RAPPORT DE STAGE DEA ID

Etudes de performance sur une machine parallèle de type « Grappe de PCs » : La machine MPC

Laurent LECHEVALLIER

Mars-Aout 2001

Table des matières

Chapitre 1

Introduction et contexte du stage

I Le projet MPC

I.1 Le projet MPC au sein du LP6

Mon stage s'est effectué dans le laboratoire de recherche du département *ASIM* (Architecture des systèmes intégrés et micro-électronique) du *LIP6* (Laboratoire d'Informatique de Paris 6). Au sein de l'équipe de recherche dont le travail repose sur trois projets :

- La machine MPC
- Projet d'indexation multimédia
- Projet CAO de circuits et systèmes (Alliance)

Le LIP6 est composé de 9 thèmes de recherche dont les activités recouvrent une large part de l'informatique. Outre leurs activités propres, ces thèmes collaborent dans diverses actions transversales et sont engagés dans de nombreux projets en partenariat avec des entreprises.

Le projet MPC est l'un des huit projets transversaux qui déterminent la politique scientifique du LIP6. L'idée générale est d'aider certains demandeurs de puissance de calcul du LIP6 à mener des expériences sur la machine MPC développée au laboratoire dans le thème ASIM. Mais surtout, le but du projet MPC est la conception d'une machine parallèle performante et à faible coût. La contribution du laboratoire s'est traduite par l'acquisition d'un réseau à 4 nœuds. Ceci est suffisant pour mettre au point des expériences mais il serait bon d'avoir au moins huit nœuds pour obtenir des résultats significatifs.

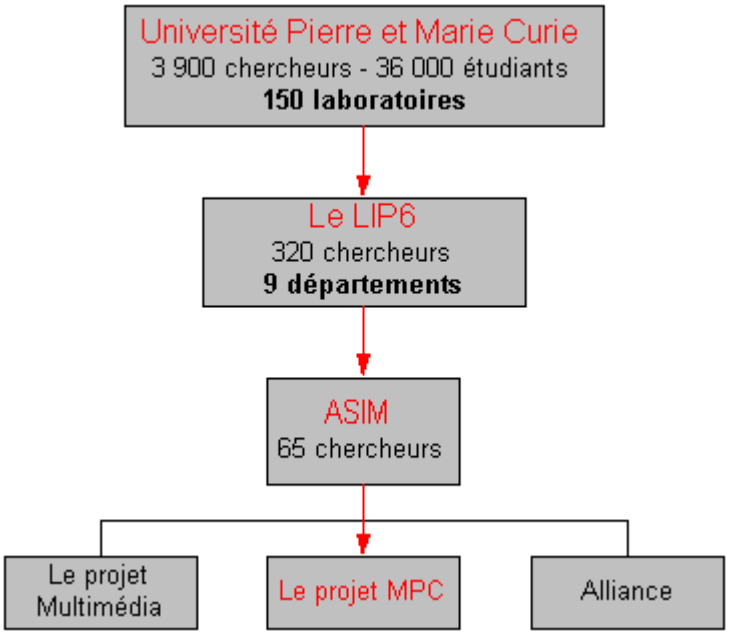


Figure 1 : De l'UPMCP6 au projet MPC

I.2 Pourquoi le projet MPC ?

Les années 1990 ont vu l'émergence des réseaux locaux haut-débit. Citons par exemple, ATM, Ethernet100Mb/s, Ethernet Giga-bit, Myrinet, SCI. Ces progrès sont la conséquence logique de l'avancement de la technologie et en particulier d'une augmentation de la densité d'intégration des composants électroniques. La latence de ces réseaux est de l'ordre de la dizaine de micro-secondes et le débit dépasse souvent le Giga-bit.
Une conséquence directe de cette évolution est l'apparition d'un nouveau type d'architectures parallèles à base d'un réseau de stations. En effet, les réseaux d'interconnexion à haute performance permettent à différents processeurs de collaborer grâce à l'échange de données entre les différents nœuds de calcul. Le rapport coût/performance de ces nouvelles machines parallèles s'approche de celui des calculateurs parallèles classiques.

Le projet MPC (Multi-PC), démarré en janvier 1995 sous la responsabilité d'Alain GREINER, s'intègre complètement dans cette évolution. Il a d'abord consisté, à partir de 1993, à développer une technologie d'interconnexion haute performance. Il s'agit de la technologie HSL qui est devenue le standard IEEE 1355. Cette technologie s'appuie sur deux composants VLSI développés au département ASIM du LIP6 : PCI-DDC et RCUBE
L'objectif du projet MPC vise à la réalisation d'une machine parallèle performante à faible coût, de type grappe de PCs, basée sur un réseau d'interconnexion utilisant la technologie HSL. Le matériel spécifique se limite à une carte d'extension réseau HSL. La cellule processeur est une carte mère standard pour PC possédant un bus PCI.

Les projets comparables au projet MPC sont des projets industriels : SCI, Myrinet et Memory Channel. SCI est une norme (standard IEEE 1596) qui s'appuie sur la connexion de plusieurs anneaux par des passerelles. Myrinet est un réseau qui utilise des liaisons point à points à 1 Gb/s. A la différence du réseau HSL, Myrinet utilise des liaisons parallèles. De plus, la fonction de routage est centralisée et nécessite un boîtier séparé alors que chaque nœud du réseau HSL possède un ou plusieurs routeurs RCUBE. Myrinet utilise également des cartes d'interface PCI. La densité d'intégration des composants électroniques utilisés par la technologie HSL est plus importante que celle de la technologie Myrinet. Enfin, Memory Channel est un réseau basé sur une carte PCI, des liaisons point à point et une fonction de routage centralisée.

II Le sujet du stage

II.1 Objectif

L'objectif du stage est d'évaluer les performances de la machine MPC, en particulier au niveau de l'environnement de programmation par passage de message MPI. Les applications choisies sont des benchmarks considérés comme des standards dans le monde parallèle. Il s'agit tout d'abord de la résolution de l'équation de Laplace. Puis de l'évaluation de la machine MPC avec les NAS Parallèle Benchmarks.

II.2 Description

Les nœuds de calcul de la machines MPC sont interconnectés non seulement par les couches de communications rapide MPC mais aussi par un réseau de contrôle Ethernet 100 Mb/s.

L'environnement de programmation par passage de message MPI a été porté sur la machine MPC. Cependant, aucune véritable application n'avait encore été testée sur cette plate-forme. Il s'agit donc d'abord de faire des séries de tests afin de comparer les performances de communication de MPI-MPC avec un MPI standard sur Ethernet. Ces tests serviront aussi pour analyser les différentes caractéristiques de l'implémentation MPI sur MPC.

Dans un second temps, des mesures et des évaluations seront faites à partir de la parallélisation de l'équation de Laplace selon deux méthodes et deux modes de communications différents.

Enfin, l'évaluation des performances de la machine MPC avec les NAS Parallèle Benchmarks permettra d'une part de valider le portage de MPI sur MPC, et d'autre part, de situer la machine MPC par rapport aux autres machines parallèles.

II.3 Responsables

Directeur de stage : Professeur Alain Greiner (LIP6)
Encadrant de stage : Olivier Gluck (LIP6)

Chapitre 2 : Description de la machine MPC

Ce chapitre présente la machine MPC. Il se décompose en deux parties : la présentation de l'architecture matérielle puis la présentation des couches basse de communication qui s'appuient sur le principe de l'écriture distante.

I Architecture matérielle

La machine MPC du LIP6 est une machine parallèle de type « grappe de PCs ». Elle est constituée d'un ensemble de PCs standard interconnectés par un réseau à très haut débit dont la technologie a été développée par le laboratoire LIP6. Elle est composée de 4 nœuds de calcul Bi-Pentium haut de gamme et d'une console pour améliorer l'exploitation de la machine (cf. Figure 2).

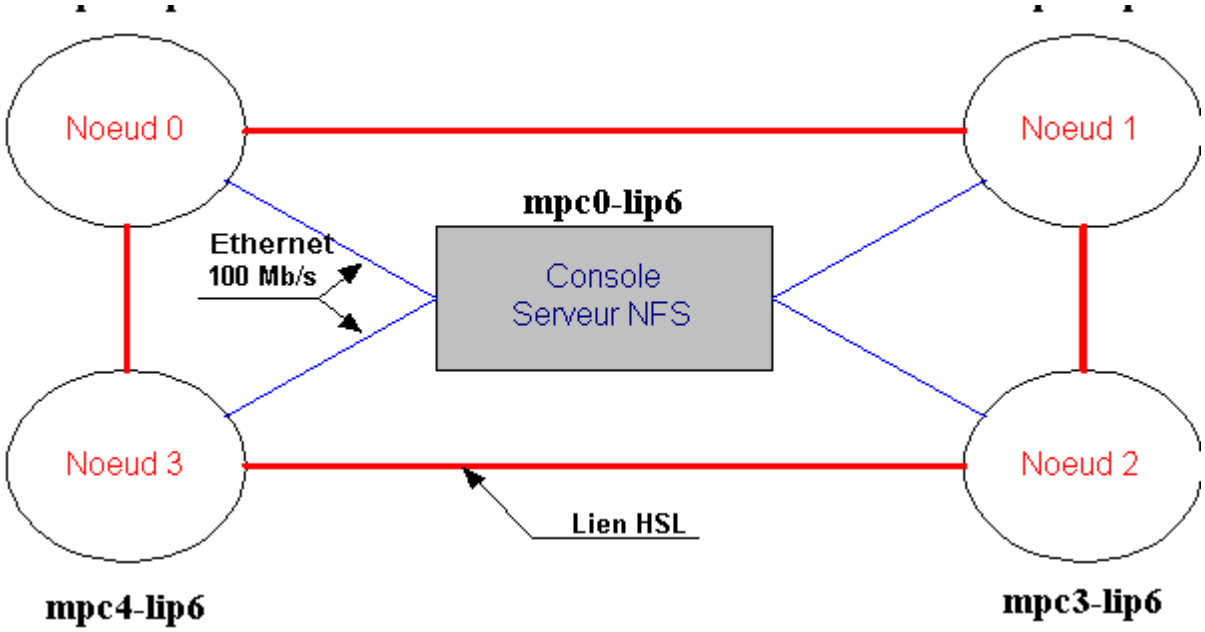


Figure 2 : Architecture de la machine MPC du LIP6

Le réseau d'interconnexions rapides est un réseau à 1 Gigabit/s en full duplex à la norme IEEE 1335 HSL. Il est composé de **liens HSL** (câbles coaxiaux) et d'une **carte FastHSL** sur chaque nœud. Tous les nœuds sont également équipés d'une carte Ethernet pour constituer un réseau de contrôle non seulement, entre eux, mais aussi avec la console.

La carte FastHSL a été conçue au laboratoire LIP6. Elle contient, en particulier, 2 circuits VLSI (cf. Figure 3):

- Un contrôleur de bus PCI intelligent appelé ***PCI-DDC***
- Un routeur rapide possédant 8 liens HSL à 1 Gbit/s appelé ***Rcube***

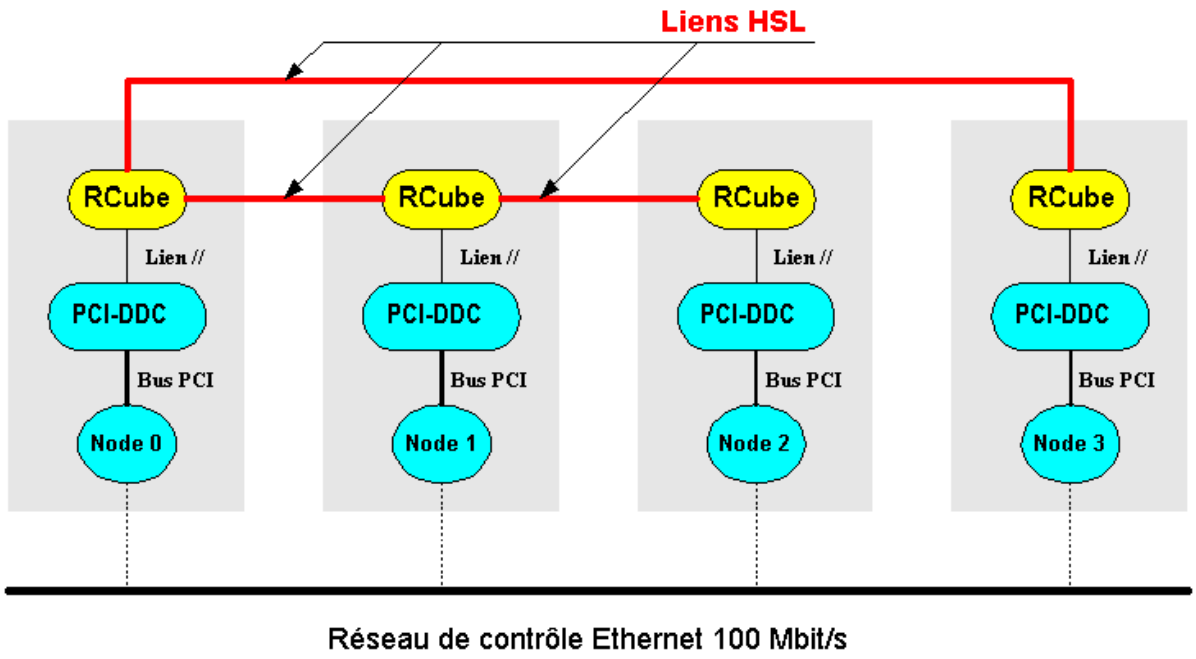


Figure 3 : Quatre cartes FastHSL

PCI-DDC réalise le protocole de communication et **RCUBE** le routage des paquets.

Le réseau ainsi constitué fournit une primitive de communication extrêmement efficace d'écriture en mémoire distante, qui peut être assimilée à un « Remote DMA ». L'enjeu est de faire bénéficier les applications de la très faible latence matérielle du réseau, en minimisant le coût des couches logicielles. MPC promet donc de très bonnes performances via son réseau d'interconnexion haut-débit.

II Architecture logicielle

II.1 L'écriture distante

II.1.2 Principe de l'écriture distante

Le principe général de l'écriture distante ou « Remote-write » est de permettre à un processus utilisateur du réseau HSL et local à un nœud, d'aller écrire dans la mémoire physique d'un nœud distant. Cette opération peut se faire par l'intermédiaire de PCI-DDC qui peut accéder directement à la mémoire physique locale. Il prend ses ordres dans une structure de données présente en mémoire centrale, appelée LPE (Liste des pages à Emettre). Les couches de communication MPC permettent aux processus émetteurs d'ajouter dans la LPE des descripteurs de type DMA qui définissent les paramètres du transfert à effectuer, et de prévenir le composant PCI-DDC de cet ajout. Il se chargealors du transfert et signale au processeur émetteur et/ou récepteur la fin du transfert.

Chaque entrée de LPE définit une page de données à transmettre. Nous parlons ici d'une page au sens HSL : une page est une zone de mémoire physique contiguë dans la mémoire de l'émetteur comme dans la mémoire du récepteur.

