Résumé

Dans les machines parallèles, les processus qui s'exécutent sur des processeurs qui ne sont pas directement connectés communiquent par des envois de messages. Ces messages sont alors acheminés, ou routés, jusqu'à leur destination à travers les canaux qui relient les processeurs élémentaires (PE) du réseau.

Dans le routage *wormhole*, les messages progressent dans le réseau de processeurs flit par flit (1 flit est égal à la taille de la queue, ou tampon, associé au canal). La tête du message avance d'un canal chaque fois que cela est possible, libérant ainsi le dernier canal en queue de message, alors à nouveau disponible pour un autre message.

Le problème de l'*interblocage* dans un réseau de ce type peut être caracterisé par la présence de circuits dans le graphe de dépendance des canaux [3]. La technique utilisée pour donner des fonctions de routage sans interblocages utilise les notions de *canaux ou de réseaux virtuels* [3] et [8]. Le nombre de canaux virtuels utilisés pour réaliser une fonction non bloquante est critique tant du point de vue implantation VLSI que de celui des performances. Ce nombre doit donc être minimisé.

Nous donnons ici une synthèse des résultats obtenus ainsi qu'une modélisation plus rigoureuse du problème. Nous montrons ensuite que dans tout réseau on peut éviter l'interblocage, soit avec un routage des plus courts chemins et avec D canaux virtuels (où D est le diamètre du réseau), soit avec seulement deux canaux virtuels mais avec un routage plus long (au plus 2D) et un goulot d'étranglement. Nous montrons comment appliquer cette idée aux réseaux de de Bruijn (qui sont parmi les plus performants connus) et comment obtenir pour ces réseaux un routage équilibré sans interblocage avec seulement deux canaux virtuels.

1 Routage wormhole

Nous nous placons dans le contexte des architectures parallèles à mémoire distribuée où les communications se font par envoi de messages (voir chapitre de P. Fraigniaud dans cet ouvrage).

Les machines à mémoire distribuée les plus récentes ont abandonné le routage dit *store-and-forward* pour adopter une technique de type commutation de circuit communément appellée *wormhole*. Ce nom est admis même si l'origine n'est pas claire; fait-on référence à un vers ou bien aux "cosmic wormholes" des astrophysiciens ? [11, Seitz, page 37].

Dans le routage *store-and-forward* les messages (ou paquets) sont entièrement stockés dans un processeur avant d'être transmis au processeur suivant. Dans le routage *wormhole* , les messages progressent dans le réseau de processeurs flit par flit (1 flit 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 le dernier canal en queue de message, alors à nouveau disponible pour un autre 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 pour pouvoir être stocké sur le circuit virtuel, la source est liberée 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 ne sera pas passé ("on ne peut pas couper le vers"). 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 PE permet de négliger les erreurs de transmission. Les flits sont envoyés sans attendre d'accusé de reception, ils passent directement du canal de sortie d'un PE au canal en entrée du PE suivant : il n'y a pas le stockage intermédiaire du message qui est coûteux en temps [11]. Seul le flit routé ayant besoin d'être stocké, la fonction de routage peut être assurée sans accéder à la mémoire du processeur de calcul. On remarque que dans ce modèle les tampons sont de longueur réduite, ce qui facilite l'implantation VLSI du routeur.

Sur les machines courantes comme les iPSC/2 d'Intel [9], un PE est constitué d'un module comprenant le processeur de calcul (de conception identique à ceux des stations de travail), de la mémoire externe, et un routeur réalisé en technologie CMOS qui implante la gestion des canaux d'entrée sortie et la fonction de routage.

On trouve une première étude d'une technique semblable dans [6], c'est le *virtual cut through*. Le routage est identique en tout point au wormhole sauf quand un noeud ne peut pas faire suivre les flits (les canaux de sortie sont déjà utilisés). Dans ce cas le message est stocké en attendant la libération du canal de sortie. Le routeur doit donc disposer de buffers importants, ce qui contredit les hypothèses énoncées plus haut.

