Université de Nice – Sophia Antipolis
Ecole doctorale Sciences Pour l'Ingénieur
Centre National de la Recherche Scientifique
Laboratoire I3S

Thèse de doctorat

Spécialité : informatique

Communications dans les architectures à mémoire distribuée

Michel Syska

Thèse soutenue le mardi 15 décembre 1992

Jury :
Jean-Claude Bermond : directeur
Ioan Bond : président
Afonso Ferreira : examinateur
Philippe Mussi : examinateur
Joseph Peters : rapporteur, invité
Denis Trystram : rapporteur

Je tiens à remercier ici les membres du jury :

mon directeur, *Jean-Claude Bermond*, qui m'a donné tous les moyens de la recherche,

Ioan Bond, qui a accepté de présider ce jury, et a souvent discuté longuement avec moi,

Afonso Ferreira, qui m'a conseillé en de nombreuses occasions (de la Bretagne au Canada),

Philippe Mussi, à qui je dois certainement mon inscription en thèse et bien d'autres choses (comme la rencontre avec les transputers),

Joseph Peters et *Denis Trystram*, qui ont accepté de rapporter sur cette thèse dans des délais très réduits. Tous deux ont apporté beaucoup de rigueur à ce document.

Je tiens également à remercier *Dominique Sotteau*, *Marie-Claude Heydemann* et *Pierre Fraigniaud* pour leur nombreuses remarques, ainsi que tous les membres de RUMEUR.

Enfin, je remercie *Francois Baccelli* pour la confiance et l'aide qu'il m'a donné depuis mon stage de DEA, ainsi que *Zhen Liu* avec qui cela a été un plaisir de travailler.

Table des matières

Liste des figures

Introduction

Communications dans les architectures à mémoire distribuée

Cette thèse porte sur l'étude des communications dans les machines parallèles à mémoire distribuée. En particulier on s'intéresse au réseau d'interconnexion des processeurs et aux algorithmes d'échange de données sur ce réseau.

Aujourd'hui, les ordinateurs parallèles permettent d'approcher des puissances de traitement de l'ordre du Tera Flops sur des Tera octets de données (Tera = mille milliards). De telles performances sont atteintes par des algorithmes parallèles exécutés sur des machines dites massivement parallèles, constituées de plusieurs milliers de nœuds. Chaque nœud est constitué d'un microprocesseur et de sa mémoire locale, la taille du système ne permettant pas d'implanter une mémoire partagée, globale à tous les processeurs. Un processeur peut donc traiter de manière indépendante les données stockées localement; les données qui se trouvent sur d'autres nœuds peuvent être communiquées par envoi de messages à travers le réseau. Ces messages sont alors acheminés à travers le réseau d'interconnexion entre le nœud du processeur qui stockait la donnée et celui qui la demandait. Les algorithmes parallèles peuvent donc être décrits comme des successions de phases de calcul et de phases de communication, qui permettent aux processeurs d'échanger des données. Le plus souvent ce sont ces phases de communication qui ralentissent le traitement, et les puissances de crête annoncées plus haut ne sont atteintes que dans le cas d'algorithmes qui ne comportent en fait que très peu de communications, ou dans lesquels les nœuds communiquent uniquement avec leurs voisins. Pour les autres applications, on est bien loin du Tera Flops, et il faut chercher à minimiser ces temps de communication.

Le temps nécessaire à un échange de données entre les processeurs est principalement lié à l'architecture (topologie) du réseau d'interconnexion, et aux protocoles (algorithmes) utilisés pour réaliser cet échange. Les travaux présentés dans cette thèse portent sur l'étude de ces deux facteurs.

Le premier chapitre est une synthèse sur les machines parallèles actuelles et a été écrit en collaboration avec Philippe Michallon. Une première version est parue dans le document distribué à l'école d'été RUMEUR (août 1992), et sera publié dans [4].

Ce chapitre permet de dégager les orientations en matière d'architectures parallèles, et souligne l'importance du réseau d'interconnexion entre les processeurs vis à vis des performances globales du système. On y montre en particulier les choix topologiques des constructeurs, qui sont le plus souvent guidés par les contraintes technologiques des circuits VLSI, ainsi que par des contraintes de fabrication et de coût. Ces dernières ont favorisé l'utilisation de réseaux en grilles (ou hypercubes), bien qu'il existe de meilleurs candidats au vu de paramètres critiques de ces réseaux, comme le diamètre ou la bissection. Ainsi les réseaux de *de Bruijn* semblent mieux convenir à l'assemblage de milliers de processeurs que les réseaux en grilles ou en hypercubes. Enfin, les derniers projets des grands constructeurs de machines parallèles comme *Intel* ou *Thinking Machine Corp.* montrent que les machines à mémoire distribuée du type MIMD sont le choix architectural d'avenir. Le second point souligné dans ce chapitre est l'émergence d'un nouveau type de commutation pour le routage des messages : le routage *wormhole* qui est utilisé sur les machines les plus récentes, suppléant le classique *store-and-forward*.

Le deuxième chapitre est consacré à l'étude de protocoles de communication sur ces réseaux, avec le modèle *wormhole*. Dans le routage *wormhole*, un message est décomposé en *flits* (acronyme pour "flow control unit"), et le premier flit, nommé entête, contient l'adresse de destination du message. Un flit est de la taille de la queue d'un canal. Au cours du processus de routage, l'entête progresse de nœud en nœud vers la destination, en positionnant convenablement les routeurs des nœuds intermédiaires, le reste du message suivant à la queue leu leu, à travers les queues des canaux empruntés par le chemin. Une fois l'entête arrivée à destination, tout le message est lu en un temps égal à $L\tau$, où L est la longueur du message et τ le débit d'un lien (inverse de sa bande passante). Dans le mode *store-and-forward* le message était lu par chaque nœud intermédiaire avant d'être re-routé.

Après une brève synthèse des divers problèmes de communication étudiés dans la littérature, et des modèles associés, on étudie en détail deux points : l'interblocage et la diffusion en mode *wormhole*.

La section qui porte sur l'étude de l'interblocage est une mise à jour d'un article écrit avec Jean-Claude Bermond [2]. Ce problème, qui a fait l'objet de nombreuses études dans le cas du *store-and-forward*, est étudié en modèle *wormhole*. Une solution proposée par Dally consiste à multiplexer les liens du réseau en un nombre minimum de *canaux virtuels*, afin de rendre la fonction de routage non bloquante. Le principal résultat de cette partie est que l'on peut toujours donner une telle fonction, quel que soit le réseau, en utilisant seulement deux canaux virtuels par lien, si on relâche la contrainte du routage sur les plus courts chemins. Ce résultat est précisé pour le réseau de de Bruijn $B(d, D)$, où l'on exploite l'existence de d arborescences arc-disjointes, ce qui permet de donner un routage avec deux canaux virtuels sur des chemins de longueur au plus $2D - 1$.

Dans le chapitre trois on présente un protocole de diffusion qui exploite les caractéristiques du modèle *wormhole*. En particulier, le résultat réfute l'idée communément admise que l'on ne peut pas diminuer le temps des algorithmes de com-

munications globales utilisés pour le cas du *store-and-forward* sur les machines qui utilisent une commutation de type *wormhole*.

Dans cet article en commun avec Joseph Peters [3], nous montrons qu'il existe un algorithme de diffusion sur le tore bidimensionnel en $\log_5(N)$ étapes, où N est le nombre de processeurs. Ce temps est obtenu en découpant récursivement le tore en 5 régions. Le découpage est possible quand $N = 5^{2k+1}$, et aussi pour certaines autres valeurs de N particulières.

Les hypothèses retenues pour cet algorithme sont aussi proches que possible de la réalité des machines présentées dans le premier chapitre. On se place dans un modèle où un nœud peut communiquer avec tous ses voisins à la fois, et où le temps de transmission d'un message de taille L à distance d est du type $\alpha + d\delta + L\tau$. Le facteur α est un temps d'initialisation de la communication, qui est prédominant dans les systèmes distribués, δ est un délai induit par la commutation de chaque nœud intermédiaire entre la source et la destination, et τ est le temps de transmission unitaire sur le lien, ou débit de ce lien.

La borne inférieure du temps de diffusion en *wormhole* (de même qu'en *store-and-forward*) est difficile à établir, mais si le paramètre α est dominant, l'algorithme presenté est optimal. Le principe de l'algorithme est de transmettre la donnée au plus grand nombre possible de processeurs à chaque étape, une étape consistant en la transmission de la donnée par les nœuds déjà informés. Des expériences menées par *Intel* sur la machine *Touchstone Delta*, avec un algorithme plus grossier, ont déjà établi le bien fondé de ces techniques. La technique que nous présentons peut être généralisée aux grilles toriques multi-dimensionnelles, et s'adapte à d'autres topologies conventionnelles.

Le quatrième chapitre est consacré à l'étude des graphes de *de Bruijn*. On y détermine plus précisement l'*excentricité moyenne* d'un sommet [1]. Ce paramètre est égal à la moyenne des distances entre un sommet et tous les autres. Ce calcul est difficile sur ce graphe car la fonction de distance entre deux sommets n'est pas donnée par une forme close. De plus, ce graphe n'est pas sommet transitif; on a donc peu d'arguments de symétries, et des calculs faits pour les premières valeurs des paramètres du *de Bruijn* montrent que la distribution des sommets pour l'excentricité n'est pas triviale, tant dans le cas du graphe orienté que dans le cas non-orienté.

Dans l'article écrit avec Jean-Claude Bermond et Zhen Liu, nous donnons des bornes pour le cas orienté, en déterminant les sommets qui atteignent ces bornes. Le résultat est asymptotiquement compris entre $D-1$ et $D-2$ pour les *de Bruijn* de demi-degré extérieur 2. Le cas non-orienté étant beaucoup plus complexe, le calcul n'a pu être mené que pour un sommet particulier, le sommet $a \cdots a$, dont nous conjecturons qu'il est celui de plus forte excentricité moyenne. Ici aussi les valeurs obtenues sont proches du diamètre du graphe, et on observe que la majeure partie des sommets sont à distance $D-3$ de l'origine, l'excentricité moyenne étant de l'ordre de $D-2,2$. On en déduit que le calcul du routage optimal n'est pas payant sur ce graphe, et qu'il vaut mieux utiliser des chemins de longueur D que

l'on obtient naturellement par un routage du type "bit-erosion" de l'adresse de destination.

Un chapitre complémentaire donne un résultat sur la *conjonction* des *de Bruijn*, et une application au calcul de la transformée de Fourier rapide (FFT) sur ce graphe est décrite. Cette implantation met en évidence l'adéquation de ce réseau à une grande classe d'algorithmes parallèles.

Bibliographie

[1] J.C. Bermond, Z. Liu, and M. Syska. Mean eccentricities of de Bruijn networks. submitted to Networks, 1992.

[2] J.C. Bermond and M. Syska. Routage wormhole et canaux virtuels. In Y. Robert M. Cosnard, M. Nivat, editor, *Algorithmique Parallèle*, pages 149–158. Masson, 1992.

[3] J. Peters and M. Syska. Broadcasting in torus networks with circuit-switched routing mode. In preparation, 1992.

[4] "Rumeur". Communications dans les réseaux de processeurs. Livre en cours de préparation, 1992.

Chapitre 1

Les machines parallèles actuelles

Avant-propos

Le marché des machines parallèles et les technologies associées évoluent si vite que ce texte ne peut offrir qu'une photographie prise en fin d'année 1992, et sous un angle choisi par les auteurs. Ces mêmes auteurs souhaitent que les compromis faits entre la technique et la facilité de lecture pour des non initiés conviennent au plus grand nombre.

Le but de ce chapitre est de présenter sommairement la technologie et les techniques mises en œuvre dans les machines parallèles, mais aussi de montrer que l'étude des modèles de communication, ainsi que des algorithmes de communication et de routage sont très utiles vu la complexité de ces machines. En effet, l'augmentation constante de la demande de puissance en calcul a conduit les constructeurs à produire des machines performantes, mais aussi à construire des processeurs qui calculent de plus en plus en plus vite, ce qui nécessite des algorithmes de communication efficaces afin de réduire le temps d'attente des processeurs.

Après un bref exposé des techniques utilisées en VLSI, nous présenterons une série de paramètres qui permettent de comparer et de classer les différents types de machines parallèles. Cet exposé souligne le lien entre les choix des constructeurs pour la topologie des réseaux, et les contraintes de fabrication.

Nous décrirons ensuite les principales machines parallèles de ces dernières années, avant de conclure par des perspectives sur les futures machines parallèles.

1.1 Architecture des machines parallèles

Les ordinateurs parallèles sont composés d'un ensemble de nœuds de calcul (voir figure 1.1) reliés entre eux par un réseau. Ces nœuds sont généralement constitués par :

- un processeur de calcul scalaire.

- une ou plusieurs unités flottantes.
- une mémoire locale (parfois les nœuds ont accès à une mémoire globale. L'accès à cette mémoire dite partagée se fait par le réseau).
- une horloge interne.
- des liens de communication.

On notera que bien souvent, on assimile un nœud à un processeur, surtout lors de la description des algorithmes sur ces machines. Cependant, un nœud peut comporter plusieurs processeurs, ou encore en simuler un grand nombre; on parle alors de processeurs virtuels. Les nœuds de la nouvelle génération de machines parallèles comportent en plus :

- des mémoires caches de données et d'instructions. Ce sont des organes de mémoire à accès très rapide mais de faible capacité, dans lesquels sont stockées les informations les plus souvent utilisées, ou celles que l'on va employer 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.
- un noyau système du type UNIX, qui permet d'avoir les mêmes facilités de travail que sur une station de travail (accès aux fichiers sur disque, affichage de données, mémoire virtuelle).
- un analyseur de performances qui enregistre certains événements se déroulant sur le nœud (accès mémoire, communications, etc..). Ceci permet de visualiser sur un écran le déroulement du programme en chaque point du réseau.
- des composants qui gèrent le routage des messages entre les nœuds : calcul de la fonction de routage, multiplexage des liens.

Deux familles de machines parallèles peuvent être distinguées : celles à mémoire partagée, et celles à mémoire distribuée. Sur les machines à mémoire partagée, les processeurs accèdent tous à une mémoire commune, par l'intermédiaire d'un réseau d'interconnexion, qui gère les accès concurrents à la mémoire. Sur les machines à mémoire distribuée, chaque processeur possède sa propre mémoire, ce qui oblige les processeurs à communiquer entre eux par échange de messages.

En raison des coûts trop élevés et de la complexité de réalisation (il est très difficile aujourd'hui de gérer une mémoire partagée avec quelques dizaines de processeurs), les constructeurs ont opté pour le distribué.

Malheureusement, pour exécuter un programme n fois plus vite, il ne suffit pas de mettre n processeurs sur une carte. Les n processeurs doivent échanger des informations, et les communications entre processeurs prennent du temps (et même constituent dans certains cas l'essentiel du temps total).

Le support de ces communications est constitué de trois classes de réseaux :

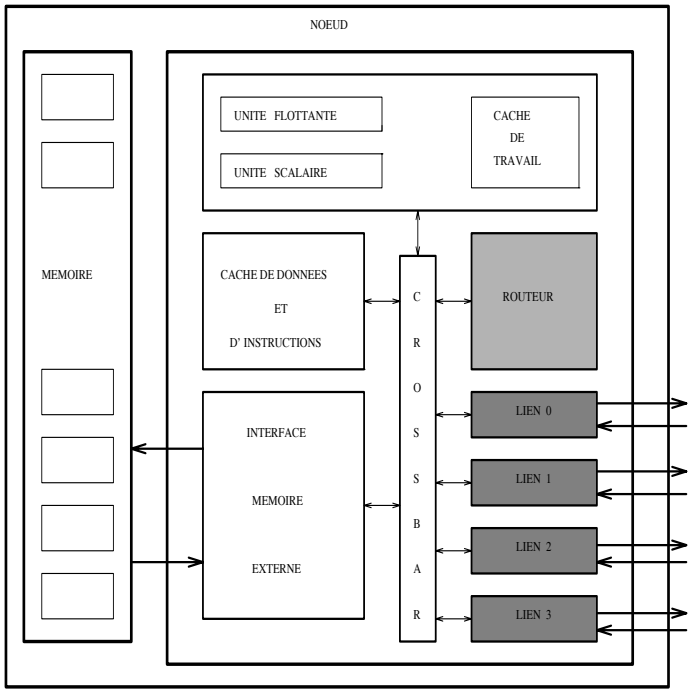

Figure 1.1: Architecture d'un nœud du réseau.

- Point à point : chaque nœud peut communiquer avec ses voisins par des liens directs. C'est le cas de la plupart des machines.

- Multi-étage : les nœuds du réseau sont des commutateurs (switches). Le réseau permet d'établir des chemins entre les processeurs d'un coté et les bancs mémoire de l'autre. Souvent les bancs coïncident avec la mémoire locale d'un processeur (exemple de l'IBM RP3).

  La machine de Bolt Boranek et Newmann (BBN) dispose d'un réseau de type butterfly, et un banyan 2-3 était utilisé sur le "Texas Reconfigurable Array Computer" à la fin des années 70.

- Crossbar : dans ce type de réseau tous les nœuds sont à égale distance les uns des autres : il existe une connexion entre tout couple de nœuds. Malheureusement la complexité du réseau est telle qu'il est difficile aujourd'hui de construire des crossbars avec plus de 32 entrées. On doit donc les chaîner pour construire de plus grands réseaux et cela entraîne des délais plus importants lors des transmissions interprocesseurs.

  Plutôt que de vrais crossbars, les constructeurs proposent des composants programmables qui permettent d'établir des connexions entre un nombre limité de liens de communication. C'est le principe introduit par Charles

Clos (Bell Laboratories) dans les années 50. Le réseau est construit récursivement à partir de sous-réseaux. Au plus bas niveau se trouvent des commutateurs élémentaires à m entrées et n sorties. Le réseau d'interconnexion est reconfigurable par programme, contrairement au réseau point à point où la topologie est fixée lors de la construction de la machine.

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. Il y a aussi plusieurs niveaux d'intégration des composants, et 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 sont réalisés en technologie VLSI (Very Large Scale Integration : technologie de construction de circuits intégrés [53], [32] et [64]), et doivent respecter les contraintes de l'intégration en deux dimensions. Le circuit est en fait le graphe des connexions entre des cellules élémentaires qui permet de réaliser le calcul d'une fonction booléenne. Le réseau des nœuds doit lui aussi être dessiné 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 du dessin des circuits intégrés (graphe des interconnexions entre les cellules élémentaires [59]) 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 dessiner 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. En 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 [69] et [45] pour une introduction du problème ou [49] pour des références complètes sur la théorie des VLSI.

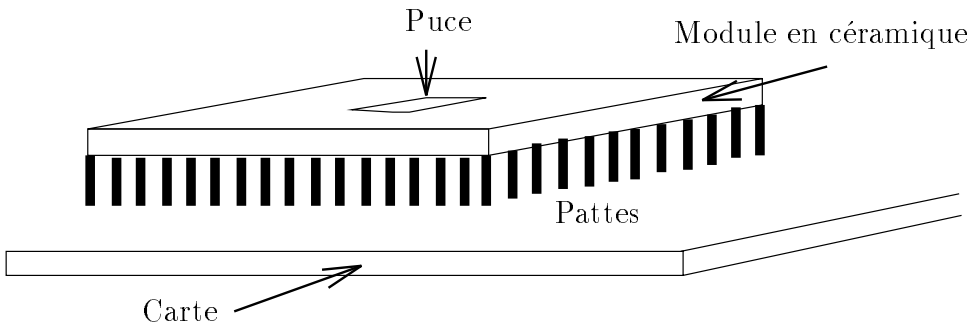

Figure 1.2: Différents niveaux d'intégration en VLSI

1.2.2 Contraintes sur le nombre de liens

La figure 1.2 montre les trois niveaux "d'encapsulage" que sont la puce, le module et la carte [6] (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 le diamètre des substrats réalisables.

Les puces comportent aujourd'hui environ 3 millions de transistors disposés sur un peu plus d'1 cm^2, alors que le nombre de pattes issues du module est, par exemple, de 208 sur le T9000 d'Inmos. La plupart des processeurs sont destinés à équiper les 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 que l'on peut tirer d'un module et non les terminaisons de la puce qui limite l'interconnexion des VLSI, à cause de la différence d'échelle entre ces deux niveaux [67, p141]. Comme la surface des circuits doit être minimisée pour les raisons évoquées ci-dessus, la portion reservée aux communications externes ne peut être augmentée sans entraîner des pertes de performances globales du circuit. Un transistor MOS peut changer d'état en 0.1 ns, mais si on ajoute quelques centaines de microns sur sa ligne de connexion, le délai est augmenté de plusieurs nano secondes [61] !

De plus, la différence d'échelle entre puce et carte implique 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 [58].

Parmi ces pattes en nombre limité, une grande partie est reservée pour l'interface avec la mémoire externe du processeur, d'autres devant transmettre des signaux utiles au système et à la synchronisation des processeurs entre eux.

Il reste donc un nombre de pattes très limité pour réaliser les liens de commu-

nication entre processeurs [18]. Ceci explique en partie la préference pour les liens "série", où l'on transmet une suite de bits sur une patte aux liens parallèles où 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 comme la parité ou un accusé de réception).

Il faut aussi trouver un équilibre entre le nombre de liens et la largeur (ou 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 de 1 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 [21] montrent que les réseaux en grilles de faibles dimensions sont plus performants que les hypercubes pour un même nombre de processeurs. L'hypothèse est de normaliser 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 égales en nombre de nœuds) du réseau considéré. Il donne même une valeur optimale de 2 dimensions pour 256 processeurs. Cependant cette étude porte sur les grilles multi-dimensionnelles (k-ary-n-cubes) et ne prend pas en compte des réseaux moins courants comme les *de Bruijn* ou les cube-connected-cycles, qui offrent cependant un diamètre faible pour un degré (nombre de liens) constant.

Si 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'1 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 1024 dans une grille carrée 1024×1024.

Certains travaux récents ([15],[14],[65]) ont montré que la réalisation d'un réseau de processeurs avec une topologie de *de Bruijn* était réalisable. Un décodeur de codes de *Viterbi* a été réalisé dans le cadre de la mission Galileo pour la NASA. Le résultat est un $B(2, 13)$ dont chaque nœud est un circuit arithmétique élémentaire (papillon) du décodeur. Dans une réalisation plus générale cela pourrait être un vrai processeur de calcul. D'ailleurs on a ici affaire plus à une véritable machine MIMD qu'à un simple décodeur, car chaque circuit a sa propre horloge et sa mémoire. La construction est modulaire, et est composée de 16 cartes. Chaque carte comporte 16 puces, une puce implantant 32 papillons. 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 (elles passent par le chassis). 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).

Quoiqu'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 "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 [20].

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 [67, p143].

1.2.4 Nécessité d'une mémoire distribuée

Plus grande est la mémoire, plus le délai moyen d'accès aux données est important. Les techniques employées pour diminuer ce temps d'accès sont l'utilisation de caches encapsulés avec la CPU (dans la même puce), ce qui est rendu possible par le nombre de transistors qui va toujours croissant. Parfois, on utilise plusieurs niveaux de caches entre le processeur et la mémoire [38]. 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. Toutefois, les nouvelles architectures ayant un espace d'adressage très important, la limite sera imposée par la taille physique des nœuds. Chaque processeur gère sa mémoire locale et accède à celle des autres par envois de messages.

De plus, on ne pourrait pas gérer la cohérence des caches au delà d'un certain nombre (20 ou 30) sans diminuer les performances du système. Le problème de la cohérence des caches consiste à vérifier qu'une donnée chargée dans des caches différents soit mise à jour dans tous ces tampons en cas de modification par l'un des processeurs.

1.2.5 Interface entre les processeurs et le réseau

Le temps passé à recevoir et transmettre les messages doit être minimisé, mais surtout l'interface doit permettre de recevoir et d'envoyer plusieurs messages en même temps. Ce n'est pas le cas de la plupart des processeurs élémentaires actuels. La technique employée par Fujitsu sur l'AP1000, qui permet d'envoyer directement

un message de la mémoire cache au routeur aide à minimiser le temps de transfert. En effet, les données à émettre sont souvent celles que l'on vient de traiter, et se trouvent donc dans le cache.

1.2.6 Refroidissement

De tout temps les constructeurs :-) ont multiplié les MFlops des processeurs par le nombre de processeurs en magasin pour annoncer la puissance maximale de leur machine parallèle. Cependant, il ne suffit pas d'assembler indéfiniment des "lego" pour avoir des systèmes toujours plus puissants. Un premier problème est la longueur des connexions, et c'est la raison principale de la limite à 16K nœuds de la CM-5. Un autre problème qui a été mis en avant par la firme Parsytec 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. Les techniques de refroidissement liquide employées par Cray ne sont sûrement pas applicables directement sur des machines de la taille d'un gymnase, et il faudra donc reconsidérer certains paramètres des réseaux.

1.3 Architectures des processeurs

Le temps d'exécution d'un programme sur un processeur est donné par la fonction suivante : $T = NI \times NC \times TC$ [33], où T est la durée exprimée en secondes, NI le nombre d'instructions machines 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.

Une architecture vise à minimiser cette fonction. Les architectures les plus employées dans les processeurs actuels sont : RISC, pipeline, superpipeline, superscalaire et VLIW.

- **RISC** : les années 80 ont vu l'apparition des architectures RISC (Reduced Instruction Set Computer) [33]. L'objectif de ces architectures est de pouvoir exécuter une instruction par cycle ($NC = 1$). Dans les architectures précédentes (CISC : Complex Instruction Set Computer), le jeu d'instructions proposé était très étendu, et ceci avait au moins deux conséquences :

  - les compilateurs avaient du mal à exploiter au mieux ce jeu d'instructions
  - la partie contrôle du processeur était micro-programmée, et utilisait une surface importante du composant (60 % dans le cas du 68000 de chez Motorola).

  Au contraire, le RISC est un jeu plus réduit d'instructions qui permet un contrôle câblé plus rapide (cela consiste en un automate d'états finis), 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 (voir plus loin).

Il faut noter que les dernières architectures, comme le 960 d'Intel combinent les techniques RISC et CISC : c'est un processeur de type superscalaire.

- **Pipeline [46]** : le principe est de décomposer les instructions en n phases élémentaires de longueur 1 cycle. Ainsi on peut exécuter n instructions en parallèle. Pendant un cycle de l'horloge, la phase k de l'instruction i est exécutée en même temps que la phase $k-1$ de l'instruction $i+1$, ... Ces phases sont pour une instruction simple du Sparc (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;

  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.

- **Superpipeline** : pipeline dont le nombre d'étages est supérieur à 5.

- **Superscalaire** : sur ce type d'architecture on peut exécuter plusieurs instructions par cycle. De plus, chaque opérateur peut posséder son pipeline. Par exemple, cela consiste en l'exécution d'un calcul flottant et d'un calcul scalaire en même temps. Le problème est de pouvoir lire plusieurs instructions à la fois dans un cycle. Pour cela, il faut utiliser un bus plus large.

- **VLIW** : Very Long Instruction Word. C'est la généralisation du superscalaire : plusieurs instructions du programme sont exécutées au cours du même cycle par des unités fonctionnelles différentes. Un mot d'instruction est la concaténation des instructions destinées à chacune des unités qui travaillent en parallèle.

1.4 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 (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) et de plus la plupart des algorithmes parallèles nécessitent de nombreuses communications qui limitent les performances.

Mots clés pour l'étude de performances

A quand la machine ad hoc ? Mille milliards de mille sabords !

- MFlops (Million Floating point operation per second) : nombre de millions d'opérations flottantes effectuées en une seconde. Actuellement, on mesure plutôt le nombre d'opérations en GFlops ce qui correspond aux milliards d'opérations flottantes par seconde (G pour Giga). La prochaine étape sera le TFlops qui correspondra aux milliers de milliards d'opérations flottantes par seconde (T pour Tera, ou Trillion en anglais).
- MIPS (Million Instruction Per Second) : nombre de millions d'instructions réalisées en une seconde. De même, il existe le GIPS et prochainement le TIPS.
- Bande passante d'un lien : taux de transfert d'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 Mega octets par seconde noté Mo/s.
- Délai ou latence : temps moyen entre le moment où le début d'un message est émis par la source et le moment où la fin du message est reçue par le destinataire.
- 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 40MHz correspond à un cycle de 25 nano-secondes.
- Benchmark : programme standard dont le but est de comparer les performances de différentes machines ou processeurs. Les principaux benchmark sont Whestone, Dhrystone, les boucles de Livermore, LINPACK benchmark, SPEC.
  - Le benchmark Whestone a été le premier élaboré en 1976 au National Physical Laboratory (Grande-Bretagne). Il est constitué de plusieurs modules, et chacun contient des instructions de différents types (arithmétique sur des entiers, arithmétique sur des flottants, des instructions "if", des appels à des fonctions, etc..) [72]. Ces modules sont pondérés, et exécutés plusieurs millions de fois, ce qui donne des résultats en MWIPS (Mega Whestone Instructions per second).

- Le benchmark Dhrystone est une amélioration du Whestone. Il est apparu en 1984. Ce benchmark est constitué d'une boucle qui comprend 94 instructions. Les résultats sont donnés en Dhrystone par seconde [72].
- Les boucles de Livermore sont 24 boucles FORTRAN qui ont été réalisées au Lawrence Livermore National Laboratory. 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..) aux noyaux de tri ou de recherche [35]. Ces boucles sont utilisées pour évaluer les performances en MFlops des processeurs et des machines parallèles.
- LINPACK (LINear PACKage) est une bibliothèque de procédures FORTRAN, facilement transportable, permettant la résolution de systèmes linéaires par diverses méthodes [10] [23]. 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 [29]. 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 dans [28])
- SPEC (System Performance Evaluation Consortium) regroupe un ensemble standard d'applications réelles afin de comparer les performances des différents systèmes. Cet ensemble a été créé en 1989. SPEC est constitué par 10 programmes de tests (6 en FORTRAN et 4 en C). Ces programmes sont des applications réelles, 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 [37]. Les programmes sont exécutés sur la machine cible, avec prise de temps. Le rapport du temps d'exécution sur la machine cible et sur un DEC VAX 11/780 donne une mesure que l'on exprime en SPECmark. Cette échelle permet de comparer les différents processeurs et machines du marché. C'est un des tests les plus utilisés, car il permet de tester des architectures sur des applications concrètes. De plus, il tient bien compte de la gestion de la mémoire et des caches.

Le numéro spécial "benchmarking of high performance supercomputers" de Parallel Computing [27], fait un tour d'horizon sur toutes les méthodes pour évaluer les performances des ordinateurs.

Exemple de calcul des MFlops sur une application

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

culer une addition ou une multiplication.

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

```
for i := 1 to 100
  for j := 1 to 100
    C[i,j] := 0;
    for k := 1 to 100
      C[i,j] = C[i,j] + A[i,k] * B[k,j];
```

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

Un cycle prend 25 nano-secondes, ce qui donne 75 nano-secondes 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 2×100^3 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 80 millions d'opérations par seconde, soit 80 MFlops.

On retiendra dans les deux cas qu'on ne tient pas compte du temps d'accès à la mémoire, et dans le cas des architectures pipelines du temps d'amorçage de ceux-ci.

1.5 Classification des machines

La grande diversité des machines parallèles ([4],[3], [1], [30], [41], [66] et [40]) a donné lieu, de la part de Flynn, à une classification [36] qui fait toujours autorité même si aujourd'hui elle semble un peu dépassée. En effet, cette classification n'est basée que sur 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 soit multiples. De ce fait, on a du mal à classer certaines machines (par exemple celles à flot de données), ou à distinguer les machines à mémoire distribuée des machines à mémoire globale.

Les ordinateurs "parallèles" qui nous concernent ici sont ceux des types SIMD (Single Instruction Multiple Data flows) et MIMD (Multiple Instruction Multiple Data flows) à mémoire distribuée.

- **SIMD** : sur ce type de machine les processeurs effectuent les mêmes instructions, mais sur des données différentes réparties sur les processeurs. L'avantage des machines SIMD est que seules les unités de calcul doivent

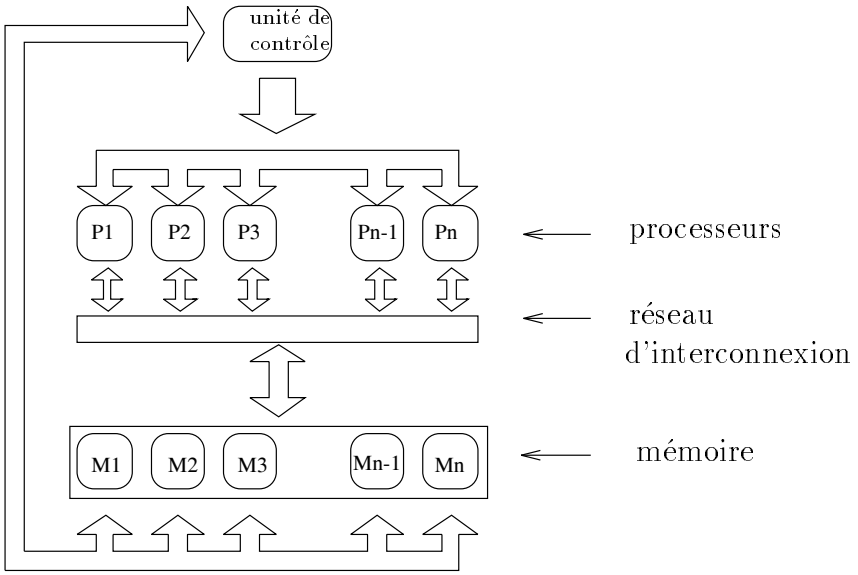

Figure 1.3: Architecture SIMD

être dupliquées. En effet, une seule unité de contrôle suffit à décoder les instructions pour tous les processeurs (voir figure 1.3). Le programme est exécuté sur une station frontale. Certaines instructions (celles qui traitent les données en parallèle), sont diffusées à tous les processeurs afin d'être appliquées aux données réparties dans les mémoires locales de ces processeurs, aux mêmes endroits relatifs.

En pratique, de tels ordinateurs opèrent sur des vecteurs et des matrices. La Connection Machines de Thinking Machines Corporation, la MP-1 de MasPar ou les DAP (Digital Array Processors) de AMT et la Zephir de Wavetracer sont des machines SIMD. Ces machines ont un point commun : assembler un grand nombre de processeurs simples (64 K processeurs 1 bit sur la CM-2) qui travaillent en mode synchrone (c'est-à-dire que tous les processeurs effectuent la même instruction au même moment). C'est ce que l'on appelle un parallélisme à grain fin (le rapport nombre de données traitées sur le nombre de processeurs utilisés est proche de 1).

- **MIMD** : ce type de machines permet aux différents processeurs élémentaires de travailler sur des données différentes, et aussi à chaque processeur élémentaire d'exécuter des instructions différentes. 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'interruptions

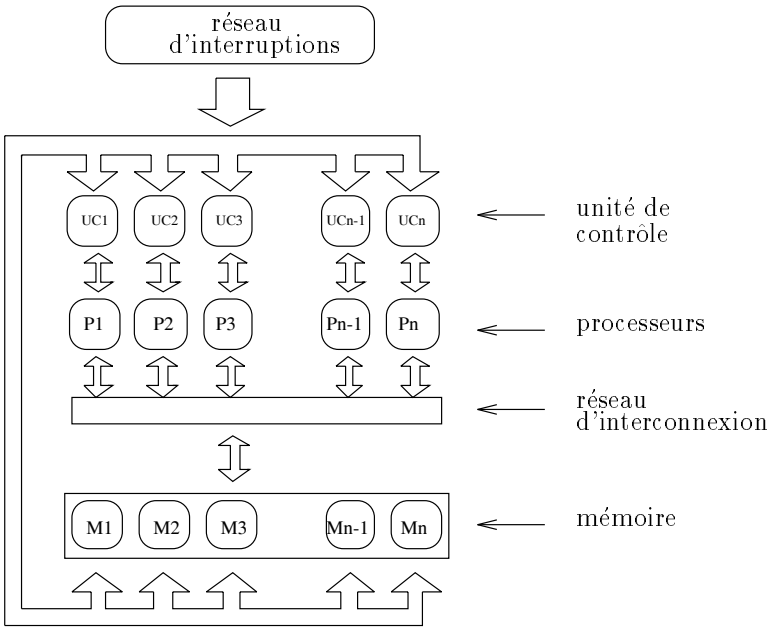

Figure 1.4: Architecture MIMD

(voir figure 1.4). Les iPSC/2 iPSC/860, la Paragon d'Intel, les machines à base de Transputers Inmos (T-node et Mega-Node de Telmat, Computing Surface de Meiko, ...) ou les fameux supercalculateurs Cray comme le Y-MP/832 sont des machines MIMD. Ces machines ont un nombre plus faible de processeurs que dans le cas des SIMD (parallélisme à grain moyen et grossier), 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.

