Chapitre 3

Réseaux d'interconnexion

3.1 Introduction et définitions

Un ordinateur parallèle se compose de processeurs, de mémoires et d'un réseau d'interconnexion reliant ces éléments entre eux. Ce réseau permet la communication (l'échange de données intermédiaires ou d'informations) entre les différentes unités de traitement et joue, pour cette raison, un rôle primordial dans l'architecture des machines parallèles. Un bon réseau doit, à un prix raisonnable et avec fiabilité, être capable de transmettre assez rapidement les données là où elles sont requises.

Dans de nombreux problèmes, la phase de communication peut être aussi importante que la phase de calcul proprement dite. Un mauvais réseau de communication peut être responsable d'une dégradation importante des performances, pouvant conduire l'utilisateur à se tourner vers une autre machine. Dans une machine parallèle, les processeurs sont fortement couplés et on s'attend à des connexions inter-processeurs à fort débit (de l'ordre du GigaBytes/s entre pairs de processeurs) et faible latence (de l'ordre de la micro-seconde).

3.1.1 Topologie et propriétés des réseaux d'interconnexion

L'idéal serait évidemment d'avoir un système de connexions qui relie chaque processeur avec tous les autres mais cela n'est possible que pour des machines avec peu de processeurs. Le prix et les problèmes technologiques qui en résulteraient rendent cette approche irréalisable pour une architecture qu'on souhaite massivement parallèle.

En pratique, les processeurs sont reliés entre eux selon une topologie déterminée. On verra de nombreux exemples concrets tout au long de ce chapitre. La **topologie** du réseau définit la structure des liens entre les processeurs. Certains processeurs sont adjacents s'il existe une connexion directe qui les relie. D'autres sont éloignés s'il faut passer par plusieurs processeurs intermédiaires pour trouver un chemin qui les relie. La figure 3.1 donne un exemple d'une topologie de

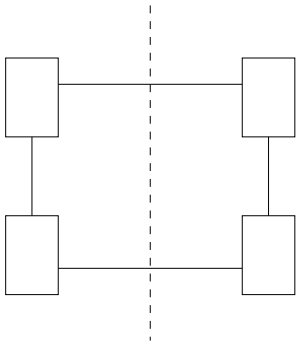


FIGURE 3.1 – *Exemple de réseau d'interconnexion entre 4 processeurs. La ligne pointillée divise le réseau en deux sous systèmes identiques, ce qui donne une largeur bissectionnelle de 2.*

connexion avec 4 processeurs.

On peut comparer un réseau d'interconnexion à un graphe : les **noeuds** du graphe sont les processeurs ou des modules mémoires ; les arcs sont les câbles de connexion qui peuvent être bidirectionnels ou ne permettre des échanges que dans un sens à la fois (semi-duplex). La théorie des graphes permet d'ailleurs de formaliser plusieurs problèmes importants liés à la topologie des réseaux : comment trouver les chemins de longueur minimale entre deux noeuds, chercher un chemin qui relie une et une seule fois chaque noeud, etc.

Ces questions sont liées à la nécessité de faire transiter le plus efficacement possible les messages entre divers noeuds. Cette opération importante s'appelle le **routage**. Elle dépend évidemment de la topologie du réseau considéré. On distingue deux types de topologies dans les machines parallèles :

— Les topologies statiques dans lesquelles les connexions physiques entre les noeuds du réseau sont fixes et ne changent plus après la construction de la machine. Dans ce cas, un noeud est typiquement constitué d'un processeur et d'un routeur.

— Les topologies dynamiques qui permettent une reconfiguration des connexions entre les noeuds du réseaux. On peut ainsi établir des chemins directs entre certaines paires de noeuds au détriment d'autres. Cette reconfiguration peut se faire totalement dynamiquement, en cours d'exécution d'un programme ou, plus simplement, être spécifiée avant exécution par un fichier de configuration. Dans ce cas, les noeuds internes de la topologie sont des switches et les processeurs sont les noeuds externes.

