet quelconque de V. Il existe au moins λ arbres de recouvrement arcs disjoints dans G de racine x.*

Comme dans la grille torique l'arc-connexité est égale au degré, nous en déduisons qu'à partir de chaque nœud de la grille torique il existe quatre arbres de recouvrement arcs disjoints.

Les quatre arêtes incidentes à chaque nœud sont caractérisées par les quatre points cardinaux N, E, S, W. Comme les liens sont orientés nous distinguerons N_i, E_i, S_i, W_i les arcs entrants d'un nœud, et N_o, E_o, S_o, W_o les arcs sortants d'un nœud (voir figure 4.2).

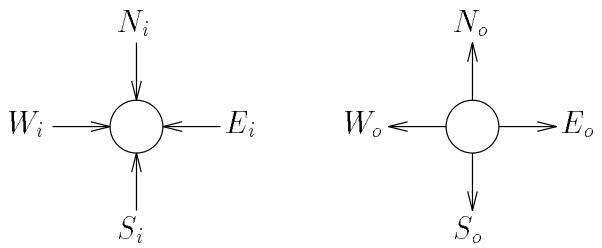


FIG. *4.2 : Les quatre arcs entrants et sortants d'un nœud.*

Chaque arbre est étiqueté par ces mêmes points cardinaux. L'arbre de recouvrement à arcs disjoints noté EST pour *East Spanning Tree* (respectivement NST, SST, WST) correspond à l'arbre qui utilise l'arc E_o (respectivement N_o, S_o, W_o) de la racine.

4.3 Une première solution

En utilisant le principe des arbres de recouvrement à arcs disjoints, Fraigniaud [4] donne la construction de quatre arbres de profondeur $2\sqrt{p}$. En fait il s'agit là de la formalisation d'un résultat de Saad et Schultz [8]. La figure 4.3 montre la construction de l'arbre EST.

Les trois autres arbres sont obtenus par rotations successives de $\frac{\pi}{2}$ de l'arbre EST. Il est facile de vérifier qu'ils sont arcs disjoints.

En pipelinant l'envoi des messages sur les quatre arbres [8], on obtient alors un temps de diffusion de :

$$T_{diffusion} = \left(\sqrt{(2\sqrt{p}-1)\beta} + \sqrt{\frac{L}{4}\tau}\right)^2 \tag{4.1}$$

La profondeur de ces arbres est grande, ce qui a pour conséquence de produire un *start-up* important et de rendre coûteuse cette technique de diffusion pour des

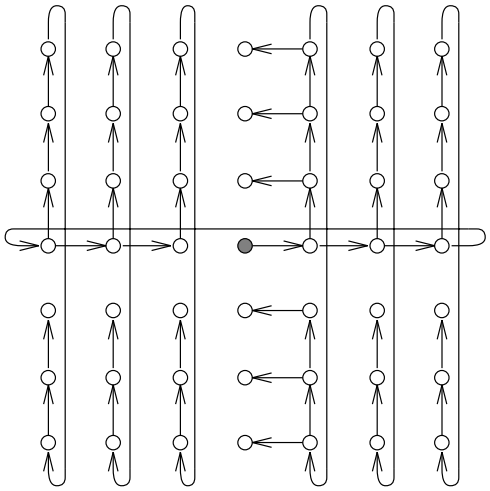


FIG. *4.3 : Arbre de recouvrement EST de profondeur* $2\sqrt{p}$.

messages de petite taille, d'où l'idée de trouver la profondeur minimale des arbres de recouvrement que l'on peut construire dans les grilles toriques, et de construire de tels arbres.

4.4 Des arbres de recouvrement de profondeur minimale

Le diamètre d'une grille torique étant $D = 2\lfloor\frac{\sqrt{p}}{2}\rfloor$, il était possible d'espérer trouver quatre arbres ayant cette profondeur, mais la proposition suivante montre que ceci est impossible et donne une borne inférieure sur la profondeur de tels arbres.

Proposition 4.4.1 *Dans une grille torique de diamètre D quatre arbres de recouvrement à arcs disjoints ayant la même racine sont de profondeur au moins $D+1$.*

Preuve. Soit R la racine des quatre arbres, et P_i $(1 \leq i \leq 4)$ ses quatre voisins directs. Sous l'hypothèse d'avoir des arcs disjoints, chaque arbre utilise exactement 1 arc sortant de R, ce qui « force » une direction de départ pour chaque arbre. Les nœuds à distance D des P_i dans la grille torique, sont donc dans l'arbre correspondant à distance $D+1$ de la racine R.

□

Pour espérer atteindre cette borne lors de la construction des arbres de recouvrement, il faut aller au plus vite aux nœuds les plus éloignés. De plus chaque

arbre ne doit pas utiliser plus d'un quart des arcs du graphe. Enfin pour des raison de simplicité nous cherchons des arbres qui se déduisent les un des autres par rotations ou symétries. Ces trois remarques nous ont permis de construire un principe de base applicable pour toutes les tailles de grilles toriques. Après avoir présenté ce principe, nous présenterons la construction des arbres pour les différentes tailles de grilles toriques, et nous montrerons que ces arbres sont arcs-disjoints.

4.4.1 Le principe de base

Le principe de base que nous allons présenter s'applique à tous les nœuds sauf ceux situés sur les bords. Afin de mieux comprendre ce principe, et de démontrer plus facilement par la suite que ce principe permet de construire des arbres disjoints, la grille (grille torique à laquelle on a supprimer les bords) est divisée en six zones (voir figure 4.4).

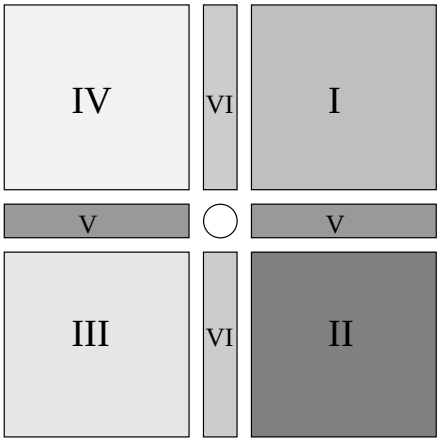


FIG. *4.4 : Représentation des six zones.*

Expliquer en détail la construction des arbres (arcs entrants et sortants pour chaque nœud) nuirait à la clarté du document, c'est pourquoi nous ne détaillons que les arcs entrants de chaque nœud. Voici la construction du principe de base dans chacune des zones, pour un arbre de recouvrement EST dans une grille torique pour les nœuds non situés sur le bord.

- Dans la zone I, tous les nœuds des colonnes impaires sont atteints par leur lien S_i, et les nœuds des colonnes paires, par leur lien E_i ;
- Dans la zone II, les liens N_i sont utilisés pour atteindre les nœuds des colonnes impaires et les liens E_i pour atteindre les nœuds des colonnes paires ;

- Dans la zone III, tous les nœuds des lignes paires sont atteints par leur lien E_i, et les nœuds des lignes impaires par leur lien S_i ;
- Dans la zone IV, les liens E_i sont utilisés pour atteindre les nœuds des lignes paires et les liens N_i pour atteindre les nœuds des lignes impaires ;
- Dans la zone V, tous les nœuds sont atteints par leur lien E_i ;
- Dans la zone VI, tous les nœuds sont atteints par leur lien W_i ;

La figure 4.5 illustre cette construction.

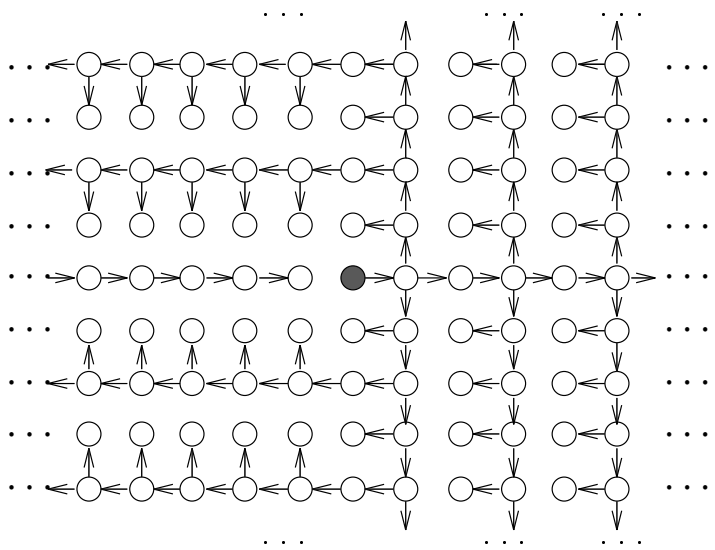


FIG. *4.5 : Le principe de base pour l'arbre EST.*

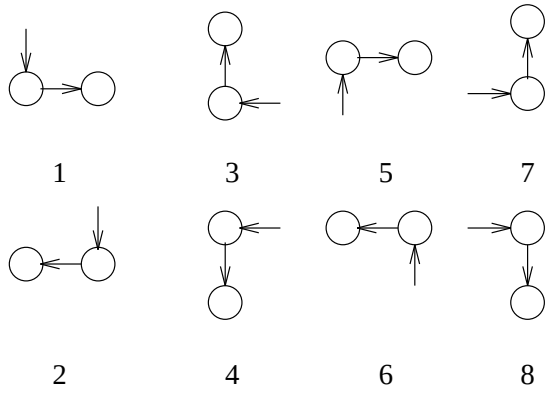


FIG. *4.6 : Liste des « crochets » possibles.*

En appliquant le principe de base nous n'utilisons qu'une colonne sur deux dans les zones I et II, et une ligne sur deux dans les zones III et IV, il est facile

de constater que les bords vont demander un traitement particulier. Une solution pour atteindre ces bords est l'utilisation de « crochets ». Les « crochets » possibles sont au nombre de huit, ils sont issus de motifs du principe de base. La figure 4.6 montre les huit types possibles.

De plus si $\sqrt{p}$ est pair, l'émetteur ne peut plus être vu comme le centre de la grille torique. Il s'avère que nous devons étudier quatre cas : $4k+1$, $4k+3$ pour $\sqrt{p}$ impair, et $4k$, $4k+2$ pour $\sqrt{p}$ pair.

4.4.2 Cas 1 : 4k+1

Pour ce type de grilles toriques il est facile d'obtenir des arbres de recouvrement de profondeur $D+2$. En effet il suffit de continuer le principe de base jusqu'aux dernières lignes, c'est-à-dire $j = y_{min}$ et $j = y_{max}$. Par contre, les nœuds de la dernière colonne $i = x_{max}$ sont atteints par des « crochets » de types 7 et 8. Si on regarde plus précisément le « crochet » de type 8 entre les nœuds $(x_{max}-1, y_{max})$, (x_{max}, y_{max}), $(x_{max}, y_{max}-1)$ on constate que ce « crochet » se transforme par rotation en « crochet » de type 5 entre les nœuds $(x_{min}, y_{max}-1)$, (x_{min}, y_{max}), $(x_{min}+1, y_{max})$, en « crochet » de type 2 entre les nœuds $(x_{max}, y_{min}+1)$, (x_{max}, y_{min}), $(x_{max}-1, y_{min})$, et en « crochet » de type 3 entre les nœuds $(x_{min}+1, y_{min})$, (x_{min}, y_{min}), $(x_{min}, y_{min}+1)$. Comme à aucun moment deux arcs ne se superposent, l'utilisation du « crochet » de type 8 pour atteindre la colonne $i = x_{min}$ est possible. En effectuant le même raisonnement on peut justifier l'utilisation des autres « crochets » de types 7 et 8.

Les nœuds de la première colonne $i = x_{min}$ sont atteints depuis les nœuds de la colonne $i = x_{max}$ par les arcs W_i, ceci est possible car les autres arbres sont obtenus par rotations successives. En effet le seul arbre qui utilise ce lien est l'arbre WST, mais comme il l'utilise dans l'autre sens, il n'y a pas de problème. La figure 4.7 donne la construction de l'arbre EST. Les trois autres arbres sont obtenus par des rotations successives de $\frac{\pi}{2}$ de l'arbre EST.

Si l'on désire réduire le diamètre de ces arbres, tout en conservant la construction obtenue par le principe de base, il faut « casser » les chemins les plus longs. La figure 4.8 montre ces chemins, et les quatre nœuds à distance $D+2$ (en noir sur la figure). Ces quatre nœuds sont : $(x_{min}, y_{max}-1)$, $(x_{min}, y_{min}+1)$, $(x_{min}+1, y_{max}-1)$, $(x_{min}+1, y_{max}-1)$.

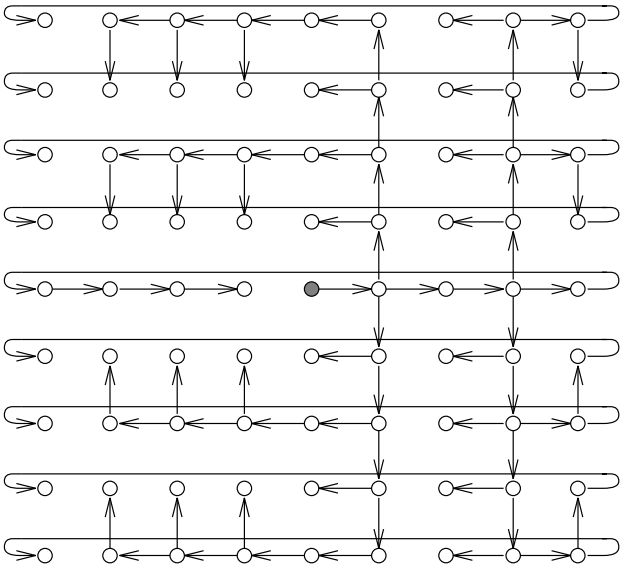


FIG. *4.7 : L'arbre EST de profondeur* $D+2$ *dans une grille torique* 9×9.

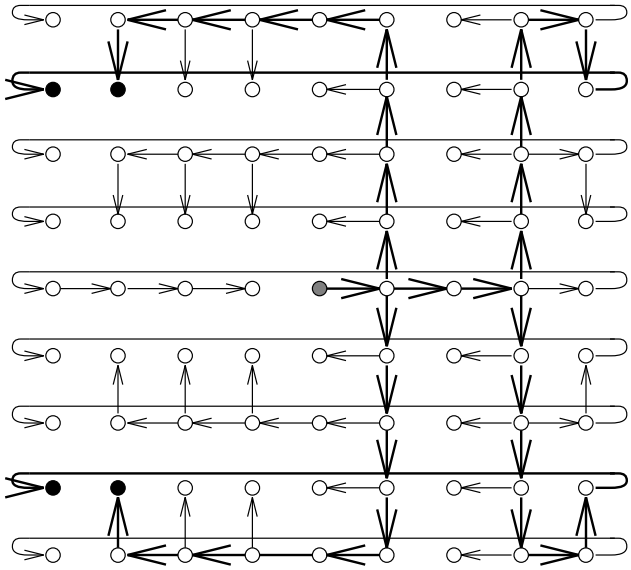


FIG. *4.8 : Les chemins de longueur* $D+2$ *de l'arbre EST.*

Pour réduire la distance entre l'émetteur $(0,0)$ et $(x_{min}, y_{max}-1)$, il est nécessaire de modifier le chemin entre ces deux points. On constate que le« crochet » de type 8 entre les nœuds $(x_{max}-1, y_{max})$, (x_{max}, y_{max}), $(x_{max}, y_{max}-1)$ augmente la longueur du chemin, d'où la nécessité de supprimer ce « crochet ».

La solution consiste à atteindre ces nœuds depuis le nœud $(x_{max}, y_{max}-2)$ en remontant le long du bord droit $(i = x_{max})$ jusqu'à (x_{max}, y_{max}). Les nœuds

$(x_{max}-1, y_{max})$ et $(x_{max}-1, y_{max}-1)$ sont atteints depuis les nœuds du bord par leur lien E_i. Par ailleurs, comme on désire construire des arbres de recouvrement arcs disjoints par rotation, les arcs $(x_{max}, y_{max}-2)$ vers $(x_{max}, y_{max}-1)$ et $(x_{max}, y_{max}-1)$ vers $(x_{max}-1, y_{max})$ nous obligent à supprimer les arcs $(x_{min}+2, y_{max})$ vers $(x_{min}+1, y_{max})$ et $(x_{min}+1, y_{max})$ vers $(x_{min}+1, y_{max}-1)$ de l'arbre EST. En effet, ces arcs sont maintenant utilisés par l'arbre NST.

La même opération doit être faite sur le « crochet » de type 7 formé par les nœuds $(x_{max}-1, y_{min})$, (x_{max}, y_{min}), $(x_{max}, y_{min}+1)$. Ces nouvelles modifications ont, pour les mêmes raisons que précédemment, des conséquences sur les arcs entre les nœuds $(x_{min}+2, y_{min})$ vers $(x_{min}+1, y_{min})$ et $(x_{min}+1, y_{min})$ vers $(x_{min}+1, y_{min}+1)$.

L'ensemble de ces modifications permet de réduire d'1 la distance entre les nœuds $(x_{min}, y_{min}+1)$, $(x_{min}, y_{max}-1)$ et l'émetteur. Le fait supprimer les arcs $(x_{max}-1, y_{max}-2)$ vers $(x_{max}-1, y_{max}-1)$ et $(x_{max}-1, y_{max}-1)$ vers $(x_{max}-1, y_{max})$, nous permet d'utiliser les arcs $(x_{min}+1, y_{max}-2)$ vers $(x_{min}+1, y_{max}-1)$ et $(x_{min}+1, y_{max}-1)$ vers $(x_{min}+1, y_{max})$, sinon ces arcs auraient été utilisés par l'arbre WST. De même, il est possible d'utiliser les arcs $(x_{min}+1, y_{min}+2)$ vers $(x_{min}+1, y_{min}+1)$ et $(x_{min}+1, y_{min}+1)$ vers $(x_{min}+1, y_{min})$.

Après cette première série de modifications, certains nœuds (au nombre de quatre) se trouvent isolés (c'est-à-dire qu'il n'appartiennent plus à l'arbre EST) comme le montre la figure 4.9. Ces nœuds sont représentés en noirs.

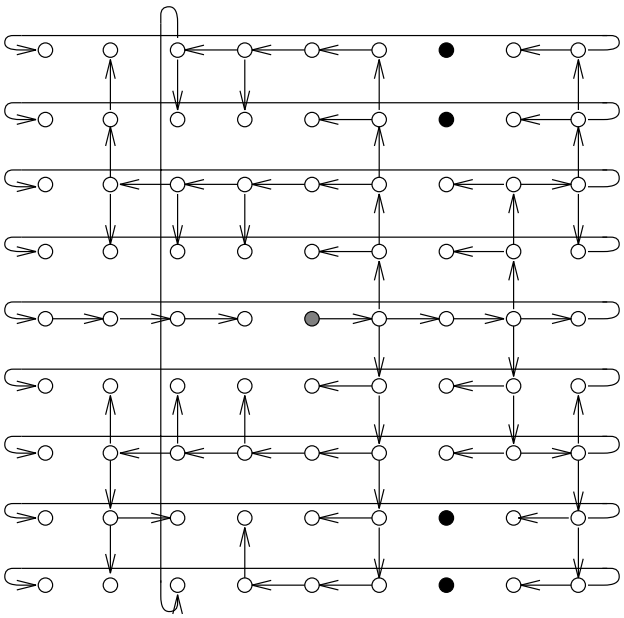


FIG. *4.9 : Arbre EST après la première série de modifications.*

Pour atteindre les nœuds $(x_{max}-2, y_{max})$ et $(x_{max}-2, y_{max}-1)$ une solution consiste à utiliser un « crochet » de type 8 à partir du nœud $(x_{max}-3, y_{max})$.

Mais ceci à pour conséquence d'interdire l'utilisation des arcs $(x_{min}+3, y_{min})$ vers $(x_{min}+2, y_{min})$ et $(x_{min}+2, y_{min})$ vers $(x_{min}+2, y_{min}+1)$ dans l'arbre EST, car ces arcs sont utilisés dans l'arbre WST.

Si on applique le même procédé (c'est-à-dire un « crochet » de type 7) aux nœuds $(x_{max}-2, y_{min})$ et $(x_{max}-2, y_{min}+1)$ il va y avoir des répercutions sur les nœuds $(x_{min}+2, y_{max})$ et $(x_{min}+2, y_{max}-1)$, ce qui ne change pas le problème (toujours quatre nœuds isolés et surtout positionnés de la même façon!). D'où la nécessité d'atteindre le nœud $(x_{max}-2, y_{min})$ d'une autre manière. La solution que nous allons décrire maintenant s'applique aussi au nœud $(x_{min}+2, y_{min})$. En fait, il s'agit d'atteindre ces deux nœuds par leur lien S_i, depuis les nœuds $(x_{max}-2, y_{max})$ et $(x_{min}+2, y_{max})$. Cela à pour conséquence de « casser » les arcs $(x_{max}, y_{max}-2)$ vers $(x_{min}, y_{max}-2)$ et $(x_{max}, y_{min}+2)$ vers $(x_{min}, y_{min}+2)$, car ceux-ci seront utilisés dans l'arbre SST par les arcs dernièrement crées. Ces modifications, font apparaître deux nouveaux nœuds isolés que l'on peut facilement rattacher à l'arbre EST. Le nœud $(x_{min}, y_{max}-2)$ peut être atteint par son lien N_i depuis le nœud $(x_{min}, y_{max}-1)$, et le nœud $(x_{min}, y_{min}+2)$ peut quant a lui être atteint par son lien S_i depuis $(x_{min}, y_{min}+1)$. Cette opération est possible, grâce à la toute première modification effectuée sur EST. Après cette deuxième série de modifications, il reste plus que deux nœuds isolés, représentés en noir sur la figure 4.10.

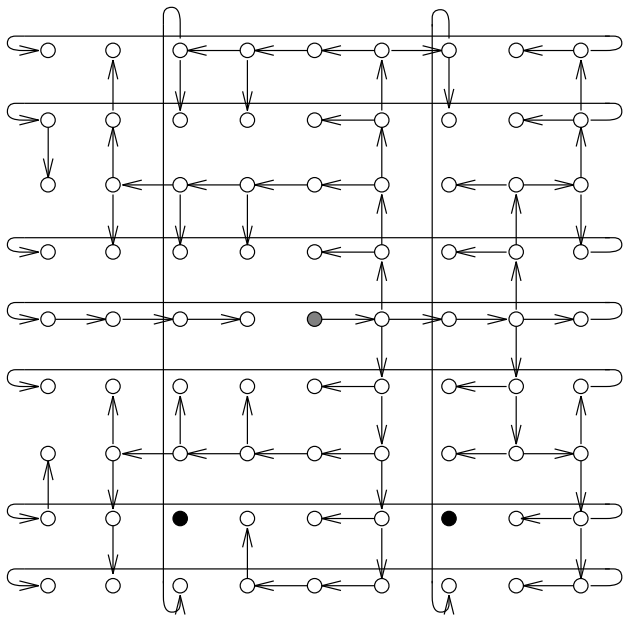


FIG. *4.10 : Arbre EST après la deuxième série de modifications.*

Pour atteindre le nœud $(x_{min}+2, y_{min}+1)$ il semble exister quatre possibilités que nous allons maintenant détailler :

- accès par N_i, c'est-à-dire utiliser l'arc $(x_{min}+2, y_{min}+2)$ vers $(x_{min}+2, y_{min}+1)$. Ceci est impossible car cet arc se transforme en $(x_{min}+2, y_{max}-2)$ vers $(x_{min}+1, y_{max}-2)$ dans l'arbre SST, or EST utilise déjà cet arc.

- accès par E_i, c'est-à-dire utiliser l'arc $(x_{min}+3, y_{min}+1)$ vers $(x_{min}+2, y_{min}+1)$. Ceci est impossible car cet arc se transforme en $(x_{max}-1, y_{min}+3)$ vers $(x_{max}-1, y_{min}+2)$ dans l'arbre NST, or EST utilise déjà cet arc.

- accès par S_i, c'est-à-dire utiliser l'arc $(x_{min}+2, y_{min})$ vers $(x_{min}+2, y_{min}+1)$. Ceci est impossible car cet arc se transforme en $(x_{max}-2, y_{max})$ vers $(x_{max}-2, y_{max}-1)$ dans l'arbre WST, or EST utilise déjà cet arc.

- Il reste donc l'accès par W_i, c'est-à-dire l'arc $(x_{min}+1, y_{min}+1)$ vers $(x_{min}+2, y_{min}+1)$. En effectuant les divers rotations par rapport à l'émetteur $(\frac{\pi}{2}, \pi, \frac{3\pi}{2})$ on constate que cet arc ne se superpose avec aucun autre.

En effectuant le même type de raisonnement on peut monter que la seule possibilité pour atteindre $(x_{max}-2, y_{min}+1)$ est son lien E_i, c'est-à-dire créer l'arc de $(x_{max}-1, y_{min}+1)$ vers $(x_{max}-2, y_{min}+1)$.

La figure 4.11 montre l'arbre de recouvrement ainsi obtenu. Cet arbre est de profondeur minimale égale à $D+1$.

Les trois autres arbres s'obtiennent facilement par rotations successives de $\frac{\pi}{2}$ de l'arbre EST.

La construction de tels arbres est généralisable à toutes les valeurs de $k > 1$. En effet, pour passer d'une grille torique $4k+1$ à une de taille $4(k+1)+1$ il faut ajouter 2 colonnes à droite, 2 colonnes à gauche, 2 lignes en haut, et 2 lignes en bas. Donc lors de la construction de l'arbre de recouvrement EST il suffit de prolonger le principe de base sur les nœuds supplémentaires. Une construction systématique plus rigoureuse de ces arbres est donnée dans les tableaux qui se trouvent en annexe à la fin de ce document. Ces tableaux détaillent les conditions sur i et j pour qu'un nœud (i, j) ait des arcs entrants N_i, E_i, S_i, W_i.

Remarque.
Pour $k = 1$, c'est-à-dire pour les grilles toriques de taille 5×5, il existe quatre arbres de recouvrement de profondeur $D+2$ soit 6. Il ne nous a pas été possible de construire des arbres de profondeur plus faible avec ce mode de construction. Comme le montre la figure 4.12 la grille torique de taille 5×5 possède deux nœuds « intérieurs » (représentés en noir sur la figure) pour l'arbre de recouvrement EST, qui sont à une distance (6 par construction) supérieure à celle du diamètre $(D = 4)$.

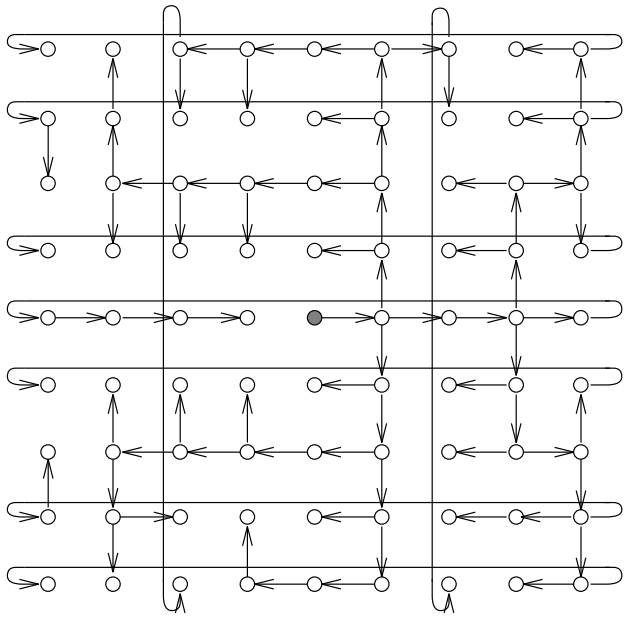


FIG. *4.11 : L'arbre EST de profondeur $D+1$ dans une grille torique 9×9.*

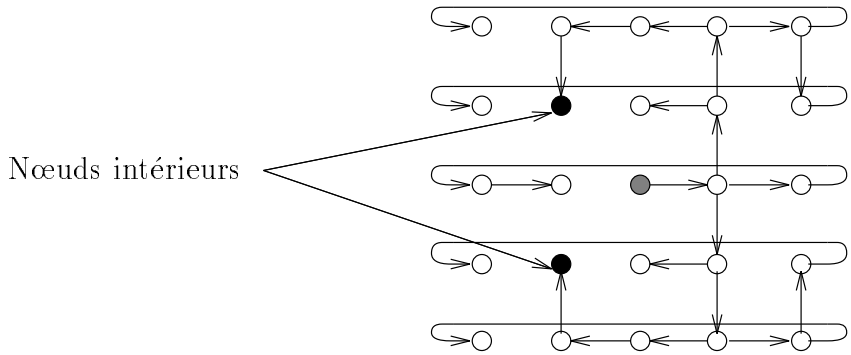


FIG. *4.12 : L'arbre EST de profondeur $D+2$ dans une grille torique 5×5.*

4.4.3 Cas 2 : 4k+3

Comme précédemment, il est facile d'obtenir des arbres de recouvrement de profondeur $D+2$. Du fait de la parité de $\sqrt{p}$, le principe de base peut être appliqué jusqu'à la dernière colonne $i = x_{max}$, pour la construction de l'arbre EST (voir figure 4.13). Les nœuds de la colonne $i = x_{min}$ sont alors atteints depuis les nœuds de la colonne $i = x_{max}$, par les arcs W_i. Pour atteindre les nœuds des lignes $j = y_{min}$ et $j = y_{max}$ il faut utiliser des « crochets » de types 5 et 7.

Les trois autres arbres s'obtiennent facilement par rotations successives de $\frac{\pi}{2}$ de l'arbre EST.

En réalisant des modifications sur cet arbre, on peut construire un arbre de profondeur minimale $D + 1$, comme le montre la figure 4.14. Les modifications consistent à rompre les chemins de longueur $D + 2$. Ces modifications entraînent un jeu de taquin entre les arcs, comme nous l'avons vu dans le cas précédent.

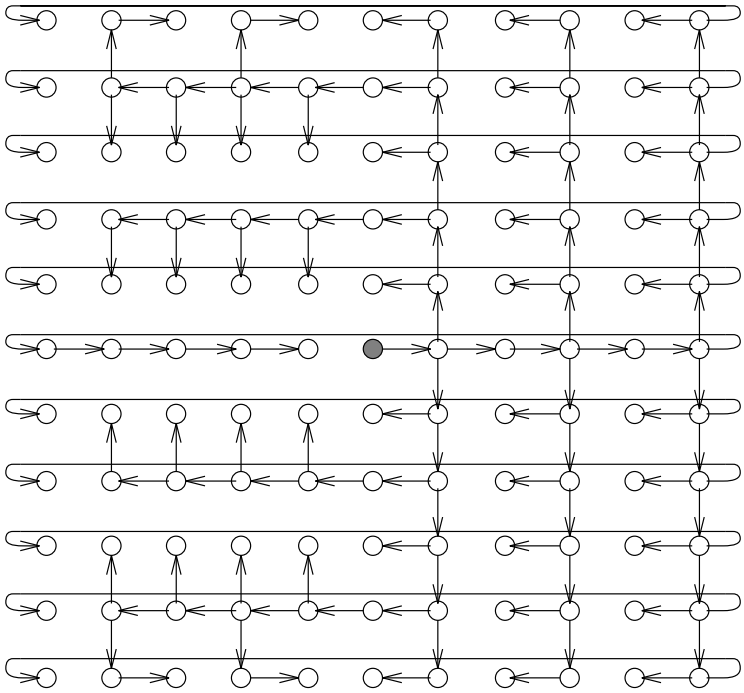


FIG. *4.13 : L'arbre EST de profondeur* $D + 2$ *dans une grille torique* 11×11.

La construction de tels arbres est généralisable à toutes les valeurs de k. En effet, pour passer d'une grille torique $4k + 3$ à une de taille $4(k + 1) + 3$ il faut ajouter 2 colonnes à droite, 2 colonnes à gauche, 2 lignes en haut, et 2 lignes en bas. Donc, lors de la construction de l'arbre de recouvrement EST il suffit de prolonger le principe de base et les modes de construction sur les bords pour les nœuds supplémentaires.

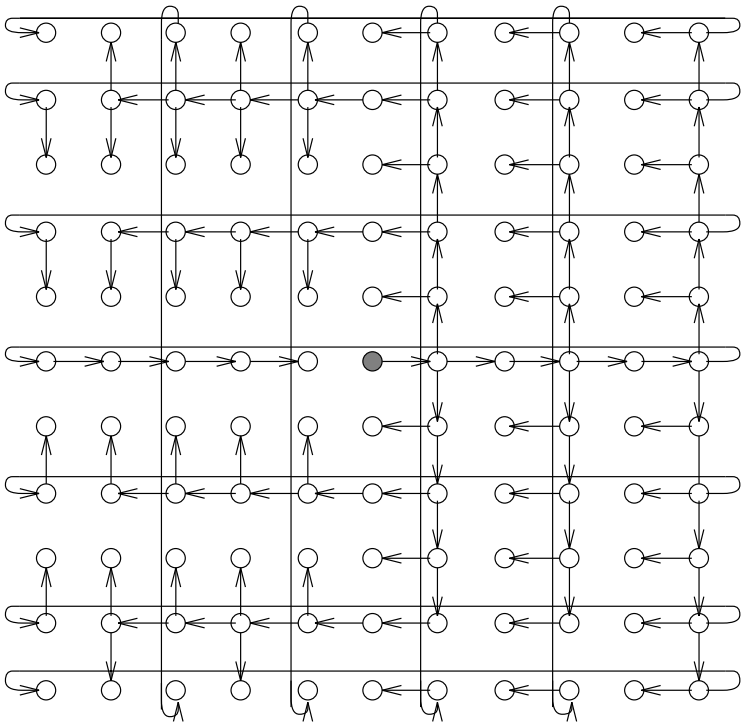


FIG. *4.14 : L'arbre EST de profondeur* $D+1$ *dans une grille torique* 11×11.

Une construction complète de ces arbres est de même donnée en annexe.

4.4.4 Cas 3 : 4k

Du fait que les grilles toriques de taille paire ne possèdent pas de centre, les arbres doivent couvrir quatre zones de tailles différentes. Par conséquent, dans chaque zone le principe de base va être appliqué sur des nombres de lignes et de colonnes différents. Les arbres de recouvrement ne peuvent pas se déduire par simples rotations à partir d'un seul arbre. On s'aperçoit que la construction de deux arbres est suffisante, et que les deux autres se déduisent des premiers par rotations et symétries. La construction des arbres de profondeur $D+1$ EST et NST se fait de la façon suivante :

- pour l'arbre EST on applique le principe de base tant que cela est possible. Puis on utilise des « crochets » de type 5 pour atteindre les nœuds de la ligne $j = y_{max}$ et des « crochets » de types 7 et 8 pour les nœuds de la colonne $i = x_{max}$. Toutefois le nœud (x_{max}, y_{max}) n'est pas atteint. Il peut l'être facilement par son lien S_i depuis le nœud $(x_{max}, y_{max} - 1)$. Afin de réduire la profondeur de l'arbre la même stratégie peut être appliquée aux nœuds (x_{max}, y_{min}) et $(x_{max}, y_{min} + 1)$. En effet, le nœud $(x_{max}, y_{min} + 1)$ est atteint par son lien N_i depuis le nœud $(x_{max}, y_{min} + 2)$, et le nœud (x_{max}, y_{min}) est

atteint par son lien N_i depuis le nœud $(x_{max}, y_{min} + 1)$. Les nœuds de la colonne $i = x_{min}$ sont atteints depuis la colonne $i = x_{max}$ par les arcs W_i.

- pour l'arbre NST le principe de base est appliqué le plus longtemps possible, et l'utilisation de « crochets » de types 3 et 4 permet d'atteindre les nœuds de la colonne $i = x_{min}$. Mais le nœud $(x_{min}, y_{min} + 1)$ ne peut être atteint de cette façon, il peut l'être facilement par son arc S_i depuis le nœud (x_{min}, y_{min}). Du fait de la construction de l'arbre EST, les nœuds $(x_{max}, y_{min} + 1)$ et $(x_{max} - 1, y_{min} + 1)$ ne peuvent pas être atteints par des « crochets ». La solution consiste à atteindre ces deux nœuds par leur arc S_i depuis les nœuds (x_{max}, y_{min}) et $(x_{max} - 1, y_{min})$. Par ailleurs, les nœuds de la ligne $j = y_{min}$ sont atteints depuis la ligne $j = y_{max}$ par les arcs S_i.

La figure 4.15 illustre cette construction.

