La communication dans un cadre MIMD distribué

Loïc LAGADEC

1er juillet 1996

Rapport de DEA (étude Bibliographique)
IFSIC - Université Rennes I

The hypothesis behind our design is that performance will be bounded by interactions between kernel and user space rather than by the bandwidth delivered by the LAN Data-Link Layer.
(J.Bruck, D. Dolev, C-T. Ho, M-C. Roşu, R. Strong, [1])

TABLE DES MATIÈRES

Table des figures

Table des tableaux

Introduction

> "Depuis le premier ordinateur, l'évolution des machines obéit à deux grandes règles fondamentales: l'évolution des performances résulte des progrès technologiques d'une part, et le coût des développements logiciels est devenu très rapidement prépondérant par rapport aux coûts des matériels d'autre part." (D. Etiemble [2])

Les performances des technologies MOS ont progressé de façon exponentielle ces vingt dernières années, jusqu'à "rattraper" les technologies performantes d'autrefois. La technologie standard (microprocesseurs, mémoires dynamiques, disques,...) est donc le moteur de l'évolution. La facilité de programmation et son efficacité sont deux critères également très importants qui permettent de réduire les coûts logiciels; une adéquation simple des modèles de programmation et d'exécution est donc souhaitée. Cependant, bien que leurs modèles de programmation et d'exécution soient proches, les processeurs SIMD sont de plus en plus remis en cause par des questions matérielles (impossibilité de diffuser à haute fréquence une instruction à plusieurs milliers de processeurs, spécificité des PEs ne permettant pas de tirer le meilleur parti des microprocesseurs standard,..). Le problème des MIMD est en revanche d'harmoniser les relations entre un modèle de programmation simple et les accès aux données physiquement réparties. Ces architectures semblent tendre de plus en plus nettement vers un parallélisme à gros grain (processeurs plus puissants mais moins nombreux).
Plus récemment des réseaux de stations de travail sont apparus. La modularité, la disponibilité des machines, le coût restreint sont autant d'arguments qui permettent d'expliquer la multiplication des réseaux de stations en vue de réaliser des machines parallèles performantes.

Cette étude bibliographique s'appuie principalement sur l'article **"Decoupling synchronization and data transfer in message passing systems of parallel computers"** [3] de J. Strichnoth, T. Stricker, D. O'Hallaron, S. Hinrichs et T. Gross. La technique du "passage de messages" a été présentée par les chercheurs de Caltech [1] [4] afin de permettre le recouvrement entre calculs et communications. Pour ce faire, un processus qui désire effectuer une communication doit réaliser un appel système, ce qui implique des coûts supplémentaires (création de processus, commutation de contexte [5],...).

Le problème est donc d'obtenir un gain en temps supérieur aux délais induits par le système. Dans ce but, divers modèles de communication ont vu le jour: modèles *rendez-vous* (1.2.1), *postal* (1.2.2) ou *écriture distante* (1.2.3), qui se distinguent principalement par la nature du contrôle (plus ou moins prononcé) et le couplage entre celui-ci et le transfert de données (1.4).
Les performances (2) de ces différents systèmes dépendent de la nature du travail à effectuer (grain du parallélisme, taille des messages, importance des opérations de communication,...) comme de l'architecture cible.
Cependant, la portabilité d'un code étant une qualité importante, des bibliothèques de passages de messages (3) ont permis de se libérer de ces contraintes architecturales spécifiques. De plus ces bibliothèques ont permis l'émergence de nouvelles architectures: les réseaux de stations de travail (NOW).

Les principales articulations du plan sont schématisées par la figure 0.1.

[1] CALifornia institute of TECHnology

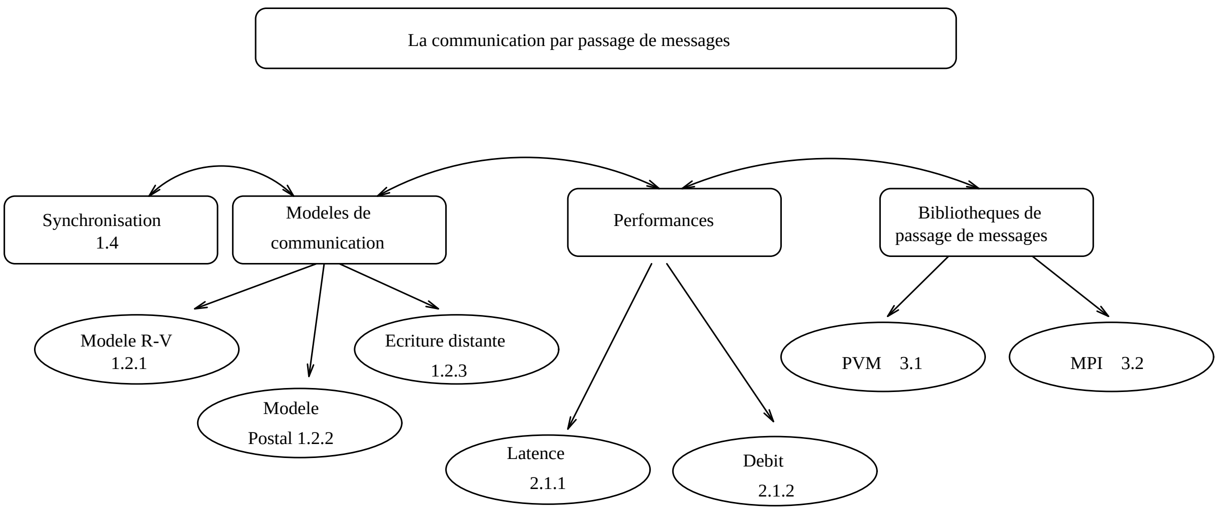


Figure 0.1 : Plan simplifié

Chapitre 1

La communication.

Le traitement parallèle permet aujourd'hui d'augmenter la capacité de calcul en faisant coopérer plusieurs microprocesseurs.
L'idée consiste à répartir le travail total entre plusieurs entités (processeurs) de façon à obtenir un gain en temps potentiellement proportionnel au nombre d'acteurs. Pour ce faire, divers supports architecturaux ont vu le jour (1.1), qui se caractérisent principalement par leur gestion mémoire: les architectures à mémoire distribuée d'une part et d'autre part celles à mémoire partagée.
Le moyen de communication utilisé dépend directement du choix architectural: envoi de messages dans le premier cas et modification de variables partagées dans le second.

Nous nous proposons dans cette première partie de présenter les différents modèles de passage de messages existants. Nous introduirons également la notion de mémoire distribuée partagée avant de nous intéresser au contrôle nécessaire dans le cadre du passage de messages.

1.1 Contexte du travail.

Avant d'aborder les différentes techniques de passage de messages, il nous paraît intéressant de présenter les différents types d'ordinateurs parallèles. Il existe en effet deux grandes classes de machines parallèles: les machines à mémoire distribuée et celles à mémoire matériellement partagée. La figure 1.2 correspond à une schématisation des différentes architectures matérielles existantes.

Les architectures à mémoire partagée présentent comme avantage d'être plus simples à programmer en raison de leur espace d'adressage global. En revanche la complexité des mécanismes à mettre en œuvre (qui sont transparents pour l'utilisateur) limite l'extensibilité matérielle du système.

Les architectures à mémoire distribuées sont, en revanche, plus modulaire. La première machine à mémoire distribuée crée fut le *cosmic cube* développé à Caltech en 1985.

La classification suivante est extraite du cours de François Bodin. Elle présente différentes catégories de machines parallèles.

SASM (*Shared Address space Shared Memory*): les machines à mémoire partagée ou UMA pour *Uniform Memory Access*(exemple CRAY C90).

SADM (*Shared Address space Distributed Memory*): les machines à mémoire virtuellement partagée. On peut différencier:

- NUMA pour *non Uniform Memory Access* (exemple CRAY T3D)
- CC-NUMA pour *Cache Coherent NUMA* (exemple ALEWIFE)
- COMA pour *Cache Only Memory Access* (exemple KSR1)
- OSMA pour *Operating System Memory Access*: l'équivalent d'une architecture NORMA où la gestion de la mémoire virtuelle est étendue afin qu'elle soit globale à tous les processeurs. La

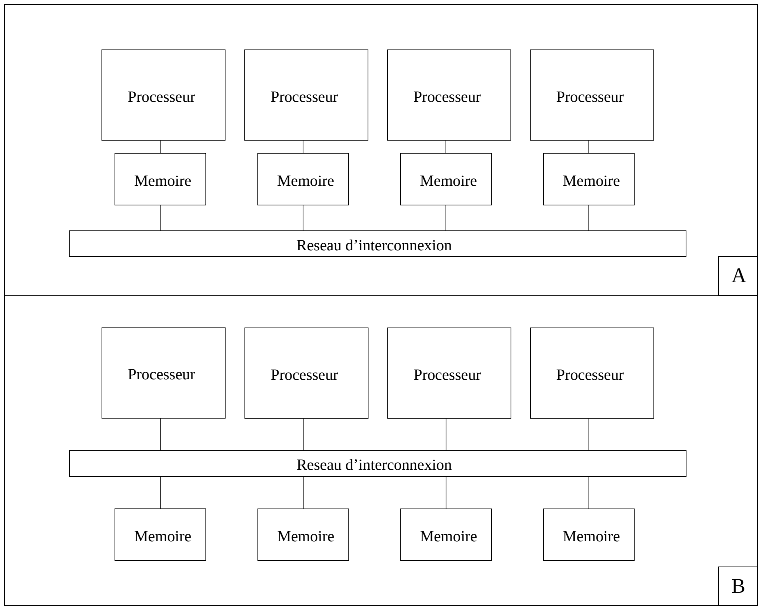


Figure 1.1 : Architectures à mémoire distribuée (A) et à mémoire partagée (B)

cohérence est alors gérée au niveau des pages. Le modèle OSMA est parfois qualifié de *COMA soft*.