Les performances de chaque réseau sont déterminées par un ensemble de critères et de propriétés. Nous décrivons ci-dessous les principales. Il faut remarquer que certaines de ces propriétés sont plus ou moins significatives selon qu'il s'agit de topologies statiques ou dynamiques.

Le diamètre D d'un réseau est la longueur du chemin minimal (en nombre de noeuds à traverser) qui relie les deux noeuds les plus éloignés.

La connectivité qui est le nombre de liens de chaque noeud, ou encore le nombre de voisins qui lui sont directement accessibles. Ce nombre s'appelle aussi le **degré** de chaque noeud. En anglais on parle parfois du «radix» d'un noeud pour indiquer le nombre de liens dont il dispose.

La latence qui est le temps (exprimé en micro-seconde, typiquement) de transit maximum d'un signal à travers le réseau. C'est aussi une mesure de la distance qui relie les deux points les plus éloignés du réseau. Cependant, à la différence du diamètre, c'est une grandeur qui ne dépend pas uniquement de la topologie mais aussi de la technologie utilisée : longueur et nature des connexions, rapidité des circuits, etc. De plus, la latence inclus souvent aussi le temps t_s dit de *startup* qui est le temps de préparation d'un message : création de l'entête, de la fin de message, des codes correcteurs d'erreurs (ECC), ainsi que le temps d'activation du dispositif d'envoi. Le temps t_s n'intervient qu'une fois pour chaque message. L'ordre de grandeur du temps de latence dans une architecture parallèle est de plusieurs nanosecondes pour la partie hardware du réseau et quelques μsec en incluant l'overhead du logiciel. La figure 3.2 compare ce temps de latence (noté "remote memory" pour indiquer qu'on accède à des données hors du processeur) aux autres temps de latence caractéristiques d'un ordinateur.

La bandwidth qui mesure le débit d'information (généralement en MByte/s) qu'un noeud peut transmettre dans le réseau. La bandwidth dépend de la nature physique des liens existants, de leur nombre (largeur des canaux) et de la vitesse d'horloge du réseau. Une valeur de 500 Mbytes/s par connexion, est considéré comme une performance moyenne.

La largeur bissectionnelle qui est une mesure topologique globale du débit d'information qui peut circuler dans un réseau d'interconnexion. Pour la déterminer, on divise le réseau en deux moitiés identiques et on compte le nombre de canaux ou de connexions qui vont d'une moitié vers l'autre. Cela donne une information réaliste sur le volume d'informations qui peut être échangé dans le réseau. La figure 3.1 illustre cette construction. Il y a parfois plusieurs manières de diviser un réseau en deux moitiés égales. Dans ce cas la largeur bissectionnelle est définie pour la subdivision la plus défavorable, c'est à dire celle pour laquelle le nombre de canaux reliant les deux moitiés est le plus petit. On définit aussi la bandwidth bissectionnelle comme la largeur bissectionnelle multipliée par la bandwidth d'une connexion. Par exemple, le Cray T3D, avec 256 processeurs offrait une bandwidth bissectionnelle d'environ 20Gbytes/s.