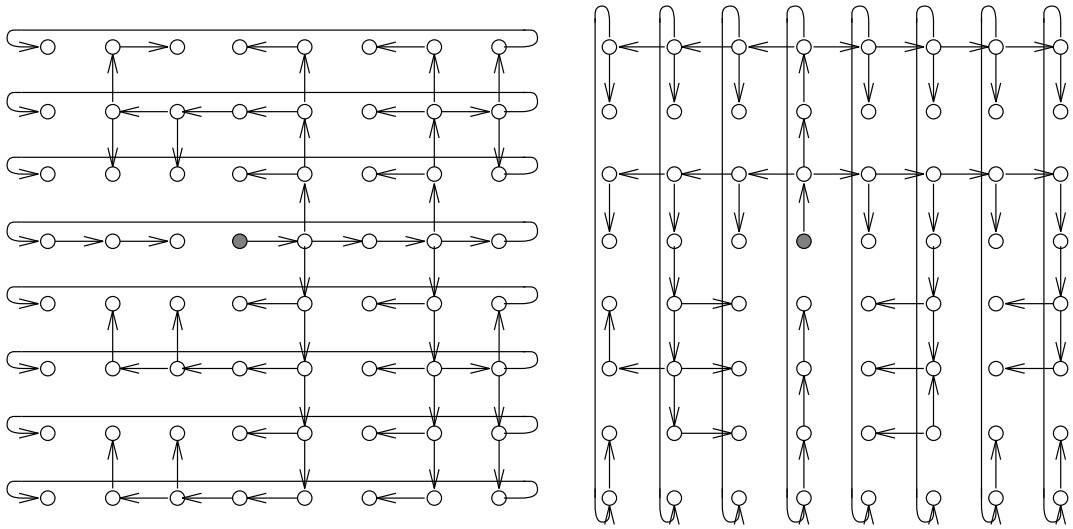


FIG. *4.15 : Les arbres EST et NST de profondeur $D + 1$ dans une grille torique 8×8.*

L'arbre SST se déduit de l'arbre EST, par une rotation de $-\frac{\pi}{2}$ et une symétrie par rapport à l'axe $(i = 0)$. De même l'arbre WST se déduit de l'arbre NST, par une rotation de $-\frac{\pi}{2}$ et une symétrie par rapport à l'axe $(j = 0)$. Comme dans les deux cas précédents, la construction de tels arbres se généralise facilement à toutes les valeurs de k. Une construction complète de ces arbres est donnée dans les tableaux qui se trouvent en annexe.

4.4.5 Cas 4 : 4k+2

Comme nous l'avons fait au paragraphe précédent pour les tailles paires, il est nécessaire de construire deux arbres de recouvrement (NST et EST). La

construction des ces arbres de profondeur $D+1$ EST et NST se fait de la façon suivante :

- pour l'arbre EST le principe de base est appliqué le plus longtemps possible, et l'utilisation de « crochets » de type 4 permet d'atteindre les nœuds de la ligne $j = y_{min}$. Mais le nœud $(x_{min}+1, y_{min})$ ne peut pas être atteint par ces moyens, cependant il peut l'être facilement par son arc S_i depuis le nœud $(x_{min}+1, y_{max})$. Par ailleurs les nœuds de la colonne $i = x_{min}$ sont atteints depuis la colonne $i = x_{max}$ par les arcs W_i.

- pour l'arbre NST l'utilisation du principe de base et des « crochets » de types 3, 4, 6 permettent de couvrir presque toute la grille. Mais le nœud (x_{max}, y_{max}) ne peut pas être atteint. La solution consiste à l'atteindre depuis le nœud $(x_{max}-1, y_{max})$ par son arc W_i. Afin de réduire la profondeur de l'arbre le nœud $(x_{max}, y_{min}+2)$ ne peut être atteint par un « crochet » de type 4, mais depuis le nœud $(x_{min}, y_{min}+2)$ par son arc E_i. Cet arc et certains autres de l'arbre EST obligent le nœud $(x_{min}+1, y_{min})$ à être atteint par son arc W_i depuis le nœud (x_{min}, y_{min}) et le nœud $(x_{max}-2, y_{min})$ à être atteint par son arc E_i depuis le nœud $(x_{max}-1, y_{min})$. Enfin, les autres nœuds de la ligne $j = y_{min}$ sont atteints depuis la ligne $j = y_{max}$ par les arcs S_i.

La figure 4.16 illustre cette construction.

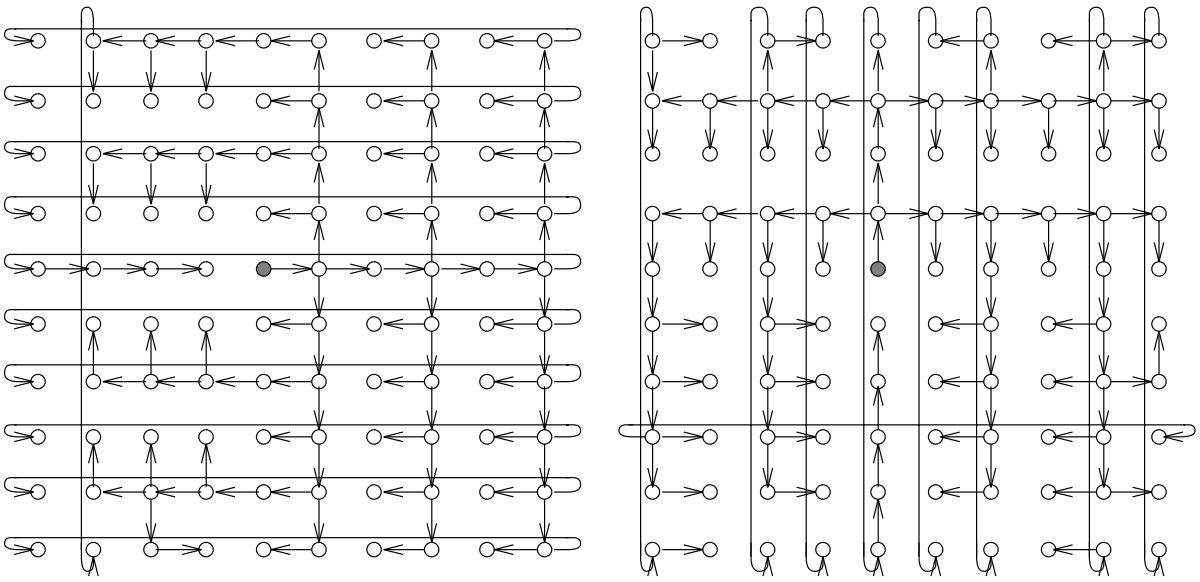


FIG. *4.16 : Les arbres EST et NST de profondeur $D+1$ dans une grille torique* 10×10.

Comme dans le cas précédent l'arbre SST (respectivement WST) se déduit facilement de l'arbre EST (respectivement NST) par une rotation et une sy-

métrie. Comme dans les trois cas précédents la construction de tels arbres se généralise facilement à toutes les valeurs de k. Une construction complète de ces arbres est donnée dans les tableaux qui se trouvent en annexe.

4.4.6 Etude théorique des arbres

Les arbres que nous avons construits l'ont été en deux phases : une application systématique du principe de base et une série d'ajustements sur les bords. Pour démontrer que ces arbres sont arcs disjoints, nous allons montrer que les parties obtenues par le principe de base de chaque arbre sont disjointes, et que les ajustements locaux sont disjoints.

Lemme 4.4.2 *Si un nœud (i, j) est tel que le principe de base lui soit applicable pour chaque arbre construit, alors les arcs entrants de ces quatre arbres sont disjoints.*

Preuve. Rappelons les différents types d'arcs entrants qui se trouvent dans chaque zone, pour l'arbre EST.

zone I	E_i pour les colonnes paires S_i pour les colonnes impaires
zone II	N_i pour les colonnes impaires E_i pour les colonnes paires
zone III	E_i pour les lignes paires S_i pour les lignes impaires
zone IV	N_i pour les lignes impaires E_i pour les lignes paires
zone V	W_i
zone IV	E_i

Il nous faut prouver que les arbres construits avec le principe de base sont disjoints dans les six zones. Pour cela, regardons ce qui se passe dans la zone I.

Pour les tailles de grilles toriques impaires les arbres se déduisent les un des autres par des rotations successives et pour les tailles impaires par des rotations et des syméties. Comme le principe de base est régulier et possède un axe de symétrie suivant lequel on effectue des symétries sur les arbres pour les tailles impaires, il n'y a pas différence pour le principe de base entre les tailles paires et impaire.

- L'arbre NST est obtenu par une rotation de $\frac{\pi}{2}$ de l'arbre EST, c'est-à-dire que les arcs entrants de la zone II vont subir une rotation de $\frac{\pi}{2}$ avant de venir se placer dans la zone I. Après rotation, les arcs N_i deviennent W_i et les arcs E_i deviennent N_i. Les arcs W_i et N_i ne sont pas utilisés par EST dans la zone I, donc EST et NST sont disjoints dans la zone I.

- L'arbre SST est obtenu par une rotation de $\frac{-\pi}{2}$ de l'arbre EST, c'est-à-dire que les arcs entrants de la zone IV vont subir une rotation de $\frac{-\pi}{2}$ avant de venir se placer dans la zone I. Après rotation, les arcs N_i deviennent E_i et les arcs E_i deviennent S_i. Les arcs N_i concernaient les nœuds des lignes impaires, donc par rotation les arcs E_i vont atteindre les nœuds des colonnes impaires. De même les arcs E_i concernaient les nœuds des lignes impaires, donc par rotation les arcs S_i vont atteindre les nœuds des colonnes paires. Comme EST utilisait les arcs E_i pour atteindre les nœuds des colonnes paires et les arcs S_i pour atteindre les nœuds des colonnes impaires, EST et SST sont disjoints dans la zone I. De plus, comme SST et NST n'utilisent pas le même type d'arc (E_i et S_i pour SST et W_i et N_i pour NST), SST et NST sont disjoints dans la zone I.

- L'arbre WST est obtenu par une rotation de π de l'arbre EST, c'est-à-dire que les arcs entrants de la zone III vont subir une rotation de π avant de venir se placer dans la zone I. Après rotation, les arcs E_i deviennent W_i et les arcs S_i deviennent N_i. Comme WST n'utilise pas le même type d'arc que EST et SST (E_i et S_i pour EST et SST), WST est disjoint de EST et de SST dans la zone I.

  Mais il nous reste à vérifier que WST et NST sont disjoints.
  Les arcs E_i concernaient les nœuds des lignes paires, donc par rotation les arcs W_i vont atteindre les nœuds des colonnes paires, de même les arcs S_i concernaient les nœuds des lignes impaires, donc par rotation les arcs N_i vont atteindre les nœuds des colonnes impaires. Comme l'arbre NST utilise les arcs N_i pour atteindre les nœuds des colonnes paires, et les arcs W_i pour atteindre les nœuds des colonnes impaires, les arbres WST et NST sont disjoints dans la zone I.

Donc les arbres construits avec le principe de base sont arcs disjoints dans la zone I. Du fait de la symétrie du problème, on déduit facilement que les arbres construits avec le principe de base dans les zones II, III, et IV sont arcs disjoints.

Dans la zone V l'arbre EST utilise les arcs W_i, l'arbre NST utilise les arcs N_i (rotation de $\frac{\pi}{2}$ des arcs de la zone VI), l'arbre SST utilise les arcs S_i (rotation de $\frac{-\pi}{2}$ des arcs de la zone VI), et l'arbre WST utilise les arcs E_i (rotation de π des arcs de la zone VI), ils sont donc disjoints. En effectuant le même raisonnement sur la zone VI on montre que les arbres construits avec le principe de base sont arcs disjoints dans les zones V et VI.

□

Nous venons de démontrer que la partie des arbres crée avec le principe de base était arcs disjoints. Il nous reste donc à monter que la construction des bords est aussi arcs disjoints.

Nous avons vu lors de la construction des arbres pour la grille torique de taille $\sqrt{p} = 4k + 1$ (avec k=2) que le bord de la grille est disjoint, et de plus comme sur les autres nœuds le principe de base s'applique et d'après le lemme 4.4.2, les quatre arbres de recouvrement sont arcs disjoints. La figure 4.17 permet de confirmer ce fait.

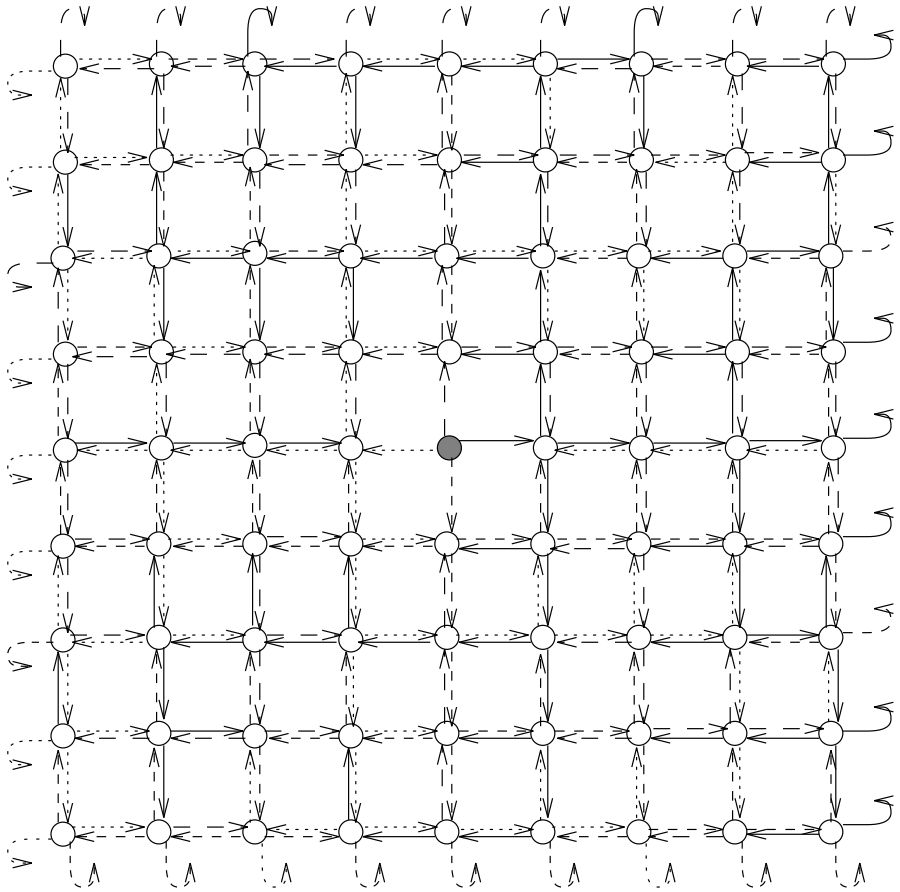


FIG. *4.17 : Les quatre arbres de recouvrement de profondeur $D + 1$ dans une grille torique 9×9.*

En effectuant le même travail sur les bords lors de la construction des autres arbres, pour les différentes tailles de grilles toriques on montre que les arbres présentés sont arcs disjoints pour $k = 2$.

Maintenant il s'agit de montrer que pour tout k supérieur à deux, nous avons une construction arcs disjoints. Pour cela nous allons procéder par récurrence.

Supposons que les arbres soient disjoints pour une grille torique de taille $4k'+1$ (respectivement $4k'+3$, $4k'$,$4k'+2$) avec $k' \geq 2$, montrons que pour $4(k'+1)+1$ (respectivement $4(k'+1)+3$, $4(k'+1)$,$4(k'+1)+2$) les arbres de recouvrements sont toujours disjoints. Pour un type de taille donné (par exemple $4k'+1$) le

passage à la taille supérieure du même type (c'est-à-dire $4(k+1)'+1$) s'effectue en rajoutant deux lignes en haut de la grille torique, deux lignes en bas, deux colonnes à droite et deux colonnes à gauche.

Il est possible de décomposer la grille torique en sept zones (voir figure 4.18) :

- zone 1 : partie centrale de la grille torique où s'applique le principe de base ;
- zone 2 : constituée par les nœuds du bord nord-est ;
- zone 3 : constituée par les nœuds du bord sud-est ;
- zone 4 : constituée par les nœuds du bord sud-ouest ;
- zone 5 : constituée par les nœuds du bord nord-ouest ;
- zone 6 : constituée par les nœuds du bord ayant pour abscisses 0 ;
- zone 7 : constituée par les nœuds du bord ayant pour ordonnées 0 ;

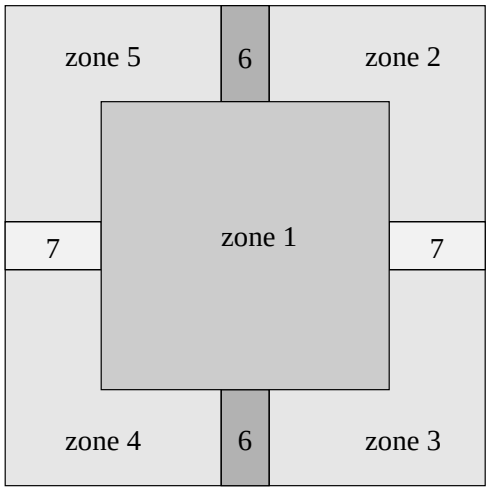


FIG. *4.18 : Localisation des sept zones.*

En enlevant les bords de la grille torique de taille $4k'+1$ (respectivement $4k'+3$, $4k'$, $4k'+2$) on obtient la zone 1, que l'on retrouve dans la grille torique de taille $4(k'+1)+1$ (respectivement $4(k'+1)+3$, $4(k'+1)$, $4(k'+1)+2$). De la même façon on peut réutiliser les six autres zones (voir figure 4.19).

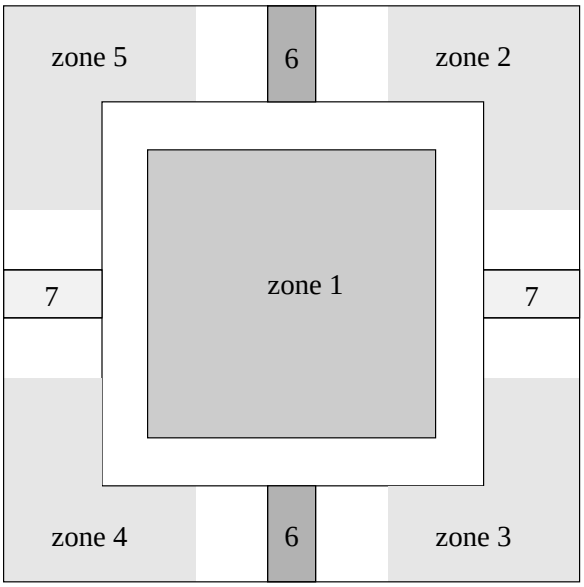


FIG. *4.19 : Localisations des sept zones pour la taille supérieure.*

Ceci nous donne un recouvrement partiel de la grille torique de taille supérieure. Dans la zone comprise entre la zone 1 et les bords, il est possible d'étendre le principe de base, puisque cette zone ne fait pas partie des bords de la grille torique $4(k'+1)+1$ (respectivement $4(k'+1)+3$, $4(k'+1)$, $4(k'+1)+2$). Donc par construction la zone centrale de grille torique de taille supérieure est arcs disjoints. Pour combler les trous qui restent dans le bord nous allons prendre des *patterns* déjà utilisés dans le bord. En fait, dans chaque trou du bord nous avons quatre nœuds que nous allons atteindre par le *pattern* défini par les quatre derniers nœuds de chaque zone. Comme le montre la figure 4.20 chaque zone du bord possède deux *patterns* que nous allons appeler a et b. Pour boucher les trous, il suffit de dupliquer les *patterns* a et b comme le montre la figure 4.20.

Remarque.
Il est très important de noter le passage de la grille torique 9×9 à la grille torique 13×13 ne se fait pas aussi simplement. Du fait de la très grande irrégularité des arbres, et du faible nombre de lignes et de colonnes de cette grille torique, les *patterns* a et b n'ont pas une forme stable. Mais dès la taille 13, nous ne rencontrons plus de problème pour la récurrence.

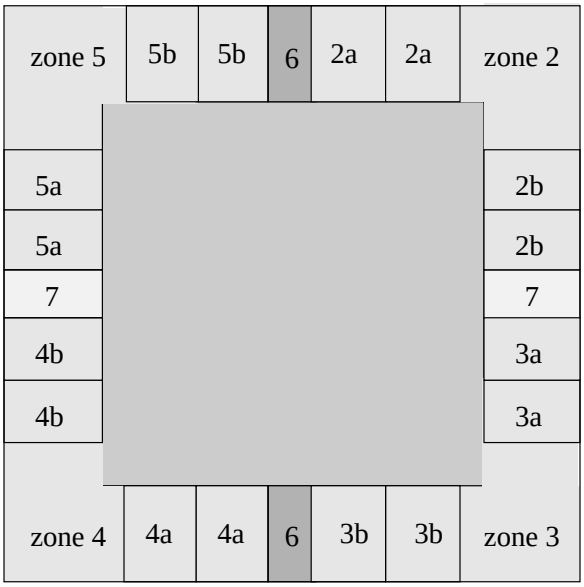


FIG. *4.20 : Construction de la taille supérieure.*

Or, par hypothèse de récurrence les bords sont arcs disjoints, donc les arbres de recouvrements dans la grille torique de taille $4(k'+1)+1$ (respectivement $4(k'+1)+3$, $4(k'+1)$,$4(k'+1)+2$) sont arcs disjoints.

Comme la récurrence est vrai pour $k = 2, .., k', k'+1$ pour les grilles torique du type $4k+3$, $4k$, $4k+2$ et pour $k = 3$ pour les grilles torique du type $4k+1$ elle est vraie pour tout $k \geq 3$.

Ceci nous permet d'énoncer le théorème suivant :

Théorème 4.4.3 *Il existe dans une grille torique* $\sqrt{p} \times \sqrt{p}$ *quatre arbres disjoints de profondeurs minimale* $2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor + 1$.

4.4.7 Algorithme de diffusion

Pour réaliser la diffusion, nous considérons une grille torique de taille $\sqrt{p} \times \sqrt{p}$, dans laquelle les communications sont en mode commutation de messages (*store et forward*) *full-duplex*,Δ-ports. L'utilisation des quatre arbres de recouvrement disjoints va permettre de découper le message à diffuser en quatre parties égales (L_N, L_E, L_S, L_W), et de pipeliner l'envoi de chacun de ces paquets dans l'arbre de recouvrement correspondant. L'algorithme s'écrit de la façon suivante :

Algorithme 4.1 *Diffusion optimale dans les grilles toriques*

si racine alors
  Couper L en quatre partie égales L_N, L_E, L_S, L_W ;
  Pipeliner L_N, L_E, L_S, L_W dans l'arbre de recouvrement de racine R correspondant ;
sinon
  Faire en parallèle
    Recevoir de son père dans chaque arbre ;
    Envoyer à ses fils dans chaque arbre ;

Du fait que cet algorithme utilise des arbres de recouvrement de profondeur minimale, et un nombre maximal de ces arbres, on peut énoncer la proposition suivante :

Proposition 4.4.4 *Il existe une diffusion optimale en mode pipeline sous les hypothèses communication en commutation de message Δ-ports, full-duplex, dans les grilles toriques de taille $\sqrt{p} \times \sqrt{p}$. Le temps de cette diffusion est de :*

$$T_{diffusion} = \left(\sqrt{2\lfloor\frac{\sqrt{p}}{2}\rfloor\beta} + \sqrt{\frac{L}{4}\tau}\right)^2 . \tag{4.2}$$

Preuve. D'après le résultat 2.6 nous savons que le temps nécessaire pour envoyer un message de taille L à une distance d en utilisant Δ chemins disjoints est de :

$$\left(\sqrt{(d-1)\beta} + \sqrt{\frac{L}{\Delta}\tau}\right)^2 \tag{4.3}$$

Les quatre arbres utilisés ont une profondeur de $2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor + 1$, d'où en remplaçant dans la formule ci-dessus, on obtient le temps de la proposition.

Cette diffusion est optimale en mode pipeline, car nous utilisons le maximum d'arbre que permet la grille torique et chaque arbre est de profondeur minimale. □

4.5 Généralisation

Il est facile d'étendre le principe de base à des grilles toriques de taille arbitraire $\sqrt{p} \times \sqrt{q}$, toutefois le problème des nœuds situés sur les bords subsiste.

Pour les tailles telles que $\sqrt{p} = 4k + s$ et $\sqrt{q} = 4k' + s$, il n'y pas de problème sur les bords, les arbres « s'emboîtent » bien. On peut vérifier grâce aux tables qui se trouvent en fin de document, que la construction est toujours à arcs disjoints. Mais pour les autres tailles cela se passe moins bien. En effet, pour les tailles telles que $\sqrt{p} = 4k + s$ et $\sqrt{q} = 4k' + t$, avec $s \neq t$, cela conduit à la construction de plusieurs arbres (en général trois, et le quatrième se déduit d'un des trois autres). L'utilisation des arbres de profondeur minimale n'étant pas possible, la solution consiste à utiliser comme précédemment des arbres de profondeur minimale plus un et de réaliser des adaptations au cas par cas. Une présentation de ces arbres est donnée dans [7]. Le tableau suivant est une synthèse des résultats obtenus.

$\sqrt{p}$	$\sqrt{q}$	profondeur des arbres de recouvrement
4k	4k	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 1$
4k+1	4k+1	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 1$
4k+2	4k+2	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 1$
4k+3	4k+3	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 1$
4k	4k'	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 1$
4k+1	4k'+1	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 1$
4k+2	4k'+2	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 1$
4k+3	4k'+3	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 1$
4k	4k'+1 4k'+2 4k'+3	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 2$
4k+1	4k'+2 4k'+3	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 2$
4k+2	4k'+3	$\lfloor\frac{\sqrt{p}}{2}\rfloor + \lfloor\frac{\sqrt{q}}{2}\rfloor + 2$

Généralisation au *half-duplex*

Une étude similaire a été réalisée en gardant un modèle Δ-port (*link-bounded*), une transmission de messages entre nœuds par commutation de messages (*store and forward*), mais avec des liens de communication *half-duplex*. Cette étude a été réalisée avec Jean-Claude Bermond et Denis Trystram [1].

Après avoir présenté les différentes constructions d'arbres de recouvrement existantes ainsi que les temps de diffusion pipeline dans ces arbres, nous décrirons la construction d'une nouvelle famille d'arbres de profondeurs minimales, ce qui nous permettra de réaliser la diffusion pipeline de façon quasi optimale.

4.5.1 Les solutions existantes

Ici nous allons présenter les différentes constructions proposées par Saad et Schultz [8] et Simmen [9].

Une première solution

La première solution, consiste à construire un seul arbre de recouvrement, comme le montre la figure 4.21

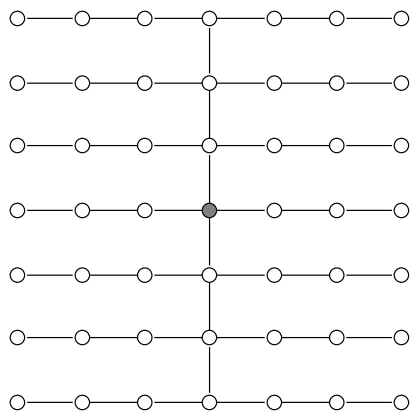


FIG. *4.21 : Un arbre couvrant de la grille torique.*

La profondeur de cet arbre est minimale et est égal au diamètre $2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor$. Cet arbre permet de réaliser un diffusion pipeline en un temps de :

$$T_{diffusion} = (\sqrt{(2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor - 1)\beta} + \sqrt{\tau L})^2. \tag{4.4}$$

L'algorithme de Saad et Schultz

Saad et Schultz ont proposé dans [8] une solution basée sur l'utilisation de deux arbres de recouvrant partiels de profondeur $\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor$ comme le montre la figure 4.22. On constate que ces arbres ressemblent beaucoup à l'arbre donné par la figure 4.21.

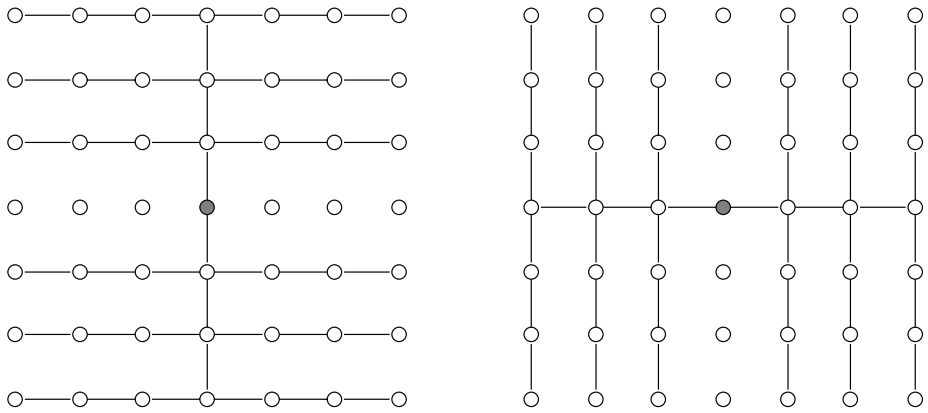


FIG. *4.22 : Deux arbres de recouvrement partiels.*

Le deuxième arbre de recouvrant partiel est obtenu par rotation de $\frac{\pi}{2}$ du premier. On peut remarquer que ces deux arbres sont arêtes disjointes.

Saad et Schultz utilisent ces deux arbres pour réaliser la diffusion pipeline du message, en pipelinant l'envoie de la moitié du message dans chaque arbres. Toutefois, comme chaque arbre ne couvre pas totalement la grille torique, certain nœuds ne connaîtront que la moitié du message. Ces nœuds ont pour abscisse 0 ou pour ordonnée 0. Pour remédier à ce problème Saad et Schultz propose de diffuser de façon pipeline la moitié de message manquante durant la phase de vidage du pipeline. En effet dés que le dernier paquet du message de taille $\frac{L}{2}$ est envoyé, l'émetteur commence à envoyer le message de taille $\frac{L}{2}$ manquant. Malheureusement, ceci ne fonctionne pas aussi simplement. Simmen a montré [9] que pour de grandes tailles de L cette méthode était mauvaise. En effet la première diffusion s'effectue sur une distance de $2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor$ alors que la deuxième s'effectue sur une distance de $2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor$. Du fait du découpage du message en paquets de taille optimal (voir chapitre 2), dans les deux phases consécutive de pipeline la taille des paquets est différente, ce qui influe sur la vitesse de progression des paquets dans le réseau.

En fait pour $L \leq L_0$ (où $L_0 = \frac{\beta}{2\tau}\frac{D^2}{D-1}$) la deuxième phase de pipeline à le temps de finir avant la première. D'où le temps de diffusion est de :

$$T_{diffusion} = \left(\sqrt{(2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor - 1)\beta} + \sqrt{\frac{\tau L}{2}}\right)^2. \tag{4.5}$$

Par contre pour $L \geq L_0$, la deuxième phase n'a pas le temps de finir, ce qui donne un temps d'exécution de :

$$\left(\sqrt{(2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor - 1)\beta} + \sqrt{\frac{\tau L}{2}}\right)^2 + \frac{\tau L}{2} + \sqrt{\frac{\beta\tau L(2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor - 1)}{2}} - \frac{2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor}{2}\left(\beta + \sqrt{\frac{\beta\tau L}{2(2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor - 1)}}\right). \tag{4.6}$$

L'algorithme de Simmen

Simmen [9] propose deux arbres de recouvrement de profondeur $4\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor+1$. La figure 4.23 montre un de ces arbres. Le deuxième arbre s'obtient en effectuant une rotation de $\frac{\pi}{2}$ du premier suivit par une symétrie axiale (opération miroir).

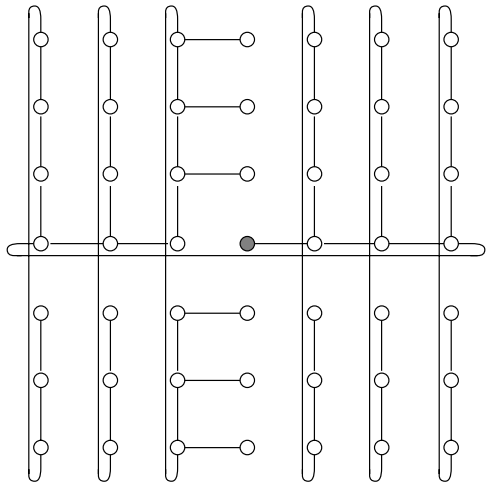


FIG. *4.23 : Un des deux arbres de Simmen.*

Le temps de diffusion pipeline avec ces arbres est donc de :

$$T_{diffusion} = \left(\sqrt{4\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor\beta} + \sqrt{\frac{\tau L}{2}}\right)^2 . \tag{4.7}$$

4.5.2 Quelques améliorations

Nous allons présenter de nouveaux algorithmes issus de modifications effectuées sur la construction des arbres.

Dans le cas du *full-duplex*, nous avons présenté des arbres de profondeur minimale qu'il est possible d'utilisé dans le cas *half-duplex*. En effet, il suffit de décomposer chaque étape de communication en deux sous-étapes. Durant la première on effectue les communications dans un sens, et durant la seconde, on effectue les communications dans l'autre sens. Ceci revient à doubler le diamètre. Avec cet algorithme le temps de diffusion est égale à :

$$T_{diffusion}\left(\sqrt{(2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor+1)\beta} + \sqrt{\frac{\tau L}{4}}\right)^2 . \tag{4.8}$$

Parmi les résultats présentés précédemment, il est possible d'effectuer une amélioration aux arbres de Simmen (voir figure 4.24). Cette amélioration consiste

en une meilleure utilisation des particularités de la grille torique. Les arbres ainsi obtenus ont une profondeur de $\frac{3D}{2}$ soit $3\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor$.

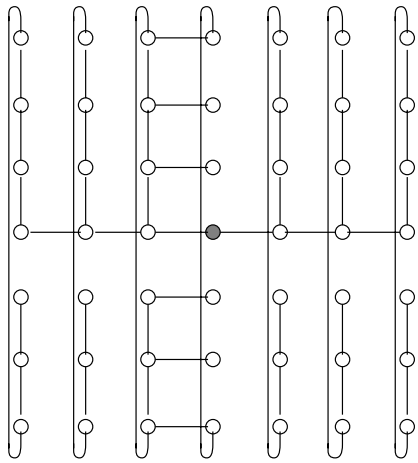


FIG. *4.24 : Un des arbres de profondeur* $\frac{3D}{2}$.

Le deuxième arbre de recouvrement est obtenu par une rotation $\frac{\pi}{2}$ suivie par une symétrie axiale. La diffusion pipeline le long de ces arbres donne un temps de :

$$T_{diffusion} = \left(\sqrt{(3\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor - 1)\beta} + \sqrt{\frac{\tau L}{2}}\right)^2. \tag{4.9}$$

Du fait que nous cherchons à réaliser une diffusion pipeline en temps minimum, il est nécessaire d'avoir le nombre maximum d'arbres c'est-à-dire deux, et d'avoir des arbres de profondeur minimale

4.5.3 Construction d'arbres de profondeur minimale

Afin de construire des arbres de profondeur minimale, il faut connaître celle-ci. Le théorème 4.5.1 donne une borne sur la profondeur minimale de deux arbres disjoints dans une grille torique de taille $\sqrt{p} \times \sqrt{p}$

Théorème 4.5.1 *Dans une grille torique de taille* $\sqrt{p} \times \sqrt{p}$*, il existe deux arbres de recouvrement de profondeur minimale* $\sqrt{p}$.

Preuve. Les arbres ont au moins une profondeur de D, donc ce résultat est optimal dans le cas où $\sqrt{p}$ est pair, puisque $D = \sqrt{p}$. Pour le cas où $\sqrt{p}$ est impair, le diamètre est $\sqrt{p}-1$, mais il est impossible de construire deux arbres de recouvrement arêtes disjointes de profondeur D. Il y a quatre nœuds (x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{min}), (x_{max}, y_{max}) qui sont à distance D de l'émetteur ,de

plus il y a quatre arêtes entre ces quatre nœuds. Ces arêtes ne peuvent pas être utilisées si on désir construire des arbres de recouvrement de profondeur D. Ainsi nous pouvons utiliser au plus $2n^2 - 4$ arêtes pour ces deux arbres. Mais $n^2 - 1$ arêtes sont au minimum nécessaire pour construire un arbre, donc nous avons une contradiction. Donc pour les tailles impaires la profondeur minimale est $D + 1$ soit $\sqrt{p}$. □

Afin de construire les arbres de profondeur minimale, nous allons diviser la grille torique en quatre parties égales (ou quasi égale) (voir figure 4.25). Pour faciliter la construction nous allons imposer que ces arbres soient symétriques par rapport à l'émetteur et de plus que le deuxième se déduise du premier par une simple rotation de $\frac{\pi}{2}$. Le premier arbre contient tous les nœuds dont l'ordonnée est nulle (c'est-à-dire (i,0)) et le second contient donc tous les nœuds dont l'abscisse est nulle (c'est-à-dire (0,j)).