DADM (*Distributed Address space Distributed Memory*): les machines à mémoire distribuée classiques ou NORMA pour *NO Remote Memory Access* (exemple Paragon, CM-5 [6])

Le choix de l'organisation mémoire reste à ce jour un des derniers choix architecturaux non tranchés.

1.2 La communication par passage de messages

Dans le cadre de mémoire distribuée, les communications prennent effet par l'intermédiaire de messages. Ce mécanisme a été proposé par Seitz [4] en 1985 en vue de fournir des communications non bloquantes. Le mécanisme de passage de messages consiste à envoyer d'un nœud à un autre un paquet de données

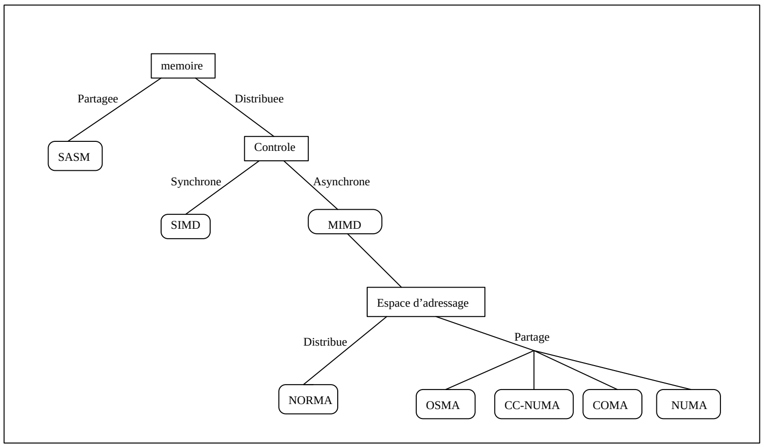


Figure 1.2 : Modèles matériels

encapsulé dans un message. Celui-ci se compose de deux parties distinctes, son enveloppe et son contenu (les données à expédier). L'enveloppe contient diverses informations telles que l'identité de l'émetteur, celle du récepteur, la longueur du message, et une estampille temporelle. Cette estampille permet de préserver les relations de causalité dans un système distribué par l'introduction d'un temps logique (en particulier le "temps de Lamport" [7] est bien connu).

Il existe deux catégories de messages:

Les messages de données qui transportent les valeurs. Ces messages sont fragmentés en paquets de taille fixe pour simplifier la gestion des tampons. Il faut alors les réassembler dans le bon ordre.

Les messages de contrôle de taille faible qui permettent de gérer la synchronisation (1.4), les acquittements et les régulations de flux [8]. En particulier la plupart des réseaux sont fiables (au sens *sans interblocage* ou *dead-lock free*) à la condition qu'un receveur consomme continuellement les messages qui arrivent. Pour garantir cette propriété une politique de gestion de flux est nécessaire.

Définition 1 (Contrôle) *Nous entendons par contrôle le contrôle interne ou système. Le contrôle de plus haut niveau n'est pas considéré dans cette étude bibliographique.*

Les différentes techniques de passage de messages peuvent se regrouper en deux catégories:

Communications bloquantes Dans ce cas l'émetteur reste bloqué jusqu'à la réception d'un acquittement puis reprend son exécution.

Cette propriété peut conduire à une situation d'interblocage.

Communications non bloquantes Dans cette configuration il est possible d'opérer un recouvrement entre communications et calculs. Le modèle le plus répandu est le *modèle postal* que nous décrivons dans la partie 1.2.2. Le modèle *Deposit/fetch* est, lui, présenté dans 1.2.3.

La nature asynchrone des communications non-bloquantes implique qu'un processus peut envoyer une série de messages. Si le délai d'émission est inférieur au cycle de réception, soit le récepteur sera toujours en attente de réception, soit il aura alloué un tampon pour recevoir de façon asynchrone chaque message. Dans le cas contraire, le récepteur doit asservir sa vitesse de "consommation" ou celle d'émission faute de quoi des messages seront perdus. On parle de contrôle de flux.
Le contrôle de flux est ou à la charge du programmeur ou géré par le système suivant les modèles.

Lorsque les liens matériels ne garantissent pas l'arrivé des messages dans l'ordre d'émission (les canaux sont non FIFO), il est possible d'imposer certaines contraintes sur les messages.
Principalement quatre types de messages existent [9].

Les messages ordinaires (typiquement des messages de données ne subissant pas de contrainte d'ordonnancement particulière).

Les messages contrôle passé ne peuvent pas doubler les messages émis avant eux.

Les messages contrôle futur ne peuvent pas être doublés.

Les marqueurs ne peuvent ni être doublés ni doubler un autre message.

Ces caractéristiques peuvent en particulier permettre de réassembler les paquets de données dans le bon ordre pour reconstituer le message (déséquencement) et de capturer des états locaux.

1.2.1 Modèle Rendez-Vous

Dans le modèle *Rendez-vous*, une synchronisation totale est requise entre les deux protagonistes: on parle de poignée de main, ou d'affectation à distance. Les deux opérations SEND et RECEIVE sont bloquantes. L'échange ne s'opère que lorsque les deux requêtes sont effectives. Bien que ce modèle présente un intérêt théorique, il n'est que rarement utilisé du fait de ses faibles performances. Nous ne le considérerons plus dans la suite de ce travail.

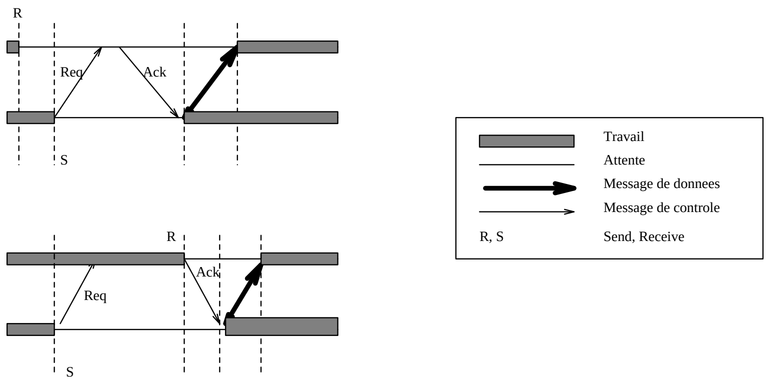


Figure 1.3 : Modèle Rendez-Vous

1.2.2 Modèle Postal

Dans le *Modèle postal*, l'émetteur et le récepteur participent à l'échange. L'ordre des opérations *SEND* et *RECEIVE* est indifférent. Lors de l'émission d'un message, celui-ci est placé dans un tampon jusqu'à ce que le port du réseau soit libre. Il est ensuite envoyé au récepteur puis de nouveau tamponné en attendant que ce dernier y accède. L'adresse source est donc déterminée par l'émetteur tandis que l'adresse cible est définie par le récepteur.
En général, la réception n'est pas bloquante: le récepteur n'interrompt son exécution que s'il attend un message nécessaire et non encore arrivé.
Une politique de synchronisation (au niveau de l'application) ou de gestion efficace des tampons est donc nécessaire pour assurer de bonnes performances.

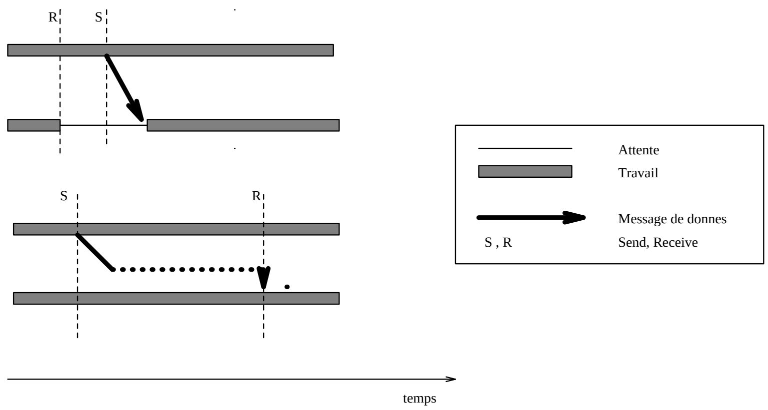


Figure 1.4 : Modèle Postal

Le *modèle postal* correspond à un échange de messages standard. Il est implémenté dans de nombreuses librairies de passage de messages comme PVM [10] et MPI [1] qui sont étudiées en 3.1.
Le modèle *request-response message passing* constitue une variante du schéma précédent. Les messages sont tamponnés de manière optimale chez l'émetteur et ne parcourent le réseau qu'après qu'une requête soit intervenue, de façon à ne pas provoquer de congestion du réseau.

1.2.3 Ecriture distante

Contrairement au modèle précédent, seul un des acteurs de l'échange intervient réellement. Dans le modèle *Deposit*, l'émetteur dépose les données dans l'espace mémoire du receveur.

