M1 / MIHPS / U13 / TD2
Techniques d'optimisation de la parallélisation
Premières mesures avec MPI

1) Mise en place de l'environnement de travail.

1.a) Installation de mpich.

Pour ce TP et les suivants, nous allons utiliser la librairie MPI mpich que vous pouvez télécharger à l'adresse www.mcs.anl.gov. Nous travaillerons avec la dernière version stable (version 3.0.3). Vous pouvez aussi utiliser OpenMPI de la même manière.

Commencez par installer cette version dans votre dossier HOME, allez dans le dossier de téléchargement et extrayez l'archive :

```
cd Téléchargement
tar -xvzf mpich-3.0.3.tar.gz
cd mpich-3.0.3
```

Configurez mpich pour l'installer dans le dossier *$HOME/usr*. Nous n'allons pas utiliser fortran, vous pouvez donc désactiver son support dans le cas ou gfortran ne serait pas disponible sur votre station de travail.

```
./configure --prefix=$HOME/usr --disable-f77 --disable-fc
```

La méthode utilisant « mpd / mpdboot » (nécessite une option à la compilation dans les nouvelles versions) est plus simple, car plus besoin de clé ssh. Mais elle ne marche pas sur les machines de l'université (problème de détection d'IP à priori).

Compilez et installez mpich :

```
make -j
make install
```

Configurez votre environnement pour qu'il utilise cette installation, pour cela, ajoutez à votre fichier *~/.profile* les lignes suivantes :

```
export PATH=$HOME/usr/bin:$PATH
export LD_LIBRARY_PATH=$HOME/usr/lib:$LD_LIBRARY_PATH
```

Une version a été pre-compilée dans le dossier :

```
/users/doinf-2012/20908769/usr
```

1.b) Première exécution.

Pour vérifier que notre installation fonctionne, on peut simplement exécuter un programme MPI « hello world » ou bien la commande « hostname » qui a l'intérêt de montrer clairement quel hôte est utilisé :

```
mpirun -np 2 hostname
```

Normalement, vous devriez obtenir deux lignes correspondant aux deux processus qui ont été créés.

1.c) Création de clé ssh

À moins d'utiliser l'ancienne méthode basée sur « mpd » vous aurez besoin de vous connecter en ssh sur les différents nœuds utilisés. L'utilisation de clé ssh nous permettra de ne pas avoir une demande d'authentification à chaque lancement. Commencez par créer vos clés si ce n'est pas déjà fait :

```
ssh-keygen
```

Autorisez la clé (crée ou met à jour *$HOME/.ssh/authorized_keys*) :

```
ssh-copy-id localhost
```

Déverrouillez la clé pour la session courante (à refaire sur chaque session ayant besoin d'utiliser les clés ssh) :

```
ssh-agent -- $SHELL
ssh-add
```

1.d) Exécution sur plusieurs noeuds.

Créez un fichier contenant la liste des hôtes que vous voulez utiliser (par exemple hosts.txt). Attention, tous ces hôtes doivent avoir un accès ssh et partager leur dossier HOME (via NFS par exemple). Sur les stations de l'université, utilisez les IP au lieu des noms de domaines :

```
hostname1.domainname1
hostname2.domainname1
hostname3.domainname1
```

Tenter l'envoi de la commande *hostname* comme précédemment pour vous assurer que MPICH répartit bien les tâches.

```
mpirun -np 2 -f hosts.txt hostname
```

Félicitation, vous venez de mettre en place votre clusteur personnel !

Stations utilisables :

centre.ens.uvsq.fr	*Bi-socket Intel E5310 - 4 coeurs @ 1.6Ghz*
aquitaine.ens.uvsq.fr	*Bi-socket Intel E5310 - 4 coeurs @ 1.6Ghz*
alsace.ens.uvsq.fr	*Bi-socket Intel E5310 - 4 coeurs @ 1.6Ghz*

ATTENTION, les noms de domaines ne fonctionnent sur les machines de l'université,par contre cela fonctionne avec les IP des stations.

1.e) Gestionnaire de tâches

Pensez-vous qu'un clusteur réel soit utilisé de la sorte ? Quel(s) problème(s) cela pose pour ce type d'usage ? Donner d'une solution logiciel traitant cette question et largement utilisé dans le domaine du HPC ?

2) Mesure d'un pingpong.

Dans cette partie, vous allez réaliser vos premières mesures de performance en prenant le pingpong comme benchmark afin de se familiariser avec les temps caractéristiques d'une communication MPI.