2 Coût du routage

Soit un message M de longueur L (exprimée en octets par exemple) à transmettre à un processeur à distance d. Nous utilisons le modèle couramment admis pour les machines à mémoire distribuée d'un "temps linéaire". On note par β le temps requis pour l'initialisation du processus de communication ("start-up time") et par τ le temps de propagation d'un octet sur le lien ($1/\tau$ représente la bande passante du lien)

Proposition 2.1 *Le délai de transmission d'un message de taille L à un processeur à distance d est en mode wormhole :*

$$T_w = d\beta + L\tau$$

Preuve : en effet en routage wormhole il faut un temps $d\beta$ pour établir le chemin, et ensuite il ne reste plus qu'à recevoir tout le message en un temps $L\tau$.

On ne doit pas confondre le temps T_w avec le temps obtenu par l'utilisation d'une technique *pipeline* en store-and-forward . Si le message était découpé en P paquets, le temps serait alors de $(d + P - 1)(\beta + \frac{L}{P}\tau)$.

On voit l'avantage du routage wormhole sur le modèle store-and-forward pour de longs messages.

3 Fonction de routage

Les hypothéses que nous prenons sur le routage dans ce chapitre sont celles de Dally et Seitz [3] :

- absence de pannes sur les liens et sur les noeuds
- routage déterministe et distribué, c'est-à-dire que le chemin emprunté par le message est determiné localement par le routeur du processeur courant, en utilisant seulement l'adresse du destinataire (et le canal d'entrée).

Le cas du routage adaptatif a été consideré par Duato [4].

Rappellons aussi la propriété importante qu'une fois que le flit de tête est accepté sur un lien, tout le reste du message doit être transmis avant d'accepter les flits d'autres messages, ceci se fait via un tampon, ou queue, de la taille d'un flit.

On modélise le réseau d'interconnexion par un graphe $G = (V \cup V', E = C \cup I \cup O)$. où l'ensemble $V \cup V'$ représente les processeurs élémentaires (PE). Plus exactement, V modélise les routeurs et V' isomorphe à V les mémoires. C représente les canaux de communication entre les routeurs. De plus chaque sommet v est relié à son homologue v' par un canal d'entrée i_v de v' vers $v \in I$ (inputs), et par un canal de sortie o_v de v vers $v' \in O$ (outputs). Ces canaux modèlisent les échanges entre le routeur et la mémoire du processeur. Bien souvent, pour ne pas alourdir la présentation, on utilise simplement le graphe $G = (V, C)$ des communications. (Voir figure 1 pour la représentation d'un anneau de 4 processeurs et figure 3 pour la représentation simplifiée d'un réseau analogue de 4 processeurs avec plus de canaux).

La fonction de routage est alors définie :

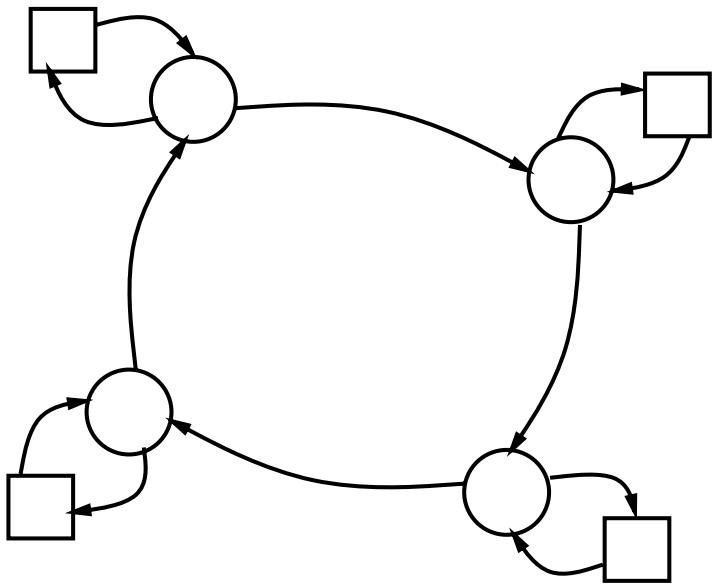

Figure 1: $G = (V \cup V', E = C \cup I \cup O)$.