Sur les machines à mémoire partagée, la faible granularité est dûe aux limitations du réseau d'interconnexion et aux conflits mémoire à gérer. Les machines de type MIMD n'ont pas d'horloge globale et opèrent donc en mode asynchrone.

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 se fait grâce à une variable locale qui contient le numéro du processeur. En fait, ce type de programmation est une extension du mode de programmation sur les machines SIMD. Il se développe grâce à sa facilité de réalisation et aux langages très performants disponibles tels que FORTRAN 90 et C* qui permettent d'exploiter le parallélisme des données (data parallelism). Les machines qui offrent ce mode

sont les machines MIMD telles que le T de FPS, la Paragon d'Intel ou la CM-5 de Thinking Machines Corporation.

1.6 Historique des machines parallèles

1.6.1 Les premières machines

Les premiers réseaux de processeurs sont apparus il y a environ 10 ans. Une des premières machines à avoir été construite est le Cosmic Cube [62], conçue sous la direction de C. Seitz à Caltech en 1981 dans le cadre d'un projet de recherche. Le Cosmic Cube était constitué de 64 processeurs Intel 8086 organisés en hypercube de dimension 6. Ce projet 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 à topologie d'hypercubes comme celles produites par les societés Ametek [60] et nCUBE. La taille moyenne de ces machines était en 1985 d'une centaine de processeurs. D'autre part, FPS proposait les séries T dont les nœuds étaient constitués de Transputers T414 d'Inmos couplés avec des coprocesseurs vectoriels. Les 4 liens des Transputer permettaient de construire des hypercubes de dimension 4, et pour les hypercubes de taille supérieure, il a été nécessaire de multiplexer les liens. Cela a pénalisé les communications entre processeurs, et comme les calculs vectoriels étaient très rapides, on a rompu l'équilibre entre les communications et les calculs .

La caractéristique essentielle de ces machines est l'échange de données entre processeurs par communication le long de liens physiques. Le routage est de type "store and forward", c'est-à-dire que les messages sont stockés dans les processeurs intermédiaires le long du chemin entre la source et la destination avant d'être retransmis. Ce routage est fait de manière logicielle, ce qui le rend peu efficace comme le montrent les temps de communication présentés dans le chapitre suivant.

1.6.2 Les machines de la deuxième génération

La deuxième génération introduit des machines issues des mêmes technologies, mais avec des processeurs plus performants (80386 et i860 à la place des 8086 d' Intel). Les communications sont plus performantes mais restent toujours pénalisantes par rapport au potentiel de calcul. C'est le cas des iPSC/2 et iPSC/860 d'Intel, du nCUBE etc...

Un certain nombre de machines telles que la série T-node de Telmat [12], Computing Surface de Meiko, Volvox d'Archipel sont construites autour du Transputer T800, ou parfois du i860 pour le calcul et des T800 pour les communications.

Le Transputer n'autorise que le routage "store and forward" mais dispose de 4 liens de communication autonomes qui peuvent travailler en parallèle avec l'unité centrale (CPU) et ont des accès mémoire directs (DMA). Les machines à base de Transputers utilisent des crossbars pour interconnecter les processeurs

élémentaires. Si l'on veut en relier un grand nombre comme dans le cas du Mega-Node, il faut un deuxième niveau de crossbars. Le Mega-Node est en fait un ensemble de "T-Node". Les Transputers appartenant au même Node (paquet de 32) sont reliés par des crossbars NEC 32x32, et certains liens de ces crossbars sont connectés par un second niveau de crossbars (des Inmos C004) entre les Nodes. Cela entraîne un délai plus grand sur les liens qui traversent les deux niveaux (le débit passe de 1.82 Mo/s à 1.33 Mo/s).

Ceci donne des réseaux hétérogènes où l'utilisation du mode SPMD n'est pas bonne, car c'est le processeur le plus en retard qui va donner le rythme.

Pour cette génération de machines le routage reste encore du type "store and forward" sauf pour Intel qui introduit le "direct connect" (dérivé de la commutation de circuit). Ce mode présenté plus en détail dans le chapitre suivant permet de diminuer le temps de transfert d'un message entre deux nœuds éloignés. A cette fin, deux techniques sont exploitées. Le contrôle devient indépendant du processeur de calcul et est du type hardware (routeur implanté avec des mémoires programmables : gate array CMOS). On établit d'abord un circuit entre la source et la destination avant d'émettre, au lieu de répéter l'opération pour chaque nœud intermédiaire. De plus, il n'y a pas de stockage dans ces nœuds intermédiaires comme en "store and forward".

Une des premières machines à avoir mis en œuvre cette nouvelle technique est la Simult (ex Ametek) 2010, en collaboration avec Caltech [67, chap 1]. Par la suite, tous les constructeurs de machines ont adopté ce type de routage, comme les versions iPSC/2 et iPSC/860 de l'hypercube d'Intel où les i860 remplacent les 386, ou encore le Torus Routing Chip (Message Driven Processor du MIT [19]) qui autorise un routage "wormhole" sans interblocage sur un k-ary-n-cube, de préférence de dimension $n = 2$. On y exploite les techniques exposées par Dally et Seitz dans [22].

Dans un autre domaine (orienté vers le traitement du signal, où la régularité des calculs permet un traitement pipeline efficace), Texas Instrument propose le processeur TMS320C40, qui est un module doté de 6 liens de communication bidirectionnels [43].

1.7 Les projets ou réalisations actuelles

1.7.1 Les processeurs

A l'heure actuelle, on semble préférer les processeurs intégrant 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 ils 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 (ex Computing surface de Meiko, Volvox d'Archipel, etc..).

- **Alpha**

  Le processeur Alpha est la nouvelle architecture RISC proposée par DEC. Comme pour Sun avec le Sparc, DEC propose une architecture ouverte pour diverses implantations, en particulier pour les implantations parallèles. Les registres sont des 64 bits [24] [25]. 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.

  Prévision de 150 SPECmarks, 300 MIPS, et 150 MFlops.

  Cray, qui est un partenaire industriel de DEC dans ce projet, prévoit d'utiliser l'Alpha pour construire des machines massivement parallèles (Cray MPP), et travaille dans ce but sur les améliorations à apporter à cette architecture.

- **i860 XR/XP**

  Ce processeur est sorti vers la fin des années 80. Son objectif est de concilier les fonctions d'un processeur courant et la puissance de calcul sur des réels.

  Le i860 a une architecture RISC 64 bits ayant une fréquence d'horloge de 40 Mhz. Une version améliorée existe (i860 XP avec cohérence de cache et PAX); elle 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 gestion 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 [13].

  Sa conception ne prévoyant pas de liens de communication, les cartes utilisées sur les machines Meiko ou Archipel sont dotées de Transputers pour pallier à cette lacune.

- **PA-RISC**

  Hewlett-Packard développe des composants PA-RISC, que Convex doit utiliser dans sa prochaine machine massivement parallèles. Ces processeurs auront une puissance de 120 SPECmarks.

- **Sparc**

  Son nom est l'acronyme de "Scalable Processor ARChitecture". C'est un processeur RISC 32 bits [42], utilisé par Sun dans la construction de ses stations de travail. Actuellement, ce processeur a une fréquence d'horloge de 40 Mhz, et peut fournir 30 MIPS, avec une puissance de crête de l'ordre de 5 MFlops. Grâce à sa MMU (Memory Management Unit) très performante, le processeur Sparc est capable de changer très rapidement de contexte entre

différents processus, ce qui lui permet de gérer les applications de plusieurs utilisateurs.

Sun s'est associé avec Texas Instruments, afin de construire un processeur à architecture "super Sparc". Le nom de ce processeur est le Viking; il est capable de délivrer plus de 100 SPECmark.

- **Texas Instrument TMS320C40**

  Ce processeur 32 bits est construit pour le traitement parallèle du signal (Digital Signal Processor) et est doté de 6 ports de communication avec accès de type DMA. La bande passante totale est de 320 Mo/s, et les performances atteintes sont de 25 MIPS et 50 MFlops [43].

  Il comporte des instructions de transfert de bloc avec l'adressage bit-reverse utile dans le calcul de la FFT (transformée de Fourier rapide).

  Texas Instruments propose aussi un "Digital crossbar switch", c'est le SN74ACT8841 qui offre 16 connexions de liens. Le délai de reconfiguration est de 20 ns.

- **T800**

  Ce processeur RISC à 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 20 Mbits/s. Ces processeurs sont interconnectés par un crossbar (C004) [44]. Le T800 est peu performant, et sert surtout dans les systèmes embarqués, ou est couplé a des processeurs plus performants pour gérer les communications.

- **T9000 et C104**

  - **T9000**

    Le T9000 est le processeur qui doit succéder au T800 dans la gamme Inmos. Tout comme le T800, il sera destiné à la construction de machines MIMD. Chacun de ses liens bidirectionnels pouvant être connecté à un crossbar décrit ci-après, le C104. Le T9000 est un processeur 32 bits (180 mm^2, 1 μ CMOS à 3 niveaux), doté d'une FPU 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 [56].

    Comme dans le T800, il y a un ordonnanceur intégré qui permet un temps de changement de processus actif très faible (de l'ordre de la micro seconde). De plus, les processus inactifs, ne consomment pas de temps CPU. Ceci permet d'écrire des programmes parallèles performants (en Occam ou en C) [11], [52].

Les principales évolutions par rapport au T800 (encore largement utilisé, surtout dans les systèmes embarqués) sont :

* la fréquence d'horloge qui passe de 25 à 50 MHz
* le débit des liens de communication qui passe de 20 Mbits/s à 100 Mbits/s. Cela donne une bande passante (en bidirectionnel) globale de 80 Mo/s. A noter que si un lien de T800 autorise des transferts à 1.8 Mo/s en transmission mono-directionnelle, le taux n'est plus que de 1.2 Mo/s si le lien est utilisé en mode bidirectionnel.
* 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 la CPU. Ce VCP gère les 4 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 crossbar adaptés à ces Transputers, les C104.

– **C104**

Le C104 est un crossbar 32x32 non bloquant, qui permet d'interconnecter des liens de T9000 entre eux. Les messages transmis sur ces liens sont découpés en paquets de 32 octets et sont routés en mode "wormhole" dans le C104 [50]. 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 suivant, mais on peut préciser ici qu'il se différencie de la commutation de circuit car on n'attend pas d'accusé de reception dans ce mode de commutation. On transmet les données en même temps que l'on établit le chemin, car on est certain de ne pas rester bloqué ou devoir rebrousser chemin.

Pour éviter les interblocages, Inmos a implanté un routage de type "interval labelling". Ce routage 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 sur les topologies courantes.

On peut aussi utiliser un mode "routage universel" [70] 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 16 384 nœuds).

Voici un tableau récapitulatif des différents types de processeurs commercialisés (ou qui vont l'être) qui permet de comparer les performances et les optiques de chaque constructeur.

Processeurs	Architecture	MFlops	Mips	Horloge (MHz)	Liens
Alpha	RISC	150	300	200	aucun
i860 XR/XP	RISC	75	42	40	aucun
Sparc	RISC	5	40	30	aucun
TI C40	RISC	50	25	25	6
T800	RISC	1.5	30	25	4
T9000	RISC	25	200	50	4

Notons que les laboratoires japonnais travaillent aussi sur des processeurs très performants, par exemple Fujitsu et Hitachi qui produisent des processeurs dont le temps de cycle n'est que de $2.5ns$ [73].

1.7.2 La dernière génération de machines MIMD

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

- **Alliant Campus FX800**

  Dans la lignée des FX800 et FX2080, Alliant vient de sortir la Campus FX800 [2]. Cette machine interconnecte 800 processeurs connectés via un réseau de clusters, 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 cluster. La Campus/800 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. La latence est de 40 μs.

  On a mesuré 4.781 GFlops Linpack sur une version à 192 processeurs – machine vendue \$5.21 M–, à comparer aux 13.7 GFlops sur le même benchmark pour le Cray Y-MP C90 (16 processeurs) vendu \$30.5 M. Cela fait le GFlops à moitié prix de celui du Cray.

- **CM-5**

  C'est la dernière née de Thinking Machines Corporation [17]. Elle permet d'interconnecter actuellement de 32 à 2K processeurs. Le but de TMC est d'augmenter le nombre de processeurs élémentaires jusqu'à 16K, 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 256 GFlops.

  Il y a 4 nœuds sur une carte, 8 cartes par rack, 8 racks par cabinet. La version 16K processeurs est donc composée de 64 cabinets

  La taille maximale est limitée par la longueur des câbles utilisés, sinon on pourrait atteindre 256K processeurs. Les processeurs élémentaires sont constitués d'un processeur SPARC, d'une mémoire de 32 Mo, et de 4 unités de calcul flottant de 32 MFlops chacune.

  le tout de la taille d'un gymnase, ou plutôt d'une patinoire pour le refroidissement !

  .

  Une de ses principales originalités 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 le "fat-tree" (voir figure 1.5) Elle a été définie par C. Leiserson dans [47]. 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.

  La figure 1.6 montre comment est réalisé un des contrôleurs-routeurs du "fat-tree" [48]. Chaque cellule de routage a 4 fils, et 2 ou 4 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.

  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 2 contrôleurs le débit est de 10 Mo/s et, pour plus de contrôleurs le débit est de 5 Mo/s. Lors d'une communication point à point, le message applique la technique suivante : remonter de façon aléatoire dans le réseau de contrôleur-routeur jusqu'à trouver celui qui permettra la redescente vers le destinataire, de façon déterministe.

  la bande passante globale est de $\frac{1}{2}$ Goctets/s et la latence est de 5 μs.

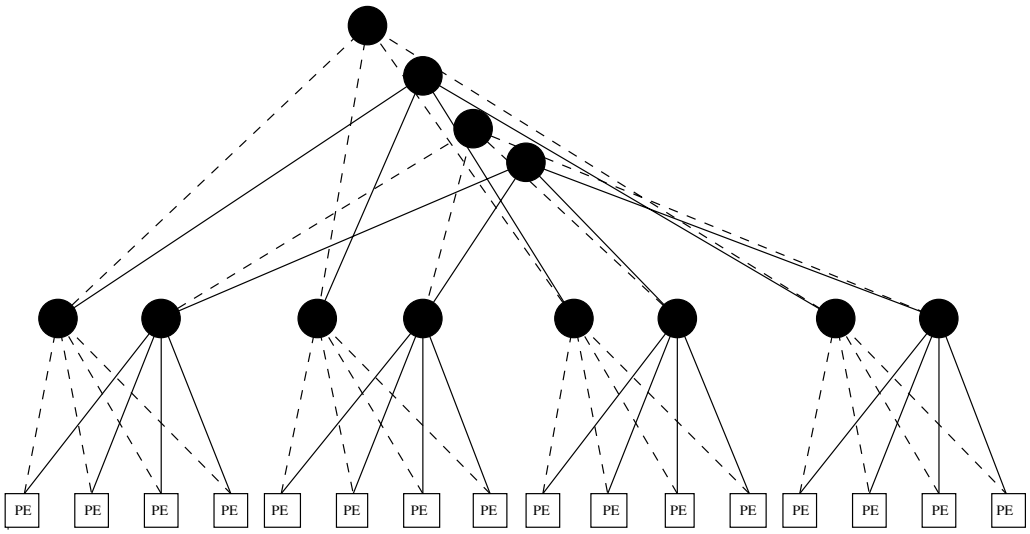

Figure 1.5: FAT - TREE du réseau de données de la CM5

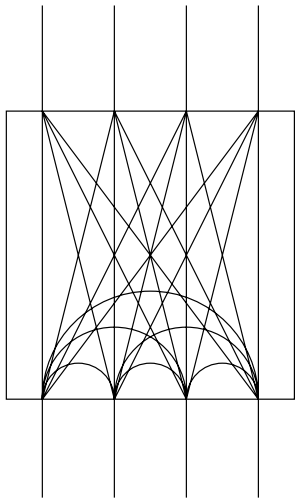

Figure 1.6: 1 contrôleur du fat-tree

La topologie du réseau de contrôle est beaucoup plus simple. En effet, il s'agit d'un arbre binaire complet [48] dont les feuilles sont les processeurs et les autres nœuds des contrôleurs. Ce réseau permet de réaliser toutes les opérations globales (diffusion, synchronisation, or, xor, 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.

Notons que le temps moyen entre 2 pannes (MTBF :Mean Time Between Failures) est de 1 à 2 heures selon les configurations ! Ceci est dû à la compléxité de la machine.

La programmation se fait en mode SPMD et est compatible avec le langage C* utilisé sur les CM-2 SIMD.

- **Concerto**

  Cette machine est le fruit de l'association de trois constructeurs européens Meiko, Parsys et Telmat qui forment le consortium PCI (Performant Computing Industrie) [51]. La Concerto est une machine MIMD à mémoire distribuée; elle possède de 2 à 76 processeurs élémentaires, ce qui permet d'avoir une puissance de crête de l'ordre de 3 GFlops. Chaque processeur élémentaire est constitué d'un i860, d'une mémoire de 32 Mo et de 2 T800 qui assurent les communications entre processeurs élémentaires. La particularité de cette machine est sa reconfigurabilité. En effet, on peut créer sur cette machine n'importe quelle topologie de degré 8 avec les 4 liens de chaque Transputer. Notons que la société française Archipel propose aussi des machines mixtes en mélangeant des i860 et des T800 qui échangent leur données par la mémoire locale du nœud.

- **Cray MPP**

  Le MPP est un futur supercalculateur de Cray Research [26]. Son architecture sera MIMD, avec une mémoire partagée, mais accessible globalement par tous les processeurs. Chaque nœud du réseau devrait être constitué de deux processeurs ALPHA. Cette machine, comme les autres de la même génération, fournira le mode SPMD. Elle devrait être capable d'atteindre 100 GFlops en 1993 et 1 TFlops en 1995.

- **CS-2**

  La dernière machine du consortium PCI.

- **Fujitsu AP1000**

  Cette machine MIMD mono-utilisateur comporte jusqu'à 1024 cellules. 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 cellules.

Chaque cellule est dotée d'un processeur Sparc possédant 16 Mo de mémoire RAM et est reliée à trois réseaux de communication distincts :

- Le T-net est le tore qui assemble les cellules. Il offre des communications à 25 Mo/sec et les liens sont larges de 16 bits. Une cellule peut simultanément recevoir et émettre un message (modèle 1-port, bidirectionnel du chapitre suivant). Le routage des messages entre cellules est du type wormhole.
- le B-net est un réseau 32 bits dédié à la diffusion d'un message d'un nœud à tous les autres (Broadcast). Il connecte les cellules par paquets de 32 à un anneau. Chaque paquet a la structure d'un arbre dont la racine est un nœud de l'anneau. Il offre une bande passante de 50 Mo/sec qui permet de le considérer comme un bus partagé malgré sa structure multi-couches.
- Le S-net permet d'établir des points de synchronisation entre les cellules. C'est un arbre dont les arêtes sont des lignes porteuses de signaux et les nœuds sont des portes logiques ET. Chaque cellule écrit son signal sur le S-Net et reçoit en retour le ET de toutes les cellules.

Les machines qui ont plein de petites LED clignotantes sont des Hollywood-set, à opposer aux "frigos" des années cinquante comme les serveurs classiques.

Cette machine exploite efficacement les caches des processeurs RISC en permettant d'échanger directement leur données au routeur, sans utiliser d'accès DMA [63].

- **IBM**

  Pour le moment IBM ne se lance pas dans la construction d'un super ordinateur massivement parallèle, mais s'y intéresse fortement. Les ingénieurs de chez IBM travaillent sur des logiciels en collaboration avec ceux de Thinking Machines Corporation.

  De plus, IBM développe actuellement des réseaux de stations de travail RS/6000, reliées entre elles par des interconnexions en fibre optique (Cluster RS/6000). Le RS/6000 est un processeur RISC très performant qui équipe les stations unix (AIX) du constructeur.

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

  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 2D. 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 4 liens de communication parallèles bidirectionnels confère à la cellule une bande passante totale de 320 Mo/s (4 x 2 x 40 Mo/s) [7]. Vingt canaux virtuels, multiplexés sur les ports physiques, peuvent être gérés en parallèle. Le routage est de type "wormhole" (source Patrice Quinton, Irisa).

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

- **Kendall Square Research**

  La KSR est une machine pouvant posséder de 32 à 1088 processeurs. Le réseau de processeurs est connecté en un anneau d'anneaux (voir figure 1.7). 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 connectés en un anneau. Sur chaque contrôleur vient se connecter un anneau de niveau 1. 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. Les processeurs sont construits par Sharp. Ils possèdent une architecture RISC 64 bits et chacun est couplé à : 3 coprocesseurs, une unité de calcul flottant (FPU), un IPU (Integer and logical Operation Unit) et un coprocesseur gérant les entrées/sorties avec l'extérieur. Chaque processeur possède 32 Mo de mémoire cache (mémoire très rapide). Les communications se font par accès à des pages de 16Ko divisées en sous-pages de 128Ko. 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 sous-page.

- **nCUBE 2**

  Cette machine date du début des années 90; elle permet d'interconnecter jusqu'à 8192 processeurs assemblés en hypercube (13-cube) et développe une puissance maximale de 20 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 dispose de 32 Mo de mémoire. Le débit de chaque lien est de 2.22 Mo/s [54].

  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 de cette machine. 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". Quand un nœud veut diffuser un message à tous les autres, il envoie ce message à la destination -1, ce qui a pour effet de router le message jusqu'au nœud 0 qui utilise alors des facilités matérielles pour réaliser la diffusion. On doit

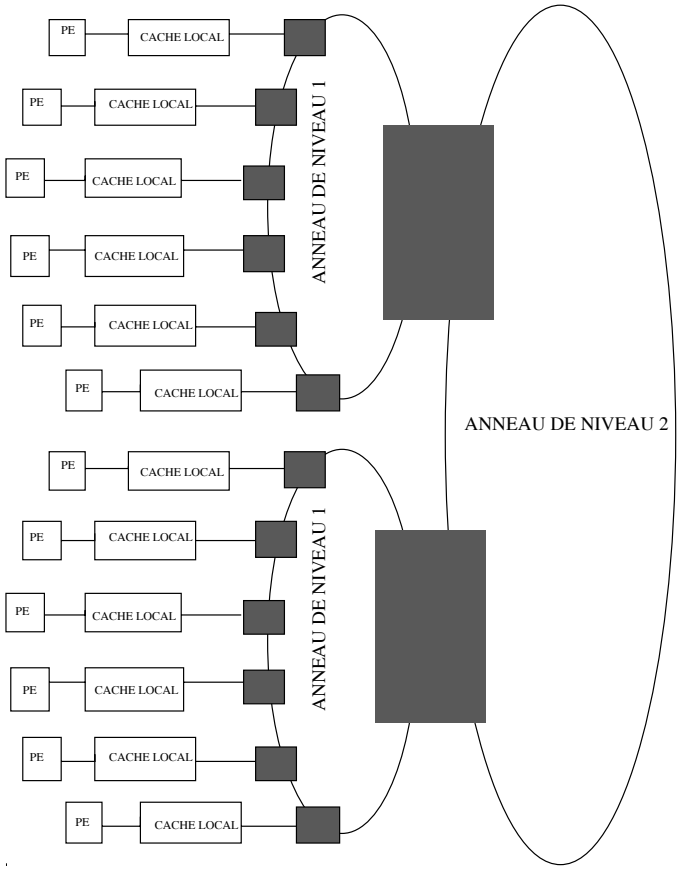

Figure 1.7: Anneau d'anneaux

noter que les diffusions "one-to-many" ainsi réalisée peuvent générer des interblocages (voir chapitre suivant pour les solutions à ce problème).

- **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'a 65 536 processeurs 64 bits. Comme sur son ainé, 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". Ce processeur sera cadencé à 50 Mhz et développera 100 MFlops et 3 GIPS. Chaque processeur disposera de 40 canaux de communication et d'1 Go de mémoire locale. La machine dans sa version complète aurait une puissance de 6,5 TFlops.

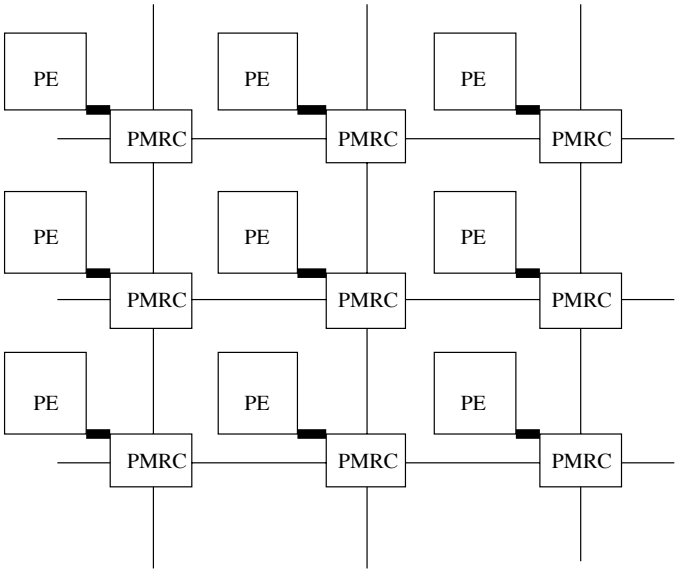

Figure 1.8: Réseau de la Paragon

En attendant cette machine, nCUBE annonce (septembre 1992) le nCUBE 2S qui offrira un gain de performance de 50% par rapport au nCUBE2.

- **Paragon**

  Cette machine est la version commerciale du projet Delta-Touchstone mené par Intel. Elle dispose de 56 à 2048 nœuds connectés physiquement en grille 2D (voir figure 1.8) et développe une puissance de crête de 4 à 150 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 128Mo, 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 200Mo/s. Le temps de latence est de 25 μs.

- **Projets autour du T9000**

  Enfin, l'arlésienne de Bristol, le T9000 d'Inmos et son circuit d'interconnexion, le C104, bien que prometteur au vu des annonces, tarde à entrer en phase de production. Il devrait permettre la construction de grands calculateurs MIMD européens et offrir les techniques de "wormhole", de canaux virtuels

et de routage aléatoire.

Il existe un programme européen pour construire une machine à base de T9000. Ce programme s'appelle GPMIMD; il regroupe les constructeurs Meiko, Parsys et Telmat. Cette machine serait constituée de T9000 pour gérer les communications, et probablement de i860 comme processeurs de calcul.

Un autre projet de machine réalisé cette fois par Parsytec associe le T9000 et le C104 [55]. 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. Evidemment, ce serait une machine MIMD à mémoire distribuée. Son architecture serait une grille 3D formée de 8x8x16 clusters (ou groupes). Chaque cluster formerait un nœud de la grille 3D, contenant 16 T9000 et 4 C104, où chaque T9000 aurait 1 lien connecté à chaque C104. Cette machine serait refroidie par eau.

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.

1.7.3 La dernière génération de machines SIMD

- **CM-2**

  La Connection Machine 2 [39] est une machine du constructeur Thinking Machines Corporation (TMC). Elle est de type SIMD. Elle est constituée de cartes identiques, regroupant 16 modules de 32 processeurs élémentaires 1-bit. Dans sa configuration maximale elle contient 65 536 processeurs élémentaires. 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'instructions et un bus d'adresses. Les circuits sont reliés entre eux par un réseau de type hypercube (de degré 12 pour la configuration maximale). Les liaisons internes à un groupe de 16 processeurs élémentaires sont réalisées par un réseau d'interconnexion complet. L'essentiel des applications testées sur la CM-2 étant de type scientifique, une unité flottante a été ajoutée afin d'accélérer les calculs sur les réels (réalisés dans le format IEEE simple précision). Sur les dernières versions de la machine, il est possible d'utiliser des processeurs travaillant avec le format double précision (64 bits) pour améliorer la précision. Le gain obtenu grâce à l'apport de l'unité flottante est de l'ordre de 20 par rapport au temps série sur les processeurs élémentaires .

  Les communications possibles sont de trois types :

  - a/ Les communications générales qui servent à envoyer un message d'un processeur à n'importe quel autre. Elles utilisent une unité de

communication interne à la CM-2, ce qui entraîne une perte sensible au niveau des performances.

- b/ Les communications NEWS (North,East,West,South) qui permettent d'envoyer un message sur un processeur voisin dans la topologie choisie (par exemple nord, est, ouest ou sud pour une grille 2-D). Les processeurs exécutant la même instruction, il n'existe pas de conflit de liens. Ces communications sont donc très rapides.
- c/ Les communications avec recombinaisons. Elles permettent de communiquer entre les processeurs d'une même dimension de la topologie choisie. Elles associent à la fois communications et opérations de recombinaisons binaires des messages. Ces communications utilisent les liens de l'hypercube reliant tous les circuits. Par exemple, elles permettent de réaliser un produit scalaire sur une ligne de matrice. Elles coûtent environ 10 fois plus de temps que les communications NEWS.

- **DAP**

  Le 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. Les communications se font selon les lignes et les colonnes de la grille. De plus, AMT a développé des librairies de communications et de calculs qui permettent au DAP d'être particulièrement adapté au traitement du signal et de l'image.

- **MasPar**

  La MasPar MP-1 est une machine de type SIMD dont les processeurs constituent une grille torique [9], chaque processeur étant relié à ses 8 voisins. 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. Sur cette machine il existe trois types de communication :

  - a/ 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. Les processeurs destinataires doivent être sur une des 8 directions définies par les liens.
  - b/ Les communications via un routeur permettent de faire communiquer entre eux n'importe quels couples de processeurs. Le routage est réalisé grâce à trois niveaux de crossbars.
  - c/ 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. Elle possède entre 1024 et 16384 processeurs 32-bits capables de réaliser 133 MIPS, et 12,5 MFlops.

Nom de la machine	Puissance de crête	Nombre de processeurs	Architecture de la machine	Topologoie du réseau de communication
Campus FX800	32 GFlops	800	MIMD	full connected
CM-5	262 GFlops	2048	MIMD	fat-tree et arbre binaire
Concerto	5 GFlops	64	MIMD	reconfigurable
Cray MPP	?	?	MIMD	grille 3D
CS-2	?	?	MIMD	?
AP1000	4 GFlops	1024	MIMD	tore
KSR	43 GFlops	1088	MIMD	anneau d'anneau
nCUBE2	27 GFlops	8192	MIMD	hypercube
nCUBE3	6,5 TFlops	65536	MIMD	hypercube
Paragon	300 GFlops	4096	MIMD	grille 2D
CM-2	40 GFlops	65536	SIMD	hypercube
DAP		4096	SIMD	grille 2D
MasPar	1.2 GFlops	16384	SIMD	tore 2D
Zephir		8192	SIMD	grille 3D

Figure 1.9: Les principales machines parallèles.

- **Zephir**

  C'est la première machine de Wavetracer [71]. Zephir est du type SIMD et possède jusqu'à 8192 processeurs. La topologie adoptée par Wavetracer est la grille 3D, 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 3D. Un des avantages de cette machine réside dans le déploiement de la grille 3D, afin d'obtenir une grille 2D. Avec 8192 processeurs on peut donc avoir soit une grille 3D ($16 \times 32 \times 16$) soit une grille 2D (64×128).

Un tableau récapitulatif des principales caractéristiques des machines parallèles est donné sur la figure 1.7.3.

1.7.4 Les grandes lignes actuelles

Les tendances qui se dégagent de ces nouvelles machines sont la généralisation du routage "wormhole" ou du type commutation de circuit, l'apparition de routeurs matériels performants, ainsi que la disparition des ordinateurs frontaux comme sur 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. On voit aussi 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 apparaît deux stratégies. La première consiste à chercher la plus grande puissance de calcul : c'est celle adoptée par les grands constructeurs tels Cray, Intel, Thinking Machine Corporation, etc.. La deuxième consiste à maximiser la puissance à prix constant : c'est le cas des machines à base de Transputer, Wavetracer, Alliant, etc...

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

1.8 Perspectives

Les constructeurs annoncent des performances de l'ordre d'une centaine de GFlops, mais sur des applications réelles, les machines que nous avons présentées ne dépassent guère que quelques dizaines de GFlops. Ceci est dû à la lenteur relative des communications entre processeurs, et au manque d'optimisation du code pour chaque processeur. En effet, les applications doivent rester portables pour éviter de récrire le code chaque fois que l'on change de machine. Ceci rend difficile l'exploitation des spécificités des processeurs sur lesquels on travaille.

Pour répondre à la demande croissante de puissance de calcul, un certain nombre de constructeurs tels que Cray, Intel ou Thinking Machines Corporation ont pour but de construire prochainement des machines atteignant les 3T (Tera Flops, Tera octets, Tera octets/seconde). Pour atteindre les 3T, on utilise d'une part un mixage des techniques SIMD et MIMD, et d'autre part on essaye de rendre les processeurs plus performants. Le CMOS n'a pas encore atteint ses limites (aujourd'hui on parle moins de l'arséniure de gallium (GaAs), le "tout optique" [34] ne semble pas recueillir beaucoup de suffrages), et on compte sur les supraconducteurs, la fibre optique, ou encore l'augmentation des possibilités d'intégration. Intel sait intégrer 4 processeurs élémentaires sur un seul substrat en laboratoire (Micro 2000), mais seulement deux en pratique (le P5 ou futur 586). Ces technologies restent encore expérimentales et sont très onéreuses. Cependant, Nec a réussi à produire avec ces techniques des processeurs développant 0.2 GFlops [73]. Une production à grande échelle devrait pouvoir réduire les coûts de fabrication.

Ces supercalculateurs ne remplaceront pas des calculateurs moins ambitieux mais efficaces pour les applications courantes. Ils seront dédiés à une classe d'applications qui nécessitent ces 3T : les prévisions météorologiques sur de longues périodes, la modélisation du génome humain, la chromodynamique quantique ou encore la dynamique des battements du cœur (actuellement 1 jour de CPU Cray modélise 1 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 doit 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] S.G. Akl. *The Design and Analysis of Parallel Computers.* Prentice-Hall, 1989.

[2] Alliant. *The CAMPUS/800 massively parallel supercomputer*, 1991.

[3] G. Almasi and A. Gottlieb. *Highly Parallel Computing.* Benjamin Cummings Publishing Co., 1989.

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

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

[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 proceeding*, pages 186–193, 1991.

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

[9] T. Blank. The MASPAR MP-1 architecture. *IEEE Computers*, pages 20–24, 1990.

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

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

[12] T. Champion and B. Tourancheau. Tnode: document utilisateur. Technical report, LIP-IMAG 89-02, ENS Lyon, France, 1989.

[13] Henri-Pierre Charles. Le processeur i860. Technical Report 90-02, LIP - IMAG, February 1990.

[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, California, 1989.

[15] O. Collins, F. Pollara, S. Dolinar, and J. Statman. Wiring Viterbi decoder (splitting de Bruijn graphs). Tda progress report 42-96, Jet Propulsion Laboratory, Pasadena, California, 1988.

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

[17] Thinking Machine Corporation. *CM5*, 1992.

[18] R. Cypher. Theoretical aspects of VLSI pin limitations. TR 89-02-01 (appeared in Proc. of the Sixth MIT Conf. on Advanced Research in VLSI), Dpt. of Comp. Sc., University of Washington, 1989.

[19] 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.

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

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

[22] 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.

[23] 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.

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

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

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

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

[28] J. J. Dongarra. Performances comparées de 80 ordinateurs sur des programmes FORTRAN. *Technique et science informatique*, pages 355 – 360, 1984.

[29] J. J. Dongarra. The LINPACK benchmark: an explanation. In supercomputing, editor, *first international conference*, pages 456 – 474, 1988.

[30] J.J. Dongarra and I.S. Duff. Advanced architecture computers. Technical report, The university of Tennessee, 1990.

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

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

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

[34] Dror G. Feitelson. *Optical Computing.* The MIT Press, 1988.

[35] J. T. Feo. An analysis of the computational and parallel complexity of the livermore loops. *Parallel computing*, pages 163 – 185, 1988.

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

[37] J. D. Gee, M. D. Hill, D. N. Pnevmatikatos, and A. J. Smith. Cache performance of the SPEC benchmark suite. Technical 1049, University of California at Berkeley computer science division, 1991.

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

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

[40] R.W. Hockney and C.R. Jesshope. *Parallel Computers 2, Architecture, Programming and Algorithms.* IOP Publishing ltd, 1988.

[41] K. Hwang and F.A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, 1984.

[42] Sun Microsystem Inc. *The SPARC architecture manual*, 1987.

[43] Texas Instruments Inc. Texas instruments parallel-processing floating-point *dsp.* Documentation, 1990.