2.a) Implémentation.

Pour commencer, implémentez un pingpong entre deux tâches MPI. Pour cela, nous allons émettre un message (MPI_Send) sur le nœud 0. Le nœud 1 va recevoir ce message (MPI_Recv) et le renvoyer en réponse.

Nous ferons en sorte de pouvoir tester différentes tailles de messages, vous ferez donc en sorte que la taille du message puisse être donnée en argument du programme.

```
if (rank == 0)
{
     MPI_Send(buffer,size,MPI_CHAR,1,0,MPI_COMM_WORLD);
     MPI_Recv(buffer,size,MPI_CHAR,1,1,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
} else {
     MPI_Recv(buffer,size,MPI_CHAR,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
     MPI_Send(buffer,size,MPI_CHAR,0,1,MPI_COMM_WORLD);
}
```

Vérifiez que votre code fonctionne.

2.b) Mesure de temps.

Nous allons maintenant mesurer le temps d'un allez retour, pour cela nous allons utiliser la fonction MPI_Wtime() et placer un point de mesure avant et après les deux échanges sur le nœud 0.

```
if (rank == 0)
{
      t = MPI_Wtime();
      MPI_Send(buffer,size,MPI_CHAR,1,0,MPI_COMM_WORLD);
      MPI_Recv(buffer,size,MPI_CHAR,1,1,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      t = MPI_Wtime() - t;
      printf("Time : %f\n",t);
} else {
      MPI_Recv(buffer,size,MPI_CHAR,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      MPI_Send(buffer,size,MPI_CHAR,0,1,MPI_COMM_WORLD);
}
```

Pourquoi ne peut-on pas mesurer directement le temps d'un seul envoi entre le nœud 0 et 1 ?

Les deux machines peuvent ne pas avoir leurs horloges synchronisées, il est donc préférable de mesurer un allez-retour sur une même base de temps.

Effectuez plusieurs mesures en utilisant des blocs de 32o. Que constatez-vous ?

Le temps d'exécution varie énormément (observé de 0,00071 à 0,06205 sur mon portable).

Qu'avons nous réellement mesuré ; quelle erreur avons-nous commise lors de notre mesure ? Pour vous guider, on se propose d'ajouter un *sleep(1)* qui sera exécuté uniquement sur le nœud 1 juste après MPI_Init. Refaites la mesure. Qu'observe-t-on ? Comment peut-on corriger ce problème ?

Le temps a augmenté. On n'a pas tenu compte du temps d'initialisation qui peut varier d'un nœud à l'autre. On peut corriger le problème en mettant une barrière avant la mesure pour s'assurer que les deux nœuds sont prêts.

Appliquez la correction et effectuez une nouvelle mesure. Qu'observez-vous ?

Le temps d'exécution est globalement plus stable et plus court. Mais les variations restent.

2.c) Répétitions

Lorsque l'on désire réaliser une mesure, il est important de ne pas oublier que notre environnement ne nous permet pas de reproduire les conditions exactes entre chaque exécution. D'autre part, la mesure de temps elle-même est entachée d'une incertitude. Pour régler ce problème, on préfère en général répéter une mesure et calculer une moyenne ou médiane.

On ajoute des boucles autour des couples MPI_Send/MPI_Recv. De l'ordre de 1 000 00 répétitions pour obtenir un temps d'exécution d'une seconde en local.

Appliquez un nombre de répétitions que vous jugerez correctes pour obtenir une mesure propre. Quelle est l'ordre de grandeur d'un échange de ce type entre deux processus MPI fonctionnant sur la même machine (local) ?

1 microseconde en local, et 137 en distant (TCP), 7 en IB.

2.d) Communication internœud

Pour l'instant nous avons réalisé des communications locales. Ces dernières sont en réalité optimisées en évitant de passer par une communication TCP. Nous allons maintenant exécuter le même benchmark sur deux nœuds distincts.

Quel est le nouvel ordre de grandeur ?

1 microseconde en local, et 137 en distant (TCP), 7 en IB.

2.e) Impact de la taille du message.