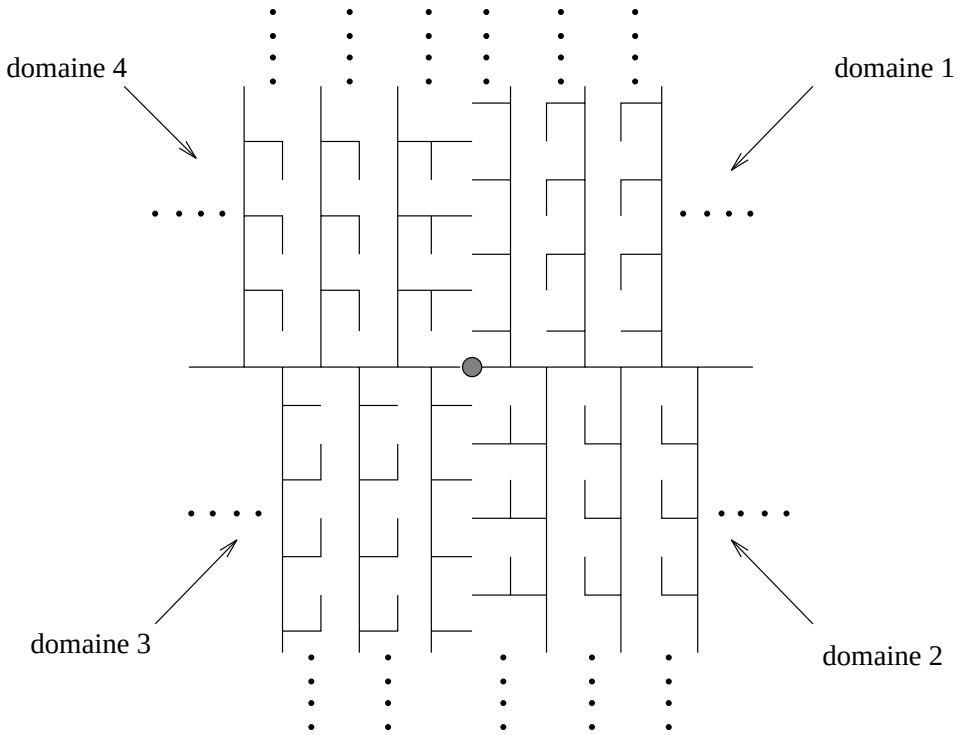


FIG. *4.25 : Idée de construction du premier arbre.*

Pour construire un arbre, l'idée de base consiste à utiliser deux domaines différents pour deux colonnes consécutives. Par exemple, si l'on considère la partie droite de la grille torique par rapport à l'émetteur, l'arbre utilise les colonnes impaires dans le domaine 1 et les colonnes paires dans le domaine 2. (voir figure 4.25) Les nœuds des colonnes non utilisées par l'arbre dans chaque domaine, sont atteints par des « crochets » identiques à ceux définis dans le cas *full-duplex*.

Comme pour le cas *full-duplex* la construction des bords diffèrent, elle dépend de la congruence de $\sqrt{p}$ modulo 4.

Les figures 4.26, 4.27, 4.28, 4.29 donne la construction des arbres pour les taille $\sqrt{p} = 10, 11, 12, 13$.

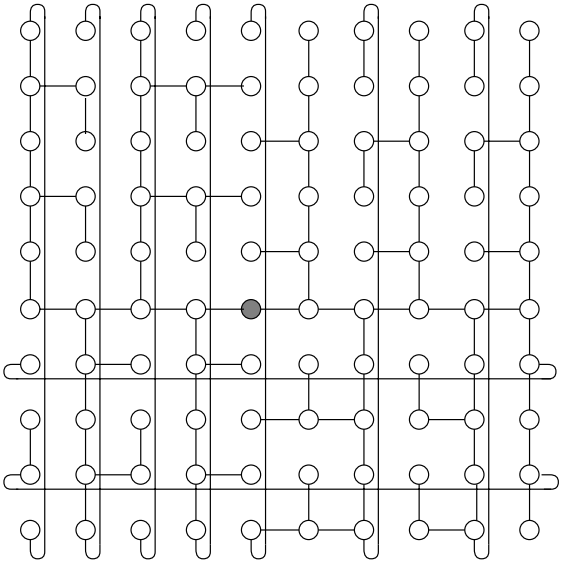


FIG. *4.26 : Construction d'un arbre de recouvrement pour* $\sqrt{p} = 10$.

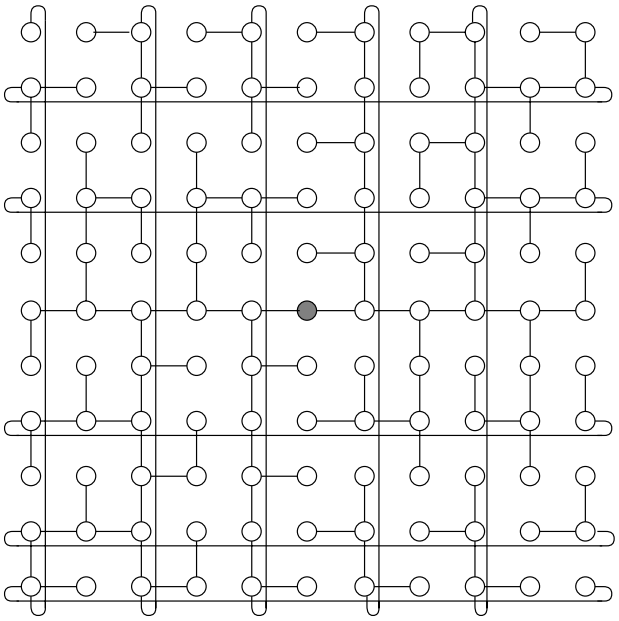


FIG. *4.27 : Construction d'un arbre de recouvrement pour* $\sqrt{p} = 11$.

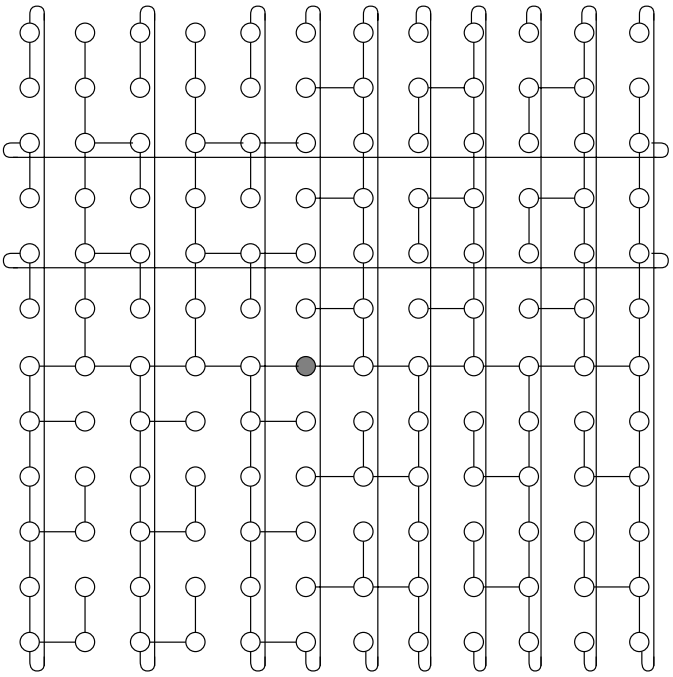


FIG. *4.28 : Construction d'un arbre de recouvrement pour* $\sqrt{p} = 12$.

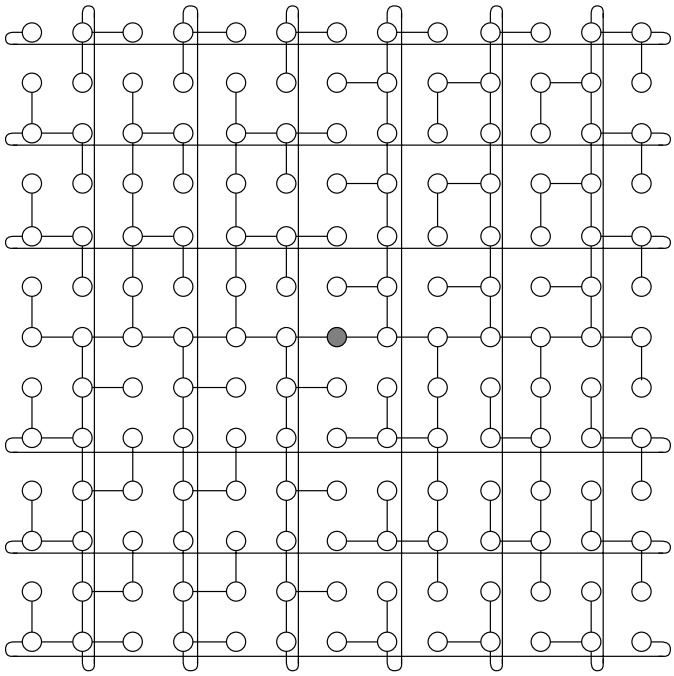


FIG. *4.29 : Construction d'un arbre de recouvrement pour* $\sqrt{p} = 13$.

La construction de ces arbres peut facilement s'étendre, en effet il suffit de rajouter quatre lignes intermédiaires et quatre colonnes intermédiaires (deux dans

chaque domaine).

L'utilisation de ces arbres pour effectué une diffusion pipeline donne le temps :

$$T_{diffusion} = \left(\sqrt{(\sqrt{p}-1)\beta} + \sqrt{\frac{\tau L}{2}}\right)^2 \tag{4.10}$$

L'étude que nous venons de mener nous a permis de présenter une série d'algorithmes de diffusion pipeline dans les grilles toriques en mode *half-duplex*. Cette étude s'est terminée par la présentation d'une nouvelle famille d'arbres de recouvrement arêtes disjointes de profondeur minimale.

4.6 Conclusion

Dans ce chapitre des nouvelles familles d'arbres couvrants de profondeur minimale ont été présentées. Ces familles permettent de réaliser des diffusions dans les grilles toriques carrées en mode pipeline de façon optimale. De plus, une généralisation du mécanisme de construction de ces arbres permet d'obtenir des résultats sur les grilles toriques rectangulaires.

Afin qu'un utilisateur puisse créer sa propre bibliothèque de communication dans les grilles toriques, nous allons présenter dans le chapitre suivant une synthèse des meilleurs algorithmes pour les autres schémas de communications.

Bibliographie

[1] J-C Bermond, P. Michallon, and D. Trystram. Broadcasting in wraparound meshes with parallel monodirectionnal links. *Parallel Computing*, pages 639–648, 1992.

[2] Jean de Rumeur. *Communications dans les réseaux de processeurs.* Masson, 1994.

[3] J. Edmonds. *Edges-disjoint branching, combinatorial algorithms.* Algorithms press, New-York, 1972.

[4] P. Fraigniaud. *Communications intensives dans les architectures à mémoire distribuée et algorithmes parallèles pour la recherche de racines de polynômes.* PhD thesis, Université de Lyon I, ENS, 1990.

[5] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. to appear in Discrete Applied Mathematics, special issue on broadcasting and gossiping.

[6] S.L. Johnsson and C.T. Ho. Optimum broadcasting and personalized communications in hypercubes. *IEEE Transaction on Computers*, 38(9):1249–1268, 1989.

[7] P. Michallon, D. Trystram, and G. Villard. Optimal broadcasting algorithms on torus. Technical Report 872-I-, LMC-IMAG (France), January 1992.

[8] Y. Saad and M.H. Schultz. Data communication in parallel architectures. *Journal of Parallel Computing*, 11:131–150, 1989.

[9] M. Simmen. Comments on broadcast algorithms for 2-dimensional grids. *Parallel Computing*, 17:109–112, 1991.

[10] Q. Stout and B.Wagar. Intensive hypercube communication; prearranged communication in link-bound machines. *Journal of Parallel and Distributed Computing*, 10(2):167–181, 1990.

Chapitre 5

Communications globales dans les grilles toriques

Dans le chapitre précédent, nous avons étudié en détail la diffusion dans les grilles toriques de processeurs, sous l'hypothèse de communication par commutation de message. Au cours de cette étude nous avons décrit et utilisé un outil : les arbres de recouvrement à arcs-disjoints. Dans ce chapitre, nous allons réaliser une étude similaire, sous les mêmes hypothèses (c'est-à-dire commutation de message avec temps linéaire et liens *full-duplex*), qui correspondent au modèle de machine dont nous disposons, par les autres schémas de communications globales. Le but de cette étude est de fournir un outil pour chaque schéma de communication.

Dans un premier temps, nous décrirons les différentes façons de réaliser un échange total. Puis nous présenterons des algorithmes qui permettent de réaliser la distribution, le regroupement et la multidistribution. Enfin, nous conclurons ce chapitre par un tableau récapitulatif des principaux résultats.

5.1 L'échange total

Comme nous l'avons vu au chapitre 3, l'échange total est un schéma de communication où chaque processeur du réseau envoie le même message à tous les autres processeurs. Une borne inférieure de cet algorithme est donnée dans la proposition suivante [1, 2] :

Proposition 5.1.1 *Soit un réseau d'interconnexion de degré Δ, de diamètre D, possédant p processeurs.*

Sous les hypothèses d'une modélisation temps linéaire et de liens parallèles full-duplex, une borne inférieure du temps d'échange total en mode commutation de circuit est :

$$max\left(\beta D, \tau L \left\lceil \frac{p-1}{\Delta} \right\rceil\right) \tag{5.1}$$

Preuve. La distance maximale entre deux processeurs étant D, il faudra donc au moins D étapes pour échanger des messages entre ces deux processeurs.

Chaque processeur doit recevoir $p-1$ messages, donc en supposant que le processeur puisse recevoir sur ses Δ liens simultanément, le temps de propagation minimal sera de $\left\lceil \frac{p-1}{\Delta} \right\rceil$.

□

Comme nous étudions les grilles toriques $TM(\sqrt{p}, \sqrt{p})$, le nombre minimal d'étapes sera de $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor$ et le temps minimal de propagation sera de $\left\lceil \frac{p-1}{4} \right\rceil$.

Pour ce schéma de communication, il est évident que nous ne pouvons pas utiliser des arbres de recouvrement disjoints comme pour la diffusion. Toutefois il est possible, comme nous allons le voir, d'utiliser encore des arbres, mais cette fois ayant des arcs disjoints dans le temps, de façon à éviter les conflits d'accès sur les liens de communication.

Parmi les algorithmes qui réalisent l'échange total, nous allons présenter trois méthodes qui permettent d'atteindre soit le temps de propagation minimal ($D\beta+$ terme), soit le nombre d'étapes minimal ($\tau L \left\lceil \frac{p-1}{\Delta} \right\rceil +$ terme), soit la somme du temps minimum de propagation et du temps de *start-up* minimal ($D\beta+\tau L \left\lceil \frac{p-1}{\Delta} \right\rceil$).

Du fait de la régularité du problème (chaque processeur envoie le même message à tout le monde), on considère qu'à chaque étape tous les processeurs effectuent la même chose. Ceci permet de considérer l'algorithme en un seul processeur, le 0, tout en faisant attention aux problèmes de conflits entre les liens de communication.

5.1.1 Décomposition de la grille torique

Dans le chapitre 2 nous avons vu que la grille torique (*torus mesh*) $TM(\sqrt{p}, \sqrt{p})$ pouvait se décomposer en somme cartésienne de cycles de longueur $\sqrt{p}$, d'où l'idée d'étudier l'échange total sur un anneau (ou cycle) de taille $\sqrt{p}$ afin d'étendre ce résultat à la grille torique.

Préliminaire : échange total sur un anneau

L'évolution des messages reçus par le processeur 0, ainsi que la progression du message émis par ce processeur sont illustrées par la figure 5.1. A l'étape 1 tous les processeurs échangent leur message avec leurs voisins, puis aux étapes suivantes ils échangent les messages qu'ils ont reçus à l'étape précédente et ainsi de suite [1, 3].

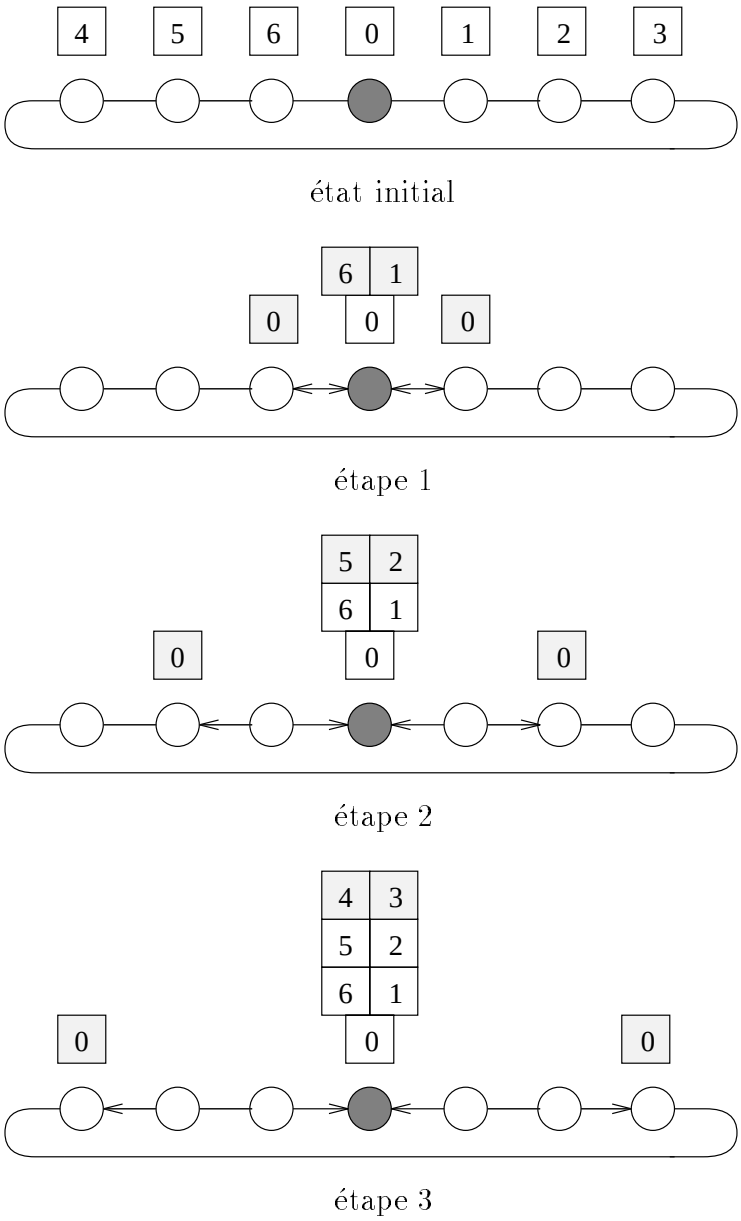


FIG. *5.1 : Echange total sur un anneau de 7 processeurs.*

Soit L_q le message du processeur q et $L_{(q+1 \ modulo \ \sqrt{p})}$ (respectivement $L_{(q-1 \ modulo \ \sqrt{p})}$) le message que le processeur q reçoit de son voisin $(q+1 \ modulo \ \sqrt{p})$ (respectivement $(q-1 \ modulo \ \sqrt{p})$). L'algorithme du processeur q est le suivant :

Algorithme 5.1 *Echange total sur un anneau ($\sqrt{p}$ impair)*

Faire en parallèle
  Envoyer L au processeur ($q+1$ modulo $\sqrt{p}$);
  Envoyer L au processeur ($q-1$ modulo $\sqrt{p}$);
  Recevoir $L_{(q+1 \text{ modulo } \sqrt{p})}$ du processeur ($q+1$ modulo $\sqrt{p}$);
  Recevoir $L_{(q-1 \text{ modulo } \sqrt{p})}$ du processeur ($q-1$ modulo $\sqrt{p}$);
Pour $i \leftarrow 2$ jusqu'à $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor$
  Faire en parallèle
    Envoyer $L_{(q-1 \text{ modulo } \sqrt{p})}$ au processeur ($q+1$ modulo $\sqrt{p}$);
    Envoyer $L_{(q+1 \text{ modulo } \sqrt{p})}$ au processeur ($q-1$ modulo $\sqrt{p}$);
    Recevoir $L_{(q+1 \text{ modulo } \sqrt{p})}$ du processeur ($q+1$ modulo $\sqrt{p}$);
    Recevoir $L_{(q-1 \text{ modulo } \sqrt{p})}$ du processeur ($q-1$ modulo $\sqrt{p}$);

Proposition 5.1.2 *Le temps nécessaire à l'échange total sur un anneau de $\sqrt{p}$ processeurs est de :*

$$T_{Echange\ total} = \left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{\sqrt{p}-1}{2} L\tau \tag{5.2}$$

Preuve. Il est facile de constater que cet algorithme nécessite $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor$ envois et réceptions, ce qui donne un temps d'exécution de $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor (\beta + L\tau)$.

Dans le cas où $\sqrt{p}$ est impair, le résultat est immédiat. Mais dans le cas où $\sqrt{p}$ est pair, il est nécessaire de modifier légèrement l'algorithme. En effet, dans ce cas-là, après $\frac{\sqrt{p}}{2} - 1$ étapes chaque processeur a reçu tous les messages sauf celui du processeur à distance $\frac{\sqrt{p}}{2}$. Mais le message manquant à chaque processeur se trouve sur ses deux voisins, et dans la dernière étape, chaque processeur reçoit de chacun de ses voisins la moitié du message. Le temps d'exécution de l'algorithme est donc : $\left(\frac{\sqrt{p}}{2} - 1\right)(\beta + L\tau) + \beta + \frac{L}{2}\tau$ ce qui correspond au temps énoncé dans la proposition.

□

Application à la grille torique

L'algorithme qui permet de réaliser l'échange total sur la grille torique est une généralisation de l'échange total sur l'anneau. En effet, sur la grille torique l'algorithme peut se décomposer en deux phases successives :

1. sur les anneaux verticaux, réaliser un échange total avec des messages de taille L.

2. sur les anneaux horizontaux, effectuer un échange total avec des messages de taille $\sqrt{p}L$.

Le temps de cet algorithme est :

$$T_{Echange\ total} = 2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{p-1}{2} L\tau \tag{5.3}$$

Toutefois on constate que, lors de chaque phase, seule la moitié des liens est utilisée, ce qui ne permet pas d'obtenir une propagation optimale du message. Il est possible d'effectuer des échanges totaux avec des moitiés de messages sur les anneaux verticaux et horizontaux simultanément. Les deux phases deviennent alors [1, 3, 5] :

1. sur les anneaux verticaux et horizontaux, réaliser un échange total avec des messages de taille $\frac{L}{2}$.
2. sur les anneaux verticaux et horizontaux, effectuer un échange total avec des messages de taille $\sqrt{p}\frac{L}{2}$.

L'algorithme d'un processeur q de la grille torique est le suivant :

Algorithme 5.2 *Echange total sur grille torique (somme cartésienne d'anneaux)*

Faire en parallèle
  Echange total sur les anneaux verticaux de messages de taille $\frac{L}{2}$*;*
  Echange total sur les anneaux horizontaux de messages de taille $\frac{L}{2}$*;*
Faire en parallèle
  Echange total sur les anneaux verticaux de messages de taille $\sqrt{p}\frac{L}{2}$*;*
  Echange total sur les anneaux horizontaux de messages de taille $\sqrt{p}\frac{L}{2}$*;*

Proposition 5.1.3 *Le temps de l'échange total de messages de longueur L dans une grille torique $\sqrt{p} \times \sqrt{p}$ processeurs de l'algorithme précédent est égal à :*

$$T_{Echange\ total} = 2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{p-1}{4} L\tau \tag{5.4}$$

Preuve. Ce temps est la somme des durées de chacune des deux phases :

1. un échange total avec des messages de taille $\frac{L}{2}$ soit :
   $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{\sqrt{p}-1}{2}\frac{L}{2}\tau$
2. un échange total avec des messages de taille $\sqrt{p}\frac{L}{2}$ soit :
   $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{\sqrt{p}-1}{2}\sqrt{p}\frac{L}{2}\tau$

En effectuant la somme on retrouve bien le temps énoncé dans la proposition.

□

Remarque.
L'implémentation de cet algorithme est assez complexe, et nécessite une zone mémoire assez importante pour stocker les messages. Toutefois il permet d'obtenir un temps de propagation et un nombre d'étapes minimaux.

5.1.2 Les chemins hamiltoniens

Comme pour la diffusion, il est possible d'utiliser des arbres de recouvrement orientés pour réaliser l'échange total. Mais cette fois-ci les arbres utilisés sont arcs disjoints dans le temps.

Définition. *α arbres de recouvrement orientés de profondeur $d_1, d_2, ..., d_\alpha$, dans un graphe orienté sont arcs disjoints dans le temps si et seulement si:*
quel que soit h, $1 \leq h \leq max(d_1, d_2, ..., d_\alpha)$, tout arc du graphe appartient au plus au niveau h d'un seul arbre.

Du fait du rôle identique joué par tous les nœuds de la grille, chaque processeur ne peut être la racine que d'un seul arbre. De plus, pour exploiter au mieux le degré de chaque nœud, chaque arbre va être constitué de quatre branches parcourant chacune une zone de $\left\lceil \frac{p-1}{4} \right\rceil$ processeurs. Ces conditions très fortes imposent l'utilisation de chemins hamiltoniens dans chaque zone [6].

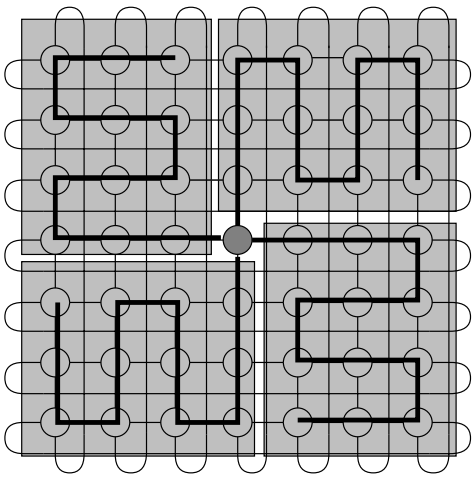


FIG. *5.2 : Parcours de chaque région par un chemin hamiltonien.*

Localement chaque processeur doit gérer les envois qu'il effectue à chaque étape. En fait, chaque processeur doit connaître le parcours de son message dans

un seul chemin hamiltonien, et en effectuant des rotations et/ou des translations il sait ce qu'il doit faire à une étape donnée dans chaque chemin. Afin d'éliminer le coût de cette gestion locale, il faut pouvoir calculer des tables de routage lors de la compilation.

L'algorithme d'un processeur q de la grille torique est le suivant :

Algorithme 5.3 *Echange total sur grille torique (chemins hamiltoniens)*

Faire en parallèle
  Envoyer son message de taille L sur les quatre chemins hamiltoniens;
  Recevoir les messages de taille L de ses quatre voisins;
Pour $i \leftarrow 2$ *jusqu'à* $\left\lceil \frac{p-1}{4} \right\rceil$
  Faire en parallèle
    Envoyer les quatre messages de taille L reçus à
    l'étape précédente sur le bon chemin;
    Recevoir les messages de taille L de ses quatre voisins;

Proposition 5.1.4 *Le temps de l'échange total dans les grilles toriques de l'algorithme utilisant les chemins hamiltoniens est :*

$$T_{Echange\ total} = \left\lceil \frac{p-1}{4} \right\rceil (\beta + L\tau) \tag{5.5}$$

Preuve. Chaque chemin hamiltonien a une longueur de $\left\lceil \frac{p-1}{4} \right\rceil$, et comme à chaque étape, on envoie un message de taille L, le temps de l'algorithme est bien celui de la proposition.

□

Remarque.
Cet algorithme a un *start-up* prohibitif pour de petits messages, mais a un coût quasi identique celui de l'algorithme précédent pour de grands messages. De plus, cet algorithme ne nécessite pas de stockage intermédiaire des messages en mémoire.

Notons que Plateau et Trystram [7] donnent une extension de cette méthode pour les grilles toriques 3-D.

5.1.3 Stratégie gloutonne

Dans cette stratégie, à chaque étape un processeur reçoit et envoie quatre messages. Mais en plus, à une étape donnée i, chaque processeur doit recevoir les messages des processeurs situés à distance i. Il est clair que cette stratégie

nécessite une concaténation des messages [5, 9]. La figure 5.3 illustre les deux premières étapes de cet algorithme pour le processeur dessiné au centre.

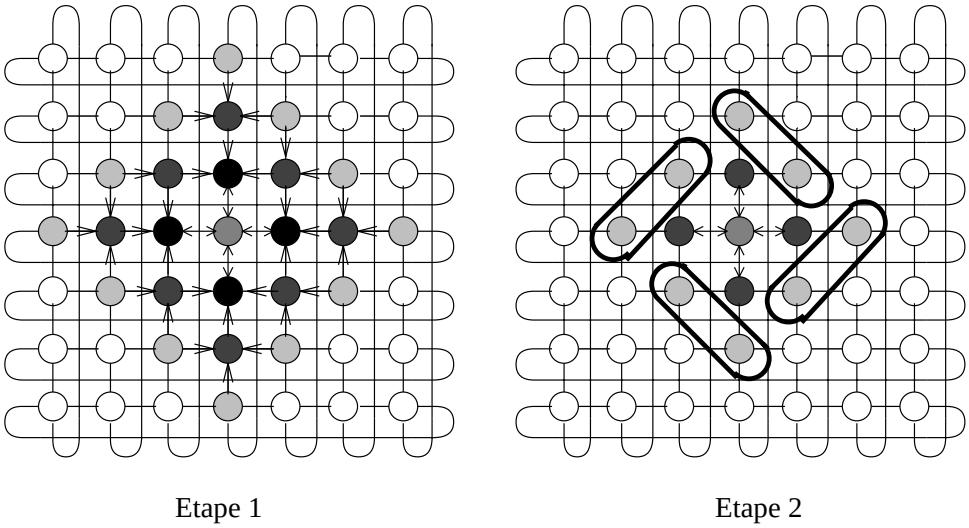


FIG. *5.3 : Les premières étapes de l'algorithme glouton.*

Il est facile de constater que si l'on applique l'algorithme tel quel, il y a redondance des messages. Afin de supprimer cette redondance, chaque message doit suivre un chemin bien précis. Une solution qui permet de recevoir à l'étape i tous les messages des processeurs situés à distance i est de diviser la grille torique en quatre zones de $\left\lceil \frac{p-1}{4} \right\rceil$ processeurs, et de recevoir à chaque étape un message depuis chaque zone. Sur la figure 5.3, on peut voir à l'étape 2 les messages que le processeur central reçoit de chaque zone. Ces messages étaient la possession des processeurs entourés d'un trait noir gras.

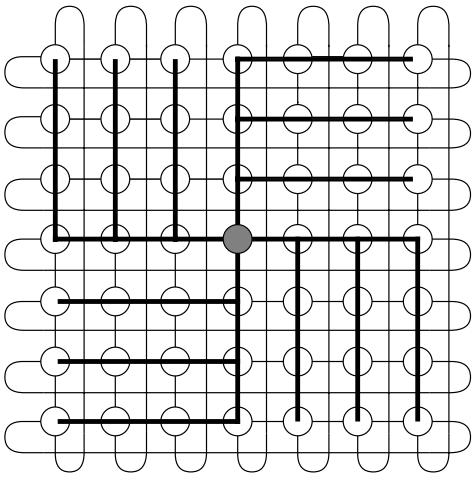


FIG. *5.4 : L'arbre couvrant équilibré.*

En analysant les chemins que doivent suivre les messages dans une zone, on s'aperçoit que ces messages transitent sur un arbre couvrant équilibré de cette zone (voir figure 5.4). Par simple rotation de cet arbre, on couvre les trois autres zones. Il est possible de voir cet algorithme comme un regroupement s'effectuant le long d'arbres de recouvrement équilibrés, enracinés en chaque nœud de la grille.

Remarque.
Notre étude se limitera au cas des grilles toriques de dimension impaire. En effet, pour une grille torique de taille paire, de nombreux problèmes se posent, du fait que chaque processeur ne peut pas être considéré comme le centre de la grille torique.

Algorithme 5.4 *Echange total sur grille torique (arbres équilibrés)*

Faire en parallèle
  Envoyer son message de taille L à ses quatre voisins;
  Recevoir les messages de taille L de ses quatre voisins;
Pour $i \leftarrow 2$ jusqu'à $2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor$
  Faire en parallèle
    Faire la gestion des messages et la concaténation des messages reçus à l'étape précédente;
    Envoyer les quatre nouveaux messages sur les bons arbres;
    Recevoir les messages de ses quatre voisins;

La gestion et la concaténation des messages consiste à faire le tri des messages. Effectivement à chaque étape, un processeur reçoit quatre messages, qu'il va devoir devoir analyser afin d'extraire des morceaux de message de taille L et de reconstituer quatre nouveaux messages que le processeur enverra à l'étape suivante.

Proposition 5.1.5 *Le temps de l'échange total dans les grilles toriques de l'algorithme utilisant les arbres de recouvrement équilibrés est :*

$$T_{Echange\ total} = 2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \left\lceil \frac{p-1}{4} \right\rceil L\tau \tag{5.6}$$

Preuve. La taille des messages varie au cours de l'algorithme : à l'étape 1 un processeur reçoit quatre messages de taille L, à l'étape 2 il reçoit quatre messages de taille $2L$, etc. La taille de chaque message va croître jusqu'à $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor$, puis va décroître jusqu'à L. Donc cet algorithme s'exécute en :

$$2\sum_{i=1}^{\lfloor \sqrt{p}/2 \rfloor} (\beta + iL\tau)$$

soit, en développant, le résultat de la proposition.

□

Remarque.
Cet algorithme permet d'obtenir un temps de propagation et un nombre d'étapes minimaux. Toutefois une zone mémoire importante est nécessaire pour stocker les messages intermédiaires.

Nous venons de présenter trois algorithmes permettant de réaliser l'échange total sur la grille torique dans un temps de propagation minimum. Implémenter l'algorithme glouton et l'algorithme à base de chemins disjoints est assez difficile du fait de la gestion locale des chemins à suivre pour chaque message. Afin de faciliter la tâche du programmeur, il faut pouvoir calculer les tables de routage lors de la compilation afin de pouvoir les utiliser lors de l'exécution. Mais la solution idéale serait d'avoir de telles procédures câblées sur les processeurs de routage afin de supprimer le *start-up* logiciel et ainsi réduire considérablement le coût de ces algorithmes pour de petits messages.

5.2 La distribution

La distribution correspond à l'envoi d'un message différent à tous les processeurs à partir d'un seul nœud. La borne inférieure de cet algorithme est identique à celle de l'échange total. En effet, la distance maximale entre deux processeurs étant D, il faudra au moins D étapes avant que le message émis par la source atteigne le processeur le plus éloigné. Par ailleurs la source ne peut pas émettre plus de Δ messages à la fois, or il faut qu'elle émette $p-1$ messages, donc le temps de propagation minimal sera de $\left\lceil \frac{p-1}{\Delta} \right\rceil L\tau$.

Dans ce paragraphe nous allons considérer le processeur indicé par 0, comme le processeur initiateur de la distribution.

Comme dans le paragraphe précédent, nous allons étudier auparavant la distribution dans l'anneau.

Préliminaire : distribution dans l'anneau

L'algorithme consiste pour le processeur 0 à envoyer à l'étape k les messages destinés aux processeurs à distance $D-k$. A l'étape 1 le processeur 0 envoie au processeur 1 le message destiné au processeur $\left\lfloor \frac{\sqrt{p}-1}{2} \right\rfloor$ et envoie au processeur $\sqrt{p}-1$ le message destiné au processeur $\left\lfloor \frac{\sqrt{p}-1}{2} \right\rfloor + 1$. A l'étape 2 les processeurs 1 et $\sqrt{p}-1$ propagent les messages reçus à l'étape 1 et reçoivent les messages destinés aux processeurs $\left\lceil \frac{\sqrt{p}-1}{2} \right\rceil - 1$ et $\left\lfloor \frac{\sqrt{p}-1}{2} \right\rfloor + 2$, etc. La figure 5.5 illustre ce principe sur un anneau à sept processeurs.