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

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

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

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

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

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

[50] Inmos Limited. Special T9000. *La lettre du Transputer*, 1991.

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

[52] INMOS Ltd. *OCCAM2: Manuel de référence.* Masson, 1989.

[53] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

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

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

[56] D. Pountain. Virtual channels: The next generation of Transputers. *Byte*, April:E&W3–12, 1990.

[57] Rojo Rakotozafy. La boudouze et le coucatre. Personal communication, 1989.

[58] 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 1990, 1990.

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

[60] C. L. Seitz, W. C. Athas, C. M. Flaig, Martin A. J., J. Seizovic, C. S. Steele, and W-K and Su. The architecture and programming of the ametek serie 2010 multicomputer. In ACM press, editor, *Proceedings of the third conference on hypercube concurrent computers and applications*, volume 1, pages 33–36, jan. 1988. Pasadena, California (USA).

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

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

[63] T. Shimizu, T. Horie, and H. Ishihata. Low-latency message communication support for the AP1000. In *The 19th International Symposium on Computer Architecture*, volume 20-2, pages 288–297. ACM SIGARCH, 1992.

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

[65] J. Statman, G. Zimmerman, F. Pollara, and O. Collins. A long constraint length VLSI Viterbi decoder for the DSN. Tda progress report 42-95, Jet Propulsion Laboratory, Pasadena, California, 1988.

[66] H.S. Stone. *High-Performance Computer Architecture.* Addison-Wesley, 1987.

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

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

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

[70] L.G. Valliant. General purpose parallel architectures. Tr-07-89, Harvard University, 1989.

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

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

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

Chapitre 2

Modèles de communications et routages

2.1 Introduction et motivations

Le temps d'exécution d'un programme parallèle est la somme du temps de calcul et du temps des communications entre les processeurs. Ainsi, les performances dépendent directement de l'équilibre entre le temps de calcul et le temps des communications. La topologie du réseau et les protocoles d'échange sur ce réseau doivent permettre d'alimenter en permanence les unités de traitement scalaires et flottantes des processeurs, afin de minimiser le temps total d'exécution des programmes.

De nombreux articles ont déjà été publiés sur le sujet des communications. Une étude approfondie de ces articles montre que les hypothèses utilisées sont souvent différentes, ce qui rend difficile une quelconque comparaison de l'efficacité des solutions proposées. Nous allons introduire ici les modèles courants et les problèmes classiques, en insistant sur les modélisations les plus proches des machines actuelles. En effet, de nombreuses études utilisent un modèle issu du téléphone, et les résultats associés sont d'un intérêt limité dans le cadre des machines MIMD.

2.2 Modélisation du réseau d'une machine MIMD

2.2.1 Modélisation d'un nœud

Une machine MIMD se présente habituellement sous la forme d'une armoire contenant un ensemble de cartes. Sur ces cartes se trouvent des modules, ou nœuds, qui assemblent les processeurs élémentaires, leurs mémoires externes et les éventuels composants chargés des communications. Souvent, l'armoire est connectée à une station de travail conventionnelle, appelée hôte (une station Sun par exemple).

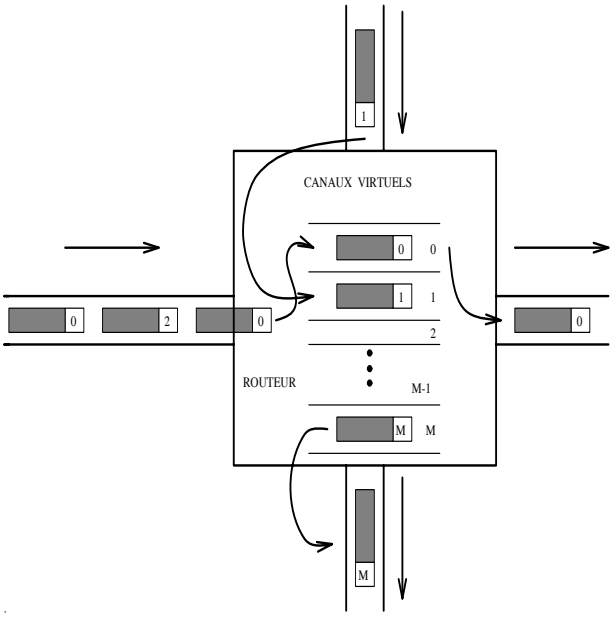

Figure 2.1: Multiplexage des liens du réseau.

Ici nous considérons une machine avec des nœuds du type de celui représenté sur la figure 1.1. On y trouve des unités de traitement scalaires et flottantes, des caches, une interface avec la mémoire, plusieurs interfaces de liens avec pour chacune des systèmes DMA *(Direct Memory Access)* indépendants. Chaque interface DMA peut accéder aux données contenues dans la mémoire ou dans les caches, sans ralentir les unités de traitement ni les autres interfaces, grâce à un bus qui lui est propre. Le T9000 d'Inmos [20] est construit sur le modèle présenté.

Les nœuds communiquent avec leurs voisins en échangeant des messages sur des *canaux*. Un canal est une liaison point à point, unidirectionnelle, entre deux nœuds reliés par un lien physique. Plusieurs canaux peuvent partager un même lien physique; on parle alors de *multiplexage*. Un lien peut être multiplexé (ou divisé) en M canaux; chaque canal étant associé à une *queue* – ou file d'attente – qui permet d'échanger les données sur celui-ci de manière indépendante (voir figure 2.1).

Les communications entre des processus qui s'exécutent sur des processeurs distants sont assurées par un *routeur* implanté de façon locale sur chaque nœud. Le routeur choisit, pour chaque message qui arrive sur un des canaux d'entrée du processeur, un des canaux de sortie, en appliquant *une fonction de routage*. Le routage par tables n'est pas couramment utilisé dans les multiprocesseurs, alors qu'il l'est souvent dans les grands réseaux planétaires (comme pour le routage du courrier électronique sur Internet). En général, la fonction de routage est basée sur une arithmétique simple et peut être implantée avec une mémoire programmable (en technologie CMOS) sur le nœud, offrant ainsi un délai de seulement quelques dizaines de nano-secondes pour le passage d'un message.

2.2.2 Les contraintes de communication

Classiquement, les architectures MIMD sont modélisées 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 physiques de communication entre les nœuds. Si un lien est monodirectionnel, on le modélise par un arc, mais en général les liens sont bidirectionnels car de nombreux algorithmes utilisent des communications entre voisins, et cela évite aux messages d'emprunter des chemins de longueur supèrieure à 1 lors de ce type d'échanges.

Plaçons-nous dans le cas de deux processeurs p_1 et p_2 directement reliés par un lien de communication bidirectionnel.

On peut mettre en évidence différentes contraintes physiques :

1. 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, les liens sont dits *half-duplex.*

2. si deux messages peuvent circuler au même instant sur un même lien, l'un transitant de p_1 vers p_2 et l'autre de p_2 vers p_1, les liens sont dits *full-duplex.* En pratique c'est le cas le plus fréquent.

De plus, il est utile de caractériser les possibilités de communication de l'interface mémoire/lien de communication pour chaque processeur.

1. Si, lors d'une communication, chaque noeud ne peut gérer qu'un seul lien à la fois, les communications sont dites *1-port.* Des ralentissements vont se produire au niveau des processeurs, ceux-ci ne pouvant transmettre rapidement les messages. Ce schéma est appelé *processor-bound* ou *whispering* (littéralement *en murmurant*) dans la littérature anglaise. Exemple de machine : l'iPSC/1 d'Intel.

2. Au contraire, si chaque noeud peut utiliser simultanément tous ses liens, alors les communications sont dites Δ-*port*, où Δ fait référence au degré maximum des noeuds du réseau (on trouve la notation n-*port* dans le cas du n-cube); c'est maintenant le nombre de liens qui limite les communications. Ce schéma est appelé *link-bound* ou *shouting* (littéralement *en criant*) en anglais. Exemple de machine : les systèmes à base de Transputers (comme le T-node de Telmat ou la Computing Surface de Meiko), ou bien l'iWARP d'Intel.

3. Entre ces deux cas les plus fréquents, il y a celui où seulement k liens de communication sont simultanément utilisables. On se trouve alors dans un schéma k-*port.* Les communications sont limitées par le nombre maximum de canaux DMA pouvant s'établir sur le bus du processeur concerné. On trouve donc aussi l'appellation *DMA-bound.* Exemple de machine : FPS série T.

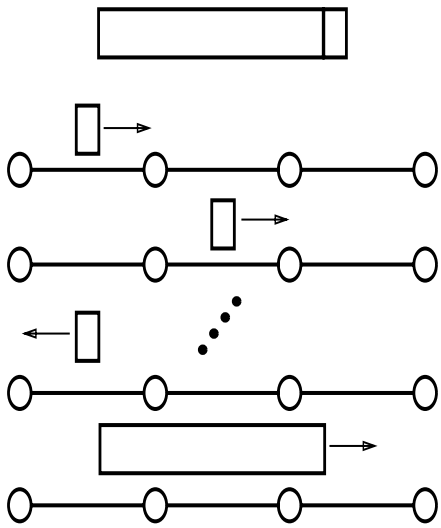

Figure 2.2: Commutation de circuit.

Certains auteurs ont introduit des notations abrégées pour se référer à ces modèles de communication [11]. F_1 et H_1 désignent respectivement les modèles full-duplex et half-duplex 1-port, le suffixe 1 indiquant qu'un nœud ne peut communiquer qu'avec un seul autre nœud à la fois. Les modèles full-duplex et half-duplex k-port sont notés F_k et H_k, où k indique le nombre maximum de liens qu'un processeur peut utiliser simultanément. Enfin, $F*$ et $H*$ désignent respectivement les modèles full-duplex et half-duplex Δ-port, dans lesquels un processeur peut utiliser tous ses liens à la fois.

2.3 Différents modes de commutation

Lors de la transmission d'un message entre deux processeurs non directement liés, le message doit être routé par des nœuds intermédiaires. On trouve une présentation des diverses techniques de commutation courantes dans l'introduction de [17].

- **Commutation de circuit :** c'est le principe du téléphone, on établit d'abord la ligne (ici cela consiste à réserver une suite de canaux), la conversation commence ensuite. Sur la figure 2.2, on observe l'établissement du circuit par la propagation de l'entête du message notée h, l'arrivée d'un accusé de réception, puis la transmission du message en un coup. D'autres communications qui voudraient emprunter une partie d'un circuit déjà établi sont bloquées jusqu'à la libération de la ligne. 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. Sur les iPSC/2 [24], un nœud est

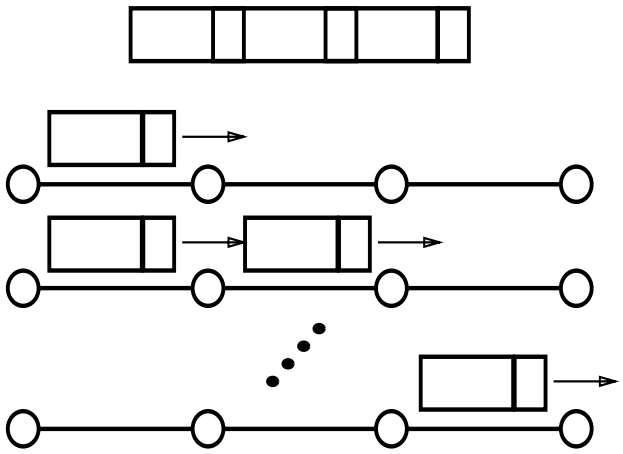

Figure 2.3: Commutation de paquets.

constitué d'un module comprenant le processeur de calcul (de conception identique à ceux des stations de travail), de la mémoire externe, et d'un routeur réalisé en technologie CMOS qui implante la gestion des canaux d'entrée/sortie et la fonction de routage.

Remarque: une erreur fréquente est de confondre cette technique avec celle du "wormhole" expliquée plus loin.

- **Commutation de messages** (*store-and-forward*) **:** les messages avancent dans le réseau vers leur destination en transitant dans les nœuds intermédiaires. A chaque étape, le canal emprunté est aussitôt libéré. Cette technique est connue sous le nom de store-and-forward dans le cadre des machines MIMD. Chaque nœud doit réserver un espace mémoire important pour stocker les messages en transit.

  Les Computing Surface de *Meiko*, les T-node et Mega-node de *Telmat*, ou les séries Volvox d'*Archipel* utilisent des routages de ce type (comme les autres machines à base de transputers).

- **Commutation de paquets :** les messages sont découpés en paquets de taille fixe. On bénéficie alors d'un "effet pipeline" (voir section suivante). Les paquets sont routés indépendamment les uns des autres et ils peuvent emprunter des chemins différents selon la charge du réseau. La capacité de stockage nécessaire sur un nœud se limite à la taille d'un paquet. On constate un surplus de communications engendré par l'adresse de destination collée à chaque paquet, au lieu d'une fois pour le message. Ceci est schématisé sur la figure 2.3.

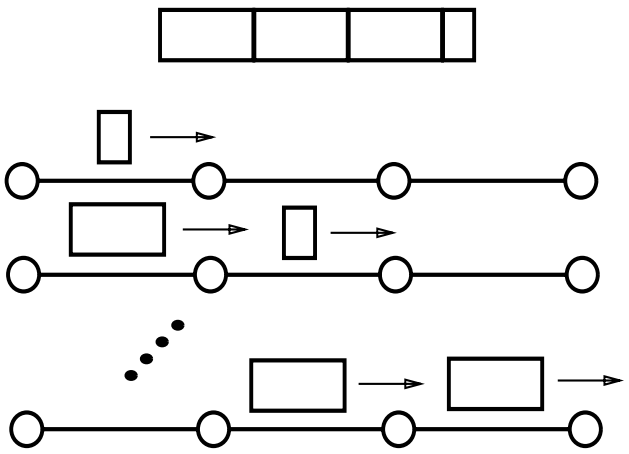

Figure 2.4: Routage wormhole.

- **Virtual cut-through et Wormhole :** les machines à mémoire distribuée les plus récentes ont abandonné le routage store-and-forward pour adopter une technique de type commutation de circuit communément appelée "wormhole". Ce nom est admis même si l'origine n'est pas claire ; fait-on référence à un ver ou bien aux "cosmic wormholes" (thèorie des "supercordes") des astrophysiciens ? [29, page 37].

  Dans le routage store-and-forward les messages (ou paquets) sont entièrement stockés dans la mémoire du processeur avant d'être transmis au processeur suivant. Dans le routage wormhole, les messages progressent dans le réseau de processeurs *flit* par flit – un flit, pour *flow control digits*, est égal à la taille de la queue d'un canal – le premier flit contenant l'adresse de destination.

  La tête du message avance d'un canal chaque fois que cela est possible. Le reste suit, libérant la queue du dernier canal qui stocke la fin du message, alors disponible pour un nouveau message. Dans ce modèle, les étapes intermédiaires entre la source et la destination consistent en l'établissement d'un circuit virtuel. Un message peut commencer à être reçu avant que l'émission ne soit terminée. De la même façon, si le message est suffisament court afin d'être stocké sur le circuit virtuel, la source est libérée avant la réception du dernier flit. 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"). Ceci interdit d'utiliser des "tampons ordonnés" pour résoudre les problèmes d'interblocage, comme dans le cas des routages store-and-forward.

  Cette technique est propre aux réseaux de processeurs où la distance entre deux processeurs élémentaires permet de négliger les erreurs de transmission. Les flits sont envoyés sans attendre d'accusé de réception, et passent directement du canal de sortie d'un nœud au canal d'entrée du nœud suivant.

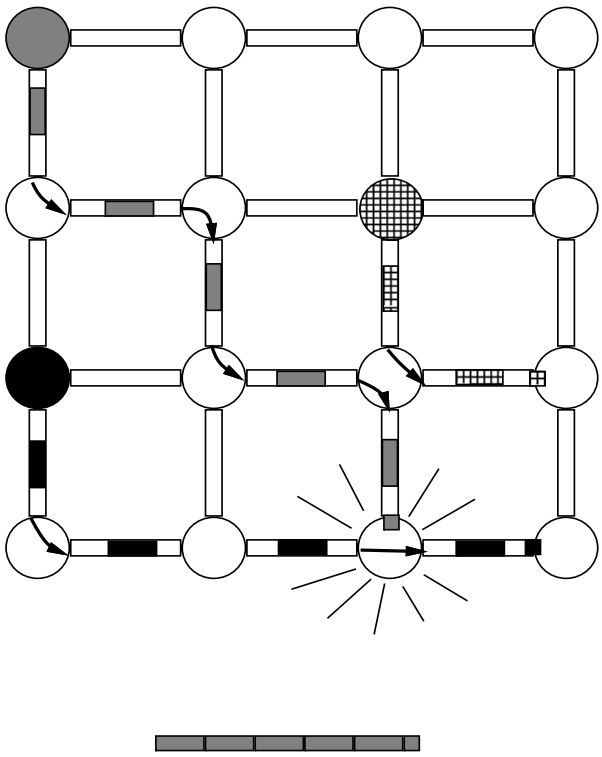

Figure 2.5: Routage wormhole sur une grille.

Ainsi il n'y a pas de stockage intermédiaire du message coûteux en temps [29]. Seul le flit routé ayant besoin d'être stocké, la fonction de routage peut être alors assurée sans accéder à la mémoire du processeur de calcul. On remarque que dans ce modèle les tampons sont de taille réduite, ce qui facilite l'implantation en VLSI du routeur.

On trouve une première étude d'une technique semblable dans [17], c'est le *virtual cut through*. Le routage est identique en tout point au wormhole sauf quand un nœud ne peut pas faire suivre les flits (les canaux de sortie étant déjà utilisés). Dans ce cas, le message est stocké en attendant la libération du canal de sortie. Le routeur doit donc disposer de tampons importants, ce qui ne permet pas de l'intégrer facilement avec le processeur. Cette technique n'a pas été implantée à notre connaissance alors que le wormhole fait l'objet d'études et de réalisations récentes. Le wormhole a été choisi par *Inmos* pour la nouvelle série des transputers T9000 et des crossbars C104.

La figure 2.5 illustre le routage wormhole sur une grille de processeurs. Le message issu de 00 et à destination de 33 est bloqué en 32 par le message en provenance de 20 qui utilise déjà le canal en sortie de 32. En revanche l'autre message n'est pas perturbé. Les flèches dans les nœuds indiquent la position des commutateurs.

Remarque: deux fonctions qui nous semblent intéressantes n'ont pas encore été implantées dans les machines :

- un mode diffusion, qui permettrait au routeur de transmettre sur tous ses canaux en sortie un flit accepté en entrée. On routerait alors sur des arbres au lieu de router sur des chemins.
- un mode transparence, dans lequel le processeur de calcul pourrait lire les flits qui transitent par le routeur auquel il est rattaché.

2.4 Modélisation du temps de communication

Maintenant que nous avons donné la liste des différentes hypothèses possibles pour modéliser les communications, il nous reste à quantifier le temps de transfert d'un message entre deux processeurs.

2.4.1 Communications entre voisins

Nous commençons par donner ce temps entre deux processeurs voisins, c'est à dire plaçés sur des nœuds directement connectés par un lien.

2.4.1.1 Modèle du temps constant

Ici une communication entre deux processeurs voisins "coûte" une unité de temps.

$$T_{\text{Voisin à voisin}} = 1$$

Dans ce modèle, les messages peuvent être découpés et recombinés sans affecter le temps de transfert. C'est le modèle classique de la théorie des graphes, ou encore du téléphone.

2.4.1.2 Modèle du temps linéaire

Le temps de transmission d'un message de longueur L entre deux processeurs voisins est la somme :

- d'un temps d'initialisation (ou *start-up*) β correspondant à des initialisations de registres mémoire, ou à des procédures d'envoi/accusé de réception.
- et d'un temps de propagation $L\tau$ directement proportionnel à la taille L du message communiqué.

$$T_{\text{Voisin à voisin}} = \beta + L\tau$$

τ est le temps de propagation d'une unité de message, souvent un bit ou un octet. La bande passante d'un lien est donc $\frac{1}{\tau}$.

Les valeurs de β et τ pour les machines courantes sont rassemblées dans le tableau suivant :

	β μsec	τ $\mu sec/octet$
iPSC/1	1000	1,000
nCUBE	446	2,500
FPS T40	830	1,430
Mega-Node	4,85	1,125
iPSC/2	350	0,200
CM-2	0,0	0,100
iPSC/860	136	0,384
Meiko	250	1,000
Ametek 2010	0,084	0,035
Fujitsu AP 1000	0,040	0,040

Pour les iPSC ([5] et [4]) le temps de propagation est supérieur si les messages sont de plus d'1K octets dans le cas de l'iPSC/1, et 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. Les valeurs du start-up ne sont pas directement comparables car pour les iPSC et le Mega-Node elles proviennent de mesures en environnement logiciel (avec des noyaux systèmes sur les nœuds).

La plupart des résultats ici exposés sur les communications globales se réfèrent à l'un des deux modèles présentés ci-dessus. Notons que si les messages envoyés sont de taille petite ou si $\beta \gg \tau$ les modèles temps constants ou linéaires sont équivalents.

Cependant d'autres modélisations du temps de communication voisin à voisin ont été étudiées. Voici une liste non exhaustive de différents autres modèles.

- Bertsekas et al. [3] ont étudié les communications globales dans l'hypercube en supposant que le transfert d'informations sur les liens se fait par paquets de taille fixe. On ne peut ni découper, ni 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 :

  $$T_{\text{Voisin à voisin}} = \left\lceil \frac{L}{m} \right\rceil m\tau$$

  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 [33] 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 :

$$T_{\text{Voisin à voisin}} = L\tau$$

où L est la longueur du message à envoyer, et $\frac{1}{\tau}$ est la bande passante des liens.

- Fraigniaud [10] a étudié les communications dans l'hypercube en supposant, suite à des expérimentations sur machines à base de Transputers, 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 souvent payer le prix de la gestion logicielle du parallélisme des liens. Le modèle est de la forme :

$$T_{\text{Voisin à voisin}} = \beta_k + L\tau_k$$

où β_k et τ_k sont respectivement les temps d'initialisation et de propagation lorsque le processeur communique sur k liens.

2.4.2 Communications à distance d

Si au lieu d'être envoyé à un nœud voisin, le message est envoyé à un nœud qui se trouve à distance d, on obtient dans le cas du modèle store-and-forward :

$$T_{\text{Processeurs à distance } d} = d(\beta + L\tau)$$

Dans le modèle commutation de circuit ou wormhole, transmettre un message à distance d n'est pas équivalent à répéter d fois un processus de communication aux voisins. La modélisation communément admise est :

$$T_{\text{Processeurs à distance } d} = \alpha + d\delta + L\tau$$

où L est la longueur du message à envoyer, $\frac{1}{\tau}$ la bande passante des liens, et α un temps de start-up dû au processus qui envoie le message. Le délai δ est un temps lié aux commutations des routeurs sur chaque nœud intermédiaire.

2.4.2.1 Technique du pipeline dans le mode Δ-port

Cette technique a été introduite dans [28] et [32].

Pour tenter de réduire le temps de transmission du modèle store-and-forward on peut découper le message en paquets, et les transmettre à la queue leu leu comme sur la figure 2.6 où un message est découpé en paquets de taille B. On suppose ici que L est divisible par B et que $\frac{L}{B}$ est un entier (m sur la figure 2.6). Le temps de transmission que l'on obtient alors est :

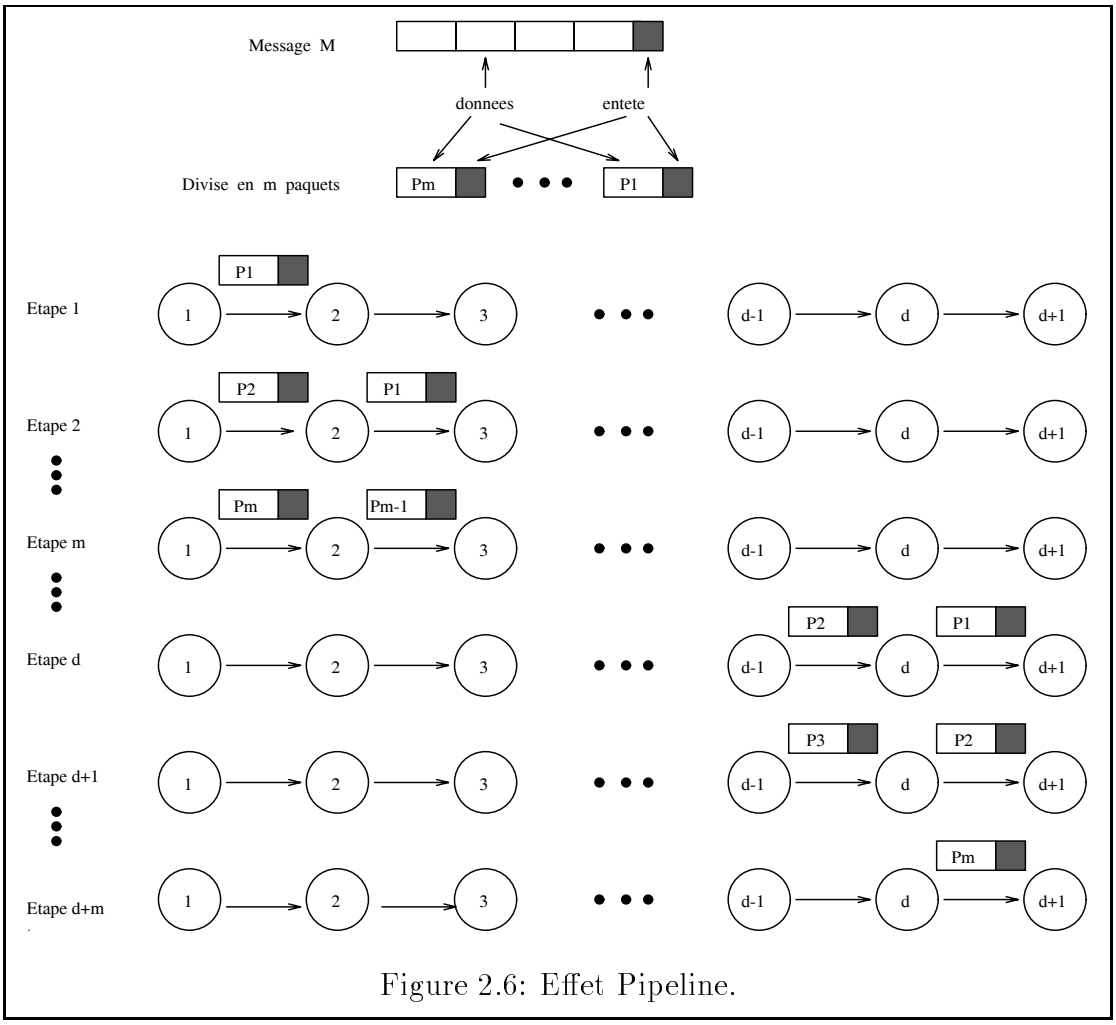

Figure 2.6: Effet Pipeline.

$$\begin{aligned} T_{\text{Processeurs à distance } d} &= \underbrace{d(\beta + B\tau)}_{\text{pour le } 1^{er} \text{ paquet}} + \underbrace{(\frac{L}{B} - 1)(\beta + B\tau)}_{\text{réception des autres}} \\ &= (d + \frac{L}{B} - 1)(\beta + B\tau) \end{aligned}$$

Proposition 2.4.1 *La taille de paquet B qui minimise $T_{\text{Processeurs à distance } d}$ est $B_{min} = \sqrt{\frac{L\beta}{(d-1)\tau}}$. Le coût total pour cette taille de paquet est alors $\left(\sqrt{L\tau} + \sqrt{(d-1)\beta}\right)^2$.*

Preuve: il faut trouver le minimum de la fonction $f(B) = (d + \frac{L}{B} - 1)(\beta + B\tau)$. Le minimum est atteint pour $f'(B) = (d + \frac{L}{B} - 1)\tau - \frac{L}{B^2}(\beta + B\tau) = (d-1)\tau - \frac{L\beta}{B^2} = 0$ d'où $B^2_{\min} = \frac{L\beta}{(d-1)\tau}$ □

Ainsi, pour un L très grand, on obtient un protocole de communication store-and-forward en $L\tau + o(L)$, ce qui masque la distance.

2.4.2.2 Technique des chemins disjoints dans le mode Δ-port

On peut router les paquets sur des chemins arêtes disjointes quand cela est possible, et donc diminuer encore $T_{\text{Processeurs à distance } d}$ pour le modèle store-and-forward. De plus, si on utilise p chemins disjoints, on peut pipeliner p blocs de longueur $\frac{L}{p}$ sur chacun des chemins. En effet si on a p chemins arêtes disjoints de longueur au plus d', le temps de transmission devient :

$$T_{\text{Processeurs à distance } d} = d'(\beta + \frac{L}{p}\tau)$$

Pour des exemples d'utilisation de ces techniques sur l'hypercube se référer à [16] ou [9], et [23] pour les tores.

On peut combiner cette technique à celle du pipeline et on obtient :

$$T_{\text{Processeurs à distance } d} = \left(\sqrt{\frac{L}{p}\tau} + \sqrt{(d'-1)\beta}\right)^2 .$$

2.4.3 Comparaison du wormhole et du store-and-forward

Le tableau suivant résume les principales différences entre ces deux modes.

Actions	Store-and Forward	Wormhole ou Circuit
Contrôle	Logiciel	Matériel
Un nœud stocke	tout un message (ou paquet)	un flit
Blocage possible	sur un nœud	sur un chemin
Exemples de machines	T-node, iPSC/1, nCUBE, Cosmic-cube	iPSC/2,iPSC/860, T9000 nCUBE-2, MDP (MIT)
$T_{\text{Processeurs à distance } d}$	$d(\beta + L\tau)$ (sans pipeline)	$\alpha + d\delta + L\tau$
Ordre de grandeurs (exemples)	iPSC/1 : $\beta = 1000\mu s$ $\tau = 1\mu s/octet$	iPSC/860 : $\alpha = 350\mu s$ $\delta = 10\mu s, \tau = 0,2\mu s/octet$

Remarque: la technique du pipeline est difficile à exploiter en pratique. En effet, l'utilisateur programme à partir des routines de communication de base fournies par le constructeur. Ce sont donc ces routines qui doivent être optimisées, pas les applications qui les utilisent.

2.5 Problèmes de congestion

Quand plusieurs messages veulent emprunter le même chemin au même instant, on parle de congestion du réseau.

2.5.1 Caractéristiques des fonctions de routage

Dans l'étude des fonctions de routage on suppose souvent que l'on a un routage distribué : la décision de routage est locale au processeur intermédiaire traversé par un message. Ceci est à opposer au routage centralisé dans lequel un nœud maître gère toutes les routes. Ainsi, on n'a pas une vision globale du système, en particulier du trafic dans le réseau. Dans le cas le plus simple on a :

- **un routage statique :** il existe une fonction de routage qui associe à chaque couple (source, destination) un chemin unique. Cela pose notamment des problèmes de congestion quand le trafic n'est pas homogène. Un routage statique peut être vu comme un plongement du graphe complet K_n dans le graphe du réseau considéré. Ici K_n représente l'ensemble des routages possibles, et la dilatation obtenue est la longueur maximale des chemins. La congestion de ce plongement donne une mesure des goulots d'étranglement possibles.

Le plus court chemin d'un point à un autre, c'est encore de ne pas y aller [Le Chat (Géluk)]

- **un routage sur les plus courts chemins :** c'est le cas où la distance (nombre de processeurs intermédiaires) parcourue par un message est minimale.

2.5.2 Fonctions adaptatives

Pour faire face aux problèmes de congestion, on peut implanter des fonctions de routage adaptatives. Le chemin est construit dynamiquement en fonction du trafic. Par exemple, en chaque nœud on peut choisir le premier canal libre qui rapproche le message de sa destination. Une autre technique pour homogénéiser le flux des données sur l'ensemble du réseau est le routage aléatoire. C'est le cas du *routage universel*, qui consiste à se rendre d'abord à une destination tirée au hasard, puis à effectuer un routage statique vers la destination [34].

Cette possibilité sera offerte par le C104 d'Inmos, le crossbar qui permettra d'interconnecter des T9000 [20]. Toutefois, une telle politique de routage "casse" les propriétés de localité du réseau. Pour que les algorithmes puissent utiliser la topologie du réseau, il faut éviter cette technique.
Enfin, le *routage forcé* est une technique décrite dans [12] à propos de la machine Mega. Le but est de ne jamais bloquer un message, et dans le cas où un message arrivé à destination ne peut être consommé tout de suite, il est re-routé et sera accepté ultérieurement. Des techniques similaires sont mises en œuvre sur la CM-2 [15].

2.6 Problèmes d'interblocage

L'interblocage est bien souvent la mauvaise surprise rencontrée lors de l'écriture d'un premier programme parallèle. C'est le résultat d'une dépendance cyclique dans la gestion des communications entre processus, ces derniers se bloquant mutuellement. Nous allons montrer comment construire des routages non bloquants, notamment en utilisant la technique des canaux virtuels introduite dans [7], ou de réseaux virtuels comme dans [19].

Ces techniques assurent seulement qu'aucun interblocage d'un programme ne sera dû aux routines de communication.

On modélise le réseau d'interconnexion par un graphe $G = (V, E)$ où $V = P \cup M$ et $E = C \cup I \cup O$. L'ensemble $V = P \cup M$ représente les processeurs élémentaires. Plus exactement, P modélise les routeurs associés aux processeurs et M, isomorphe à P, les mémoires. C représente les canaux de communication entre les routeurs. De plus, chaque sommet x de P est relié à son homologue $x' \in M$ par un canal d'entrée i_x de I (inputs des routeurs) de x' vers x, et par un canal de sortie o_x de O (outputs) de x vers x'. Ces canaux modélisent les échanges entre le routeur et la mémoire du processeur. Bien souvent, pour ne pas alourdir la présentation, on utilise simplement le graphe $G = (P, C)$ des communications (voir la figure 2.7 pour la représentation d'un anneau de 4 processeurs et la figure 2.10 pour la représentation simplifiée d'un autre réseau de 4 processeurs). Sur la figure 2.7, on a $P = \{0, 1, 2, 3\}$, $M = \{0', 1', 2', 3'\}$, et $C = \{a, b, c, d\}$.

La fonction de routage est alors définie :

Définition 1 *Une fonction de routage sur G est une fonction $R : E \times V \longrightarrow E$ qui associe à un canal d'entrée $c = (u, v)$ (au sommet courant v) et une destination d, un canal de sortie $c' = (v, w)$ (parmi ceux issus de v).*

Notons qu'un message est émis par sa source s sur le canal i_s et est reçu par sa destination d sur le canal o_d.

Dans l'exemple de la figure 2.7, soit un message émis par 0 et à destination de 3. Il entre en 0 par le canal i_0, puis est envoyé sur a, c'est-à-dire $R(i_0, 3) = a$. De même $R(a, 3) = b$; $R(b, 3) = c$ et $R(c, 3) = o_3$.

Notons qu'ici la fonction de routage est effectivement distribuée : la route suivie par le message avant son arrivée sur le canal d'entrée n'a aucune influence sur le choix du canal de sortie. Dans un routage adaptatif, on aurait le choix de plusieurs canaux de sortie possibles.

La proprieté essentielle que doit vérifier une fonction de routage est d'être sans interblocages. Nous allons étudier cette fonction en prenant les mêmes hypothèses que Dally et Seitz [7] :

- Routage wormhole (la fonction est définie sur les canaux et non les sommets du réseaux, mais on peut facilement déduire les résultats pour le store-and-forward en termes de buffers sur les sommets)
- Routage déterministe et distribué

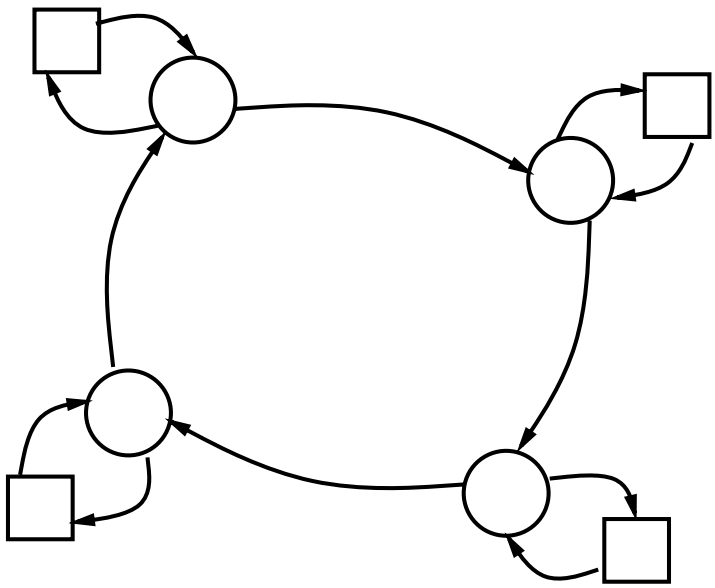

Figure 2.7: $G = (P \cup M, E = C \cup I \cup O)$.