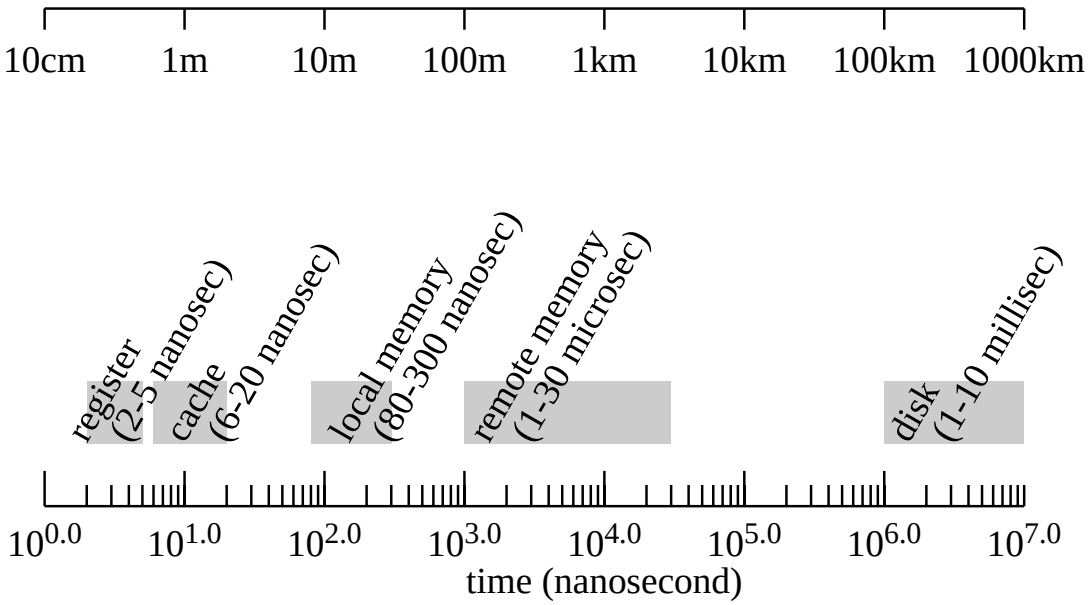


FIGURE 3.2 – *Ordre de grandeur des temps d'accès caractéristique (latence) aux différentes ressources d'un ordinateur (en l'an 2000). Pour mieux appréhender les écarts qui existent entre ces latences, l'échelle en haut de l'image donne une représentation en terme d'une distance à parcourir. On a choisi arbitrairement d'associer 10 cm au temps d'une nano-seconde (cela correspond en fait au parcours effectué en se déplaçant à 1/3 de la vitesse de la lumière). Cette échelle montre que si les données dans la mémoire cache sont à portée de main, celles qui sont sur le disque ont un voyage long comme la traversée de la Suisse.*

La scalabilité qui est la possibilité d'augmenter la taille du réseau en ajoutant des processeurs. Pour des raisons pratiques, cela implique que le degré de chaque noeud est indépendant de la taille du système (car le nombre de canaux dont dispose un processeur est fixe). La scalabilité décrit ici la nature modulaire du réseau.

Le prix (ou la complexité) de fabrication du réseau. C'est essentiellement le nombre de composants dans le réseau en fonction du nombre de noeuds présents.

La fonctionnalité du réseau qui est sa capacité de combiner des messages allant vers une même destination et à gérer des conflits de routage.

La capacité qui mesure le nombre maximum de messages pouvant être contenus en même temps dans le réseau.

La symétrie. Un réseau est dit symétrique s'il est le même vu de chaque noeud.

3.1.2 Le routage

Le routage est un algorithme adapté à un réseau donné et définissant une recette permettant aux noeuds de diriger un message *rapidement* vers sa destination finale. Cet algorithme utilise les propriétés mathématiques et géométriques du réseau pour offrir un routage dit **minimal**, c'est à dire un choix de chemins les plus courts entre la source et la destination.

Le circuit chargé du routage est appelé le **routeur**. Un message est composé tout d'abord d'une adresse de destination, puis d'un ensemble de données à transmettre (corps du message). Le routeur, en fonction de cette adresse et de l'algorithme de routage qu'il connaît, redirige le message sur un noeud voisin, afin de s'approcher de la destination finale. Le routeur doit être capable de gérer des conflits de chemin (deux messages désirant emprunter simultanément le même canal) c'est à dire, en pratique, de pouvoir mémoriser temporairement un message pour l'envoyer plus loin une fois la voie libre. L'algorithme de routage, en raison des conflits potentiels peut devenir assez complexe. Il faut être sûr qu'aucun message ne puisse être bloqué éternellement dans le réseau : risque de **deadlock** causé par deux ou plusieurs messages se bloquant mutuellement ou de **livelock** indiquant un message tournant sans fin sans jamais arriver à destination.

Plus loin dans ce chapitre, on décrira les algorithmes élémentaires de routage pour plusieurs réseaux différents, en supposant toutefois qu'un seul message circule à la fois. Les algorithmes de routage utilisés en pratique doivent en plus faire face à tous les cas de conflits qui peuvent se produire et c'est un problème difficile que de s'assurer théoriquement qu'un algorithme de routage donné fonctionne effectivement de façon correcte.