Un descripteur de page (entrée LPE) contient les champs suivant :

- Message Identifier (MI) : permet d'étiqueter les messages
- Numéro du nœud destinataire
- Adresse physique locale des données à émettre
- Nombre d'octets à transférer
- Adresse physique distante (dans la mémoire du récepteur) où écrire les données
- Drapeaux

La transmission d'un message se décompose en trois principales étapes (cf. Figure 4).

Préparation : (1) ne entrée est ajoutée dans la LPE. Désormais, le processeur émetteur n'intervient plus. (2) PCI-DDC est informé par les couches de communications de la modification de la LPE.

Transmission : (3) Le PCI-DDC émetteur demande le bus PCI et lit le descripteur de LPE par accès DMA. (4) Il transmet les données, toujours par accès DMA. PCI-DDC décompose le message en plusieurs paquets qui contiennent chacun l'adresse physique distante où les paquets doivent être écrits. Quand le dernier paquet est parti, le processeur émetteur put être prévenu par une interruption (IT1) (si un drapeau est positionné).

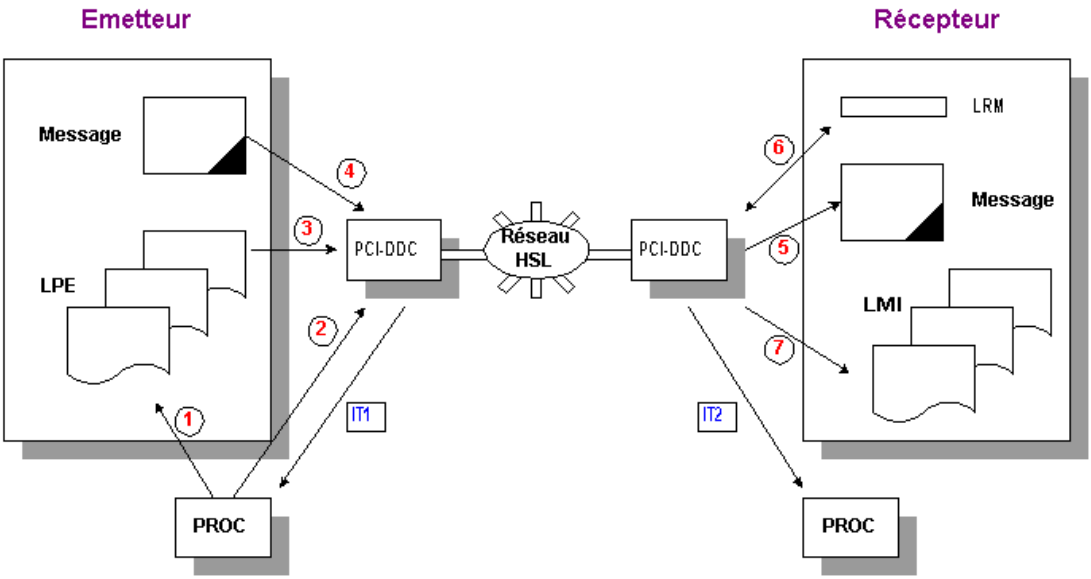


Figure 4 : Les étapes d'une écriture distante

Réception : (5) Dès que le PCI-DDC récepteur commence à recevoir des paquets, il demande le bus PCI et écrit les données en mémoire centrale par accès DMA. (6) Dans le cas d'un réseau adaptatif, les paquets peuvent arriver dans un ordre quelconque (dépendant de la configuration des tables de routage). PCI-DDC utilise la LRM (List of Received Messages) pour compter les paquets reçus. Ce n'est pas le cas de la machine MPC actuelle. (7) Une fois que le dernier paquet a été reçu, PCI-DDC prévient le processeur récepteur soit par un signal d'interruption (IT2) soit par une écriture dans la LMI (List of Message Identifiers). Chaque message est étiqueté par un MPI.

Le message est l'entité au niveau applicatif. Un message étant défini en mémoire virtuelle, il peut-être discontinu en mémoire physique s'il est réparti sur plusieurs pages. Il lui correspond alors plusieurs entrées de LPE. C'est pourquoi, il est nécessaire d'étiqueter le message par un MI afin de pouvoir reconstituer le message à la réception. Chaque descripteur de page contient le MI. Toutes ces manipulations (traduction adresses physiques en adresses virtuelles et réciproquement, gestion de MI, gestion de la LPE, etc.) sont gérées par les couches de communication MPC.

II.1.2 Avantages et inconvénients du « Remote-write »

L'avantage principal de l'écriture distante est qu'elle utilise un protocole simple. Mais surtout, ce protocole est « zéro copie ». PCI-DDC prend directement les données dans la mémoire de l'émetteur et écrit directement dans la mémoire centrale du récepteur. Cela permet de bénéficier de la très faible latence matérielle.
Un autre avantage est l'utilisation d'un lien bi-directionnel. Les communications peuvent se faire dans les deux sens, sans concurrences. Le contrôle de flux se fait au niveau des couches matérielles.

En revanche, ce protocole n'est pas classique dans la mesure où le récepteur est actif. Le modèle de communication est le passage de message sous le mode Receicer-Driven. L'écriture est asynchrone vis à vis du récepteur. Auterment dit, le transfert du message se déroule sous le contrôle du récepteur. Ce modèle suppose un rendez-vous préalable entre l'émetteur et le récepteur sinon, l'émetteur ne sait pas où écrire dans la mémoire du récepteur. De plus, l'émetteur ne prévient pas quand il va écrire. Il écrit directement en mémoire physique ce qui suppose que les applications utilisateurs ont verrouillé à l'avance des tampons en mémoire physique. Enfin, le récepteur ne connaît pas la taille des messages qu'il reçoit.

De ce fait, il est difficile d'adapter des environnements de communication comme MPI aux couche basses MPC car la philosophie n'est pas la même. Par exemple, une émission asynchrone au niveau de PMI peut être pénalisée par le fait qu'une émission suppose d'attendre au préalable un message du récepteur indiquant l'adresse physique dans la mémoire du récepteur.

II.2 Les couche basses de communication MPC

Une pile de drivers noyaux et de démons forment les couches de communication MPC.
La description de ces différentes couches est la suivante :

- La couche PUT constitue le service de communication de plus bas niveau et fournit la fonction *remote write* de l'espace d'adressage physique local vers l'espace physique d'un nœud distant.
- La couche SLRP fournit un service d'échange de zone de mémoire physique, entre nœuds, sur des canaux virtuels.
- La couche SLRV fournit un service équivalent mais en manipulant des zones localisées dans des espaces de mémoire virtuel.
- La bibliothèque UserAcces permet d'accéder aux fonctionnalités de la couche PUT depuis un processus en mode utilisateur.
- CMEM et HSL sont les deux drivers MPC qui permettent d'accéder aux fonctionnalités de la carte FastHSL.

On distingue quatre niveaux d’accès aux couches logicielles MPC :

- Accès depuis le noyau à SLRV
- Accès depuis le noyau à SLRP
- Accès depuis le noyau à PUT
- Accès depuis un processus à PUT (USRPUT)

Ainsi, le portage d’une application peut se faire à différents niveaux suivants les fonctionnalités souhaitées.

Chapitre 3 : Un environnement de programmation parallèle : MPI sur MPC

PVM a déjà été implémenté au dessus de la couche SLR. Malheureusement, les performances obtenues sont faibles, ceci est due a la latence induites par la couche SLR, et des multiples recopie mémoire générées par PVM qui annulent les effort d'implémentation du mode zéro-copie effectués dans la couche SLR.
Le choix d'une API de passage de message s'est donc porté sur une implémentation de MPI au dessus de la couche PUT. Il existe plusieurs API "standards" pour MPI. L'une d'elle, MPICH, développée au " Argonne National Laboratory", a l'avantage de combiner portabilité et performances. Cette API est donc toute désignée pour un portage "relativement facile" sur des nouvelles plate-forme comme la machine MPC.

I Généralités MPI

Sans faire une description exhaustive de l'environnement de programmation MPI, il me paraît cependant nécessaire d'en étudier quelques point qui seront utile pour la présentation du portage de MPICH sur MPC. Car le but de ce portage est de permettre à une application MPI standard de s'exécuter sur la machine MPC.

I.1 Trois types de mécanisme

Le channel interface implémente trois mécanismes différents de communication.

- Eager : Les données sont envoyées directement au destinataire, si le destinataire n'attend pas de données, le receveur doit allouer l'espace utile pour stocker les données. Ce mécanisme offre une bonne performance, en particulier avec une bufferisation convenable, mais peut poser des problèmes si la mémoire du processus receveur est dépassé.
Le protocole eager est le choix par défaut de MPICH standart
- Rendez-vous : Les données sont envoyées seulement quand le destinataire le demande.
Les informations de contrôle décrivant le message sont envoyés. Lorsqu'un Recv est posté, le processus destinataire envoie une requête vers la source pour recevoir les données.
- Get : Dans ce protocole, les données sont lues directement par le receveur. Ce protocole a besoin d'une méthode de transfert directe de données d'une mémoire à l'autre. « Ce protocole n'est pas implémenté dans la version standard de MPICH »

I.2 Sémantique des primitives MPI

1) Communications bloquantes.

MPI_Send et MPI_Recv

Le Send standard bloquant ne retourne pas tant que les données du message et l'enveloppe n'ont pas été envoyées au destinataire, c'est à dire que l'émetteur peut accéder librement au buffer contenant le message envoyé. Ce message peut-être directement copié dans le buffer de réception ou bien dans un buffer temporaire (chez le processus récepteur).

Ce mode est non-local, il peut dépendre de l'occurrence des MPI_Recv postée chez le récepteur.

2) Communications bloquantes avec bufferisation

MPI_Bsend

Le message a envoyer est copié dans un buffer local. La fonction MPI_Bsend retourne si les données et l'enveloppe ont bien été copié dans le buffer local. Le retour de la fonction ne dépend que de l'espace libre dans le buffer local et non des Recv exécutés par le processus receveur.

Un buffer doit donc être alloué en mémoire par le programmeur grâce à la fonction MPI_Buffer_attach().

3) Communications bloquantes avec le mode Ready

MPI_Rsend retourne avec succès seulement si un Recv a déjà été posté par le destinataire. Il n'y a pas de copie.

4) Communications en mode synchronisé

MPI_Ssend ne retourne que lorsqu'un Recv est posté par le processus destinataire. Il n'y a donc pas de copie. Les informations de contrôle décrivant le message sont envoyées. Lorsqu'un Recv est posté, le destinataire envoi une requête vers la source pour recevoir les données.

5) Communications non-bloquantes

Les quatre modes (standard, bufferisé, synchronisé, ready) peuvent être non-bloquants. Les primitives d'envoi retournent avant que le buffer soit copié vers le destinataire. Les opérations non-bloquantes utilisent un request object pour identifier les opérations de communication. Le request handler (de type MPI_REQUEST) est passé en argument de la fonction d'envoi ou de réception non-bloquante. Il est ensuite utilisé pour se renseigner sur l'état de l'exécution ou pour attendre la fin de l'exécution de la primitive de communication.

MPI_Wait, MPI_Test,…

II L'implémentation de MPI sur la machine MPC

II.1 Architecture de MPICH

MPICH requière une interface qui fournit les services de bufferisation et de contrôle de flux. Ces services n'existe pas au niveau de PUT. Il faut donc implémenter une interface entre L'API MPI et PUT. Dans ce domaine, le laboratoire RESAM (Lyon) a développé un portage de MPICH sur BIP (Basic interface for parallélisme).

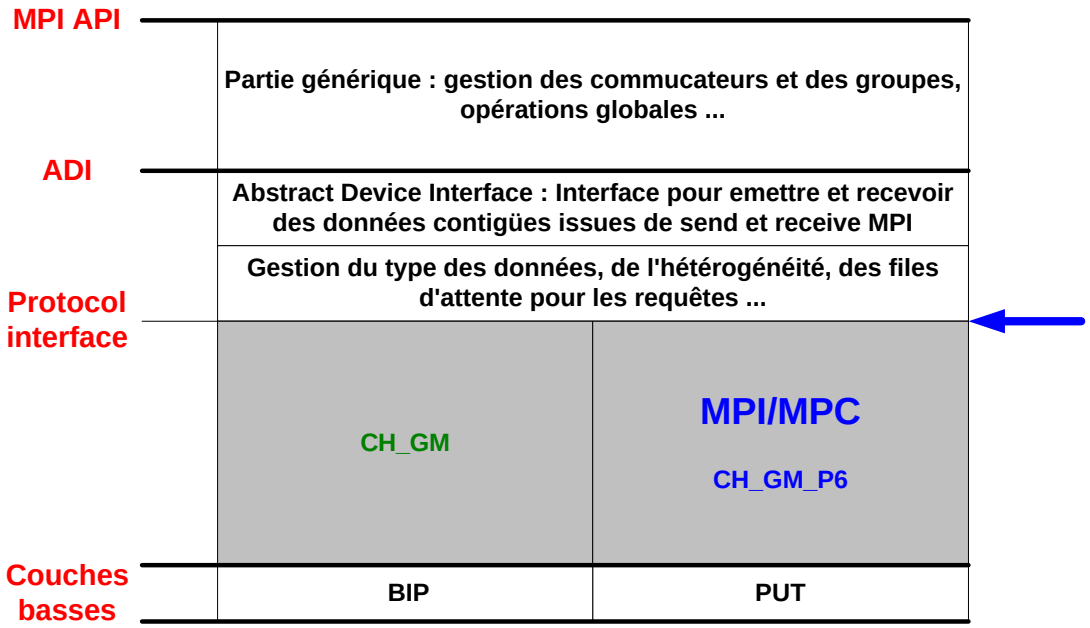


Figure 5. MPICH architecture

II.2 Problématiques

Les spécifications de la couche PUT (cf. chapitre précédent) impliquent deux principales problématiques.

1) Les données transmises par PUT doivent être localisées de manière contiguë en mémoire physique.

La fonction PUT d'écriture distante envoie des donnés au niveau physique. Un tableau de caractères contiguë dans l'espace d'adressage d'un processus ne l'est pas forcément dans l'espace d'adressage physique.

Pour envoyer un bloc de mémoire virtuelle, il faut d'abord faire un mlock() sur ce bloc pour être sur qu'il reste en mémoire sans changer d'adresse physique. Une traduction d'adresse est alors effectuée pour déterminer les fragments de mémoire physiques contiguë qui composent ce bloc. Comme le montre la Figure 6, il peut y avoir différentes fragmentation de bloc de mémoire virtuelle en mémoire physique. Pour l'envoi du message, il faudra autant de PUT que de fragments.

Dans le cas du schéma ci-dessous, trois buffers physique sont nécessaire.