Le cas du routage adaptatif sans interblocage a été étudié par Duato [8], [30] et [19].

2.6.0.1 Interblocages et canaux virtuels

On observe un exemple d'interblocage sur la figure 2.8. A l'étape 1, les processeurs 10, 02, 23 et 31 commencent l'émission de messages destinés respectivement à $22, 21, 11$ et 12. Le premier flit de chaque message qui contient l'adresse de destination est alors déposé dans la queue du canal de sortie convenable. A l'étape 2, les premiers flits ont avancé d'un canal vers leur destination, et le second flit de chaque message a pu être émis par chacune des sources. Ici, on suppose de façon non restrictive que les transmissions se font "en même temps". Après cette deuxième étape, on se trouve dans une situation d'interblocage : toutes les queues que les flits de tête voudraient emprunter sont occupées par d'autres flits qui ne peuvent pas avancer pour cette même raison. Il y a une dépendance cyclique entre les queues des canaux dans cette instance de routage.

Graphe de dépendance

Pour observer les propriétés de la fonction de routage R définie sur G, on définit un *graphe de dépendance des canaux associé à* R, noté $\mathcal{D}(G, R)$.

Définition 2 *L'ensemble E des sommets de $\mathcal{D}(G, R)$ sont les canaux de G, et l'ensemble U des arcs est :* $U = \{(c, c') \in E \times E \mid \exists\, x \in V, c' = R(c, x)\}$.

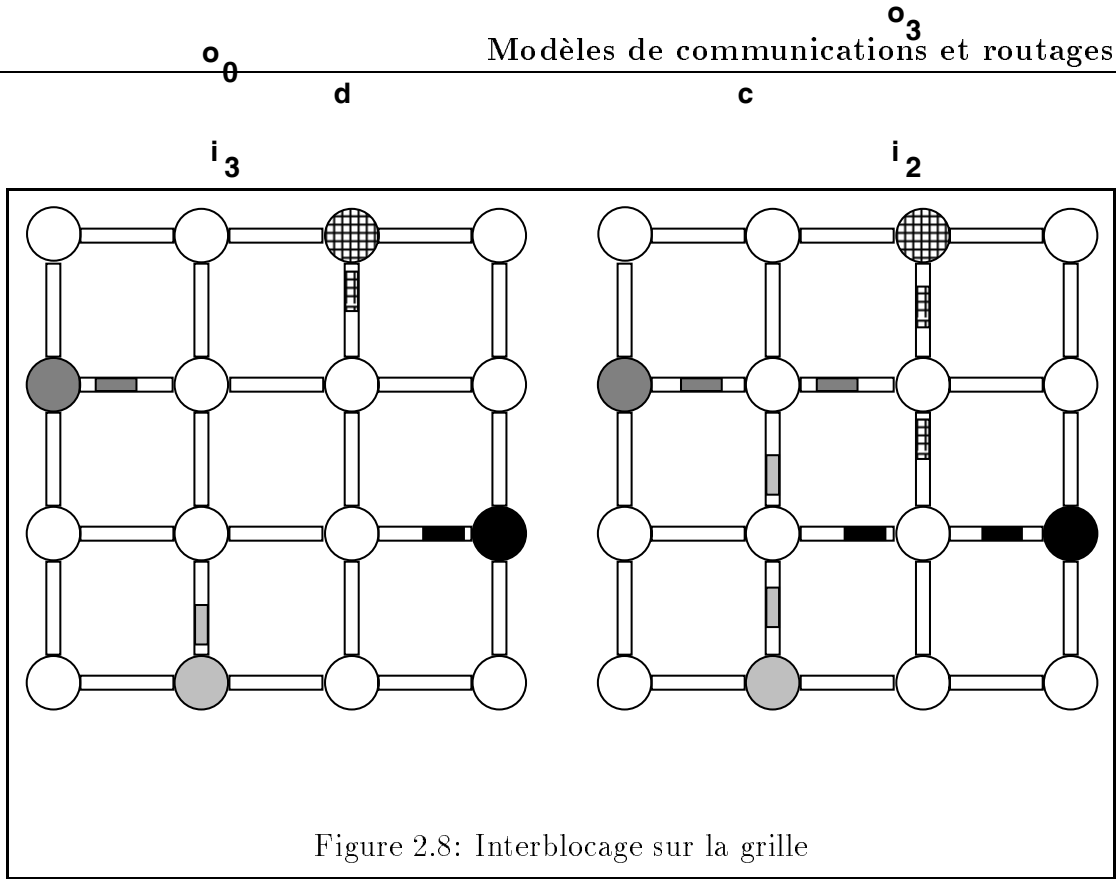

Figure 2.8: Interblocage sur la grille

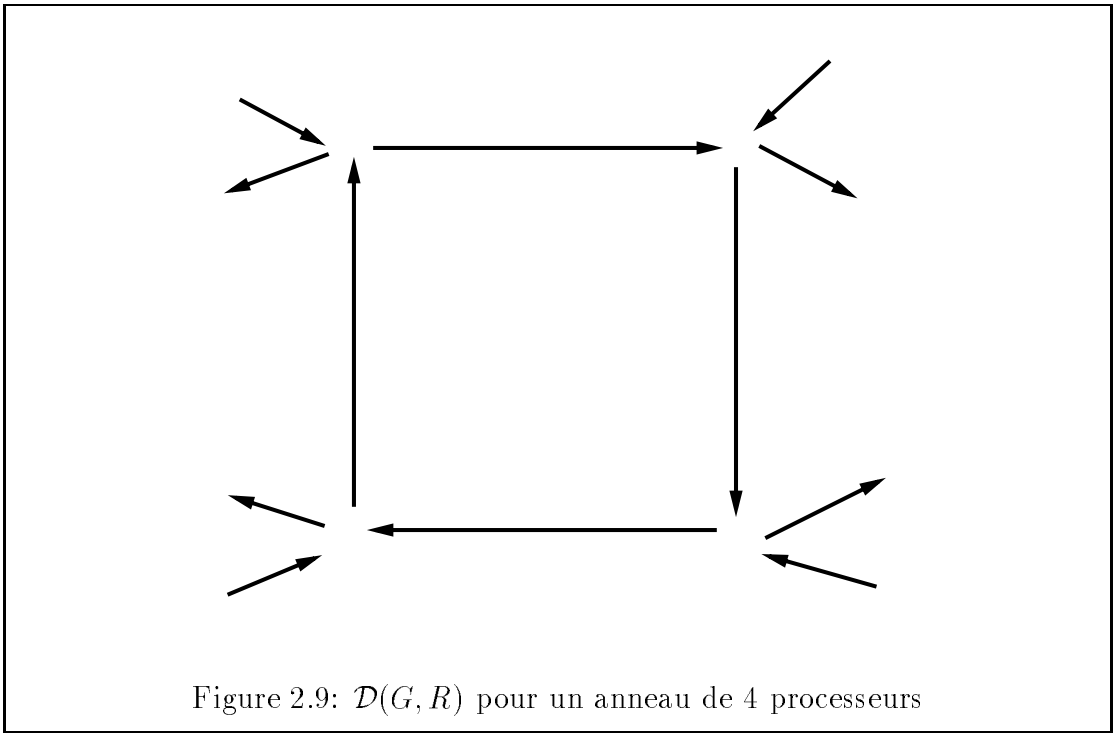

Figure 2.9: $\mathcal{D}(G,R)$ pour un anneau de 4 processeurs

Notons que la notion de graphe de dépendance est très liée à la fonction de routage choisie pour le réseau.
Remarque: $\mathcal{D}(G,R)$ est un sous-graphe partiel du graphe représentatif des arcs de G.

Un exemple de graphe de dépendance pour l'anneau de la figure 2.7 est donné sur la figure 2.9.

Théorème 2.6.1 *(*Dally-Seitz [7]*) Une fonction de routage R est non bloquante pour un réseau d'interconnexion G, si et seulement si le graphe de dépendance $\mathcal{D}(G,R)$ des canaux associé à R ne comporte pas de circuits.*

Preuve: (informelle)
$\Rightarrow$ On prend G tel que $\mathcal{D}(G,R)$ contienne au moins un circuit. Ce circuit est au moins de longueur 2 car il n'y a pas de boucles dans $\mathcal{D}(G,R)$. On peut alors construire une situation d'interblocage en remplissant les queues des canaux du circuit avec des flits à destination de nœuds à distance 2, et pour lesquels le premier canal à emprunter est sur le circuit.
$\Leftarrow$ On prend G tel que $\mathcal{D}(G,R)$ ne contienne pas de circuits. Rappelons qu'un graphe est sans circuit si et seulement si on peut ordonner ses sommets, de telle sorte qu'il existe un arc de i vers j seulement si $i < j$. Ce type de graphe est communément appelé DAG (pour Directed-Acyclic-Graph).
Considérons un tel ordre sur les sommets de $\mathcal{D}(G,R)$ et supposons qu'il puisse y avoir blocage. Soit alors c_n le plus grand canal dans cet ordre dont la queue est pleine. Tous les canaux qui peuvent recevoir des flits de c_n sont plus grands et donc ne sont pas saturés. Les flits de c_n ne sont donc pas bloqués ! Si c_n n'a pas de successeurs, les flits sont arrivés à destination et attendent d'être consommés par le nœud. Il n'y a pas d'interblocages possibles. □

Remarque: les canaux d'entrée et de sortie étant respectivement des sources et des puits, ils n'interviennent pas dans les circuits du graphe de dépendance. On peut donc définir un graphe de dépendance simplifié (associé au graphe simplifié $G = (P,C)$) ayant comme sommets les seuls canaux de C.
Notons que l'idée de graphe de dépendance était déjà appliquée au cas du store-and-forward dans [22]. Au lieu du graphe de dépendance des canaux, on avait un graphe de dépendance des tampons (ou *buffers*).

Le problème que l'on doit résoudre est : étant donné un graphe G et un routage R sur G, que faire si $\mathcal{D}(G,R)$ contient des circuits ?

2.6.0.2 Utilisation de canaux virtuels

Un lien physique ne peut "faire passer" qu'un flit à la fois, mais si ce flit appartient à un message dont la progression est stoppée plus en avant, aucun

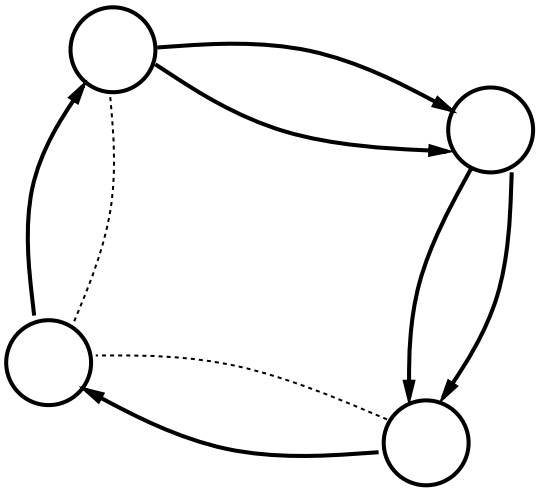

Figure 2.10: Canaux virtuels sur un anneau de 4 processeurs

autre message ne peut l'utiliser et ceci entraîne généralement le blocage d'un grand nombre d'autres messages, voire de tous.

Pour permettre aux messages qui ne sont pas concernés par l'embouteillage de suivre leur chemin, on peut allouer de nouvelles ressources sous la forme de queues supplémentaires ou *canaux virtuels* qui permettront de partager le lien physique en autant de canaux indépendants, comme sur l'exemple de la figure 2.1 où les liens sont multiplexés en M canaux virtuels (ici on parle de canaux virtuels car le lien physique supporte plusieurs canaux). Du point de vue du réseau d'interconnexion, cela revient à ajouter des arcs dans C; G est donc un multi-graphe orienté.

Pour décongestionner notre circuit de l'exemple 2.7, il suffit de diviser chaque lien x en deux canaux (virtuels) numérotés (i, x), avec $i = 0, 1$ et d'adopter la fonction de routage suivante : en un sommet courant on choisit le canal de sortie ayant le plus petit numéro possible supérieur au canal d'entrée (la relation d'ordre est lexicographique). Par exemple, pour envoyer un message de 0 à 3, on utilise i_0, puis $(0, a)$, $(0, b)$, $(0, c)$, et o_3. Pour envoyer un message de 3 à 2 on utilise i_3, puis $(0, d)$, puis $(1, a)$ car $a < d$, puis $(1, b)$ et o_2.

Notons que les canaux $(1, c)$ et $(1, d)$ sont inutiles et que $\mathcal{D}(G, R)$ est sans circuit (voir figure 2.11).

La technique générale mise en œuvre dans [7] est la suivante. A chaque canal c de C, on associe M canaux virtuels $c_0, \cdots c_{M-1}$ et on définit une fonction de routage associée de telle sorte que le graphe de dépendance des canaux associé soit sans circuit. Pour des raisons d'homogénéité du réseau, on multiplexe également chaque lien, bien que tous les canaux virtuels ne soient utiles dans certains cas.

Pour obtenir un graphe de dépendance sans circuit, on choisira d'étiqueter les canaux de manière à ce que tous les chemins correspondant à un routage possible dans le graphe donnent une suite d'étiquettes strictement croissante, comme dans

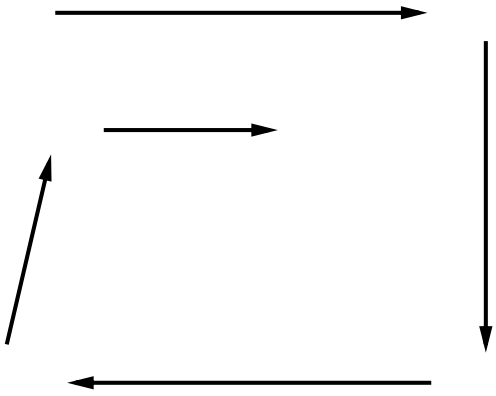

Figure 2.11: $\mathcal{D}(G,R)$ pour l'anneau de 4 processeurs avec des canaux virtuels

l'exemple de l'anneau de la figure 2.10.

2.6.0.3 Fonctions de routage des réseaux classiques

Pour les définitions concernant ces réseaux voir [18]. Par la suite, quand on parlera de n canaux virtuels, cela signifiera que l'on divise un lien physique du réseau en au plus n canaux, sans tenir compte de l'orientation.

Hypercube

L'algorithme de routage qui consiste à changer de dimension dans le cube dans un ordre défini à l'avance est sans interblocage [7]. En général on utilise l'ordre naturel croissant des dimensions, mais toutes les permutations des dimensions sont valides. Si on autorise plus de canaux virtuels, on peut alors choisir un grand nombre de chemins dans le cadre d'un routage adaptatif.

L'anneau unidirectionnel

Il suffit de prendre deux canaux virtuels par lien comme dans l'exemple de la figure 2.10 (voir Dally et Seitz [7]). On remarquera que les canaux de type (1,x) ne sont pas tous utiles.

L'anneau bidirectionnel[27]

Les figures 2.12 et 2.13 montrent le graphe de dépendance associé à un anneau de 8 processeurs avec un routage des plus courts chemins dans le cas des 2 canaux originels, puis avec 3 canaux virtuels. Sur la figure 2.13 on n'a pas representé la deuxième composante du graphe de dépendance (arcs de type *in*).

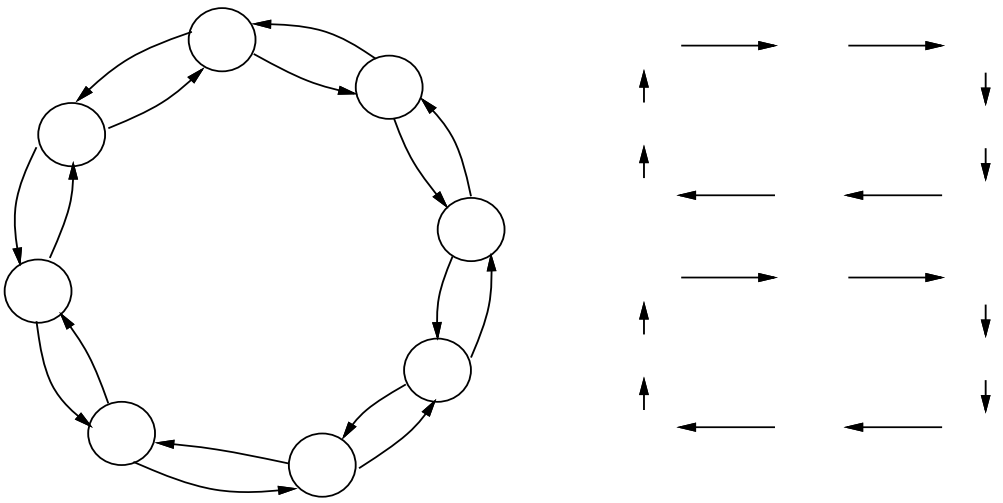

Figure 2.12: C_8 et son graphe de dépendance

Superposer les canaux du routage du cas unidirectionnel aboutirait à l'utilisation de 4 canaux virtuels. Une analyse plus approfondie des graphes de dépendance permet de n'utiliser que trois canaux virtuels.

On considère deux types de canaux, des canaux *out*, orientés dans le sens des aiguilles d'une montre, et des canaux *in*, orientés dans le sens inverse. Chaque type de canal est partitionné en deux classes de canaux virtuels a et b.

Alors le graphe de dépendance est formé d'une composante de type *out* et d'une composante de type *in* (on ne rebrousse jamais chemin dans un routage des plus courts chemins) qui sont les chemins suivants (donc sans circuit) :

$$in^a_{N-1}, in^a_{N-2}, \cdots, in^a_0 \cdots, in^b_{N-1}, \cdots, in^b_{N-\lfloor \frac{N}{2} \rfloor +2}$$

et

$$out^a_0, out^a_1, \cdots, out^a_{N-1}, out^b_0 \cdots, out^b_{\lfloor \frac{N}{2} \rfloor -2}$$

On utilise donc les canaux de type b $out^b_0 \cdots, out^b_{\lfloor \frac{N}{2} \rfloor -2}$ et $in^b_{N-1}, \cdots, in^b_{N-\lfloor \frac{N}{2} \rfloor +2}$

Comme $N - \left\lfloor \frac{N}{2} \right\rfloor + 2 > \left\lfloor \frac{N}{2} \right\rfloor - 2$, ces deux partitions sont disjointes, et il suffit alors d'une seule queue supplémentaire pour implanter ces canaux virtuels.

Grille torique et somme cartésienne

La grille torique est la somme cartésienne (aussi appelée produit cartésien) de 2 cycles. Il suffit de remplacer chaque lien par 2 canaux virtuels et d'étiqueter d'abord les canaux d'une dimension, puis ceux de l'autre dimension. D'une manière générale, dans une somme cartésienne, le nombre de canaux virtuels à

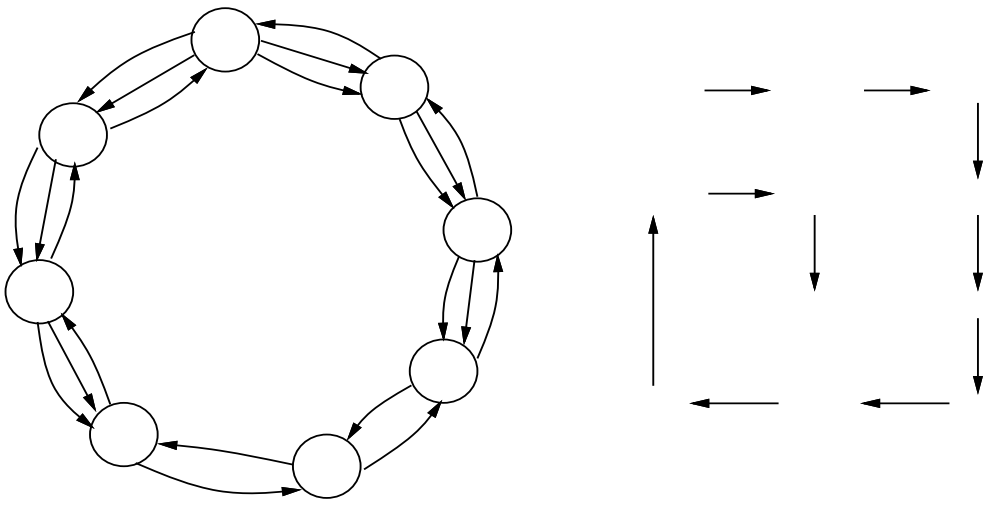

Figure 2.13: C_8 avec 3 canaux virtuels

utiliser est le maximum du nombre de canaux dans chaque dimension (voir le théorème ci-dessous). Ceci s'applique également aux grilles d-dimensionnelles (ou "k-ary-n-cube" dans [7]).

On peut ainsi étendre le résultat de l'anneau bidirectionnel au tore, et donc obtenir un routage avec 3 canaux virtuels, soit un de plus qu'en orienté, mais qui utilise tous les plus courts chemins du réseau.

Théorème 2.6.2 (Hilbert-Lukkien [14]) *Soient un graphe G_1 dont chaque lien est multiplexé en C_1 canaux virtuels et un graphe G_2 dont chaque lien est multiplexé en C_2 canaux virtuels. Soient R_1 une fonction de routage non bloquante sur G_1 et R_2 une fonction de routage non bloquante sur G_2, alors il existe une fonction non bloquante sur le graphe $G_1 \Box G_2$ avec un multiplexage des liens en max(C_1,C_2).*

Preuve:(version graphe de dépendance) les sommets de $G_1 \Box G_2$ sont notés (s_1, s_2), où s_1 est un sommet de G_1 et s_2 un sommet de G_2. On construit la fonction de routage R de la façon suivante : pour aller du sommet (x_1, x_2) au sommet (y_1, y_2), on applique d'abord R_1 pour aller de x_1 à y_1, ce qui nous mène en (y_1, x_2), puis on applique R_2 pour aller de x_2 à y_2, on atteint (y_1, y_2). Le graphe de dépendance des canaux associé à cette fonction, $\mathcal{D}(G_1 \Box G_2, R)$ a pour sommets les canaux de $G_1 \Box G_2$, notés $[(x_1, x_2), (y_1, y_2)]$, et on a des arcs du type :

- $[(x_1, a), (y_1, a)] \longrightarrow [(y_1, a), (z_1, a)]$ avec $(x_1, y_1) \longrightarrow (y_1, z_1) \in \mathcal{D}(G_1, R_1)$ (application de R_1).

- $[(a, x_2), (a, y_2)] \longrightarrow [(a, y_2), (a, z_2)]$ avec $(x_2, y_2) \longrightarrow (y_2, z_2) \in \mathcal{D}(G_2, R_2)$ (application de R_2)

- $[(x_1, a), (y_1, a)] \longrightarrow [(y_1, a), (y_1, b)]$ (transition de G_1 à G_2), mais jamais du type $[(a, x_2), (a, y_2)] \longrightarrow [(a, y_2), (b, y_2)]$, ce qui correspondrait à appliquer R_2 puis R_1.

Etiquetons les arcs du type $[(x_1, a), (y_1, a)]$ avec $(1, c_i)$, où c_i est l'étiquette de (x_1, y_1) dans $\mathcal{D}(G_1, R_1)$; ceux du type $[(a, x_2), (a, y_2)]$ avec $(2, c_j)$, où c_j est l'étiquette de (x_2, y_2) dans $\mathcal{D}(G_2, R_2)$; alors les chemins correspondant au routage proposé utilisent des arcs dans le sens des étiquettes croissantes. □

Cube-Connected-Cycle

Dally et Seitz [7] ont montré que 3 canaux virtuels suffisaient pour un routage qui n'est pas sur les plus courts chemins (on considère des cycles orientés). Ici, le thérorème du produit cartésien ne s'applique évidemment pas (en combinant les fonctions définies dans le cycle et dans l'hypercube), le cube-connected-cycle n'étant pas le produit du cycle par l'hypercube. Pour le routage dans le cube-connected-cycle, voir [21].

Graphes généraux

Si l'on a une notion de dimension dans G (ce qui est le cas des graphes de Cayley fortement hiérarchiques), il suffit de résoudre le problème pour chaque dimension, d'ordonner les canaux suivant l'ordre des dimensions, et d'associer une fonction de routage qui respecte cet ordre. Sinon, un premier problème à résoudre est : étant donné un graphe G et un routage sur les plus courts chemins R associé à G, quel est le nombre minimum M de canaux virtuels à utiliser à la place de chaque lien de G pour que $\mathcal{D}(G, R)$ ne contienne pas de circuits ? La proposition suivante donne une borne supérieure de ce nombre, et est en fait une adaptation de la technique de *buffer pool* utilisée en store-and-forward [26].

Proposition 2.6.3 *[2] Dans tout réseau de diamètre D on peut trouver une fonction de routage des plus courts chemins telle que, si l'on utilise D canaux virtuels, le graphe de dépendance des canaux associé à cette fonction est sans circuit.*

Preuve: considérons un routage des plus courts chemins cohérent (c'est-à-dire que si w est sur un plus court chemin de s à p, la portion de chemin entre w et p est le routage entre w et p). L'idée consiste à emprunter le $i$ème canal virtuel si l'on est sur le ième arc du chemin entre s et p. Plus formellement, la fonction de routage associe à un canal d'entrée (i, e), le canal virtuel de sortie $(i+1, f)$, f étant l'arc du plus court chemin entre le point courant w et la destination p . Si le canal d'entrée est i_s, on associe le canal de sortie $(1, f)$, f étant le premier arc sur le plus court chemin de s à p. Le graphe de dépendance de ce routage est sans circuit puisque ses sommets peuvent être partitionnés en D classes (correspondant

aux D canaux virtuels), et les arcs de ce graphe sont toujours orientés d'une classe i vers une classe $i+1$, avec $i = 1 \cdots D$. □

Remarque: ici on décompose le processus de routage en D étapes distinctes, mais parfois on a une décomposition plus avantageuse (moins de canaux virtuels) comme dans le cas du réseau *shuffle-exchange* [7], où une étape regroupe le franchissement de deux arêtes : une du type *shuffle* et une du type *exchange*.

En fait, si l'on n'impose pas un routage des plus courts chemins et si on ne tient pas compte de la charge des nœuds et des canaux, l'interblocage peut toujours être évité en n'utilisant que 2 canaux virtuels à la place de chaque lien physique. Ici c'est la contrainte sur R qui est relachée pour obtenir un M petit et indépendant de la taille de G.

Proposition 2.6.4 *[2] Pour tout graphe G, il existe un routage de G tel que, si l'on remplace chaque canal par au plus 2 canaux virtuels, le graphe de dépendance associé à ce routage est sans circuit et où tout chemin est de longueur au plus $2D$.*

Preuve: choisissons dans le cas orienté une arborescence et une antiarborescence de même racine r, et dans le cas non-orienté 2 arbres couvrants de même racine (on peut choisir 2 fois le même arbre). Etiquetons les canaux du premier arbre (ou de l'antiarborescence) avec 1 et ceux du deuxième arbre (ou de l'arborescence) avec 2. Ainsi pour les canaux communs on aura 2 canaux virtuels.

Le routage est ainsi défini : dans une première phase, on envoie le message à la racine r en utilisant le premier arbre (ou l'antiarborescence), c'est-à-dire les canaux de type 1. Dans une deuxième phase, on envoie le message de la racine r vers la destination p en utilisant les canaux de type 2. Le graphe de dépendance des canaux associé à cette fonction de routage est sans circuit. De plus, on peut toujours choisir les arbres ou arborescences de profondeur D (arbres des plus courts chemins obtenus par "Breadth first search" par exemple). Donc, tout chemin du routage est de longueur au plus $2D$. □

Notons qu'on peut dans certains cas ne pas avoir besoin de canaux virtuels. Il suffit que les deux arbres considérés (ou l'antiarborescence et l'arborescence) n'aient pas de canaux en commun. On peut aussi améliorer le routage en évitant d'aller jusqu'à la racine si un sommet intermédiaire dans la phase 1 est sur la branche à destination de p. Notons enfin que la racine est un point de congestion.

L'idée ci-dessus peut être améliorée en utilisant plusieurs racines, ce que l'on va faire pour les réseaux de *de Bruijn*. De plus, pour utiliser plus de chemins, on peut chercher des DAG plutôt que des arborescences.

Routage sans interblocage dans les de Bruijn

Les réseaux de *de Bruijn* (ou leurs généralisations) apparaissent comme d'excellents candidats pour les futures architectures parallèles [1, 25], car ils permettent d'interconnecter

un grand nombre de processeurs avec un petit diamètre et un degré fixé.

Il existe des algorithmes de tri très performants qui utilisent leur structure. Un tel réseau a été construit afin d'être utilisé comme décodeur de signal dans le cadre de la mission spatiale de la NASA Galileo [31],[6].

Le graphe de *de Bruijn* $B(d, D)$ est le graphe orienté dont les sommets sont les mots de longueur D sur un alphabet de taille d. Les arcs sont définis de la façon suivante : il y a un arc du sommet $(x_1, x_2, \cdots, x_D)$ vers le sommet $(x_2, x_3, \cdots, \lambda)$, λ étant une lettre quelconque de l'alphabet. Il est facile de voir que ce graphe a pour demi-degré extérieur et intérieur d et que son nombre de sommets est égal à d^D.

On peut, ayant défini les graphes de *de Bruijn* orientés, définir maintenant les graphes de *de Bruijn* non orientés : il suffit d'oublier les orientations des arcs, puis de supprimer les arêtes multiples, éventuellement créées. De tels graphes sont représentés ci-dessous. Pour une présentation plus détaillée de ces graphes, consulter le chapitre 4.

Les arcs de $B(d, D)$ peuvent être partitionnés en d arborescences (et aussi d antiarborescences) deux à deux disjointes de racines les sommets $(a, a, \cdots, a)$, $a \in \{1, \cdots, d\}$.

L'arborescence T_a, de racine $(a, a, \cdots, a)$, a pour arcs les canaux allant d'un sommet de niveau i (dont l'écriture commence par $(D - i)$ lettres a, la lettre suivante étant différente de a), vers un sommet de niveau $i + 1$, dont l'écriture commence par $(D - i - 1)$ lettres a. Chaque sommet a un degré sortant d sauf la racine qui est de degré sortant $d - 1$. On définit de même d antiarborescences AT_a. Un exemple pour $d = 2$ et $D = 3$ est donné sur la figure 2.14, où l'on a représenté les deux arborescences en traits pleins, et les deux antiarborescences en pointillés (elles s'obtiennent par symétrie).

On applique l'idée de la proposition 2.6.4, mais si le sommet de destination p est de la forme $(a, \cdots, a)$, on route dans la première phase vers la racine $(a, \cdots, a)$ en utilisant les canaux de AT_a, et dans la deuxième phase vers la destination p en utilisant T_a. Le chemin dans T_a sera de longueur au plus $D - 1$, ce qui fait une longueur totale de chemin au plus $2D - 1$. Ceci permet d'équilibrer les charges si on suppose les destinations uniformément réparties.

On peut aussi appliquer cette technique au réseau *shuffle-exchange*, qui est simulé par le *de Bruijn* (voir [18, section 3.3]).

2.6.1 Conclusion

Les communications des données entre les nœuds ne se limitent pas à un délai imposé aux programmes parallèles, dû au temps de transmission d'un signal sur un canal. Elles requièrent aussi une gestion du trafic, tant pour optimiser le débit du réseau que pour éviter les goulots d'étranglement, ou pire, les interblocages.

La gestion des messages sur les nœuds devenue indépendante du processeur de calcul, a permis non seulement de diminuer le coût des échanges de messages dans un algorithme, mais aussi de proposer un nouveau modèle pour la latence en mode

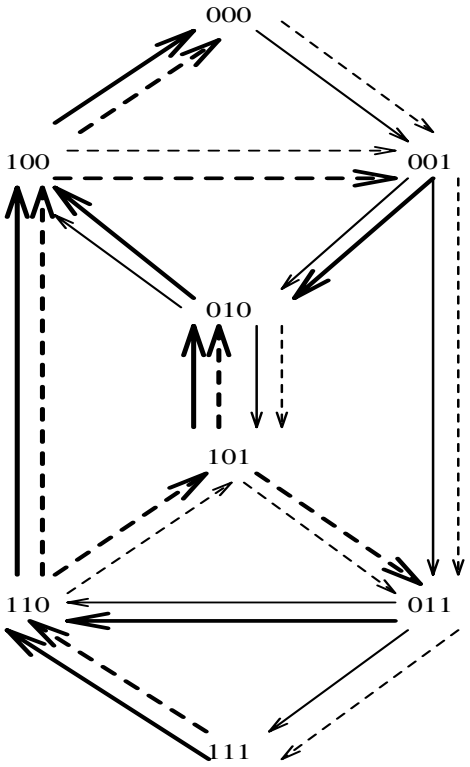

Figure 2.14: 2 canaux virtuels dans le *de Bruijn* binaire

commutation de circuit ou wormhole. L'analyse de certains schémas structurés d'échanges de données dans ce mode nous amène à reconsiderer les algorithmes écrits pour le mode store-and-forward, comme nous allons le voir dans la suite.

Bibliographie

[1] J.C. Bermond and C. Peyrat. de Bruijn and Kautz networks: a competitor for the hypercube? In J.P. Verjus F. André, editor, *Hypercube and Distributed Computers*, pages 279–294. North-Holland, 1989.

[2] J.C. Bermond and M. Syska. Routage wormhole et canaux virtuels. In Y. Robert M. Cosnard, M. Nivat, editor, *Algorithmique Parallèle*, pages 149–158. Masson, 1992.

[3] 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.

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

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

[6] O. Collins, F. Pollara, S. Dolinar, and J. Statman. Wiring viterbi decoder (splitting de Bruijn graphs). Tda progress report 42-96, Jet Propulsion Laboratory, Pasadena, California, 1988.

[7] 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.

[8] J. Duato. On the design of deadlock-free adaptative routing algorithms for multicomputers : design methodologies. In M.Rem E.H.L.Aarts, J.van Leeuwen, editor, *PARLE 91*, volume I, pages 390–405. Springer-Verlag, 1991.

[9] 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, E.N.S.L, 1990.

[10] P. Fraigniaud. Performance analysis of broadcasting in hypercubes with restricted communication capabilities. *Journal of Parallel and Distributed Computing*, (à paraître).

[11] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. Preprint, LRI and LIP-IMAG, France, 1991.

[12] C. Germain-Renaud. *Etude des mécanismes de communication pour une machine massivement parallèle: MEGA*. PhD thesis, Université de Paris-sud, Centre d'Orsay, 1989.

[13] F. Guidec. Performances des communications sur le T-Node. 547, IRISA, Sept 1990.

[14] P.A.J. Hilbers and J.J. Lukkien. Deadlock-free message routing in multicomputer networks. *Distributed Computing*, 3:178–186, 1989.

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

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

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

[18] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures : Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1991.

[19] D.H. Linder and J.C. Harden. An adaptative and fault tolerant wormhole routing strategy for $k-$ary $n-$cubes. *IEEE Transactions on Computers*, 40(1):2–12, 1991.

[20] INMOS Ltd. *The T9000 Transputer Products Overview Manual*. SGS-Thomson Microelectronics, 1991.

[21] D.S. Meliksetian and C.Y. Roger Chen. Optimal routing algorithm and other communication issues of the cube connected cycles. Research report, Syracuse University NY, 1990.

[22] P. Merlin and P. Schweitzer. Deadlock avoidance in store-and-forward networks-I : store-and-forward deadlock. *IEEE Transaction on Communication*, COM-28,Mar.:325, 1980.

[23] P. Michallon, D. Trystram, and G. Villard. Optimal broadcasting algorithms on torus. Technical report RR-872-I-, LMC-IMAG, 1992.

[24] S.F. Nugent. The iPSC/2 direct-connect technology. In G.C. Fox, editor, *Proceedings of 3rd Conference on Hypercube Concurrent Computers and Applications*, pages 51–60. ACM, 1988.

[25] D. K. Prhadhan. Fault-tolerant VLSI architectures based on de Bruijn graphs (Galileo in the mid nineties). In *Reliability of Computer and Communication Networks, DIMACS series 5*, pages 183–195, 1991.

[26] E. Raubold and J. Haenle. A method of deadlock-free resource allocation and flow control in packet networks. In *Proceedings of ICCC 1976, Toronto, Canada*, page 483, 1976.

[27] Rumeur. Communications dans les réseaux de processeurs, 1992.