Définition 1 *R fonction de routage sur G*

$$\begin{array}{rcl} R : E \times V & \longrightarrow & E \\ (e, p) & \longmapsto & f \end{array}$$

Etant donné un canal d'entrée $e = (u, v)$ (au sommet courant v) et une destination p, R associe un canal de sortie $f = (v, w)$ (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 p sur le canal o_p.

Dans l'exemple de la figure 1, soit un message émis par 0 et à destination de 3. Il entre en 0 par le canal i_0 et 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 que certains auteurs (voir par exemple le chapitre de P. Fraigniaud) utilisent une fonction de routage de $V \times V \longrightarrow V$, mais une telle fonction est insuffisante pour modéliser les solutions apportées à l'interblocage. Notons aussi que la fonction de routage est bien distribuée en ce sens que la route suivi par le message avant son arrivée sur le canal e n'a aucune influence sur le choix du canal f. Dans un routage adaptatif on aurait le choix de plusieurs canaux de sortie possibles.

4 Interblocages, Graphe de dépendance et canaux virtuels

4.1 Interblocage et graphe de dépendance

L'exemple classique [3] de l'interblocage est le suivant : sur un anneau unidirectionnel, tous les processeurs veulent émettre un message au même instant, et à une distance d'au moins 2. Un interblocage se produit alors car les messages

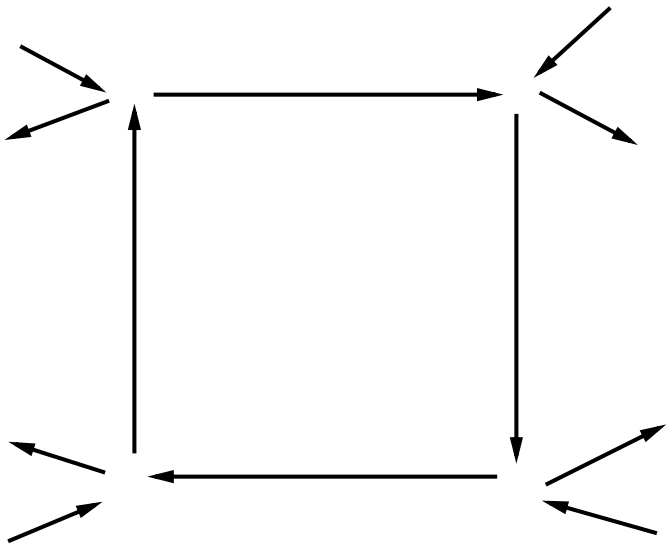

Figure 2: $\mathcal{D}(G)$ pour l'anneau de 4 processeurs

ne peuvent plus avancer, les queues des canaux qu'ils voudraient emprunter sont remplies par des flits bloqués. Il y a une dépendance cyclique entre les queues des canaux dans cette instance de routage.

Pour observer les propriétés de la fonction de routage f, on définit un graphe de dépendance des canaux $\mathcal{D}(G)$. Les sommets de $\mathcal{D}(G)$ sont les canaux de G, et il y a un arc entre deux sommets de ce graphe si les canaux correspondants de G peuvent être empruntés successivement dans un routage de G.

Théorème 4.1 [3] *Une fonction de routage R est non bloquante pour un réseau d'interconnexion G, si et seulement si le graphe de dépendance $\mathcal{D}(G)$ des canaux associé à R ne comporte pas de circuits.*

En fait, comme on peut le remarquer, les canaux d'entrée et de sortie n'interviennent pas dans les circuits du graphe de dépendance. On peut donc définir un graphe de dépendance simplifié (associé au graphe simplifié $G = (V, C)$) ayant comme sommets les seuls canaux de C.