Les *messages actifs* permettent d'implanter ce modèle [3]: c'est la méthode qu'ont retenue Katevenis et Markatos dans le cadre du projet telegraphos [11].

La tête des *messages actifs* (développés à l'université de Berkeley [12] [13]) ou *active messages* comporte une adresse qui correspond à une séquence d'instructions de niveau utilisateur qui va extraire le message du réseau et l'intégrer au travail en cours.
Le corps du message joue le rôle d'argument de cette séquence d'instructions.
Cette technique repose donc sur une image du code uniforme sur tous les nœuds (modèle SPMD).

Les *messages actifs* constituent un mécanisme de communication asynchrone, l'émetteur continue son exécution après avoir envoyé le message. Le receveur est notifié de l'arrivée d'un nouveau message par une interruption. Ceci signifie que les *messages actifs* ne sont jamais tamponnés (sauf par la couche transport du réseau si nécessaire). Cette absence de bufférisation est la clef de l'optimisation des *messages actifs* par rapport aux modèles classiques *Send/Receive*.
Par ailleurs les *messages actifs* se distinguent des appels de procédures à distance classiques en ce sens que le rôle du *message handler* n'est pas d'effectuer un quelconque traitement sur le corps des messages mais de les extraire du réseau et de les intégrer dans le calcul courant. Lorsqu'on utilise des *messages actifs*, le bus peut être vu comme un pipeline opérant à un rythme déterminé par l'*overhead* (2.1.2) et la latence (fonction de la longueur du message et de la profondeur du réseau).

1.2.4 Les communications globales

Il est souvent nécessaire de réaliser des échanges impliquant plus de deux acteurs. On définit les principales communications globales comme suit:

La diffusion (ou *broadcast*) consiste à expédier le même message à tous les autres nœuds. Si seule une portion de ces nœuds est concernée par l'échange, on parle de *diffusion partielle* ou *multi-cast*.

Le regroupement (ou *gathering*) est l'opération inverse: plusieurs processus regroupent une information vers un processus particulier.

L'échange total (ou *all-to-all*) correspond à une diffusion de tous les processus.

Les communications personnalisées impliquent que le message envoyé à un processus dépend de celui-ci.

Les communications globales constituent donc un paradigme: des fonctions spécifiques permettent de remplacer l'envoi de n messages par l'expédition d'un nombre de messages restreint (qui dépend de l'architecture). Chacun des n nœuds destinataires recevra une copie de ce message.

1.3 La Mémoire virtuellement partagée.

Afin de pouvoir combiner les avantages d'une mémoire partagée et des techniques de passage de messages, la notion de *Mémoire virtuellement partagée* semble vouloir s'imposer comme une solution agréable [14] [15] (on se reportera avec intérêt aux travaux de T. Priol [16]).

La mémoire virtuellement partagée est un mécanisme qui offre une vue globale de la mémoire physiquement distribuée en effectuant des copies de mémoire locale si nécessaire. Le coût en temps engendré par cette technique peut être réduit par la mise en œuvre de *prédictions de demandes*. Les données sont dans ce cas copiées avant que la requête n'en soit faite. On parle de pré-placage ou *pre-fetching* [17].

Dans un cadre de *mémoire partagée distribuée*, les communications inter-processeurs s'effectuent par envois de messages [18] [15], mais les échanges locaux à un processeur (eg. à des processus présents sur le même processeur) correspondent à une suite d'opérations de lecture et d'écriture sur des variables partagées.

Les *threads* ou *processus légers* sont des processus qui partagent le même espace d'adressage (hérité du père). Ils peuvent donc communiquer en modifiant des variables partagées et réaliser des opérations à grain

de parallélisme très fin.
Le coût de cette méthode tient dans les mécanismes à mettre en œuvre pour assurer la cohérence des données (conflit de type *read after write*).

L'intérêt évident de ce type d'architecture (par exemple le projet *Alewife* du MIT [19]) est d'offrir deux moyens de communication différents qui seront utilisés de façon optimale.
De plus, cette configuration permet de paralléliser des programmes séquentiels, ce qui est particulièrement difficile lorsque l'architecture ne supporte d'autre moyen de communication que le passage de message.
En vue de simplifier et de standardiser la gestion de la mémoire virtuellement partagée, SCI (*the Scalable Coherent Interface*) [1] [20] est née en 1987. La complexité de la préservation de la cohérence mémoire est cachée par le système; les protocoles de communication peuvent utiliser les variables communes pour gagner en simplicité [21].

La mémoire virtuellement partagée permet la manipulation de données localement présentes mais appartenant à un autre processeur. Toute modification doit être diffusée pour éviter le risque de faux partage. Les deux techniques pour maintenir la cohérence des données sont:

- l'invalidation des autres copies,
- la mise à jour des autres copies.

Les données sont regroupées par pages (blocs de taille fixe), ce qui implique que certains algorithmes sont inadaptés. En effet la modification d'un portion de la page entraine la mise à jour ou l'invalidation de l'ensemble de la page. On souhaite donc généralement utiliser la règle des écritures locales ou *owner compute rule* pour éviter ces problèmes.

1.4 La synchronisation

Dans un cadre parallèle distribué, le contrôle intervient à deux niveaux: le contrôle effectué par le système (gestion de lien, allocation de tampons, etc...) et contrôle de processus à processus (de plus haut niveau — application distribuée ou parallèle).
Il apparaît donc nécessaire de lever cette ambiguïté: nous ne traiterons ici que du contrôle interne (ou système). La synchronisation a donc pour but de garantir l'allocation de tampons.

Lorsque le contrôle et les données sont couplés (comme dans le cas du modèle postal), le système effectue le même traitement sur tous les messages. La distinction entre contrôle et données n'apparaît qu'au moment de l'interprétation du contenu du message.

Stricker, Stichnoth, O'Hallaron, Hinrichs et Gross [3] suggèrent de découpler la synchronisation et l'échange de données. Les trois raisons invoquées sont:

1. Coupler les échanges et le contrôle peut impliquer des opérations de stockage coûteuses des messages de contrôle si celles-ci sont nécessaires pour les données.
2. Si aucun tampon système n'est fourni et que l'échange se déroule sous l'entière responsabilité de l'utilisateur (ce qui est souhaitable pour éliminer les copies), alors on ne peut se reposer sur la synchronisation induite par les transferts.
3. Si on désire envoyer les données directement vers leur destination finale, la synchronisation est requise avant l'échange.

Dans le modèle *request-response* (1.2.2) les messages ne sont envoyé que lorsque le récepteur est prêt à les recevoir. Ici, le contrôle permet de garantir que le message peut être directement déposé dans l'espace mémoire du receveur et à son adresse finale. Ce qui élimine toute congestion du réseau.

[1] standard ANSI/IEEE 1596

De plus ce découplage offre des opportunités d'amélioration du système de communication. Les programmes parallèles générés par le compilateur peuvent gérer eux-même les tampons ainsi qu'inclure du code de synchronisation. Ces programmes ne s'appuient plus sur la synchronisation induite par les transferts de donnés [2]. Il n'est plus nécessaire de renvoyer un acquittement à chaque demande d'allocation de tampon individuel; les acquittements correspondent à des ensembles de tampons gérés comme des unités par le compilateur.

Enfin, les messages de contrôle de flux sont souvent très réguliers. Dès lors que la nature des communications est connue, chaque nœud connait précisément les ressources (tampons) concernées. Les messages de gestion de flux ne transportent plus que des informations de séquencement, ce qui permet de les remplacer par des messages vides qui seront interprétés comme contrôle [3].

Une seconde solution étudiée consiste à utiliser des mécanismes matériels pour la synchronisation (des barrières matérielles) ce qui permet des gains de complexité (réduction de $\mathcal{O}$ (n^2) à $\mathcal{O}$ $(n \times log\ n)$ pour une diffusion sur le CRAY T3D [3]). Ce genre de possibilités n'existe pas sur des réseaux de stations de travail, nous ne les considèrerons plus dans la suite de ce travail.

En résumé

L'usage des messages actifs pour découpler le contrôle du transfert de données apparaît donc comme une solution intéressante. Il permet d'optimiser le protocole d'envoi de messages en fonction de son utilisation, soit d'éviter le cas général d'un protocole coûteux (à condition d'utiliser un mode de programmation SPMD). Les messages se regroupent donc en deux catégories suivant leur taille (standard pour les messages de données et réduite pour le contrôle). La distinction de ces deux catégories de messages permet par ailleurs de développer deux voies d'optimisation totalement différentes: la bande passante ou débit pour les messages de taille standard et la latence pour les messages courts (de contrôle). Ces deux notions sont explicitées dans la partie suivante.

[2]Dans le modèle Deposit/fetch il existe une possibilité de voir des donnés valides écrasées, on ne peut donc pas se fier à la synchronisation associée aux transferts de donnés.

[3]Les mécanismes de passages de messages sont généralement prévus pour transporter d'importants paquets de donnés (2.1.2) mais des solutions ont été étudiées pour les messages de courte taille [22].

Chapitre 2

Performances en communication.

