

Introduction à MPI

Il s'agit de s'initier à MPI et d'observer les comportements de la latence réseau. On pourra garder sous les yeux les pages de man des fonctions MPI, mais sur http://mpi.deino.net/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> .

1 Utilisation de MPI au CREMI

1.1 Paramètres MPI

Modifiez le fichier `mymachines` pour y mettre le nom de votre machine sur la première ligne, et une machine voisine sur la deuxième ligne.

Vérifiez que vous pouvez vous connecter via `ssh` sur toutes les machines listées dans `mymachines`.

Pour lancer le programme `hellow` au CREMI entrer `make PROG=hellow run`. C'est le même programme qui est lancé sur différentes machines (appelées *nœuds* et numérotés à partir de 0).

Note : il peut arriver que l'exécution échoue :

```
mpirun.openmpi was unable to launch the specified application as it could not access
or execute an executable:
```

```
Executable: ./hellow
```

C'est simplement parce que le fichier n'a pas eu le temps d'apparaître, via le réseau, sur l'autre machine. Relancez la commande, et cette fois cela fonctionnera.

1.2 Utilisation de `ssh` avec une clé (pour ceux qui n'en n'ont pas)

Pour éviter de taper son mot de passe sans arrêt générez une paire clé publique/privée :

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

Validez autant de fois qu'il le faut (gardez les valeurs par défaut). 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 localhost` que vous n'avez pas à taper de mot de passe pour vous connecter à la main à la machine de votre voisin. Il faudra au besoin confirmer l'identité de la machine.

Il se peut que `ssh` vous indique **WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!** C'est en général simplement parce que les machines ont été réinstallées et ont donc changé d'identité. Il faut supprimer l'ancienne identité (*Offending key*) du fichier `~/.ssh/known_hosts` à la ligne indiquée.

2 Communication inter processus

Modifier le programme `hellow` pour faire communiquer les deux processus en utilisant :

- `MPI_Send(buf,1,MPI_CHAR,1,0,MPI_COMM_WORLD)` ; pour que le processus 0 envoie un tableau `buf` de 1 caractère (`MPI_CHAR`) au nœud 1, avec le tag 0.
- `MPI_Recv(buf,1,MPI_CHAR,0,0,MPI_COMM_WORLD,&status)` ; pour que le processus 1 réceptionne un tableau `buf` de 1 caractère (`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` de type `MPI_Status`.

3 Mesures de performances

3.1 Latence

Le programme `ping.c` envoie un caractère d'une machine à une autre ("ping") et mesure le temps pris par les fonctions d'envoi et de réception. Est-ce une manière correcte de mesurer le temps que prend la communication ?

Complétez le programme pour que la deuxième machine renvoie ce caractère ("pong") et la première machine le réceptionne, et mesurez la latence de l'aller-retour. Lancez plusieurs fois, constatez que la mesure fluctue.

3.2 Courbe de latence en fonction de la taille du message

Ajoutez une boucle pour faire progresser la taille du message `N` de manière géométrique (de facteur 2) jusqu'à $1024 * 1024$. Puis modifiez l'affichage de façon à afficher chaque mesure ainsi :

```
fprintf(stderr, "network %d %ld\n", N, TIME_DIFF(t1,t2));
```

Récoltez les performances : `make PROG=ping perf` puis utilisez le script pour tracer une courbe en fonction de la taille : `Rscript tracer-courbe.R latence.data`

Pour améliorer la qualité de la courbe, utilisez une boucle effectuant la mesure de ping-pong 100 fois et ajoutez avant celle-ci une première boucle, non mesurée, effectuant la même chose, mais quelques dizaines de fois pour « préchauffer les fils » (c'est-à-dire, passer dans le code de MPI quelques fois pour que les heuristiques des caches et prédiction de branchement, etc. se stabilisent).

Tracer à nouveau la courbe.

3.3 Mémoire partagée vs. réseau

En mettant dans le fichier `mymachines` plusieurs fois le même nom de machine les processus seront lancés sur les différents processeurs de cette machine et les communications se feront par mémoire partagée. Modifier comme ci-dessous l'affichage de votre programme :

```
fprintf(stderr, "memory %d %ld\n", N, TIME_DIFF(t1,t2));
```

Relancer la prise de mesures (sans effacer le fichier `latence.data`) puis retracer les courbes.

3.4 Et la bande passante ?

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

4 Produit de Matrices

4.1 Communications point à point

Le programme `mul_mat` met en œuvre un produit de matrice ligne par ligne. Compléter ce programme en suivant l'algorithme ci-dessous :

```
tranche = N / size // on suppose que c'est N est divisible par size
```

```
// Code processus 0
```

```
Pour chaque processus i > 0 faire
```

```
    Envoyer b à i
```

```
    Envoyer à i les lignes a d'indices dans [i*tranche, (i+1) * tranche[
```

```
Calculer les lignes de c pour les indices de [0, tranche[
```

```
Pour chaque processus i > 0 faire
```

```
    Recevoir dans c les lignes calculées par i
```

```
// Code processus k > 0
```

Recevoir de 0 la matrice b
Recevoir de 0 les lignes de a

Calculer les lignes de c (correspondant à la tranche de k)

Envoyer à 0 les lignes de c

On notera que les processus esclaves n'ont pas besoin de recevoir toute la matrice a.

4.2 Communications collectives

Recopier le programme `mul-mat.c` et le script du batch dans des nouveaux fichiers. Modifier ces fichiers pour remplacer les communications point à point par des communications collectives (bcast, scatter, gather). Comparer les temps de transmission (phases de distribution et de collecte) et d'exécution obtenus à ceux obtenus par la version point à point.

5 Mesures de performances (suite)

Reprendre le code du programme `ping.c`.

5.1 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 `gettimeofday()` pour mesurer séparément le temps mis par `MPI_Isend()` et par `MPI_Wait`. Tracez trois jolies courbes sur un même graphique à l'aide de 3 `printf` :

```
fprintf(stderr,"isend %d %ld\n", N, TIME_DIFF(t1,t2));  
fprintf(stderr,"wait %d %ld\n", N, TIME_DIFF(t2,t3));  
fprintf(stderr,"latence %d %ld\n", N, TIME_DIFF(t1,t3));
```

pour tracer les courbes et leur somme en même temps.

On aperçoit vraiment nettement 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()`.

5.2 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 (au CREMI modifiez l'option `-np` dans `Makefile` pour exécuter plus que 2 processus, il faudra ajouter d'autres noms de machines dans `mymachines`). Comment la latence croît-elle avec n ?