4.2 Utilisation de canaux virtuels

Un canal physique ne peut stocker qu'un flit à la fois, si ce flit appartient à un message dont la progression est stoppée plus en avant, aucun autre message ne peut l'utiliser et ceci entraîne le plus souvent le bloquage d'un grand nombre d'autres canaux, 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 buffers ou canaux virtuels qui permettrons de partager le lien physique en autant de liens virtuels.

Pour décongestionner notre circuit de l'exemple 1, il suffit de diviser chaque lien de communication de C en 2 canaux virtuels numérotés (i, c), avec $i = 0, 1$ et

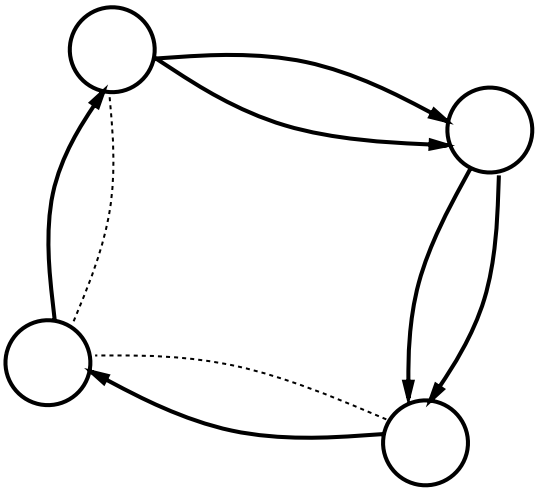

Figure 3: Canaux virtuels sur l'anneau de 4 processeurs

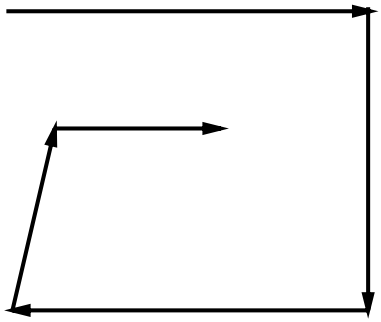

Figure 4: $\mathcal{D}(G)$

d'adopter la fonction de routage suivante. En un sommet courant on choisit le canal de sortie ayant le plus petit indice possible supérieur au canal d'entrée (la relation d'ordre est celle 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 $\mathcal{D}(G)$ est sans circuits (voir figure 4).

La technique générale mise en oeuvre dans [3] 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.

Un graphe est sans circuits si et seulement si il existe un ordre total de ses sommets de sorte que il existe un arc de i vers j seulement si $i < j$. On choisira donc 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 l'exemple de l'anneau de la figure 3.

5 Fonctions de routage des réseaux classiques

Pour les définitions concernant ces réseaux voir [7].

5.1 Hypercube

Si on considère l'algorithme de routage qui consiste à changer de dimensions du cube dans un ordre défini à l'avance, alors cet algorithme est sans interblocage [3]. En général on utilise l'ordre naturel des dimensions, mais toutes les permutations sont valides. Si on autorise plus de canaux virtuels, alors on peut choisir un grand nombre de chemins dans le cadre d'un routage adaptatif.

5.2 L'anneau uni-directionnel

Il suffit de prendre 2 canaux virtuels comme dans l'exemple 3 (cf Dally et Seitz [3]).

5.3 Grille torique et somme cartésienne

La grille torique est la somme cartésienne (aussi appelée produit cartésien) de 2 cycles. Il suffit de remplacer chaque canal par 2 canaux virtuels et d'étiqueter d'abord les canaux dans une dimension, puis ceux de l'autre dimension. D'une manière générale, dans une somme cartésienne, le nombre de canaux virtuels à utiliser est le maximum du nombre de canaux dans chaque dimension (voir [5]). Ceci s'applique donc aussi aux grilles d-dimensionelles ou aux "k-ary-n-cubes" [3].