Nous avons présenté dans le chapitre précédent plusieurs modèles de passage de messages. Après cette approche qualitative, nous nous intéressons maintenant à une vision quantitative.
Pour ce faire, cette partie présente quelques critères essentiels qui permettent d'évaluer les performances en communication dans un cadre de passage de messages. La gestion des entrées/sorties étant un paramètre déterminant, nous nous attacherons à expliciter les problèmes rencontrés et à présenter diverses solutions proposées.
Enfin nous indiquerons quelques chiffres significatifs en relation directe avec les notions introduites précédemment.

2.1 Définitions

Les deux caractéristiques principales d'un système de communication qui définissent les performances sont **la latence** et **le débit**.

Définition 2 *La latence correspond au délai entre l'émission et la réception d'un message. Elle peut parfois être occultée par le recouvrement entre calculs et communications.*

Définition 3 *Le débit correspond à la quantité d'informations qui peut transiter par unité de temps.*

Un des problèmes inhérents aux systèmes distribués auquel on est confronté pour mesurer la latence est l'absence d'horloge globale. Il convient donc de définir un moyen de mesurer cette grandeur sur un seul site. La méthode utilisée consiste à effectuer un aller-retour. Le temps considéré est alors la moitié du temps nécessaire à cet aller-retour.

$$\boxed{t_{A \rightarrow B} = \frac{t_{A \rightleftharpoons B}}{2}}$$

Notons que la valeur retenue correspond à la moyenne des temps mesurés lors d'une rafale d'aller-retour.

2.1.1 La latence

Le temps total écoulé entre l'envoi d'un message et sa réception est le cumul des temps intermédiaires que Dillon, Gamboa Dos Santos et Guyard présentent dans [23]. Ces temps sont les suivants:

t_{Start} : Temps nécessaire à des calculs internes (établir la connexion, allouer des tampons,...) avant la communication.

t_{Pack} : Temps de codage des données en format XDR [1] dans le cas de réseaux hétérogènes, de copie des messages et de gestion des tampons.

t_{Send} : Temps réellement utilisé pour envoyer le message, c'est-à-dire à le transmettre à l'interface réseau.

t_{Net} : Le temps de parcours du réseau, qui correspond à

$$\frac{L_{lien}}{V_{propagation}} + \underbrace{n_{commutateur} \times \delta_{commutateur}}_{Si\ le\ réseau\ est\ commuté}$$

t_{Recv} : Délai entre la disponibilité du message et le moment où le receveur l'extrait du réseau.

t_{Unpack} : Temps de décodage des données, de copie du message chez l'utilisateur et de gestion des tampons. Comme pour t_{Pack} ce temps dépend de la nature du réseau (e.g. nul si le réseau est homogène et positif si celui-ci est hétérogène).

Puisque t_{Start} ne dépend pas de la taille du message et n'est consommé qu'une seule fois (lors de la première connexion), on peut poser $t_{Start} = 0$ pour simplifier le problème bien qu'en réalité la seule relation valide soit:

$$\lim_{n \to \infty} \frac{t_{Start}}{t_{Total}} = 0$$

Si on introduit les notations $t_{delay_{Send}}(n)$ correspondant au temps total nécessaire pour expédier un message de taille n, et $t_{sys_{send}}$ le délai requis par le système pour expédier un message vide, alors:

$$t_{delay_{Send}}(n) = n \times (t_{pack} + t_{send}) + t_{sys_{send}} \tag{2.1}$$

où $n \times t_{pack} \simeq t_{pack}(n)$ *et* $n \times t_{send} \simeq t_{send}(n)$ (e.g. les valeurs croissent linéairement, ce qui est vérifié en [23]). Il apparaît que le temps d'exécution d'un *Send* ou d'un *Receive* ne dépend pas uniquement de l'architecture hôte mais de la librairie de fonctions utilisée (différentes librairies sont présentées en 3.1). En effet, $t_{Sys_{Send}}$ ne dépend pas de la taille des messages (il peut se mesurer en envoyant des messages vides). Il devient négligeable pour des messages de grande taille.

Les auteurs proposent donc de distinguer deux catégories de messages, ceux dont la taille n'excède pas 10 Kbytes d'une part, et d'autre part les autres que nous appellerons messages de grande taille.
D'une façon plus générale, on considère qu'un message est de "grande taille" lorsqu'il dépasse la taille de message standard utilisée par le protocole de transport (ce qui est très variable: 54 octets pour ATM, 8 Koctets pour UDP, etc).

2.1.2 Le débit ou *bandwidth* et surcoût ou *overhead*

Définition 4 *Le débit correspond au taux auquel les primitives de communication peuvent délivrer un message de taille n, c'est-à-dire le ratio taille du message par temps de transfert.*

Nous noterons le débit ρ.

$$\boxed{\rho(n) = \frac{n}{t_{A \to B}}} \tag{2.2}$$

Et pour les messages de grande taille, ρ devient:

[1] eXternal Data Representation

$$\boxed{\rho(n) = \frac{1}{\Gamma_{A.B}(n) + \frac{t_{net}(n)}{n}}} \tag{2.3}$$

avec

$$\Gamma_{A.B}(n) = \underbrace{t_{pack_A}(n) + t_{unpack_B}(n)}_{Support\ hétérogène} + \underbrace{t_{send_A}(n) + t_{recv_B}(n)}_{Envoi\ standard} \tag{2.4}$$

$\Gamma_{A.B}(n)$ correspond à l'*overhead.*

Définition 5 *L'overhead est le surcoût de temps engendré par les librairies de communication.*

Il importe cependant de différencier les débits théoriques ($\frac{n}{t_{net}(n)}$ et effectifs. Le débit théorique est une caractéristique du média utilisé (en particulier, A. Agarwal estime que les progrès réalisés au niveau des commutateurs ou *switchs* tandis que la vitesse de propagation dans les liens reste sensiblement constante ont fait de celle-ci un facteur limitant [24]), mais il n'est jamais atteint en pratique (à plus forte raison lorsque les messages sont de petite taille [22]).

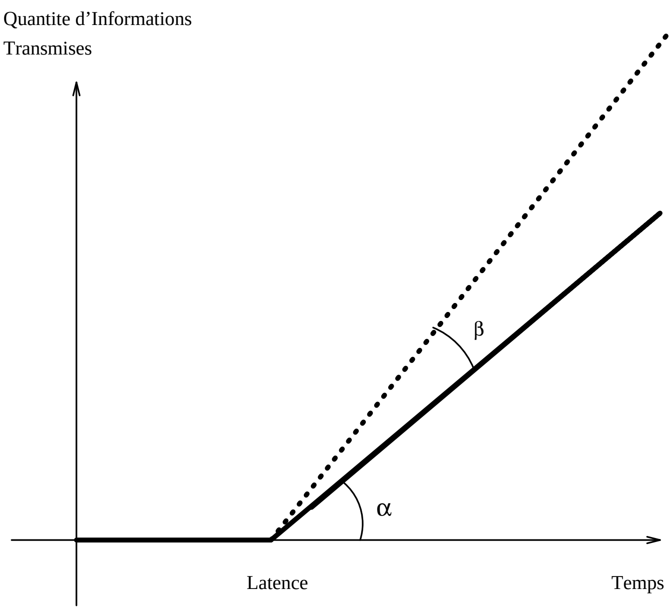


Figure 2.1 : Schématisation de la latence et du Débit

La figure 2.1 schématise le débit, la latence et le sur-coût.

$$\begin{aligned} débit &= tan(\alpha) && (2.5)\\ débit\ théorique &= tan(\alpha + \beta) && (2.6)\\ overhead &= tan(\alpha + \beta) - tan(\alpha) && (2.7) \end{aligned}$$

Une bonne caractéristique pour évaluer les performances reste la taille de messages qui permet d'atteindre la moitié du débit en crête. On pose

$$\boxed{n_{1/2}\ tel\ que:\ \rho(n_{1/2}) = \frac{\rho(n \to \infty)}{2} = \frac{\rho_{max}}{2}}$$

Plus $n_{1/2}$ est faible, plus les performances sont élevées [22].
Le tableau [25] suivant compare plusieurs supports architecturaux:

Linpack	SP2 64	C90 16	T3D 128	SP1 8
$r_\infty(Mflops)$	12.1	13.7	10.7	0.66
n_{max}	27000	10000	24576	10000
$n_{1/2}$	6200	650	3264	1662

Tableau 2.1 : Performances comparée de plusieurs architectures

2.2 Les choix non tranchés

Une gestion efficace des entrées/sorties est une condition nécessaire à de bonnes performances. En effet les ressources partagées (en particulier les liens et les fichiers [26]) risquent toujours de constituer un goulot d'étranglement ou *bottleneck* et donc de réduire *l'extensibilité* du système.

Lorsque toutes les communications sont non bloquantes, les communications et les calculs s'effectuent simultanément; le temps d'exécution est donc égal au maximum du temps de calcul et du temps de communication.

$$t_{TOTAL} = \max(t_{CALCUL}, t_{COMMUNICATION}) \tag{2.8}$$

On distingue donc deux types d'applications:

- Les applications bornées par le calcul ou *Compute-bound*: $t_{CALCUL} > t_{COMMUNICATION}$,
- Les applications bornées par les communications ou *IO-bound*: $t_{CALCUL} < t_{COMMUNICATION}$.

La gestion des ressources de communication est donc particulièrement importante pour les exécutions dites IO-bound.