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

[29] C.L. Seitz. Concurrent architectures. In R. Suaya and G. Birtwist, editors, *VLSI and Parallel Computation*, pages 1–84. Morgan Kaufmann, 1990.

[30] N.T. Son and Y. Paker. Adaptative deadlock-free packet routeing in transputer-based multiprocessor interconnection networks. *The Computer Journal*, 34(6):493–502, 1991.

[31] J. Statman, G. Zimmerman, F. Pollara, and O. Collins. A long constraint length VLSI viterbi decoder for the DSN. Tda progress report 42-95, Jet Propulsion Laboratory, Pasadena, California, 1988.

[32] 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.

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

[34] L.G. Valliant. General purpose parallel architectures. Tr-07-89, Harvard University, 1989.

Chapitre 3

Communications Globales

3.1 Schémas statiques de communication

Les processeurs communiquent en échangeant des messages, mais ces schémas de communication, pendant l'exécution d'un algorithme, dépendent fortement du problème. Heureusement, l'étude d'algorithmes classiques montre qu'il existe un certain nombre de "communications généralisées" qui apparaissent très souvent, par exemple en algèbre linéaire ou non-linéaire (voir [5, 13]), ou en traitement d'images (voir [21]). Ici, nous allons nous intéresser aux problèmes de communication suivants, que l'on rencontre le plus fréquemment.

- **La diffusion :** la diffusion est l'opération qui consiste à envoyer un message à tous les processeurs à partir d'un initiateur unique (on trouve différentes appellations : One to All, Broadcasting, etc ...).

- **L'échange total :** l'échange total est l'opération qui consiste à effectuer une diffusion à partir de tous les processeurs simultanément (on retrouve encore différentes appellations : "All to All", "Gossiping", "Total Exchange", ...).

- **La distribution :** lors de la distribution, un initiateur unique envoie un message différent à chacun des autres processeurs. Cette opération se différencie de la diffusion par le fait que les messages échangés ne sont plus les mêmes tout au long de la diffusion : lorsqu'un processeur a reçu le message qui lui était destiné, il n'a plus besoin de le relayer vers les autres. (Les autres appellations sont par exemple : "Personalized One to All", "Distributing", "Scattering", ...).

  L'intérêt de savoir faire une distribution est aussi de pouvoir faire son inverse, le *rassemblement* (ou encore "gathering"), c'est-à-dire récupérer en un processeur des données différentes, provenant de tous les autres.

- **La multidistribution :** tout comme l'échange total est une multi-diffusion, la multidistribution est l'opération qui consiste à effectuer une distribution

simultanément à partir de tous les processeurs : chaque processeur veut envoyer un message différent à chacun des autres ("Personalized All to All", "Complete Exchange", "Multi-Scattering", ...).

Bien évidemment, il existe quantité d'autres problèmes de communications globales comme par exemple le "one-to-many" où un processeur envoie un même message à un groupe particulier de processeurs. Une version particulière du one-to-many consiste pour un processeur à envoyer un message à tous ses voisins. En fait, toutes les permutations sur l'ensemble des nœuds du réseau peuvent motiver une étude.

Notations

Soit G un graphe connexe modélisant un réseau R et u un nœud de G. Le temps de diffusion du nœud u, $b_M(u)$, est le temps minimum nécessaire pour compléter la diffusion à partir du nœud u, avec le modèle M (M étant F_1,H_1,F_k,H_k,$F*$ ou $H*$). Le temps de diffusion du graphe G, $b_M(G)$, est le maximum de tous les temps de diffusion des nœuds de G, i.e. $b_M(G) = \max\{b_M(u) \mid u \in V(G)\}$. Des définitions similaires peuvent être données pour le temps de l'échange total (*gossiping*) $g_M(G)$, le temps de la distribution (*scattering*) $s_M(G)$ et le temps de la multidistribution (*multi-scattering*) $m_M(G)$.

Si R possède des liens de communication half-duplex, alors G est une graphe non orienté. Au contraire, si R possède des liens de communication full-duplex, alors G est un graphe orienté symétrique, c'est-à-dire un graphe orienté tel que pour chaque arc de u vers v, il existe un arc de v vers u. Si un H est un graphe non orienté, on note H^* le graphe orienté symétrique correspondant.

3.2 Communications globales en wormhole

3.2.1 Critères de comparaison

Les techniques de routage de type commutation de circuit ou wormhole permettent d'écrire des algorithmes de communication différents de ceux utilisés en store-and-forward . Quand c'est le temps de start-up qui est prépondérant dans l'envoi d'un message, on peut diminuer ce facteur par rapport au mode store-and-forward comme dans les exemples présentés plus loin.

Proposition 3.2.1 *Dans un graphe G de degré maximum Δ, le nombre maximum de nœuds qui peuvent être informés en k étapes est (en routage de type commutation de circuit) :* $(\Delta + 1)^k$.

Preuve: En une étape on ne peut informer que Δ fois plus de nœuds. Le nombre total de nœuds qui peuvent être atteints à la fin de chaque étape est :

étape 0 : 1 nœud, (le sommet de départ)

étape 1 : $\Delta + 1$ nœuds, (le premier et les Δ destinataires)

étape 2 : $\Delta + 1 + \Delta(\Delta + 1) = (\Delta + 1)^2$

$\vdots$

étape k : $(\Delta + 1)^k$

La preuve se fait par induction en notant que $\underbrace{(\Delta + 1)^k}_{\text{anciens}} + \underbrace{\Delta(\Delta + 1)^k}_{\text{nouveaux}} = (\Delta + 1)^{k+1}$

□

3.2.2 Diffusion dans l'anneau

Ici on montre une diffusion optimale en nombre d'étapes dans le cas de l'anneau (non orienté). On se limite (pour ne pas charger les notations) à l'étude de C_N tel que $N = 3^k$.

Algorithme

A l'étape i, un nœud informé envoie le message aux nœuds à distance $\frac{N}{3^i}$. Ainsi, dans l'anneau à 27 sommets, numérotés de 1 à 27, si 1 est l'initiateur de la diffusion on informera :

- les sommets 19 et 10 à l'étape 1,
- les sommets 4,25,7,13,16 et 22 à l'étape 2,
- les sommets 27,2,3,5,6,8,9,11,12,14,15,17,18,20,21,23,24 et 26 à l'étape 3.

Complexité

Chaque étape i coûte 1 start-up (α), $\frac{N}{3^i}$ commutations de liens (δ) et le coût d'une transmission du message ($L\tau$); soit un coût total de :

$$b_{F_*} = \frac{1}{2}(N-1)\delta + \log_3(N)(\alpha + L\tau)$$

L'arbre de diffusion obtenu étant de profondeur minimale, le nombre d'étapes est optimal (et donc le nombre de α).

De plus, on a besoin de $\frac{1}{2}(N-1)$ sauts pour traverser le cycle, le coefficient de δ est aussi minimum (on ne peut pas en faire moins que le diamètre).

3.2.3 Diffusion simple dans les grilles toriques

On considère l'algorithme de diffusion dans le tore $n \times n$. On rappelle que le diamètre de ce réseau est $D = 2\left\lfloor \frac{n}{2} \right\rfloor$ et on se situe dans le modèle F_*.

3.2.3.1 Algorithme de diffusion simple pour le modèle Store-and-Forward

L'algorithme s'exécute en D étapes, chaque étape consistant pour un sommet à informer ses voisins. Cet algorithme n'utilise pas de pipeline ni d'arbres de recouvrement disjoints, et sa complexité en temps est de : $b_{F_*} = D(\beta + L\tau)$

3.2.3.2 Algorithme de diffusion pour le modèle Wormhole

Dans cet algorithme on découpe récursivement le tore (pris comme une grille de centre le sommet émetteur) en quatre sous-réseaux. Une étape consiste donc à informer le centre des 4 sous-grilles, ensuite on réapplique récursivement cette approche dans les quatre sous-grilles. A la dernière étape, on peut diffuser en 2 coups dans des grilles 2x2 ou 3x3 selon la taille du tore. Si, comme sur l'exemple de la figure 3.1, on a un tore $n \times n$, avec $n = 2^k$, alors la dernière étape consiste à informer une grille 2x2. L'algorithme est illustré sur cette figure 3.1, où l'on a représenté en gras les chemins empruntés lors de la première étape par le message à diffuser. L'origine de la diffusion est le sommet noir pris au centre du tore.

Dans le cas général (n quelconque) on a :

$$b_{F_*} = D\delta + 2\log_4 n(\alpha + L\tau)$$

Ce temps est meilleur que pour le store-and-forward simple pour ce qui est du facteur α (start-up) ! On a payé quelques δ pour obtenir moins de α. Pour le nombre de $L\tau$ ceci est meilleur que le store-and-forward classique, mais moins bon que dans le cas où L est grand avec l'emploi de la technique du pipeline [9]. Notons que l'on peut combiner la technique du pipeline et le routage wormhole.

Cet algorithme n'est pas optimal (en nombre d'étapes), car le nombre minimal d'étapes pour la diffusion dans le tore $n \times n$ est $\lceil \log_5 n^2 \rceil$ à cause de la proposition 3.2.1. En effet, en k étapes on informe au plus 5^k sommets du tore. La restriction au mode commutation de circuit s'impose ici si l'on veut parler d'optimalité, car en mode wormhole un nœud peut émettre un nouveau message avant que le précedent ne soit arrivé à destination; alors la notion d'étape n'est plus claire.

Un algorithme basé sur un découpage récursif du tore en 5 domaines est présenté dans la section suivante (article [24]). Une idée de cet algorithme est donnée par la figure 3.4. Le nœud au centre du tore est l'initiateur de la diffusion (on reconnaît le tore en se repérant avec les gros ronds noirs, et en repliant ce qui dépasse). A chaque étape, on informe le nœud pris au centre d'une des 5 régions dessinées, et mis en évidence par une croix. La croix symbolise les 4 chemins disjoints de l'étape suivante, qui consiste à informer 4 nouveaux centres.

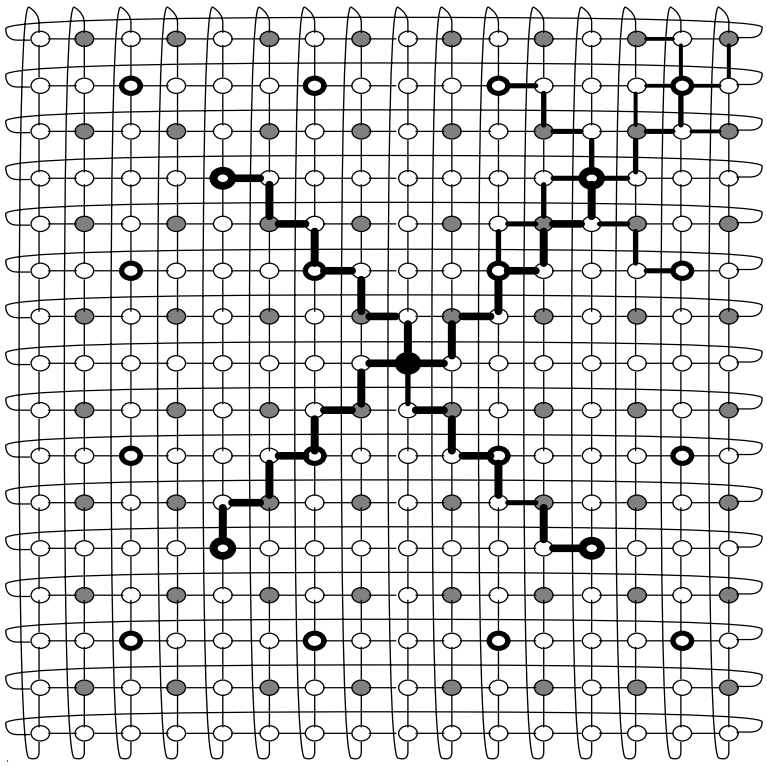

Figure 3.1: Diffusion simple

3.2.3.3 Conclusion

La comparaison entre store-and-forward et wormhole n'est pas évidente. En effet il existe des techniques logicielles utilisées pour accélérer les communications (voir le pipeline et les arbres disjoints du chapitre précédent). De plus, si l'on considère l'algorithme de l'échange total, la technique précédente ne peut plus être appliquée et aucun algorithme n'a été encore proposé pour faire mieux qu'en store-and-forward.

Pour le problème de la diffusion, des expériences ont été faites sur une Intel Touchstone Delta, constituée d'une grille 2D de 520 processeurs i860, avec routage wormhole [1]. Les algorithmes implantés utilisent une technique de partitionnement, comme dans l'algorithme de diffusion pour le tore présenté plus haut, adapté aux grilles non-toriques. On retrouve ainsi un nombre logarithmique d'étapes pour la diffusion. Les résultats expérimentaux montrent une amélioration du temps de diffusion, dans le cadre de routines BLACS.

Les réseaux bidimensionnels et de faible degré, comme le tore, sont les meilleurs candidats pour des réalisation en VLSI, comme cela a été expliqué dans le chapitre 1. Dans le cadre de routages du type store-and-forward, ces topologies sont pénalisées par la durée des communications qui dépend du diamètre, en $O(\sqrt{N})$, alors que cette durée n'est que de $O(\log N)$ pour les réseaux "hypercubiques" [15]. Le

routage de type wormhole ou circuit-switched apporte un nouvel intérêt à ces réseaux en fournissant des schèmas de communication aussi performants que sur les hypercubes. On peut aussi optimiser les communications dans ces réseaux en mode wormhole, mais le handicap des grilles n'est plus aussi grand.

3.3 Circuit-Switched Broadcasting in Torus Networks

Joseph G. Peters[1,2]
School of Computing Science
Simon Fraser University
Burnaby, B.C.
V5A 1S6, Canada

Michel Syska
I3S, CNRS
Université de Nice-Sophia Antipolis
Bât 4, 250 avenue A. Einstein
Sophia Antipolis
06 560 Valbonne, France

Abstract

In this paper we describe an optimal broadcasting algorithm for 2-dimensional torus networks (wrap-around meshes) that uses synchronous circuit-switched routing. The algorithm is based on a recursive tiling of a torus. The algorithm requires $log_5(n^2)$ phases and $n-1$ intermediate switch settings to broadcast in an $n \times n$ torus. Both of these quantities match the lower bounds.

Keywords: broadcasting, torus networks, circuit-switched routing, tilings
AMS (MOS) subject classification: 94A05, 68M10

3.4 Introduction

Distributed memory multicomputer systems in which the processors communicate by exchanging messages over an interconnection network are an increasingly popular method for achieving cost-effective high-performance computing. The performance of a message-passing system is strongly dependent on the topology of the interconnection network and on the routing mechanism that is used to move information around the network. *Multi-dimensional tori (wrap-around meshes, toroidal meshes, k-ary n-cubes)* are currently popular interconnection networks because their low degrees permit efficient layouts and construction with standard components [8]. The large diameters of tori are a disadvantage when *store-and-forward* routing is used because communication time for store-and-forward routing is proportional to the diameter of the network. Store-and-forward routing has been displaced by *circuit-switched* routing in many recent multicomputer systems such as the Intel Touchstone Delta, Fujitsu AP1000, Symult 2010, nCUBE-2, and iWARP. Since the cost of circuit-switched routing is less dependent than store-and-

[1]Part of this research was done while the first author was visiting CNRS, Université de Nice-Sophia Antipolis

[2]Supported by the Centre National de la Recherche Scientifique and the Conseil General des Alpes-Maritimes.

forward routing on the diameter of a network, torus networks with circuit-switched routing are a practical choice.

Broadcasting is a one-to-all information dissemination problem in which a message originating at one node of a network must be distributed to all other nodes of the network. The broadcasting problem has been studied for many different network topologies and routing strategies. Until recently, research on broadcasting concentrated on *unit cost* store-and-forward models in which each message transmission travels along one communication link and takes one unit of time. Much of the recent research on broadcasting has used *linear cost* models in which the propagation time of a message is proportional to the length of the message. Linear costs models have been used to study both store-and-forward routing [9, 13, 25, 28] and various types of circuit-switched routing including *direct connect* [23], *virtual cut-through* [14], and *wormhole* routing [6, 8, 26]. See [12] for a thorough survey of earlier work on unit cost models and [10] for a recent survey of store-and-forward routing under both unit cost and linear cost models.

In this paper, we present a new broadcasting algorithm for 2-dimensional torus networks which uses circuit-switched routing and a linear cost model. We prove that our algorithm is optimal when the messages are short or when the time to initiate a message transmission is much larger than the unit propagation time of a message along a link. This is the situation in many current multiprocessor networks. When the start-up time for message transmissions is small or when messages are long, better performance can be obtained by simulating a store-and-forward algorithm based on the arc disjoint spanning trees in [22].

In the next section, we describe both store-and-forward and circuit-switched routing and define the linear cost model for both types of routing. We also survey previous work on broadcasting in torus networks. In the third section, we give lower bounds on the various components of the linear cost model. In Section 4, we develop and analyze our new algorithm.

3.5 Models of Communication

In a $p \times q$ 2-dimensional torus network, each node has a label (i, j) and four neighbours $(i, j+1)$, $(i, j-1)$, $(i+1, j)$ and $(i-1, j)$ where the first component of a label is an integer mod p and the second is an integer mod q. The diameter of a $p \times q$ torus is $D = \lfloor \frac{p}{2} \rfloor + \lfloor \frac{q}{2} \rfloor$. We will use the *link-bounded* [10] (or *shouting* [12]) model of communication in which a processor can use all of its communication links simultaneously. In contrast, the *processor-bounded* (or *whispering*) model permits the use of only one link at any given time. We also assume that the communication links are *full-duplex* so that messages can travel in both directions simultaneously. Thus, in a 2-dimensional torus network, each node has 4 ports for incoming messages and 4 ports for outgoing messages. We assume that a node can *switch through* a message by connecting an input port to an output port. In a 2-dimensional torus, as many as 4 messages can be switched through a node

simultaneously.

When *store-and-forward* routing is used to send a message along a path of d links, the message is stored in buffers at intermediate nodes on the path. An intermediate node does not begin to send the message to the next node on the path until it has received the entire message. Thus, the transmission time for a message of length L to be sent distance d in the linear cost model is $d(\beta + L\tau)$ where β is the time to initiate a message transmission and $1/\tau$ is the bandwidth of the communication links. (We assume that all links have the same bandwidth.) The total time to complete a broadcast can be reduced by partitioning the message into packets and using a *pipelining* technique to send the packets consecutively along the communication links [25, 28, 13]. The time can be further reduced by distributing the broadcast over several *arc-disjoint spanning trees*. Using pipelining and k arc-disjoint spanning trees of maximum depth h, and choosing the packet size carefully, gives a propagation time of $\left(\sqrt{(h-1)\beta} + \sqrt{\frac{L\tau}{k}}\right)^2$ [2]. Combining this result with the four arc-disjoint spanning trees of depth $D+1 = \lfloor \frac{p}{2} \rfloor + \lfloor \frac{q}{2} \rfloor + 1$ from [22] gives time $\left(\sqrt{D\beta} + \sqrt{\frac{L\tau}{4}}\right)^2$ to broadcast in a $p \times q$ torus.

When *circuit-switched routing* is used, a header containing the destination address is sent through the network to "build" a path. At each intermediate node on the path, the input and output ports used by the header are connected. When the destination node receives the header, it sends an acknowledgement back to the source node establishing a direct connection between source and destination. The bytes of the message are then sent in pipeline fashion. Since the message is switched through intermediate nodes, there is no need to buffer the entire message. The links of the path can be released as the last byte passes through each node or by an acknowledgement from the destination when the last byte is received. The former case is known as *direct connect* [23]. In a *wormhole* implementation of circuit-switching, the header establishes a path to the destination in the same way as in a direct connect implementation, but an acknowledgement is not sent back to the source node. Instead, the remaining bytes immediately follow the header in pipeline fashion with the last byte releasing the switches as it passes through. The Torus Routing Chip described in [7] uses wormhole routing for *point-to-point* (i.e., one-to-one) communications and can be used to build multidimensional tori [6].

In the linear cost model, the transmission time for a message of length L to be sent distance d is $\alpha + d\delta + L\tau$, where α is the time to initiate a new message transmission, δ is the time to switch an intermediate node, and $1/\tau$ is the bandwidth of the communication links. In most current machines, message transmissions are initiated in software and switching is done in hardware, so δ is usually much smaller than α. Furthermore, α is usually much larger than τ. For example, in the *i*PSC/860, the time to transmit $L \leq 100$ bytes over a distance d has been measured to be $65 + 10d + 0.425L\mu s$ [3, 4]. Store-and-forward routing can be simulated by circuit-switched routing by restricting d to be 1 for all transmissions, so the store-and-forward upper bounds stated above are also true for circuit-switched

systems with $\beta = \alpha + \delta$.

In the wormhole routing model, only the total switching time depends on the distance d, so it is possible that the last byte of a message has left the originator before the header reaches the destination. This could happen, for example, if the message is very short or d is large. This cannot happen in the direct connect model because the source and destination nodes are synchronized. The transmission times for wormhole routing and direct connect routing are similar when messages are long or the paths are short. When messages are short or the paths are long, the two routing schemes behave quite differently, but the dominant factor is α in both cases, and the α factor (i.e., number of phases) cannot be improved by using wormhole routing instead of direct connect routing. Since direct connect routing is also easier to analyze than wormhole routing, we will adopt the direct connect model in this paper.

When any type of circuit-switched routing is used, and communication patterns are arbitrary, deadlock is possible and many papers on wormhole routing are more concerned with deadlock avoidance than efficiency [8, 17, 18, 19]. The most common deadlock avoidance method is the use of *virtual channels* which use multiplexing to share physical links. Since our algorithm uses fixed, predetermined communication patterns, there is no possibility of deadlock. Since our algorithms are also synchronous, virtual channels offer no performance advantages that our algorithms can exploit, so we will ignore virtual channels.

The minimum phase wormhole broadcasting algorithm in [1] shows that broadcasting can be done more efficiently in a processor-bounded system with circuit-switched routing than with store-and-forward routing. The algorithm that we present in Section 4 is a minimum phase circuit-switched broadcasting algorithm for link-bounded systems.

3.6 Lower Bounds

The total transmission time for broadcasting in the linear cost circuit-switched model has three components: the *total start-up time*, the *total switching time*, and the *total propagation time*. These components are measured in terms of α, δ, and τ respectively. We can prove lower bounds on these components individually and in combination.

The total start-up time depends on the number of *phases* in a broadcast. A phase for a node that is sending a message starts when the node begins the initiation of the transmission and ends when the links over which the message was sent are released. For a node that is receiving a message, a phase starts when the header reaches the node and ends when the link on which the message arrived is released. Notice that the phases of the source and destination nodes of a message transmission do not start and end at exactly the same times. Furthermore, our definition of phase ignores the fact that a node can be sending and receiving different messages on each of its links and the starting and ending times for these

transmissions can all be different. Thus, according to our definition, a node in a 2-dimensional torus could be in as many as eight different phases simultaneously.

To simplify our lower and upper bound analyses, we will assume that a node is in at most one phase at any given time. This assumption restricts the class of broadcasting algorithms that we can analyze, but has no effect on the generality of our lower bounds. Since none of the algorithms in the next section take advantage of the more general definition of phase, the assumption does not affect any of our upper bounds.

The movement of information in a broadcast is partially ordered because a node cannot send information before it has received it. Some of our lower bounds are based on determinations of minimum length critical paths in partial orders and other are based on analyses of bottlenecks in networks. None of our arguments rely on the definition of *phase*, so none of our lower bounds are affected by the simplifying assumption.

Proposition 3.6.1 *In a vertex-transitive graph G with N nodes, degree Δ, edge connectivity λ, and diameter D, the minimum time to broadcast a message of length L is*

$$\max\left(\lceil \log_{\Delta+1} N \rceil \alpha, D\delta, \frac{L\tau}{\lambda}, \alpha + D\delta + \frac{L\tau}{\Delta}\right)$$

Proof: First consider the total start-up time. When a source node initiates a message transmission, it incurs a cost of α and can transmit the message to at most Δ other nodes without incurring further start-up costs. Each of these Δ nodes can begin to forward the message to at most Δ more nodes as soon at it receives the header of the message from the source node, but each of these "second generation" transmissions incurs a cost of α. After it has completed its first set of transmissions, the source node can also start to inform a "second generation" of Δ more nodes. These $\Delta + 1$ "second generation" transmissions can start at different times, but no "third generation" node can receive the header of the message before at least 2α time units have elapsed. Continuing in this way, it is easy to see that at most $(\Delta + 1)^k$ nodes can have the message after $k\alpha$ time units, and it follows that the total start-up time is at least $\lceil \log_{\Delta+1} N \rceil \alpha$.

The following arguments are simple extensions of the arguments developed in [13, 28] for store-and-forward routing in hypercubes.

The lower bound of $D\delta$ on total switching time is immediate since the message must travel along a path of length at least D from the source to at least one other node, and this path contains at least D switches.

If the edge connectivity of G is λ, then there is an edge cut of size λ which separates the source node from at least one other node. The bandwidth of this cut is $\frac{\lambda}{\tau}$, so the minimum propagation time for a message of length L through this edge cut is $\frac{L\tau}{\lambda}$.

The final term of the lower bound is the minimum time for the message to reach a node at distance D from the source. The header of the message cannot

be received by this node in less than $\alpha + D\delta + \tau$ time units (assuming that the header has unit length). Since the node can receive information on all Δ of its edges simultaneously, it can receive Δ units of message by time $\alpha + D\delta + \tau$ and the remaining $L - \Delta$ units of message require at least $\frac{L-\Delta}{\Delta}\tau$ more time.

□

Corollary 3.6.2 *In an $n \times n$ 2-dimensional torus, the minimum time to broadcast a message of length L is*

$$\max\left(2\lceil \log_5 n \rceil \alpha, \alpha + 2\lfloor \frac{n}{2} \rfloor \delta + \frac{L\tau}{4}\right)$$

When the messages are short or when α is much larger than δ and τ, the minimum time to broadcast in a 2-dimensional $p \times q$ torus approaches $\lceil log_5 pq \rceil \alpha$.

3.7 An Optimal Algorithm for Short Messages

The broadcasting algorithm presented in this section can be viewed as a recursive tiling of a torus. We will take a direct approach to the development of our algorithm that avoids the sophisticated machinery that has been developed to study tilings. For an extensive treatment of tilings in general and tilings of the torus in particular, the reader is referred to [11] and [27] respectively.

Figure 3.2 is a pictorial representation of the algorithm for a 5×5 torus. The nodes have labels (i, j) where i and j are integers mod 5. The originator is at the point in the centre of the black cross and we assume that it has label $(0, 0)$. In the first phase, the originator broadcasts the message to the four other nodes that are the centres of white crosses in the diagram. These nodes have indices $(1, 2), (-1, -2), (2, -1)$, and $(-2, 1)$. The paths used to accomplish this are shown as diagonal lines in the diagram for clarity. In reality, each path will consist of three links of the torus and the four paths are arc disjoint. In the second phase, each informed node (x, y) sends the message to its four immediate neighbours $(x + 1, y), (x - 1, y), (x, y + 1)$, and $(x, y - 1)$. Shading is used in the diagrams to show the "areas" informed during different phases of the broadcast. It is easy to see that the shaded area in Figure 3.2 can be wrapped around to form a 5×5 torus. The broadcast time is $2\alpha + 4\delta + 2L\tau$.

We can view the shaded area in Figure 3.2 as a 5×5 "square" which we denote S_1. The four "corners" of S_1 are shown as black circles. The top "edge" of S_1 consists of nine line segments with orientations "right-down-right-up-right-up-right-down-right". We will call this edge an E_1 edge with the clockwise orientation being understood. Clearly, the other three edges of S_1 are rotations of E_1. An E_1 edge has 180° rotational symmetry so the E_1 edges on the top and bottom of an S_1 fit perfectly into each other as do E_1 edges on the left and right sides of an S_1. A 5×5 torus is obtained by identifying all four corners of an S_1 with a single point.

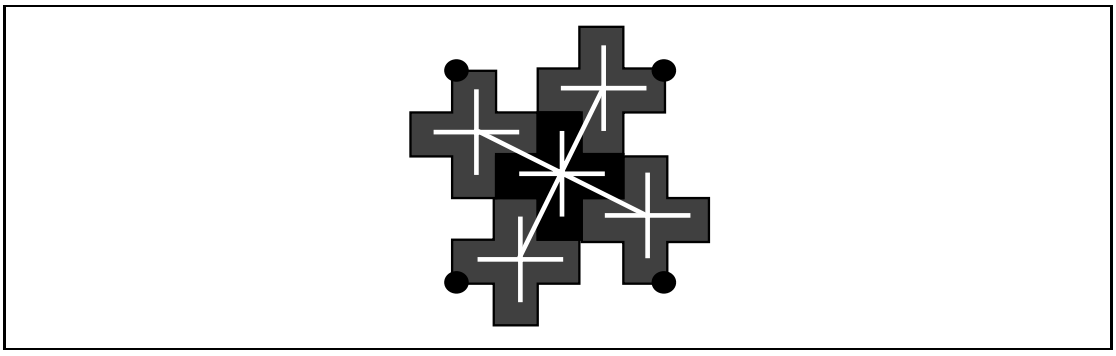

Figure 3.2: A 5×5 torus drawn as an S_1 square.

We can build bigger tilings by identifying the corners of several S_1 squares. Figure 3.3 shows one possible tiling using five S_1 squares. Some of the details of the broadcasting scheme are omitted from the four outer S_1 squares to simplify the diagram. The tiling in Figure 3.3 does not wrap around to form a torus, but we can view the shaded area as a "cross" in the same sense that S_1 is a square. Each corner of the cross is shown as a black circle and each "edge" of the perimeter of the cross is an E_1. We will denote this cross by C_1. We will use S_0 to denote a simple square consisting of a single node and C_0 to denote a simple cross formed from five S_0's. A C_1 is formed by arranging five S_1's into the same pattern as the S_0's are arranged to form a C_0. Similarly, the symmetries of C_1's allow us to form the S_2 in Figure 3.4 by arranging five C_1's into the same pattern as the C_0's are arranged to form an S_1 in Figure 3.2.

We can view an S_2 as a square with corners indicated by the large black circles. Each of the four "edges" of S_2 consists of 9 E_1 edges (delimited by small black circles in Figure 3.4) arranged in the same "right-down-right-up-right-up-right-down-right" pattern as nine simple edges are arranged in an E_1. It is therefore immediate that E_2 edges also have 180° rotational symmetry and that a 25×25 torus results when all four corners of an S_2 are identified with a single point. It is curious that the outlines of our S_k tilings, as drawn in Figures 3.2 and 3.4, are Koch curves [20].

Theorem 3.7.1 *Broadcasting in a torus of size* $5^k \times 5^k$, $k \geq 1$, *requires time at most* $2k\alpha + (5^k - 1)\delta + 2kL\tau$.

Proof: Clearly, we can continue the construction described above to obtain a tiling for a torus of size $5^k \times 5^k$ for any $k \geq 1$. Furthermore, all of the data paths in use during any particular phase are arc disjoint. To calculate the time required by this broadcasting algorithm, we need to determine the number of edges on each data path. To simplify notation in this proof, we will number the phases from last to first. During phase 1 (the last phase), the central node (x, y) of each C_0 broadcasts to its four immediate neighbours $(x+1, y)$, $(x-1, y)$, $(x, y+1)$, and $(x, y-1)$. During phase 2, the central node of the central C_0 of each S_1 broadcasts

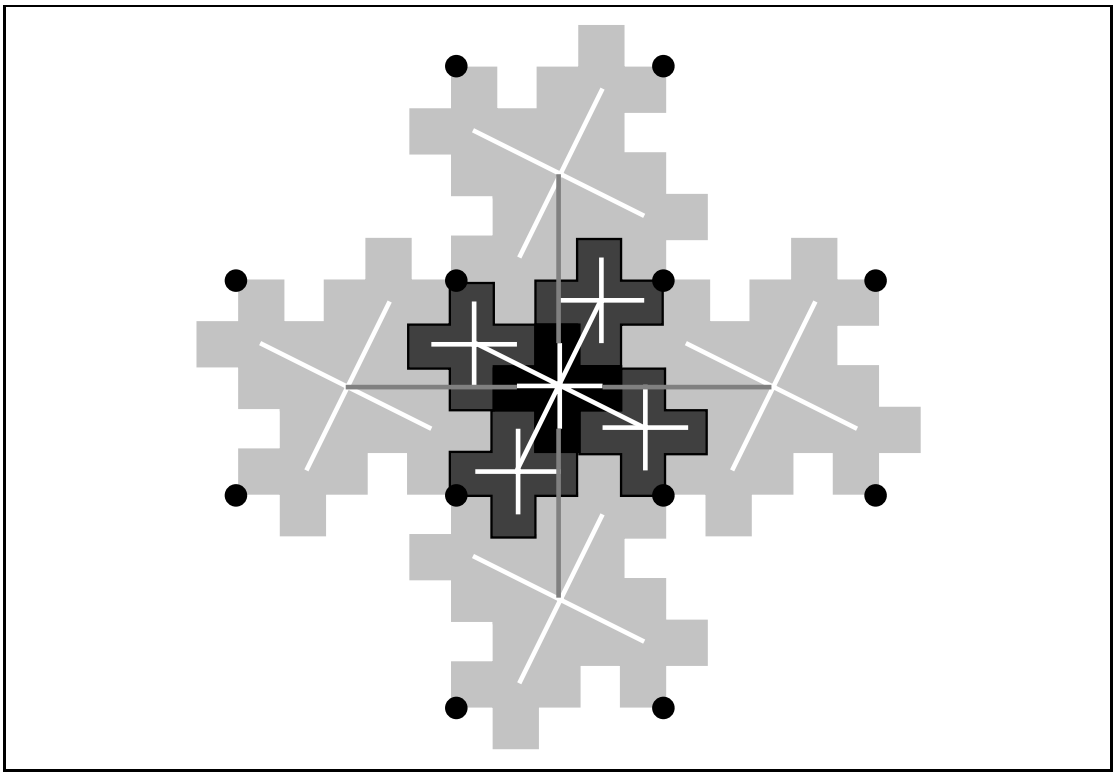

Figure 3.3: A C_1 cross.

to the central nodes of the four other C_0's in the same S_1. In particular, (x, y) broadcasts to $(x+1, y+2), (x-1, y-2), (x+2, y-1)$, and $(x-2, y+1)$ (see Figure 3.2). In general, in an odd-numbered phase $2i+1$, the centre of the central S_i of each C_i is broadcasting to the centres of the four other S_i's in the same C_i. Since S_i's are $5^i \times 5^i$ "squares", the centres of two adjacent S_i's in a C_i are distance 5^i apart (either horizontally or vertically). Similarly, in an even-numbered phase $2i$, the centre (x, y) of the central C_i of each S_{i+1} broadcasts to the centres of the four other C_i's in the same S_{i+1} and the data paths to these centres have $3 \cdot 5^i$ edges. In particular, during phase j, an informed node (x, y) will broadcast to the four nodes $(x+u_j, y+v_j)$, $(x-u_j, y-v_j)$, $(x+v_j, y-u_j)$, and $(x-v_j, y+u_j)$ where

$$u_j = \begin{cases} 5^{\frac{j}{2}-1} & \text{if } j \text{ even} \\ 0 & \text{if } j \text{ odd} \end{cases} \qquad \text{and} \qquad v_j = \begin{cases} 2 \times 5^{\frac{j}{2}-1} & \text{if } j \text{ even} \\ 5^{\frac{j-1}{2}} & \text{if } j \text{ odd.} \end{cases}$$

Since each informed node broadcasts to four other nodes during each phase, the number of phases in the broadcasting scheme is $log_5(5^{2k}) = 2k$. The total time is therefore

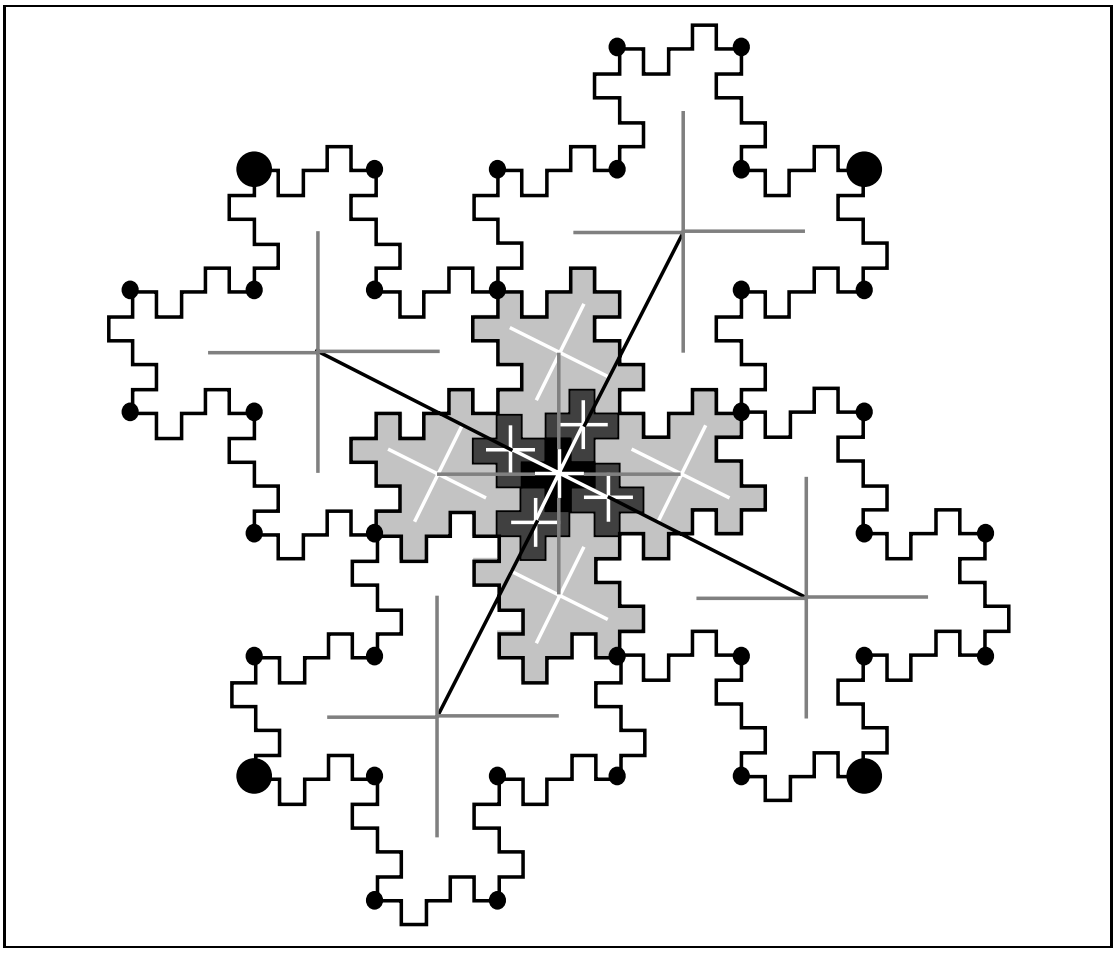

Figure 3.4: A 25×25 torus drawn as an S_2 square.

$$\begin{aligned}
\sum_{j=1}^{2k}[\alpha + (u_j + v_j)\delta + L\tau] &= 2k\alpha + \delta\sum_{i=1}^{k}[u_{2i-1} + u_{2i} + v_{2i-1} + v_{2i}] + 2kL\tau \\
&= 2k\alpha + \delta\sum_{i=1}^{k}[4 \cdot 5^{i-1}] + 2kL\tau \\
&= 2k\alpha + (5^k - 1)\delta + 2kL\tau.
\end{aligned}$$

□

Theorem 3.7.1 establishes that our algorithm is optimal in terms of both α and δ for tori for which both dimensions are the same even power of 5. We can use different tilings to obtain algorithms for other sizes of tori. These algorithms are optimal in terms of α and are usually optimal or close to optimal in terms of δ.