5.4 Cube-connected-cycle

Dally et Seitz [3] ont montré que 3 canaux virtuels suffisaient.

5.5 Graphes généraux

Si on a une notion de dimension dans G, 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, on a les deux propositions suivantes.

Proposition 5.1 *Dans tout réseau de diamètre D on peut trouver une fonction de routage des plus courts chemins telle que, si on utilise D canaux virtuels, le graphe de dépendance des canaux associé 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 ième canal virtuel si 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 $(i+1, f)$, où f est 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 $(1, f)$, où f est le premier arc sur le plus court chemin de s à p. $\diamond$

En fait, si on n'impose pas un routage des plus courts chemins et si on ne tient pas compte de la charge des noeuds et canaux, on peut toujours éviter l'interblocage dans un réseau en utilisant seulement 2 canaux virtuels.

Proposition 5.2 *Il existe un routage de G (où tout chemin est de longueur au plus $2D$) tel que, si on remplace chaque canal par au plus 2 canaux virtuels, le graphe de dépendance associé soit sans circuit.*

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. Donc 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"). Donc tout chemin du routage est de longueur au plus $2D$. $\diamond$

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.

6 Routage sans interblocages dans les de Bruijn

Les réseaux de de Bruijn (ou leur généralisation) apparaissent comme d'excellents candidats pour les futures architectures parallèles [1] et [10], car ils permettent d'interconnecter un grand nombre de processeurs avec un petit diamètre et un degré fixé (4 liens pour les transputers T800 d'Inmos ou pour leur successeurs T9000, 6 liens pour les TMSC40 de Texas Instrument).

6.1 Définition des réseaux de de Bruijn

Le graphe orienté $B(d, D)$ a pour sommets les mots de longueur D sur un alphabet de d symboles. Il y a un arc d'origine le sommet $(a_1, \cdots, a_D)$ vers le sommet $(a_2, \cdots, a_D, \alpha)$ pour tout α pris dans l' alphabet. Les successeurs sont donc obtenus par des décalages à gauche. Ce graphe a un degré entrant et sortant de d, d^D sommets et un diamètre de D. Il contient d boucles, à savoir les sommets $(a, a, \cdots, a)$. De plus les arcs de ce graphe peuvent être partitionnés en d arborescences (et aussi d antiarborescences) deux à deux disjointes de racines respectives ces boucles.

L' arborescence T_a, de racine a, a pour arcs les canaux allant d'un sommet de niveau i (dont l'écriture commence par $(D-i)$ a, la lettre suivante étant différente

de a), vers un sommet de niveau $i+1$, dont l'écriture commence par $(D-i-1)\,a$. 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.

On définit $UB(d,D)$ comme le graphe non orienté de de Bruijn en oubliant l'orientation de $B(d,D)$, en supprimant les boucles mais pas les arêtes multiples.

6.2 Routage sans interblocage

On applique l'idée de la proposition 2, mais si le sommet de destination p est de la forme $(a,\cdots)$, 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 (cf [7, section 3.3]).

Remerciements

Ce travail a été effectué grâce au soutien financier du GDR-PRC C^3. Il a bénéficié de l'aide précieuse des membres du groupe RUMEUR de C^3, en particulier Pierre Fraigniaud et, pour le premier auteur, du record de pluviosité de SFU en janvier 1992.

Références

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

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

[3] 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.

[4] J. Duato. On the design of deadlock-free adaptative routing algorithms for multicomputers : design methodologies. In Springer-Verlag, editor, *PARLE 91*, pages 390–405, 1991.

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

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

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

[8] 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.

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

[10] 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.

[11] R. Suaya and G. Birtwist. *VLSI and Parallel Computation*. Morgan Kaufmann, 1990. ISBN 0-201-16802-2.