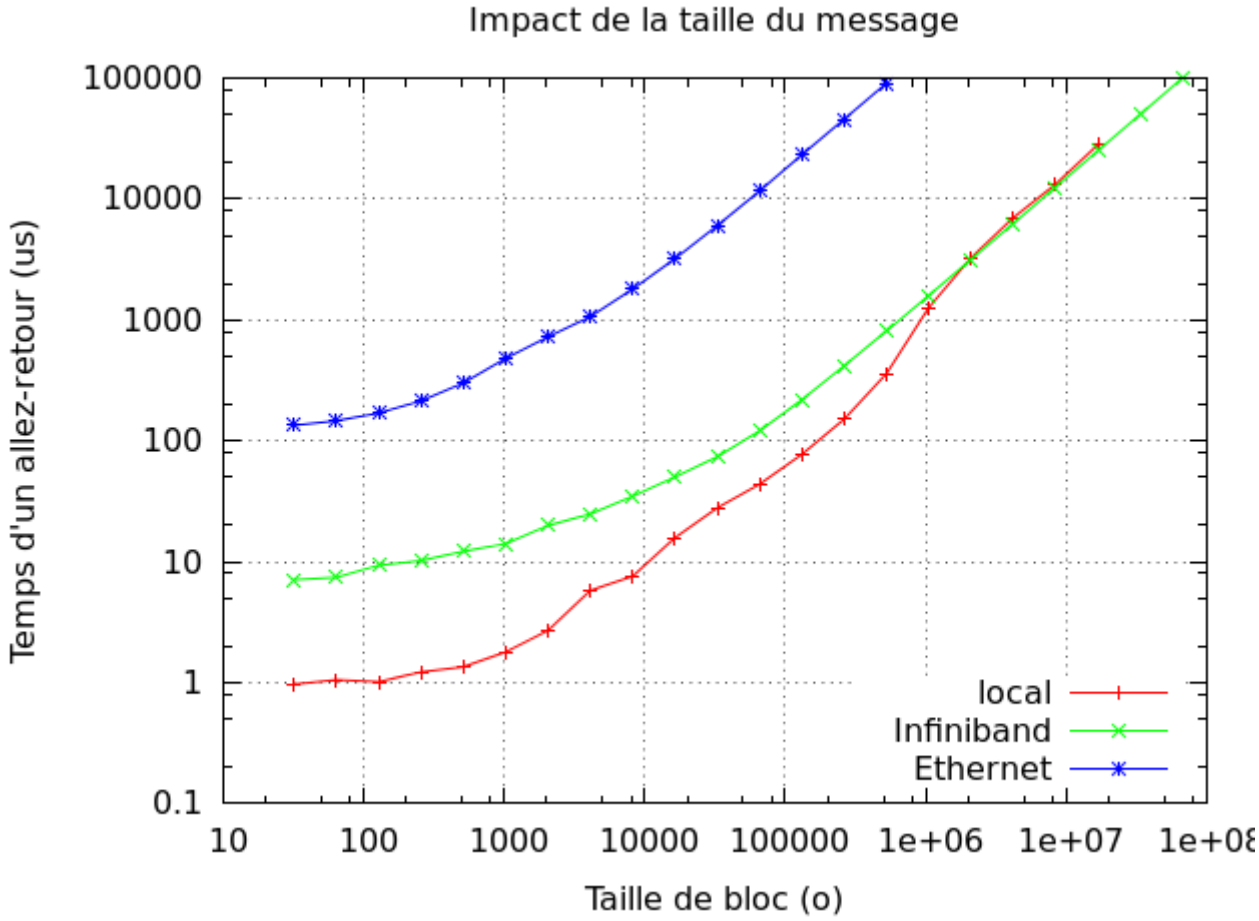
Effectuez maintenant la même mesure pour différentes tailles de messages (de 32o à 10Mo par exemple). Effectuez la mesure en local ou sur deux nœuds. Choisissez un échantillonnage adapté. Que constatez-vous ?

```
for tmp in $(seq 5 34)
do
```

```
      size=$(echo "2^$tmp" | bc);
      mpirun -np 2 ./a.out $size;
done | tee out.txt
```

Ou directement dans le programme :

```
unsigned long i ;
for (i = 5 ; i = 34 ; ++i)
{
      size = 1 << i ;
      do_ping_pong(size) ;
}
```



Le graphique pourra être généré avec gnuplot si vos données sont rangées sous forme de colonnes séparées par des espaces ou tabulations dans un fichier texte. Par exemple :

```
#Taille         Temps
32              12345
64              22222
128             33333
```

Si ces données sont dans un fichier data.txt, vous pouvez tracer simplement les valeurs en lançant gnuplot et en utilisant la commande :

```
plot 'data.txt' using 1:2 with linespoints title 'ethernet'
```

Un peu d'aide sur gnuplot : http://t16web.lanl.gov/Kawano/gnuplot/index-e.html

TODO => ajouter courbe internoeud.