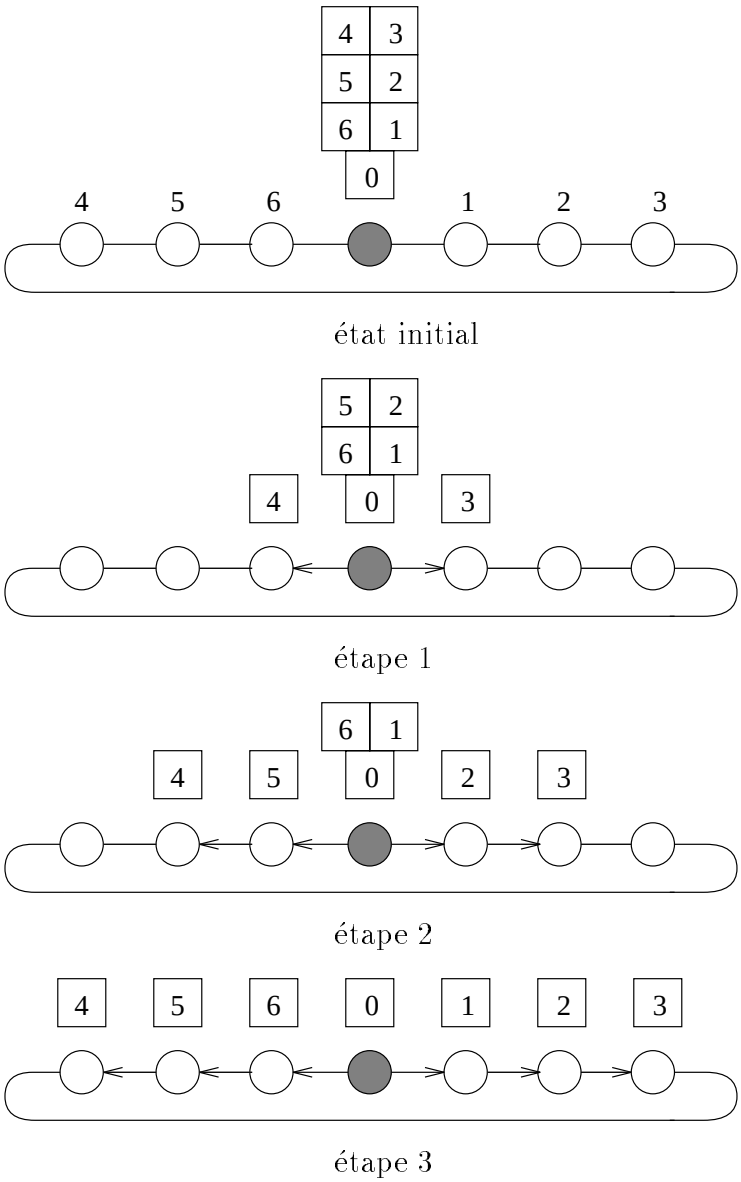


FIG. *5.5 : Distribution dans un anneau.*

Voici l'algorithme d'un processeur q avec $0 < q \leq \left\lfloor \frac{\sqrt{p}}{2} \; : \right\rfloor$

Algorithme 5.5 *Distribution sur un anneau ($\sqrt{p}$ impair)*

Pour $i \leftarrow q+1$ *jusqu'à* $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor$ *faire*
  faire en parallèle
    Recevoir L *du processeur* $q-1$ *modulo* $\sqrt{p}$;
    Envoyer L *au processeur* $q+1$ *modulo* $\sqrt{p}$;
Recevoir L *du processeur* $q-1$ *modulo* $\sqrt{p}$;

Remarque.
Pour les anneaux de taille paire il suffit de modifier la première étape de l'algorithme, en envoyant la moitié du message destiné au processeur $\left\lfloor \frac{\sqrt{p}-1}{2} \right\rfloor$ dans chacune des directions, puis d'exécuter l'algorithme normalement.

Fraigniaud, Miguet et Robert [4] ont montré que cet algorithme était optimal.

Proposition 5.2.1 *Le temps nécessaire pour réaliser une distribution sur l'anneau est :*

$$T_{Distribution} = \left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{\sqrt{p}-1}{2} L\tau \tag{5.7}$$

5.2.1 Application aux grilles toriques

Comme pour l'échange total, il est possible de généraliser ce résultat à la grille torique [8]. En gardant l'idée qui consistait à utiliser les anneaux verticaux et horizontaux simultanément, on trouve un algorithme de distribution, qui se décompose en deux étapes successives :

1. Le processeur 0 réalise une distribution simultanée sur son anneau vertical et son anneau horizontal avec des messages de taille $\sqrt{p}\frac{L}{2}$.

2. Tous les processeurs ayant reçu des messages sur leur anneau vertical (respectivement horizontal) effectuent une distribution sur leur anneau horizontal (respectivement vertical). Comme chacun de ces processeurs a reçu lors de la première étape, un message de taille $\sqrt{p}\frac{L}{2}$ contenant $\sqrt{p}$ messages de taille $\frac{L}{2}$ (destinés aux $\sqrt{p}$ processeurs de l'anneau), la distribution se fait avec des messages de taille $\frac{L}{2}$.

La figure 5.6 illustre ces deux étapes. Les processeurs qui effectuent dans chaque étape des distributions sont représentés en grisé. On notera que le processeur 0 effectue lors de l'étape 2 une nouvelle distribution, en effet lors de la première étape seulement la moitié des messages a été distribuée dans chaque anneau, d'où la distribution de la seconde moitié.

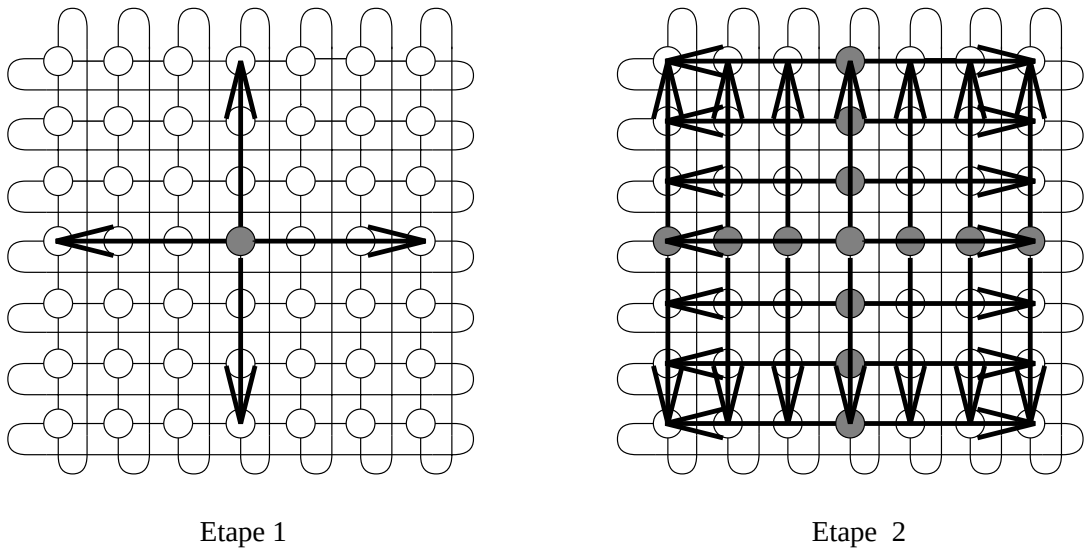


FIG. 5.6 : *Les deux phases d'une distribution dans une grille torique.*

Proposition 5.2.2 *Le temps nécessaire pour réaliser une distribution sur la grille torique est :*

$$T_{Distribution} = 2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{p-1}{4} L\tau \tag{5.8}$$

Preuve. Cette proposition se démontre facilement, car c'est la somme des durées de chaque phase :

1. une distribution avec des messages de taille $\sqrt{p}\frac{L}{2}$ soit :
   $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{\sqrt{p}-1}{2}\sqrt{p}\frac{L}{2}\tau$

2. une distribution avec des messages de taille $\frac{L}{2}$ soit :
   $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{\sqrt{p}-1}{2}\frac{L}{2}\tau$

ce qui donne bien le résultat de la proposition en additionnant les deux temps.

□

5.3 Le regroupement

Le regroupement correspond à l'opération inverse de la distribution, c'est-à-dire que tous les processeurs envoient un message à un processeur donné.

La façon la plus simple de réaliser un regroupement consiste à couper la grille torique en quatre zones et d'effectuer un regroupement dans chaque zone en réalisant une remontée selon un arbre couvrant. Les arbres utilisés ici sont les mêmes que ceux présentés lors de la stratégie gloutonne de l'échange total.

A l'étape 1 la racine reçoit les quatre messages de ses quatre voisins, à l'étape 2 la racine reçoit les messages des processeurs situés à distance 2, etc.

Proposition 5.3.1 *Le temps d'un regroupement sur la grille torique est de :*

$$T_{Regroupement} = 2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \left\lceil \frac{p-1}{4} \right\rceil L\tau \tag{5.9}$$

La preuve de cette proposition est identique à la preuve 5.1.1.

Remarque.
Le temps d'un regroupement est identique au temps d'un échange total.

Généralement le regroupement est utilisé pour effectuer le calcul d'un maximum ou encore le produit scalaire de vecteurs distribués, ce qui nécessite à chaque étape des calculs intermédiaires sur les données contenues dans les messages. Nous allons étudier les différentes façons de réaliser une telle opération sur le produit scalaire distribué. Considérons deux vecteurs de taille n répartis chacun équitablement sur les p processeurs de la grille torique, et soit τ_a le coût d'une opération arithmétique.

La première chose à faire, quelle que soit la stratégie, est de calculer sur chaque processeur le produit scalaire de taille $\frac{n}{p}$, puis d'effectuer le regroupement avec des messages de taille 1.

Une première solution consiste à effectuer un regroupement selon l'algorithme décrit précédemment. Mais on va remplacer la concaténation des messages par une phase de calcul, en effet les processeurs qui reçoivent plusieurs messages vont faire la somme des données contenues dans les messages, avant de renvoyer la nouvelle donnée sous forme de message. Le temps d'une phase correspond au temps du processeur le plus lent. A chaque étape la racine reçoit quatre données, qu'elle additionne avec la sienne, soit au total quatre additions. Dans le reste du réseau, les processeurs les plus lents sont ceux qui reçoivent deux données et qui les additionnent à leur donnée, soit deux additions. Le temps d'une étape est donc égal à $max(\beta + \tau + 2\tau_a, 4\tau_a)$. Il y a exactement $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor - 1$ étapes comme celle-ci.

La dernière phase de l'algorithme correspond à l'envoi par les quatre voisins de la racine des quatre dernières données, et à leur somme sur la racine. Cette dernière phase coûte $\beta + \tau + 4\tau_a$. La figure 5.7 illustre cette solution sur une grille torique de 25 processeurs.

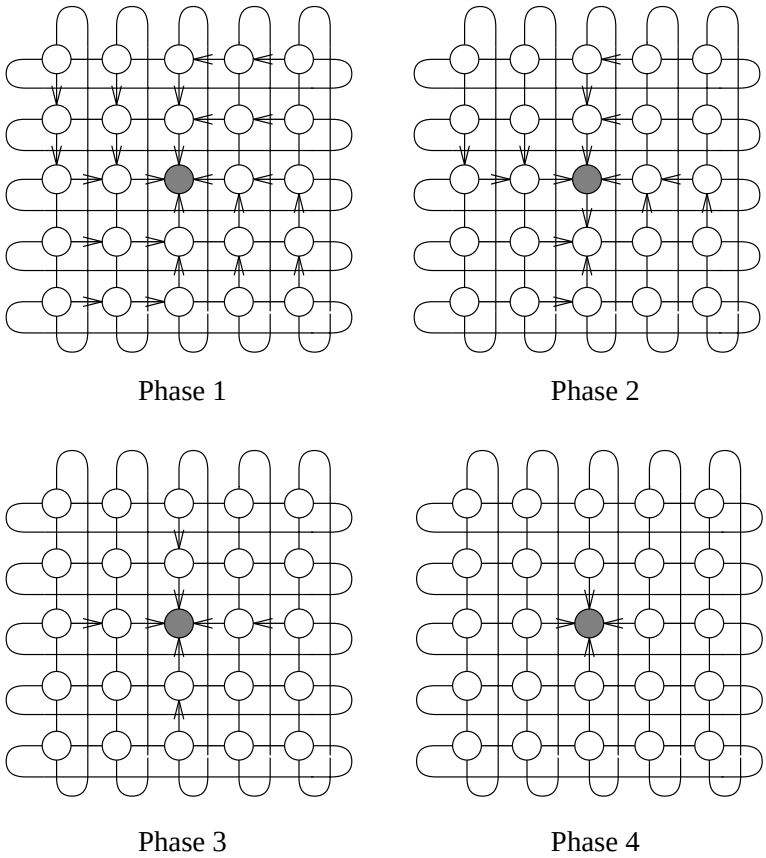


FIG. 5.7 : *Les quatre phases du regroupement.*

Etant données les caractéristiques des machines actuelles, $\beta \gg \tau$ et $\beta \gg \tau_a$, $max(\beta + \tau + 2\tau_a, 4\tau_a)$ est égal à $\beta + \tau + 2\tau_a$.

Cette solution permet de réaliser un regroupement en :
$\left(2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor - 1\right)(\beta + \tau + 2\tau_a) + (\beta + \tau + 4\tau_a)$

soit

$$2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor(\beta + \tau) + \tau_a\left(4\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor + 2\right) \tag{5.10}$$

Une deuxième solution consiste à exécuter l'algorithme de distribution en sens inverse, c'est-à-dire que le regroupement va s'effectuer en deux temps. Dans un premier temps, la racine effectue un regroupement des données situées sur les deux anneaux qui les contiennent, et simultanément les processeurs sur ces deux anneaux effectuent un regroupement sur les autres anneaux (plus précisément sur des « demi-anneaux »). Cela prend un temps de : $\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor max(\beta + \tau + \tau_a, 4\tau_a)$. Dans un deuxième temps, les deux anneaux contenant la racine effectuent un deuxième regroupement, ce qui coûte $\left(\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor - 1\right) max(\beta + \tau + \tau_a, 4\tau_a) + (\beta + \tau + 4\tau_a)$.

La figure 5.8 illustre cette solution où les différents regroupements sont représentés par des flèches en traits pleins et pointillés.

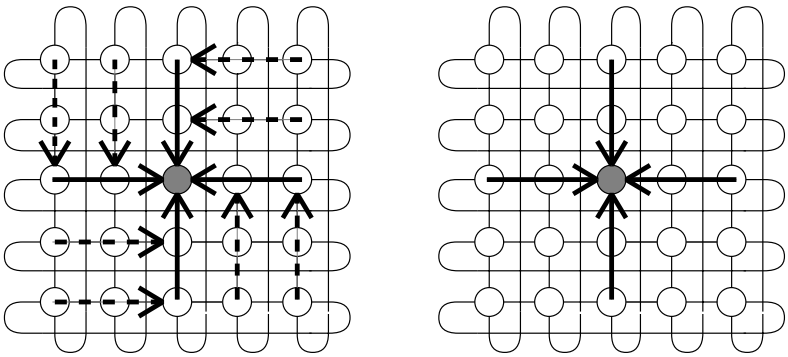


FIG. *5.8 : Le principe du regroupement en utilisant les anneaux.*

Du fait que β est très supérieur aux autres paramètres (τ, τ_a), le $max(\beta + \tau + \tau_a, 4\tau_a)$ est égal à $\beta + \tau + \tau_a$. Donc cette solution permet de réaliser un regroupement en :

$$2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor (\beta + \tau) + \tau_a \left(\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor + 4 \right) \tag{5.11}$$

On constate que cette deuxième solution permet de réduire le coût des calculs. De plus cette solution est plus facile à implémenter, car elle s'exécute en deux phases distinctes très simples à programmer.

En pratique lorsque l'on calcule un produit scalaire distribué, tous les processeurs ont besoin du résultat (voir l'algorithme du gradient conjugué dans le chapitre 3), ce qui oblige après le regroupement à une diffusion de ce résultat. Comme le résultat est de taille 1 il est impossible de pipeliner cette diffusion, donc il faut rajouter au coût du regroupement le coût de la diffusion qui est de : $2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor (\beta + \tau)$ ce qui donne dans le meilleur des cas un temps de regroupement-diffusion égal à :

$$4\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor (\beta + \tau) + \tau_a \left(\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor + 4 \right) \tag{5.12}$$

L'utilisation de deux schémas de communication successifs conduit à un *start-up* prohibitif. Comme nous l'avons remarqué auparavant, le coût d'un regroupement étant le même que celui d'un échange total, on peut donc effectuer un échange total plutôt qu'un schéma de regroupement-diffusion. Comme le message est de taille 1, on ne peut pas couper le message, donc le temps de l'échange total est de : $2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{p-1}{2}\tau$ (voir formule 5.3).

Il est possible de réaliser des calculs lors de l'échange total, mais nous n'envisagerons pas cet aspect, car à l'heure actuelle il n'existe pas de schéma de ce type. Donc, après avoir effectué l'échange total chaque processeur a $(p-1)$ additions à réaliser, ce qui donne un temps de :

$$2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta + \frac{p-1}{2}\tau + \tau_a(p-1) \tag{5.13}$$

Malgré l'augmentation du nombre d'opérations, cette stratégie reste la meilleure, du fait que $\beta \gg \tau_a$. A l'heure actuelle, il existe un facteur 100 entre ces deux paramètres, ce qui implique que pour les tailles de machines existantes on a toujours $(p-1)\tau_a < \left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \beta$.

Si on compare cette dernière formule avec celle donnée par (5.10), on constate que le coût de la transmission des messages est plus élevé et que le *start-up* est identique. Mais avec cette dernière solution, tous les processeurs possèdent le résultat du regroupement, ce qui évite d'effectuer une diffusion par la suite (ce qui est souvent le cas lorsqu'on calcule la norme d'un vecteur distribué ou encore un test d'arrêt). La diffusion va doubler le coût du *start-up*, ce qui va fortement pénaliser le premier algorithme.

5.4 La multidistribution

La multidistribution est une distribution simultanée à partir de tous les processeurs. Comme pour les autres schémas, le résultat de la multidistribution dans les grilles toriques se déduit du résultat dans l'anneau.

Préliminaire : multidistribution dans l'anneau

La multidistribution dans l'anneau correspond à une extension du résultat de la distribution sur l'anneau. A la première étape chaque processeur envoie, dans chaque direction, les messages destinés aux processeurs situés à distance $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor$ [3]. Lors de la deuxième étape chaque processeur doit envoyer ses messages aux processeurs situés à distance $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor - 1$. De plus, chaque processeur doit envoyer à ceux-ci les messages reçus à l'étape précédente, d'où la concaténation des messages, et l'envoi d'un message de taille double. En répétant ce processus, on obtient une multidistribution dont le temps est donné par la proposition suivante :

Proposition 5.4.1 *Le temps de la multidistribution sur l'anneau est de :*

$$T_{Multidistribution} = \begin{cases} (\sqrt{p}-1)\beta + \frac{(p-1)}{8}L\tau & si\ \sqrt{p}\ est\ impair \\ \sqrt{p}\beta + \frac{p}{8}L\tau & si\ \sqrt{p}\ est\ pair \end{cases} \tag{5.14}$$

Preuve. Durant la première étape, un message est échangé entre chaque processeur, pendant la deuxième phase deux messages sont échangés, etc. d'où un temps d'exécution donné par (pour $\sqrt{p}$ impair) :

$$\sum_{i=1}^{\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor} (\beta + iL\tau)$$

Pour les tailles paires, comme pour l'échange total, la première étape consiste à envoyer la moitié du message destiné au processeur le plus éloigné dans chacune des directions. Puis il suffit d'appliquer l'algorithme normalement, en tenant compte qu'à chaque étape nous avons une moitié de message supplémentaire, soit :

$$\sum_{i=1}^{\frac{\sqrt{p}}{2}-1} (\beta + (i + \frac{1}{2}L\tau)$$

En développant on obtient le temps de la proposition. □

L'algorithme d'un processeur q de l'anneau est :

Algorithme 5.6 *Multidistribution dans l'anneau*

Pour $i \leftarrow 1$ *jusqu'à* $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor$
  Faire en parallèle
    Envoyer les messages de taille iL *destinés*
    aux processeurs à distance $\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor - i - 1$;
    Recevoir de chacun de ses voisins un message de taille iL;

Remarque.
Saad et Schultz [8] proposent un algorithme similaire. A l'étape 1 chaque processeur envoie la totalité des messages, puis à l'étape 2 chaque processeur garde le message qui lui est destiné et propage le reste. Au lieu d'avoir une taille de message qui va en augmentant à chaque étape, dans cet algorithme la taille diminue. On constate, que cette stratégie donne le même temps d'exécution que l'algorithme présenté ci-dessus.

5.4.1 Application aux grilles toriques

Là encore, ce schéma de communication se décompose en deux phases successives sur les anneaux verticaux et horizontaux.

La première phase consiste en une multidistribution sur les anneaux verticaux et horizontaux avec des messages de taille $\sqrt{p}\frac{L}{2}$. Pendant cette phase chaque processeur a reçu des messages sur chacun de ses anneaux.

Lors de la deuxième phase chaque processeur va réaliser une nouvelle multidistribution avec des messages de taille $\sqrt{p}\frac{L}{2}$. Les messages reçus lors de la première étape sur les anneaux verticaux vont être distribués sur les anneaux horizontaux et vice-versa.

Le coût total de la multidistribution dans la grille torique correspond à la somme des deux multidistributions dans les anneaux.

Proposition 5.4.2 *Le temps de la multidistribution dans une grille torique est de :*

$$T_{Multidistribution} = \begin{cases} (\sqrt{p}-1)\beta + \frac{(p-1)\sqrt{p}}{8}L\tau & si\ \sqrt{p}\ est\ impair \\ \sqrt{p}\beta + \frac{p\sqrt{p}}{8}L\tau & si\ \sqrt{p}\ est\ pair \end{cases} \quad (5.15)$$

L'algorithme de multidistribution dans les grilles toriques se déduit facilement de celui dans l'anneau.

5.5 Synthèse des résultats

Dans ce paragraphe nous allons faire une synthèse des résultats présentés dans ce chapitre. Afin d'avoir tous les meilleurs résultats concernant les schémas de communications globales dans les grilles toriques sous les hypothèses commutation de messages (modélisé par le temps linéaire), liens *full-duplex* et mode

Δ-port, nous les avons réunis dans le tableau suivant :

Types de communication	Temps de l'algorithme
Diffusion pipeline	$(\sqrt{2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor\beta}+\sqrt{\frac{L}{4}\tau})^2$
Echange total	
Produit cartésien ou glouton	$2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor\beta+\frac{p-1}{4}L\tau$
Distribution	$2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor\beta+\frac{p-1}{4}L\tau$
Regroupement	$2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor\beta+\left\lceil\frac{p-1}{4}\right\rceil L\tau$
Regroupement avec accumulation-diffusion	$2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor\beta+\frac{p-1}{2}\tau+\tau_a(p-1)$
Multidistribution	$\sqrt{p}\beta+\frac{(p-1)\sqrt{p}}{8}L\tau$ si $\sqrt{p}$ est impair $\sqrt{p}\beta+\frac{p\sqrt{p}}{8}L\tau$ si $\sqrt{p}$ est pair

Tableau récapitulatif des temps des schémas de communications globales

5.6 Conclusion

Au cours de ce chapitre, nous avons décrit les schémas de communications globales dans les grilles toriques, tout en mettant en valeur les outils utilisés tels que les arbres de recouvrement disjoints dans le temps, les chemins hamiltoniens, la décomposition en somme cartésienne d'anneaux. Les algorithmes présentés ici sont les meilleurs connus, même s'ils ne sont pas tous optimaux. Ils atteignent tous au moins une des bornes inférieures soit en nombre d'étapes, soit en temps de propagation, soit les deux en même temps.

L'ensemble de ces algorithmes peuvent être plus ou moins facilement implémentés sur une machine parallèle. Toutefois lors de notre étude nous n'avons pas

tenu compte des coûts de gestion mémoire ni des coûts réels des communications en mode Δ-ports. Ce dernier point va être abordé dans le chapitre suivant sur l'exemple de la diffusion. Toutefois, afin de diminuer ces temps, une solution consisterait à calculer les tables de gestion des messages lors de la compilation, ou à implémenter matériellement ces algorithmes.

Bibliographie

[1] M. Cosnard and D. Trystram. *Algorithmes et architectures parallèles.* InterEditions, 1993.

[2] Jean de Rumeur. *Communications dans les réseaux de processeurs.* Collection ERI. Masson, 1994.

[3] P. Fraigniaud. *Communications intensives dans les architectures à mémoire distribuée et algorithmes parallèles pour la recherche de racines de polynômes.* PhD thesis, LIP-ENS-Lyon, December 1990.

[4] P. Fraigniaud, S. Miguet, and Y. Robert. Scattering on a ring of processors. *Parallel Computing*, 13(3):377–383, 1990.

[5] P. Michallon and D. Trystram. Résolution de schémas de communications globales sur réseau de processeurs en grille torique. In *5ème rencontres sur le parallélisme (Brest)*, 1993.

[6] B. Plateau and A. Touzene. Optimal multinode broadcast on a mesh connected graph with reduced bufferization. In A. Bode, editor, *Distributed Memory Computing*, number 487 in Lecture Notes in Computer Science, pages 143–152, 1991.

[7] B. Plateau and D. Trystram. Optimal total exchange for a 3d torus of processors. *Information Processing Letters*, (42):95–102, 1992.

[8] Y. Saad and M. Schultz. Data Communication in Parallel Architectures. *Parallel Computing*, (11):131–150, 1989.

[9] D. Trystram. Total exchange algorithms on wraparound meshes. En préparation, 1993.

Chapitre 6

Expérimentations sur machine à base de Transputers

6.1 Introduction

L'étude théorique des schémas de communications globales sur différentes topologies permet d'obtenir des algorithmes optimaux. Mais lorsque l'on désire utiliser ces algorithmes sur une machine parallèle, divers problèmes se posent :

- d'abord celui de la programmation : en effet tous ces algorithmes utilisent localement une gestion de la mémoire, une gestion des chemins utilisés par les messages, et éventuellement une gestion des messages (concaténation, découpage du messages en plusieurs morceaux, etc.) qui ne sont pas toujours faciles à réaliser ;

- il faut ensuite bien connaître le modèle de communication qui correspond à la machine, afin de choisir l'algorithme approprié. En effet, dans la pratique, un processeur ne peut pas communiquer simultanément sur tous ses liens sans une perte de performance. Or les études et les algorithmes existants ont été réalisés en mode d-ports, ce qui peut avoir des conséquences sur le comportement de certains algorithmes avec des messages de petite taille, où le *start-up* est prépondérant.

Le Laboratoire de Modélisation et de Calcul dispose d'une machine MIMD configurable à 128 Transputers. Après en avoir modélisé les communications, nous avons implémenté et expérimenté divers algorithmes de diffusion sur plusieurs topologies. Les résultats obtenus permettent de comparer et de choisir la meilleure topologie pour réaliser une diffusion sur cette machine en fonction de la taille du message.

6.2 Présentation du MégaNode

6.2.1 Les machines SuperNode

Le projet Esprit P1085 [13], regroupant les Universités de Liverpool, Southampton, l'Institut d'Informatique et de Mathématiques Appliquées de Grenoble, des entreprises privées allemandes, britanniques et françaises avait pour but de développer une architecture multiprocesseurs à hautes performances à faible coût. Ce projet s'est concrétisé par un processeur, le T800, et par la famille des machines SuperNode (T-Node, TandemNode et MégaNode) basées sur ce processeur. Ce sont des machines MIMD reconfigurables (statiquement et dynamiquement). Une machine SuperNode possède entre 16 et 1024 processeurs qui lui permettent d'atteindre des performances de crête variant de 24 à 1500 MFlops.

Ces ordinateurs sont constitués de plusieurs unités modulaires, appelées *nodes*. Le T-Node possède 16 ou 32 processeurs regroupés dans un *node*, le TandemNode possède 64 processeurs et 2 *nodes*. Le MégaNode est la véritable machine multi-*nodes*. Elle comprend un nombre variable de *nodes*, permettant de disposer de 128 à 1024 processeurs. Les *nodes* sont reliés entre eux par un *InterNode Switch*, ce qui la différencie du TandemNode où les 2 *nodes* sont connectés *switch* à *switch*.

La machine dont dispose le Laboratoire de Modélisation et de Calcul est une machine à 128 processeurs, que nous allons décrire plus en détail maintenant.

6.2.2 Le MégaNode

Le MégaNode est une machine mono-utilisateur reliée à un ordinateur hôte (station de travail) qui contient une carte multiprocesseurs à base de Transputers. Cette carte assure toutes les entrées/sorties (y compris le chargement des programmes) avec le MégaNode. C'est aussi sur ces Transputers que sont compilés les programmes destinés au MégaNode.

Le MégaNode possède 128 Transputers répartis en 4 *tandems*. Chaque *tandem* contient 32 Transputers (T800) de travail, des contrôleurs (T414) et des gestionnaires de *switches* (T212). Les processeurs de travail d'un même *tandem* sont reliés entre eux par un *crossbar* RSRE 72×72 (de chez NEC) entièrement reconfigurable grâce au contrôleur (voir figure 6.1). Les communications entre *tandems* sont assurées par l'*InterNode Switch*. Ce composant est constitué de 4 *crossbars* C004 32×32. Ce *crossbar* intègre en sortie un dispositif de recalibrage des signaux. Ce dispositif permet aux messages de parcourir une plus grande distance sur les fils et de traverser un second niveau de commutation. Ceci augmente de façon importante le temps de traversée du C004 [12, 7].

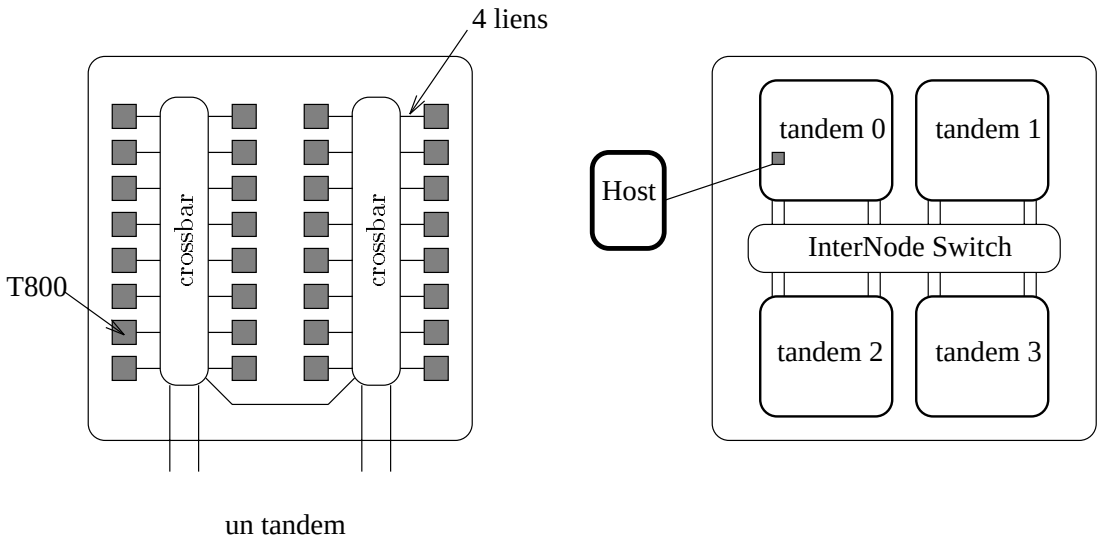


FIG. *6.1 : Architecture du MégaNode 128 processeurs.*

6.2.3 L'environnement logiciel

L'utilisation d'une telle machine n'est pas simple, c'est pourquoi des outils ont été développés au sein du laboratoire [2].

- TéNOR_RITA : les deux niveaux de commutateurs, et les 128 Transputers rendent pénible la configuration du réseau, de plus il faut une bonne connaissance de l'architecture MégaNode pour savoir quel Transputer est connecté à quel autre et par quel lien ! C'est pour résoudre tous ces problèmes que le logiciel TéNOR [1] a été conçu pour le C de Logical System. Mais pour des raisons de facilité d'utilisation nous avons décidé de travailler en C de 3L. Pour cela il a fallu adapter TéNOR, ce qui a donné TéNOR_RITA. Il offre à l'utilisateur la possibilité de décrire à un niveau algorithmique l'ensemble de ses tâches ainsi que le réseau à utiliser.

- OUF : il s'agit d'une bibliothèque de fonctions C, qui permet la programmation du Transputer et qui intègre des primitives de parallélisme comme la notion de canal, la gestion des communications et des processus. Les communications se font selon le principe du rendez-vous, comme en Occam. Il n'existe aucune primitive permettant directement d'effectuer des communications sur plusieurs canaux en parallèle, il faut pour cela créer des processus (suivant le mode *fork and join*) qui effectuent chacun une communication, puis se synchronisent avant d'être détruits par le processus père.

  La modélisation et l'étude qui vont être faites ont été réalisées avec OUF et en langage C de 3L.

6.3 Modélisation des communications

Du fait de l'utilisation de Transputers dans cette machine, les communications se font en commutation de messages (*store and forward*), et la modélisation du temps de communication entre deux voisins est de la forme : $\beta + L\tau$, où β est le *start-up* et τ le temps de propagation. D'après Inmos [10] β est égal à 1μs et τ est égal à 1,1μs/octet quand le langage utilisé est OCCAM.

Après quelques expériences réalisées sur le MégaNode, on s'aperçoit que la modélisation du temps de communication est plus complexe. En effet, ce temps dépend du nombre de liens utilisés, de la façon de les utiliser (*half-duplex* ou *full-duplex*) et de l'emplacement physique des processeurs. Le temps de communication entre deux processeurs se trouvant dans deux *tandems* différents est nettement plus élevé que s'ils sont situés dans le même *tandem*.

Des études similaires ont été faites sur les hypercube *FPS T20* par [6, 8]. Desprez et Tourancheau [4] ont étudié la modélisation des communications en Logical C sur T-Node. Cette modélisation n'est pas applicable sur le MégaNode pour deux raisons. Premièrement, le langage de programmation est différent, ce qui influe sur les performances de communication. En effet les compilateurs sont plus ou moins optimisés. Deuxièmement, les deux niveaux de *switch* du MégaNode, qui n'existent pas sur T-Node, sont ignorés dans cette modélisation.

Afin de prendre en compte le maximum de critères nous proposons [11] le modèle suivant :

$$\beta_{i,j,k} + \tau_{i,j}L \tag{6.1}$$

où

- l'indice i indique l'existence ou non de communication entre différents *tandems* (0 dans le même *tandem*, et 1 dans différents *tandems*);
- l'indice j représente le type de communication (0 pour *half-duplex* et 1 pour *full-duplex*);
- l'indice k est le nombre de processus parallèles qu'il a fallu créer pour la gestion des liens en parallèle.

Coût des communications monodirectionnelles en parallèle

La figure 6.2 montre les différents coûts des communications dans un *tandem* en fonction de la taille du message envoyé. La courbe *seq1* correspond à

l'envoi d'un message sur un seul lien entre deux processeurs voisins, les résultats obtenus donnent $\beta_{0,0,0} = 11,5$ μs et $\tau_{0,0} = 1,125$ μs/octet.

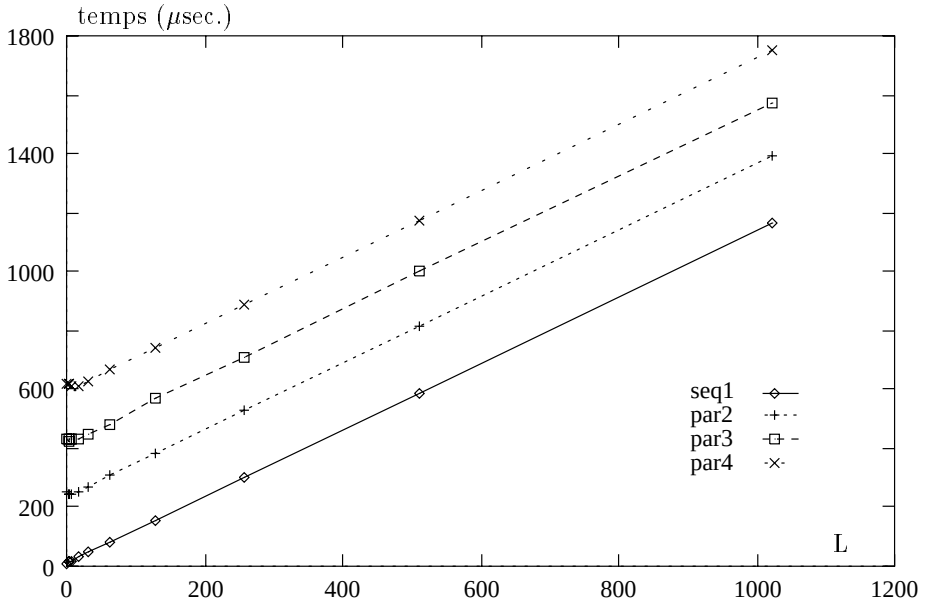


FIG. 6.2 : *Comparaison des communications monodirectionnelles dans un tandem.*

Sur la même figure, les courbes par$n$ correspondent à l'envoi d'un message à n voisins en parallèle. L'expérience révèle que le taux de transmission reste identique (même pente), mais que les *start-ups* sont différents :
$\beta_{0,0,0} = 11,5$ μs.
$\beta_{0,0,k} = 100 + 176k$ μs pour k=1,2,3.

Les différences entre $\beta_{0,0,0}$ et $\beta_{0,0,k}$ sont dues à la gestion des processus parallèles qui effectuent les envois. Les Transputers possèdent 2 *DMA* par lien (un pour les entrées et un pour les sorties) et toutes les *DMA* peuvent fonctionner indépendamment les unes des autres, mais l'initialisation de ces *DMA* est gérée par des processus qui s'exécutent plus ou moins séquentiellement du fait du *scheduler* du Transputer. Le temps d'initialisation est donc d'autant plus grand que le nombre de liens à initialiser est grand. Une fois les communications initialisées les *DMA* gèrent les processus d'envoi de façon indépendante, ce qui explique que le taux de transmission reste le même.