Les liens qui relient les noeuds d'un réseau d'interconnexion peuvent être soit bidirectionnels (deux messages peuvent circuler en même temps dans chaque sens) ou unidirectionnels (un seul sens à la fois). Dans les systèmes sophistiqués, le routeur peut envoyer et recevoir plusieurs messages en même temps par des liens différents. Mais souvent, l'architecture considérée ne permet qu'un message par direction. La deuxième option est plus simple techniquement et, bien sûr, moins chère. Par contre elle est plus lente car elle nécessite de parcourir séquentiellement les directions du réseau pour envoyer et recevoir les messages. Les communications n'utilisant qu'un seul canal à la fois sont dites **single-port** ou encore *processor bound*. Par contre, celles qui utilisent plusieurs canaux simultanément sont dites **multi-port** ou encore *link bound*. Il faut remarquer que les transferts multi-port demandent des accès mémoire plus rapides pour alimenter la communication et aussi des algorithmes plus complexes pour réaliser certains échanges de données spécifiques.

L'action du routeur est **locale**, par opposition à une situation **centralisée** dans laquelle les chemins seraient déterminés sur la base de l'ensemble des messages à acheminer. Une stratégie globale peut s'avérer très complexe avec des grands systèmes et des communications irrégulières mais elle permet en principe de contribuer à empêcher les conflits entre divers messages qui veulent emprunter la même connexion.

Dans une stratégie de routage **locale** (ou répartie), chaque noeud achemine les messages qui lui arrivent uniquement sur la base d'une information purement locale. Si cette information est l'adresse de destination portée par le message, on parle de routage local **déterministe**. Il existe aussi des stratégies de routage dite **adaptatives** qui utilisent le fait que, souvent, plusieurs chemins de même longueur sont possibles. Ces stratégies prennent en compte une information sur la contention locale du réseau (ou éventuellement des pannes) pour choisir vers quel noeud suivant le message doit être envoyé. Lorsque les chemins alternatifs sont tels que le message s'approche toujours de la destination, on parle de routage **profitable**. Dans le cas contraire, on accepte que le message fasse un détour localement. Un routage profitable évite les livelocks mais pas les deadlocks.

De plus, le routage est soit *synchrone*, soit *asynchrone*. Dans le premier cas, tous les messages sont envoyés simultanément et sont soumis à des cycles de routage complets. Le temps de transfert est alors celui du message le plus lent. C'est la stratégie utilisée dans les machines SIMD.

Le mode d'opération asynchrone est, quant à lui, plus souple et mieux adapté à une architecture MIMD : chaque noeud envoie ou reçoit les messages selon les besoins du programme, à des moments différents.

On définit le **trafic** dans le réseau comme le nombre de canaux d'interconnexion utilisé lors d'une communication. Souvent le trafic n'est pas homogène sur tout le réseau et il arrive qu'un noeud unique reçoive un volume disproportionné de données. Un tel noeud est alors un *hot-spot* et il est important d'avoir des capacités de routage ou de combinaison de messages efficaces afin de ne pas

dégrader les performances du réseau.

Il peut paraître curieux de remarquer que les réseaux à basses dimensions, avec une faible connectivité, telles les grilles bidimensionnelles, peuvent être plus efficaces que des réseaux à grande connectivité (hypercubes). En effet, avec moins de chemins disponibles, mais une bandwidth importante, il est plus facile de bien utiliser les ressources disponibles et d'équilibrer le trafic. S'il y a trop de chemins possibles, on aura souvent plusieurs chemins inoccupés, ce qui résulte en une faible utilisation du réseau.