On observe deux phénomènes :

1. *À petite taille, on augment très peu le temps => effet de latence (de l'ordre de la microseconde comme vu précédemment en local)*
2. *Lorsque la taille augment le temps finit par également augmenter rapidement. => effet de débit (de l'ordre de 1 à 2 Go/s en local). On peut tracer la courbe de début (taille/temps).*

2.f) Bande passante

Modifier votre graphique pour représenter la bande passante en fonction de la taille des blocs.

2.g) Regrouper ou découper ?

Au vu des résultats précédents, vaut-il mieux envoyer un groupe de données distinctes de manière séparée ou tenter de les regrouper lorsque c'est possible ? Est-ce valable pour toutes les tailles ?

Il vaut mieux grouper au maximum les données pour réduire le nombre de requêtes. Au-delà d'une certaine taille (de l'ordre de 512Ko/1Mo), il n'y a plus d'intérêt à passer trop de temps à grouper des paquets, car on atteint un régime linéaire.

3) Évaluation expérimentale de la scalabilité

Dans cette partie, nous allons essayer d'évaluer la scalabilité de divers code très simple afin de nous familiariser avec cette notion.

3.a) Construction du benchmark

Nous allons commencer avec un benchmark très simple. Notre programme MPI n'effectuera dans un premier temps aucune communication. Faites en sorte que le nœud 0 mesure le temps d'exécution du programme. Il effectuera une simple somme de deux vecteurs, sous la forme :

$$X_i^{(t+1)} = X_i^{(t)} + c$$

Avec X un vecteur de taille N et c une constante réelle. Il est clair qu'une telle équation peut être répartie sur plusieurs processus MPI sans effectuer la moindre communication. Nous allons donc simplement implémenter la méthode de somme et l'exécuter en boucle de sorte à obtenir un temps d'exécution convenable pour notre mesure.

Vous devez donc :

- Initialiser le contexte MPI
- Vous arranger pour recevoir la taille du vecteur en paramètre du programme.
- Allouer et initialiser la mémoire des deux vecteurs. Chaque nœud devra effectuer le calcul sur un vecteur de tailles N/nb_tâche.
- Faire en sorte de pouvoir mesurer le temps d'exécution comme nous l'avons fait dans le début du TP.

3.b) Speedup / efficacité

Nous allons maintenant évaluer la scalabilité de cette application.

- Rappeler la définition du speedup.
- Rappeler la définition de l'efficacité.
- Dans le cas idéal que doit ont obtenir ?
- En cas d'écart au cas idéal, que va-t-on observer et pourquoi ?
- Dans quel cas dit-on d'une application qu'elle est scalable ?

3.c) Mesure de la scalabilité forte

Que représente la scalabilité forte ? Évaluez-la sur votre application et comparez au cas idéal.

Imaginez maintenant que vous voulez l'évaluer sur 20 000 tâches, quel problème allez-vous rapidement rencontrer avec cette procédure ?

3.d) Mesure de la scalabilité faible

Que représente la scalabilité faible ? Évaluez-la sur votre application et comparez au cas idéal.

4) Impact des communications sur la scalabilité

Effectuez le même travail en testant deux modifications différentes de ce programme.

4.a) Introduction de communications

Pour introduire des communications dans notre programme, on peut considérer un cas inspiré des éléments finis. En modifiant l'équation pour obtenir :

$$X_i^{(t+1)} = \frac{X_{(i-1)}^{(t)} + 2X_i^{(t)} + X_{(i+1)}^{(t)}}{4}$$

Ici, un découpage MPI nécessite l'ajout d'une communication des éléments en frontière.

Modifiez l'application de la sorte et évaluez la scalabilité. On considère des valeurs nulles et fixes aux extrémités. Choisissez un nombre d'éléments différents pour X et réévaluez la scalabilité. Quelle remarque pouvez-vous faire quand à l'objectivité de cette mesure ?

4.b) Introduction d'une barrière

Introduisez (bien que ce soit très virtuel au vu de notre cas) une synchronisation globale (MPI_Barrier) dans la boucle de répétition et réévaluez la scalabilité de l'application. Que remarquez-vous ?

5) Remarques

Quelles remarques pouvez-vous faire quand à l'utilisation des communications et des barrières MPI en ce qui concerne la scalabilité des applications ? Que convient-il d'éviter autant que possible ?