De plus, certaines ressources ne sont accessibles qu'en exclusion mutuelle (*MUTEX* [26]), d'autres sont accessibles à plusieurs lecteurs simultanément (modèle CREW pour *Concurrent Read, Exclusive Write*). Le conflit qui résulte des demandes de réservations et les échecs/ré-essais qui découlent du mécanisme de prévention d'interblocage se traduisent par une perte de performances.

Afin de résoudre ce problème, différentes solutions ont été proposées.

2.2.1 La topologie du réseau d'interconnexion

La première ressource critique est le réseau d'interconnexion. Un nœud de machine MIMD se compose d'une mémoire locale, d'une unité de calcul et d'un mécanisme de communication.
Le commutateur (ou *switch*)[8] permet de relier les canaux entre eux. Le routeur est chargé d'acheminer les messages du nœud source au nœud cible. Bien que ce travail puisse être laissé à la charge du processeur, il est en général effectué par un composant spécifique, ce qui permet de garantir des communications véritablement non-bloquantes [27]. On considère donc deux entités distinctes: le nœud de calcul (formé de la mémoire locale et du processeur) et le nœud de communication (le routeur et le commutateur).

Afin d'éviter l'engorgement (ou *contention*) des liens, les liaisons point-à-point sont souvent utilisées pour créer des réseaux fortement couplés. La contrepartie d'un couplage fort est le manque d'extensibilité du système.

La figure 2.2 ([28]) indique pour quatre types de réseaux la complexité en temps et en nombre de liens utilisés lors d'une diffusion totale (en considérant que les nœuds peuvent recevoir n messages simultanément).

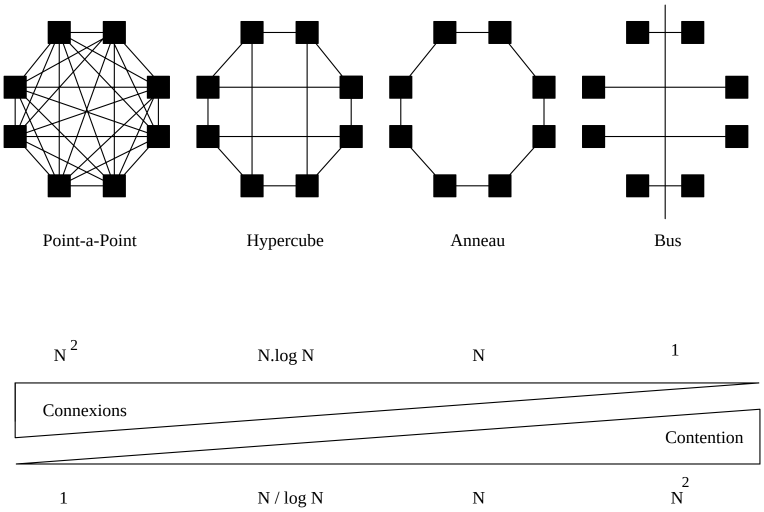


Figure 2.2 : Connectivité des réseaux.

algorithme	Msg Passing System	MFLOPS per node	MFLOPS total	MByte/s per node	Mbyte/s total
FFT	PVM	2.15	1103	0.64	326
	RRMSG	13.55	6938	9.39	4809
	DMSG	19.7	2516*	20.27	10379
SOR	PVM	11.7	5977	21.1	10845
	RRMSG	15.4	7882	47.3	24243
	DMSG	15.5	7939	48.4	24822

Tableau 2.2 : Gain en performances sur le CRAY T3D suivant le modèle de passage de messages

2.2.2 Le protocole de transport: Latence contre débit

Adéquation application - modèle

Le tableau suivant est tiré de l'article [3]. Il compare trois modèles de communications pour deux algorithmes implémentés sur un CRAY T3 D (512 nœuds). Le premier modèle (PVM 3.3 — version optimisée par le co nstructeur) correspond à un échange de type postal. Le second (*Request- Response Message Passing*) est une optimisation du modèle précédent (**??**). Le troisième (*Deposit Message Passing*) utilise des barri ères matérielles pour découpler le contrôle de l'échange de donnés. Les problèmes traités sont de taille 4096×4096. Afin de mieux pouvoir interpréter ces chiffres, les auteurs fournissent les caractéristiques plus détaillées de leur système: Le débit mesuré sur cette machines est de Moctets/s et la latence d'un message est de 2,7 μs (en utilisant les primitives de communication). Les auteurs ont obtenu une latence pure de 1.33 μs en utilisant les mêmes primitives ($<$ *shmem_put* $>$) mais le sur-coût logiciel est de 3.8 μs.

* Ce résultat est surprenant. La valeur attendue se situait autour de 10097 ± 10 (valeur calculée à partie des autres cases du tableau)

Analyse des chiffres

La différence entre les deux algorithmes:

FFT réalise des communications globales denses

SOR correspond à une série de filtres. Les communications ne s'effectuent que avec deux voisins. Les messages sont plus importants que pour FFT.

Réseau	Sur-coût en μs	Temps par octet. en μs	Bande passante en crête Mbits/s
Ethernet	1014	1.80	9.3
ForeNI/Cisco Switch	1449	0.28	74.5
SynOptics NI/Switch	1613	0.35	62.2
Myricom NI/Switch	2236	0.50	64.0

Tableau 2.3 : Performances de différents protocoles

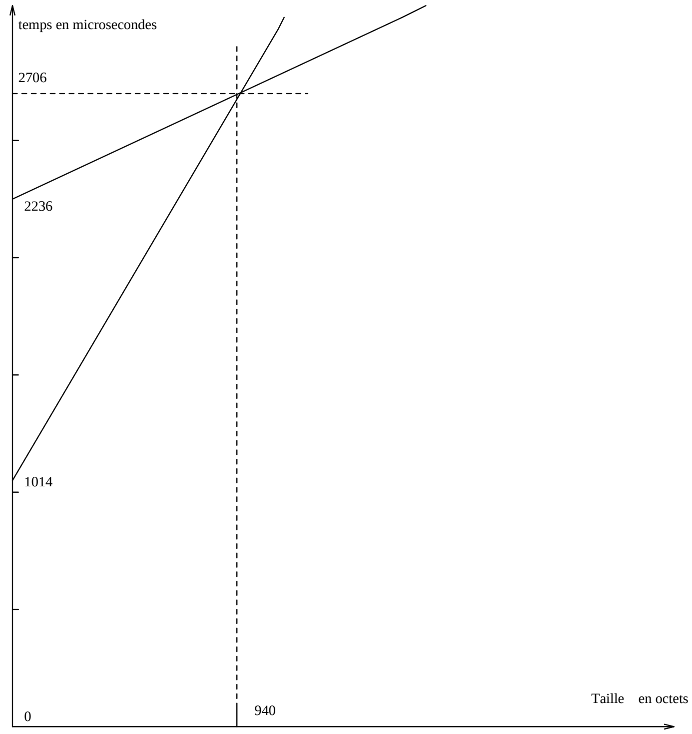
Le gain est plus important pour FFT; l'optimisation concerne donc surtout les messages importants (donc avec un séquencement important) et les communications globales (controle plus important).

Débit ou latence: choix du critère d'optimisation

Par ailleurs, le réseau est lui aussi primordial. le tableau suivant (issu du même article) correspond à des mesures effectuées en utilisant le standard TCP/IP sur des SPARC 10 suivant le modèle de temps fixe auquel s'ajoute un coût par octet.

Note 1: Cette approche ne constitue qu'une approximation. Lorsque un envoi dépasse la taille standart, le tampon ne permet plus de stocker l'ensemble des données: un second espace mémoire sur lequel pointe le tampon est alors créé. On obtient donc des fontions $q(t)$ affines par intervalles.

Note 2: On voit immédiatement que le choix du support dépend de la taille moyenne des messages qui doivent transiter par le réseau. Par exemple, la bande passante de Ethernet est environ sept fois inférieure à celle de Myricom, Ethernet reste cependant plus rapide pour des envois de moins de 940 octets.



2.2.3 La gestion logicielle

La gestion logicielle des communications semble particulièrement importante. En effet, la différence entre le débit théorique et le débit effectif peut souvent être réduit par une gestion efficace des tampons.

Une politique de "couches logicielles" peut conduire à une amélioration sensible des performances. En particulier, la répartition des tâches entre le noyau et l'espace utilisateur est des plus importantes.

Dans [1], J. Bruck, D. Dolev, C-T. Ho, M-C. Roşu et R. Strong se sont intéressés aux communications globales sous MPI (3.1). Ils ont développé un système composé de deux couches principales: la couche MPI-

CCL (qui délivre les fonctionnalité de communications globales de MPI, gère le mise en paquets,...) et un protocole de transport fiable de niveau utilisateur (URTP) qui interface la couche réseau. Un mécanisme d'extension noyau permet à leur système de s'intégrer au système d'exploitation.