One way to create new tilings is to expand each node of a torus into a block of nodes. For example, Figure 3.5 shows a C_0 cross in which each node has been expanded into a 2×2 block of nodes. The node in the lower left corner of each block is the "boss" of the block. We can make an expanded S_1 square by combining five of these expanded C_0's using the same pattern as in Figure 3.2. To broadcast in

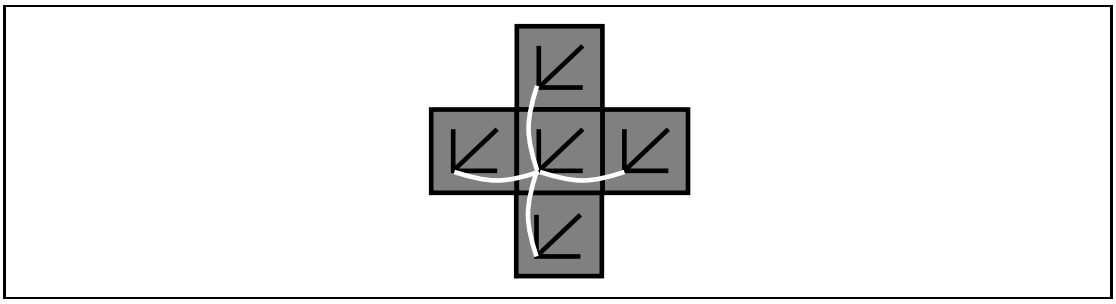

Figure 3.5: An expanded C_0 cross with each node replaced by a 2×2 block.

this 10×10 torus, first perform a broadcast to the block bosses using the algorithm for an S_1 square. Since all the message paths are exactly twice as long as they were in the S_1 square, the total switching time for the two phases of the algorithm is doubled giving a time of $2\alpha + 8\delta + 2L\tau$. In the last phase, each boss broadcasts within its block. There are several ways to do this. The most efficient way is for the block boss (i, j) to route the message directly to $(i + 1, j)$, indirectly to $(i+1, j+1)$ via $(i, j+1)$, and indirectly to $(i, j+1)$ via $(i-1, j)$ and $(i-1, j+1)$. This adds $\alpha + 3\delta + L\tau$ to the cost. (The other ways to broadcast within a block either use an extra phase or use virtual channels.) We can improve on this result by changing the shape of the blocks for the last phase. The algorithm is the same during the first two phases, but during the last phase each representative (i, j) broadcasts directly to $(i-1, j)$ and $(i, j+1)$, and to $(i+1, j+1)$ via $(i+1, j)$. It is clear that these "slanting" blocks give a valid tiling. The last phase using slanting blocks adds only $\alpha + 2\delta + L\tau$ to the cost for a total cost of $3\alpha + 10\delta + 3L\tau$. The 3α term is optimal, and so is the 10δ term since the diameter of a 10×10 torus is 10.

We can also form a 10×10 torus by combining four S_1 squares as shown in Figure 3.6. In effect, we are replacing a single S_1 by a 2×2 block of S_1's. The bosses of the four S_1's are the nodes at the centres of the white crosses, and the originator is the boss of the lower left S_1. In the first phase, the originator broadcasts to the three other bosses. Each boss then starts a normal S_1 broadcast in its S_1. The first phase requires time $\alpha + 10\delta + L\tau$ and the last two phases require $2\alpha + 4\delta + 2L\tau$ for a total time of $3\alpha + 14\delta + 3L\tau$. Intuitively, the reason that this method is less efficient than the method in the preceding paragraph is because the inefficient phase in which informed nodes inform only three other nodes occurs between S_1's instead of S_0's. We can move the inefficient phase even further out in the recursion for larger tori, but this will always be worse than doing the block broadcast in the last phase.

Our construction methods also work for tori that are not square. For example, a 5×10 torus can be formed by expanding each node of an S_1 square into a block of two horizontally adjacent nodes. In the first two phases, perform a broadcast to the left nodes of the blocks, and in the last phase each node broadcasts to its right neighbour. In the first two phases, the horizontal distances are twice as large as

in an S_1 and the vertical distances are the same. The total cost is $3\alpha + 8\delta + 3L\tau$. The 3α term is optimal and the 8τ term is close to the lower bound of 7δ.

We can generalize our constructions to any size of torus by expanding nodes into blocks of the appropriate dimensions. The broadcasting time will be closest to the lower bounds when the torus is close to square and when the dimensions are close to powers of 5.

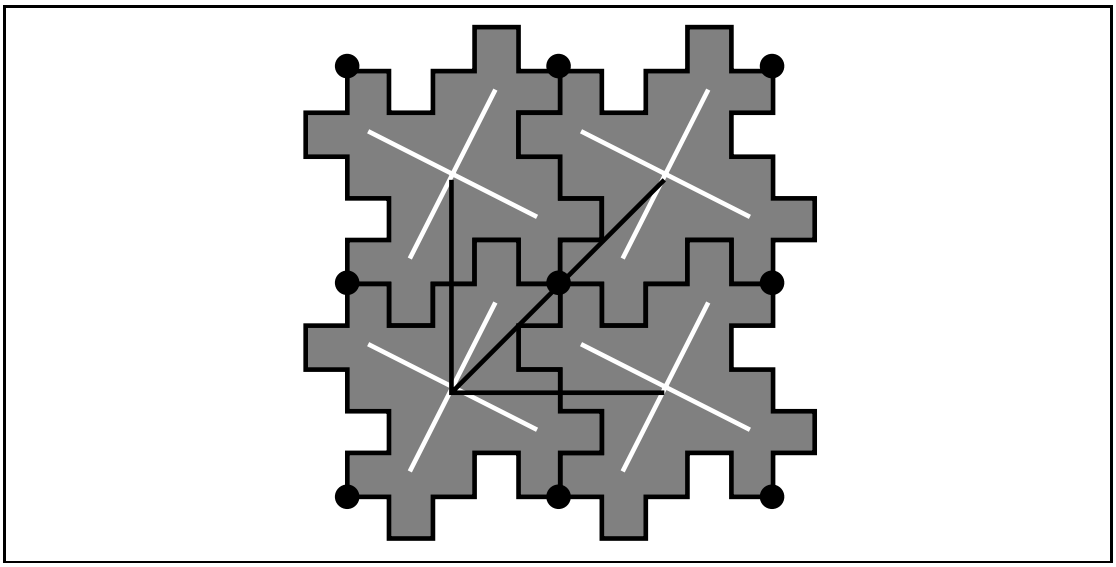

Figure 3.6: A 10×10 torus made from four S_1 squares.

3.8 Conclusion

Since VLSI chips are wire-limited, 2-dimensional networks with low degree, such as the torus are good candidates for VLSI implementations [6]. With store-and-forward routing, the communication algorithms for these topologies require times that are at best proportional to the diameter of the network. The circuit-switched algorithms in this paper require exponentially less time when the messages are short.

3.9 Acknowledgements

We would like to thank Tom Shermer for suggesting the presentation of our diagrams.

Bibliographie

[1] M. Barnett, D.G. Payne, and R. van de Geijn. Optimal broadcasting in mesh-connected architectures. Technical report, University of Texas at Austin, 1988.

[2] J.-C. Bermond and P. Fraigniaud. Communications in interconnection networks. *Proc. Combinatorial Optimization in Science and Technology '91*, 1991.

[3] S. Bokhari. Communication overhead on the intel *i*PSC-860 hypercube. Technical report, ICASE, NASA Langley Research Center, 1990.

[4] R. Boppana and C. Raghavendra. All-to-all personalized communication on circuit-switched hypercubes. Technical report, Dept EE-Systems, USC, Los-Angeles, 1990.

[5] M. Cosnard and P. Fraigniaud. Finding the roots of a polynomial on an MIMD multicomputer. *Parallel Computing*, 15:75–85, 1990.

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

[7] W.J. Dally and C.L. Seitz. The torus routing chip. *J. Distributed Computing* 1(3):187–196, 1986.

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

[9] 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, E.N.S.L, 1990.

[10] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discr. Appl. Math.*, to appear.

[11] B. Grünbaum and G.C. Shephard. *Tilings and Patterns.* W.H. Freeman, 1987.

[12] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1986.

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

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

[15] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures : Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1991.

[16] F.T. Leighton. Methods for message routing in parallel machines. pages 77–96, 1992.

[17] D.H. Linder and J.C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Trans. Comp.* 40(1):2–12, 1991.

[18] X. Lin, P.K. McKinley, and L.M. Ni. Performance evaluation of multicast wormhole routing in 2D-mesh multicomputers. Proc. 1991 Int. Conf. on Parallel Processing, I-435–I-442, 1991.

[19] X. Lin and L.M. Ni. Deadlock-free multicast wormhole routing in multicomputer networks. Proc. 18th Int. Symp. on Computer Architecture, 116–125, 1991.

[20] B.B. Mandelbrot. *The Fractal Geometry of Nature.* W.H. Freeman, San Francisco, 1982.

[21] S. Miguet. *Programmation Dynamique et Traitement d'Images sur Machines Parallèles à Mémoire Distribuée.* PhD thesis, ENS-Lyon, France, 1990.

[22] P. Michallon, D. Trystram, and G. Villard. Optimal broadcasts algorithms on the torus. Tech. Rep. 872-I-01-92 LMC-IMAG, Grenoble, France 1992.

[23] S.F. Nugent. The iPSC/2 direct-connect technology. In G.C. Fox, editor, *Proceedings of 3rd Conference on Hypercube Concurrent Computers and Applications*, pages 51–60. ACM, 1988.

[24] J. Peters and M. Syska. Circuit-Switched Broadcasting in Torus Networks. in preparation, 1992.

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

[26] C.L. Seitz. Concurrent architectures. In R. Suaya and G. Birtwist, editors, *VLSI and Parallel Computation*, pages 1–84. Morgan Kaufmann, 1990.

[27] M. Senechal. Tiling the torus. *Discr. and Comp. Geometry*, 3:55–72, 1988.

[28] 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.

Chapitre 4

Excentricités moyennes des réseaux de de Bruijn

Introduction au calcul des excentricités

Certains calculs présentés dans ce chapitre sont assez techniques, aussi nous donnons en introduction une synthèse des résultats, ainsi que la démarche adoptée pour le calcul de l'excentricité moyenne de $x = 0 \cdots 0$.

Notations et définitions

Quelques définitions nécessaires à ce chapitre.
Un *graphe orienté* (en anglais, digraph) G est constitué d'un ensemble X fini de points $\{x_1, x_2, \ldots, x_N\}$, et d'une famille U d'éléments du produit cartésien $X \times X$, un élément (x, y) de $X \times X$ pouvant apparaître plusieurs fois dans la famille U. X est dit ensemble des sommets de G et U est dit ensemble des arcs.
Le nombre de sommets du graphe est appelé *ordre* du graphe et sera noté N.
Un arc de G de la forme (x, x) est appelé *boucle* (loop). Pour un arc $u = (x, y)$, le point x est son extrémité initiale et le point y, son extrémité terminale.
On dit que y est un successeur de x si il existe un arc ayant son extrémité initiale en x et son extrémité finale en y. L'ensemble des successeurs de x se note $\Gamma_G^+(x)$ et le nombre d'éléments de $\Gamma_G^+(x)$ est dit *demi-degré extérieur* (out-degree) de x et se note $d^+(x)$.

On peut définir de manière analogue les prédécesseurs de x par l'ensemble $\Gamma_G^-(x)$ et le *demi-degré intérieur* (in-degree) de x, noté $d^-(x)$.
Etant donné un sommet x de G, on appelle *degré* (degree) de x et on note $d(x)$, le nombre d'arêtes incidentes à x.

On appelle degré maximum de G et on note Δ le maximum, pris sur tous les sommets x de G, du degré de x.

On appelle degré minimum de G et on note δ le minimum, pris sur tous les sommets x de G, du degré de x.

On appelle *chaîne* (path) entre deux sommets x et y de G, une suite de sommets $x_1, x_2, \ldots, x_k$ dont deux consécutifs sont adjacents, avec $x_1 = x$ et $x_k = y$. Une chaîne qui n'utilise pas deux fois le même sommet est dite *élémentaire*. Ici on ne parlera que de "chaîne élémentaire", et on dira simplement "chaîne".

La *longueur d'une chaîne* est le nombre d'arêtes de cette chaîne.

Etant donnés deux sommets x et y d'un graphe G, on appelle *distance* de x à y et on note $d(x, y)$, la longueur d'une plus courte chaîne d'extrémités x et y.

On appelle *diamètre* du graphe, que l'on note D le maximum, pris sur toutes les paires de sommets x et y de G, de la distance de x à y.

On peut definir les mêmes notions pour les graphes orientés. On parle alors de *chemin* (dipath), deux sommets consécutifs étant joints par un arc. On notera que dans ce cas la distance ne possède plus la propriété de symétrie, les plus courts chemins de x à y étant différents de ceux de y à x.

L'*excentricité* d'un sommet X est définie [5] comme la distance au nœud le plus éloigné de ce sommet : $e(X) = \max\{d(X, Y) : Y \in V\}$.

Nous définissons l'*excentricité moyenne* d'un sommet X, notée $\bar{e}(X)$, comme la distance moyenne entre ce sommet et tous les autres sommets du graphe :

$$\bar{e}(X) = \frac{1}{N-1} \sum_{Y \in V - \{X\}} d(X, Y). \tag{4.1}$$

Le graphe de *de Bruijn* $B(d, D)$ est le graphe orienté dont les sommets sont les mots de longueur D sur un alphabet $\mathcal{A}$ de taille d. Les arcs sont définis de la façon suivante : il y a un arc du sommet $(x_1, x_2, \cdots, x_D)$ vers le sommet $(x_2, x_3, \cdots, \lambda)$, λ étant une lettre quelconque de $\mathcal{A}$. Il est facile de voir que ce graphe a pour demi-degré extérieur et intérieur d et que son nombre de sommets est égal à d^D. Le diamètre de ce graphe est D.

On peut, ayant défini les graphes de *de Bruijn* orientés, définir maintenant les graphes de *de Bruijn* non orientés : il suffit d'oublier les orientations des arcs, puis de supprimer les arêtes multiples éventuellement créées.

Cas orienté

Dans le cas du graphe orienté, on connaît l'unique plus court chemin entre toutes paires de sommets (voir par exemple Fiol et al. [7]). La distance entre $X = x_1 \cdots x_D$ et $Y = y_1 \cdots y_D$ est donnée par le plus petit i tel que $x_{i+1} \cdots x_D = y_1 \cdots y_{D-i}$. On obtient le plus court chemin correspondant en introduisant successivement $y_{D+1-i}, \cdots, y_D$ dans X par des décalages à gauche. Ceci nous permet de calculer directement $d(X, Y)$, et donc $\bar{e}(X)$ pour tout X, mais ne donne malheuresement pas une forme close pour l'excentricité moyenne.

Cependant, nous pouvons donner une borne inférieure et une borne supérieure, et montrer que ces bornes sont atteintes par certains sommets.

Pour $d \geq 2$, $D \geq 2$, et pour tout $X \in B(d,D)$, on a :

$$D - \frac{d}{(d-1)^2} + \frac{d}{d-1} \cdot \frac{D}{N-1} \leq \bar{e}(X) \leq D - \frac{1}{d-1} + \frac{D}{N-1}. \qquad (4.2)$$

En particulier, quand $d = 2$, on obtient :

$$D - 2 + \frac{2D}{N-1} \leq \bar{e}(X) \leq D - 1 + \frac{D}{N-1}. \qquad (4.3)$$

Le sommet de plus forte excentricité moyenne est $a \cdots a$, et la demonstration est basée sur le lemme 1 qui montre que l'arbre des plus courts chemins de racine $a \cdots a$ est celui qui comporte le plus de sommets dans les niveaux éloignés de la racine, parmi tous les arbres de même degré et même ordre possibles. Intuitivement, comme le sommet $a \cdots a$ est à l'origine d'une boucle, il a moins de voisins à distance 1 que les autres ($\Gamma^+(a \cdots a)$, et ainsi de suite ...

Pour le sommet de plus faible excentricité moyenne, on essaye de construire l'arbre des plus courts chemins le moins profond possible en saturant chaque niveau en fonction du degré maximum du graphe, ce qui revient dans le cas du $B(d,D)$ à prendre l'arbre $d-$aire complet et compléter le dernier niveau. On construit ainsi l'arbre des plus courts chemins du sommet $a \cdots a\bar{a}$.

Cas non orienté

Les calculs sont beaucoup plus difficiles dans le cas non orienté, et nous formulons la conjecture suivante :

Conjecture : pour $d \geq 2$ et $D \geq 2$, les sommets $a \cdots a$, $a \in \mathcal{A}$, sont ceux de plus forte excentricité moyenne dans $UB(d,D)$.

Cette conjecture (qui devrait bientôt devenir une proposition) est confirmée par des calculs exhaustifs jusqu'à $D = 18$ pour le cas binaire.

On peut définir le voisinage d'un sommet dans $UB(d,D)$ par des opérations de décalages à gauche et à droite, et donc représenter les chaînes de ce graphe comme des séquences des décalages correspondants. La longueur d'une chaîne reliant X à Y est égale à la longueur de la séquence de décalages.

De manière plus formelle, soit L l'opération correspondant à une séquence de l décalages à gauche, qui consiste à introduire un suffixe $B = b_1, \cdots, b_l$, avec $|L| = l$, et soit R l'opération correspondant à une séquence de r décalages à droite, qui consiste à introduire un préfixe $A = a_r \cdots a_1$ avec $|R| = r$. Alors, appliquer L au sommet $X = x_1 \cdots x_D$, donne le sommet $x_{l+1} \cdots x_D b_1 \cdots b_l$, ce qui correspond à la chaîne :

$$x_1 \cdots x_D \to x_2 \cdots x_D b_1 \to x_3 \cdots x_D b_1 b_2 \to \cdots \to x_{l+1} \cdots x_D b_1 \cdots b_l.$$

Appliquer R au sommet $X = x_1 \cdots x_D$ donne le sommet $a_r \cdots a_1 x_1 \cdots x_{D-r}$, qui correspond à la chaîne :

$$x_1 \cdots x_D \to a_1 x_1 \cdots x_{D-1} \to a_2 a_1 x_1 \cdots x_{D-2} \to \cdots \to a_r \cdots a_1 x_1 \cdots x_{D-r}.$$

Avec ces notations, nous formulons la proposition suivante :

Proposition 4.0.1 *Une plus courte chaîne entre un sommet X et le sommet $a\cdots a$ est de la forme LR (avec $0 \leq r < l$) ou bien de la forme RL (avec $0 \leq l < r$).*

Dans le cas général (X quelconque), les plus courtes chaînes alternent trois séquences, le cas de $a\cdots a$ est donc plus simple.

Soient $X \in UB(d,D)$ et $0 \leq h \leq D$. On définit $E^h(X)$, l'ensemble des sommets à distance au plus $D-h$ de X:

$$E_h(X) = \{Y |\ d(X,Y) \leq D-h\}.$$

Par définition, $E_D(X)$ contient tous les sommets de $UB(d,D)$ et $E_0(X)$ est un singleton. Soit $e_h(X) = |E_h(X)|/N$ la proportion de sommets qui sont à distance au plus $D-h$ de X.

Lemme : pour $d \geq 2$, $D \geq 2$, et pour tout $X \in UB(d,D)$,

$$\bar{e}(X) = \frac{N}{N-1}\left(D - \frac{1}{N} - \sum_{h=1}^{D-1} e_h(X)\right). \tag{4.4}$$

"Il ne reste plus" qu'à calculer les $e_h(X)$! Dans la suite nous menons le long calcul des $e_h(a\cdots a)$, qui nous permet d'encadrer $\bar{e}(a\cdots a)$.

Quand $X = a\cdots a$, (on peut simplement étudier $X = 0$ sans perte de généralité), soit $E^h = \{Y |\ d(0,Y) \leq D-h\}$. On a alors la proposition suivante :

Proposition 4.0.2

$$E^h = \bigcup_{0 \leq k \leq \lfloor \frac{D-h}{2} \rfloor} E_h^k$$

où : $E_h^k = L_k^h \cup R_k^h$ avec :

- $L_k^h = \{x | x = awb$ avec $w = 0\cdots 0$ et $|w| = k+h, |a| = k\}$
- $R_k^h = \{x | x = awb$ avec $w = 0\cdots 0$ et $|w| = k+h, |b| = k\}$

Autrement dit, on connaît l'écriture des sommets appartenant à ces ensembles, et un sommet de E_h^k contient un "paquet" de $k+h$ '0' dans son écriture, collé après/avant un préfixe/suffixe de taille k. Cette proposition nous amène à énumérer les mots de taille D qui s'écrivent de cette façon, et nous les énumérons en terme de *proportions* (par exemple il y a exactement une proportion de 2^{-h} sommets qui commencent par h '0').

Ce calcul aboutit car on se rend compte que seuls les ensembles correspondants à des h petits sont importants : la proportion de sommets proches de la racine est faible, et quand D est grand, on peut se contenter de ne compter que les ensembles pour les premières valeurs de h, ce qui permet tout de même d'obtenir une précision de l'ordre de 10^{-4}.

Le détail de ces calculs est donné dans l'article suivant, et le résultat (pour D suffisament grand) est :

$$\boxed{D - 2.199528175 \leq \bar{e}(0 \cdots 0) \leq D - 2.199589203}$$

L'excentricité moyenne de $0 \cdots 0$ (et donc $1 \cdots 1$) est d'environ $D - 2.2$ dans le *de Bruijn* non orienté.

Mean eccentricities of *de Bruijn* networks

Jean-Claude Bermond and Michel Syska
I3S, CNRS-Université de Nice-Sophia Antipolis
Bât 4, 250 avenue A. Einstein
Sophia Antipolis, 06 560 Valbonne

Zhen Liu
INRIA, Centre de Sophia Antipolis
2004 route des Lucioles
06 560 Valbonne

Abstract

Given a graph $G = (V, E)$ we define $\bar{e}(x)$, the mean eccentricity of a vertex x, as the average distance from x to all the others vertices of the graph. Let $N = |V|$ and $d(x, y)$ the distance from x to y, $\bar{e}(x) = \frac{1}{N-1} \sum_{y \in V - \{x\}} d(x, y)$. The computation of this parameter appears to be difficult in the case of the *de Bruijn* networks. In this article we give a lower bound and an upper bound for $\bar{e}(x)$.

In the case of a *de Bruijn* digraph the bounds, given are sharp and we exhibit the extremal vertices, whereas in the undirected case, the computation is more difficult, even for degree 4. However we give a way to compute the value of $\bar{e}(x)$ at least for D sufficiently large and give a precise estimate for $x = 0 \cdots 0$. We also give some conjectures, induced by the computations done for graphs up to diameter 25.

4.1 Introduction and Notation

Graphs are widely used in the design and analysis of computer networks. A vertex in the graph denotes a node or processor in the corresponding network, and an edge denotes a communication link between two nodes. If a network is unidirectional, i.e., the communication links in the network are unidirectional, a directed graph can be used. Whereas for bidirectional network (undirected) graphs are used.

Let $G = (V, E)$ be a (strongly) connected graph (digraph). V denotes the set of vertices; E denotes the set of edges (or arcs for digraphs). We will denote by $N = |V|$ the number of vertices in G. A path (resp. a dipath) between two vertices x and y (from x to y) in a graph (resp. a digraph) G is a sequence of vertices $x = x_1; x_2; \cdots; x_k = y$ such that two consecutive vertices in the sequence are joined by an edge (resp. an arc). The length of a shortest path (resp. dipath) between x and y (from x to y) is called the distance and is denoted by $d(x, y)$. Note that in the case of digraphs it is not a classical distance as $d(x, y)$ might be different from $d(y, x)$.

Mean eccentricity

Definition 1 *The eccentricity of a vertex x is defined [5] as the distance to the farthest node from this vertex :* $e(x) = \max\{d(x, y) : y \in V\}$

Definition 2 *The mean eccentricity of the vertex x, denoted $\bar{e}(x)$, is the average distance from x to all the others vertices.* $\bar{e}(x) = \frac{1}{N-1} \sum_{y \in V-\{x\}} d(x,y)$

Definition 3 *The weight of a tree is equal to the sum of all the distances from the root to the others vertices.*

Remark 1 *The mean eccentricity of a vertex is the weight of the shortest paths tree rooted at this vertex, divided by $N-1$.*

de Bruijn networks

Definition 4 *Given a word $x = x_1 \cdots x_D$ on an alphabet $\mathcal{A}$ with D letters and $\lambda \in \mathcal{A}$, the operation :*

- $x_1 \cdots x_D \longrightarrow x_2 \cdots x_D \lambda$ *is called a left shift*
- $x_1 \cdots x_D \longrightarrow \lambda x_1 \cdots x_{D-1}$ *is called a right shift*

We define the directed and the undirected network.

The *de Bruijn* **digraph** $B(d,D)$ **:** the vertices are the words of length D over an alphabet of d symbols. There is an arc from each vertex $x_1 x_2 \cdots x_D$ to $x_2 \cdots x_D \lambda$ for every λ in the alphabet. That is successors are obtained by left-shift operations. These digraphs are characterized by the following parameters :

- the in-degree and the out-degree are equal to d
- the number of vertices is $N = d^D$
- the diameter is $D = \log_d(N)$

The undirected *de Bruijn* **graph** $UB(d,D)$ **:** it is obtained from $B(d,D)$ by removing the orientation of the edges and the loops. Neighboors are obtained by either a left shift or a right shift.

We refer the reader to one of the two recent surveys concerning *de Bruijn* networks written by Bermond and Peyrat [3] and Samatham and Pradhan [11] or to the recent book of Leighton [9].

These networks have been discovered by many authors and are named after de Bruijn [6]. They are sometimes also called Good graphs [8].

They present many attractive features. In particular they are among the best known networks for a given degree and diameter (see the survey [1] for more details on this problem known as the (d,D) or (Δ,D) graph problem).

They have a good vulnerability, being able to tolerate up to $d-1$ faults in the directed case and $2d-2$ in the undirected case, while the diameter can still be left small (see [2]).

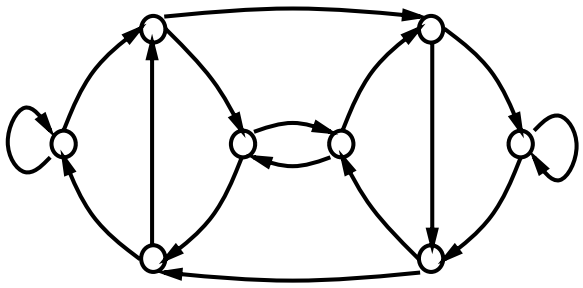

Figure 4.1: $B(2,3)$

They are adequate for various applications as one can embed them in linear arrays, rings, and complete binary trees. They can also emulate without loss of time shuffle-exchange or hypercubes for the class of ascend-descend algorithms. They have also many others interesting properties like easy greedy routing.

In the case of a digraph, there is a unique shortest dipath from a given vertex $x = x_1 \cdots x_D$ to a vertex $y_1 \cdots y_D$. To find the distance $d(x, y)$ and the shortest dipath one has to find the smallest i such that $x_{i+1} \cdots x_D = y_1 \cdots y_{D-i}$. The distance is then i, and the shortest path is obtained by doing the left shifts introducing successively $y_{D+1-i}, \cdots, y_D$.

Using this fact, one can have an easy algorithm to compute $d(x, y)$ and so $\bar{e}(x)$ for any x. But unfortunately that does not give a closed formula. However, in the case of *de Bruijn* digraphs we will be able in section 4.2 to determine lower bound and upper bound (relatively close) for $\bar{e}(x)$, prove that these bounds are reached and determine the vertices reaching them.

For undirected graphs, the determination of the distance and shortest path are less simple, we will use the fact that one can always find a shortest path described as obtained by a suite of at most 3 directed paths to give a method of computation. We will apply this technique to compute $\bar{e}(0 \cdots 0)$. We will also give some computations wich support the fact that $(0 \cdots 0)$ is a vertex of maximal mean eccentricity.

4.2 Directed case

A particular instance of the directed graph is the binary *de Bruijn* digraph. The alphabet is $\{0, 1\}$ and so, the vertices are the words of D bits on this alphabet. It can be viewed as the superposition of the state diagrams for all possible shift registers of D stages [8].

Example : for $D = 3$ we obtain the digraph represented on figure 4.1.

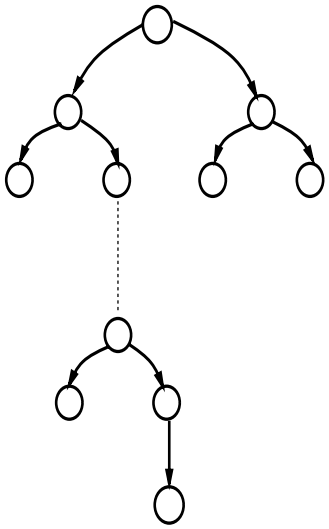

Figure 4.2: Vertex of minimal eccentricity

The following equalities will be extensively used in what follows :

$$\sum_{k=0}^{p} d^k = \frac{d^{p+1}-1}{d-1}. \tag{4.5}$$

$$\sum_{k=1}^{p} k d^{k-1} = \frac{pd^{p+1}-(p+1)d^p+1}{(d-1)^2}. \tag{4.6}$$

Lower bound

We are now looking for the vertex of minimal mean eccentricity. The best we can have for a vertex is described on figure 4.2 in the case $d = 2$. Due to its out-degree, a vertex has at most d vertices at distance 1, and so d^2 vertices at distance 2, $\cdots$, and d^k vertices at distance k (level k of the tree rooted at this vertex).

As $1 + d + d^2 + \cdots + d^{D-1} = \frac{d^D-1}{d-1}$ we have still at least $x = d^D - \frac{d^D-1}{d-1}$ vertices wich be at level D.

The weight of a shortest path spanning tree is therefore at least :

$$\begin{aligned}
\sum_{k=1}^{D-1} kd^k + xD &= d\sum_{k=1}^{D-1} kd^{k-1} + xD \\
&= d\left[\frac{(D-1)d^D - Dd^{D-1}+1}{(d-1)^2}\right] + xD \text{ applying (4.6)} \\
&= D\left[\frac{d^{D+1}-d^D}{(d-1)^2} + x\right] - \frac{d^{D+1}-d}{(d-1)^2}
\end{aligned}$$

$$
\begin{aligned}
&= D\left[\frac{d^D}{d-1} + d^D - \frac{d^D-1}{d-1}\right] - d\frac{d^D-1}{(d-1)^2} \\
&= D\left[\frac{1}{d-1} + N\right] - \frac{d}{(d-1)^2}(N-1) \\
&= \frac{D}{d-1} + DN - \frac{d}{(d-1)^2}(N-1) \\
&= \frac{D}{d-1} + D(N-1) - \frac{d}{(d-1)^2}(N-1) + D.
\end{aligned}
$$

We obtain the mean eccentricity by dividing by $N-1$, and we have the following proposition :

Proposition 4.2.1

$$\bar{e}(x) \geq D - \frac{d}{(d-1)^2} + \frac{d}{d-1}\varepsilon(D) \tag{4.7}$$

Where $\varepsilon(D) = \frac{D}{N-1}$.

Proposition 4.2.2 *The vertices of the form* $a\cdots aa'$ *reach the bound 4.7.*

Proof: the vertices at distance k from $a\cdots aa'$ are of the form : $a_1\cdots a_{D-k-1}a_{D-k}a_{D-k+1}\cdots a_D$ with $a_1 = \cdots = a_{D-k-1} = a$, $a_{D-k} = a'$ and $a_{D-k+1}, \cdots, a_D$ can be any letter of the alphabet. None of these vertices are at distance j, $j < k$. Their number is d^k and so the bound is reached. □

Upper bound

We will now give an upper bound attained by the vertices $a\cdots a$.