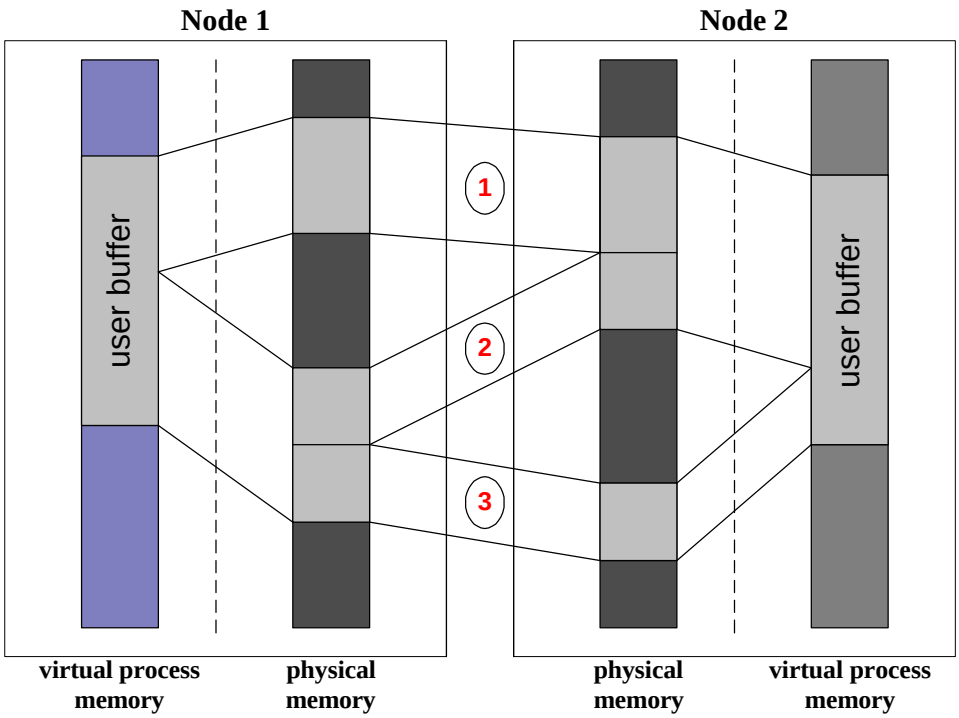


Figure 6. Mémoire physique et mémoire virtuelle

2) Pour émettre les données grâce à la primitive d'écriture distante, il faut savoir où écrire les données dans la mémoire physique distante.

Pour obtenir les meilleurs performances, ces spécifications ont une grande importance dans l'implémentation de MPICH sur PUT. Le choix des protocoles et des types de messages est donc déterminé en fonction de ces spécifications.

II.3 Les messages de l'API GM-P6

Comme nous l'avons vu au niveau de l'architecture, une couche appelée CH_GM_P6 a été développée en s'inspirant des protocoles de la couche CH_GM.
Rappelons que pour l'implémentation initial, pour des raisons de simplicité et de performance, on considère qu'une seule tache s'exécute par nœud. Une implémentation avec plusieurs taches par nœuds est prévu par la suite.

II.3.1 Les différents types de messages

Au niveau de CH_GM_P6, deux types de messages sont utilisés :
- Les messages de contrôle.
- Les messages de données

II.3.2. Les messages de contrôle

Les messages de contrôle sont utilisés pour transférer rapidement sur le réseaux, des informations de contrôle ou des données utilisateur de petites tailles.
Il y a quatre principaux types de messages de contrôle:
- SHORT : émission des données utilisateur encapsulées dans un message de contrôle.
- REQ : envoi d'une requête pour un protocole de rendez-vous.
- RSP : réponse a une requête (contient la description du tampon de réception en mémoire physique).
- CRDT : contrôle de flux au niveau MPI (Algorithme à crédit).

A l'initialisation, MPC alloue un tableau de mémoire physique contiguë pour chaque nœud. Lorsqu'une tache MPI démarre (pour l'instant, une par nœud), elle récupère un slot de cette mémoire. Chaque nœud récupère aussi l'adresse physique de tous les slots distants grâce au réseau de contrôle (Tous les nœuds sont connectés par un réseau Ethernet pour la configuration). Chaque slot est découpé en sub-slots correspondants a l'émission et aux réceptions des messages. Par exemple, pour 3 nœuds, les sub-slots 2 et 3 du nœud 1 sont utilisés pour recevoir les messages des nœuds 2 et 3. Le sub-slot 1 du nœud 1 est utilisé pour les émissions de la tache local. Les sub-slots sont composés de N tampons qui contiendront les messages.

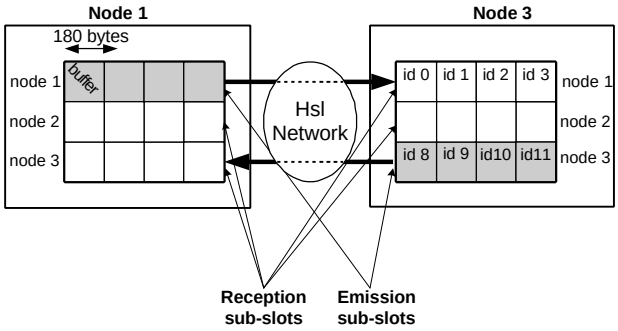


Figure 7. Structure des slots en mémoire

Les messages de contrôle utilisent les tampons contigus en mémoire physique pré-alloués. Il y a donc deux copies, une en émission, une en réception, pour les messages courts. Vu la taille des messages, ce n'est pas trop pénalisant pour les performances.

II.2.3. Les messages de données

Ce type de message est utilisé pour transmettre des données dont la taille est supérieure a la taille maximum d'un message de contrôle, ou lors d'envoi de messages MPI en mode zéro-copie. Les messages de grande taille ne peuvent être contenus dans une tampon alloué statiquement. Pour une bonne utilisation des propriétés de la couche PUT, le receveur doit donc envoyer un message a l'émetteur lorsqu'il est prêt a recevoir. Il faut donc un protocole de rendez-vous.

Afin de garantir de bonnes performances en utilisant le principe de l'écriture distante, l'échange de messages de données se fait en mode zéro-copie, par un protocole de rendez-vous. L'envoi de messages de données est donc précédé d'un échange de messages de contrôle. L'émetteur envoie une requête au récepteur, celui-ci répond en envoyant la description du tampon en mémoire physique qui a été alloué. Les données sont alors transmises.

Dans le cas où la requête d'émission est envoyé avant qu'une réception n'ait été postée par la tache du nœud récepteur, cela pourrait coûter trop chère à la tache émettrice d'attendre la réception. Un tampon de réception est alors alloué dans la mémoire physique pour la réception. Le descripteur de ce tampon est envoyé dans la réponse RSP. Une recopie sera faite au moment où la réception sera postée, pour la récupération des données.

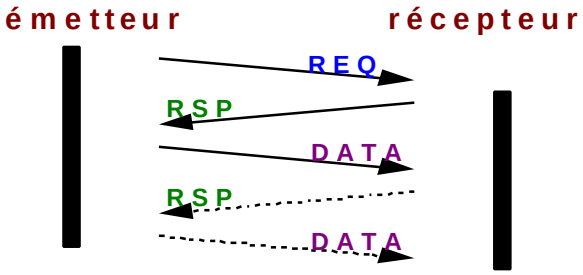


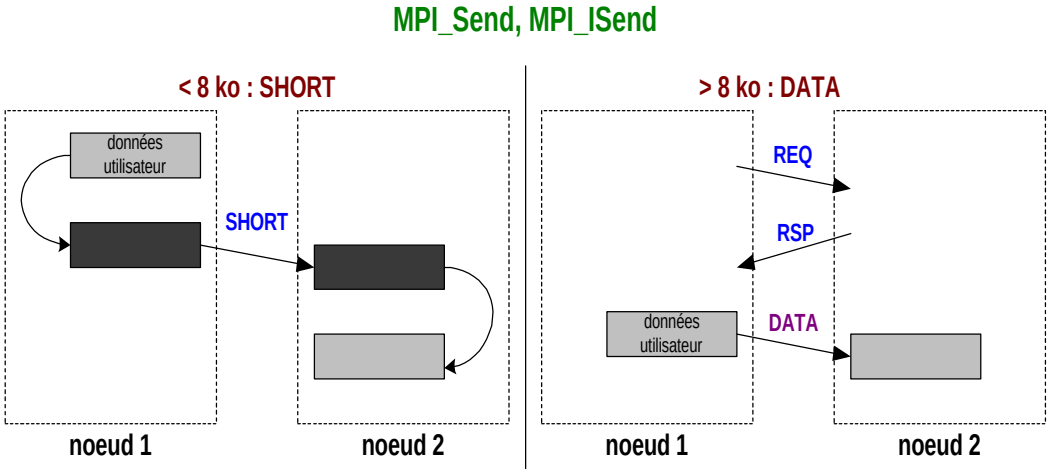
Figure 8. Protocole pour le transfert des messages de donnée

II.4 Les primitives MPI

Nous ne considérons ici que le primitives d'envoi et de réception de message

- MPI_Send, MPI_Isend

Il y a deux types d'émission en fonction de la taille du message :



2 copies, 1 message 0 copie, 3 messages.

Figure 9. Les deux types d'émission des données.

- Mode synchronisé
MPI_Ssend et MPI_Issend

Pour l'envoi synchronisé, on se place dans le mode DATA quelques soit la taille des messages.

- Mode bufferisé
MPI_Bsend et MPI_Ibsend

Le protocole est identique a celui de MPI_Send et MPI_Isend

- Mode Ready
MPI_Rsend et MPI_Irsend

Le protocole est identique a celui de MPI_Send et MPI_Isend mais on n'a plus besoin de REQ. On gagne donc un message de contrôle par rapport au protocole de rendez-vous.

II.5 Les API dans MPI sur MPC

Le schéma ci-dessous montre les différents composants de l'implémentation de MPI sur MPC.

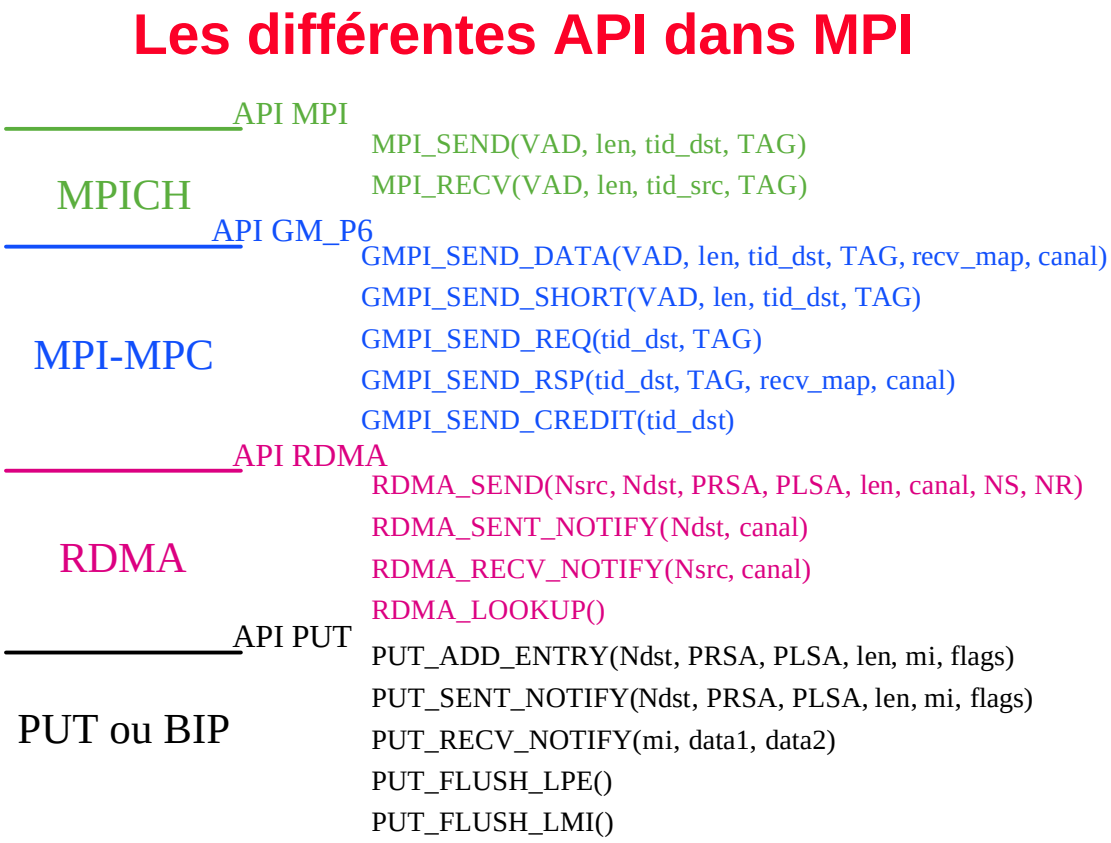


Figure 10. Les API dans MPI

III Performances de la machine MPC

Cette partie a pour but d'évaluer les performances des la machine MPC, et en particulier le portage de l'API MPICH

III.1 Cadre de mesures

La machine MPC du laboratoire LIP6 a été modifiée aussi bien au niveau matériel que logiciel pendant le déroulement de mon stage.

Au début du stage, la machine MPC est constituée de quatre nœuds bi-proc PII 350Mhz avec chacun 128 Mo de mémoire. Le réseaux HSL est complet. Les nœuds sont tous reliés par un lien HSL. Le réseau de contrôle est un réseau éthernet 100 Mbits/s. Il est centralisé sur la console MPC0.

Vers la fin du stage, le laboratoire s'est doté de nouveaux nœuds de calcul. Il s'agit de 8 bi-pro PIII1Ghz avec 1 Go de mémoire. Le réseau HSL a été installé sur ces nouveaux nœuds, ainsi que le réseau de contrôle ethernet 100 Mbits/s. Il subsiste encore quelques problème de configuration sur ces nouveaux nœuds. Les mesures pour la couche MPI-MPC sur ces 8 nouveaux nœuds n'ont pu être faites.

Pour la partie logicielle, les différentes couches de la machine MPC qui étaient précédemment sous FreeBSD ont été portées sous LINUX. Des optimisations comme le passage de PUT noyau en un PUT utilisateur et le remapage de la mémoire sont en cours de développement..

Au niveau matérielle, c'est a dire le lien HSL le débit théorique de 1Gbit/s et la latence est de l'ordre 5µs pour le PUT en mode noyau.

III.2 Mesures de latence et de débit

III.2.1 Principe et programme

Des programmes ont été écrits pour mesurer les différents temps de transfert des messages. Le principe est le même pour chaques mesures. Il s'agit d'un ping-pong. Pour une taille M donnée, un message est échangé N fois entre deux processus. Le temps de transfert du message de taille M est donc égal à la moyenne du temps des N aller-retours, divisé par 2.

```
// Ping
pour i allant de 1 à 17
    taille = 2^i
    pour j allant de 1 à 1000
        déclencher timer
        send(message[taille])
        recv(message[taille])
        arrêter timer
    fin pour
    calcul temps de transfert(i)
fin pour
```

```
// Pong
pour i allant de 1 à 17
    taille = 2^i
    pour j allant de 1 à 1000

        recv(message[taille])
        send(message[taille])

    fin pour
    calcul temps de transfert(i)
fin pour
```

III.2.2 Les différentes mesures

Les mesures ont été réalisée au niveau de la couche MPI-MPC et de la couche MPICH standard sur Ethernet. Le principe et le code sont identiques dans les deux cas.

Pour expliquer les différentes mesures, il est utile de détailler quelques points importants.