Un processus possède donc quatre couches logicielles qui sont représentées par la figure 2.3.
URTP est implémenté comme une combinaison de l'extension noyau et d'une bibliothèque de niveau uti-

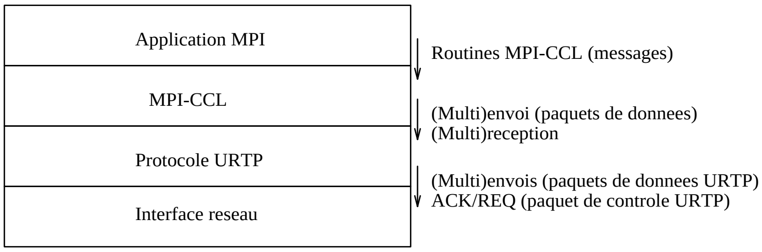


Figure 2.3 : Les quatres couches logicielles.

lisateur contenant la majorité du code. L'extension noyau permet d'éliminer rapidement les messages par lesquels le processeur n'est pas concerné (de façon à réduire le sur-coût coté receveur) et les requêtes qui entraineraient un délai trop important (cette couche peut éventuellement être matérielle).
Notons que tous les tampons utiles se trouvent dans l'espace utilisateur. Lors de la réception d'un message, celui-ci est copié dans un tampon noyau puis de ce tampon dans un tampon URTP. Un pointeur sur ce tampon est alors retourné à la couche MPI-CCL. Le transfert entre le noyau et le niveau utilisateur s'opère le plus rapidement possible de manière à réduire le risque de perte de paquets due au manque de tampons disponibles. Par ailleurs, afin de réduire le nombre de paquets de contrôle, tout acquittement d'un paquet acquitte implicitement les paquets antérieurs. Ici la solution retenue pour la détection de perte de données est basée sur des compteurs d'envois/réception pour chaque paire émetteur/récepteur. Par ailleurs une contrainte de type FIFO est imposée.

Sur la base d'un mécanisme simple, implémenté sur le média le plus lent, les auteurs ont mesuré un gain en performances de l'ordre de 2,5 par rapport à une implémentation de diffusion sur ATM (publiée en novembre 1994). La bande passante utile est environ 8,8 Mbits par seconde, à mettre en relation avec les 10 Mbits théoriques.

La répartition des tâches au niveau système semble donc être une voie d'optimisation particulièrement intéressante.

2.2.4 La gestion matérielle

Le matériel est un facteur important pour les performances: choix du co-processeur, possibilité de DMA (Direct Memory Access),... Ceci explique que de nombreux prototypage soient en cours pour déterminer les solutions optimales. Le tableau extrait de [29] compare les performances de différents systèmes pour une implémentation de GAM (Generic Active Messages Layer).

Par ailleurs S. Packin, M. Lauria et A. Chien se sont intéressés aux messages de taille faible (une fonction d'émission spécifique existe pour les messages de quatre mots) [22] — les mécanismes qu'ils décrivent nous paraissent intéressants dans la mesure où ils peuvent s'appliquer de façon spécifique aux messages de contrôle. Trois critères décisifs pour les performances sont présentés:

- la répartition des tâches entre le processeur et le co-processeur réseau,
- la gestion des entrées/sorties sur le bus,
- la gestion des tampons.

Machine	interface réseau	sur-coût + latence	Débit
Calculateurs paralléles			
Paragon	processeur dédié symétrique	5.1 + 5.6 = 10.7	140 Mb/s
Meiko	CoProcesseur	1.1 + 11.4 =12.5	39 Mb/s
Réseaux de station			
Myrinet	CoProcesseur	inférieur à 20	supérieur à 20 Mb/s
HP/Medusa	interface FDDI sur bus Graphique	10.1 + 10 = 14.5	12 Mb/s
SS10/SAHI	ATM interface sur Sbus	11+200 = 211	12 Mb/s

Tableau 2.4 : Importance de l'interface réseau

Les auteurs proposent d'allouer la responsabilité du maximum de tâches au processeur , de façon à décharger le co-processeur de toute autre activité que la gestion des canaux du bus.

Si l'interface réseau peut accéder directement à la mémoire (*DMA* pour *Direct Memory Access*), alors le bus d'entrées/sorties peut être utilisé de manière asymétrique, le processeur déplaçant les données vers le réseau et exploitant les DMA pour les rapatrier.

En effet, laisser à la charge du processeur le transfert des données vers le réseau réduit la latence, et l'utilisation des DMA (initiée par le co-processeur) pour consommer les messages qui arrivent maximise le débit pour un coût minime en latence.

La gestion des tampons est alors très simple dans le co-processeur, lequel partage les files d'attente avec le processeur. Cette simplicité minimise par ailleurs le sur-coût induit par la gestion logicielle, et libère l'interface réseau qui reste dédiée aux liens. Enfin le partage des files d'attente permet de regrouper plusieurs messages de taille faible en un message de taille plus importante pour gagner en performances.

Le rôle des tampons se borne alors à accorder les vitesses du processeur, du co-processeur réseau et des canaux.

2.2.5 Le traitement des requêtes

Définition 6 (Requêtes) *(a): Une requête est dite séquentielle si le décalage par rapport au début du fichier (offset) est plus important que celui de la dernière requête issue de ce nœud.*
(b): Une requête est dite consécutive si elle est séquentielle et qu'elle débute où a cessé la requête précédente.
(c): Un intervalle est la distance entre la fin d'une requête et le début de la suivante

L'ordonnancement consiste à définir un ordre sur des requêtes concurrentes (concurrentes signifiant ici temporellement incompatibles). Pour ce faire, diverses techniques sont utilisées:

Premier arrivé Premier servi ou *First-Come-First-Served* : Les demandes sont traitées dans l'ordre d'arrivée.

Plus court d'abord ou *Shortest Job First (SJF)* : Les demandes les moins consommatrices en temps sont prioritaires. Autrement dit il existe un risque de famine pour les “gros consommateurs”.

Répartition équitable ou *Round Robin* : Les processus bénéficient d'une fenêtre temporelle à tour de rôle. L'inconvénient de cette méthode réside dans le temps de réponse moyen élevé.

Attente active ou *Busy Wait* : On cherche à minimiser le chemin critique.

En revanche, lorsque les requêtes ne sont pas concurrentes, il est alors possible de regrouper les demandes, ce qui implique la mise en œuvre d'un mécanisme de prédiction. Si les applications scientifiques sont souvent

très régulières, et donc prévisibles, il en va autrement pour les applications "ordinaires". De plus les applications scientifiques ne nécessitent en général que peu d'accès mais ceux-ci sont importants; en revanche de nombreuses applications réclament des milliers voire des millions de petites requêtes. Cependant dans [30], Nils Nieuwejaar et David Kotz montrent que les accès au disque requis par de nombreuses applications, s'ils ne sont pas consécutifs, n'en demeurent pas moins très réguliers. Le gain possible en tenant compte des propriétés des demandes est alors conséquent.
Dans [30], les auteurs introduisent des notions permettant de décrire ces accès, et fournissent des extensions pour l'interface standard de façon à augmenter la quantité d'informations disponibles pour le système de fichiers bas-niveau. De cette manière il est possible de faire des demandes d'entrées/sorties à un plus haut niveau sémantique. Les auteurs affirment avoir ainsi réduit de 90% le nombre de requêtes. Les trois modes d'accès qu'ils ont considérés sont:

Simple-strided access pattern Chaque requête a la même taille et l'intervalle est une constante.

Nested-strided access pattern plusieurs *simple-strided access patterns* régulièrement espacés.

Nested-batched access pattern plusieurs *nested-strided access patterns* régulièrement espacés.

On voit immédiatement les trois niveaux d'abstraction qui permettent de formaliser finement l'échange.

Récapitulatif

Nous pouvons donc récapituler les principaux critères de performances de la façon suivante:

- Limiter (ou éliminer) les copies intermédiaires — 1.2.3
- Placer l'interface logicielle dans l'espace utilisateur — 2.2.3
- Implanter des mécanismes de protection matériels
- Interface matérielle "proche de la mémoire" — 2.2.4
- Interface matérielle active — 2.2.4
- Commutateurs rapides et réseau haut-débit
- Bibliothèques de passage de messages performantes — 3

Conclusion

Nous avons introduit deux notions qui permettent de définir les performances en communication: le débit et la latence.
Le débit peut être augmenté en réduisant $\Gamma_{A.B}(n)$, le sur-coût entrainé par les bibliothèques de passage de messages (3.3), ou en augmentant le débit théorique du média utilisé ($\frac{n}{t_{net}(n)}$).
La latence, elle, peut bénéficier d'une réduction de chacun des temps intermédiaires qui la composent (2.1.1). En particulier $t_{Pack}(n)$ *et* $t_{Unpack}(n)$ peuvent être réduits efficacement — comme c'est le cas pour LAM (3.3) —, $t_{Send}(n)$ dépend des mécanismes matériels mis en œuvre (2.2.4) et $t_{Receive}$ est minimal dans le cas des messages actifs.
Si les nombreuses optimisations qui existent sont parfois incompatibles (un *overhead* plus important peut permettre un gain en débit mais pénalise les messages de taille faible), lorsque l'on différencie le traitement des messages de contrôle de celui des messages de données, on peut espérer tirer partie des diverses améliorations présentées dans ce chapitre.