3.1.3 Primitives de communication

Dans une machine parallèle, il y a certains types de communications qui reviennent fréquemment et que le routeur doit être capable de réaliser efficacement. Souvent, ces communications donnent lieu à des instructions particulières dans les bibliothèques d'échange de messages. Parmi les primitives de communication les plus courantes, on peut mentionner

- Les communications **point-à-point** ou **one-to-one** qui consistent en des échanges de données entre paires de processeurs. C'est le cas le plus simple et le plus générique de communication.
- Les **permutations** : le processeur i communique avec le processeur $f(i)$ où f est une bijection quelconque dans l'ensemble des noeuds du réseau. C'est un cas particulier où tous les processeurs sont simultanément impliqués dans une communication point à point.
- Le **broadcast**, **one-to-all** ou la **diffusion** qui est une opération très courante : un processeur donné envoie un message à tous les autres. Dans le même ordre d'idée, on a aussi le **multicast** ou le **one-to-many**, où un processeur communique une information à un groupe de processeurs.
- L'opération de **réduction**, ou le **many-to-one**, par laquelle un ensemble de processeurs envoie un message différent sur un processeur unique où toutes les informations sont combinées (par exemple au moyen une opération arithmétique ou logique).
- L'**échange total** ou le **all-to-all** qui implique que tous les processeurs envoient une information à tous les autres. C'est en quelque sorte un broadcast multiple simultané.
- Le **one-to-all personnalisé**, la **distribution** ou encore l'opération de **scatter**. Un processeur source envoie un message **différent** à tous les autres.
- L'opération de **gather** qui est en quelque sorte l'inverse du scatter : un processeur destination reçoit un message différent de tous les autres et les stocke dans un buffer de sorte que le message venant du $i$ème processeur soit en position i.
- L'**échange total personnalisé** qui désigne une communication dans laquelle chaque processeur envoie un message différent à tous les autres.

— Les opérations de **préfixes parallèles** ou **scans** qui combinent communications et calcul de sorte que la donnée finale du $i$ème processeur soit la somme (ou le résultat d'une autre opération arithmétique ou logique) des données originales contenues dans les processeurs 1 à $i-1$. Ces opérations permettent de faire des sommes partielles à travers les processeurs, ou encore, si l'opération utilisée est par exemple un `max`, d'obtenir la valeur maximum d'une variable dans tous les processeurs de rang inférieurs.

Plusieurs de ces primitives (par exemple l'échange total, les scans ou le broadcast) impliquent la participation de tous les processeurs et donc éventuellement leur coordination ou synchronisation. On désigne souvent ces primitives par le terme de **communications collectives**.

3.1.4 Techniques de commutation

Il y a plusieurs techniques associées à la manière dont un message (paquet) est transmis de proche en proche, du noeud initial jusqu'à sa destination. Le mode de commutation le plus simple est le **Store-and-Forward** qui correspond à une commutation de message. Le principe est d'envoyer un message du noeud i au noeud $i+1$ où il est stocké à mesure qu'il arrive. Après réception complète du message, le noeud $i+1$ l'envoie de la même manière au noeud $i+2$, selon le chemin spécifié par le routeur. Le temps de transfert du message est ainsi proportionnel au nombre de noeuds à traverser et au temps de transit du message entre deux noeuds consécutifs (c'est à dire, la longueur du message). Cette technique est illustrée sur la figure 3.3. Si W est la bandwidth du réseau (c'est à dire le nombre de Bytes qui peuvent traverser chaque ligne en une seconde), alors le temps de transfert T_{sf} du message est

$$T_{sf} = t_s + (n-1)\frac{L}{W} = t_s + L(n-1)t_w$$

où L est la longueur du paquet en bytes, n le nombre de noeuds à traverser, t_s le temps de préparation du message et $t_w = 1/W$ le temps de transfert d'un byte.