Remarque.
Pour réaliser quatre communications en parallèle, il suffit de créer trois processus, en effet le processus qui a besoin de réaliser ces quatre communications en parallèle peut en gérer une.

Coût des communications bidirectionnelles en parallèle

Lorsque l'on veut réaliser des communications bidirectionnelles il faut créer plus de processus, en effet il en faut un pour la réception et un pour l'émission. La figure 6.3 compare l'envoi et la réception d'un message de même taille en séquentiel et en parallèle. Cette figure montre qu'à partir d'une certaine taille de message (420 octets) il est préférable de faire la réception et l'émission en parallèle. Pour des messages de taille inférieure la gestion des processus devient trop coûteuse. Dans ces cas-là, on peut faire une réception et une émission en séquentiel, c'est-à-dire l'une après l'autre.

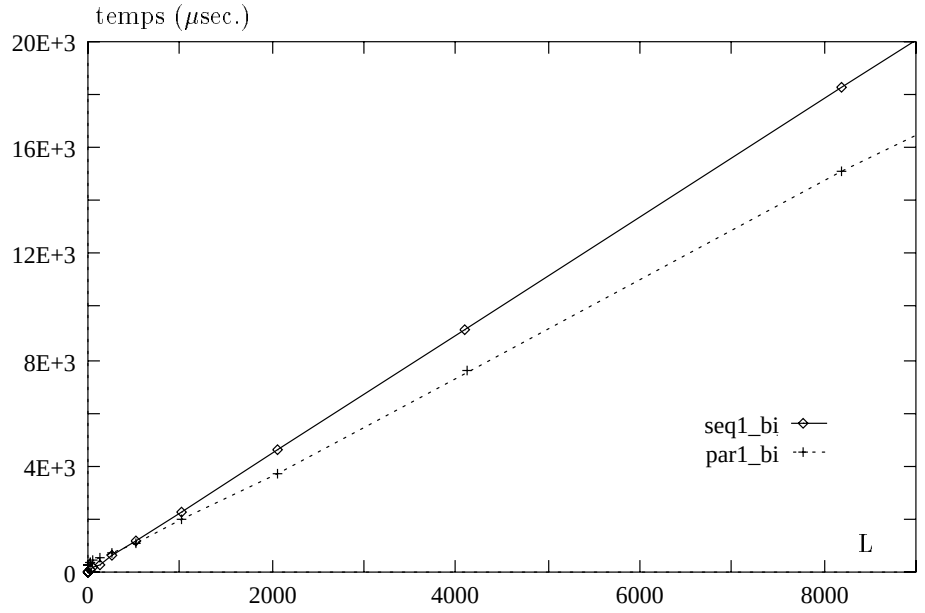


FIG. *6.3 : Communications bidirectionnelles dans un tandem.*

La valeur du taux de transmission devient $\tau_{0,1} = 1,65$ μs/octet. Cette augmentation est due au protocole de communication. L'envoi d'un message entre deux Transputers voisins se fait octet par octet avec accusé de réception pour chaque octet [9] ce qui provoque des ralentissements dans le débit des liens. Dans le cas de communications monodirectionnelles, les accusés de réception se font sur un lien non utilisé, ce qui ne perturbe pas les

transmissions.

Comme pour le cas monodirectionnel, les *start-ups* des communications bidirectionnelles en parallèle sont différents. Pour utiliser k liens simultanément en bidirectionnel, il faut $2k$ opérations simultanées, ce qui nécessite la création et la gestion de $2k-1$ processus. Cela conduit à des *start-ups* de :
$\beta_{0,1,k} = 100 + 176k$ μs pour k=1,3,5 et 7.

Coût des communications entre *tandems*

Du fait du deuxième niveau de *switch* entre les *tandems*, le taux de transmission entre deux *tandems* est plus faible. La figure 6.4 montre la perte de performance des communications inter-*tandems*. Comme le *start-up* ne dépend pas du *switch*, il reste inchangé.

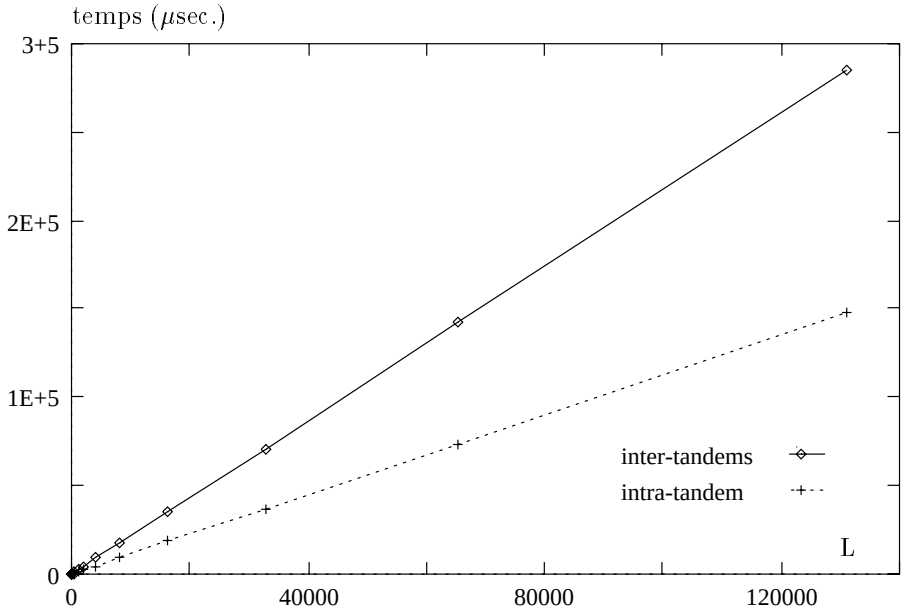


FIG. *6.4 : Comparaison des communications inter et intra tandem.*

Pour les communications monodirectionnelles inter-*tandem* on obtient les résultats suivants :
$\beta_{1,0,0} = 11,5$ μs.
$\tau_{1,0} = 2,17$ μs/octet.

Comme pour le cas des communications à l'intérieur d'un *tandem*, le fait d'effectuer plusieurs communications en parallèle entre des processeurs se

trouvant dans deux *tandems* différents influe uniquement sur le *start-up* comme le montre la figure 6.5.

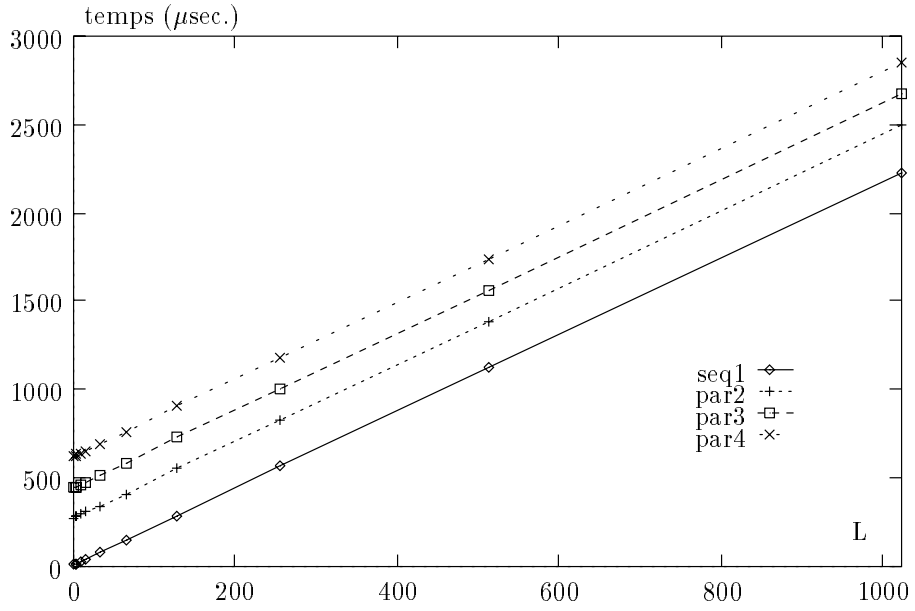


FIG. *6.5 : Communications parallèles monodirectionnelles inter tandems.*

Ces *start-ups* sont identiques à ceux du cas intra-*tandem* :
$\beta_{1,0,k} = 100 + 176k$ μs pour k=1,2,3.

La figure suivante (figure 6.6) montre qu'il existe peu de différences entre le taux de transmission entre deux *tandems* en monodirectionnel et en bidirectionnel, on trouve $\tau_{1,1} = 2,2$ μs/octet. En effet, lors de la communication monodirectionnelle entre deux *tandems* le taux de transmission a diminué, c'est-à-dire qu'il y a un peu plus de temps entre deux octets, ce qui permet pour des communications bidirectionnelles de faire circuler les accusés de réception sans trop gêner les envois de messages.

Comme pour les communications bidirectionnelles intra-*tandem*, il est nécessaire de générer $2k - 1$ processus pour réaliser simultanément des communications sur k liens, d'où les *start-ups* suivants :
$\beta_{1,1,k} = 100 + 176k$ μs pour k=1,3,5 and 7.

Nous allons faire un tour d'horizon des algorithmes de diffusion sur les diverses topologies de degré quatre que l'on peut réaliser sur le MégaNode.

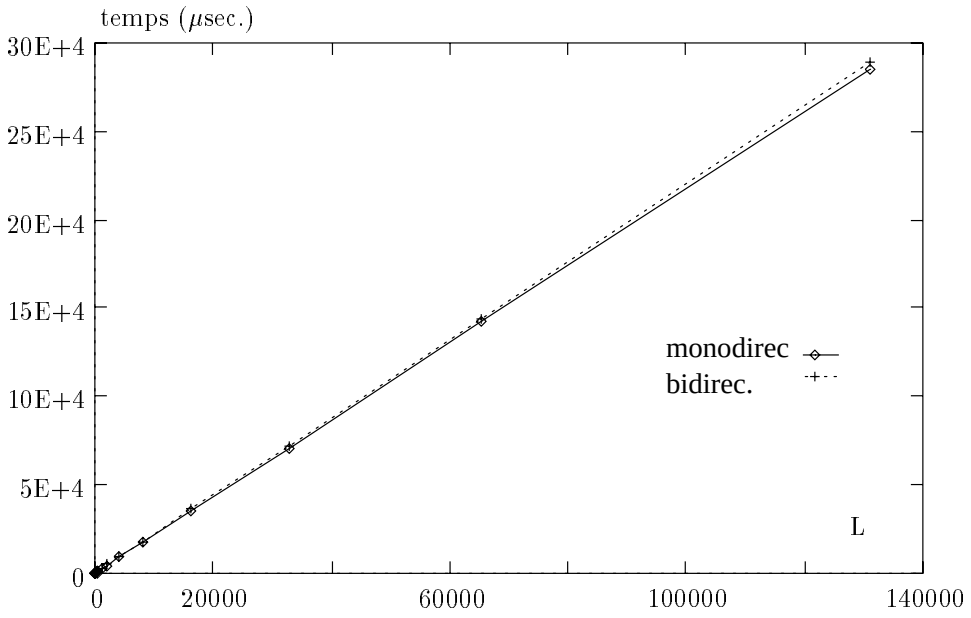


FIG. 6.6 : *Comparaison entre les communications mono et bidirectionnelles inter tandems.*

6.4 Description des algorithmes de diffusion

L'étude des communications sur le MégaNode a permis de montrer que le modèle temps linéaire n'était pas valide sur cette machine. En effet les paramètres d'une communication (*start-up*, taux de transmission) changent avec les caractéristiques du réseau et des communications. Par exemple, si les chemins sont trop longs on va devoir changer de *tandem*, et si l'on utilise au maximum le degré d'un nœud, on va se retrouver avec un *start-up* prohibitif.

Dans les chapitres 2 et 4, on a vu que le temps minimal pour envoyer un message de taille L à deux nœuds à distance H, avec la technique du pipeline, en utilisant Δ arbres disjoints, est de :

$$b(\frac{L}{\Delta}, H) = (\sqrt{(H-1)\beta} + \sqrt{\tau \frac{L}{\Delta}})^2 \tag{6.2}$$

d'où en utilisant notre modèle :

$$b(\frac{L}{\Delta}, H) = (\sqrt{(H-1)\beta_{i,j,k}} + \sqrt{\tau_{i,j} \frac{L}{\Delta}})^2 \tag{6.3}$$

En regardant de plus près l'équation 7.1, on constate que si le paramètre $\frac{1}{\Delta}$ décroît, les paramètres $\beta_{i,j,k}$ et $\tau_{i,j}$ augmentent ! Le MégaNode étant re-

configurable, on peut tester différents algorithmes de diffusion sur diverses topologies, afin de proposer un algorithme qui réalise le meilleur compromis possible.

Remarque.
Du fait que l'on utilise la technique du pipeline, c'est le processeur qui communique le plus lentement qui va donner la cadence. En effet, si deux processeurs P_1 et P_2 communiquent ensemble sur un lien, et si P_1 communique simultanément sur l_1 liens (nécessitant la création de k_1 processus), et si P_2 communique aussi sur l_2 liens (nécessitant la création de k_2 processus), alors le temps de transfert d'un message de taille L entre ces deux processeurs va être : $\beta_{i,j,k} + L\tau_{i,j}$ où $k = max(k_1, k_2)$.

Cas des arbres ternaires et binaires

Un arbre ternaire permet de construire le réseau de degré quatre ayant la plus faible profondeur. Comme nous ne pouvons utiliser qu'un seul arbre dans cette topologie, Δ est égal à 1. La racine découpe le message initial en paquets de taille optimale, et les envoie en parallèle à ses trois fils. Les autres processeurs du réseau reçoivent un paquet et envoient en parallèle à leurs trois fils le paquet reçu à l'étape précédente. Sur cette topologie les liens sont donc utilisés en monodirectionnel, et la création de trois processus est nécessaire pour gérer les communications en parallèle. Le temps de la diffusion dans un arbre à p processeurs est :

$$b(L, \lceil log_3(2p+1) - 1\rceil) = (\sqrt{(\lceil log_3(2p+1) - 1\rceil - 1)\beta_{x,0,3}} + \sqrt{\tau_{x,0}L})^2 \quad (6.4)$$

x varie suivant le nombre de nœuds considérés dans le réseau. Si le nombre de processeurs utilisés est supérieur à 32 alors $x = 1$ sinon $x = 0$. Dans la suite de ce chapitre, nous garderons la même convention.

Afin de gérer moins de processus parallèles, il est possible de réaliser la diffusion sur un arbre binaire. Le temps de la diffusion devient alors :

$$b(L, \lceil log_2(p+1) - 1\rceil) = (\sqrt{(\lceil log_2(p+1) - 1\rceil - 1)\beta_{x,0,2}} + \sqrt{\tau_{x,0}L})^2 \quad (6.5)$$

Cas des grilles toriques

La diffusion dans les grilles toriques a été étudiée au chapitre 4 où nous avons décrit des algorithmes pour le cas monodirectionnel et pour le cas

bidirectionnel. Il est intéressant de comparer ces deux modes de diffusion. En effet la diffusion monodirectionnelle utilise moins le parallélisme du réseau et nécessite moins de processus pour les communications, par contre l'algorithme bidirectionnel dispose d'une meilleure bande passante.

Le coût de la diffusion bidirectionnelle est de :

$$b\left(\frac{L}{4}, 2\lfloor\frac{\sqrt{p}}{2}\rfloor + 1\right) = \left(\sqrt{2\lfloor\frac{\sqrt{p}}{2}\rfloor\beta_{x,1,7}} + \sqrt{\tau_{x,1}\frac{L}{4}}\right)^2 \tag{6.6}$$

Le coût de la diffusion monodirectionnelle est de :

$$b\left(\frac{L}{2}, \sqrt{p}\right) = \left(\sqrt{(\sqrt{p}-1)\beta_{x,1,3}} + \sqrt{\tau_{x,0}\frac{L}{2}}\right)^2 \tag{6.7}$$

Remarque.
Il est possible de réaliser une grille torique sur le MégaNode, mais du fait que tous les nœuds ont un degré quatre, plus aucun lien ne reste libre pour relier la grille torique avec le frontal ! Afin de remédier à ce problème, il a fallu ajouter un nœud supplémentaire à la grille torique. Le choix de l'adjonction de ce nœud supplémentaire est fait de façon à ne pas augmenter la profondeur des arbres.

Cas du De Bruijn

Bermond et Fraigniaud [3] ont montré pour les graphes de De Bruijn $(2, log_2(p))$, avec des liens bidirectionnels, qu'il existe deux arbres de recouvrement disjoints. Si la racine est (000...0) ou (111...1) la profondeur de ces arbres est $log_2(p) + 1$ sinon la profondeur est $2log_2(p) + 1$ pour n'importe quelle autre racine.

Le temps de la diffusion dans ce réseau est de :

$$b\left(\frac{L}{2}, 2log_2(p) + 1\right) = \left(\sqrt{2log_2(p)\beta_{x,1,5}} + \sqrt{\tau_{x,1}\frac{L}{2}}\right)^2 \tag{6.8}$$

De ces divers temps théoriques, on peut prévoir qu'asymptotiquement l'algorithme de diffusion bidirectionnelle sur la grille torique sera le meilleur, du fait de l'exploitation maximale du degré de chaque nœud. Mais pour ce qui est des messages de petite taille ou encore de taille moyenne l'étude théorique ne permet pas de conclure.

6.5 Expérimentations

Dans cette section, nous présentons les résultats obtenus avec les différents algorithmes de diffusion sur le MégaNode. De ces résultats, nous allons pouvoir déduire la topologie et l'algorithme le mieux adapté au MégaNode pour réaliser une diffusion, en fonction de la taille du message à diffuser.

Mais avant cela, vérifions sur la diffusion bidirectionnelle dans les grilles toriques si la modélisation que nous avons faite concorde avec les résultats expérimentaux que nous obtenons. La figure 6.7 montre les résultats obtenus pour différentes tailles de grilles.

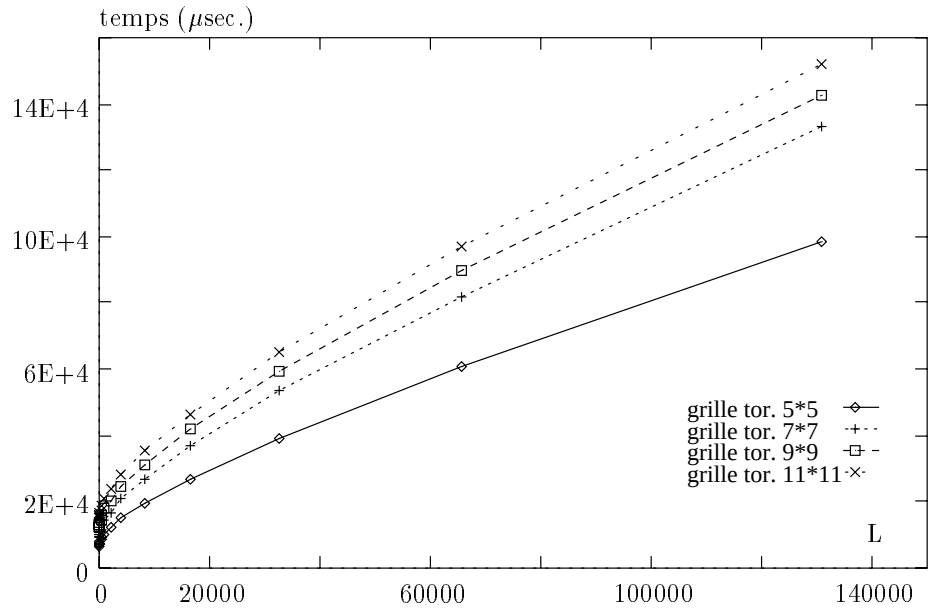


FIG. 6.7 : *Diffusion bidirectionnelle dans les grilles toriques.*

Remarque.
La bonne performance relative pour la grille torique 5×5 est due au fait que cette grille est contenue dans un seul *tandem*.

La figure 6.8 compare pour une grille torique 9×9 la courbe théorique et la courbe expérimentale. On constate que les résultats obtenus sont très proches des valeurs théoriques. Les petites différences sont dues au coût des gestions mémoires et surtout aux problèmes de synchronisation. En effet il est impossible de synchroniser l'ensemble des processeurs sur le MégaNode.

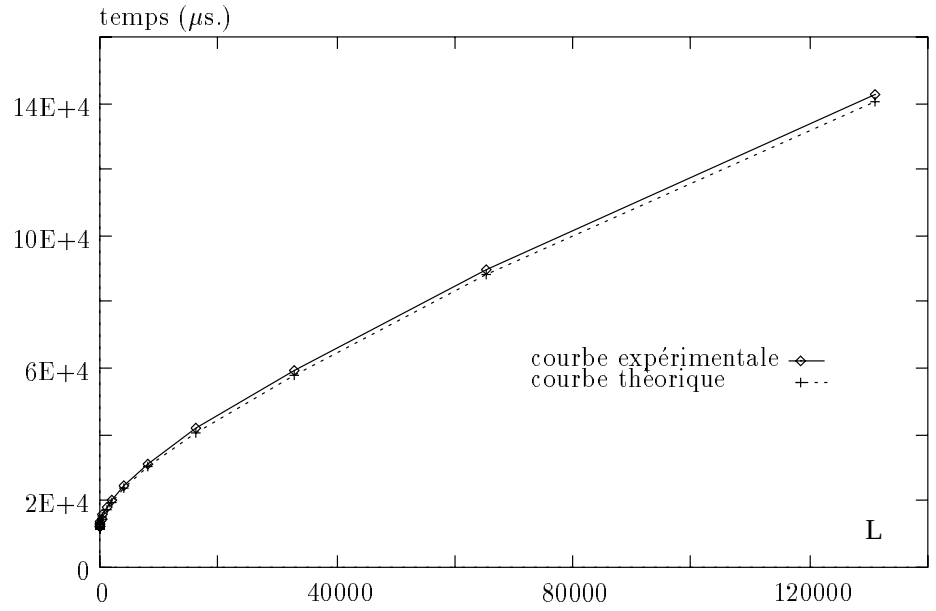


FIG. *6.8 : Comparaison des courbes théorique et expérimentale pour la grille torique* 9×9.

La figure 6.9 permet de voir les résultats des expériences de diffusion dans les grilles toriques en mode *half-duplex*. Comme précédemment les résultat expérimentaux sont trés proches de ceux théoriques.

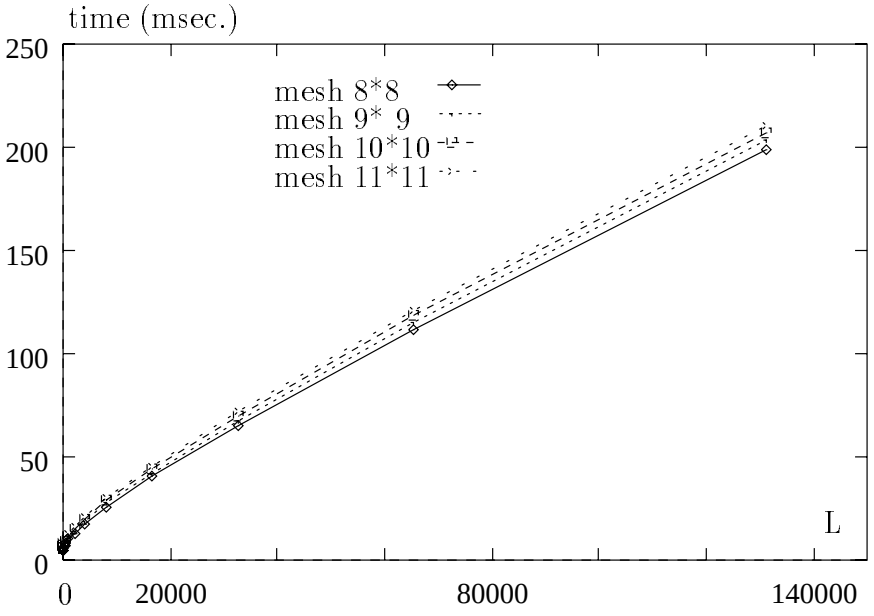


FIG. *6.9 : Diffusion monodirectionnelle dans les grilles toriques.*

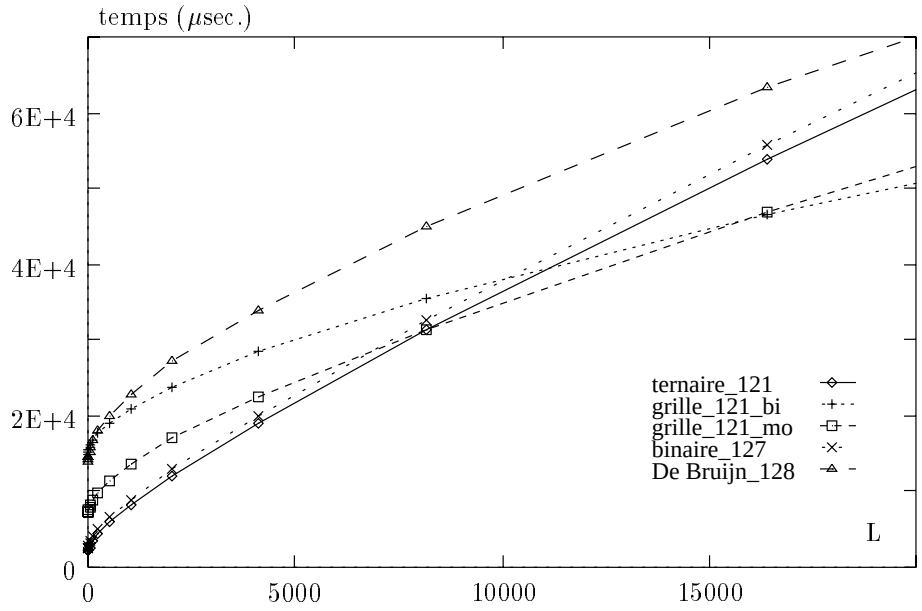


FIG. *6.10 : Comparaison des stratégies de diffusion pour des petits messages.*

Les figures 6.10 et 6.11 montrent les résultats obtenus avec les différents algorithmes et topologies. Afin d'avoir des réseaux ayant quasiment le même nombre de nœuds, notre étude est réalisée sur les grilles toriques et l'arbre ternaire à 121 nœuds, sur l'arbre binaire à 127 nœuds, et sur le De Bruijn à 128 nœuds.

En accord avec l'étude théorique, les expériences montrent que pour des messages très grands, l'algorithme de diffusion bidirectionnelle sur les grilles toriques est le meilleur. Mais pour des messages de taille inférieure à 8192 octets l'arbre ternaire est idéal. Quant aux messages de taille supérieure à 8192 octets mais inférieure à 15000 octets la meilleure performance est obtenue par l'algorithme de diffusion monodirectionnelle sur grille torique.

En fait l'idéal serait de configurer son réseau en cours de programme, ainsi l'utilisateur pourrait choisir la meilleure topologie et le meilleur algorithme de diffusion en fonction de la taille de son message. Mais encore faut-il tenir compte du temps pris par la reconfiguration du réseau. Une étude du coût de reconfiguration du réseau a été réalisée par Desprez et Tourancheau [5] sur un T-Node. Mais cette étude n'est pas applicable au MégaNode du fait des deux niveaux de *switch* à reconfigurer. De plus la reconfiguration en cours d'exécution d'un programme n'a jamais fonctionné sur notre machine.

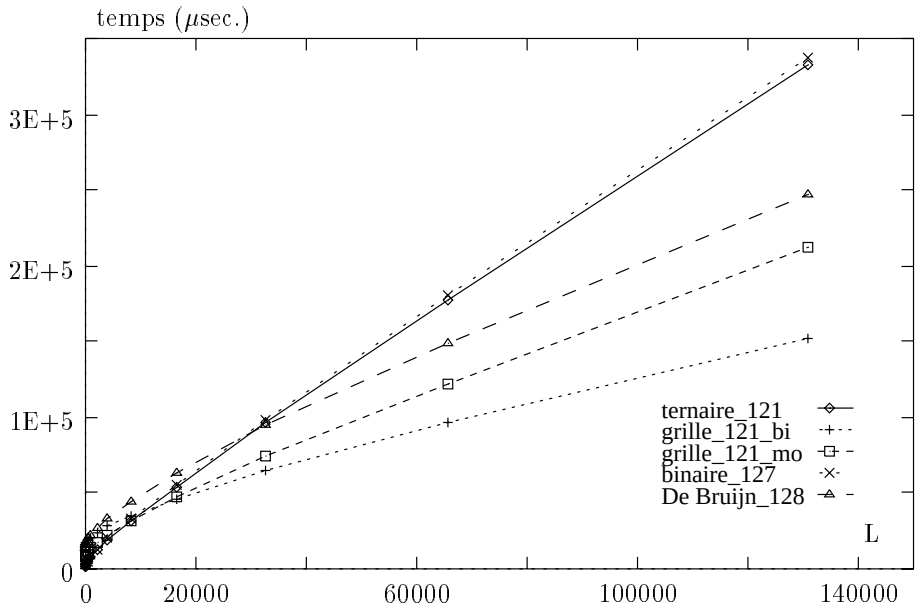


FIG. *6.11 : Comparaison des stratégies de diffusion pour des grands messages.*

6.6 Conclusion

La modélisation que nous avons présentée dans ce chapitre permet de montrer que les performances des algorithmes de diffusion dépendent de l'architecture de la machine. Le fait de comparer ces différents algorithmes avec le modèle de communication linéaire n'est pas suffisant. Il ressort de cette étude expérimentale que l'arbre ternaire offre le meilleur compromis entre performances et facilité d'utilisation pour des petits et moyens messages. Cependant les arbres de recouvrement dans les grilles toriques peuvent être très utiles dans de nombreuses applications (comme le chargement de programme) traitant de gros messages.

Bibliographie

[1] J-M. Adamo and C. Bonello. Ténor++ : a dynamic configurer for SuperNode machines. In Springer Verlag, editor, *COMPAR 90*, September 1990.

[2] Y. Arrouye, F. Bouvry, P. Bouvry, J-P. Kitajima, P. Michallon, J. Prevost, J-L. Roch, and G. Villard. Manuel du MegaNode. Technical Report RT 79, LMC-IMAG (France), April 1992.

[3] J-C. Bermond and P. Fraigniaud. Broadcasting and gossiping in De Bruijn network. Technical Report RR-9104, LIP-ENS-Lyon (France), 1991.

[4] F. Desprez and B. Tourancheau. Modélisation des performances de communication sur T-Node avec le Logical System Transputer Toolset. *La lettre du Transputer*, (7):65–72, September 1990.

[5] F. Desprez and B. Tourancheau. A theoretical study of reconfigurability for basic communication algorithms. In *CONPAR 92-VAPP V*, number 634 in Lecture Notes in Computer Science. Springer Verlag, 1992.

[6] P. Fraigniaud. Performance analysis of broadcasting in hypercube with restricted communication capabilities. *Journal of Parallel and Distributed Computing*, 16(1):15–26, 1992.

[7] Telmat informatique. *T-Node user manual.* 1990.

[8] S. Kuppuswami and B. Tourancheau. Evaluating the performances of Transputer based hypercube vector computer. *La lettre du Transputer*, 4, 1990.

[9] Inmos Ltd. *Transputer reference manual.* Prentice Hall, 1988.

[10] Inmos Ltd. *The Transputer databook.* 1989.

[11] P. Michallon and D. Trystram. Practical experiments of broadcasting algorithms on a configurable parallel computer. *à paraître dans Discrete Applied Math., special issue on broadcasting and gossiping.*

[12] T. Muntean and P. Waille. L'architecture des machines SuperNode. *La lettre du Transputer*, (7):11–40, September 1990.

[13] D. Nicole. Esprit project 1085 : reconfigurable Transputer processor architecture. In *CONPAR 88*, 1988.

Chapitre 7

Recouvrements calculs/communications

7.1 Introduction

La plupart des applications parallèles sont du domaine scientifique (physique, météorologie, médecine, etc.) et leur parallélisation est souvent réalisée avec des bibliothèques de calculs et de communications (voir chapitre 3). Ces parallélisations sont effectuées en décomposant l'application en phases de calculs et de communications bloquantes. Ce type de programmation est facile à mettre en œuvre, mais ne donne pas les meilleurs résultats (faible efficacité). En effet, les processeurs n'effectuent pas de calculs pendant les phases de communications, d'où la nécessité d'utiliser des bibliothèques de communications non-bloquantes. Mais dans certains cas, il subsiste des problèmes, notamment quand le processeur émetteur a plus de calculs à effectuer que le processeur récepteur. Le processeur émetteur peut, dès qu'il a calculé une certaine quantité de données, les envoyer au récepteur qui pourra commencer ces calculs. Ainsi, en pipelinant les envois, on arrive à recouvrir les communications par des calculs.

Toutefois le choix de la taille des données à envoyer n'est pas simple. Ce choix va dépendre de la taille du problème, de la vitesse de calcul des processeurs et du coût des communications. Nous illustrerons la difficulté du calcul de la meilleure taille de paquets à envoyer à chaque étape sur un exemple simple tel que le produit matrice vecteur sur un anneau.

7.2 Pourquoi faire du recouvrement ?

Le temps d'exécution d'un programme sur une machine parallèle à mémoire distribuée est fonction du temps de calcul et du temps de communication. Si

dans ce programme il n'y a pas de recouvrement calculs/communications, alors le temps total d'exécution est égal à la somme des temps de calcul et de communication. Ce type de programme est facile à mettre en œuvre, mais si on observe le comportement d'un tel programme avec un outil de visualisation, on constate que les processeurs perdent du temps à attendre des données, d'où l'idée de faire du calcul pendant que les processeurs communiquent, c'est-à-dire faire du recouvrement calculs/communications.

Afin de mieux comprendre la démarche qui a conduit King, Chu et Ni [7] à proposer une solution *pipeline* des envois de données, nous allons prendre un exemple, et analyser les problèmes qui se posent.

Nous allons considérer deux processeurs P_1 et P_2 sur lesquels s'exécute un programme parallèle. Sur chaque processeur il va s'exécuter un noyau de communication et deux noyaux de calcul T_i^j où i est le numéro du processeur et j, le numéro du noyau de calcul. Afin de simplifier le problème, on considérera que seul le deuxième noyau de calcul d'un des processeurs a besoin des résultats du premier noyau s'exécutant sur l'autre processeur.

Si l'on considère des communications bloquantes, sur cet exemple on obtient le schéma d'exécution représenté sur la figure 7.1.

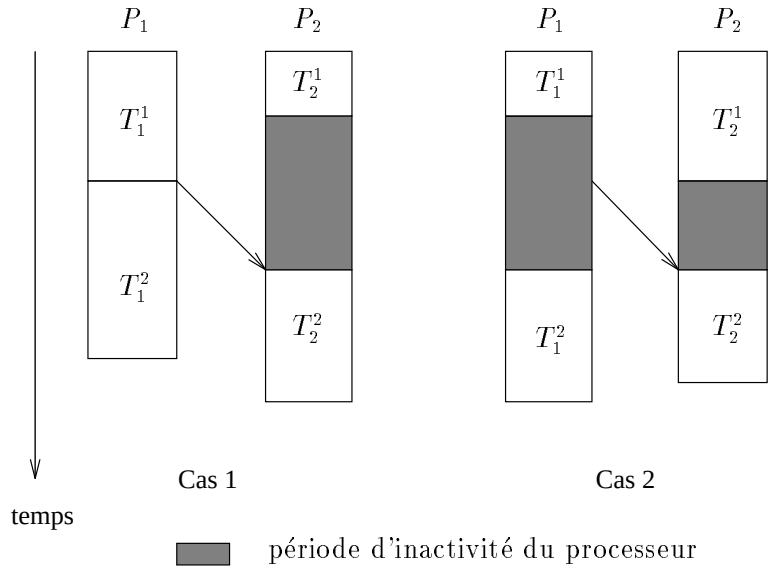


FIG. 7.1 : *Exécution avec des communications bloquantes.*

Dans le cas 1 le processeur P_2 a fini ses calculs avant le processeur P_1, mais il est obligé d'attendre la fin de la phase de calcul de P_1 (T_1^1) avant de recevoir des données, puis de recommencer une seconde phase de calcul (T_2^2). Dans le cas 2 c'est le processeur P_1 qui ne peut pas envoyer ses données tant que le processeur P_2 n'a pas fini sa phase de calcul. Dans les deux cas, on

remarque qu'un des processeurs attend que l'autre arrive dans une phase de communication pour envoyer ou recevoir un message, ce qui crée des périodes d'inactivité du processeur.