1) Les deux modes de signalisation

Il y a deux différents modes pour la signalisation lors de la réception ou de l'émission des messages : le mode avec interruptions et le mode sans interruption (Polling). Sans rentrer dans les détails qui dépassent le cadre de ce rapport, les caractéristiques de ces deux différents modes sont les suivantes. Dans le premier cas, le processus qui reçoit un message de contrôle est prévenu par un signal d'interruption (cf Figure 0). Dans le second cas, chaque processus scrute la table des messages reçus (LMI, List of Message Identifiers) pour se renseigner sur l'arrivée de nouveaux messages.

2) Les deux options de MPI-MPC

Dans l'implémentation de MPI sur MPC, plusieurs politiques sont possibles au niveau de l'allocation mémoire et de la traduction d'adresse virtuelle en adresses physiques.

1) Il existe une primitive MPI_Malloc_MPC() dans la librairie MPI spécifique à MPC. Cette primitive remplace le malloc() de la libc et permet de réserver un espace contiguë en mémoire physique. Ceci pour se rapprocher des spécifications de la couche PUT.
2) Pour la traduction d'adresse virtuelle en adresse physique (II.2 Problématiques), soit cette traduction est faite à chaque envoi du message, soit elle est faite une fois pour toutes. Dans ce cas, le résultat de la première traduction est sauvegardée dans une table pour être réutilisé lors de futurs envois de la même zone mémoire. Les blocs de mémoire sont verrouillés en mémoire physique.

Dans les tableaux et les courbes, les abréviations utilisées ont la signification suivante :

- **MPI-ETH** : communication avec MPI standard sur Ethernet

- **MPI-MPC PUT IT** : communication avec MPI-MPC et PUT avec interruptions

Pour MPI-MPC avec PUT sans Interruption :

- **MPI-MPC** : communication avec MPI-MPC

- **MPI-MPC (MPI_Malloc_MPC)** : communication avec MPI-MPC en utilisant la primitive MPI_Malloc_MPC.

- **MPI-MPC (Cache_Trad)** : communication avec MPI-MPC en conservant le résultat de la traduction d'adresse.

- **MPI-MPC (MPI_Malloc_MPC et Cache_Trad)** : : communication avec MPI-MPC en conservant le résultat de la traduction d'adresse et en utilisant la primitive MPI_Malloc_MPC().

III.2.3 Tableau des temps de transfert

Taille (octets)	MPI-ETH	MPI-MPC PUT IT Cache_trad	MPI-MPC	MPI-MPC MPI_Malloc_MPC	MPI-MPC Cache_trad	MPI-MPC MPI_Malloc_MPC et Cache_trad
1	163	28	21	21	21	21
2	161	28	21	20	21	20
4	161	28	21	21	21	21
8	162	29	22	21	21	21
16	161	29	22	21	21	21
32	164	29	22	21	21	21
64	162	30	22	21	22	22
128	164	32	25	23	24	23
256	167	35	26	27	27	27
512	181	41	33	32	32	32
1024	232	54	47	46	46	46
2048	324	79	74	73	74	73
4096	520	131	128	126	127	133
8192	955	250	252	250	253	251
16384	1837	442	570	558	394	389
32768	3576	753	871	849	683	680
65536	7091	1384	1475	1402	1269	1239
131072	14592	2646	2694	2547	2428	2378
262144	29069	5163	5146	4828	4757	4664
524288	58920	10283	10175	9387	9450	9244
1048576	117531			18575		18394
2097152				37091		36766

Figure 11. Tableau des temps de transfert.

Lorsque MPI_Malloc_MPC n'est pas utilisé, il n'est pas encore possible de transférer des très long messages. Ceci est du à la taille limitée de la table contenant la description de la répartition des messages en mémoire physique . La taille de cette table est de 255 (NB_DMA_MAX). Grâce au MPI_Malloc_MPC, il n'y a qu'un seul descripteur DMA.

II.2.4 MPI sur MPC et MPI sur Ethernet

Les performances de transfert de message sur la machine MPC sont nettement meilleurs que celles sur Ethernet.

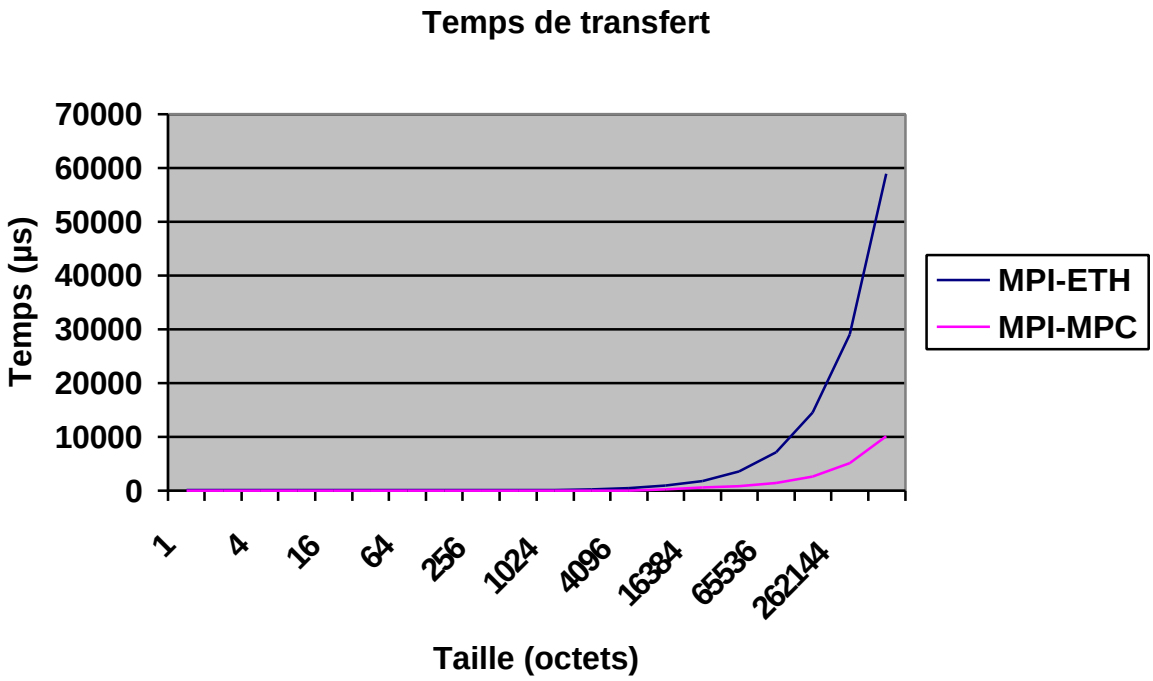


Figure 12. Temps de transfert pour MPI sur MPC et MPI sur Ethernet

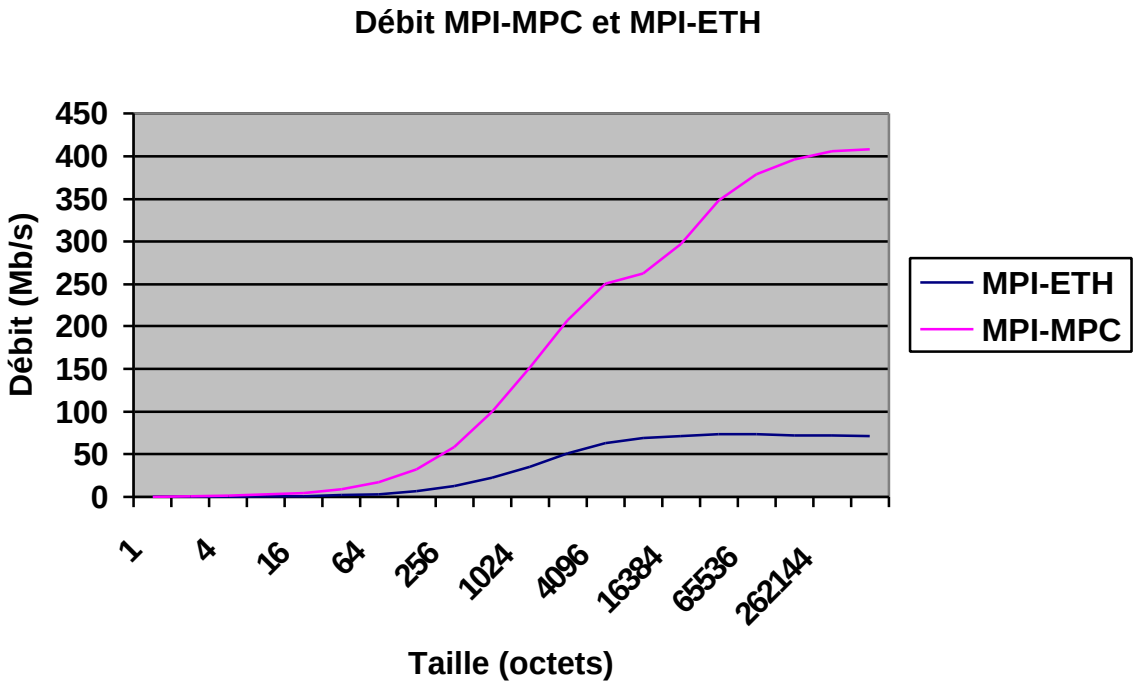
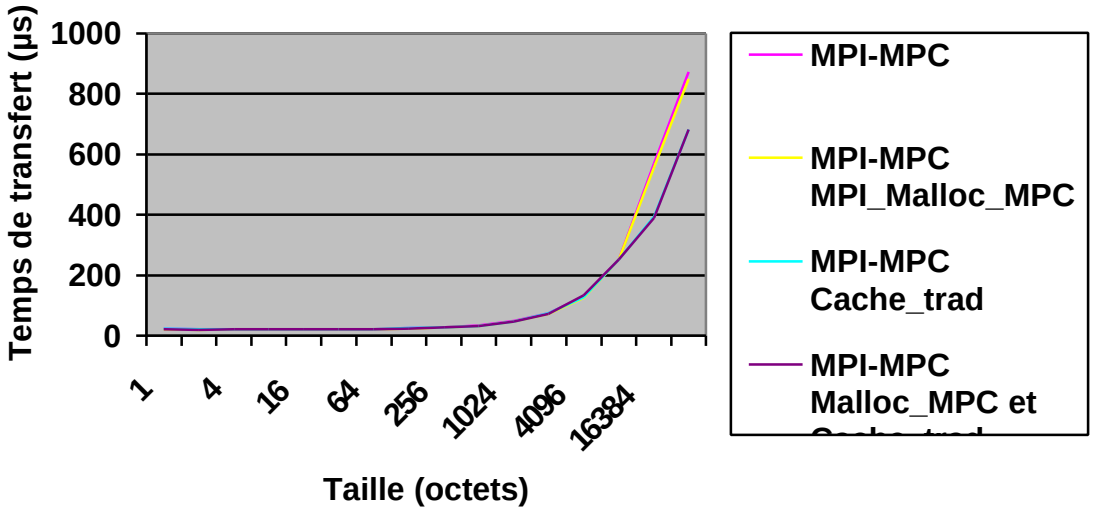


Figure 13. Débit MPI sur MPC et MPI sur Ethernet

II.2.5 Les quatre modes de transfert avec MPI-MPC

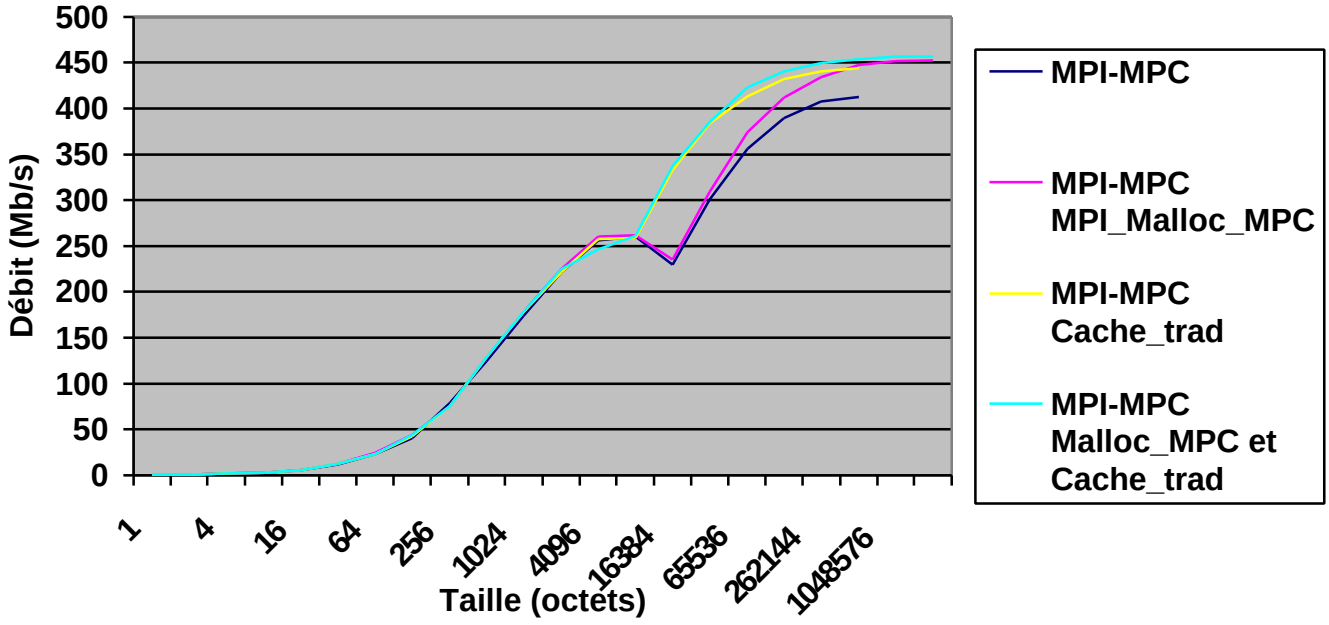
Ces mesures sont réalisée avec la couche PUT en mode sans interruption. Elles permettent d'étudier les différents coût induits par la traduction d'adresse et la répartition des données du message en mémoire physique.

Figure 14. Temps de transfert pour MPI-MPC



On peut remarquer que pour des messages dont la taille est inférieure à 8k0, le temps de transfert est sensiblement le même pour les quatre modes différents. En effet, les données sont encapsulées dans des messages de contrôle. Pour une taille supérieure à 8ko, le graphique des temps de transfert montre bien le coût de la traduction d'adresse.
La latence mesurée est de 21 µs.

Figure 15. Débits pour MPI-MPC



Dans tous les cas, il y a une chute du débit à partir de 8Ko. Il s'agit de la taille maximum des messages de contrôle. A partir de cette taille, les données ne peuvent pas être encapsulées dans les messages de contrôle, et il y a un protocole de rendez-vous entre les deux nœuds. Ce protocole ajoute 2 messages de contrôle et éventuellement une recopie du message chez le processus récepteur.
Les courbes des débits montrent bien l'influence du MPI_Malloc_MPC et de la traduction d'adresse. Les meilleurs performances sont obtenues lorsque l'espace mémoire des données est alloué grâce à MPI_Malloc_MPC() et que la traduction d'adresse ne s'effectue qu'une seule fois.
L'utilisation du MPI_Malloc_MPC seul permet d'atteindre un meilleur débit à partir de 16ko. Il permet en effet de réduire le nombre d'informations échangées dans les messages de contrôle précédant l'envoi des données.
Le graphique des débits met en évidence le coût important de la traduction d'adresse pour des messages dont la taille est supérieure à 8ko. Cette traduction peut représenter jusqu'à 35% du temps de transfert et le débit maximum est amélioré de 30Mb/s soit 7,6% de mieux.