Chapitre 3

Deux bibliothèques de passage de messages: PVM et MPI

Introduction

Lorsque sont apparues les premières architectures parallèles, l'idée communément admise consistait à concevoir des machines multiprocesseurs. Leur coût de fabrication élevé a entrainé l'apparition des réseaux de stations de travail qui simulent une machine parallèle (on parle d'*hypercomputing*). Le protocole le plus utilisé en vue de réaliser cette émulation reste PVM (pour *Parallel Virtual Machine*). La plupart des machines parallèles actuelles proposent un modèle de programmation par échange explicite de messages. L'utilisation de bibliothèques de communication permet de réaliser des applications portables (e.g. ne dépendant pas de l'architecture cible). Cette constatation a été une des principales raisons de l'apparition de PVM qui est aujourd'hui la bibliothèque la plus utilisée. Pourtant une volonté de standardisation et de participation réelle des acteurs industriels a conduit à la mise en œuvre de MPI (pour *Message Passing Interface*).

Nous nous proposons dans cette partie de présenter les fonctionnalités de PVM et MPI [31].

Définition 7 (Bibliothèque de passage de messages) *: Une bibliothèque est un ensemble de fonctions utilisables par le programmeur pour réaliser un envoi ou une réception de message.*

Définition 8 (Machine Virtuelle) *: Une machine virtuelle est une abstraction des fonctionnalités d'une machine.*

3.1 PVM

PVM [1] a permis de simuler des machines parallèles classiques sur des réseaux de stations de travail. Ainsi on obtient une machine évolutive, puisque modulaire, à un coût sensiblement inférieur à celui des super-calculateurs.
PVM est l'un des environnements de programmation parallèle les plus utilisés; il propose des primitives de communication, de synchronisation par barrière et de gestion de processus [32]. Le placement et l'activation des processus sont explicites ou gérés par le système, ce qui procure une grande souplesse d'utilisation.
Les principales caractéristiques de PVM sont présentées par Christophe Calvin dans [32].

Une part importante du logiciel est consacrée à la gestion des processus, des tâches et des signaux systèmes; cette gestion est effectuée par l'intermédiaire d'un processus démon lancé lors de l'initialisation (qui gère entre autres le contrôle de flux).
Tous les processus sont munis d'un identificateur unique affecté par le démon pour l'exécution de la machine

[1] PVM a été développé à l'université du Tennessee et á *l'Oack Ridge National Laboratory*

virtuelle. L'utilisateur peut également définir des noms, des groupes de processus. Chacun d'entre eux est alors identifié par son numéro de groupe et son rang dans le groupe.
La gestion des processus est dynamique: toutes les fonctions utilisées par l'interface utilisateur peuvent également être appelées par un programme.

PVM est par ailleurs à même de détecter la défaillance d'un nœud de la machine virtuelle (auquel cas celui-ci est supprimé de la liste des machines utilisées). La gestion dynamique des processus permet alors de gérer une tolérance aux pannes (qui reste à la charge du programmeur).

Les communications en PVM

Les communications s'effectuent par envois et réceptions de tampons (espaces mémoire). Les avantages de cette méthode sont nombreux: elle facilite l'échange de données structurées, elle permet de découper facilement un message (d'où la possibilité de mettre en œuvre des mécanismes de communication rapides). Le codage des messages dans le cas d'un réseau hétérogène est lui aussi simplifié.
Une communication est toujours structurée comme suit:

- Envoi
  1. Allocation du buffer d'envoi,
  2. Empaquetage du message,
  3. Envoi du message.
- Reception
  1. Reception du message,
  2. Dépaquetage du message.

Le reséquencement est ici basé sur des étiquettes.

Performances

Le tableau suivant est extrait d'un article de B.K. Schmidt et V.S Sunderam [33]. Les auteurs cherchent à déterminer les facteurs limitant les performances en communication.
L'équation 1 peut s'écrire sous la forme:

$$T_{Comm} = \alpha \ + \ \beta \times n$$

Les résultats expérimentaux obtenus par les auteurs sur des stations SPARC (à 12 MIPS[2]) connectées par un réseau Ethernet à dix Mbits sont les suivants:

$$\begin{aligned}
\alpha_{Pack} &= 0 \ ms \\
\beta_{Pack} &= 0.00017 \ ms/octet \\
T_{Pack} &= 0.00017 \times N \ ms \\
T_{Send-Datragramm} &= 4.527^{*} \ + \ 0.0025 \ \times N \ ms \\
T_{Send-Stream} &= 1.661^{*} \ + \ 0.0014 \ \times \ N \ ms \\
T_{Communication} &= T_{Pack} \ + \ T_{Send} \\
&\text{d'où:} \\
T_{comm-Datagramm} &= 4.527 \ + \ 0.00267 \ \times \ N \ ms \\
T_{Comm-Stream} &= 1.661 \ + \ 0.00157 \ \times \ N \ ms
\end{aligned}$$

Les nombres marqués d'un * correspondent à t_{Start} défini précédemment.

[2]Million d'Instructions Par Seconde

Taille en octets	1	32	64	512	1k	4k	16k	128k
msg packing	0.09382	0.088342	0.095959	0.0155628	0.255457	0.670895	2.212407	22.738652
stream msg	1.59715	1.47073	1.54337	1.61310	1.98495	10.20227	20.08871	184.47232
datagram msg	4.972248	4.2199	3.85209	4.42586	5.34203	14.61778	45.43572	332.53938

Tableau 3.1 : Performances en PVM suivant le type d'envoi

3.2 MPI

MPI comme PVM fonctionne sur des machines massivement parallèles (MPPs pour *Massively-Parallel Machines*) ou faiblement couplées et sur des réseaux de stations (*NOW*) homogènes ou hétérogènes [34] [35] mais ces deux bibliothèques ne possèdent pas les mêmes fonctionnalités [36] [32].
MPI [1] est le fruit d'une volonté collective d'intégration des meilleures caractéristiques des systèmes de passage de messages existants au sein d'une même structure standardisée [34]. Ceci explique que les innovations de MPI soient des généralisations de fonctions présentes dans d'autres bibliothèques:

Construction de types : La généralisation de la description des tampons permet de construire de nouveaux types de données pour décrire des structures complexes ou envoyer des mémoires non contiguës en mémoire.

Notion de contexte : Introduction de contexte lié à une famille de messages, ce qui constitue une généralisation de la notion de type du message.

Notions de Groupe et de Communicateur : Extension de la notion d'identificateur de processus (le communicateur sert à définir la portée de l'opération de communication).

Les mécanismes de gestion d'entrées/sorties de MPI sont décrits en [37].

Dans [36], les auteurs exposent plusieurs raisons de préférer MPI à PVM, entre autres:

- Plusieurs implémentations de qualité sont disponibles (3.3).
- MPI est totalement portable, spécifié de manière formelle et constitue un standard.
- Les communications sont complètement asynchrones.
- Les groupes sont robustes et efficaces.
- MPI permet de programmer efficacement les machines massivement parallèles comme les réseaux de stations.

3.3 Performances

Les trois principales implémentations de MPI sont à ce jour LAM [3], MPICH [4] et CHIMP [5]. Dans [31], les auteurs recommandent l'usage de MPI pour des multiprocesseurs et de PVM pour des réseaux de stations de travail hétérogènes. De fait certains résultats (obtenus sur des réseaux) sont exposés dans [23]. Il apparaît que PVM est le système le plus performant, que MPICH a des performances à peu près semblables mais que LAM est sensiblement moins efficace.

Différents résultats issus de [23] sont regroupés dans le tableau suivant (cas de réseau hétérogène, les messages ont une taille de 120 Koctets).

[3] (Ohio state), ftp site: tbag.osc.edu
[4] (Aragone-Mississippi state), ftp site: info.mcs.anl.gov
[5] (University of Edinburgh), ftp site: ftp.epcc.ed.ac.uk

Librairie utilisée	numéro de version	$t_{A \to B}$ en ms	% de $t_{A \to B}$	débit en Mbits/s	sur-coût em microsecondes	sur-coût XDR
PVM	3.3.7	210	46	4.6	0.80	40%
MPICH	1.0.8	265	57	3.6	1.20	40%
LAM	5.2	235	51	4.0	1.00	-
CHIMP	2.1	218	47	4.4	0.85	25%

Tableau 3.2 : Performances comparées de PVM et MPI

Notons que LAM, contrairement aux autres librairies testées, n'utilise pas XDR mais une représentation externe particulière. Les performances de LAM se dégradent moins que celles des autres librairies lorsque le réseau est hétérogène.

PVM semble donc globalement plus performant que MPI (même si CHMIP a des performances presque semblables). Est-ce pour autant un choix d'avenir? La réponse ne viendra pas des constructeurs (pour IBM l'effort d'implémentation de PVM et MPI est fait de manière équivalente [31]); ceux-ci attendent une demande explicite de la part des utilisateurs pour faire un choix entre les deux bibliothèques. A ce jour, il existe une implémentation de PVM sur MPI et vice-versa. Les utilisateurs peuvent donc de toute façon développer leurs programmes à l'aide de l'une ou l'autre bibliothèque puis envisager de migrer vers la seconde.