Afin de réduire les périodes d'inactivité des processeurs et de permettre aux utilisateurs de faire du recouvrement calculs/communications, des bibliothèques de communication non-bloquantes se sont développées (NX/2, PVM, PICL, etc.). L'utilisation de ces bibliothèques permet de supprimer les périodes d'inactivité des processeurs P_1 et P_2 dans le cas 2. Le processeur P_1 peut commencer T_1^2 sans attendre la fin de la communication, et le processeur P_2 peut enchaîner ses deux noyaux de calcul, puisqu'il a reçu les données nécessaires au noyau T_2^2 pendant le calcul de T_2^1 (voir figure 7.2), d'où recouvrement calculs/communications.

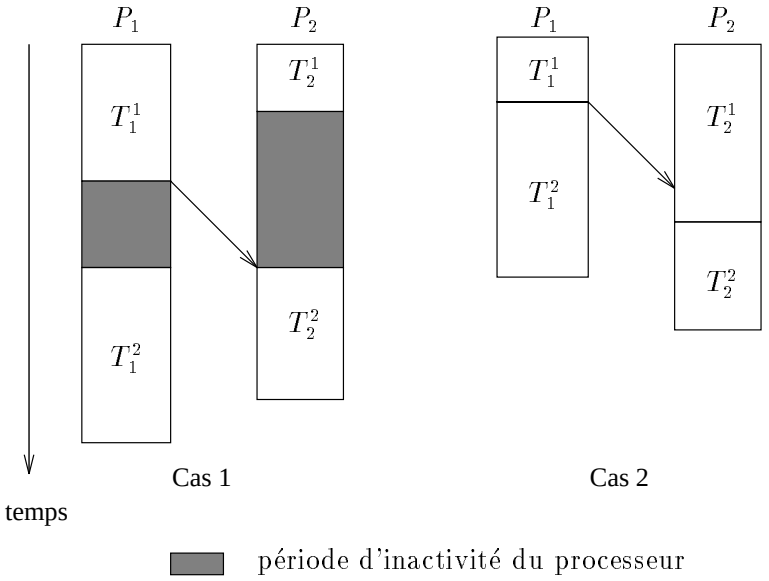


FIG. 7.2 : *Exécution avec des communications non-bloquantes.*

Mais l'utilisation de communications non-bloquantes ne permet pas d'effectuer du recouvrement dans le cas 1. En effet, dans ce cas le processeur P_1 ne peut pas envoyer son message avant d'avoir fini ses calculs. Mais est-ce que le deuxième noyau de calcul T_2^2 a besoin de toutes les données de T_1^1 pour commencer ? Si oui, il est clair que l'on ne pourra pas réaliser de recouvrement calculs/communications plus fin. Mais dans le cas contraire, il est possible de couper le noyau T_1^1 en n noyaux plus petits qui s'exécuteront les uns à la suite des autres. Ainsi, chaque petit noyau peut envoyer à T_2^2 les données qu'il vient de calculer. De cette façon, T_2^2 peut commencer ses calculs avec les données qu'il vient de recevoir.

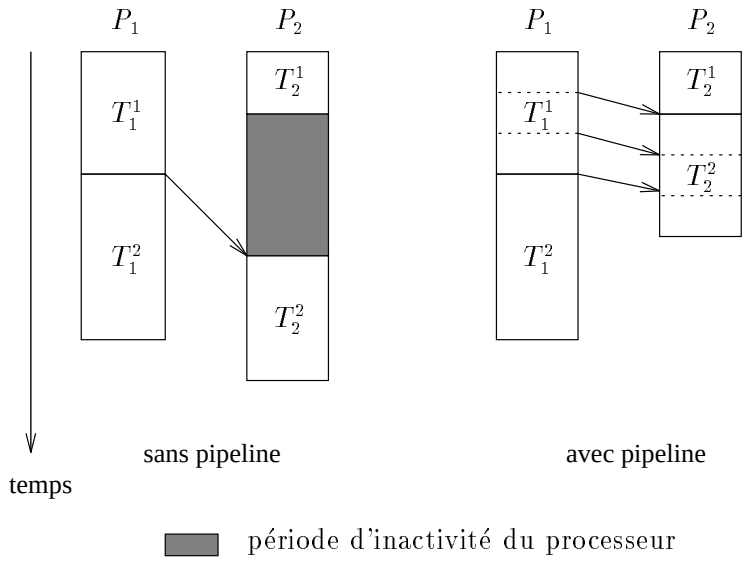


FIG. *7.3 : Exécution avec des communications non-bloquantes pipelinées.*

De nombreux chercheurs ont étudié le problème du recouvrement calculs/communications [1, 4, 7] pour ne citer qu'eux. Mais personne n'a proposé d'outils faciles à utiliser pour le programmeur. Desprez et Tourancheau [3, 4] travaillent sur une bibliothèque permettant de gérer les pipelines et d'exprimer plus facilement les programmes parallèles. Afin de rendre cet outil performant et facilement utilisable par tout programmeur spécialiste ou non, il est nécessaire d'avoir un calcul automatique de la taille des messages à envoyer. L'idéal serait que cette taille soit calculée lors de la compilation ou se fasse de manière dynamique au cours de l'exécution de l'algorithme. Le problème qui se pose est de savoir comment découper ces données et quelle doit être la taille des envois afin que les deux processeurs n'aient pas de période d'inactivité? En effet si les paquets sont trop gros, le recouvrement ne sera pas assez efficace, et au contraire si les paquets sont trop petits il va y avoir beaucoup trop d'initialisations, ce qui nuira aux performances de l'algorithme. Comme nous allons le voir dans la section suivante sur un exemple simple et régulier, ce choix n'est pas si facile.

7.3 Problème du choix de la taille des paquets

L'exemple que nous avons choisi d'étudier pour illustrer le problème de la taille des paquets à échanger est le produit matrice-vecteur, sur une

architecture à mémoire distribuée connectée en anneau [2] (cette étude a été réalisée avec Laurent Colombet et Denis Trystram). Nous allons montrer, en fonction des paramètres de la machine, comment calculer la taille des paquets, et ainsi obtenir un algorithme avec un coût de communication le plus petit possible.

Nous nous sommes intéressés au produit matrice-vecteur, car c'est une des opérations les plus simples et des plus utilisées de l'algèbre linéaire [5]. Le problème consiste à calculer $v = Ax$ où A est une matrice (n, n) et x, v deux vecteurs de dimension n. Chaque composante du vecteur v est le résultat du produit scalaire suivant :

$$v[i] = \sum_{j=1}^{n} A[i,j]\, x[j]. \tag{7.1}$$

Notations et modèles

Si l'on considère un anneau de p processeurs (numérotés de 1 à p), il est possible de réaliser une répartition équilibrée de la matrice A en p blocs de $\lceil \frac{n}{p} \rceil$ ou $\lfloor \frac{n}{p} \rfloor$ lignes. Le type d'allocation choisi des lignes (consécutives, cyclique,*wrap*) n'influe pas sur l'étude qui va suivre. De même on effectue une répartition équilibrée du vecteur x (chaque processeur possède $\lceil \frac{n}{p} \rceil$ ou $\lfloor \frac{n}{p} \rfloor$ composantes du vecteur). De façon à simplifier les formules, nous allons considérer que $\frac{n}{p}$ est entier.

Afin de simplifier le problème, nous allons utiliser des liens *half-duplex*, [8], et un mode 2−ports (voir chapitre 2). De plus, un processeur peut réaliser simultanément des communications et du calcul. L'échange de messages entre deux processeurs est réalisé par commutation de messages (ou *store-and-forward*) [6, 8] que l'on modélise par : $\beta_c + L\tau_c$ (voir chapitre 2).

Le coût d'une opération élémentaire (multiplication suivie d'une addition) est donné par : τ_a

Après avoir présenter l'implémentation standard du produit matrice-vecteur, un principe de recouvrement est donné. Nous étudierons alors, le coût de ce nouvel algorithme et étendrons le modèle aux liens *full-duplex*.

7.3.1 Analyse du produit matrice-vecteur

Comme nous l'avons présenté dans le chapitre 3, la méthode pour paralléliser une application numérique consiste à identifier le découpage des données (décomposition de matrices par ligne, colonnes ou blocs), d'organiser les mouvements de données en utilisant des bibliothèques de communications.

La solution standard

L'équation 7.1 montre que le calcul de chaque composante du vecteur v requière la connaissance de tout le vecteur x, donc chaque processeur doit envoyer ses $\frac{n}{p}$ composantes du vecteur x à tous les autres processeurs. Ce type de mouvement de données correspond à un échange total, plus souvent appelé ATA (pour All To All) [8]. L'algorithme du produit matrice-vecteur est le suivant :

Algorithme 7.1 *produit matrice-vecteur standard*

ATA de taille $\frac{n}{p}$ (partie du vecteur x);
Faire en parallèle sur chaque processeur
Calcul de $\frac{n}{p}$ produits scalaires de taille n (lignes de A et vecteur x);

L'échange total de taille $\frac{n}{p}$ coûte $(p - 1)(\beta_c + \frac{n}{p}\tau_c)$ unités de temps (voir chapitre 5). Le calcul des $\frac{n}{p}$ produits scalaires de taille n nécessite $\frac{n^2}{p}\tau_a$ unités de temps. Ainsi l'algorithme standard du produit matrice-vecteur sur un anneau possédant des liens *half-duplex* a un coût de :

$$T_{stand} = \frac{n^2}{p}\tau_a + (p - 1)\left(\beta_c + \frac{n}{p}\tau_c\right) \tag{7.2}$$

Sans recouvrement calculs/communications, cet algorithme est optimal, et il atteint le temps minimal d'exécution.

Recouvrement des communications

Afin de réduire le temps d'exécution, la solution consiste à masquer au maximum les communications par du calcul.

Durant l'échange total de l'algorithme précédent, il est facile de voir que les processeurs perdent du temps. Ainsi tous les processeurs peuvent commencer les calculs sur leurs données initiales pendant l'échange total. Mais le temps d'un échange total sur un anneau est plus grand que $(\frac{n}{p})^2$ opérations flottantes. Une idée consiste à décomposer l'échange total en une série de communications de voisin à voisin, de façon à masquer le plus possible les communications. En effet, à une étape donnée un processeur peut recevoir de nouvelles données alors qu'il calcule avec les données reçues à l'étape précédente.

Principe de la méthode

Au début chaque processeur possède $\frac{n}{p}$ lignes de la matrice A et $\frac{n}{p}$ composantes du vecteur x et peut ainsi calculer $\frac{n}{p}$ produits scalaires partiels v_i. Ainsi, un processeur peut exécuter $(\frac{n}{p})^2$ opérations flottantes et recevoir simultanément au plus $\frac{n}{p}$ données de ses voisins. En fonction des paramètres de la machine $(p, \beta_c, \tau_c$ and $\tau_a)$ et de la taille du problème, nous pouvons observer deux cas :

- $\beta_c + \frac{n}{p}\tau_c \leq \frac{n^2}{p^2}\tau_a$ (recouvrement total)
- $\beta_c + \frac{n}{p}\tau_c > \frac{n^2}{p^2}\tau_a$ (recouvrement partiel)

Dans le premier cas, comme le temps de communication des $\frac{n}{p}$ éléments est plus petit qu'un calcul local, toutes les communications peuvent être masquées.

Dans le deuxième cas, le temps des calculs locaux correspond à une communication d'un nombre de données plus petit que $\frac{n}{p}$. Soit λ_1 le plus grand entier tel que $\beta_c + \lambda_1\tau_c \leq \frac{n^2}{p^2}\tau_a$, ainsi toutes les communications de taille λ_1 données peuvent être masquées $(\lambda_1 < \frac{n}{p})$.

Maintenant chaque processeur a λ_1 nouvelles opérations locales à exécuter. On itère ce principe jusqu'à ce qu'il ne soit plus possible de réaliser des communications. Ce principe permet de construire une série de λ_t $(t = 0, ...N)$ de $\lambda_0 = \frac{n}{p}$ à $\lambda_{N+1} = 0$ tel que l'on puisse calculer un maximum d'opérations locales, tout en masquant les communications.

7.3.2 Description des algorithmes

Recouvrement total $(\beta_c + \frac{n}{p}\tau_c \leq \frac{n^2}{p^2}\tau_a)$

Dans ce cas toutes les communications peuvent être masquées. L'algorithme parallèle du produit matrice-vecteur devient :

Algorithme 7.2 *produit matrice-vecteur parallèle avec recouvrement total*

/ Algorithme du processeur q avec q=1...p */*
Pour $t \leftarrow 1$ *à* $p-1$ *faire en parallèle*
  Envoyer à (q+1 modulo p) les $\frac{n}{p}$ *éléments de x reçus à l'étape précédente;*
  Calculer les $\frac{n}{p}$ *produits scalaires locaux;*
  Recevoir de (q-1 modulo p) les nouveaux $\frac{n}{p}$ *éléments de x;*
Calcul des $\frac{n}{p}$ *derniers produits scalaires;*

Il y a $(p-1)$ étapes où les communications sont totalement masquées par les calculs locaux d'un coût de $\frac{n^2}{p^2}\tau_a$, plus une dernière étape de calculs locaux sans communication. Cet algorithme a un coût total de $\frac{n^2}{p}\tau_a$.

Recouvrement partiel $(\beta_c + \frac{n}{p}\tau_c > \frac{n^2}{p^2}\tau_a)$

- **Phase I: recouvrement total**

Dans ce cas à l'étape t un processeur calcule $\lambda_{t-1}\frac{n}{p}$ et reçoit (envoit) λ_t données. La valeur de λ_t est donnée par la formule suivante :

$$\lambda_t = \left\lfloor \frac{n}{p}\frac{\tau_a}{\tau_c}\lambda_{t-1} - \frac{\beta_c}{\tau_c} \right\rfloor \tag{7.3}$$

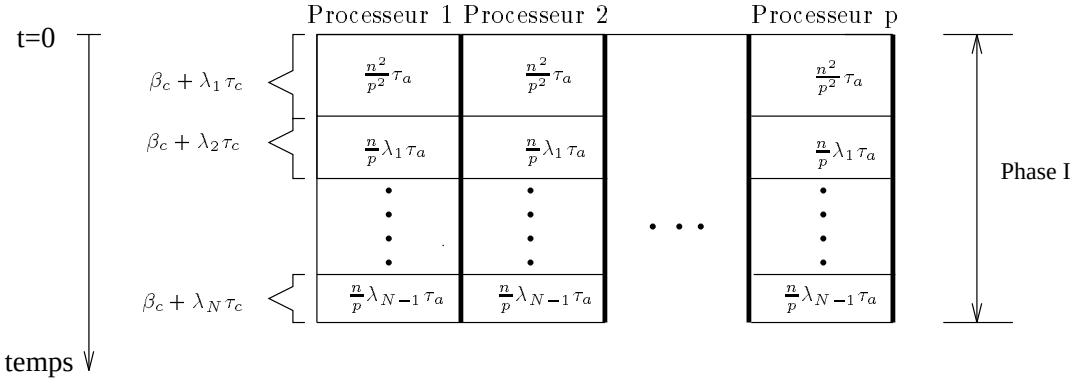


FIG. *7.4 : Schéma d'exécution de la Phase I.*

Nous pouvons facilement vérifier par induction que la série des λ_t est strictement décroissante. Pendant les N premières étapes les communications sont totalement masquées. Après, il ne reste plus assez de données localement

pour effectuer des calculs permettant de masquer une communication (voir figure 7.4). Par la suite nous nommerons Phase I l'exécution des N premières étapes. Après avoir calculé au préalable la série des λ_t, l'agorithme de la Phase I est le suivant :

Algorithme 7.3 *Phase I*

/ algorithme du processeur q avec q=1...p */*
$t \leftarrow 1$;
Tant que $\lambda[t] \geq 1$ *faire en parallèle*
  Envoyer au processeur (q+1 modulo p) les $\lambda[t]$ *éléments de x reçus à l'étape précédente;*
  Calculer les $\lambda[t-1]$ *produits scalaires locaux;*
  Recevoir du processeur (q-1 modulo p) les nouveaux $\lambda[t]$ *éléments de x;*
  $t \leftarrow t+1$;
Fin tant que

- **Le reste des calculs**

Après la Phase I, il reste λ_N calcul locaux à effectuer en parallèle sur chaque processeur. Pendant ces calculs, chaque processeur peut communiquer le maximum de données (c'est-à-dire $\frac{n}{p}$) à son voisin. Ainsi, il est possible de redémarrer une nouvelle phase de recouvrement total (voir figure 7.5 où les périodes d'inactivité des processeurs sont représentées en gris). On constate facilement que pour de grand β_c, cette stratégie est la meilleure que l'on puisse réaliser.

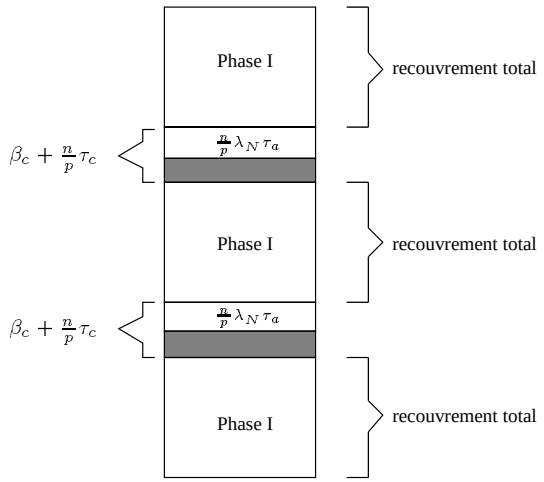


FIG. *7.5 : Phases successives.*

Nous répétons $\left\lceil \frac{n}{\sum_{i=0}^{N-1} \lambda_i} \right\rceil$ fois la succession de Phase I.

L'algorithme de recouvrement partiel est donc :

Algorithme 7.4 *Les phases successives*

/ algorithme du processeur q avec q=1...p */*
Exécuter la Phase I;
Pour $k \leftarrow 1$ *à* $\left\lceil \frac{n}{\sum_{i=0}^{N-1} \lambda_i} \right\rceil$
  Faire en parallèle
    Envoyer au processeur (q+1 modulo p) les $\frac{n}{p}$ *éléments de* x *reçus à l'étape précédente;*
    Calculer les $\lambda[N]$ *produits scalaires locaux;*
    Recevoir du processeur (q-1 modulo p) les nouveaux $\frac{n}{p}$ *éléments de* x*;*
  Fin faire en parallèle;
  Exécuter la Phase I;
Fin faire;

Remarque.
Dans le pire des cas, la dernière exécution de la Phase I se termine avant la N^{th} étape. Donc avant d'exécuter la dernière phase de recouvrement il est possible de trouver un λ_t plus petit que $\frac{n}{p}$ afin de minimiser le temps d'inactivité des processeurs, mais ceci est très difficile à réaliser automatiquement. La solution consiste à tester pendant la Phase I le nombre de produits scalaire locaux, et de stopper l'algorithme lorsque tous les produits scalaires ont été calculés. Ceci peut être réalisé en ajoutant un test dans l'algorithme 7.3.

Evolution de l'allocation des données

Comme à chaque étape tous les processeurs exécutent la même tâche n'importe quel processeur peut être choisi comme référence. Notons **RP** ce processeur.

A une étape **t** un processeur exécute $\lambda_{t-1}\frac{n}{p}$ opérations, envoie et reçoit λ_t éléments du vecteur x.

On rappel que la série des λ_i décroit, et la Phase I s'arrête à l'étape N. En fait à cette étape λ_{N+1} est plus petit que 1. De plus, à cette étape le processeur **RP** n'a pas encore reçu toutes ses données. Avant de continuer il est nécessaire de connaître la distance entre **RP** et le processeur le plus

éloigné qui possède encore des données destinées à **RP** ainsi, que le nombre d'éléments contenus dans ce processeur. A chaque étape **t** de l'algorithme, nous pouvons calculer la distance entre **RP** et ce processeur. Nous noterons d_t cette distance, et λ_c le nombre de composantes du vecteur x nécessaire aux calculs locaux de **RP** qui se trouvent dans ce processeur (processeur à distance d_t de **RP**). La proposition 7.3.1 donne le nombre d'éléments destinés à **RP** restant sur chaque processeurs.

Dans un échange total, le processeur **RP** doit recevoir $(p - 1)$ messages de taille $\frac{n}{p}$ c'est-à-dire un message de taille $\frac{n}{p}$ à chaque étape, d'où $d_{t+1} = d_t - 1$. Mais dans notre algorithme à chaque étape **RP** reçoit un message de taille λ_t, ce qui complique l'expression de d_t, qui est donnée par la proposition 7.3.2. Cette valeur nous permettra de calculer le temps final de l'algorithme.

Proposition 7.3.1
*Les processeurs situés à une distance supérieure à d_t ne possèdent pas d'éléments destinés à **RP**.*
*Le processeur situé à une distance d_t possède λ_c éléments destinés à **RP**.*
*Les processeurs situés à une distance inférieure à d_t possèdent $\frac{n}{p}$ éléments destinés à **RP**.*

Preuve. On peut facilement vérifier que la proposition est vraie pour l'étape 1, lorsque tous les processeurs envoient λ_1 éléments.

Supposons que cette proposition soit vraie pour l'étape **t≥1**, montrons qu'elle est aussi vraie pour l'étape **t+1**.

Nous avons deux cas à considérer $\lambda_{t+1} < \lambda_c$ ou $\lambda_{t+1} \geq \lambda_c$.

- si $\lambda_{t+1} < \lambda_c$ alors nous avons $d_{t+1} = d_t$ et $\lambda_c \leftarrow \lambda_c - \lambda_{t+1}$:

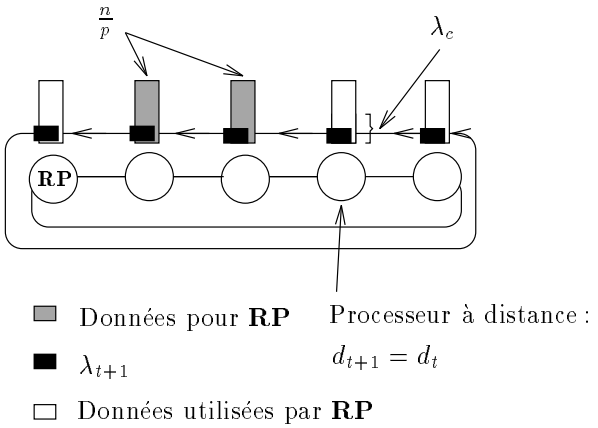


FIG. 7.6 : *Cas :* $\lambda_{t+1} < \lambda_c$.

Les processeurs à distance plus petites que d_t envoient λ_{t+1} et reçoivent λ_{t+1} données utiles à **RP**, ainsi ils possèdent toujours $\frac{n}{p}$ données destinées à **RP**. En utilisant le même argument on prouve que les processeurs situés à une distance supérieure à d_t ne possèdent plus de données destinées à **RP** (voir figure 7.6). Le processeur situé à la distance d_t envoie λ_{t+1} données destinées à **RP** et il ne reçoit pas de données. Donc il reste $\lambda_c \leftarrow \lambda_c - \lambda_{t+1}$ données sur ce processeur. Dans ce cas on a $d_{t+1} = d_t$.

• si $\lambda_{t+1} \geq \lambda_c$ alors nous avons $d_{t+1} = d_t - 1$:

Le processeur à distance d_{t+1} reçoit λ_c éléments de x nécessaire à **RP** et $\lambda_{t+1} - \lambda_c$ ont déjà été utilisées par **RP**.

Ce processeur dispose maintenant de $\lambda_c = \frac{n}{p} + \lambda_c - \lambda_{t+1}$ données et il est facile de voir que les processeurs à distance inférieure à d_{t+1} possèdent $\frac{n}{p}$ éléments et que ceux à distance supérieure à d_{t+1} ne possèdent plus d'éléments destinés à **RP** (voir figure 7.7).

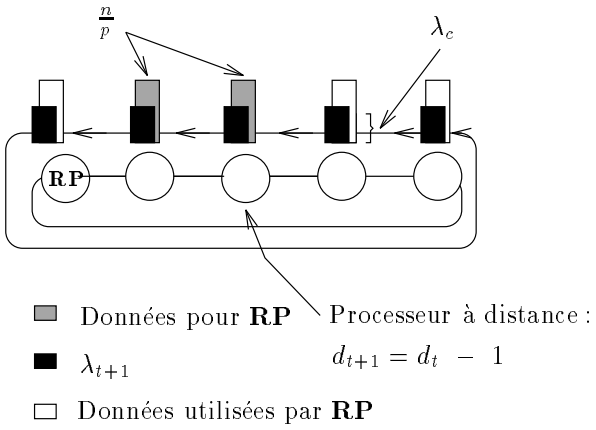


FIG. 7.7 : *Cas :* $\lambda_{t+1} \geq \lambda_c$.

□

Proposition 7.3.2 *La distance d_t est égale à :*

$$d_t = p - 1 - \left\lfloor \frac{p}{n} \sum_{i=1}^{t} \lambda_i \right\rfloor \tag{7.4}$$

Preuve. Quand **RP** reçoit un message de taille λ_{t+1}, le nombre de messages de taille $\frac{n}{p}$ déjà reçu augmente et $d_{t+1} = d_t - 1$ ou alors le nombre de

messages reçus de taille $\frac{n}{p}$ reste le même et dans ce cas $d_{t+1} = d_t$. Donc nous devons considérer deux cas.

- cas $\lambda_{t+1} < \lambda_c$ Nous avons $\left\lfloor \frac{p}{n} \sum_{i=1}^{t+1} \lambda_i \right\rfloor = \left\lfloor \frac{p}{n} \sum_{i=1}^{t} \lambda_i \right\rfloor$.
  Alors $d_{t+1} = d_t = p - 1 - \left\lfloor \frac{p}{n} \sum_{i=1}^{t} \lambda_i \right\rfloor = p - 1 - \left\lfloor \frac{p}{n} \sum_{i=1}^{t+1} \lambda_i \right\rfloor$.
- cas $\lambda_{t+1} \geq \lambda_c$ Nous avons $\left\lfloor \frac{p}{n} \sum_{i=1}^{t+1} \lambda_i \right\rfloor = 1 + \left\lfloor \frac{p}{n} \sum_{i=1}^{t} \lambda_i \right\rfloor$.
  Alors $d_{t+1} = d_t - 1 = p - 1 - \left\lfloor \frac{p}{n} \sum_{i=1}^{t} \lambda_i \right\rfloor - 1 = p - 1 - \left\lfloor \frac{p}{n} \sum_{i=1}^{t+1} \lambda_i \right\rfloor$.

□

La proposition suivante nous permet de connaître la valeur de λ_c à chaque étape.

Proposition 7.3.3 *A une étape* **t**, *la valeur de* λ_c *est égale à* $\frac{n}{p} - (\sum_{i=1}^{t} \lambda_i) \; mod \; \frac{n}{p}$.

Preuve. Si nous considérons l'étape 1, tous les processeurs envoient λ_1 éléments de x et masquent le coût $\beta_c + \lambda_1 \tau_c$ des communications. Alors le processeur à distance $d_t = d_1$ possède $\lambda_c = \frac{n}{p} - \lambda_1$ or $\lambda_1 < \frac{n}{p}$ donc on peut écrire $\lambda_c = \frac{n}{p} - (\lambda_1 \; mod \frac{n}{p})$.

La formule est vérifiée pour l'étape 1, supposons la vraie à l'étape **t** et prouvons qu'elle est vraie à l'étape **t+1** quand nous envoyons λ_{t+1} éléments.

- Si $\lambda_{t+1} < \lambda_c$
  Nous savons que $d_{t+1} = d_t$ et le nouveau $\lambda_c = \lambda_c - \lambda_{t+1}$.
  Donc
  $$new\lambda_c = \frac{n}{p} - \left(\sum_{i=1}^{t} \lambda_i\right) \; mod \; \frac{n}{p} - \lambda_{t+1} mod \; \frac{n}{p}$$
  Alors la valeur de λ_c à l'étape **t+1** est :
  $$new\lambda_c = \frac{n}{p} - \left(\sum_{i=1}^{t+1} \lambda_i\right) \; mod \; \frac{n}{p}$$
- Si $\lambda_{t+1} \geq \lambda_c$
  Nous savons que $d_{t+1} = d_t - 1$ et le nouveau $\lambda_c = \frac{n}{p} + \lambda_c - \lambda_{t+1}$.
  Donc
  $$new\lambda_c = \frac{n}{p} + \frac{n}{p} - \left(\left(\sum_{i=1}^{t} \lambda_i\right) \; mod \; \frac{n}{p} + \lambda_{t+1}\right)$$

Ainsi

$$new\lambda_c = \frac{n}{p} + \frac{n}{p} - \left(\frac{n}{p} + \left(\sum_{i=1}^{t+1} \lambda_i\right) \ mod \ \frac{n}{p}\right) = \frac{n}{p} - \left(\sum_{i=1}^{t+1} \lambda_i\right) \ mod \ \frac{n}{p} \tag{7.5}$$

En utilisant la formule suivante dans l'équation 7.5, on démontre la proposition.

$$\left(\sum_{n=1}^{t} \lambda_i\right) \ mod \ \frac{n}{p} \ + \ \lambda_{t+1} = \frac{n}{p} \ + \ \left(\sum_{i=1}^{t+1} \lambda_i\right) \ mod \ \frac{n}{p}$$

□

7.3.3 Calcul du temps d'exécution

Après la Phase I, les processeurs doivent envoyer des données sans recouvrement. Afin de terminer le calcul du vecteur v, l'algorithme consiste à répéter tant que cela est nécessaire la phase I (voir 7.3.2). D'où le temps final de :

$$T_{exec} = \frac{n^2}{p}\tau_a + \left\lfloor \frac{n}{\frac{n}{p} + \sum_{i=1}^{N-1} \lambda_i} \right\rfloor (\beta_c + \frac{n}{p}\tau_c) \tag{7.6}$$

où $\left\lfloor \frac{n}{\frac{n}{p}+\sum_{i=1}^{N-1} \lambda_i} \right\rfloor$ représente le nombre de communications $(\beta_c + \frac{n}{p}\tau_c)$ non masquées nécessaires à la ré-initialisation des Phases I.

7.3.4 Adaptation au lien *full-duplex*

Nous allons maintenant considérer qu'un processeur peut envoyer et recevoir en parallèle sur chacun de ses liens, et allons réaliser la même étude que précédemment.

On constate que dans ce cas un processeur reçoit deux fois plus de données. Pendant le temps $\frac{n^2}{p^2}\tau_a$ correspondant aux calculs locaux un processeur peut envoyer $\lambda_1 = \left\lfloor \frac{n^2}{p^2}\frac{\tau_a}{\tau_c} - \frac{\beta_c}{\tau_c} \right\rfloor$ à ses deux voisins et recevoir $2\lambda_1$ éléments du vecteur x.

Ainsi nous obtenons deux cas:

- $2\lambda_1 \geq \frac{n}{p}$
  A l'étape suivante, chaque processeur a plus de calculs locaux à exécuter, et peut envoyer un nombre d'éléments de x plus grand que λ_1. D'où à la prochaine étape chaque processeur pourra envoyer à son voisin de droite (gauche) les données reçues de son voisin de gauche (droite) à l'étape précédente, plus une partie des données initiales restantes. On notera toutefois qu'un processeur ne peut pas envoyer plus de $\frac{n}{p}$ données dans la même direction.
  Le nombre d'éléments de x qui doivent être envoyés à l'étape **t** est donné par: $\lambda_t = min\left(\left\lfloor \frac{n}{p}\frac{\tau_a}{\tau_c}2\lambda_{t-1} - \frac{\beta_c}{\tau_c}\right\rfloor, \frac{n}{p}\right)$. On constate que cette série de λ_i croît et atteint la borne supérieure $\frac{n}{p}$. Dans ce cas, toutes les communications sont masquées (voir algorithme 7.5).

Algorithme 7.5

/ algorithme du processeur q avec q=1...p */*
$\lambda[0] \leftarrow \frac{n}{p}$;
$\lambda[1] \leftarrow \left\lfloor \frac{n}{p}\frac{\tau_a}{\tau_c}\frac{n}{p} - \frac{\beta_c}{\tau_c}\right\rfloor$;
$t \leftarrow 1$;
Tant que $(\frac{n}{p} + 2\sum_{i=1}^{t-1}\lambda_i \leq n)$ *faire en parallèle*
  Envoyer au processeur (q+1 modulo p) les $\lambda[t]$ *éléments de x utilisés par le processeur (q-1 modulo p);*
  Envoyer au processeur (q-1 modulo p) les $\lambda[t]$ *éléments de x utilisés par le processeur (q+1 modulo p);*
  Si (t=1) alors calculer les $\lambda[t-1]$ *produits scalaires sinon calculer les* $2\lambda[t-1]$ *produits scalaires;*
  Recevoir du processeur (q+1 modulo p) et processeur (q-1 modulo p) les nouveaux $\lambda[t]$ *éléments de x;*
  $t \leftarrow t+1$;
  $\lambda[t] \leftarrow min\left(\left\lfloor \frac{n}{p}\frac{\tau_a}{\tau_c}2\lambda[t-1] - \frac{\beta_c}{\tau_c}\right\rfloor, \frac{n}{p}\right)$;
Fin tant que;

Le temps total d'exécution de cette algorithme est :

$$T_{exec} = \frac{n^2}{p}\tau_a$$

- $2\lambda_1 < \frac{n}{p}$

  Dans ce cas la série des $\lambda_t = \left\lfloor \frac{n}{p}\frac{\tau_a}{\tau_c}2\lambda_{t-1} - \frac{\beta_c}{\tau_c}\right\rfloor$ est strictement décroissante. Nous ne pouvons pas avoir un recouvrement total, mais une succession de Phase I entrecoupées de communications d'un coût de $\beta_c + \frac{n}{p}\tau_c$ permettant la ré-initialisation des Phases I. Après la première ré-initialisation, chaque processeur doit effectuer $2\frac{n^2}{p^2}$ calculs locaux (voir figure 7.8). Ainsi il est possible de commencer une nouvelle Phase I avec $\lambda_1' = min\left(\left\lfloor 2\frac{n^2}{p^2}\frac{\tau_a}{\tau_c} - \frac{\beta_c}{\tau_c}\right\rfloor, \frac{n}{p}\right)$ plus grand que $\lambda_1 = \left\lfloor \frac{n^2}{p^2}\frac{\tau_a}{\tau_c} - \frac{\beta_c}{\tau_c}\right\rfloor$. La nouvelle série de λ_t' est définie par :$\lambda_t' = min\left(\left\lfloor \frac{n}{p}\frac{\tau_a}{\tau_c}2\lambda_{t-1}' - \frac{\beta_c}{\tau_c}\right\rfloor, \frac{n}{p}\right)$. Du fait des coûts des communications et du coût des calculs locaux, il faut distinguer deux cas.