II.2.6 Comparaison avec MPI-BIP

Les mesures suivantes n'ont pas été effectuées pendant mon stage. Elles sont issue de l'article Protocol and Performance Analysis of the MPC Parallel Computer[1].

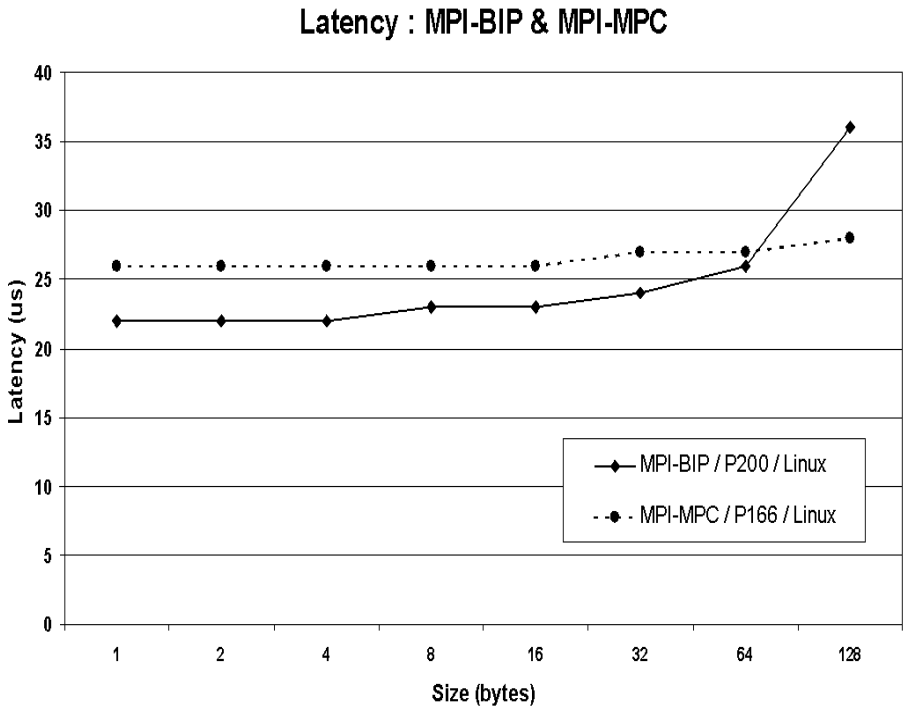


Figure 16. Latence MPI-BIP et MPI-MPC

La latence observée sur MPI-PMC (P166/Linux) est de 26µs, et celle de MPI-BIP (P200/Linux) de 22µs.

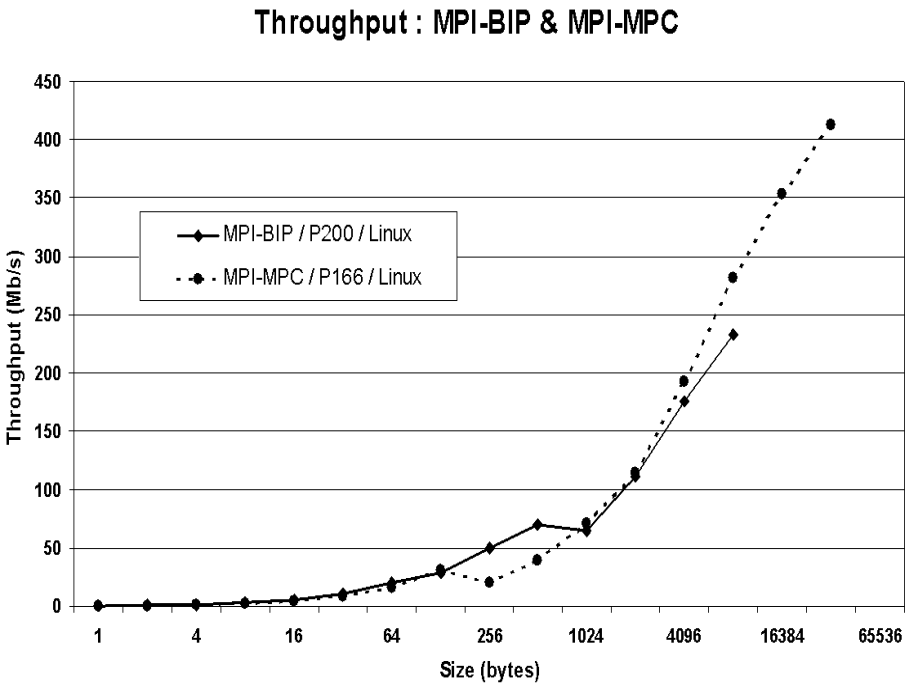


Figure 17. Débit MPI-BIP et MPI-MPC

II.2.7 Tableau des latences et des débits

II.2.7.1 MPI-MPC et MPI-ETH

	MPI-MPC	MPI-ETH
Latence (µs)	28	161
Débit (Mb/s)	407.9	71.4
Demi-débit (octets)	16384	1024

II.2.7.1 MPI-MPC avec PUT sans interruption

	MPC	MPC MPI_Malloc_MPC	MPC Cache_trad	MPC MPI_Malloc_MPC et Cache_trad
Latence (µs)	21	21	21	21
Débit (Mb/s)	412.2	452.3	443.8	456.3
Demi-débit (octets)	1536	2048	2048	2048

II.2 Les optimisations pour MPI-MPC

II.2.1 Le réordonnancement de la mémoire.

Malgré de bonne performances, les mesures ont mis en évidence les lacunes de la machine MPC au niveau du transfert de très gros messages. MPI_Malloc_MPC permet de résoudre ce problème mais l'inconvénient est qu'il ne s'agit pas d'une fonction standard. Remaper la mémoire d'un processus dans un espace contiguë en mémoire physique avant l'exécution d'une tache MPI est une solution.

II.2.1 Passage du PUT Noyau en PUT utilisateur

La couche PUT se trouvant dans l'espace noyau, un processus utilisateur est obligé de faire un appel système pour l'utiliser. Les changement de contexte induisent un perte de temps non négligeable. Cette couche de communication a donc été portée en mode utilisateur.

Chapitre 4

Parallélisation de l'équation de Laplace sur une grille bi-dimensionnelle

Ce chapitre traite de la parallèlisation de l'équation de Laplace sur une grille bi-dimensionnelle. Il ne fait pas une liste exhaustive de toutes les façons de procéder afin d'arriver à ses fins. Dans un premier temps, je rappelle ce qu'est l'équation de Laplace et la méthode de résolution séquentielle. Ensuite, une première méthode de parallélisation est donnée (méthode SOR). Enfin, je présente une optimisation classique de cette parallélisation : la méthode « Red & Black ». Cette application m'a aussi permit de tester les communications non-bloquantes et d'en mesurer le gain en performance.

I. L'équation de Laplace

La résolution de l'équation de Laplace a de nombreuses applications dans les domaines suivants : mathématiques appliquées, mécanique des fluides, résistance des matériaux, électromagnétisme, équation de poisson, etc. Il s'agit d'une équation aux dérivées partielles. Son équation en dimension deux est la suivante :

$$\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial u^2} = 0 \qquad \nabla^2 F(x,y) = 0$$

La résolution de cette équation consiste à trouver les valeurs de F(x,y) sur un domaine Ω inclus dans $\mathbf{R}^2$ dont la frontière est Γ. La seule donnée de cette équation n'est pas suffisante à la détermination d'une solution unique. Pour cela, il faut des conditions limites qui portent aussi parfois le nom de conditions initiales. Il existe deus types de conditions limites : les conditions de Neumann et celles de Dirichlet [4.4]. Par exemple, les conditions de Dirichlet consistent en la donnée des valeurs de F sur Γ, la frontière du domaine de résolution.

Pour résoudre cette équation, il existe des méthodes directes (utilisant des inversions de matrices bandes) et des méthodes itératives. Les méthodes directes sont parfois plus rapides mais les méthodes itératives ont l'avantage de permettre le choix de la précision de résolution et donc de pouvoir accélérer la convergence. Nous nous intéressons désormais aux méthodes itératives avec des conditions limites de Dirichlet.

En utilisant une discrétisation selon x et y, on montre que la résolution de l'équation de Laplace revient à mettre à jour chaque point intérieur de la grille par la moyenne de ses 4 voisins, les valeurs des frontières étant fixées par les conditions initiales. La mise à jour s'arrête lorsque chacun des points de la grille est suffisamment proche de la moyenne de ses quatre voisins [4.3][4.2]. Il y a alors convergence.

Les points de la grille peuvent être mis à jour suivant différentes méthodes itératives. Citons par exemple : **l'itération de Jacobi** :

$$U_{i,j}^{(k+1)} = \frac{1}{4} * (U_{i+1,j}^{(k)} + U_{i-1,j}^{(k)} + U_{i,j+1}^{(k)} + U_{i,j-1}^{(k)})$$

où $U_{i,j}^{(k)}$ est la valeur du point (i,j) après la k^{ième} itération [4.4]. La mise à jour des valeurs calculées à l'itération k+1 utilise uniquement les valeurs calculées à l'itération k. Donc, cette métode nécessite de conserver avant chaque nouvelle itération les valeurs calculées à l'itération précédente.

Une deuxième méthode de résolution classique est basée sur **l'itération de Gauss-Seidel :**

$$U_{i,j}^{(k+1)} = \frac{1}{4} * (U_{i+j,j}^{(k)} + U_{i-1,j}^{(k+1)} + U_{i-1,j}^{(k+1)} + U_{i,j+1}^{(k)} + U_{i,j-1}^{(k+1)})$$

Le procédé est identique à la méthode de Jacobi mis à part le fait que la mise à jour des valeurs calculées à l'itération k+1 utilise les valeurs fraîchement calculées pendant cette itération (k+1) pour les voisins nord et ouest .Il n'est donc plus nécessaire de conserver les valeurs de l'itération précédente. Cette méthode est parfois appelée la méthode SOR (Successive Over Relaxation) lorsqu'on utilise également l'ancienne valeur des Ui,j pour la mise à jour.
L'avantage de cette méthode est que la convergence est plus rapide puisqu'elle accélère la circulation de l'information sur la grille mais elle engendre un inconvénient pour la parallélisation.

II Parallélisation de l'équation de Laplace

II.1 La méthode de Jacobi

La principale question est celle de la décomposition du problème sur les différents processus. La matrice (NxN) est découpée en plusieurs bandes (N/nprocs lignes) en raison de la simplicité des communications. Chacune de ses sous-grilles est attribuée à un processus. A chaque itération, les processus échangent leurs frontières avec leurs voisins puis mettent à jour leur partie de la grille.

Pseudo-code du programme

```
Pour niter allant de 1 à MAX_ITER
        mylocal_change = 0 ;          // le plus grand écart localement
        send(MesFrontières)           // si niter > 1
        recv(FrontièresVoisines)      // si niter > 1
        calcul                        // calcul des nouveaux points
        si test_convergence alors
                send(mylocal_change)          // envoi au processus 0
                recv(réponse)
                si réponse=convergence alors quitter pour
        fin si
fin pour
```

Une deuxième version a été écrite en utilisant des communications non-bloquantes. Le but est d'optimiser le temps d'exécution grâce au recouvrement calcul/communication.

Pseudo-code méthode Jacobi avec les primitives non-bloquantes

```
Pour niter allant de 1 à MAX_ITER
        mylocal_change = 0 ;          // le plus grand écart localement
        Isend(MesFrontières)          // aux voisins si niter > 1
        Irecv(FrontièresVoisines)     // si niter > 1
        calcul(Points_intérieurs)
        Test(send & recv)             // attendre la fin des réceptions
        calcul(MesFrontières)
        si test_convergence alors
                send(mylocal_change)          // envoi au processus 0
                recv(réponse)
                si réponse=convergence alors quitter pour
        fin si
fin pour
```

II.2 La méthode Red&Black

Le principe de la parallélisation par la méthode R&B s'inspire de la parallélisation par la méthode de Jacobi et la méthode de Gaus-Seidel. La grille est vu comme un échiquier de points rouges et noirs. Un point rouge est systématiquement entouré par quatre points noirs et réciproquement. L'intérêt est que la mise à jour des points rouges se fait uniquement à partir des points noirs. De même, la mise à jour des points noirs utilise seulement les points rouges.

On peut alors imaginer une résolution en deux phases successives : la phase 1 consiste en la mise à jour des points rouges. La phase 2 utilise les nouvelles valeurs des points rouges pour mettre à jour les points noirs. Chaque ligne de la grille est traitée en deux temps.

Pseudo-code du programme rbmpi.c (résolution R&B)

```
Pour niter allant de 1 à MAX_ITER
        mylocal_change = 0 ;          //le plus grand écart local

        // Phase 1 : mise à jour des points rouges
        recv(FrontièresNoires_Voisines)     // si niter > 1
        calcul(MesFrontièresRouges          // utilise les points noirs
        send(MesFrontièresRouges)           // aux voisins
        calcul(Points_intérieurs_Rouges)    // utilise uniquement les points noirs

        // Phase 2 mise à jour de points noirs
        recv(FrontièresRouges_Voisines)
        calcul(MesFrontièresNoires)         // utilise uniquement les points rouges
        send(MesFrontièresNoires)           // aux voisins
        calcul(Points_intérieurs_noirs)     // utilise uniquement les points rouges

        // Test de la convergence toutes les 50 itérations
        si test_convergence alors
                send(mylocal_change)        // envoi au proc 0
                recv(réponse)               // attendre réponse du proc 0
                si réponse=convergence alors quitter pour
        fin si
fin pour
```

Cette méthode permet de calculer les valeurs des frontières à l'avance et de les envoyer aussitôt afin de réduire les effets de synchronisation tout en évitant une recopie de anciennes valeurs avant chaque itération. De plus la convergence est accélérée car à chaque itération, les points noirs utilisent les nouvelles valeurs des rouges.
Des études [4.1][4.2][4.3][4.4] ont montré l'intérêt des cette parallélisation.
La méthode R&B est très utilisée dans le monde du parallélisme. Elle permet de résoudre de nombreux problèmes au-delà de l'équation de Laplace

De même que pour la méthode Jacobi, un programme a été écrit en utilisant des communications non-bloquantes.