Lemma 1 *A tree of degree at most* d *with* d^D *vertices and* D *levels is such that:*

$$\sum_{k=i}^{D} l_k \leq \sum_{k=i}^{D} d^{k-1}(d-1) \quad \forall i,\ 1 \leq i \leq D$$

where l_k *is the number of vertices at distance* k *from the root.*

Proof: suppose the lemma is not true and let i_0 be the largest i such that $\sum_{k=i}^{D} l_k > \sum_{k=i}^{D} d^{k-1}(d-1)$. Then $l_{i_0} > d^{i_0-1}(d-1)$ and

$$
\begin{aligned}
l_{i_0} > d^{i_0-1}(d-1) &\Rightarrow l_{i_0-1} > d^{i_0-2}(d-1), \\
&\vdots \\
l_2 > d(d-1) &\Rightarrow l_1 > d-1.
\end{aligned}
$$

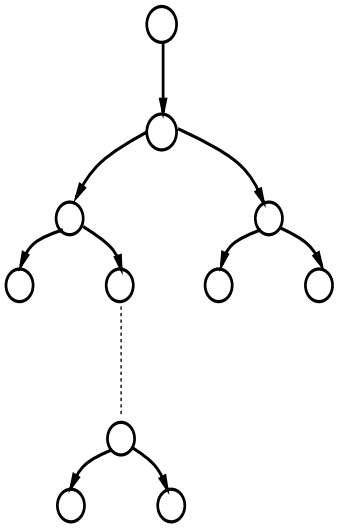

Figure 4.3: Vertex of maximal eccentricity

Furthermore, there is at least $i_0 - 1$ more vertices between the root and level $i_0 - 1$ in that tree (to connect the additionnal vertex (vertices) of level i_0 to the root). By definition of i_0, the number of vertices in the $i_0 - 1$ first levels is :

$$\sum_{k=1}^{i_0-1} l_k \geq i_0 - 1 + \sum_{k=1}^{i_0-1} d^{k-1}(d-1).$$

The number of vertices from level i_0 to D is :

$$\sum_{k=i_0}^{D} l_k > \sum_{k=i_0}^{D} d^{k-1}(d-1) \geq 1 + \sum_{k=i_0}^{D} d^{k-1}(d-1). \tag{4.8}$$

Thus, the total number of vertices in the tree wich is d^D is also :

$$\sum_{k=1}^{D} l_k \geq i_0 + d^D - 1.$$

The above inequality is true only if $i_0 = 0$, and this is not possible. □

Proposition 4.2.3 *The weight of any shortest path tree the $B(d, D)$ is bounded by :*

$$\sum_{h=1}^{D} h l_h \leq \sum_{h=1}^{D} h d^{h-1}(d-1).$$

Proof: lemma 1 for $i = 1, \cdots, D$

$$\sum_{h=1}^{D} l_h \quad \leq \quad \sum_{h=1}^{D} d^{h-1}(d-1),$$

$$\begin{aligned}
\sum_{h=2}^{D} l_h &\leq \sum_{h=2}^{D} d^{h-1}(d-1), \\
&\vdots \quad\quad \vdots \\
\sum_{h=D}^{D} l_h &\leq \sum_{h=D}^{D} d^{h-1}(d-1).
\end{aligned}$$

if we sum each part we get

$$\sum_{h=1}^{D} h l_h \leq \sum_{h=1}^{D} h d^{h-1}(d-1)$$

□

The corresponding weight is :

$$\begin{aligned}
\sum_{k=1}^{D} k d^{k-1}(d-1) &= (d-1)\left[\frac{D d^{D+1} - (D+1)d^D + 1}{(d-1)^2}\right] \\
&= D d^D - \frac{d^D - 1}{d-1} \\
&= D(N-1) + D - \frac{N-1}{d-1},
\end{aligned}$$

thus :

$$\bar{e}(x) \leq D - \frac{1}{d-1} + \varepsilon(D) \tag{4.9}$$

Proposition 4.2.4 *the bound is reached by and only by the vertices of the form* $a \cdots a$.

Proof:

$\Rightarrow$ vertices of the form $a \cdots a$ satisfy $l_k = d^{k-1}(d-1)$ and so $\bar{e}(a \cdots a)$ realizes the bound. All the inequalities of the proof of lemma 1 become equalities.

$\Leftarrow$ By lemma 1, we need to have $l_1 = d-1$. Only the loops have this property.

□

We obtain :

$$\bar{e}(a \cdots a) = D + \frac{D}{N-1} - \frac{1}{d-1},$$

4.2.1 Conclusion

We conclude that the mean eccentricity of a vertex in the directed *de Bruijn* graph is :

$$D - \frac{d}{(d-1)^2} + \frac{d}{d-1}\varepsilon(D) \leq \bar{e}(x) \leq D - \frac{1}{d-1} + \varepsilon(D).$$

The lower bound and the upper bound are close to the diameter. Even if we use D shifts to route instead of the optimal routing, the global behavior of the network should not be affected too much. The routing scheme in D steps corresponds to a simple bit-erosion of the destination address, a very simple routing function that could be implemented with only a very few VLSI components on chip.

4.3 Undirected case

Introduction

Now we are interested in $UB(d, D)$. This case is much more difficult than the previous one (directed case). Indeed, the distance trees associated to the vertices are of various forms, and even 2 vertices of same weight can have different trees. For instance, in $UB(2, 6)$, vertex 1 has tree [[4],[7],[13],[22],[17],[0]], whereas vertex 4 has tree [[4],[9],[15],[17],[13],[5]], although the weight of both trees are 230 !

Moreover large computations on binary graphs of up to diameter 25 were not sufficient to get the idea of the vertex of minimal mean eccentricity. It appears obvious that vertex $a \cdots a$ have the maximal mean eccentricity, but we were not able to prove it and leave that as a conjecture.

Conjecture 4.3.1 *The vertex* $0 \cdots 0$ *is the vertex of maximal mean eccentricity.*

4.3.1 R and L sequences

Given a path in the $UB(d, D)$, the edges from one vertex to its neighboor on the path correspond to left or right shifts (see definition 4) on the label of the vertex. Thus one can describe a path by the sequence of corresponding shifts. The length of a path connecting x to y (distance between x and y) is equal to the length of the corresponding shifts sequence that must be applied on x to reach y.

As the undirected de Bruijn graph is defined from its directed graph, a path can be described as the concatenation of directed paths with alternating directions, corresponding respectively to sequences of left shifts or right shifts. In this section we will show that a shortest path of $UB(d, D)$ is made of at most 3 such directed paths (this result was already established in [4, p92] and [10, annexe 3.2], therefore, for the sake of completness we give a proof), and then we will derive interesting results for the case of paths issued from vertex $a \cdots a$.

Notations

- Let L be the operation consisting in a sequence of l left shifts, introducing a suffix $b = b_1, \cdots, b_l$ with $|L| = l$

  Given a vertex $x = x_1 \cdots x_D$, then by applying L we get :
  $x_1 \cdots x_D \xrightarrow{L} x_{l+1} \cdots x_D b_1 \cdots b_l$

- Let R be the operation consisting in a sequence of r right shifts, introducing a prefix $a = a_r \cdots a_1$ with $|R| = r$

  Given a vertex $x = x_1 \cdots x_D$, then by applying R we get :
  $x_1 \cdots x_D \xrightarrow{R} a_r \cdots a_1 x_1 \cdots x_{D-r}$

A finite path of the de Bruijn can be denoted $L_1R_1L_2R_2 \cdots L_nR_n$ with $|L_i| = l_i$ and $|R_i| = r_i$, meaning that we successively apply L_1, then R_1, then L_2 and so on. The length of the path is equal to $l_1 + r_1 + l_2 + r_2 + \cdots$. Note that one can have $l_i = 0$ or r_i.

4.3.1.1 Characterisation of the shortest paths

Lemma 2 *If a path of the form LRL', ($l > 0, r > 0$ and $l' > 0$) is a shortest path, then $l < r$ (respectively if a path of the form RLR', ($l > 0, r > 0$ and $l' > 0$) is a shortest path, then $r < l$).*

Proof:

Given a path from x to y of the form $L_1R_1L_1'$, suppose that $l \geq r$, then after the application to x of L_1 we reach $x_{l+1} \cdots x_D b_1 \cdots b_l$.

Then the application of R_1 leads to the vertex $a_r \cdots a_1 x_{l+1} \cdots x_D b_1 \cdots b_{l-r}$.

The first shift of L_1' leads to the vertex $a_{r-1} \cdots a_1 x_{l+1} \cdots x_D b_1 \cdots b_{l-r}\beta$.

But the vertex $a_{r-1} \cdots a_1 x_{l+1} \cdots x_D b_1 \cdots b_{l-r}\beta$ could be reached by a shorter path L_2R_2 with $l_2 = l_1$ and $r' = r - 1$, (L_2 introduces the word $b_1 \cdots b_{l-r}\beta \cdots$ and R_2 introduces the word $a_{r-1} \cdots a_1$)

Thus $L_1R_1L_1'$ was not a shortest path. □

Proposition 4.3.2 *A shortest path of $UB(d, D)$ is made of the concatenation of at most 3 directed paths, that is the path is of the form L or R or LR or RL or LRL' or RLR'.*

Proof:

Suppose that a shortest path from x to y is made of more than 3 directed paths, then a part of this shortest path is of the form $L_1R_2L_3R_4$ (or $R_1L_2R_3L_4$).

If $L_1R_2L_3R_4$ is a shortest path, then $R_2L_3R_4$ is also a shortest path and then by lemma 2 we get $r_2 < l_3$.

If $L_1R_2L_3R_4$ is a shortest path from x to y, then we have $\bar{R}_4\bar{L}_3\bar{R}_2\bar{L}_1$ is a shortest path from y to x, where $\bar{L}$ is defined as the inverse of L, that is, the

operation consisting in a right shift of size l (we take the same edges of the path, but in reverse order). The notation $\bar{R}$ has the same meaning.

If $\bar{R}_4\bar{L}_3\bar{R}_2\bar{L}_1$ is a shortest path, then $\bar{L}_3\bar{R}_2\bar{L}_1$ is also a shortest path and we conclude (lemma 2) $\bar{r_2} > \bar{l_3}$. $\bar{r_2} = r_2$ and $\bar{l_3} = l_3$, thus $r_2 > l_3$, wich is a contradiction. □

Proposition 4.3.3 *A shortest path between a vertex x and the vertex $0\cdots0$ is of the form L, R, LR (with $r < l$) or RL (with $l < r$).*

Proof: Without loss of generality, suppose that the path from $0\cdots0$ to x is of the form LRL'. If LRL' is a shortest path we must have $l < r$. So, the vertex reached after the operation LR is $a_r\cdots a_1 0\cdots0$, which could have been reached directly by the operation R. The path RL' is thus shorter. □

4.3.1.2 Characterisation of the set of vertices at distance at most $D-h$ from $0\cdots0$

Let this set be $E^h = \{x|\ d(0,x) \leq D-h\}$

Proposition 4.3.4

$$E^h = \bigcup_{0\leq k\leq \left\lfloor \frac{D-h}{2} \right\rfloor} E_h^k$$

where : $E_h^k = L_k^h \cup R_k^h$ with

- $L_k^h = \{x | x = awb$ with $w = 0\cdots0$ and $|w| = k+h, |a| = k\}$
- $R_k^h = \{x | x = awb$ with $w = 0\cdots0$ and $|w| = k+h, |b| = k\}$

Proof:

- $x \in E_k^h \Longrightarrow d(0,x) \leq D-h$.

  Suppose $x \in L_k^h$; $x = awb$, with $w = 0\cdots0$ and $|w| = k+h$ and $|a| = k$. We can reach x by an operation L of size $D-(k+h)$ applied to $0\cdots0$ to introduce the suffix b, and then an operation R of size k to introduce the prefix a. The length of the corresponding path is $D-(k+h)+k = D-h$; thus x is at distance at most $D-h$ from $0\cdots0$. We can prove the same result for R_k^h.

- $d(0,x) \leq D-h \Longrightarrow x \in E_k^h$ for some k, $k \leq \left\lfloor \frac{D-h}{2} \right\rfloor$.

  If $d(0,x) \leq D-h$, then (by Proposition 4.3.3) there exists a path (not necesseraly a shortest one) LR (or RL) with $r+l \leq D-h$ between $0\cdots0$ and x. Without loss of generality, suppose that this path is of the form LR. Thus $x = a_r\cdots a_1 0\cdots0 b_1\cdots b_{l-r} = aw'b'$.

Then :

$$\begin{aligned} |w'| &= D - |a| - |b'| \\ &= D - r - (l - r) \\ &\geq D - (D - h) + r = r + h. \end{aligned}$$

Let us take $k = r$, then $|a| = k$ and $|w'| \geq k + h$. Let w be the prefix of size $k + h$ of w'. Then $x = awb$ with $|w| = k + h$ and so $x \in L_k^h$

□

4.3.1.3 Computation of the cardinality of E^h

If we were able to count $|E^h|$, the mean eccentricity of $0 \cdots 0$ would be trivial. Here, we are only able to give bounds for $|E^h|$, but these bounds are close enough to give a good approximation of $\bar{\epsilon}(0)$.

The method presented here consists in counting E_k^h, for $k = 0 \cdots \left\lfloor \frac{D-h}{2} \right\rfloor$. At each step we try to take into account only new vertices, that is those which were not counted in a previous set. This count is made in terms of proportions, denoted e_k^h, where

$$e_k^h = \frac{|E_k^h - \bigcup_{i<k} E_i^h|}{N}.$$

The total proportion of a set E^h will be equal to :

$$e^h = \sum_{k=0}^{\frac{D-h}{2}} e_k^h.$$

Then we will compute :

$$\begin{aligned} \bar{\epsilon}(0 \cdots 0) &= \frac{1}{N-1} \sum_{h=0}^{D-1} (D-h) N (e^h - e^{h+1}) \\ &= \frac{N}{N-1} \Big(D \times e^0 - D \times e^1 + (D-1) \times e^1 + \cdots + (D-h) \times e^h \\ &\quad -(D-h-1) \times e^h + \cdots + e^{D-1} - e^D \Big) \\ &= \frac{N}{N-1} \left(D - \sum_{h=1}^{D-1} e^h \right) \end{aligned}$$

We introduce the method with the computation of the first values of k, and then we generalize for e_k^h. A more precise computation will be done for e^1 and e^2, as they are important factors of the final computation of the eccentricity of $0 \cdots 0$. Note that $e^0 = 1$ as all the vertices of a graph are at distance at most the diameter. Note also that in all the computations below, we suppose that D is large enough.

In what follows we will intensively use basic principles of counting the proportion of words satisfying different conditions (one can think also in terms of probabilities).

Let us denote by p_i the proportion of words satisfying a condition C_i.

Two conditions C_i and C_j are independent if they concern disjoint subgroups of letters of the word. $\bar{C}_i$ will denote the inverse condition of C_i. Two conditions C_i and C_j are incompatible if they cannot occur simultaneously (in particular they should concern subgroups of letters non disjoint).

Basic lemmas:

1. if C_i and C_j are independent then the proportion of words satisfying both C_i and C_j is $p_i \times p_j$
2. the proportion of words satisfying $\bar{C}_i$ is $1 - p_j$
3. if C_i and C_j are incompatible the proportion of words satisfying both $\bar{C}_i$ and $\bar{C}_j$ is $1 - p_i - p_j$

Computation of $e_0^h = \frac{|E_0^h|}{N}$.

We can derive from Proposition 4.3.4 that the vertices of E_0^h (case $k = 0$) are of the form :

- $x = \underbrace{0 \cdots 0}_{h} X$ if $x \in L_0^h$, where X is any sequence of letters. There is a proportion of 2^{-h} vertices of this kind.
- $x = X \underbrace{0 \cdots 0}_{h}$ if $x \in R_0^h$. The proportion of such vertices is also 2^{-h}.
- But we have counted twice some vertices, namely the $x = \underbrace{0 \cdots 0}_{h} X \underbrace{0 \cdots 0}_{h}$, that is, when $x \in L_0^h \cap R_0^h$. (Here we consider the case $h < D/2$, if $h \geq D/2$, then $x = 0$). Their proportion is 2^{-2h}.

Total : $e_0^h = 2^{-h+1} - 2^{-2h}$.

Computation of e_1^h

Now we want to count the vertices of E_1^h, $h \geq 1$, that we have not counted before, that is $|E_1^h - E_0^h|$. Indeed, as we want to sum the e_i^h, no redundance is allowed.

In term of proportion : $e_1^h = \frac{|E_1^h - E_0^h|}{N}$.

We derive again from Proposition 4.3.4 that the vertices of E_1^h (case $k = 1$) are of the form :

- $x = a \underbrace{0 \cdots 0}_{h+1} X$ if $x \in L_1^h$ and we have two remarks on x :

- $a = 1$, otherwise $x \in L_0^h$. Therefore, there is a proportion of $2^{-(h+2)}$ vertices of this pattern ($h+2$ letters are fixed, a '1' and $h+1$ '0').
- Among these vertices, all those finishing with h zeros belong to R_0^h, their proportion is 2^{-h}. So, $|L_1^h - E_0^h| = 2^{-(h+2)}[1 - 2^{-h}]N$.

- $x = X\underbrace{0\cdots0}_{h+1}b$ if $x \in R_1^h$. The computation is the same as in the case of L_1^h.

- $x = 1\underbrace{0\cdots0}_{h+1}X\underbrace{0\cdots0}_{h+1}1$ if $x \in L_1^h \cap R_1^h$, **and they do not belong to** E_0^h.

  Here we have a proportion of $2^{-2(h+2)}$.

Total : $e_1^h = 2[2^{-(h+2)} - 2^{-(2h+2)}] - 2^{-2(h+2)}$.

$$e_1^h = 2^{-(h+1)} - 9 \times 2^{-2(h+2)}$$

Computation of e_2^h.

We want now to compute the vertices of E_2^h, $h \geq 2$ that we have not counted before, that is $|E_2^h - E_1^h - E_0^h|$, a proportion $e_2^h = \frac{|E_2^h - E_1^h - E_0^h|}{N}$

We derive again from Proposition 4.3.4 that the vertices of E_2^h (case $k = 2$) are of the form :

- $x = a1\underbrace{0\cdots0}_{h+2}X$ where a is any letter and X is any sequence, if $x \in L_2^h$.

  Therefore, there is a proportion of $2^{-(h+3)}$ vertices of this kind ($h+3$ letters are fixed). We must remark here that the case $h = 1$ will be studied in another section, hence the case $a = 0$ is not a problem.

  Among these vertices, all those finishing with h zeros belong to R_0^h, and all those finishing with $h+1$ zeros and a '1' belong to R_1^h ; their proportion is respectively 2^{-h} and $2^{-(h+2)}$.

  So, $|L_2^h - E_1^h - E_0^h| = 2^{-(h+3)}[1 - 2^{-h} - 2^{-(h+2)}]N$

  The same count is done for $|R_2^h - E_1^h - E_0^h|$.

- $x = a1\underbrace{0\cdots0}_{h+2}X\underbrace{0\cdots0}_{h+2}1b$ if $x \in L_2^h \cap R_2^h$,

  and they do not belong to E_0^h **if** $h \geq 2$. If $h = 1$, we have already subtracted E_0^1.

  Here we have a proportion of $2^{-2(h+3)}$ redundant vertices.

Total : $e_2^h = 2[2^{-(h+3)}(1 - 2^{-h} - 2^{-(h+2)})] - 2^{-2(h+3)}, h > 1.$

$$e_2^h = 2^{-(h+2)}(1 - 21 \times 2^{-(h+4)}), h > 1.$$

We will first compute the case $h = 1$ due to the condition above. Then, we will derive e^2 from e^1. The aim is to introduce the method, and to have a better precision for these important factors of the final computation. The result for the general case e^h, will come after.

4.3.1.4 Computation of e^1.

We can use the same method to evaluate exactly e^1_k. Recall that this is valid only for D large enough. Furthermore, to have an explicit evaluation of e^1, we will have to give approximation. According the precision one wants, the computation can be done as far as needed. Here we will do them exactly for $k \leq 11$.

The vertices in L^1_k are of the form :

$$x = a_1 \cdots a_k \underbrace{0 \cdots 0}_{k+1} X b_{D-2k+2} \cdots b_D.$$

We want to compute those not in $\bigcup_{i<k} E^1_i$. We will have two kinds of condition, those of left type due to the fact $x \notin L^1_i$ implying restrictions on $a_1 \cdots a_k$, and those of right type due to the fact $x \notin R^1_i$, implying restrictions on $b_{D-2k+2} \cdots b_D$. Note that if $D \geq k + k + 1 + 2k - 1 = 4k$ these conditions are on disjoint subsets.

Let us first consider the conditions of left type. If $x \in L^1_i$ then $a_{i+1} \cdots a_{2i+1} = 0 \cdots 0$. In fact we present the conditions in a way we can compute the proportion of vertices.

(l^1_{k-1}): $a_k \neq 0$, otherwise $x \in L^1_{k-1}$	
(l^1_0): $a_1 \neq 0$, otherwise $x \in L^1_0$	
(l^1_1): $a_2 a_3 \neq 00$, otherwise $x \in L^1_1$	(only for $k \geq 4$)
(l^1_2): $a_2 a_3 a_4 a_5 \neq 1000$, otherwise $x \in L^1_2 - L^1_1$	(only for $k \geq 6$)
(l^1_3): $a_3 \cdots a_7 \neq 10000$, otherwise $x \in L^1_3 - L^1_2 - L^1_1$	(only for $k \geq 8$)
(l^1_4): $a_4 \cdots a_9 \neq 100000$, otherwise $x \in L^1_4 - L^1_3 - L^1_2$ **and** $a_2 a_3 \neq 00$, otherwise $x \in L^1_1$	(only for $k \geq 8$)
and more generally, for $p \geq 1$ we have :	
(l^1_p): $a_p \cdots a_{2p+1} \neq 10 \cdots 0$, and $x \notin \bigcup_{2i+1 \leq p} L^1_i$	(only for $2p + 2 \leq k$)

Moreover, we have conditions on the suffix $b_{D-2k+2} \cdots b_D$ that we call *conditions of right type*, and they are listed below :

(r^1_0): $b_D \neq 0$, otherwise $x \in R^1_0$	
(r^1_1): $b_{D-2} b_{D-1} \neq 00$, otherwise $x \in R^1_1$	
and more generally, for $p \geq 1$ we have :	
(r^1_p): $b_{D-2p} \cdots b_{D-p+1} \neq 0 \cdots 01$, and $x \notin \bigcup_{i \leq p-1} R^1_i$	(only for $p < k$)

Recall that these conditions are disjoint only if D is large. For example condition (r^1_p) applies only if $D \geq k + k + 1 + 2p + 1$. So all the conditions apply if

$D \geq 4k$ and in that case lead to the following value :

$$|L_k^1| = \alpha_k \beta_k \times 2^{-(k+4)} N,$$

where :

- $2^{-(k+4)}$ is due to the fact that we have $k+4$ fixed letters, namely the $k+1$ zeroes, $a_k = 1$ (condition (l_{k-1}^1)), $a_1 = 1$ (condition (l_0^1)) and $b_D = 1$ (condition (r_0)).

- The value β comes from conditions (r_p^1). $\beta_k = \frac{3}{4}$ for $k = 2$.

  $$\beta_{k+1} = \beta_k - \beta_{\lfloor \frac{k}{2} \rfloor} 2^{-(k+2)}$$

  $\sum_k^\infty \beta_k \geq \left(1 - \frac{1}{4} - \frac{1}{2^4} - \frac{1}{2^5} \cdots - \frac{1}{2^{k+1}}\right)$ Thus $\beta_k \geq \beta_{\min} = \frac{5}{8}$.

- The value α follows from condition (l_p^1) $\alpha_k = 1$ for $k = 2, 3$, $\alpha_k = \frac{3}{4}$ for $k = 4, 5$.

  $$\alpha_{2k+1} = \alpha_{2k} = \beta_k$$

  $\alpha_k \geq \left(1 - \frac{1}{4} - \frac{1}{2^4} - \frac{1}{2^5} \cdots - \frac{1}{2^{\lceil (k+1)/2 \rceil}}\right)$

  Thus $\alpha_k \geq \left(1 - \frac{1}{4} - \frac{1}{8}\right) = \alpha_{\min} = \frac{5}{8}$.

A similar count is done for R_k^1, so we will have twice the above quantity. Note that in the case of R_k^1, left and right conditions are exchanged.

The vertices in $L_k^1 \cap R_k^1 - \bigcup_{i<k} E_i^1$ are of the form :

$$x = 1 a_2 \cdots a_{k-1} 1 \underbrace{0 \cdots 0}_{k+1} X \underbrace{0 \cdots 0}_{k+1} 1 b_{D-k+2} \cdots b_{D-1} 1$$

with the same conditions as the former ones (left conditions and the corresponding ones for the count of R_k^1), that is : (l_p^1) and (r_p^1) for $2p+2 \leq k$ (conditions (l_p^1) or (r_p^1) for $2p+2 > k$ cannot apply due to the length of the prefix or suffix of x). So :

$$|L_k^1 \cap R_k^1| = \alpha_k^2 \times 2^{-(2k+6)} N,$$

$$e_k^1 = \alpha_k \beta_k \times 2^{-(k+3)} - \alpha_k^2 \times 2^{-(2k+6)}. \tag{4.10}$$

Furthermore we can compute the exact value of as many e_k^1 we want and give an upper bound using upper bounds on α_k and β_k. The values for $k = 0$ and $k = 1$ were computed in the general case of e_k^h and the values are $e_0^1 = \frac{3}{4}$, and $e_1^1 = \frac{7}{64}$.

Now we give the results for k from 2 up to 11, using (4.10), in the following table :

k	α	β	e_k^1
2	1	3×2^{-2}	23×2^{-10}
3	1	11×2^{-4}	43×2^{-12}
4	3×2^{-2}	21×2^{-5}	999×2^{-18}
5	3×2^{-2}	165×2^{-8}	1959×2^{-20}
6	11×2^{-4}	327×2^{-9}	56903×2^{-26}
7	11×2^{-4}	2605×2^{-12}	113223×2^{-28}
8	21×2^{-5}	5199×2^{-13}	862407×2^{-32}
9	21×2^{-5}	20775×2^{-15}	1722567×2^{-34}
10	165×2^{-8}	41529×2^{-16}	13443695×2^{-38}
11	165×2^{-8}	664299×2^{-20}	26878575×2^{-40}

For $k \geq 12$ we compute the two following sums :

Lower bound :

$$\begin{aligned}\sum_{k\geq 12} e_k^1 &\geq \sum_{k\geq 12} \alpha_{\min}\beta_{\min} 2^{-(k+3)} - \alpha_{12}^2 2^{-(2k+6)} \\ &\geq \sum_{k\geq 12} \left(\frac{5}{8}\right)^2 2^{-(k+3)} - (327 \times 2^{-9})^2 \times 2^{-(2k+6)} \\ &\geq 0.00002384135139\end{aligned}$$

Upper bound : we take $k = 12$ for the values of α_k and β_k.

$$\begin{aligned}\sum_{k\geq 12} e_k^1 &\leq \sum_{k\geq 12} \alpha_{12}\beta_{12} 2^{-(k+3)} - \alpha_{\min}^2 2^{-(2k+6)} \\ &\leq \sum_{k\geq 12} 327 \times 2^{-9} \times 1328433 \times 2^{-21} \times 2^{-(k+3)} - \left(\frac{5}{8}\right)^2 \times 2^{-(2k+6)} \\ &\leq 0.00002469215915\end{aligned}$$

This leads to the bounds :

$$0.8997103140 \leq e^1 \leq 0.8997111648 \tag{4.11}$$

In others words, about 10% of vertices are at distance D from 0.

4.3.1.5 Computation of e^2.

The vertices in $L_k^2 - \bigcup_{i<k} E_i^2$ are of the form :

$$x = a_1 \cdots a_k \underbrace{0 \cdots 0}_{k+2} X b_{D-2k+1} \cdots b_D.$$

For *conditions of left type*:

(l^2_{k-1}): $a_k \neq 0$, otherwise $x \in L^2_{k-1}$	
(l^2_0): $a_1a_2 \neq 00$, otherwise $x \in L^2_0$	(only for $k \geq 3$)
(l^2_1): $a_1a_2a_3a_4 \neq 1000$, otherwise $x \in L^2_1 - L^2_0$	(only for $k \geq 5$)
(l^2_2): $a_2 \cdots a_6 \neq 10000$, otherwise $x \in L^2_2 - L^2_1 - L^2_0$	(only for $k \geq 7$)
and more generally, for $p \geq 2$ we have :	
(l^2_p): $a_p \cdots a_{2p+2} \neq 10 \cdots 0$, and $x \notin \bigcup_{2i+2 \leq p} L^2_i$	(only for $2p+3 \leq k$)

For *conditions of right type*:

(r^2_0): $b_{D-1}b_D \neq 00$, otherwise $x \in R^2_0$	
and more generally, for $p \geq 1$ we have :	
(r^2_p): $b_{D-(2p+1)} \cdots b_{D-p+1} \neq 0 \cdots 01$, and $x \notin \bigcup_{i \leq p-1} R^2_i$	(only for $p < k$)

One can remark that these conditions are very similar to the ones stated for e^1. In fact the only difference from e^1 is that we do not have condition (l^1_0) and that (l^2_{p-1}) is equivalent to (l^1_p) (exclusion of the same proportion of vertices), and (r^2_{p-1}) is equivalent to (r^1_p).

Hence we have for L^2_k the same parameters α and β than for L^1_{k+1}, but in L^2_k we have 2 fixed letters less (as conditions (l^1_0) and (r^1_0)) have the same number of zeroes, namely $k+2$). So $e^2_k = 4 \times e^1_{k+1}$,

$$\begin{aligned} \sum_{k \geq 0} e^2_k &= 4 \sum_{k \geq 0} e^1_{k+1}, \\ e^2 &= 4\left(e^1 - e^1_0\right), \\ e^2 &= 4\left(e^1 - \frac{3}{4}\right). \end{aligned}$$

$$\begin{aligned} 4\left(0.8997103140 - \frac{3}{4}\right) \leq e^2 \leq\ & 4\left(0.8997111648 - \frac{3}{4}\right), \\ 0.5988412561 \leq e^2 \leq\ & 0.5988446593. \end{aligned} \tag{4.12}$$

Unfortunately, we are not able to derive so easily e^3 from e^2.

4.3.1.6 Computation of e^h, for $h > 2$

We are now ready to compute e^h for any $h \geq 2$. As in the case where $h = 1$ or $h = 2$, the vertices in L^h_k and not in $\bigcup_{i<k} E^h_i$ are of the form :

$$x = a_1 \cdots a_k \underbrace{0 \cdots 0}_{k+h} X b_{D-(2k+h-1)} \cdots b_D$$

- *conditions of left type*:

(l^h_{k-1}): $a_k \neq 0$, otherwise $x \in L^h_{k-1}$

(l^h_0): $a_1 \cdots a_h \neq \underbrace{0 \cdots 0}_{h}$, otherwise $x \in L^h_0$ (only for $k \geq h+1$)

we have in general case,

(l^h_p): $a_p \cdots a_{2p+h} \neq \underbrace{10 \cdots 0}_{p+h}$, and $x \notin \bigcup_{2i+h \leq p} L^h_i$ (only for $k \geq 2p+h+1$)

- *conditions of right type*:

(r^h_0): $b_{D-h+1} \cdots b_D \neq 0 \cdots 0$, otherwise $x \in R^h_0$

(r^2_p): $b_{D-(2p+1)} \cdots b_{D-p+1} \neq 0 \cdots 01$, otherwise $x \in R^2_p$ (only for $p < k$) and $x \notin \bigcup_{i \leq p-1} R^2_i$

(r^h_p): $b_{D-(2p+h-1)} \cdots b_{D-p} \neq 0 \cdots 01$, and $x \notin \bigcup_{i \leq p-1} R^h_i$ $(p < k)$

Hence we get :

$$e^h_k = \alpha_k \beta_k \times 2^{-(k+h)} - \alpha^2_k \times 2^{-2(k+h+1)}, \tag{4.13}$$

where :

- β_k follows from conditions (r^h_p)

$$\beta_{k+1} = \beta_k - \beta_{\lfloor \frac{k-h+1}{2} \rfloor} 2^{-(k+h+1)}$$

$$\beta_k \geq \left(1 - \frac{1}{2^h} - \frac{1}{2^{h+2}} - \frac{1}{2^{h+3}} \cdots - \frac{1}{2^{k+h}}\right) \tag{4.14}$$

$\beta_k \geq \beta_{\min} = \left(1 - \frac{1}{2^h} - \frac{1}{2^{h+1}}\right)$.

- α_k follows from conditions (l^h_p)

$$\alpha_{2k+1} = \alpha_{2k} = \beta_k$$

$$\alpha_k \geq \left(1 - \frac{1}{2^h} - \frac{1}{2^{h+2}} - \frac{1}{2^{h+3}} \cdots - \frac{1}{2^{h+\lceil \frac{k-h}{2} \rceil}}\right) \tag{4.15}$$

$$\text{Values of } \alpha_k : \begin{cases} \alpha_k = 1 & \text{for } 2 \leq k \leq h \\ \alpha_k = 1 - \frac{1}{2^h} & \text{for } k = h+1, h+2 \\ \alpha_k \geq \alpha_{\min} = \left(1 - \frac{1}{2^h} - \frac{1}{2^{h+1}}\right) & \text{for } k \geq h+3 \end{cases}$$

This corresponds to approximations of α and β, but we have done more precise computations for the case of $h = 3$ and $h = 4$

Computation of e^3

$e_0^3 = 0.234375$

k	α	β	e_k^3
1	1	7×2^{-3}	0.05371093750
2	1	27×2^{-5}	0.02612304688
3	1	53×2^{-6}	0.01287841797
4	7×2^{-3}	105×2^{-7}	0.005595922470
5	7×2^{-3}	1673×2^{-11}	0.002789199352
6	27×2^{-5}	3339×2^{-12}	0.001342705451
7	27×2^{-5}	26685×2^{-15}	0.0006708435249
8	53×2^{-6}	53343×2^{-16}	0.0003290860768
9	53×2^{-6}	213319×2^{-18}	0.0001645123812
10	105×2^{-7}	426585×2^{-19}	0.00008147262065
11	105×2^{-7}	1706235×2^{-21}	0.00004073443023

And for $k \geq 12$ we compute the two following sums :

$$\begin{aligned}
\sum_{k\geq 12} e_k^3 &\geq \sum_{k\geq 12} \alpha_{\min}\beta_{\min} 2^{-(k+3)} - \alpha_{12}^2 2^{-2(k+3)} \\
&\geq \sum_{k\geq 12} \left(\frac{13}{16}\right)^2 2^{-(k+3)} - 1673 \times 2^{-11} \times 2^{-2(k+3)} \\
&\geq 0.00004029253271
\end{aligned}$$

$$\begin{aligned}
\sum_{k\geq 12} e_k^3 &\leq \sum_{k\geq 12} \alpha_{12}\beta_{12} 2^{-(k+3)} - \alpha_{\min}^2 2^{-2(k+4)} \\
&\leq \sum_{k\geq 12} 1673 \times 2^{-11} \times 3412365 \times 2^{-22} \times 2^{-(k+3)} - \left(\frac{13}{16}\right)^2 \times 2^{-2(k+4)} \\
&\leq 0.00004056407935
\end{aligned}$$

This leads to the bounds :

$$0.3381421712 \leq e^3 \leq 0.3381424427 \tag{4.16}$$

Computation of e^4

$e_0^4 = 0.12109375$

k	α	β	e_k^4
1	1	15×2^{-4}	0.02905273438
2	1	59×2^{-6}	0.01434326172
3	1	117×2^{-7}	0.007125854492
4	1	233×2^{-8}	0.003551483154
5	15×2^{-4}	465×2^{-9}	0.001662131399
6	15×2^{-4}	14865×2^{-15}	0.0008304370567
7	59×2^{-6}	29715×2^{-15}	0.0004081445368
8	59×2^{-6}	237661×2^{-18}	0.0002040342770
9	117×2^{-7}	475263×2^{-19}	0.0001011431957
10	117×2^{-7}	1900935×2^{-21}	0.00005056926346
11	233×2^{-8}	3801753×2^{-22}	0.00002517599748

And for $k \geq 12$ we compute the two following sums :

$$\begin{aligned}
\sum_{k\geq 12} e_k^4 &\geq \sum_{k\geq 12} \alpha_{\min}\beta_{\min} 2^{-(k+4)} - \alpha_{12}^2 2^{-2(k+5)} \\
&\geq \sum_{k\geq 12} \left(\frac{29}{32}\right)^2 2^{-(k+4)} - (233 \times 2^{-8})^2 \times 2^{-2(k+5)} \\
&\geq 0.00002506368884
\end{aligned}$$

$$\begin{aligned}
\sum_{k\geq 12} e_k^3 &\leq \sum_{k\geq 12} \alpha_{12}\beta_{12} 2^{-(k+3)} - \alpha_{\min}^2 2^{-2(k+4)} \\
&\leq \sum_{k\geq 12} 233 \times 2^{-8} \times 15206779 \times 2^{-24} \times 2^{-(k+4)} - \left(\frac{29}{32}\right)^2 \times 2^{-2(k+5)} \\
&\leq 0.00002517580460
\end{aligned}$$

This leads to the bounds :

$$0.1784737832 \leq e^4 \leq 0.1784738953 \tag{4.17}$$

Computation of e^h

We compute e^h with the technic shown in the previous sections. The coefficient used in the bounds are listed below :

k	α	β	e_k^h
0			$2^{-h+1} - 2^{-2h}$
1	1	$1 - 2^{-h}$	$2^{-1-h} - 2^{-1-2h} - 2^{-4-2h}$
2	1	$1 - 2^{-h} - 2^{-(h+2)}$	$2^{-2-h} - 5 \times 2^{-4-2h} - 2^{-6-2h}$
3	1	$1 - 2^{-h} - 2^{-(h+2)} - 2^{-(h+3)}$	$2^{-3-h} - 11 \times 2^{-6-2h} - 2^{-8-2h}$
4	1	$1 - 2^{-h} - 2^{-(h+2)} - \cdots 2^{-(h+4)}$	$2^{-4-h} - 23 \times 2^{-8-2h} - 2^{-10-2h}$
5	1	$1 - 2^{-h} - 2^{-(h+2)} - \cdots 2^{-(h+5)}$	$2^{-5-h} - 47 \times 2^{-10-2h} - 2^{-12-2h}$
$k \geq 6$	≤ 1	$\leq 1 - 2^{-h} - 2^{-(h+2)} - \cdots 2^{-(h+6)}$	$\geq S_1$
$k \geq 6$	$\geq 1 - 2^{-h} - 2^{-(h+1)}$	$\geq 1 - 2^{-h} - 2^{-(h+1)}$	$\leq S_2$

S_1 and S_2 are the two sums for the minoration and majoration of all the terms for $k > 5$ (as before these sums are very small).

$$
\begin{aligned}
e_0^h + \cdots + e_5^h &= 95 \times 2^{-h-5} - 9025 \times 2^{-2h-12} \\
S_1 &= \sum_{k \geq 6} 2^{-(k+h)} - 95 \times 2^{-6-k-2h} - 2^{-2(k+h+1)} \\
&\leq 2^{-5} \left(2^{-h} - 95 \times 2^{-(2h+6)} - \frac{1}{3} 2^{-(2h+7)} \right) \\
S_2 &= \sum_{k \geq 6} \left(1 - 2^{-h} - 2^{-(h+1)} \right)^2 \left(2^{-(k+h)} - 2^{-2(k+h+1)} \right) \\
&\leq 2^{-5} \left(2^{-h} - \frac{1}{3} 2^{-(2h+7)} \right) \left(1 - 2^{-h} - 2^{-(h+1)} \right)^2
\end{aligned}
$$

$$
95 \times 2^{-h-5} - 9025 \times 2^{-2h-12} + S_1 \leq e^h \leq 95 \times 2^{-h-5} - 9025 \times 2^{-2h-12} + S_2
$$

4.3.1.7 Eccentricity of $0\cdots0$

By 4.10 we have :

$$\begin{aligned}
\bar{e}(0\cdots0) &= \frac{N}{N-1}\left(D-\sum_{h=1}^{D-1}e^h\right)\\
&= \frac{N}{N-1}\left(D-\left(e^0+e^1+e^2+e^3+e^4+\sum_{h=5}^{D-1}e^h\right)\right)\\
&\leq \frac{N}{N-1}\left(D-2.015018679+6\times\left(\frac{1}{2}\right)^D-\frac{13823}{4608}\left(\frac{1}{4}\right)^D-\frac{870913}{4718592}\right)\\
&\geq \frac{N}{N-1}\left(D-2.015016861+6\times\left(\frac{1}{2}\right)^D-\frac{7057}{2304}\left(\frac{1}{4}\right)^D+\right.\\
&\quad \left.\frac{289}{3584}\left(\frac{1}{8}\right)^D-\frac{1}{5120}\left(\frac{1}{16}\right)^D-\frac{62407017217}{338228674560}\right)
\end{aligned}$$

For large D we have :

$$\boxed{D-2.199528175\leq\bar{e}(0\cdots0)\leq D-2.199589203}$$

The mean eccentricity of $0\cdots0$ (and $1\cdots1$) is about $D-2.2$ in the undirected binary *de Bruijn* graph.