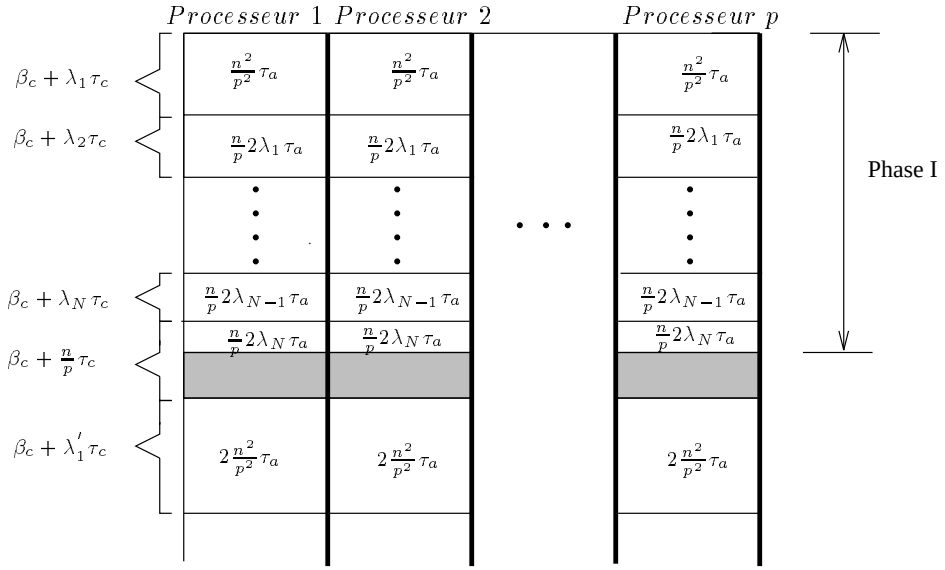


FIG. *7.8 : Le nouvel enchainement des phases.*

- Si $\beta_c + \frac{n}{p}\tau_c \leq 2\frac{n^2}{p^2}\tau_a$ alors à chaque étape un processeur peut calculer $2\frac{n}{p}$ produits scalaires et recevoir $2\frac{n}{p}$ éléments. Dans ce cas le temps total d'exécution est :

$$T_{exec} = \left(\frac{n^2}{p} - \frac{n}{p}\lambda_N\right)\tau_a + \left(\beta_c + \frac{n}{p}\tau_c\right)$$

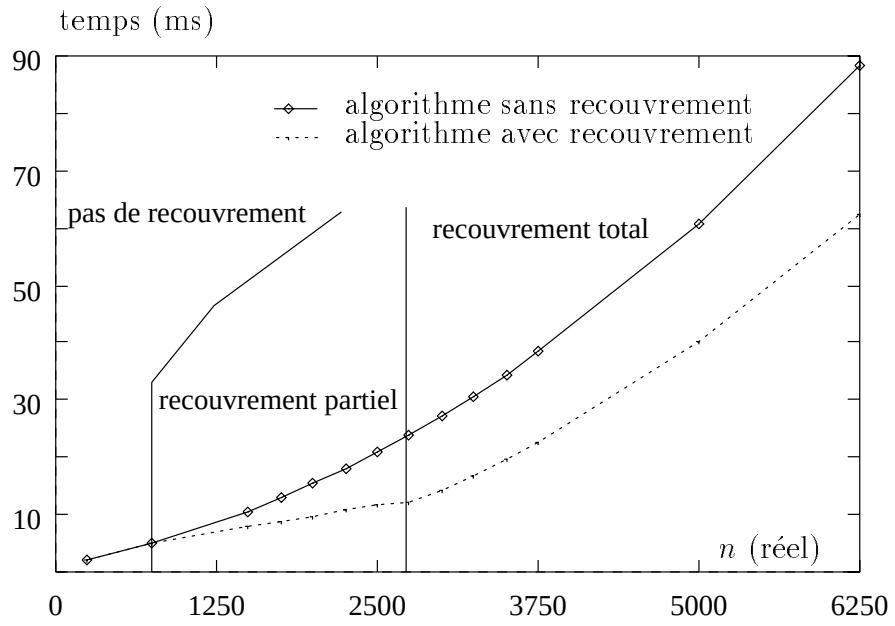
- Si $\beta_c + \frac{n}{p}\tau_c > 2\frac{n^2}{p^2}\tau_a$ alors la série des λ_i' est strictement décroissante, et nous avons une succession de Phase I, entrecoupées de communications ayant un coût de $\beta_c + \frac{n}{p}\tau_c$. Le temps total d'exécution est :

$$T_{exec} = \frac{n^2}{p}\tau_a + \left\lfloor \frac{n - \sum_{i=0}^{N}\lambda_i}{\sum_{i=0}^{N'}\lambda_i'}\right\rfloor\left(\beta_c + \frac{n}{p}\tau_c\right)$$

7.3.5 Simulations

Après avoir fait une étude théorique, nous allons présenter des résultats obtenus par simulation dans le cas des liens *half-duplex* et *full-duplex*. Les paramètres utilisés correspondent au coût des nouvelles machines parallèles avec des processeurs puissants et une faible latence. Dans le reste de notre étude nous considérons le nombre de processeurs égal à 100.

La figure 7.9 compare l'algorithme sans recouvrement avec le nouvel algorithme présenté précédemment sur un anneau en utilisant des liens *half-duplex*.



$\beta_c = 10\ \mu s$; $\tau_c = 1\ \mu s/\text{byte}$; $\tau_a = 0.01\ \mu s$

FIG. 7.9 : *Comparaison des algorithmes (liens half-duplex).*

Pour des petites tailles de n, il n'y a pas de recouvrement car le coût du *start-up* est très élevé. La série des $\lambda[i]$ est dégénérée (réduite à un terme). Pour des tailles moyennes de n nous avons un nombre important de termes dans la série des $\lambda[i]$. Les figures 7.10 et 7.11 représentent l'évolution de la série des $\lambda[i]$ pour des tailles égales à 2000 er 2625 réels. Enfin pour les grandes tailles de n, il y a un recouvrement total, car les calculs locaux sont très importants.

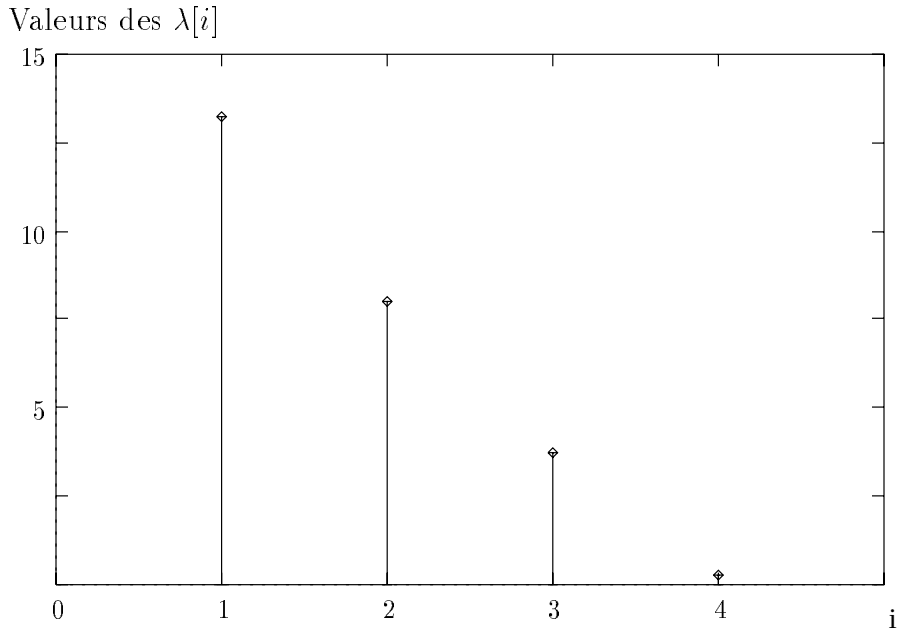


FIG. *7.10 : La série des* $\lambda[i]$ *pour* $n = 2000$.

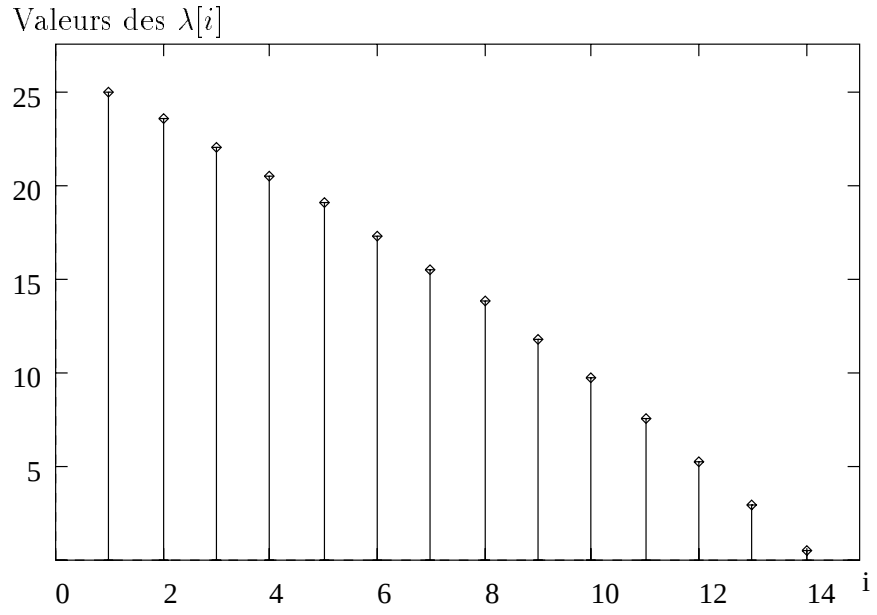
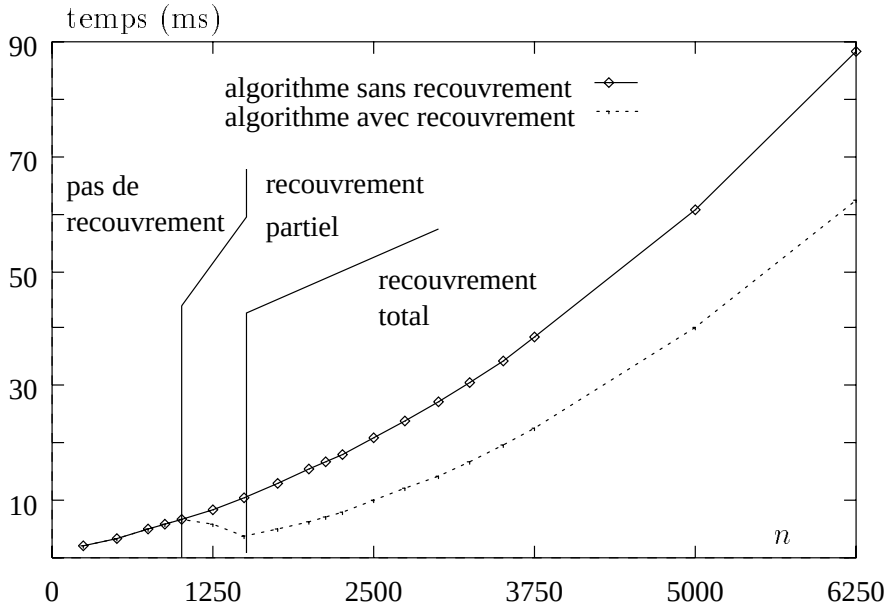


FIG. *7.11 : La série des* $\lambda[i]$ *pour* $n = 2500$.

La figure 7.12 montre les résultats obtenus avec des liens *full-duplex*. Comme les processeurs reçoivent plus de données, nous obtenons un recouvrement pour de plus petites valeurs de n.



$\beta_c = 10\ \mu s$; $\tau_c = 1\ \mu s/\text{byte}$; $\tau_a = 0.01\ \mu s$

FIG. *7.12 : Comparaison des algorithmes (liens full-duplex)*

Ce nouvel algorithme permet de masquer au mieux les communications. Les simulations montrent que sur les machines actuelles, il est possible d'obtenir des gains importants en effectuant du recouvrement calculs/communications.

7.4 Conclusion

L'utilisation de bibliothèques de communications non-bloquantes permet de réaliser des recouvrements calculs/communications. Ceci est d'autant plus facile sur les machines actuelles que tous les nœuds possèdent un processeur dédié aux communications. On a cherché à masquer au mieux les communications, pour cela nous avons étudié, sur un algorithme simple, la taille des messages que nous devons envoyer dans chaque communication. L'étude révèle que ce calcul n'est pas simple et qu'il risque de devenir très compliqué pour des problèmes plus complexes.

Afin que ce type de méthode soit utilisable il est nécessaire que le découpage des messages soit calculé lors de la compilation.

Bibliographie

[1] C. Calvin and F. Desprez. Minimizing communication overhead using pipeline for multi-dimensional FFT on distributed memory machines. In *Parallel Compar*, 1993.

[2] L. Colombet, P. Michallon, and D. Trystram. Parallel matrix-vector product on a ring with minimum of communications. Apache, LMC - IMAG (France), 1994.

[3] F. Desprez. *Procèdures de base pour le calcul scientifique sur machines paralleèles à mémoire distribuée.* PhD thesis, Institut National Polytechnique de Grenoble, January 1994.

[4] F. Desprez and B. Tourancheau. LOCCS low overhead communication and computation subroutines. In *High Performance Computing and Networking conference*, 1993.

[5] G.H. Golub and C.F. Van Loan. *Matrix Computation.* The John Hopkins University Press, 1989. Second edition.

[6] L. Johnsson and C.T. Ho. Optimal broadcast and personalized communication on hypercubes. *IEEE Transactions on Computers*, pages 1249–1268, 89.

[7] C. T. King, W. H. Chu, and L. M. Ni. Pipelined data parallel algorithms - concept and modeling. In *International Conference on Supercomputing*, pages 385–395, 1988.

[8] Y. Saad and M. Schultz. Data communication in parallel Architectures. *Parallel Computing*, pages 131–150, 1989.

Chapitre 8

Communications globales en commutation de circuit

Dans le mode de communication de message (*store and forward*) le temps des communications globales est proportionnel au diamètre du réseau, ce qui nuit à l'utilisation de réseaux tels que les grilles toriques et plus généralement les réseaux possédant un grand diamètre. Comme nous l'avons vu dans le premier chapitre, les constructeurs de machines parallèles utilisent de plus en plus la commutation de circuit, ou le *wormhole* pour effectuer les communications entre les processeurs. Or le coût des communications ne dépend plus que faiblement de la distance entre les processeurs dans le mode commutation de circuit, ce qui permet l'utilisation de réseaux possédant un diamètre élevé telles que les grilles ou de grilles toriques. Certains constructeurs comme Intel et Cray ont déjà mis en application ce mode de routage sur ce type de réseaux (Paragon, iWarp, T3D, cf. chapitre 1).

Dans ce chapitre nous allons présenter dans un premier temps la diffusion dans la grille torique, travaux de Peters et Syska [11]. La solution proposée permet de réaliser une diffusion en un nombre minimal d'étapes de communication. Puis, nous utiliserons cet algorithme pour en déduire une méthode permettant de réaliser l'échange total dans les grilles toriques. Enfin nous comparerons les différents algorithmes de diffusion et d'échange total en commutation de circuit, afin de déduire, en fonction de la taille du message, l'algorithme le mieux adapté.

8.1 Résultat préliminaire

Depuis les années 90, avec l'apparition de nouveaux modes de communication dans certaines machines, les communications globales en commutation de circuit ou en *wormhole* [5] sont très étudiées, notamment dans les grilles

et les grilles toriques de processeurs. Barnett *et al.* proposent des solutions pour la diffusion et le regroupement avec accumulation (*global combine*) dans les grilles [1, 2]. Fraigniaud et Peters présentent des algorithmes de diffusion et d'échange total dans les grilles toriques en mode 1-port [7]. Le problème de la transposition a donné lieu aussi à beaucoup d'études, notamment de la part de Ho [8] et Choi, Dongarra, Walker [4]. Lin et Ni se sont intéressés au problème du *multicast* dans les grilles. Ceci consiste à diffuser un message à un sous-ensemble de processeurs du réseau [9, 10]. L'étude des communications globales en commutation de circuit est aussi réalisée expérimentalement [3, 13] sur des machines telles que les iPSC, la Paragon, etc.

Dans ce chapitre nous considérons des grilles toriques ayant p processeurs, dans lesquelles les communications se font en mode commutation de circuit ou *wormhole*, Δ-ports, *full-duplex* (voir chapitre 2). Dans ce mode de communication, les nœuds intermédiaires par lesquels passe un message qui ne leur est pas destiné ne le mémorisent pas. Le temps pour communiquer un message de taille L entre deux processeurs distants de d est de :

$$T_d = \alpha + d\delta + L\tau \tag{8.1}$$

où α est le temps de *start-up*, δ le temps de routage dans un nœud, et $L\tau$ le temps de transmission du message.

Comme que ce mode de communication permet d'envoyer en une seule étape un message à Δ nœuds situés à une distance d, l'émetteur peut envoyer ce même message à Δ autres nœuds lors de l'étape suivante, chose qui est impossible de réaliser en *store and forward*. Le nombre maximal de nœuds que l'on peut informer en k étapes est donné par la proposition suivante :

Proposition 8.1.1 *[15] Dans un graphe G de degré maximum Δ et dans un routage de type commutation de circuit, sous les hypothèses $\Delta-$port et full-duplex, le nombre maximum de nœuds qui peuvent être informés d'un même message en k étapes est $(\Delta+1)^k$.*

Preuve. Considérons qu'un nœud possède un message à diffuser à tous les autres nœuds du graphe. Après la première étape de communication $\Delta+1$ nœuds auront eu l'information (Δ plus l'initiateur). Lors de la deuxième étape chaque nœud déjà informé peut envoyer son message à Δ autres nœuds. Au bout de la deuxième étape on aura $\Delta+1+\Delta(\Delta+1) = (\Delta+1)^2$ nœuds informés. En effectuant un raisonnement par récurrence on obtient le résultat énoncé dans la proposition. □

D'après cette proposition, on peut espérer trouver des algorithmes de diffusion Δ-port en commutation de circuit ou en *wormhole* nécessitant seulement $log_{\Delta+1}p$ étapes. Ce résultat est à opposer au nombre d'étapes (le diamètre) du mode *store and forward*, ce qui n'est pas négligeable sur les réseaux tels que les anneaux ou les grilles toriques. Le mode commutation de circuit ou *wormhole* permet de réduire le coût de *start-up*, ce qui est appréciable quand on connaît l'importance de ce paramètre par rapport aux autres (voir chapitre 2). C'est pourquoi il est important de construire des algorithmes nécessitant le plus petit nombre d'étapes possibles. En fait, il s'agit de trouver le meilleur compromis entre tous les paramètres de communication, car avoir un faible *start-up* peut entraîner une augmentation du coût de transmission préjudiciable pour des messages de grande taille.

8.2 La diffusion

Le problème de la diffusion Δ-port en mode commutation de circuit ou *wormhole* a été étudié par Peters et Syska dans [11]. L'algorithme de diffusion dans les grilles toriques se déduit de celui de la diffusion dans l'anneau.

Dans ce chapitre nous allons donner le principe des ces algorithmes seulement, pour obtenir la preuve de ces algorithmes se référer à [6, 11].

8.2.1 Diffusion dans l'anneau

Si l'on considère un anneau de p processeurs (avec $p = 3^k$), numérotés de 1 à p, nous allons voir qu'il est possible de réaliser la diffusion en un nombre minimal d'étapes.

L'étape i de l'algorithme consiste, pour les nœuds déjà informés, en un envoi aux deux processeurs situés à distance $\frac{p}{3^i}$. La figure 8.1 illustre cet algorithme dans le cas où $p = 27$ et le processeur numéro 1 est l'initiateur de la diffusion.

Chaque étape de cet algorithme coûte un temps de *start-up* (α), $\frac{p}{3^i}$ commutations de liens, et une transmission de message de taille L. Au total il y a $log_3 p$ étapes, donc cet algorithme a un coût total de : $\sum_{i=1}^{log_3} \left(\alpha + \frac{p}{3^i}\delta + L\tau\right)$ soit :

$$T_{diffusion} = \frac{1}{2}(p-1)\delta + (log_3 p)(\alpha + L\tau)$$

On constate que cet algorithme atteint le nombre optimum d'étapes et de commutations de circuit (le diamètre de l'anneau), mais que le coût de transmission est élevé comparé au temps des algorithmes point à point en *store and forward* (voir chapitre 5). Par contre le coût de la transmission

est plus élevée que dans les algorithmes point à point ($\frac{L}{2}\tau$) [12]. Donc, si le temps de commutation d'un circuit est faible, cet algorithme sera particulièrement efficace pour des messages de petite taille, par contre pour les messages de grande taille les algorithmes point à point permettent d'obtenir de meilleurs résultats ($\frac{L\tau}{2}$ [12] au lieu de $(log_3 p)(L\tau)$).

Remarque.
Il est possible d'adapter cet algorithme aux anneaux qui n'ont pas un nombre de processeurs multiple d'une puissance de trois.

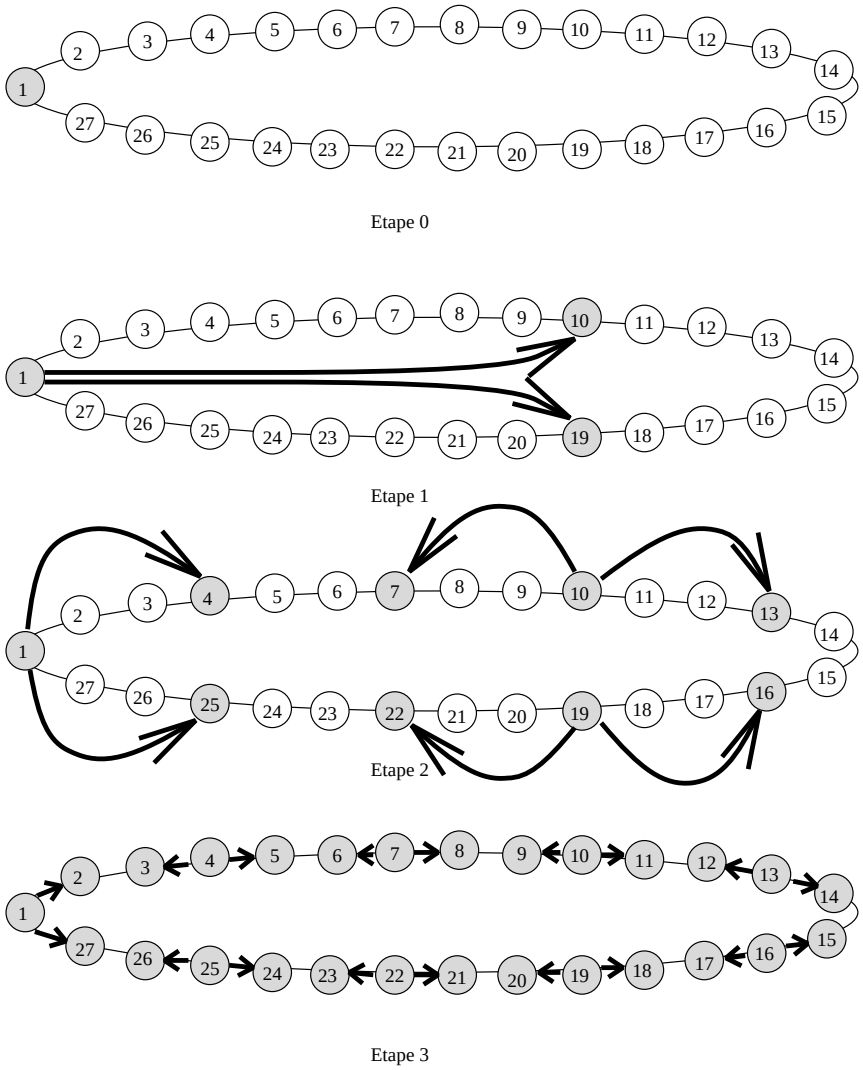


FIG. *8.1 : Diffusion wormhole dans un anneau à 27 processeurs*

8.2.2 Diffusion dans la grille torique

D'après la proposition 8.1.1, il est possible d'espérer trouver un algorithme de diffusion en $log_5 p$ étapes, pour cela considérons une grille torique de p processeurs (avec $p = 5^{2k}$).

Dans la grille torique 5×5, la solution est immédiate, et est donnée par la figure 8.2. Pour cette grille il est donc possible de réaliser la diffusion en deux étapes ($log_5 25$).

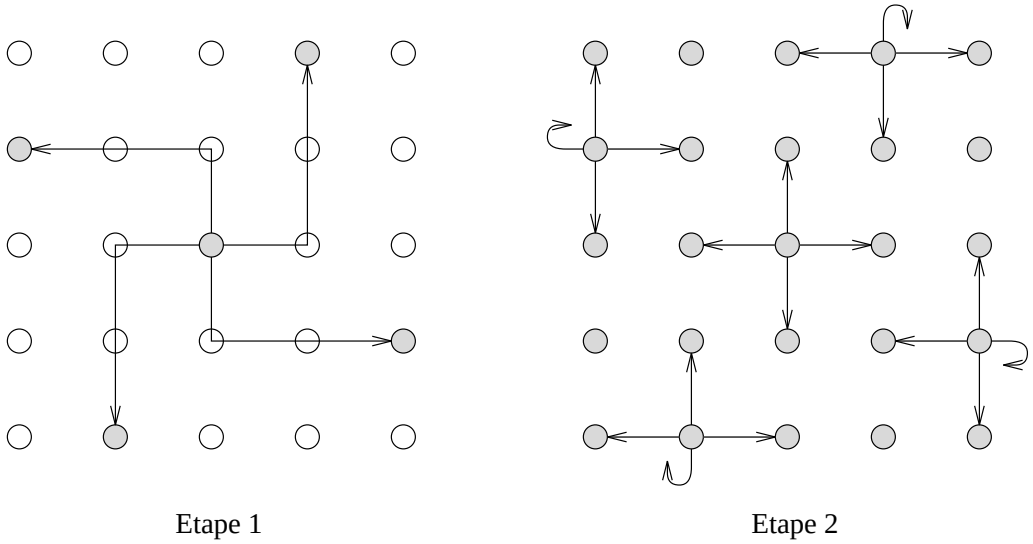


FIG. *8.2 : Diffusion wormhole dans une grille torique à 25 processeurs*

Cet algorithme peut être vu comme un pavage récursif de la grille torique [11, 14]. Pour la grille torique 5×5 il faut décomposer la grille en cinq croix, de cinq processeurs, comme le montre la figure 8.3. La croix centrale est constitué par l'initiateur de la diffusion (centre de la croix) et de ses quatre voisins. Les quatre autres croix sont construites sur le même modèle et assemblées de façon à paver la grille torique.

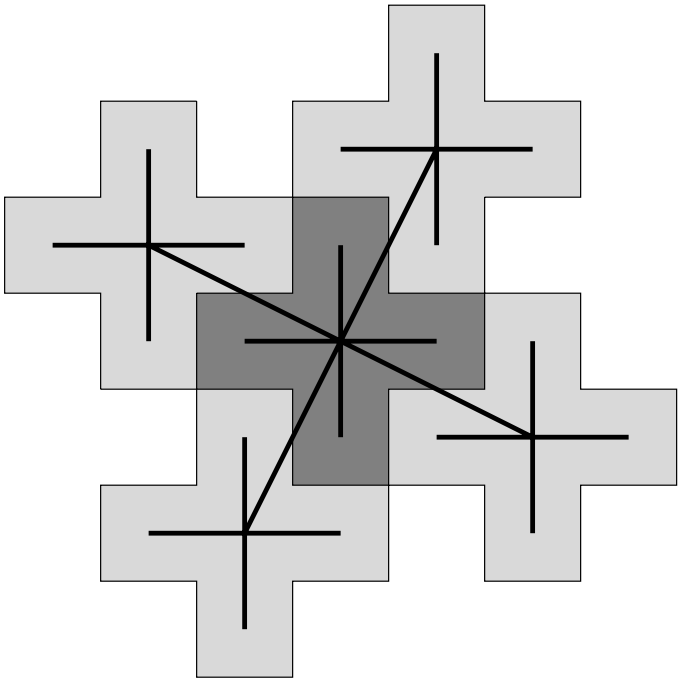


FIG. *8.3 : Pavage d'une grille torique à* 25 *processeurs*

En utilisant le pavage de la grille torique, les deux étapes de l'algorithme de diffusion correspondent donc à :

- étape 1 : l'initiateur de la diffusion envoie son message au centre des quatre autres croix ;
- étape 2 : chaque centre envoie le message à ses quatre voisins.

Il est possible de continuer le pavage en construisant des croix de tailles supérieures de façon récursive. En effet, pour l'étape suivante nous n'allons plus considérer des croix de cinq processeurs mais des croix de vingt cinq processeurs. En fait la figure 8.3 forme une nouvelle croix. En dupliquant quatre fois cette croix, on obtient une nouvelle croix de cent vingt cinq processeurs. Si l'on itère encore une fois ce procédé on obtient cinq croix de 125 processeurs qui réalisent le pavage de la grille torique 25×25.

En utilisant ce pavage Peters et Syska ont démontré la proposition suivante :

Proposition 8.2.1 *[11] La diffusion dans une grille torique* $5^k \times 5^k$ *(avec* $k \geq 1$*), nécessite un temps d'au plus :*

$$2k\alpha + (5^k - 1)\delta + 2kL\tau.$$

Cet algorithme permet de minimiser les termes en α et δ. Comme pour la diffusion dans l'anneau, cet algorithme a un coût de transmission élevé, ce qui le rend pénalisant pour les messages de grande taille. Pour ce type de message, les algorithmes point à point restent les meilleurs ($\frac{L\tau}{4}$ voir chapitre 4 au lieu de $(log_5 p)(L\tau)$).

8.3 L'échange total

L'échange total fait partie des schémas de communication présentés dans les chapitres 3 et 5. Chaque processeur possède un message qu'il envoie à tous les autres.

Après avoir présenté les bornes inférieures escomptées avec chacun des paramètres influençant le coût des communications, nous décrirons un algorithme d'échange total sur la grille torique directement inspiré des algorithmes point à point du chapitre 5. Puis nous présenterons une méthode basée sur l'algorithme de diffusion permettant de réaliser l'échange total.

8.3.1 Coût minimal suivant chaque paramètre

Soit un réseau de p processeurs, ayant un degré Δ, de diamètre D, possédant des liens parallèles *full-duplex*, les coûts minimaux pour chacun des paramètres de communication (α, δ, τ) pour réaliser un échange total en mode commutation de circuit sont obtenus de la même façon que ceux du mode *store and forward* (voir chapitre 5 et [6]).

- Le coût de *start-up* : comme l'échange total correspond à une diffusion simultanée depuis tous les processeurs, il est possible d'espérer le même coût de *start-up* que celui d'une diffusion (voir proposition 8.1.1) c'est-à-dire $\log_5(p)\alpha$.
- Le coût de commutation : la distance maximale entre deux processeurs du réseau étant le diamètre D, il faut donc au moins D commutations, donc le coût minimal est de $D\delta$.
- Le coût de transmission : chaque processeur doit recevoir $(p-1)$ messages de taille L, en supposant que tous les processeurs puissent recevoir simultanément sur leurs Δ liens, le temps de propagation minimal sera donc de $\lceil \frac{p-1}{\Delta} \rceil L\tau$.

8.3.2 Différents algorithmes

Suivant le type d'algorithme que l'on utilise, on va pouvoir minimiser le terme correspondant à tel ou tel paramètre. En effet comme pour la diffusion

les algorithmes point à point permettent d'obtenir le meilleur temps de transmission, alors l'algorithme basé sur le pavage de la grille torique permet de réduire le coût de *start-up*.

Une solution point à point

Dans le chapitre 5, nous avons présenté plusieurs algorithmes d'échange total point à point. Parmi ces algorithmes, le plus simple est sans nul doute celui qui utilise les dimensions successives de la grille torique. Cet algorithme se décompose en deux phases d'échange total sur des anneaux, une phase avec des messages de taille $\frac{L}{2}$, et une autre avec des messages de taille $\frac{L}{2}\sqrt{p}$. Cet algorithme peut directement s'adapter en mode commutation de circuit, en effet il suffit de faire des communications à distance un, ce qui permet de faire correspondre le terme $\alpha + \delta$ de la communication en commutation au terme β du *store and forward*. Le temps d'exécution de cet algorithme se déduit donc directement de celui en *store and forward* :

$$\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor (\alpha + \delta) + \left\lceil \frac{p-1}{4} \right\rceil L\tau \tag{8.2}$$

Cet algorithme permet d'atteindre le temps minimal de commutation de circuit et de transmission, mais possède un coût de *start-up* élevé.

Une solution regroupement-diffusion

L'algorithme de regroupement-diffusion est basé sur le pavage récursif de la grille torique présenté dans 8.2.2. Cet algorithme s'applique aux grilles toriques dont le nombre de processeurs est une puissance de cinq. Il se décompose en deux phases que nous allons détailler :

- le regroupement : en fait cette phase correspond à l'opération inverse de la diffusion, avec une taille de message augmentant à chaque étape. Lors de la première étape, toutes les croix formées de cinq processeurs envoient leur message au processeur central de la croix (figure 8.4). A l'étape deux, chaque centre envoie un message de taille $5L$ au centre de la croix de vingt cinq processeurs dont il fait partie (figure 8.4).

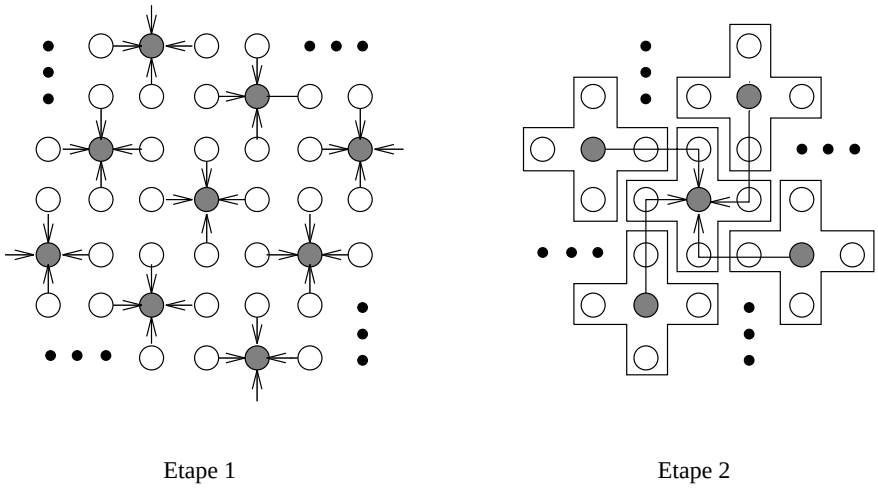


FIG. *8.4 : Les deux premières étapes du regoupement.*

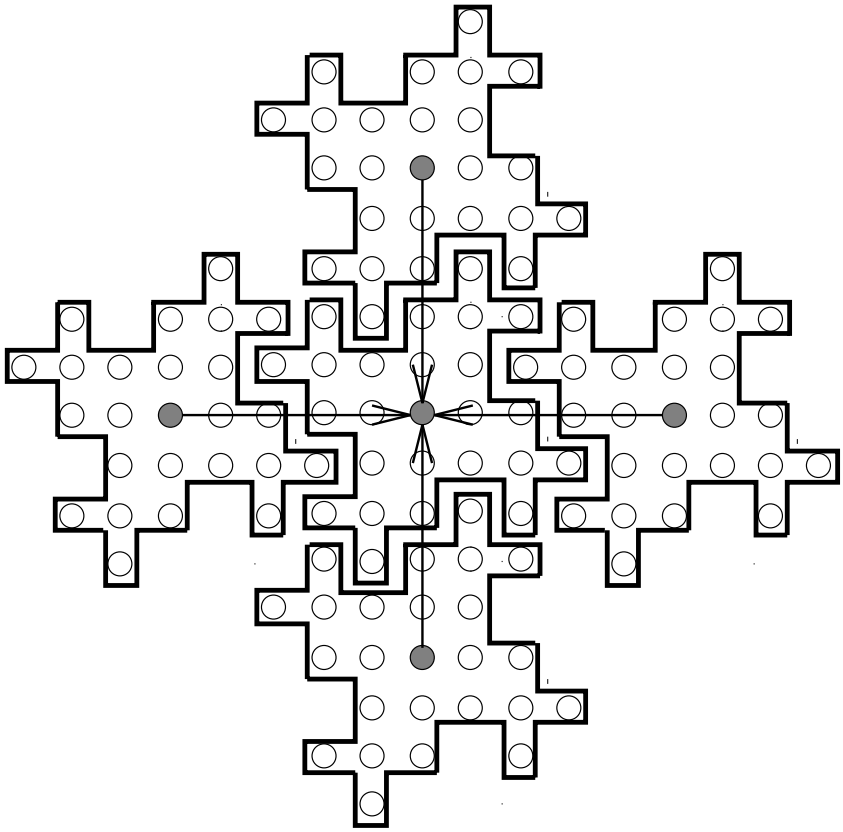


FIG. *8.5 : Etape 3 du regoupement.*

A l'étape 3 chaque centre de croix de l'étape deux envoie le message au centre de la croix de taille supérieure comme le montre la figure 8.5.

On itère ce procédé jusqu'à ce que les message soient tous centralisés sur un seul processeur.

- la diffusion : le processeur qui vient de recevoir les p messages effectue une diffusion avec un message de taille pL, selon le principe décrit précédemment.

Durant la première phase, à une étape i (i variant de 1 à $log_5 p$), il y a des communications de messages de taille 5^{i-1} entre les processeurs soit :

$$\alpha + (\sqrt{p} - 1)\delta + \sum_{i=1}^{\log_5(p)} (5^{i-1} L\tau).$$

Le coût de la seconde phase correspond à une diffusion d'un message de longueur pL soit :

$$\log_5(p)\alpha + (\sqrt{p} - 1)\delta + \log_5(p) pL\tau.$$

Le coût total de cet algorithme est donc :

$$2\log_5(p)\alpha + 2(\sqrt{p} - 1)\delta + (\frac{p}{4} + \log_5(p)p)L\tau. \tag{8.3}$$

Cet algorithme permet de diminuer le nombre d'étapes, c'est-à-dire le *start-up*, mais augmente considérablement le temps de transmission.

8.4 Comparaison des différents algorithmes

Lorsque l'on désire obtenir des performances sur une machine parallèle à mémoire distribuée, il est nécessaire d'optimiser les calculs ainsi que les communications. Comme nous l'avons vu dans le chapitre 4 pour les algorithmes de diffusion dans les grilles toriques en *store en forward*, le choix de l'algorithme est influencé par la taille du message que l'on diffuse.

Afin d'illustrer les écarts entre les différents algorithmes, nous allons les comparer sur une configuration donnée. Nous allons considérer une grille torique de 625 processeurs, avec des liens *full-duplex*, Δ-port, utilisant le mode de commutation de circuit. Nous prendrons $\alpha = 350\ \mu sec$, $\delta = 10\ \mu sec$ et $\tau = 0.2\ \mu sec/octect$ (ces valeurs correspondent aux paramètres de communication de l'iPSC860).