La technique du Store-and-Forward n'est pas rapide et n'utilise pas efficacement les possibilités de communication du réseau. Pour cette raison elle n'est plus guère utilisée dans les réseaux modernes. La commutation de circuit dynamique ou **cut-through** permet des performances beaucoup plus intéressantes. C'est une technique très utilisée depuis le début des années 90. Le principe est de permettre au message de poursuivre sa route tant qu'il n'est pas complètement reçu. La technique du **Wormhole** (ou commutation par **trains de bits**) est un exemple. Les messages sont découpés en petits morceaux de taille fixe appelés *flits* (flow-control digits). Le premier flit contient l'adresse de destination et les suivant les données. Typiquement, un flit correspond à 8 bits, mais ceci peut dépendre de la taille du réseau. Avec 256 noeuds, 8 bits sont suffisants pour coder toutes les destinations. Dans la technique du wormhole, la taille des buffers sur

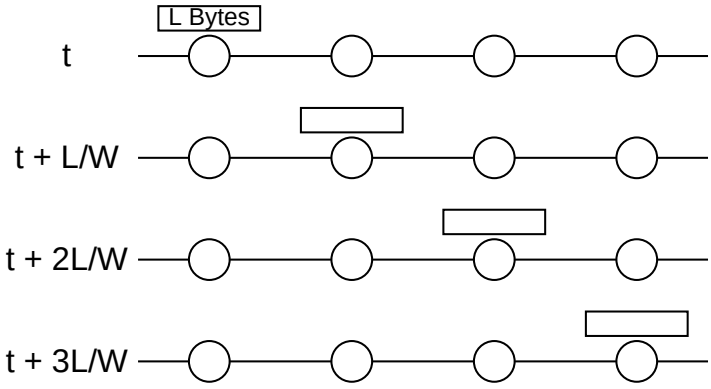


FIGURE 3.3 – *Illustration de la technique d'acheminement Store-and-Forward. Un message de L Bytes (indiqué par le rectangle) traverse 4 noeuds du réseau d'interconnexion (cercles). Le message est totalement reçu par chaque noeud avant d'être réexpédié plus loin.*

les noeuds intermédiaires est réduite à un seul flit, alors que dans le cut-through plus général, ces buffers peuvent contenir plusieurs flits.

Les flits traversent le réseau de façon "pipeline," à la manière d'un ensemble de wagons qui suivent la voiture de tête, seule à connaître l'adresse de destination du message et à "creuser" le chemin. Cette technique est illustrée sur la figure 3.4.

Le temps de transfert du message est dans ce cas le temps mis par l'entête (qui est proportionnel à la longueur du chemin) plus un temps proportionnel au nombre de flits restants du message (c'est à dire à la longueur du message). Si la bandwidth du réseau est W, chaque flit (1 byte) met un temps $t_w = 1/W$ pour passer entre deux noeuds adjacents. Donc, si n est le nombre de noeuds à traverser et L la longueur du message, le Wormhole nécessite un temps de transfert

$$T_{wh} = t_s + \frac{(n-1+L-1)}{W} = t_s + (n-1+L-1)t_w$$

On constate ainsi que la commutation Wormhole est beaucoup plus efficace que la commutation Store-and-Forward car elle est proportionelle à la somme $L+n$ de la longueur du chemin et de la longueur du message plutôt qu'à leur produit. Ainsi, si le diamètre du réseau est suffisamment petit, on peut négliger n par rapport à L et dire que le temps de transfert par Wormhole est approximativement proportionnel à la longueur du message, si celui-ci est assez long.

Un autre avantage de la commutation Wormhole est qu'elle utilise moins de mémoire que le Store-and-Forward.

La technique du cut-through est une amélioration de la commutation de circuit usuelle dans laquelle l'ensemble du chemin est réservé pour le passage du message. L'entête du message est là aussi responsable d'établir le chemin mais c'est seulement lorsqu'elle est a destination que le reste du message est envoyé