Pseudo-code pour la méthode R&B avec communication non-bloquante.

```
pour niter allant de 1 à MAX_ITER
        mylocal_change = 0 ;          //le plus grand écart local

        // Phase 1 : mise à jour des points rouges
        Irecv(FrontièresNoires_Voisines)    // si niter > 1
        calcul(Points_intérieurs_Rouges)    // utilise uniquement les points noirs
        Wait (send_FrontièresRouge)         // si niter > 1
        Wait(Recv_FrontièresNoires)
        calcul(MesFrontièresRouges)         // utilise les points noirs
        Isend(MesFrontièresRouges)          // aux voisins

        // Phase 2 mise à jour de points noirs
        Irecv(FrontièresRouges_Voisines)
        calcul(Points_intérieurs_noirs)     // utilise uniquement les points rouges
        Wait(Recv-FrontèresRouges)
        Wait((Send_FrontièresNoires)        // si niter > 1
        calcul(MesFrontièresNoires)         // utilise uniquement les points rouges
        Isend(MesFrontièresNoires)          // aux voisins

        // Test de la convergence toutes les 50 itérations
        si test_convergence alors
                send(mylocal_change)        // envoi au proc 0
                recv(réponse)               // attendre réponse du proc 0
                si réponse=convergence alors quitter pour
        fin si
fin pour
```

III Mesures de performance sur la parallélisation de l'équation de Laplace

Il faut remarquer que les programmes de résolution de l'équation de Laplace décrits ci-dessus ne permettent pas de comparer la machine MPC avec d'autres clusters au niveau des performances, puisqu'il ne s'agit pas de programmes « standard ». Faire tourner ces applications sur MPC constitue cependant une validation du portage de MPI et permet de tester des optimisations au niveau de la parallélisation.

Les quatre programmes testés sont :
- Résolution par la méthode de Jacobi avec des communications bloquantes
- Résolution par la méthode de Jacobi avec des communications non bloquantes
- Résolution par la méthode Red & Black avec des communications bloquantes
- Résolution par la méthode Red & Black avec des communications non bloquantes

Pour chacun de ces programmes, deux séries de tests ont été effectués. Tout d'abord, en faisant varier la taille de la matrice carrée, ce qui représente la taille globale de l'application. Puis en ne modifiant que la taille des lignes, ce qui permet d'augmenter les temps de communications par rapport aux temps de calcul.

Deux bancs de tests ont donc été lancé. Le premier pour des matrices de taille 40x40, 80x80, 160x160, 320x320, 640x640, 960x960, 1280x1280. Le second, pour des matrices de taille 40x40, 40x150, 40x300, 40x600, 40x1200, 40x2400, 40x4800, 40x9600, 40x19200, 40x25000, 40x3000.

Tous les tests ont été lancés sur 4 processus, un processus pas nœud.

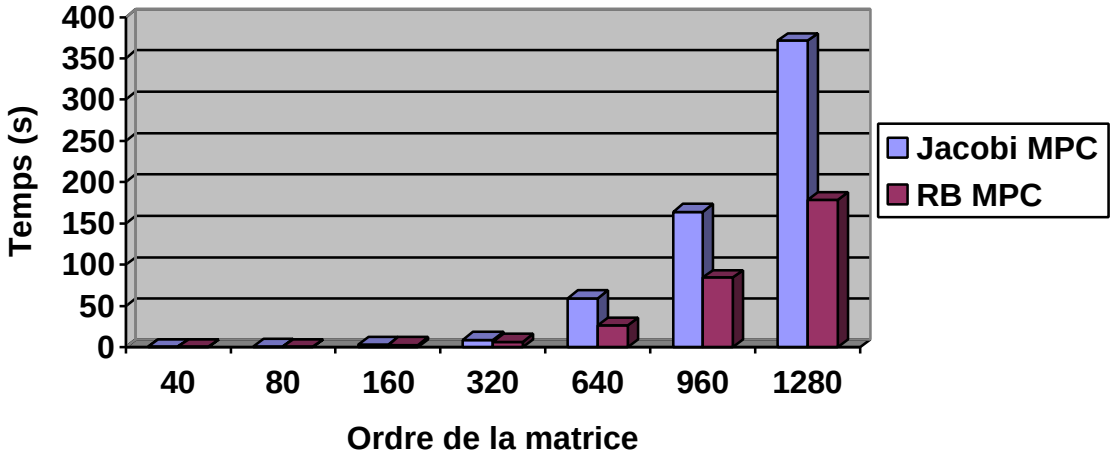
III.1 Comparaison entre les méthodes Jacobi et R&B

Tableau de résultat pour la Jacobi et R&B.

NCxNL	Taille matrice (octets)	taille msg (octets)	Temps Jacobi (s)	Temps R&B (s)	Ratio (R&B/Jacobi)
40x40	6400	160	0,515018	0,510129	0,990507128
80x80	25600	320	0,986	0,612383	0,621043451
160x160	102400	640	3,166	2,665751	0,841901625
320x320	409600	1280	9,208	6,493014	0,705123335
640x640	1638400	2560	59,395	26,592453	0,447719722
960x960	3686400	3840	164,253	84,460015	0,514207422
1280x1280	6553600	5120	371,146	178,781048	0,481700217
40x40	6400	160	0,515018	0,510129	0,990507128
40x150	24000	600	0,593984	0,427997	0,720553079
40x300	48000	1200	0,956697	0,764483	0,799085813
40x600	96000	2400	1,51898	1,012288	0,666426154
40x1200	192000	4800	2,691293	1,48236	0,550798445
40x2400	384000	9600	6,681612	2,308833	0,345550295
40x4800	768000	19200	12,945067	7,734562	0,597491075
40x9600	1536000	38400	30,858797	15,75268	0,510476154
40x19200	3072000	76800	60,985076	33,908215	0,556008408
40x25000	4000000	100000	78,573253	48,953913	0,623035335
40x30000	4800000	120000	96,285655	52,527371	0,54553683

La méthode R&B donne un gain important pour la résolution de l'équation de Laplace. On gagne 29% pour une matrice 320x320, et 51.83% pour une matrice 12800x1280.
L'amélioration des performances est du à une convergence plus rapide et a des échanges de messages dont la taille est deux fois plus petite.

Figure 18. Temps pour Jacobi et R&B avec des matrices carrees



III.2 Comparaison entre le mode de communication bloquant et non-bloquant

L'intérêt des communications non bloquantes est de permettre une optimisation grâce au recouvrement des calculs par les communications. La résolution de l'équation de Laplace sur des matrices carrée avec seulement 4 nœuds ne rend pas compte de l'amélioration du au recouvrement calcul/communication. En effet, le temps de calcul est beaucoup trop important par rapport au temps de communication. Les tests ont donc été fait pour des matrices ayant un nombre de lignes fixe et un nombre de colonnes variable, ce qui augmente la taille des messages envoyés sans trop augmenter le temps de calcul.

Tableau des résultats pour la méthode de Jacobi

NLxNC	Taille matrice (octets)	Taille message (octets)	nombre de communication pour 4 nœuds	Jacobi Bloquant	Jacobi Non-bloquant	Ratio Non-bloquant/bloquant
40x40	6400	160	240	0,515018	0,386415	0,750294164
40x150	24000	600	900	0,593984	0,626242	1,05430786
40x300	48000	1200	1800	0,956697	0,944365	0,987109816
40x600	96000	2400	3600	1,51898	1,643132	1,081733795
40x1200	192000	4800	7200	2,691293	2,604254	0,96765904
40x2400	384000	9600	14400	6,681612	5,894378	0,882179031
40x4800	768000	19200	28800	12,945067	11,913407	0,920304777
40x9600	1536000	38400	57600	30,858797	27,818523	0,901477883
40x19200	3072000	76800	115200	60,985076	57,273259	0,93913565
40x25000	4000000	100000	115200	78,573253	75,857767	0,965440072
40x30000	4800000	120000	115200	96,285655	89,081033	0,925174503

Figure 19. Temps en secondes pour une matrice de 40 lignes

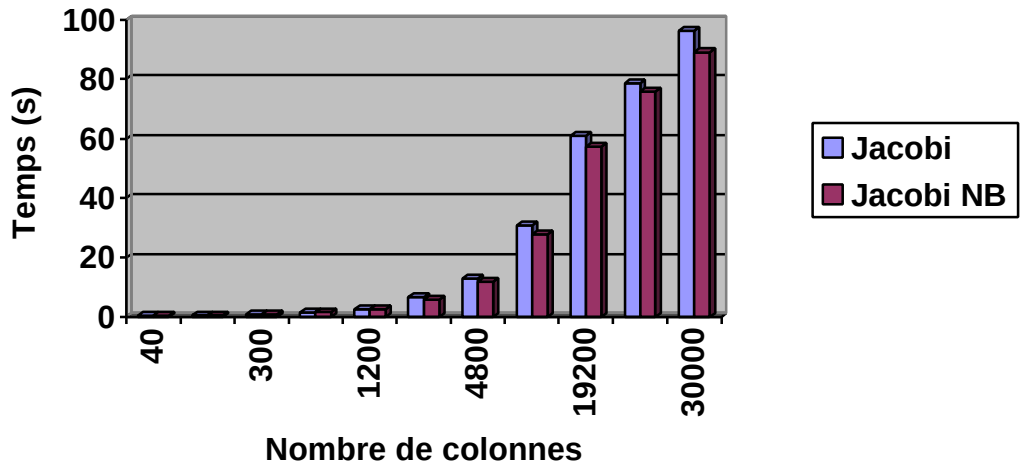


Tableau des résultats pour la méthode R&B

NLxNC	Taille matrice (octets)	Taille message (octets)	Nombre de communication pour 4 nœuds	R&B Bloquant	R&B Non-bloquant	Ratio Non-bloquant/bloquant
40x40	6400	160	9000	0,510129	0,268897	0,52711569
40x150	24000	600	108000	0,427997	0,455011	1,063117265
40x300	48000	1200	108000	0,764483	0,584503	0,764572921
40x600	96000	2400	108000	1,012288	0,853658	0,843295584
40x1200	192000	4800	108000	1,48236	1,288008	0,868890148
40x2400	384000	9600	108000	2,308833	2,351269	1,018379848
40x4800	768000	19200	108000	7,734562	6,061121	0,783641142
40x9600	1536000	38400	108000	15,75268	12,549522	0,796659489
40x19200	3072000	76800	108000	33,908215	28,439469	0,838719142
40x25000	4000000	100000	108000	48,953913	38,042486	0,777108175
40x30000	4800000	120000	108000	52,527371	44,21689	0,841787608

Les communications non-bloquantes permettent donc d'améliorer les performances de manière significative. Dans le cas du banc de test avec des matrices ayant seulement 40 lignes et plus de 600 colonnes , le temps de calcul est faible par rapport au temps de communication. Pour une matrice 40x40, le gain atteint presque les 50%.

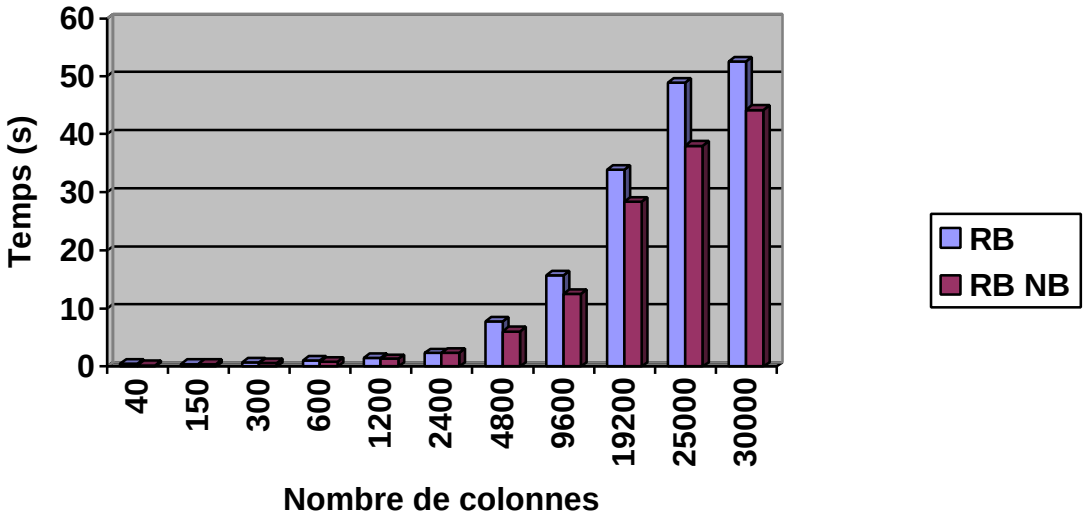


Figure 20: Temps en secondes pour une matrice de 40 lignes

III.3 Comparaison entre MPI-MPC et MPI-ETH

Des mesures ont été faites aussi pour MPI Standard sur Ethernet 100Mb/s. Les deux tableaux suivants donnent les résultats obtenus.

Tableaux des résultats pour la méthode de Jacobi.

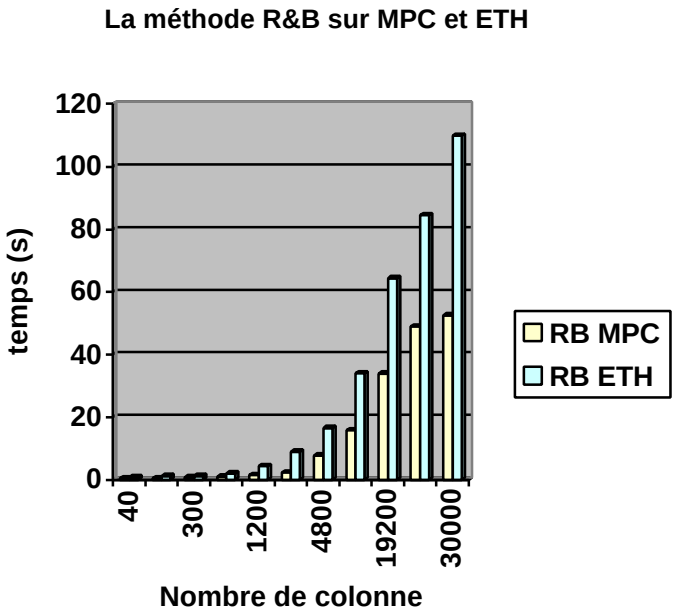
NL	NC	Taille Matrice (octets)	niter	Taille msg (octets)	tps calcul par nœud (s)	ETH	MPC	Ratios
40	40	6400	1250	160	0,515	0,672091	0,515018	1,30498546
40	150	24000	1900	600	0,243423	1,240647	0,593984	2,08868757
40	300	48000	1900	1200	0,50614	2,004544	0,956697	2,09527572
40	600	96000	1900	2400	0,969609	3,7691	1,51898	2,48133616
40	1200	192000	1900	4800	1,960642	8,0767	2,691293	3,0010482
40	2400	384000	1900	9600	3,94465	15,428413	6,681612	2,30908544
40	4800	768000	1900	19200	8,164813	31,613039	12,945067	2,44209157
40	9600	1536000	1900	38400	22,8363	61,943794	30,858797	2,00733016
40	19200	3072000	1900	76800	45,247883	117,798947	60,985076	1,93160286
40	25000	4000000	1900	100000	58,73641	157,48376	78,573253	2,00429222
40	30000	4800000	1900	120000	71,904568	186,579038	96,285655	1,93776568