Exact values for small D

Binary de Bruijn digraph

D	$D(2,D)$		
	mean $\bar{e}(x)$	max weight	min weight
2	1.125	5	4
3	1.84375	17	13
4	2.65625	49	38
5	3.53516	129	103
6	4.46143	321	264
7	5.41748	769	649
8	6.39191	1793	1546
9	7.37726	4097	3595
10	8.369	9217	8204

Undirected binary de Bruijn graph

Eccentricity of $(a, a, \cdots, a)$

D	weigth	$\bar{e}(x)$
15	420966	12.8473
16	906818	13.8372
17	1943744	14.8297
18	4147974	15.8233
19	8817849	16.8187
20	18680199	17.8148
21	39451326	18.8119
22	83086866	19.8095
23	174546832	20.8076
24	365844958	21.8061
25	765205532	22.8049

Computations for all the vertices

D	$UD(2,D)$			
	mean $\bar{e}(x)$	max weight $(a,a,\cdots,a)$	min weight	vertices
2	1.166667	4	3	01 10
3	1.642857	14	10	001, 011 110, 100
4	2.141667	40	28	0011 1100
5	2.754032	107	79	00011, 00111 11100, 11000
6	3.453373	269	202	001011 110100
7	4.214813	656	504	0001011, 0010111 1110100, 1101000
8	5.028033	1548	1226	00010111 11101000
9	5.884433	3582	2907	001111010, 010111100 110000101, 101000011
10	6.773661	8142	6720	0010111100, 0011110100 1101000011, 1100001011
11	7.688553	18270	15339	00101111100, 00111110100 11010000011, 11000001011
12	8.623205	40524	34524	001101011100, 001110101100 110010100011, 110001010011
13	9.573310	89089	76807	0011010111100, 0011110101100 1100101000011, 1100001010011
14	10.535253	194293	169362	00101111110100 11010000001011
15	11,5063	420966	370619	001011111101100, 001101111110100 110100000010011, 110010000001011
16	12,4844	906818	805661	0010111111101100, 0011011111110100 1101000000010011, 1100100000001011
17	13,4678	1943744	1740497	00101101111110100, 00101111110110100 11010010000001011, 11010000001001011
18	14,4554	4147974	3739969	001011111010100011, 001110101000001011 110100000101011100, 110001010111110100

Maple functions and procedures

The following maple procedures were used to compute the values of e_k^h.

```
 % e1e2, eh, em0
#
#  e^1
#
e10 := 3/4;
e11 := 7/64;
e12 := 23*2**(-10);  # alpha = 1
e13 := 43*2**(-12);  # alpha = 1

#
# alpha^2i+1 = alpha^2i = beta^i
#
alpha := proc(k)
local res;
if k;4
then res := 1;
else res := beta(round((k-1)/2));
fi;
res;
end;

#
# beta^i+1 = beta^i - beta^"i/2/ 2^-(i+2)
#
beta := proc(k)
local res;
if k=1 then res := 1;
   else
       if k=2 then res := 3/4;
          else
              if
                  k ¿ 2 then
                  res := beta(k-1) - beta( round((k-2)/2))* 2^(-(k+1));
                  else res := 1;
              fi;
       fi;
fi;
res;
end;


e1k := proc(k) # avec nouveau alpha et beta
local res;

res := alpha(k) * beta(k) * 2^(-(k+3)) - (alpha(k))^2 * 2^(-(2*k+6));
res;
end;

e1 := proc(borne)
local res,somme,i,resteinf,restesup,infe1,infe2;

somme := e10 + e11 + e12 + e13 + e1k(4);

for i from 5 by 1 to 11
  do
     somme := somme + e1k(i);
  od;

resteinf := sum((5/8)^2*(2^(-(k+3)))-(327*2^(-9))^2*2^(-(2*k+6)),
k=12..infinity);
restesup :=
sum(327*2^(-9)*1328433*2^(-21)*2^(-(k+3))-(5/8)^2*2^(-(2*k+6)),
k=12..infinity);
print(resteinf*1.0, restesup*1.0);
infe1 := somme + resteinf;
supe1 := somme + restesup;

if borne = 0 then print(infe1,infe1*1.0); res := infe1; fi;
if borne = 1 then print(supe1,supe1*1.0); res := supe1; fi;
res;
end;

#
#  e^2
#

e2 := proc(borne)
local res,somme,i,resteinf,restesup,infe1,infe2;

infe2 := 4*(e1(0)-e10);
supe2 := 4*(e1(1)-e10);

if borne = 0 then print(infe2,infe2*1.0); res := infe2; fi;
```

```
if borne = 1 then print(supe2,supe2*1.0); res := supe2; fi;
res;
end;

##### Calculs des e^h

# Correction du calcul de alpha le 20/11/1992
# alpha 2i+1 = alpha 2i = beta i
alpha := proc(h,k)
local res;
if k ¡= h
then res := 1;
else res := beta(h,round((k-h)/2));
fi;
res;
end;

# Correction du calcul de beta le 20/11/1992
# beta i+1 = beta i - beta "i/2/ 2^-(i+2)
beta := proc(h,k)
local res;
if k=1 then res := 1-1/2^h;
   else
      if
        k ¿ 1 then
        res:=beta(h,k-1)-beta(h,round((k-h-1)/2))*2^(-(k+h));
         else res := 1;
      fi;
fi;
res;
end;

ehk := proc(h,k)
local res;

if k=0
then res := 2^(-h+1) - 2^(-2*h);
else

#calcul de alpha
        a := alpha(h,k);

#calcul de beta
        b := beta(h,k);

        res := a * b * 2^(-(k+h)) - a^2 * 2^(-2*(k+h+1));

fi;
res;
end;

tab := proc(h,P) # table des valeurs de 0 a P
local res,S,ei,i;
    S := 0;
    for i from 0 by 1 to P #D-h/2
    do
        ei := ehk(h,i);
        S  := S + ei;
        print(i,ei,ei*1.0,S,S*1.0);
    od;
res := S;
res;
end;

resteinf := proc(h,k,aa) # approximation de la somme de k a l'infini
local a, res,s;
     if (k¡(h+3)) then print('¡ k trop petit !!!¿') ;
     else
     a := 1- 1/(2**h) - 1/(2**(h+1));
     res := sum(a^2*2^(-(s+h))- aa^2*2^(-2*(s+h+1)),s=k..infinity);
     print(res * 1.0);
     fi;
res;
end;

restesup := proc(h,k,alpha,beta) # approximation de la somme de k a l'infini
local res,a;
     a := 1- 1/(2**h) - 1/(2**(h+1));
     res := sum(alpha*beta*2^(-(s+h))-(a)^2*2^(-2*(k+s+1)),s=k..infinity);
     print(res * 1.0);
res;
end;

eh := proc(h,P,borne)
local debut,ressup,resinf;

debut := tab(h,P);
ressup := debut + restesup(h,P+1,alpha(h,P+1),beta(h,P+1));
resinf := debut + resteinf(h,P+1,alpha(h,P+1));
```

```
print(resinf*1.0,h,ressup*1.0);
if borne=0 then res := resinf fi;
if borne=1 then res := ressup fi;
res;
end;


# S05 := e`0^h + ... + e`5^h
#
S05 := ¡95*2**(-h-5)-9025*2**(-2*h-12)—h¿;

S1 := 2^(-5)*(2^(-h)-95*2^(-(2*h+6))-2^(-(2*h+7))/3);
S2 := 2^(-5)*(2^(-h)-2^(-(2*h+7))/3)*(1-1/2**h-1/2**(h+1))^2;

ehs := ¡95*2**(-h-5)-9025*2**(-2*h-12)+2^(-5)*(2^(-h)-95*2^(-(2*h+6))-
2^(-(2*h+7))/3)—h¿;

ehi := ¡95*2**(-h-5)-9025*2**(-2*h-12)+2^(-5)*(2^(-h)-2^(-(2*h+7))/3)*
(1-1/2**h-1/2**(h+1))^2—h¿;


ehinf := sum(ehi(x),x=5..D-1);

ehsup := sum(ehs(x),x=5..D-1);

eccexp := proc(P,borne)
local res,i;
res := e1(borne) + e2(borne);

for i from 3 by 1 to P
do
  res := res + eh(i,2*P+3,borne);
od;

res*1.0;
end;

ecc := proc(borne)
local res;

end;
```

Bibliographie

[1] J-C. Bermond, C. Delorme, and J-J. Quisquater. Strategies for interconnection networks: Some methods from graph theory. *JPDC*, 3:433–449, 1986.

[2] J-C. Bermond, N. Homobono, and C. Peyrat. Large fault-tolerant interconnection networks. *Graph and Combinatorics*, 5, 1989.

[3] J-C. Bermond and C. Peyrat. de Bruijn and Kautz networks: a competitor for the hypercube? In J.P. Verjus F. André, editor, *Hypercube and Distributed Computers*, pages 279–294. North-Holland, 1989.

[4] Ioan Bond. *Constructions de grands réseaux d'interconnexion.* PhD thesis, Université de Paris-Sud, Centre d'Orsay, 1984.

[5] F. Buckley and F. Harary. *Distance in Graphs.* Addison Wesley, 1990.

[6] N. G. de Bruijn. A combinatorial problem. In *Koninklijke Nederlands Akademie van Wetenschappen Proceedings*, volume 49-2, pages 758–764, 1946.

[7] M. A. Fiol, J. L. A. Yebra, and I. Alegre de Miquel. Line digraph iterations and the (d,k) digraph problem. *IEEE Trans. on Computers*, C-33(5):400–403, 1984.

[8] S. W. Golomb. *Shift Register Sequences.* Holden-Day Inc, 1967.

[9] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures : Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1991.

[10] Claudine Peyrat. *Vulnérabilité dans les grand réseaux d'interconnexion.* PhD thesis, Université de Paris-Sud, Centre d'Orsay, 1984.

[11] M.R. Samatham and D.K. Pradhan. The de Bruijn multiprocessor network: a versatile parallel procesing and sorting network for VLSI. *IEEE Trans. on Comp.*, 38(4):567–581, 1989.

Chapitre 5

Conjonction des de Bruijn

Introduction

Les produits de graphes sont très utilisés pour construire de nouveaux réseaux à partir de structures existantes. En général on cherche à combiner les propriétés de deux graphes, ou bien on crée une nouvelle dimension dans le graphe, en faisant le produit du graphe par lui-même. Des exemples classiques du produit appelé somme cartésienne sont : la grille qui est égale à la somme de deux chemins, le tore qui est la somme de deux cycles et l'hypercube de dimension n qui est égal à K_2^n (K_2 étant le graphe complet à deux sommets). Notons que certains auteurs parlent de produit cartésien pour la même opération.

Les graphes de *de Bruijn* qui ont été définis dans le chapitre 4 ne sont pas égaux à une quelconque somme cartésienne. En revanche, nous allons montrer dans la suite qu'un autre type de produit, la conjonction, permet de combiner des *de Bruijn* pour en construire des plus grands.

5.1 Conjonction de deux graphes de *de Bruijn*

Définition 3 *La conjonction de $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ (aussi appelée produit cartésien dans [1] et produit de Kroneker dans [9]), notée $G_1 \wedge G_2$ est le graphe dont :*

- *l'ensemble des sommets est $V = V_1 \times V_2$*
- *un arc relie le sommet $(x_1, x_2) \in V$ au sommet $(y_1, y_2) \in V$, si et seulement si on vérifie : $(x_1, y_1) \in E_1$ et $(x_2, y_2) \in E_2$.*

Théorème 5.1.1

$$B(d_1, D) \wedge B(d_2, D) = B(d_1 \times d_2, D)$$

Preuve: soient V_1, V_2, les ensembles de sommets respectifs de $B(d_1, D)$, et $B(d_2, D)$. Soit $X = x_1 \cdots x_D \in V_1$ avec $x_i \in 0, 1, \cdots d_1 - 1$, et $Y = y_1 \cdots y_D \in V_2$

avec $y_i \in 0, 1, \cdots d_2 - 1$, alors le sommet (X, Y) appartient à $V_{1\wedge 2} = V_1 \times V_2$, l'ensemble de sommets de $B(d_1, D) \wedge B(d_2, D)$.

1. Le nombre de sommets dans $V_{1\wedge 2}$ est : $|V_1| \times |V_2| = d_1^D \times d_2^D = (d_1 \times d_2)^D$
2. On construit l'application p:

$$\begin{array}{ccc} p : V_1 \times V_2 & \longrightarrow & V_{1\wedge 2} \\ (x_1 \cdots x_D, y_1 \cdots y_D) & \longmapsto & (z_1 \cdots z_D) \end{array}$$

où $z_i = d_2 x_i + y_i$. Donc $0 \leq z_i \leq d_1 d_2 - 1$

Cette application nous permet d'écrire tout couple (X, Y) avec un mot construit sur un alphabet comportant $d_1 \times d_2$ lettres, montrons que c'est bien un isomorphisme de graphe.

Un arc de $B(d_1, D) \wedge B(d_2, D)$ est de la forme :

$$(x_1 \cdots x_D, y_1 \cdots y_D) \longmapsto (x_2 \cdots x_D \lambda_1, y_2 \cdots y_D \lambda_2)$$

$\lambda_1 \in 0, \cdots, d_1 - 1, \lambda_2 \in 0, \cdots, d_2 - 1$

En applicant p on obtient la relation d'adjacence :

$$z_1 \cdots z_D \longmapsto z_1 \cdots z_D \beta$$

où $\beta = d_1 \lambda_1 + \lambda_2$. Ceci est la définition d'un arc de $B(d_1 d_2, D)$!

□

5.1.1 Exemple

Le produit carré de $B(2,2)$, $B(2,2) \wedge B(2,2)$, donne le $B(4,2)$ dessiné sur la figure 5.1.

5.2 La transformée de Fourier discrète

La transformée de Fourier discrète d'un vecteur x de N points est une transformation linéaire de x : $X = F_N x$ où les éléments i, j de la matrice F_N sont ω_N^{ij}, où ω_N est une racine Nième de l'unité (souvent $\omega_N = e^{\frac{-2\pi i}{N}}$). Cette expression matricielle se traduit en un algorithme direct (le produit du vecteur par la matrice des coefficients), mais la structure particulière de la matrice a permis de mettre au point un algorithme récursif dit "rapide" (présenté ci-après) basé sur la factorisation de cette matrice. La transformée de Fourier rapide ou FFT pour Fast Fourier Transform est un algorithme essentiel en traitement du signal, en analyse d'images, et peut être utilisé (entre autres), pour le calcul de convolutions

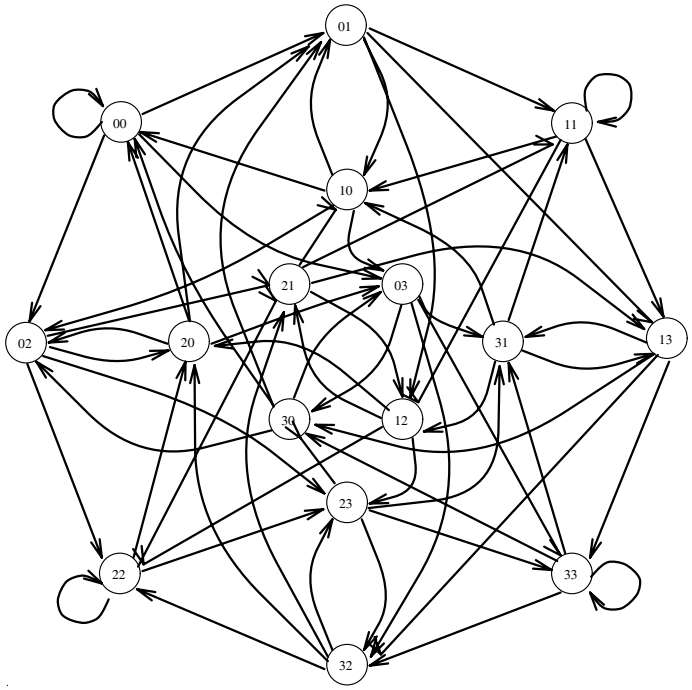

Figure 5.1: $B(4,2) = B(2,2) \wedge B(2,2)$.

ou de multiplications de polynômes. L'algorithme de la FFT permet de calculer la transformée discrète de N points en $\log N$ étapes sur un réseau butterfly de $N(\log N + 1)$ nœuds [11]. Ce réseau "papillon" modèlise le flot des données de l'algorithme.

De nombreux articles traitent le calcul en parallèle de la FFT sur des architectures en Hypercube [10], [4], [3] en arbre [8], ou diverses machines SIMD comme les DAP de chez ICL [2].

Ici, nous allons montrer comment les réseaux de *de Bruijn* se prêtent naturellement à ce calcul, et comment on peut exploiter le résultat sur le produit cartésien (conjonction) pour étendre simplement l'algorithme monodimensionnel (1-D) à l'algorithme bidimensionnel (2-D).

Dans la suite du chapitre on supposera que N est une puissance de 2.

5.2.1 Cas monodimensionnel

On calcule :

$$X(k) = \sum_{n=0}^{N-1} x(n)\omega^{nk}$$

Calcul de la FFT sur le de Bruijn $B(2,D)$ [12]

On calcule la FFT de $N = 2^D$ points. Chaque nœud u du $B(2,D)$ calcule l'élément $X(\mathrm{rev}(u))$, où $\mathrm{rev}(u)$ est le "bit-reverse" de u (par exemple $\mathrm{rev}(4) = 1$), et à l'étape initiale de l'algorithme $X(u)$ contient la valeur de $x(u), u = 0 \cdots N-1$.

Le calcul sur le $B(2,D)$ s'exécute en D étapes, et à chaque étape un nœud transmet en parallèle la valeur courante de $X(u)$ à ses deux successeurs dans le graphe, et effectue un calcul "papillon" des deux valeurs qu'il a reçu de ses prédecesseurs. Le calcul papillon consiste à multiplier une des entrées par une puissance de ω, et à l'ajouter à l'autre entrée.

Le nœud $a_1 \cdots a_D$ a pour prédecesseurs les nœuds $0a_1 \cdots a_{D-1}$ et $1a_1 \cdots a_{D-1}$. A l'étape k, $k = 1 \cdots D$, ce nœud calcule :

$$X_k(a_1 \cdots a_D) = X_{k-1}(0a_1 \cdots a_{D-1}) + \omega^{2^{D-k} \times a_{D-1} \cdots a_{D-k}} X_{k-1}(1a_1 \cdots a_{D-1}).$$

A la fin de la $D^{\text{ième}}$ étape, chaque nœud contient l'élément en position "bit reverse" de la FFT. On peut alors soit faire une permutation en D étapes supplèmentaires, soit transmettre le résultat à un autre circuit en réordonnant les valeurs, soit laisser les éléments à leur place dans le cas où le traitement suivant est local, et suivi d'une transformée inverse qui aura pour effet de les repositionner.

Le tableau suivant donne les résultats des calculs effectués à chaque étape sur les nœuds pour $N = 8$.

0=000	x_0	$x_0 + x_4$	$x_0 + x_2 + x_4 + x_6$	$x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$
1=001	x_1	$x_0 + \omega^4 x_4$	$x_0 + \omega^4 x_2 + x_4 + \omega^4 x_6$	$x_0 + \omega^4 x_1 + x_2 + \omega^4 x_3 + x_4 + \omega^4 x_5 + x_6 + \omega^4 x_7$
2=010	x_2	$x_1 + x_5$	$x_0 + \omega^2 x_2 + \omega^4 x_4 + \omega^6 x_6$	$x_0 + \omega^2 x_1 + \omega^4 x_2 + \omega^6 x_3 + x_4 + \omega^2 x_5 + \omega^4 x_6 + \omega^6 x_7$
3=011	x_3	$x_1 + \omega^4 x_5$	$x_0 + \omega^6 x_2 + \omega^4 x_4 + \omega^2 x_6$	$x_0 + \omega^6 x_1 + \omega^4 x_2 + \omega^2 x_3 + x_4 + \omega^6 x_5 + \omega^4 x_6 + \omega^2 x_7$
4=100	x_4	$x_2 + x_6$	$x_1 + x_3 + x_5 + x_7$	$x_0 + \omega x_1 + \omega^2 x_2 + \omega^3 x_3 + \omega^4 x_4 + \omega^5 x_5 + \omega^6 x_6 + \omega^7 x_7$
5=101	x_5	$x_2 + \omega^4 x_6$	$x_1 + \omega^4 x_3 + x_5 + \omega^4 x_7$	$x_0 + \omega^5 x_1 + \omega^2 x_2 + \omega^7 x_3 + \omega^4 x_4 + \omega x_5 + \omega^6 x_6 + \omega^3 x_7$
6=110	x_6	$x_3 + x_7$	$x_1 + \omega^2 x_3 + \omega^4 x_5 + \omega^6 x_7$	$x_0 + \omega^3 x_1 + \omega^6 x_2 + \omega^1 x_3 + \omega^4 x_4 + \omega^7 x_5 + \omega^2 x_6 + \omega^5 x_7$
7=111	x_7	$x_3 + \omega^4 x_7$	$x_1 + \omega^6 x_3 + \omega^4 x_5 + \omega^2 x_7$	$x_0 + \omega^7 x_1 + \omega^6 x_2 + \omega^5 x_3 + \omega^4 x_4 + \omega^3 x_5 + \omega^2 x_6 + \omega x_7$

Cet algorithme ne serait pas efficace sur une machine MIMD sans effectuer un partitionnement des données. Cependant, les tailles de *de Bruijn* ($D = 13$) effectivement réalisables en VLSI [5] confirment aujourd'hui l'adéquation de cet algorithme. On peut fabriquer un circuit dédié à ce calcul pour des tailles de N utiles en pratique.

5.2.2 Cas multidimensionnel

L'analyse bispectrale est un outil essentiel pour la reconstitution d'images en astrophysique [6]. L'algorithme proposé dans [6] pour la reconstruction d'un signal bidimensionnel utilise un schéma de communication aux voisins identique en tout point à celui de la FFT ou de son inverse (l'algorithme de l'inverse est identique, à une normalisation par $1/N$ près et à l'utilisation du conjugué de la racine de l'unité).

La transformée de Fourier bidimensionnelle calculée sur un ensemble de $N \times N$ points est définie par :

$$X(k_x, k_y) = \sum_{n_x=0}^{N-1} \sum_{n_y=0}^{N-1} x(n_x, n_y) \omega^{n_x k_x + n_y k_y}$$

L'algorithme classique se déroule en deux phases :

1. calcul de la transformée des lignes de la matrice $N \times N$ du signal échantilloné.
2. calcul de la transformée des colonnes de la matrice résultat.

dans les deux étapes le calcul utilise la FFT 1–D. Cependant, on peut mélanger les deux étapes, comme cela est expliqué dans [7, sect. 2.4].

Ici on va effectuer le calcul comme pour le cas monodimensionnel, mais sur un *de Bruijn* $B(4, D)$. A chaque étape un nœud transmet sa valeur courante à ses 4 successeurs (au lieu de 2), et calcule son papillon 4×4 au lieu de 2×2.

Un nœud qui reçoit les 4 valeurs est noté dans sa forme produit (u, v), où u et v sont la représentation binaire de chacun des sommets du $B(2, D)$ dont le $B(4, D)$ est le produit défini dans la première section. La somme effectuée est alors :

$$\begin{aligned}
X_k(a_1 \cdots a_D, b_1 \cdots b_D) = & \\
& X_{k-1}(0a_1 \cdots a_{D-1}, 0b_1 \cdots b_{D-1}) \\
+ \quad & \omega^{2^{D-k} \times a_{D-1} \cdots a_{D-k}} X_{k-1}(1a_1 \cdots a_{D-1}, 0b_1 \cdots b_{D-1}) \\
+ \quad & \omega^{2^{D-k} \times b_{D-1} \cdots b_{D-k}} X_{k-1}(0a_1 \cdots a_{D-1}, 1b_1 \cdots b_{D-1}) \\
+ \quad & \omega^{2^{D-k} \times a_{D-1} \cdots a_{D-k} + 2^{D-k} \times b_{D-1} \cdots b_{D-k}} X_{k-1}(1a_1 \cdots a_{D-1}, 1b_1 \cdots b_{D-1}).
\end{aligned}$$

Conclusion

Ainsi la structure des graphes de *de Bruijn* permet d'exprimer directement l'algorithme de la FFT, et on peut calculer identiquement les FFT 1-D, 2-D ou n-D selon les besoins, en décomposant le graphe en produit conjonctif.

Je tiens à remercier ici Joël Leroux pour ses précieuses explications sur la FFT, et aussi Johny Bond pour son aide dans la preuve de l'isomorphisme.

Bibliographie

[1] J-C. Bermond. Hamiltonian decomposition of graphs, directed graphs and hypergraphs. *Annals of Discrete Mathematics*, 3:21–28, 1978.

[2] A. Brass and G.S. Pawley. Two and three dimensional FFTs on highly parallel computers. *Parallel Computing*, 3:167–184, 1986.

[3] R.M. Chamberlain. Gray codes, fast fourier transform and hypercubes, 1988.

[4] C.Y. Chu. Comparison of two-dimensional FFT methods on hypercubes : the choice between strips and patches, 1987.

[5] O. Collins, R. MC Eliece, S. Dolinar, and F. Pollara. A VLSI decomposition of the de Bruijn graphs). *JACM*, Oct-1992, 1992.

[6] C. Coroyer, J. Leroux, and D. Rossille. Remarks on the bispectral analysis and reconstruction of mono and bidimensional sampled signals. *Multidimensional Systems and Signal Processing*, 1991.

[7] D.E. Dudgeon and R.M. Mersereau. *Multidimensional digital signal processing.* Prentice Hall, 1984.

[8] A.L. Gorin, A. Silberger, and L. Auslander. Computing the two-dimensional discrete fourier transform on the ASPEN parallel computer architecture. pages 921–923, 1988.

[9] F. Harary. *Graph Theory.* Addison-Wesley, Reading, Mass, 1969.

[10] S.L. Johnsson, C.T. Ho, M. Jacquemin, and A. Ruttenberg. Computing fast fourier transforms on boolean cubes and related networks, 1987.

[11] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures : Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1991.

[12] M.R. Samatham and D.K. Pradhan. A multiprocessor network suitable for single-chip VLSI implementation. In *Procs. of the 11th International Symposium on Computer Architecture*, pages 328–337. IEEE, 1984.

Conclusion

Communications dans les architectures à mémoire distribuée

Les divers points étudiés dans cette thèse sont complémentaires dans la phase de conception d'un système parallèle.

Le premier chapitre permet de faire le point sur le marché des machines parallèles, et repose sur une expérience assez grande des transputers (cartes Inmos B012 et B014, T-node, Mega-node et Meiko CS), mais aussi sur des tests effectués sur les machines intel iPSC/2, TMC CM-2, CM-200 et Maspar MP-1.

Les chapitres 2 et 3 s'intéressent aux routines de bas niveau à implanter sur ces machines pour accélérer et garantir les échanges de données entre les divers nœuds du réseau. La pauvreté des premiers environnements de programmation (due en grande partie à l'originalité et à la complexité de ces systèmes) était telle que les programmeurs ont du résoudre en priorité ces problèmes de communication avant d'exploiter la puissance de calcul des processeurs. Des protocoles de communications structurées ainsi que la prévention de l'interblocage doivent être intégrés physiquement dans un même souci de performances. Dans cette thèse on propose une solutions originale pour le mode de commutation du type "circuit", sur les architectures courantes que sont les tores.

La topologie du réseau détermine les performances des communications, et donc la puissance des programmes qui vont être exécutés sur ces machines. Ainsi, tout au long de cette thèse nous avons mis en avant le réseau de *de Bruijn* .

Le résultat principal pour ce graphe porte sur l'excentricité moyenne d'un sommet, et on montre que cette mesure de la distance moyenne d'un sommet à tous les autres est proche du diamètre. Cela justifie l'usage d'un routage de type glouton dans ce réseau, et ce sans perte de performances. De plus on donne un résultat sur le produit des graphes de cette famille, en montrant leur adéquation au calcul des transformées de Fourier multidimensionnelles.

Les perspectives de ce travail sont multiples :

- donner une implantation réelle des réseaux de *de Bruijn*, pour la construction d'une machine MIMD.

- généraliser les techniques de diffusion en mode circuit-switched à toutes les

topologies courantes.

- étudier les possibilité du mode circuit-switched pour d'autres schémas de communication.

Le nombre toujours croissant de processeurs dans les nouvelles machines parallèles renforce l'importance des études de ce type. Une partie du travail présenté dans cette thèse est inclus dans un ouvrage complet à paraitre sur ce thème des communications, qui fait suite à l'école d'été RUMEUR de Cargèse en août 1992.

Abstract

This thesis deals with several problems related to communication networks in multicomputers architectures.

The first chapter provides a survey of the technics of communication used in the current and future parallel computers. This study points out the critical choice of the topology of the interconnection network and introduces several models of communications. The performance of a message-passing system is strongly dependent on the topology of the interconnection network and on the routing mechanism that is used to move information around the network.

Store-and-forward routing has been displaced by *circuit-switched* routing in many recent multicomputer systems. The second chapter introduces a routing model known as wormhole routing, and is concerned with deadlock avoidance. We give routing functions for any graph that use a minimum number of virtual channels.

In chapter 3 we describe an optimal broadcasting algorithm for 2-dimensional torus networks (wrap-around meshes) that uses synchronous circuit-switched routing. The algorithm is based on a recursive tiling of the torus. The algorithm requires $log_5(n^2)$ phases and $n-1$ intermediate switch settings to broadcast in an $n \times n$ torus. Both of these quantities match the lower bounds.

Chapter 4 deals with the topology of interconnection networks. Given a graph $G = (V, E)$ we define $\bar{e}(x)$, the mean eccentricity of a vertex x, as the average distance from x to all the others vertices of the graph. Let $N = |V|$ and $d(x, y)$ the distance from x to y, $\bar{e}(x) = \frac{1}{N-1} \sum_{y \in V - \{x\}} d(x, y)$. The computation of this parameter appears to be difficult in the case of the *de Bruijn* networks. In this chapter we give a lower bound and an upper bound for $\bar{e}(x)$. In the case of a *de Bruijn* digraph the given bounds are sharp and we exhibit the extremal vertices, whereas, in the undirected case, the computation is more difficult, even for degree 4. However we give a way to compute the value of $\bar{e}(x)$ at least for D sufficiently large and give a precise estimate for $x = 0 \cdots 0$. We also give some conjectures, induced by the computations done for graphs up to diameter 25.

The last chapter propose a generic and efficient implementation of the Fast Fourier Transform algorithm, a fundamental tool in image analysis or astrophysics. The algorithm is based on the following result : the conjunction of two *de Bruijn* graphs is a *de Bruijn* graph.

Keywords: interconnection networks - parallel computers - communications - broadcasting - wormhole routing - deadlock - eccentricity - de Bruijn

Résumé

Cette thèse traite plusieurs aspects complémentaires dans les machines parallèles à mémoire distribuée.

Un premier chapitre établit un état de l'art des techniques de communications mise en œuvre dans les machines existantes ou en projet. Cette étude permet d'introduire différents modèles de communication et pose aussi le problème du choix du réseau d'interconnexion pour assembler les processeurs de ces calculateurs massivement parallèles.

Les modes de communication des nouvelles machines parallèles sont du type commutation de circuit. Le routage *wormhole* est de ce type et remplace désormais le routage de type *store-and-forward*. Le deuxième chapitre concerne l'utilisation efficace des *canaux virtuels* en wormhole pour éviter les interblocages. On montre comment utiliser un nombre minimum de *canaux virtuels* pour des réseaux quelconques et pour des familles courantes de réseaux d'interconnexion.

Dans le troisième chapitre nous décrivons un algorithme de diffusion optimal pour les tores bidimensionnels qui utilise un routage synchrone de type commutation de circuit. L'algorithme repose sur un découpage récursif du tore. La diffusion dans un tore $n \times n$ est alors effectuée en $log_5(n^2)$ étapes en traversant un total de $n-1$ nœuds intermédiaires. Ces deux quantités sont des bornes inférieures pour la diffusion.

Le quatrième chapitre traite de la topologie des réseaux d'interconnexion. Etant donné un graphe $G = (V, E)$, l'excentricité moyenne d'un sommet x notée $\bar{e}(x)$ est définie comme la distance moyenne de x à tous les autres sommets de G. Soit $N = |V|$ et soit $d(x, y)$ la distance de x à y, alors $\bar{e}(x) = \frac{1}{N-1} \sum_{y \in V-\{x\}} d(x, y)$. Le calcul de ce paramètre est difficile dans le cas des graphes de *de Bruijn* $B(d, D)$. Dans ce chapitre nous donnons des bornes inférieures et supérieures de $\bar{e}(x)$ pour ces graphes. Dans le cas du *de Bruijn* orienté, les bornes sont exactes et on connait les sommets qui atteignent les bornes. Pour le cas du graphe non orienté, le calcul est plus difficile, même pour le *de Bruijn* binaire. Cependant nous donnons une méthode qui permet de déterminer $\bar{e}(x)$, quand D est assez grand, et un encadrement précis de l'excentricité du sommet $x = 0 \cdots 0$.

Le chapitre suivant donne une implantation efficace et générique d'un algorithme fondamental en calcul numérique (Transformée de Fourier Rapide) sur les réseaux de *de Bruijn*. On utilise le résultat suivant sur la conjonction de ces réseaux : le produit cartésien de deux *de Bruijn* est un *de Bruijn*.

Mots clés : réseaux d'interconnexion - architectures parallèles - communications - diffusion - routage wormhole - interblocages - excentricité - de Bruijn