Conclusion

Les premières machines parallèles à être apparues sont les supercalculateurs. Le modèle SIMD alors utilisé tend aujourd'hui à disparaître pour faire place aux machines MIMD. Cependant les machines parallèles dans leur ensemble se trouvent désormais remises en question. L'apparition des réseaux haut-débit a, en effet, permis l'émergence de nouvelles architectures: des stations de travail interconnectées. Si celles-ci présentent nombre d'avantages (coût restreint, modularité, intégration rapide des progrès réalisés par les machines séquentielles,...), en revanche les performances de ces réseaux de stations de travail sont en grande partie bornées par les sur-coûts induits par les mécanismes de communication. Il est donc vital de réduire au minimum ces délais supplémentaires afin de ne pas dégrader de façon par trop importante les performances globales. Pour ce faire, différentes techniques ont été présentées dans cette étude bibliographique. Nous retiendrons principalement le mécanisme des écritures distantes basé sur l'usage des messages actifs qui permet de disjoindre le contrôle et l'échange de données. Cette dernière caractéristique nous semble particulièrement intéressante dans la mesure où elle offre la possibilité d'utiliser deux liens différents pour les deux types de messages. Ceux-ci ayant des propriétés très différentes (en particulier la taille des message et la nature du traitement à effectuer), les optimisations possibles semblent alors plus nombreuses. On favoriserait ainsi le débit pour les messages de données et la latence pour les messages de contrôle.

Références

[1] D. Dolev J. Bruck, C-T. Ho, M-C. Roşu, and R. Strong. Efficient message passing interface (MPI) for parallel computing on clusters of worstations. In *ACM SPAD*, 1995.

[2] Daniel Etiemble. L' évolution des machines parallèles et massivement parallèles. *Technique et science informatiques*, 14(5):529–550, 1995.

[3] J. Strichnoth T. Stricker, D. O'Hallaron, S. Hinrichs, and T. Gross. Decoupling synchronization and data transfer in message passing systems of parallel computers. *ACM*, 1995.

[4] Charles L. Seitz. The cosmic cube. *Communications of the ACM*, 28(1):22–33, January 1985.

[5] A. Agarwal, J. Kubiatowicz, D. Kranz, B-H. Lim, D. Yeung, G. D'Souza, and M. Parkin. Sparcle: An evolutionary processor design for large-scale multiprocessors. postscript version, March 1993.

[6] Andrew A.Chien Vijay Karamcheti. A comparison of achitectural support for messaging in the tmc cm-5, the cray t3d. *ACM /ICA'95 SAnta Margherita*, 1995.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), July 1978.

[8] M. Katevenist, P. Vatsolaki, A. Efthymiou, and M. Stratakis. Vc-level flow control and shared buffering in the telegraphos switch. In *IEEE Hot Interconnects III Symposium Proceedings, Stanford, Ca.*, August 1995.

[9] Michel Raynal. Cours DEA option COSY IFSIC Renens, 1995.

[10] J. Dongarra, A. Geist, R. Manchek, and V. Sunderam. Integrated PVM framework supports heterogeneous network computing. *Computers in Physics*, 7(2):166–75, April 1993.

[11] Manolis G.H. Katevenis Evangelos P. Markatos. Telegraphos: High-performance networking for parallel proceding on workstation clusters. In *Procedings of the HPCA-2, San Diego*, 1995.

[12] T. Von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: a mechanism for integrated communication, computation. ACM Trans, 1992. University of California, Berkeley.

[13] A. Mainwaring and D. Culler. *Active Messages: Organization and Applications Programming Interface. Interface Specification Version 2.0.* Computer Science Division, University of California at Berkeley, November 1995.

[14] D. Krantz, K. Johson, and A. Agarwal. Integrating message passing and shared-memory: Early experience. postscript ftp.cag.lcs.mit.edu. MIT Cambridge MA 02139.

[15] Dept. of computer science, City University, London. *Experiences with Distributed Shared Memory*, 1994.

[16] G. Cabillib, T. Priol, and I. Puaut. MYOAN: an implementation of the KOAN Shared Virtual Memory on the Intel Paragon. Technical Report 812, IRISA, April 1994.

[17] I. Watson and A. Rawsthorne. Decoupled pre-fetching for distributed shared memory. University of Manchester.

[18] J. Kubiatowicz and A. Agarwal. Anatomy of a message in the Alewife multiprocessor. *Appeared in the 7th ACM International Conference on Supercomputing Tokyo Japan*, 1993. MIT.

[19] A. Agarwal, R. Bianchibi, D. Choiken, K. L. Johson, D. Kranz, J. Kubiatowicz, B. Lim, K. Mackenzie, and D. Yeung. The mit Alewife machine: Architecture and performance. In *ISCA*, 1995.

[20] D. B. Gustavson and Q. Li. Local-area multiprocessor: the scalable coherent interface. Postscipt version, 1995. SCIzzL, Santa Clara University, California.

[21] H. Bryhni and B. Wu. Initial studies of sci lan topologies for local area clustering. Prepared for the First International Workshop on SCI-based Low-Cost/Hight-Performance Computing, Santa Clara University.

[22] ACM, editor. *High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet*, http://www-csag.cs.uinc.edu/projects/communication/sw-messaging.html, 1995.

[23] J. Guyard E. Dillon, C. Gamboa Dos Santos. Homogeneous, heterogeneous networks of workstations: Message passing overhead. Postscipt version, June 1995. Projet RESEDAS INRIA/CRIN.

[24] A. Agarwal. Limits on interconnection network performance. postscript version. MIT.

[25] Frank Cappello. Les réseuax de sations de travail. Alternative, avenir ou simple état transitoire des Architectures Parallèles Extensibles? Technical report, CNRS-LRI URA 410, Université Paris Sud, Equipe Architectures Parallèles, 1995.

[26] S. A. Moyer and V. S. Sunderam. Scalable concurrency control for parallel file systems. Technical report, Dept. of Math and Computer Science, Emory University, Atlanta, June 1995. CSTR-950202.

[27] Christophe Calvin. Minimisation du sur-coût des communications dans la parallélisation des algorithmes numériques. Master's thesis, Institut National Polytechnique de Grenoble, July 1995.

[28] C. Germain-Renaud and J-P. Sansonnet. *Les ordinateurs massivement parallèles.* Acquis avancés de l'informatique, 1991.

[29] Berkeley university, editor. *TITAN: A next-generation infrastructure for integrating computing and communication*, http://www.cs/projects/titan/index.html, Technical Progress Report (8/15/94 - 7/31/95).

[30] N. Nieuwejaar and D. Kotz. Low-level interfaces for high-level parallel i/o. In *Appeared at the Workshop for I/O in Parallel and Distributed Systems at IPPS'95*, pages 47–62, 1995.

[31] Christophe Calvin and Laurent Colombet. Le futur standard de bibliotheque de communication par échange de messages: PVM ou MPI. *Calculateurs parallèles*, 7(4):415–419, 1995.

[32] C. Calvin and L. Colombet. Introduction au modèle de programmation par processus communiquants, deux exemples: PVM et MPI. Technical Report 12, LMC-IMAG/APACHE, July 1994.

[33] B. K. Schmidt and V.S. Sunderam. Empirical analysis of overheads in cluster environments. In G. C. Fox, editor, *Concurrency: Practice and Experience*, volume 6, pages 1–32. Wiley, February 1994.

[34] S. W. Otto J. J. Dongarra, M. Snir, and D. Walter. An introduction to the mpi standard. Postscript version, January 1995.

[35] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT Press, 1994.

[36] LAM Project. Ohio Supercomputer Center. *Ten Reasons to Prefer MPI Over PVM*, 1995.

[37] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J-P. Prost, M. Snir, B. Traversat, and P. Wong. Overwiew of the mpi-io parallel i/o interface. In *Third Workshop on I/O in Parallel and Distributed Systems, IPPS'95, Santa Barbara*, http://lovelace.nas.nasa.gov/MPI-IO/, April 1995.

[38] T. Stricker and T. Gross. Optimizing Memory Performance for Communication in Parallel Computers. *In Proc. 22nd Intl. Symposium on Computer Architecture, Santa Margerita*, pages 308–319, June 1995. new.

[39] S. A. Moyer and V. S. Sunderam. Parallel i/o as a parallel application. Technical report, Dept. of Math and Computer Science, Emory University, Atlanta, December 1994. Technical report CSTR-941101.

[40] S. Zhou Z. lin. Parallelizing i/o intensive applications for a worstations cluster: a case study. *Computer Architecture News*, 21(5):15–22, December 1993.

[41] Manolis Katevenis. Telegraphos: High-speed communication architecture for parallel, distributed computers systems. Technical report, Institute of Computer Science (ICS), 1994.

[42] S Casselman, M. Thorburg, and J. Schewel. Harware Object Programming on the EVC1: a Reconfigurable Computer.

[43] Ralph D. Wittig. OneChip: An FPGA Processor With Reconfigurable Logic. Master's thesis, Ralph D. Wittig, 1995.

[44] Sriram Rajamani and Pramod Viswanath. A Quantitative [a]nalysis of Processor - Programmable Logic Interface. Technical report, Departement of EECS, University of California, Berkeley, 94.

[45] C-C. Yeh, C-H. Wu, and J-Y. Juang. Desing and Implementation of a Multicomputer Interconnection Network Using FPGAs. In IEEE, editor, *IEEE Symposium on FPGAs for CUSTOM COMPUTING MACHINES*, pages 56–60, April 1995.

[46] TriQuint semiconductor, 2300 Owen Street Santa Clara CA 95054. *Fibre Channel Products.*

[47] S. Casselman. Virtual Computing and the Virtual Computer. In *IEEE Workshop on FPGAs for Custom Computing Machines, Napa California*, April 1993.

[48] S. Casselman. Transformable computer. In *8 th International Parallel Processing Symposium , Cancun Mexico.* Sponsored by IEEE Computer Society, April 1994.