

# Introduction à MPI

Il s'agit de s'initier à MPI et d'observer les comportements de la latence réseau. Les expériences seront d'abord menées entre votre propre machine et celle de votre voisin : **modifiez** le fichier `mymachines` pour y mettre le nom de la machine de votre voisin.

## 1 Avant toutes choses

Pour pouvoir éviter de taper son mot de passe, générez une paire clé publique/privée :

```
ssh-keygen -t dsa
```

Validez autant de fois qu'il le faut. Utilisez une passphrase vide, à moins que vous sachiez gérer un agent ssh. Enfin, autorisez l'utilisation de la clé :

```
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

Vérifiez avec un `ssh` que vous n'avez pas à taper de mot de passe. Il faudra tout de même taper `yes` pour confirmer l'identité de chaque machine la première fois que vous vous y connectez.

## 2 Découverte

Le programme `ping.c` est un exemple typique de programme MPI. C'est le même programme qui est lancé sur différentes machines (appelées *nœuds* et numérotés à partir de 0), et l'on utilise quelques fonctions pour communiquer.

- `MPI_Init(&argc,&argv)` ; permet d'initialiser la bibliothèque MPI : on lui passe l'adresse de `argc` et `argv` pour qu'elle puisse savoir comment les nœuds doivent se connecter.
- `MPI_Comm_size(MPI_COMM_WORLD,&num)` ; permet de récupérer le nombre de nœuds lancés.
- `MPI_Comm_rank(MPI_COMM_WORLD,&myid)` ; permet de récupérer son propre numéro de nœud (entre 0 et `num-1`).
- `MPI_Send(buf,N,MPI_BYTE,1,0,MPI_COMM_WORLD)` ; envoie un tableau `buf` de `N` octets (`MPI_CHAR`) au nœud 1, avec le tag 0.
- `MPI_Recv(buf,N,MPI_CHAR,0,0,MPI_COMM_WORLD,&status)` ; réceptionne un tableau `buf` de `N` caractères (`MPI_CHAR`) envoyé par le nœud 0 avec le tag 0 (il faut donc que ce soit le même que du côté émetteur). L'état de la réception est stocké dans la variable `status`.
- `MPI_COMM_WORLD` est le communicateur utilisé : il inclut tous les nœuds lancés.

On pourra bien sûr garder sous les yeux les pages de man de ces fonctions MPI, mais sur [http://mpi.deino.net/mpi/mpi\\_functions/](http://mpi.deino.net/mpi/mpi_functions/) la documentation est plus fournie et d'autres exemples de codes sont disponibles. Une documentation très détaillée est disponible sur <http://www.netlib.org/utk/papers/mpi-book/mpi-book.html> . Pour lancer ce programme MPI, un simple `make run` devrait suffire.

## 3 Latence

Pour l'instant, ce programme ne fait qu'envoyer un caractère d'une machine à une autre ("ping") et de mesurer le temps pris par les fonctions d'envoi et de réception.

Est-ce que la mesure est correcte pour autant ? (pensez aux problèmes de défaut de cache et de précision d'horloge). Corrigez le programme.

Complétez le programme pour que la deuxième machine renvoie ce caractère ("pong"). Mesurez la latence de l'aller-retour. Augmentez `N` de manière géométrique (de facteur 4) jusqu'à 1048576. Utilisez `gnuplot` pour tracer une courbe en fonction de la taille. Utilisez la commande `set logscale x` pour passer en échelle logarithmique.

## 4 Isend

Dans le code du nœud 0, remplacez `MPI_Send()` par le couple « `MPI_Isend()` puis `MPI_Wait()` » : quasiment rien n'est changé, on donne juste à `MPI_Isend` l'adresse d'un tampon de requête de type `MPI_Request`, que l'on fournit ensuite à `MPI_Wait` pour attendre la fin de la requête d'émission. Constatez que cela ne change pas la latence.

Changez les `MPI_Wtime` pour mesurer séparément le temps mis par `MPI_Isend()` et par `MPI_Wait`. Tracez une jolie courbe : dans `data`, rentrez les données ainsi :

```
1 temps_MPI_Isend_1 temps_MPI_Wait_1
4 temps_MPI_Isend_4 temps_MPI_Wait_4
...
```

Et tapez ceci dans `gnuplot` :

```
set logscale
plot "data" using ($1):($2) with lines, "data" using ($1):($3) with lines, \
"data" using ($1):($2)+($3) with lines
```

On aperçoit une cassure, qui correspond au changement de stratégie entre envoi direct et envoi par rendez-vous : avec rendez-vous, ce n'est alors plus `MPI_Isend()` qui fait l'envoi effectif des données, mais `MPI_Wait()`.

## 5 Et la bande passante ?

Comment mesurer la bande passante (en méga-octets échangés par seconde) ? Tracez de même une courbe.

## 6 Et sur Myrinet ?

Connectez-vous à la machine `infini1`, lancez `make clean` pour recompiler de zéro avec la version de `MPICH` installée sur `infini[1-4]` qui utilise des cartes réseau rapides Myrinet.

Recommencez les mesures. Est-ce intéressant ? D'où viennent ces différences de débit et de latence ?

## 7 Un anneau

Généralisez le programme à  $n$  machines : le nœud 0 envoie les données au nœud 1, qui le retransmet au nœud 2, etc jusqu'au nœud  $n - 1$  qui l'envoie de nouveau au nœud 0 (modifiez l'option `-np` dans `Makefile` pour exécuter plus que 2 processus). Comment la latence croît-elle avec  $n$  ?

## 8 Nota

Avec une autre implémentation de MPI (telle que `MPICH-Madeleine`), on aurait des courbes différentes, et l'on n'aurait pas forcément de changement de comportement lors de l'insertion de `sleep(1)`. `MPICH-Madeleine`, par exemple, utilise des threads pour que les communications puissent progresser en parallèle avec le programme principal.

## 9 Mémoire partagée vs. réseau

Modifiez le fichier `mymachines` afin d'utiliser plusieurs fois la même machine. Les processus seront donc lancés sur les différents processeurs de cette machine et les communications se feront par mémoire partagée. Est-ce intéressant ? Répartissez 8 processus (en mettant `-np` à 8 dans le fichier `Makefile`) de façon à améliorer le temps obtenu pour le programme précédent.