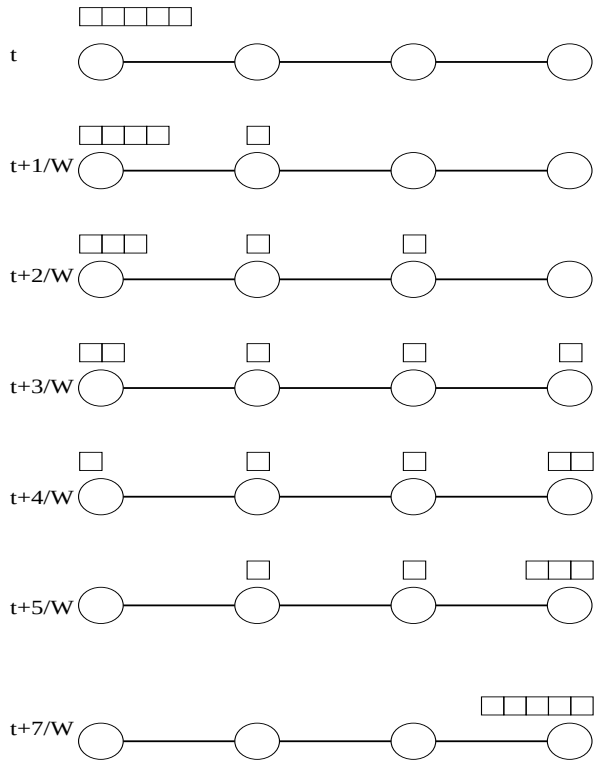


FIGURE 3.4 – *Illustration de la technique d'acheminement Wormhole. Le message initial (rectangle) est divisé en flits (carrés) et traverse le réseau à la manière d'une locomotive qui tire ses wagons.*

d'une traite. De plus, dans la technique du Wormhole, le chemin est libéré immédiatement après le passage du dernier flit et non à la fin du transfert. Le Wormhole est très efficace pour les messages relativement courts, tels que ceux que l'on rencontre dans le calcul parallèle. Elle permet aussi de facilement dupliquer un message sur un noeud donné, ce qui est utile pour les opérations comme le broadcast. La commutation de circuit classique ne permet pas cette duplication des messages mais reste une solution intéressante pour les très longs messages.

Si un chemin est bloqué par un autre message en cours, la tête du message arrivant est arrêtée, ainsi que tout le reste, jusqu'à libération du chemin. Il faut évidemment éviter tout risque de deadlock lorsque plusieurs messages se bloquent mutuellement. Une solution pour résoudre une telle situation est d'avoir des canaux supplémentaires entre les routeurs, par exemple en créant des **canaux virtuels** par un multiplexage en temps du canal physique. Ainsi, chaque direction dispose d'une tranche de temps pour passer et chaque routeur dispose de plusieurs buffers pour accueillir les flits provenant des canaux virtuels différents.

Le concept de canal virtuel est aussi très important pour augmenter le débit global du réseau. Par exemple, si le corps d'un message M_1 bloque un noeud intermédiaire P_j car sa tête ne peut momentanément pas progresser, un autre

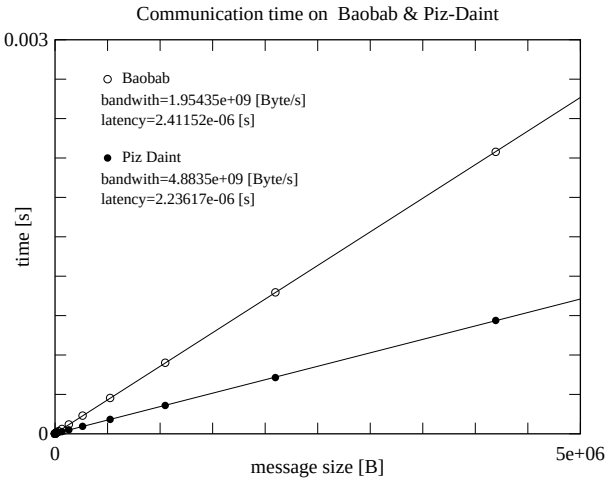


FIGURE 3.5 – Temps de communication entre deux processeurs, pour les machines Baobab de l'UNIGE et Piz Daint du CSCS. Mesure par C. Kotsalos, 2020

message M_2 ne pourra pas, sans canal virtuel, traverser P_j, même si c'est pour se rendre dans une direction qui est libre. Le partage en temps du routeur permet, par le mécanisme des canaux virtuels de libérer P_j et de permettre l'acheminement du deuxième message. On augmente ainsi le throughput par une meilleure utilisation des ressources.