Cas de la diffusion

Nous allons comparer les algorithmes de diffusion pipeline (voir chapitre 4) et de Peters-Syska. Dans le cas de la diffusion pipeline, on effectue des communications point à point avec $\beta = \alpha + \delta$. Le tableau suivant donne la borne inférieure de chacun des paramètres pour la diffusion.

	α	δ	τ
Borne inférieure	$log_5(p)$	$2\left\lfloor\frac{\sqrt{p}}{2}\right\rfloor$	$\frac{L}{4}$
Peters-Syska	oui	oui	non
pipeline	non	oui	oui asymptotiquement

On constate que les deux algorithmes atteignent la borne inférieure en coût de commutation, et que la version pipeline, étant asymptotiquement optimale permettra d'obtenir de bons résultats avec des messages de grande taille.

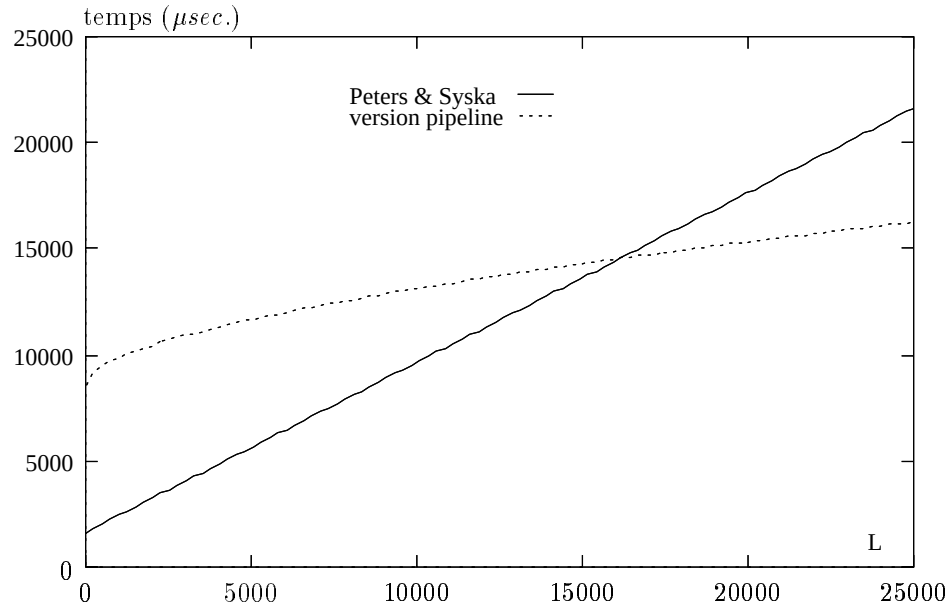


FIG. *8.6 : Comparaison des deux stratégies de diffusion.*

L'algorithme de Peters-Syska est meilleur jusqu'à des tailles de message de l'ordre de 15100 octets (soit 1888 réels double précision). En fait cet algorithme n'est pas seulement utilisable pour les petits messages, mais aussi pour les messages de taille moyenne.

Cas de l'échange total

Les deux algorithmes d'échange total présentés auparavant, atteignent les bornes inférieures pour certains paramètres, comme le montre le tableau ci-dessous.

	α	δ	τ
Borne inférieure	$log_5(p)$	$2\left\lceil\frac{\sqrt{p}}{2}\right\rceil$	$\left\lceil\frac{p-1}{4}\right\rceil L$
regroupement-diffusion	non	oui	non
point à point	non	oui	oui

Comme pour la diffusion, on peut constater que pour des messages de grande taille l'algorithme point à point sera meilleur, puisqu'il atteint la borne inférieure en coût de transmission.

La figure 8.7 permet de comparer les deux algorithmes d'échange total en commutation de circuit présentés précédemment.

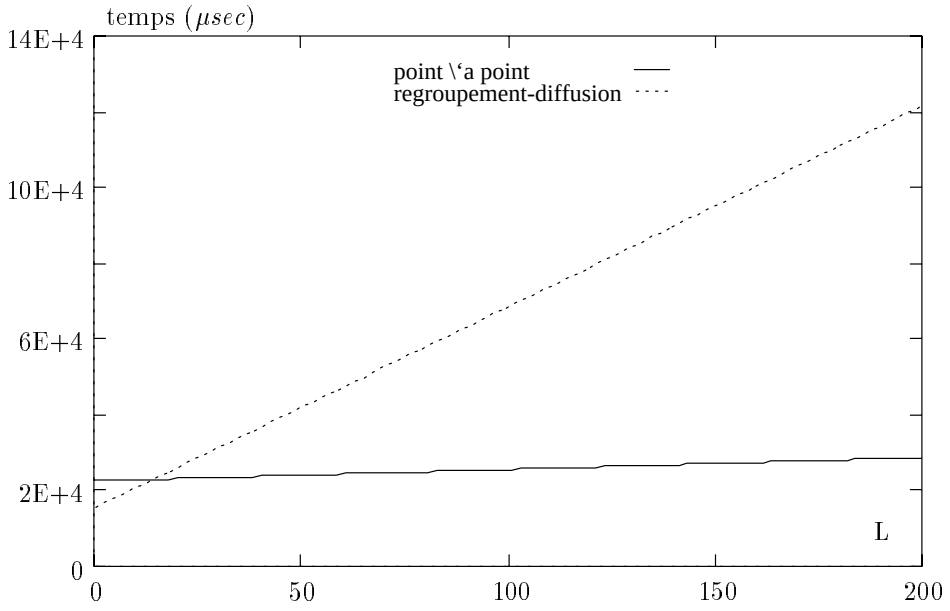


FIG. *8.7 : Comparaison des deux stratégies d'échange total.*

L'algorithme de regroupement-diffusion est meilleur pour des petites tailles de l'ordre de quelques octets. L'algorithme point à point est très rapidement meilleur du fait de son faible coût de transmission, de l'ordre de $\frac{p}{4}L\tau$ contre $p\ log_5(p)L\tau$ pour le regroupement-diffusion.

Ces deux exemples montrent bien que sur une topologie donnée, à p fixé, à paramètres de communication connus, le bon choix des algorithmes de communication est essentiel, le choix d'une stratégie regroupement-diffusion se révélant catastrophique pour de grands messages.

8.5 Conclusion

L'utilisation des communications en commutation de circuit se généralisant dans les machines à mémoire distribuée, il est nécessaire de connaître les meilleurs algorithmes de communications globales avec ce mode de communication afin d'obtenir de bonnes performances sur ces nouvelles machines.

Nous avons présenté des algorithmes qui diminuaient le nombre d'étapes de communications, c'est-à-dire le *start-up*, ce qui les rend particulièrement performants pour des messages de petite taille.

Il est difficile à l'utilisateur de choisir entre les différents algorithmes, la solution consisterait à inclure se choix dans le compilateur ce qui garantirait à l'utilisateur d'obtenir toujours les meilleurs performances de communication.

Bibliographie

[1] M. Barnett, R. Littlefield, D. Payne, and R. Van De Geijn. Global combine on mesh architectures with wormhole routing. Technical Report TR-93, Computer Science Department, University of Texas, (USA), 1993.

[2] M. Barnett, D. Payne, and R. Van De Geijn. Optimal broadcasting in mesh-connected architectures. Technical Report TR-91-38, Computer Science Department, University of Texas (USA), 1991.

[3] S. Bokhari. Communication overhead on the Intel iPSC/860 hypercube. Technical report, ICASE, NASA Langley Research Center, 1990.

[4] J. Choi, J. Dongarra, and D. Walker. Parallel matrix transpose algorithms on distributed memory concurrent computers. Technical report, Oak ridge National Laboratory (USA), October 1993.

[5] W.J. Dally and C.L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[6] Jean de Rumeur. *Communications dans les réseaux de processeurs.* Masson, 1994.

[7] P. Fraigniaud and J. Peters. Structured communications in torus networks. Technical report, School of Computing Science, Simon Fraser University (Canada), 1993.

[8] C-T. Ho. Matrix transpose on meshes with wormhole and XY routing. Technical report, IBM Research Division Almaden Reserch Center, 1993.

[9] X. Lin and L. Ni. Multicast communication in multicomputer networks. In *International Conference on Parallel Procesing*, August 1990.

[10] L. Ni and P. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, pages 62–76, February 1993.

[11] J. Peters and M. Syska. Circuit-switched broadcasting in torus network. Technical report, School of Computing Science, Simon Fraser University (Canada), 1993.

[12] Y. Saad and M.H. Schultz. Data communication in parallel architectures. *Journal of Parallel Computing*, 11:131–150, 1989.

[13] S. Seidel, M-H Lee, and S. Fotedar. Concurrent bidirectional communications on the Intel iPSC/860 and iPSC/2. Technical report, Computer Science Department, Michigan Technological University (USA), 1990.

[14] M. Senechal. Tiling the torus. *Disc. and Comp. Geometry*, pages 55–72, 1988.

[15] M. Syska. *Communications dans les architectures à mémoire distribuée*. PhD thesis, Université de Nice (France), December 1992.

Chapitre 9

Conclusion

Conclusions et perspectives

La parallélisation sur machines à mémoire distribuée d'applications numériques, fait appel à des mouvements de données qui impliquent tous les processeurs : ce type de mouvement de données est appelé communication globale. Avec l'évolution des machines, le rapport coût de communication sur coût de calcul devient de plus en plus grand, c'est pourquoi il est nécessaire de développer des algorithmes de communications efficaces.

Les études que nous avons présentées se situaient dans ce cadre-là et plus spécialement sur les machines parallèles possédant un réseau de processeurs connectés en grille torique. Nous avons développé une série d'outils théoriques, comme les arbres de recouvrement à arcs disjoints ou les chemins hamiltoniens, pour étudier et mettre en œuvre les principaux schémas de communications globales sur ces machines. En particulier nous avons présenté de nouvelles familles d'arbres de recouvrements à arcs disjoints de profondeur minimale, qui sont utilisables pour effectuer des diffusions optimales en mode pipeline. De nombreuses expérimentations réelles ont été développées sur ces schémas en mode commutation de messages pour valider les résultats théoriques.

L'identification de noyaux de calculs et de communications lors de la parallélisation d'applications numériques permet d'inclure facilement de telles procédures. Dans le cadre d'un groupe de travail qui réunit des utilisateurs et des informaticiens, nous avons commencé à développer des applications avec ces outils.

Les processeurs calculant de plus en plus vite, il devient nécessaire de développer des outils permettant d'effectuer du recouvrement calcul/communication, afin de réduire le temps d'inactivité des processeurs.

L'étude effectuée au début de ce document sur les machines parallèles montre que les nouvelles tendances des machines sont les communications en mode *wormhole* et des réseaux d'interconnexion multi-étages.

- En *wormhole* il existe peu de résultats, nous donnons une solution pour l'échange total dans la grille torique. L'étude des autres schémas de communication s'avère être une perspective intéressante. Il serait aussi souhaitable d'effectuer le même type d'analyse sur d'autres topologies.
- Dans les réseaux multi-étages se pose le problème de savoir comment réaliser des schémas de communications globales. Pour cela il va falloir étudier les mouvements de données structurés comme par exemple certains types de permutations.

Annexe A

A

Annexe

Tables des arcs entrants pour les grilles toriques $(4k+1) \times (4k+1)$

Pour ce cas on a : $x_{min} = y_{min} = -\lfloor \frac{\sqrt{p}}{2} \rfloor$; $x_{max} = y_{max} = \lfloor \frac{\sqrt{p}}{2} \rfloor$

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i \neq 0$	$0 \leq j < y_{max} - 2;\ j\ mod\ 2 = 0$
	$x_{min} + 1 < i < x_{max}; i \neq 0$	$j = y_{max} - 2$
	$i = x_{min}, x_{min} + 1, x_{max} - 1, x_{max}$	$j = y_{max} - 1$
	$x_{min} < i < x_{max}; i \neq 0; i\ mod\ 2 = 0$	$y_{min} < j < 0$
	$i = x_{min}$	$y_{min} + 1 < j < 0$
	$i = x_{max}$	$y_{min} + 2 < j < 0$
E_i	$x_{min} \leq i < 0$	$0 < j < y_{max}; j\ mod\ 2 = 1$
	$x_{min} + 2 \leq i < 0$	$j = y_{max} - 1$
	$i = x_{min}, x_{min} + 1$	$j = y_{max}$
	$i = x_{min}, x_{min} + 1$	$j = y_{min} + 1$
	$i = x_{max}$	$j = y_{min} + 2, y_{max} - 2$
	$i = x_{max} - 2$	$j = y_{min}$
	$0 < i < x_{max} - 2; i\ mod\ 2 = 1$	$y_{min} < j < 0$
	$i = x_{max} - 1$	$y_{min} + 2 < j < 0$
	$0 < i < x_{max} - 2; i\ mod\ 2 = 1$	$j = y_{max}$
S_i	$i = 0$	$j \neq 0$
	$i = x_{min}$	$j = y_{max} - 2$
	$i \neq x_{min} + 2, x_{max} - 2$	$j = y_{min}$
	$i = x_{max} - 1$	$j = y_{min} + 2$
	$x_{min} < i < x_{max}; i \neq ; i\ mod\ 2 = 0$	$j = y_{max}$
W_i	$0 < i \leq x_{max}$	$0 < j < y_{max} - 1; j\ mod\ 2 = 1$
	$0 < i < x_{max} - 1$	$j = y_{max} - 1$
	$i = x_{max} - 1, x_{max}$	$j = y_{max}$
	$i = x_{max} - 1, x_{max}$	$j = y_{min} + 1$
	$i = x_{min} + 2$	$j = y_{min}$
	$x_{min} + 2 < i < 0; i\ mod\ 2 = -1$	$j = y_{max}$
	$i = x_{min} + 1$	$j = y_{min} - 2$
	$x_{min} + 1 < i < 0; i\ mod\ 2 = -1$	$y_{min} < j < 0$
	$i = x_{min} + 1$	$y_{min} + 1 < j < 0$

Table de l'arbre NST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i = x_{max} - 2$	$j = y_{max} - 1$
	$i = x_{min} + 1$	$j = y_{min}, y_{min} + 1$
	$i = x_{min}$	$j = y_{max} - 2$
	$i = x_{max}$	$0 < j < y_{max} - 2; j \ mod \ 2 = 1$
	$0 < i < x_{max} - 1; i \ mod \ 2 = 1$	$y_{min} \leq j < 0$
	$i = x_{max} - 1$	$y_{min} + 1 < j < 0$
	$x_{min} < i < 0$	$0 < j < y_{max} - 1; j \ mod \ 2 = 1$
	$x_{min} + 1 < i < 0$	$j = y_{max} - 1$
	$i = x_{max}$	$j = y_{min}, y_{min} + 1$
E_i	$x_{min} < i < 0$	$y_{min} < j < y_{max}; j \neq 0; j \ mod \ 2 = 0$
	$x_{min} + 1 < i < 0$	$j = y_{max}$
	$x_{min} + 2 < i < 0$	$j = y_{min}$
	$0 \leq i < x_{max} - 2; i \ mod \ 2 = 0$	$j \neq 0$
	$i = x_{max} - 2$	$y_{min} < j < y_{max} - 1; j \neq 0$
	$i = x_{max} - 1$	$j = y_{min}, y_{min} + 1, y_{max} - 1, y_{max}$
S_i	$0 < i < x_{max} - 1; i \ mod \ 2 = 1$	$0 < j \leq y_{max}$
	$i = x_{max} - 1$	$0 < j < y_{max} - 1$
	$i = x_{max}$	$j = y_{max} - 1, y_{max}$
	$i = x_{max}$	$y_{min} - 2 < j < 0; j \ mod \ 2 = -1$
	$i = x_{min} + 2, x_{max} - 2$	$j = y_{min}$
	$i = x_{min}$	$j = y_{min} + 2$
	$i = x_{min} + 1$	$j = y_{max} - 1, y_{max}$
	$x_{min} < i < 0$	$y_{min} + 1 < j < 0; j \ mod \ 2 = -1$
	$x_{min} + 2 < i < 0$	$j = y_{min} + 1$
W_i	$i \neq 0$	$j = 0$
	$i = x_{max} - 2$	$j = y_{max}$
	$i = x_{max}$	$y_{min} < j < y_{max}; j \neq 0; j \ mod \ 2 = 0$
	$i = x_{min} + 2$	$j = y_{min} + 1$
	$i = x_{min}$	$j \neq y_{min} + 2, y_{max} - 2$

Table de l'arbre EST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i = 0$	$j \neq 0$
	$i \neq x_{min} + 2, x_{max} - 2$	$j = y_{max}$
	$i = x_{max}$	$j = y_{min} + 2$
	$i = x_{min} + 1$	$j = y_{max} - 2$
	$x_{min} < i < x_{max}; i \neq 0; i \ mod \ 2 = 0$	$j = y_{min}$
E_i	$x_{min} \leq i < 0$	$y_{min} + 1 < j < 0; j \ mod \ 2 = -1$
	$x_{min} + 1 < i < 0$	$j = y_{min} + 1$
	$i = x_{min}, x_{min} + 1$	$j = y_{min}$
	$i = x_{min}, x_{min} + 1$	$j = y_{max} - 1$
	$i = x_{max} - 2$	$j = y_{max}$
	$i = x_{max} - 1$	$j = y_{min} + 2$
	$0 < i < x_{max} - 1; i \ mod \ 2 = 1$	$0 < j < y_{max} - 1$
	$i = x_{max} - 1$	$0 < j < y_{max} - 1$
	$0 < i < x_{max} - 2; i \ mod \ 2 = 1$	$j = y_{min}$
S_i	$i \neq 0$	$y_{min} - 2 < j \leq 0; j \ mod \ 2 = 0$
	$x_{min} + 1 < i < x_{max}; i \neq 0$	$j = y_{min} + 2$
	$i = x_{min}, x_{min} + 1, x_{max} - 1, x_{max}$	$j = y_{min} + 1$
	$x_{min} < i < x_{max}; i \neq 0; i \ mod \ 2 = 0$	$0 < j < y_{max}$
	$i = x_{min}$	$0 < j < y_{max} - 2$
	$i = x_{max}$	$0 < j < y_{max} - 1$
W_i	$0 < i \leq x_{max}$	$y_{min} < j < 0; j \ mod \ 2 = -1$
	$0 < i < x_{max} - 1$	$j = ymin + 1$
	$i = x_{max} - 1, x_{max}$	$j = y_{min}$
	$i = x_{max} - 1, x_{max}$	$j = y_{max} - 1$
	$x_{min} + 2 < i < 0; i \ mod \ 2 = -1$	$j = y_{min}$
	$i = x_{min}$	$j = y_{min} + 2, y_{max} - 2$
	$i = x_{min} + 2$	$j = y_{max}$
	$x_{min} + 1 < i < 0; i \ mod \ 2 = -1$	$0 < j < y_{max}$
	$i = x_{min} + 1$	$0 < j < y_{max} - 2$

Table de l'arbre SST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$x_{min}+1<i<0; i \ mod \ 2=-1$	$y_{min} \leq j<0$
	$i=x_{min}+1$	$y_{min}+1<j<0$
	$i=x_{min}$	$j=y_{min}, y_{min}+1$
	$i=x_{max}-1$	$j=y_{min}, y_{min}+1$
	$i=x_{min}$	$0<j<y_{max}-2; j \ mod \ 2=1$
	$0<i<x_{max}$	$0<j<y_{max}-1; j \ mod \ 2=1$
	$0<i<x_{max}-2$	$j=y_{max}-1$
	$i=x_{max}$	$j=y_{max}-1$
	$i=x_{min}+2, x_{max}-2$	$j=y_{max}$
E_i	$i \neq 0$	$j=0$
	$i=x_{max}$	$j \neq y_{min}+2, y_{max}-2$
	$i=x_{max}-2$	$j=y_{max}-1$
	$i=x_{min}+2$	$j=y_{min}$
	$i=x_{min}$	$y_{min}<j<y_{max}; j \neq 0 : j \ mod \ 2=0$
S_i	$x_{min}+1<i<0; i \ mod \ 2=-1$	$0<j \leq y_{max}$
	$i=x_{min}+1$	$0<j<y_{max}-1$
	$i=x_{min}$	$j=y_{max}-1, y_{max}$
	$i=x_{max}-1$	$j=y_{max}-1, y_{max}$
	$i=x_{max}$	$j=y_{min}+2$
	$i=x_{min}+2$	$j=y_{min}+1$
	$i=x_{min}$	$y_{min}+2<j<0; j \ mod \ 2=-1$
	$0<i<x_{max}$	$y_{min}+1<j<0; j \ mod \ 2=-1$
	$0<i<x_{max}-1$	$j=y_{min}+1$
W_i	$x_{min}+2<i \leq 0; i \ mod \ 2=0$	$j \neq 0$
	$i=x_{min}+2$	$y_{min}+1<j<y_{max}; j \neq 0$
	$i=x_{min}+1$	$j=y_{min}, y_{min}+1, y_{max}-1, y_{max}$
	$0<i<x_{max}$	$y_{min}<j<y_{max}; j \neq 0; j \ mod \ 2=0$
	$0<i<x_{max}-2$	$j=y_{max}$
	$0<i<x_{max}-1$	$j=y_{min}$

Table de l'arbre WST

Table des arcs entrants pour les grilles toriques $(4k+3) \times (4k+3)$

Pour ce cas on a : $x_{min} = y_{min} = -\lfloor \frac{\sqrt{p}}{2} \rfloor$; $x_{max} = y_{max} = \lfloor \frac{\sqrt{p}}{2} \rfloor$.

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i \neq 0$ $i \neq 0; i \ mod\ 2 = 0$	$\leq j < y_{max}; j \ mod\ 2 = 0$ $y_{min} < j < 0$
E_i	$x_{min} \leq i < 0$ $0 < i < x_{max}; i \ mod\ 2 = 1$ $i = x_{max}$ $i = x_{min}$	$0 < j \leq y_{max}; j \ mod\ 2 = 1$ $y_{min} \leq j < 0$ $y_{min} < j < y_{max}; \lvert j \rvert \ mod\ 2 = 1$ $y_{min} < j < 0$
S_i	$i = 0$ $i = x_{min}, x_{max}$ $i \neq 0; i \ mod\ 2 = 0$	$j \neq 0$ $j = y_{min}$ $j = y_{min}$
W_i	$0 < i < x_{max}$ $x_{min} < i < 0; i \ mod\ 2 = -1$ $i = x_{max}$ $i = x_{max}$	$0 < j < y_{max}; j \ mod\ 2 = 1$ $y_{min} \leq j < 0$ $y_{min} < j < 0; j \ mod\ 2 = 0$ $j = y_{max}$

Table de l'arbre NST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$0 \leq i < x_{max}; i \ mod\ 2 = 1$ $i = x_{max}$ $x_{min} \leq i < 0$ $x_{min} < i < 0; i \ mod\ 2 = 0$	$y_{min} < j < 0$ $j = y_{min}$ $0 < j < y_{max}; j \ mod\ 2 = 1$ $j = y_{min}$
E_i	$0 \leq i < x_{max}; i \ mod\ 2 = 0$ $x_{min} < i < 0$	$j \neq 0$ $j \neq 0; j \ mod\ 2 = 0$
S_i	$0 < i \leq x_{max}; i \ mod\ 2 = 1$ $x_{min} \leq i < 0$ $x_{min} < i < 0$ $x_{min} < i < y_{max}; \lvert i \rvert \ mod\ 2 = 1$	$0 < j < y_{max}$ $y_{min} < j < 0; j \ mod\ 2 = -1$ $j = y_{max}$ $j = y_{min}$
W_i	$i \neq 0$ $i = x_{min}$ $i = x_{min}$	$j = 0$ $j = y_{min}, y_{max}$ $j \neq 0; j \ mod\ 2 = 0$

Table de l'arbre EST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i = 0$ $i = x_{min}, x_{max}$ $i \neq 0; i \ mod \ 2 = 0$	$j \neq 0$ $j = y_{max}$ $j = y_{max}$
E_i	$x_{min} < i < 0$ $i = x_{min}$ $i = x_{min}$ $0 < i < x_{max}; i \ mod \ 2 = 1$	$y_{min} \leq j < 0; j \ mod \ 2 = -1$ $0 < j < y_{max}; j \ mod \ 2 = 0$ $j = y_{min}$ $0 \leq j < y_{max}$
S_i	$i \neq 0$ $i \neq 0; i \ mod \ 2 = 0$	$y_{min} < j \leq 0; j \ mod \ 2 = 0$ $0 < j < y_{max}$
W_i	$0 < i \leq x_{max}$ $x_{min} < i < 0; i \ mod \ 2 = -1$ $i = x_{max}$ $i = x_{min}$	$y_{min} \leq j < 0; j \ mod \ 2 = -1$ $0 < j \leq y_{max}$ $0 < j < y_{max}$ $y_{min} < j < y_{max}; \lvert j \rvert \ mod \ 2 = 1$

Table de l'arbre SST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$x_{min} \leq i < 0; i \ mod \ 2 = -1$ $0 < i \leq x_{max}$ $0 < i < x_{max}$ $x_{min} < i < x_{max}; \lvert i \rvert \ mod \ 2 = 1$	$y_{min} \leq j < 0$ $0 < j < y_{max}; j \ mod \ 2 = 1$ $j = y_{min}$ $j = y_{max}$
E_i	$i \neq 0$ $i = x_{max}$ $i = x_{max}$	$j = 0$ $j = y_{min}, y_{max}$ $j \neq 0; j \ mod \ 2 = 0$
S_i	$x_{min} \leq i < 0; i \ mod \ 2 = -1$ $i = x_{min}$ $0 < i \leq x_{max}$ $0 < i \leq x_{max}; i \ mod \ 2 = 0$	$0 < j < y_{max}$ $j = y_{max}$ $y_{min} < j < 0; j \ mod \ 2 = -1$ $j = y_{max}$
W_i	$x_{min} < i \leq 0; i \ mod \ 2 = 0$ $0 < i < x_{max}$	$j \neq 0$ $j \neq 0; j \ mod \ 2 = 0$

Table de l'arbre WST

Table des arcs entrants pour les grilles toriques $(4k) \times (4k)$

Pour ce cas on a :
$x_{min} = -\lfloor \frac{\sqrt{p}}{2} \rfloor + 1$; $y_{min} = -\lfloor \frac{\sqrt{p}}{2} \rfloor$; $x_{max} = \lfloor \frac{\sqrt{p}}{2} \rfloor$; $y_{max} = \lfloor \frac{\sqrt{p}}{2} \rfloor - 1$.

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i \neq 0; i \neq x_{max}; i \ mod\ 2 = 0$ $i \neq 0$ $i \neq 0; i \ mod\ 2 = 0$	$j = y_{min} + 1$ $0 \leq j < y_{max}; j \ mod\ 2 = 0$ $y_{min} + 1 < j < 0$
E_i	$x_{min} \leq i < 0$ $i = x_{min}$ $0 < i < x_{max} - 1; i \ mod\ 2 = 1$ $i = x_{max} - 1$	$0 < j \leq y_{max}; j \ mod\ 2 = 1$ $y_{min} < j < 0; j \ mod\ 2 = 0$ $y_{min} < j < 0$ $y_{min} + 1 < j < 0$
S_i	$i = 0$ $i = x_{min}$ $i \neq 0$ $i = x_{min}, x_{max} - 1, x_{max}$	$j \neq 0$ $y_{min} < j < 0; j \ mod\ 2 = -1$ $j = y_{min}$ $j = y_{min} + 1$
W_i	$0 < i \leq x_{max}$ $x_{min} < i < 0; i \ mod\ 2 = -1$	$0 < j \leq y_{max}; j \ mod\ 2 = 1$ $y_{min} < j < 0$

Table de l'arbre NST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$0 < i < x_{max}; i \ mod\ 2 = 1$ $i = x_{max}$ $i = x_{max}$ $x_{min} < i < 0$	$y_{min} \leq j < 0$ $j = y_{min}, y_{min} + 1$ $0 < j < y_{max}; j \ mod\ 2 = 1$ $0 < j < y_{max}; j \ mod\ 2 = 1$
E_i	$0 \leq i < x_{max}; i \ mod\ 2 = 0$ $x_{min} < i < 0$	$y_{min} \leq j \leq y_{max}; j \neq 0$ $y_{min} \leq j < y_{max}; j \neq 0; j \ mod\ 2 = 0$
S_i	$0 < i < x_{max}; i \ mod\ 2 = 1$ $i = x_{max}$ $i = x_{max}$ $x_{min} < i < 0$ $x_{min} < i < 0; i \ mod\ 2 = 0$	$0 < j \leq y_{max}$ $j = y_{max}$ $y_{min} + 1 < j < 0; j \ mod\ 2 = -1$ $y_{min} < j < 0; j \ mod\ 2 = -1$ $j = y_{max}$
W_i	$i \neq 0$ $x_{min} < i < 0; i \ mod\ 2 = -1$ $i = x_{max}$ $i = x_{min}$	$j = 0$ $j = y_{min}$ $y_{min} + 1 < j < y_{max} - 1; j \neq 0; j \ mod\ 2 = 0$ $j \neq 0$

Table de l'arbre EST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i = 0$ $i = x_{min}$ $i \neq 0$ $x_{min} < i < x_{max}; i \neq 0; i \ mod \ 2 = 0$ $i \neq 0$	$j \neq 0$ $0 < j < y_{max}; j \ mod \ 2 = 0$ $j = y_{max}$ $j = y_{min}$ $j = y_{max}$
E_i	$x_{min} \leq i < 0$ $i = x_{min}$ $0 < i < x_{max}; i \ mod \ 2 = 1$ $i = x_{min}$ $0 < i < x_{max} - 1; i \ mod \ 2 = 1$	$y_{min} < j < 0; j \ mod \ 2 = -1$ $0 < j < y_{max}; j \ mod \ 2 = 0$ $0 < j < y_{max}$ $j = y_{min}$ $j = y_{min}$
S_i	$x_{min} \leq i \leq x_{max}; i \neq 0$ $x_{min} \leq i < x_{max}; i \neq 0; i \ mod \ 2 = 0$	$y_{min} < j \leq 0; j \ mod \ 2 = 0$ $0 < j < y_{max}$
W_i	$0 < i \leq x_{max}$ $i = x_{max} - 1, x_{max}$ $x_{min} < i < 0; i \ mod \ 2 = -1$ $x_{min} < i < 0; i \ mod \ 2 = -1$	$y_{min} < j < 0; j \ mod \ 2 = -1$ $j = y_{min}$ $j = y_{min}$ $0 < j < y_{max}$

Table de l'arbre SST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$0 < i < x_{max}$ $x_{min} \leq i < 0; i \ mod \ 2 = -1$	$0 < j < y_{max}; j \ mod \ 2 = 1$ $y_{min} \leq j < 0$
E_i	$i \neq 0$ $i = x_{max}$ $i = x_{max} - 1$ $0 < i < x_{max}; i \ mod \ 2 = 1$	$j = 0$ $j \neq 0$ $j = y_{min}, y_{max} - 1, y_{max}$ $j = y_{max}$
S_i	$0 < i < x_{max}$ $0 < i < x_{max} - 1$ $0 < i < x_{max} - 1; i \ mod \ 2 = 0$ $x_{min} \leq i < 0; i \ mod \ 2 = -1$	$y_{min} + 1 < j < 0; j \ mod \ 2 = -1$ $j = y_{min} + 1$ $j = y_{max}$ $0 < j \leq y_{max}$
W_i	$x_{min} < i \leq 0; i \ mod \ 2 = 0$ $0 < i < x_{max}$ $0 < i < x_{max} - 1$	$j \neq 0$ $y_{min} + 1 < j < y_{max}; j \neq 0; j \ mod \ 2 = 0$ $j = y_{min}$

Table de l'arbre WST

Table des arcs entrants pour les grilles toriques $(4k+2) \times (4k+2)$

Pour ce cas on a :
$x_{min} = -\lfloor \frac{\sqrt{p}}{2} \rfloor + 1$; $y_{min} = -\lfloor \frac{\sqrt{p}}{2} \rfloor$; $x_{max} = \lfloor \frac{\sqrt{p}}{2} \rfloor$; $y_{max} = \lfloor \frac{\sqrt{p}}{2} \rfloor - 1$.

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i \neq 0$ $i \neq 0; i \ mod\ 2 = 0$	$0 \leq j < 0; j \ mod\ 2 = 0$ $y_{min} < j < 0$
E_i	$x_{min} \leq i < 0$ $0 < i < x_{max}; i \ mod\ 2 = 1$ $0 < i < x_{max}; i \ mod\ 2 = 1$ $i = x_{max} - 2$ $i = x_{max}$	$0 < j < y_{max}; j \ mod\ 2 = 1$ $j = y_{max}$ $y_{min} < j < 0$ $j = y_{min}$ $j = y_{min} + 2$
S_i	$i = 0$ $i = x_{max}$ $i \neq 0; i \neq x_{min} + 1; i \neq x_{max} - 2$ $i \neq 0; i \ mod\ 2 = 0$	$j \neq 0$ $y_{min} + 2 < j < 0; j \ mod\ 2 = -1$ $j = y_{min}$ $j = y_{max}$
W_i	$0 < i \leq x_{max}$ $x_{min} < i < 0; i \ mod\ 2 = -1$ $x_{min} < i < 0; i \ mod\ 2 = -1$ $i = x_{max}$ $i = x_{min} + 1$ x_{max}	$0 < j < y_{max}; j \ mod\ 2 = 1$ $y_{min} < j < 0$ $j = y_{max}$ $j = y_{max}$ $j = y_{min}$ $y_{min} < j < 0; j \ mod\ 2 = 0$

Table de l'arbre NST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$0 \leq i < x_{max}; i \ mod\ 2 = 0$ $x_{min} < i < 0$ $x_{min} < i < 0; i \ mod\ 2 = 0$	$j \neq 0$ $0 < j < y_{max}; j \ mod\ 2 = 1$ $j = y_{min}$
E_i	$0 \leq i < x_{max}; i \ mod\ 2 = 0$ $x_{min} < i < 0$	$j \neq 0$ $j \neq 0; j \ mod\ 2 = 0$
S_i	$0 < i \leq x_{max}; i \ mod\ 2 = 1$ $x_{min} < i < 0$ $i = x_{min} + 1$	$0 < j \leq y_{max}$ $y_{min} < j < 0; j \ mod\ 2 = -1$ $j = y_{min}$
W_i	$i \neq 0$ $i = x_{min}$ $x_{min} < i < 0; i \ mod\ 2 = -1$	$j = 0$ $j \neq 0$ $0 < j \leq y_{max}$

Table de l'arbre EST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$i = 0$ $i \neq 0$ $i = x_{max}$	$j \neq 0$ $j = y_{min}$ $0 < j < y_{max}; j \ mod \ 2 = 1$
E_i	$x_{min} \leq i < 0$ $0 < i < x_{max}; j \ mod \ 2 = 1$ $i = x_{max}$	$x_{min} \leq j < 0; j \ mod \ 2 = -1$ $0 < j < y_{max}$ $j = y_{max} - 1$
S_i	$i \neq 0$ $i \neq 0; i \ mod \ 2 = 0$	$y_{min} < j \neq 0; j \ mod \ 2 = 0$ $0 < j < y_{max}$
W_i	$0 < i \leq x_{max}$ $x_{min} < i < 0; i \ mod \ 2 = -1$ $i = x_{max}$	$y_{min} \leq j < 0; j \ mod \ 2 = -1$ $0 < j < y_{max}$ $0 < j < y_{max}; j \ mod \ 2 = 0$

Table de l'arbre SST

arc entrant par	contrainte sur i	contrainte sur j
N_i	$x_{min} < i < 0; i \ mod \ 2 = -1$ $i = x_{min}$ $i = x_{min}$ $0 < i < x_{max}$ $i = x_{max}$ $0 < i < x_{max}; i \ mod \ 2 = 0$	$y_{min} \leq j < 0$ $j = y_{min}$ $0 < j < y_{max} : j \ mod \ 2 = 1$ $0 < j < y_{max} : j \ mod \ 2 = 1$ $j = y_{max} - 1$ $j = y_{min}$
E_i	$i \neq 0$ $i = x_{min}$ $i = x_{max}$ $0 < i < x_{max} - 2; i \ mod \ 2 = 1$	$j = 0$ $j \neq 0; j \ mod \ 2 = 0$ $j \neq 0; j \neq y_{min} + 2; j \neq y_{max} - 1$ $j = y_{min}$
S_i	$x_{min} < i < 0; i \ mod \ 2 = -1$ $i = x_{min}$ $i = x_{max}$ $0 < i < x_{max}$ $i = x_{max} - 2$	$0 < j \leq y_{max}$ $y_{min} < j < 0; j \ mod \ 2 = -1$ $j = y_{min} + 2$ $y_{min} < j < 0; j \ mod \ 2 = -1$ $j = y_{min}$
W_i	$x_{min} < i \leq 0; i \ mod \ 2 = -1$ $0 < i < x_{max}$	$j \neq 0$ $j \neq 0; j \ mod \ 2 = 0$

Table de l'arbre WST