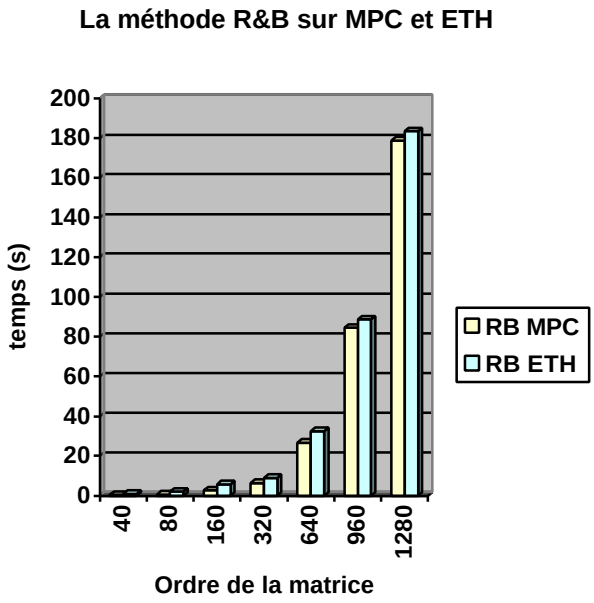
Tableaux des résultats pour la méthode de Jacobi

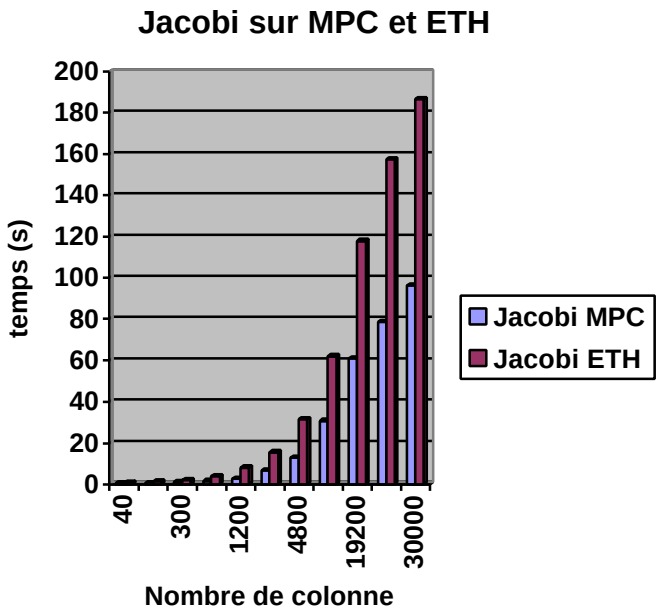
NC	NL	Taille matrice (octets)	niter	taille message (octets)	tps calcul par nœud	ETH	MPC	Ratio (MPC/ETH)
40	40	6400	1250	160	0,515018	0,672091	0,515018	0,76629206
80	80	25600	3250	320	0,986055	2,02925	0,986	0,48592091
160	160	102400	5700	640	3,956026	5,7868	3,166	0,54716683
320	320	409600	3650	1280	8,431654	10,151427	9,208	0,90709789
640	640	1638400	3650	2560	57,806665	59,951078	59,395	0,99072969
960	960	3686400	3650	3840	159,000448	168,071245	164,253	0,97728087
1280	1280	6553600	3650	5120	358,435965	368,502945	371,146	1,00717207

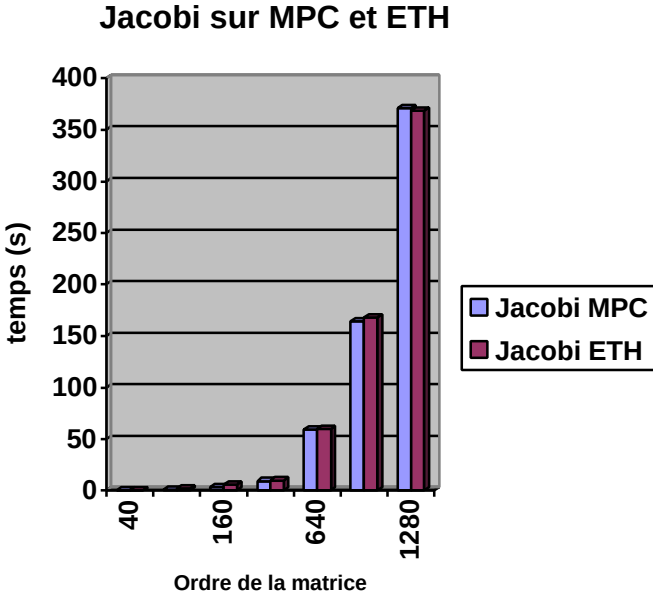
Les résultats et leurs comparaisons MPI-MPC et MPI-ETH sont représentée dans les graphiques suivants.

On peut faire le même type de commentaire que pour les précédentes comparaisons. Dans le cas où le temps de communication est important, les performances de la machine MPC sont nettement meilleurs.









Chapitre 5 : Performances avec les NAS Benchmarks

I. Introduction

Développés au centre de recherche de la NASA, les NAS Benchmarks ont été largement acceptés comme des standards pour l'évaluation des performances des supercalculateurs.

La première version, NPB1, développé en 1991 s'adressait surtout aux supercalculateurs vectoriels. Elle a montré quelques limites. En effet, l'implémentation est très souvent optimisée pour chaque machine spécifique, ce qui donne des performances qui ne peuvent être atteintes par les utilisateurs, c'est à dire la communauté scientifique. Le jeu de la concurrence fait que ces modifications spécifiques sont souvent gardées secrètes. Les résultats donnent donc plus d'informations sur les travaux d'optimisations sachant que la performance atteinte n'est qu'une faible indication pour des exécution réelles. D'autres part, les calculs et la taille de la mémoire requise ne rendent pas compte de l'accroissement des capacités des nouvelles machines.

Cette premières version, ainsi que d'autres benchmarks classiques comme Linpack ou Livermore Loops ne sont pas appropriés pour évaluer les performances des machines parallèles.

L'émergence de standardisation dans le calcul parallèle comme MPI et HPF au niveau software et l'architecture MIMD au niveau hardware, a permis le développement d'une deuxième version des NAS Benchmark adaptée aux machines parallèles. NPB2 utilise MPI. C'est donc cette version qui a été utilisée pour la machine MPC.

II Utilisation de MPI dans les NAS Benchmarks.

Pour mesurer et analyser les performances au niveau des couches hautes de la machines MPC, et donc de l'environnement MPI, il est utile d'étudier l'utilisation de cette librairie de passage de message dans les NAS Benchmark qui seront exécutés.

Relativement peu de fonctions MPI sont utilisés par les NAS. Dans l'ensemble, elle sont aux nombres de 20 : MPI_Abort, MPI_Allreduce, MPI_Alltoall, MPI_Alltoallv, MPI_Barrier, MPI_Bcast, MPI_Comm_dup, MPI_Comm_rank, MPI_Comm_size, MPI_Comm_split, MPI_Finalize, MPI_Init, MPI_Irecv, MPI_Isend, MPI_Recv, MPI_Reduce, MPI_Send, MPI_Wait, MPI_Waitall et MPI_Wtime.
Cette remarque n'est d'ailleurs pas propre aux NPB2.3, la plupart des tutoriaux sur MPI (Gropp par exemple) montre que beaucoup de programmes parallèles peuvent être écrits avec seulement 6 fonctions basics.

Une étude sur l'utilisation de MPI dans les NAS [10] montre que 89% des fonctions MPI appelées sont des Send et des Recv bloquant ou non-bloquant. Il n'y a pas d'envoi ou de réception bufferisé.

Fréquence des appels des fonctions MPI dans les NAS Benchmarks

Fonction MPI	IS	non-IS
MPI_ALLREDUCE	**30.5 %**	0.1 %
MPI_ALLTOALL	**30.5 %**	0.0 %
MPI_ALLTOALLV	**24.4 %**	0.0 %
MPI_BCAST	0.0 %	0.0 %
MPI_IRECV	3.1 %	11.0 %
MPI_ISEND	0.0 %	7.8 %
MPI_RECV	0.0 %	**33.7 %**
MPI_SEND	2.3 %	**36.8 %**
MPI_REDUCE	6.1 %	0.0 %
MPI_WAIT	3.1 %	6.7 %
MPI_WAITALL	0.0 %	3.9 %

Au niveau de la taille des messages, 74% des messages ont une taille de 600, 640, 1240 ou 2480 bytes. Ce qui implique que les messages courts dominent dans le trafic réseaux des NPB.

La taille maximum d'un message échangé est de 128Mb. Il s'agit du benchmark FT lorsqu'il est exécuté sur 2 nœuds, un message de 128Mb est échangé avec la primitive all-to-all. Le plus petit message est de 4 bytes, principalement utilisé par la primitive MPI_Bcast.

Une autre remarque intéressante sur l'utilisation des fonctions MPI concerne la différence entre le Benchmark IS et les autres. Premièrement, IS effectue des calculs sur les entiers alors que les autres utilisent des nombres flottants. Deuxièmement, le code du benchmark IS est dominé par les fonctions de réductions et de communications all-to-all, par opposition aux autres Benchmarks ou les simples envois et réceptions sont majoritairement utilisés.

Taille des NAS Parallèle Benchmark, Class A.

Benchmark	Taille du problème	Mémoire en Million de mots
Embarrasingly parallel (EP)	2^{28}	1
Multigrid (MG)	256^3	57
Conjugate gradiaent (CG)	$\approx 2*10^6$	12
3-D FFT PDE (FT)	256^2*128	59
Integer Sort (IS)	2^{23}	26
LU solver (LU)	64^3	8
Pentadiagonal solver (SP)	64^3	6
Block tridiagonal solver (BT)	64^3	6

III Adaptation des NAS pour la machine MPC.

Le compilateur mpif77 spécifique pour l'environnement de programmation MPI de la machine MPC n'est pas encore disponible pour le moment.
Rappelons que les spécifications de MPI rendent ce langage indépendant du langage avec lequel il est lié [8]. MPICH est implémenté en C, et les implémentations en Fortran ne sont que des simples appels aux routine en C. Grâce aux programmes bfort et f2c, il est possible de générer une interface correcte qui permette la compilation d'une application en MPI Fortran.

Cependant, le portage de MPI sur MPC, étant toujours en cours de développement, cette interface n'était pas disponible.

Les principaux noyaux des NAS Benchmarks sont écris en fortran, langage qui est très utilisé par la communauté scientifique pour le calcul numérique. Hormis le programme IS dont la version originale est en C, aucune version des NAS écrite en MPI C n'a été trouvée. Il a donc été décidé d'écrire, pour chaque NAS, une version en C avec MPI, et cela sans perte de performance.

III.1 Ecriture de la version MPI C des NAS Benchmarks

III.1.1 Utilisation des versions OpenMP C et MPI Fortran

Un code en Fortran MPI peut être traduit de manière automatique en utilisant des programmes tels que f2c. Pour des programme classique, f2c est assez efficace, d'après ce que j'ai pu en lire. Mais pour des programmes scientifiques comme les NAS , son utilisation ne simplifie pas vraiment les choses. Le code en C qui est généré automatiquement, peut ne pas avoir les mêmes caractéristiques que celui écrit « à la main ». De plus, pour obtenir un code cohérent et aussi efficace, il faut forcément retoucher le résultat de la traduction par f2c.

Je me suis aperçut qu'utiliser une méthode automatique aurait été presque aussi fastidieux que de faire la traduction « a la main », et pour obtenir un code beaucoup moins propre. Les principales difficultés viennent de la manipulation des matrices et du passage des paramètres entre les fonctions. Nous verrons cela par la suite.

Il existe une version parallèle des NAS écrite en Open-MP C par le Omni RCWP OpenMP Compiler Project . OpenMP est utilisé pour la programmation parralèle à mémoire partagée. Cette parallélisation se fait par des MACROS qui sont interprétées par le compilateur OpenMP pour fournir un code exécutable en parallèle. Ces NAS Benchmarks OpenMP ont donc servit de versions séquentielles. Il a fallut ensuite les paralléliser pour l'environnement de programmation parallèle par passage de message MPI.

Les NAS Benchmarks utilisés pour l'évaluation des performances de la machine MPC, ont donc été écrits à l'aide des versions MPI Fortran et OpenMP C.

J'ai réécrit tous ce qui concerne la décomposition des problèmes sur les différents processus et les échanges de messages.

III.1.2 La traduction en langage C

Il y a plusieurs points importants à prendre en compte pour la conversion.

Les indices des tableaux en fortran commencent par défaut par 1. Lors de la déclaration du tableau, il est même possible de spécifier l'indice du début et l'indice de fin, ce qui ne peut se faire dans un programme C. Les traductions automatique effectue un décalage d'indice. Dans ce cas, la taille du tableau dans le code C est supérieur a celle de son

équivalent en Fortran. S'il s'agit d'un tableau, ce n'est pas trop pénalisant , mais il en est tout autre pour une matrice de flottants de dimension 3 par exemple.

D'autres part, le fortran et le C ont deux manières différentes de ranger les éléments des matrices en mémoire. Cela à une grande importance pour les performances des boucles de calcul qui parcourent les éléments d'une matrice. Pour une matrice u[n1][n2], le tableau u[0] aura ses éléments alignés en mémoire pour un code en C, et ce sera le tableau u[1..n1][0] pour un code en Fortran. Pour passer d'un programme C en Fortran, il faut donc inverser l'ordre des boucles ou bien déclarer différemment les matrices en inversant les dimensions comme le montre les deux codes ci-dessous.

Langage Fortran	Langage C
double precision **u(n1,n2,n3)**	double **u[i3][i2][i1] ;**

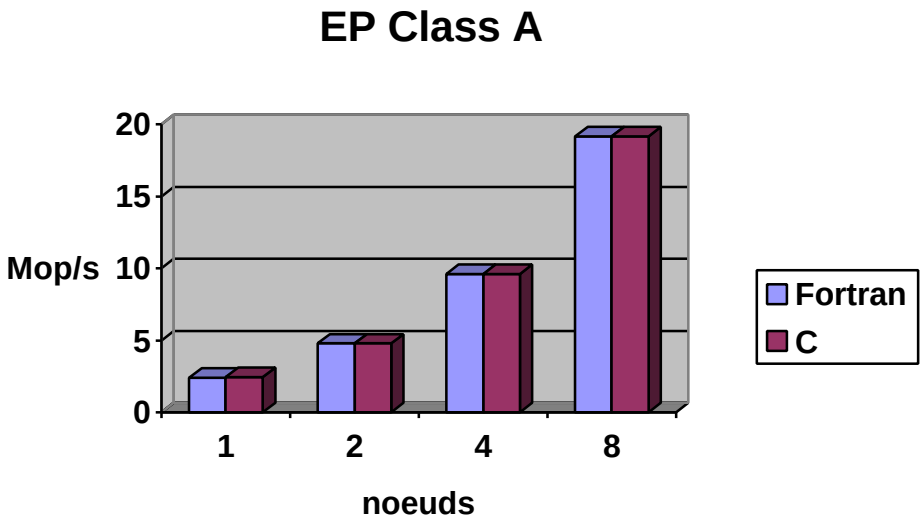
indx = 1
 	
do **i3**=1, n3
  do **i2**=1,n2
  do **i1**=1,n1
  u(i1,i2,i3) = **buff(indx,buff_id)**
  indx=indx+1
  enddo
  enddo
enddo
	indx = 0 ;
	
	for (**i3**=0 ; i3<n3 ; i3++) {
	  for (**i2**=0 ; i2<n2 ; i2++) {
	  for (**i1**=0 ; i1<n1 ; i1++) {
	  u[i3][i2][i1] = **buff[buff_id][indx]** ;
	  indx = indx+1 ;
	  }
	  }
	}

En plus de la réécriture d'une partie du code, des modifications ont été apportées au programme setparams (NPB2.3/common/setparams.c) pour que celui-ci génère des fichiers d'interfaces en C (setparams.h). Ce fichier d'interface contient toute les constantes définissant la taille du problème en fonction du nombre de processus. Le programme setparams.c prend en argument la CLASS et le nombre de processus et génère le fichier setparams.h.