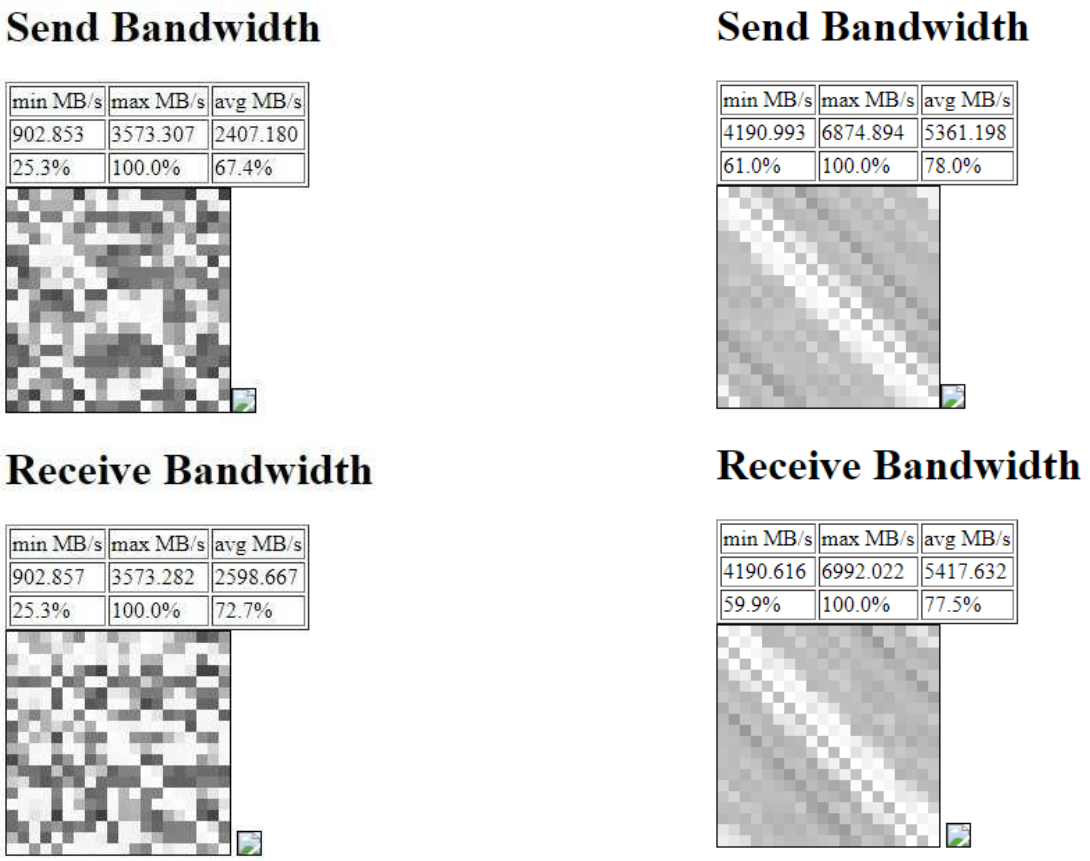
3.1.5 Exemple de performances

La figure 3.5 illustre le résultat de la section précédente, à savoir que le temps de communication entre deux processeurs peut s'approximer par une dépendance linéaire dans la taille M du message

$$T_{comm} = t_s + \frac{M}{b}$$

où t_s est un temps de latence et b est la bande passante. On obtient une latence de l'ordre de 2 μs et une bande passante variant entre 2 et 5 GB/s.

La figure 3.6 montre que si le réseau d'interconnexion est puissant et homogène (cas de Piz Daint) les mêmes performances se retrouvent entre n'importe quelle paire de noeuds. Dans le cas de Baobab, le réseau d'interconnexion est heterogène car beaucoup d'utilisateurs n'ont pas de grands besoins de communication. Dès lors le réseau infiniband n'est pas déployé entre tous les noeuds, ce qui explique la diversité des bandes passantes mesurées dans la figure 3.5.



Baobab Piz Daint

FIGURE 3.6 – Bande passante entre différentes paires de noeuds dans les machines Baobab (UNIGE) et Piz Daint (CSCS). Measures : C. Kotsalos, 2020