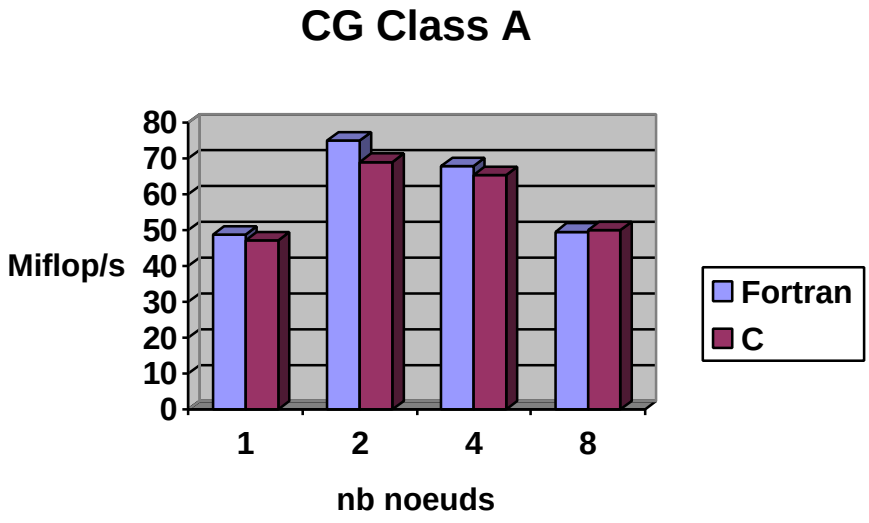
Les Benchmarks qui ont été traduits en C sont : CG, EP, MG, FT. Avec IS, nous disposons donc de 5 Benchmarks en C.

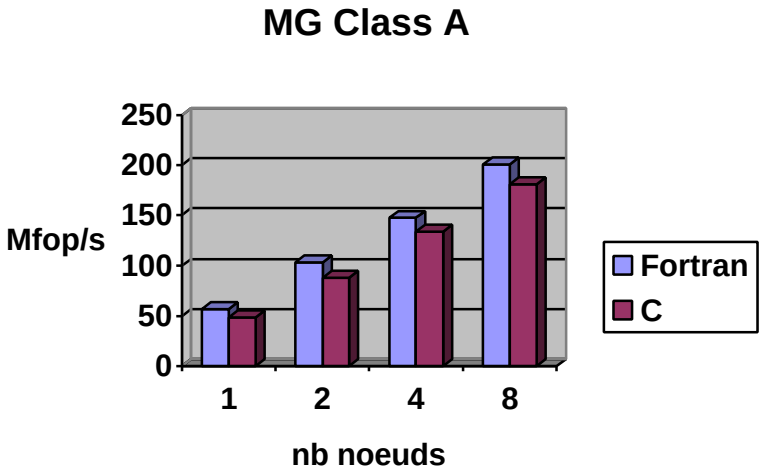
III.2 comparaison des mesures entre MPI fortran et MPI C

Les mesures suivantes ont été effectuées sur les 8 nouveaux nœuds de la machine MPC. Chaque nœuds est constitué de bi-pro Pentium III 1Ghz avec 1Go de mémoire. Pour des raisons de problème de configuration, les applications MPI ne peuvent pas encore s'exécuter en utilisant le réseau HSL. Il est seulement possible d'utiliser MPI-Standart sur Ethernet 100Mb/s.
La latence pour MPI standard sur Ethernet est de 65µs et le débit maximum 74Mb/s. Le demi-débit est atteint pour une taille de 512o.



Pour le Benchmark Embarassingly Parallel, il n'y a aucune perte de performance. Il est important de dire que ce benchmark utilise un minimum de communication. Les échanges de messages ne servent qu'à distribuer le problème entre les processus, puis à récupérer l'ensemble des résultats des différents nœuds à la fin.





Il y a une légère perte de performance pour la version en C des benchmarks CG et MG. Pour MG, cela vient du programme séquentiel qui effectue des allocations dynamiques des matrices (. Les résultats montrent que la parallélisation est bonne puisque la différence de performance se trouve déjà lors de l'exécution du programme sur un seul nœud.

Il n'y a pas eu de comparaisons pour FT, car le compilateur f90 n'a pas été installé sur la machine MPC.

IV Mesures des performance sur la machine MPC

Les mesures ont été effectuées sur les quatre nœuds bi-pro PII 350Mhz avec 128Mo de mémoire. Cette faible capacité mémoire est un facteur limitant.

IV.1 Tableaux des mesures

	Class	**Nb proc**	**Temps**	**Mflop/s**	**Mflop/s/proc**
CG	S	2	0.84	79.36	39.68
CG	S	4	0.51	129.91	32.48
CG	A	2	26.75	55.94	27.97
CG	A	4	15.65	95,6	23.90

	Class	**Nb proc**	**Temps**	**Mflop/s**	**Mflop/s/proc**
EP	S	2	18.41	1.82	0.91
EP	S	4	9.22	3.64	0.91
EP	A	2	294.86	1.82	0.91
EP	A	4	147.09	3.65	0.91

	Class	**Nb proc**	**Temps**	**Mop/s**	**Mop/s/proc**
IS	S	2	0.08	8.44	4.22
IS	S	4	0.06	10.80	2.70
IS	W	4	0.93	11.31	2.83

	Class	Nb proc	Temps (s)	Mflop/s	Mflop/s/proc
MG	S	2	0.15	50.75	25.38
MG	S	4	0.09	85.18	21.3
MG	W	2	9.71	62.65	31.32
MG	W	4	5.8	104.81	26.2
MG	A	4	575.36	6.77	1.69

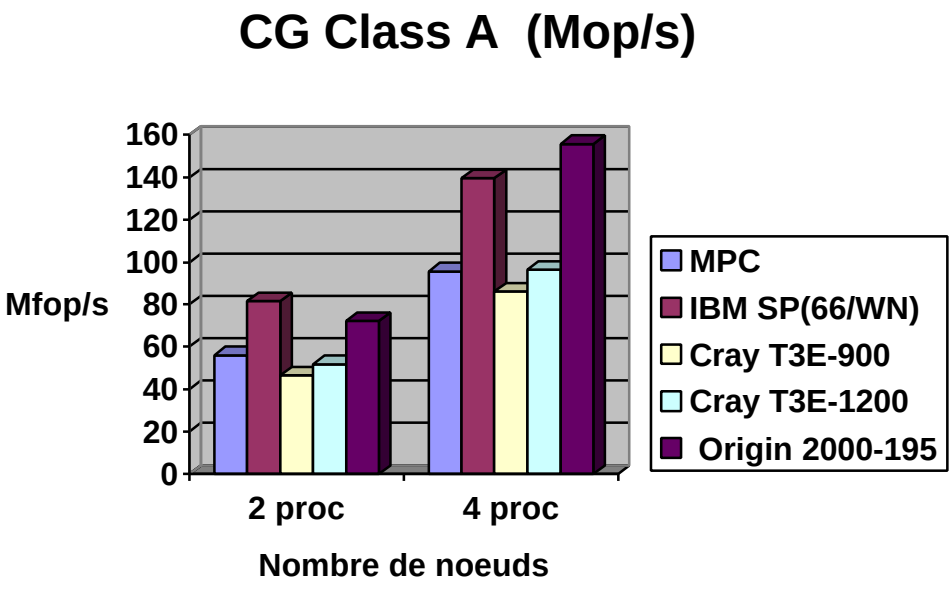
	Class	Nb proc	Temps (s)	Mflop/s	Mop/s/proc
FT	S	2			
FT	S	4	1.82	97.40	24.35

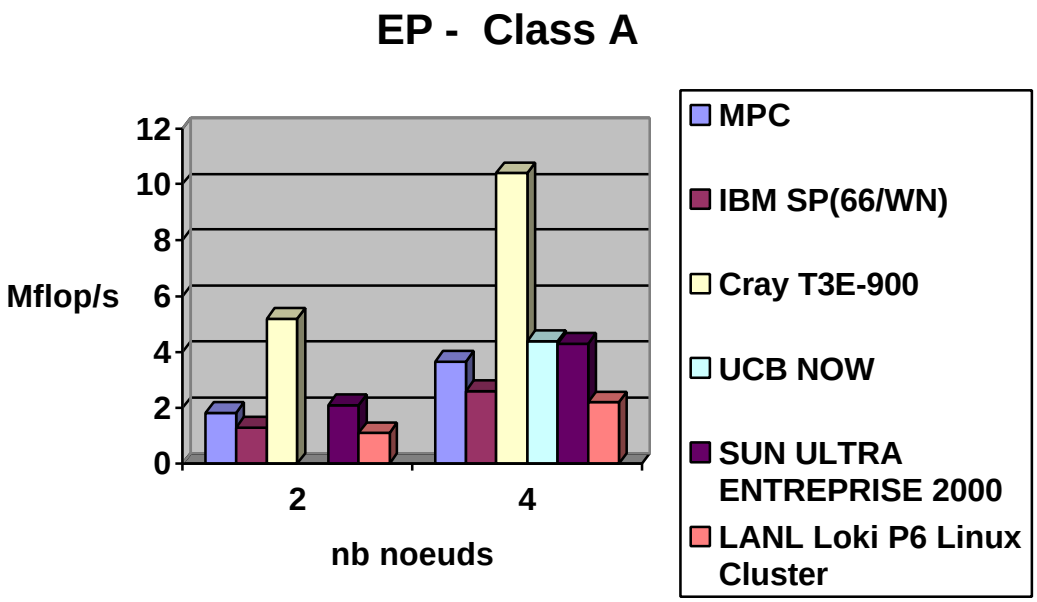
Pour la class A, seul les benchmark CG et EP tournent à l'heure actuelle sur l'environnement MPI-MPC. Le problème vient soit de la taille trop grande des messages échangés, soit de problème de mémoire. Pour un message de très grande taille, le nombre de descripteurs DMA associés au message est plus grand que la taille de la table qui doit les contenir.
Au niveau de la mémoire, il faut remarquer que les données qui sont envoyées ne peuvent pas être swappées lorsque la traduction d'adresse est conservée dans le cache (pour maintenir la cohérence). Le programme peut se trouver parfois complètement bloqué.

IV.2 Comparaison avec d'autres machines

Seul les performances pour les Class A, B, C sont disponible sur le site officiel des NAS (www.nas.nasa.gov). Je n'ai donc pu faire des comparaisons qu'avec les Benchmark CG et EP.

Comparaison des résultats avec les performances d'autres machines parallèles . On peut voir que la machine MPC a des performances meilleurs que celles de la Cray T3E.





Pour le Benchmark EP, on voit que sur 4 processus, les performance de la machine MPC se rapproche de celles de UCB NOW qui est un cluster de référence, et se situe nettement au dessus du LANL Loki P6 Linux Cluster.

Chapitre 6 : Conclusions

I Bilan du stage

Pour évaluer le portage de la couche MPI sur MPC, il m'a fallu comprendre les principes généraux de MPI afin de voir comment a été réalisé le portage de MPI sur MPC. Après avoir compris les mécanismes de communication MPI-MPC, j'ai tenté d'analyser du point de vu des performances les différentes options pour la couche PUT et la couche MPC.

Le portage de la couche MPI sur la machine MPC étant en cours de développement et d'optimisation, il a fallu m'adapter aux différentes modifications qui ont été apporté au cours de mon stage.

Les séries de tests et d'évaluations ont permis de déceler quelques problèmes qui ont été corrigés par la suite. En effet, aucune véritable application n'avait été lancée. Mes séries de tests ont pu mettre en évidence certains problèmes qui n'apparaissaient pas lors de simples communications.

La conversion des NAS Benchmarks en C a été longue et fastidieuse. Elle a permis d'obtenir des codes qui pourrons être très utiles pour étudier les performances des nouveaux nœuds de la machine MPC. Je pense que pour satisfaire l'ambition principale du projet MPC, c'est à dire de fournir une puissance de calcul aux utilisateurs externes, il faut se poser la question de l'implémentation de l'interface Fortran pour l'environnement MPI-MPC.

En plus de l'approfondissement de mes connaissances en programmation parallèle, ce stage m'a permit de découvrir le monde de la recherche. De plus, j'ai assisté à plusieurs réunions dans le cadre du projet MPC. Il s'agit de réunions qui regroupent tous les chercheurs de l'équipe dont les travaux de recherche sont en liaison avec la machine MPC. Ces réunions sont au nombre de une par mois environ.

II Perspectives

La perspective à court terme est de fournir un environnement de programmation fiable aux utilisateurs potentiels. Les résultats obtenus donnent une première validation de MPI sur MPC au niveau des primitives de communication. Quelques points posent cependant problème pour l'exécution de grosses applications. L'amélioration du matérielle avec les 8 nouveaux nœuds offrira une machine très performante comme le montrent les résultats des NAS Benchmarks avec MPI-ETH. La perspective suivante est de permettre à plusieurs tache de s'exécuter sur un même nœud.

Bibliographie

I Sites Internet

Site du LIP6 http://www.lip6.fr
Site du département ASIM http://www-asim.lip6.fr
Site de la machine MPC http://mpc.lip6.fr

Site des NAS Benchmark http://www.nas.nasa.gov

II Documentation et ouvrage

[1] Protocol and Performance Analysis of the MPC Parallel Computer, UPMC/LIP6
O.Gluck, A Zerrouki, J.L Desbarbieux, A Fenyo, A. Greiner, F. Wajsburt, C. Spasevski, F. Silva, E. Freyfus.

[2] Evolutions des mécanismes de communication par passage de messages dans les architectures parallèles, LRI
F. Cappello

[3] Architectures parallèles à partir de réseaux de stations de travail : réalités, opportunités et enjeux, LRI
F. Cappello

[4] Message-Passing Performance of Various Computers, ORNL.
J. Dongarra, T. Dunigan.

[5] Parallel Computer Architecture, Morgan Kaufmann publishers.
David E. Culler, Jaswinder Pal Singh

[6] In search of Clusters.
Gregory F. Pfister

[7] Parallel Numerical Algorithms.
T .L. Freeman, C. Phillips

[8] A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard
W. Gropp, E. Lusk, N. DOSS, A. Skjellum.

[10] The Use of the MPI Communication Library in NAS Parallel Benchmark
Theodore B. Tabe, Quentin F. Stout.