Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Parallélisme & Applications

TP n°2

MPI

Compilation et Mise en place du « *cluster* »

Autorisation des connexions SSH par bi-clé asymétrique

Pour exécuter votre code sur différentes machines, c-à-d constituer votre « *cluster* » de machines, il vous faut activer la connexion automatique par clé publique/clé privée offerte par SSH :

xterm
```
$ ssh-keygen
```

Cette commande va générer une bi-clé RSA et l'installer dans votre répertoire `/.ssh`, sous forme de deux fichiers :

- *`id_rsa`*
- *`id_rsa.pub`*

Il ne vous restera plus qu'à autoriser l'accès à votre compte par l'utilisation de la bi-clé, en copiant la clé publique `id_rsa.pub` dans le fichier `authorized_keys`

xterm
```
$ cd ~/.ssh
$ cp id_rsa.pub authorized_keys
```

Il ne vous restera plus qu'à tester la connexion automatique en essayant un ssh sur localhost : si vous vous connectez sans donner votre mot de passe alors la configuration est correcte.

Compilation de votre code MPI

Pour compiler votre code utilisant MPI, il vous faut utiliser le compilateur « `mpicc` » :

xterm
```
$ mpicc -o mon_programme mon_source.c
```

Exécution du programme sur le « *cluster* »

Pour constituer votre cluster, vous allez utiliser les différentes machines de la salle de TP...qui **doivent être allumées** si vous voulez les utiliser !

Chaque machine de la salle de TP est nommée dans le DNS de la façon suivante : `fst-o-i-211-07`, où `211` indique le numéro de la salle de TP et `07` le numéro du poste (de 1 à 12).

xterm
```
$ dig +short fst-o-i-211-07.unilim.fr
164.81.94.166
```

Vous allez créer un fichier contenant les différentes machines de votre cluster en indiquant une entrée DNS par ligne dans le fichier « `composition_cluster` » :

```
fst-o-i-211-01.unilim.fr
fst-o-i-211-02.unilim.fr
...
```

L'exécution sur le cluster se fait à l'aide de la commande `mpirun` :

xterm
```
$ mpirun -np 5 -hostfile composition_cluster mon_programme
```

Où l'option « `-n 5` » indique le nombre de nœud de la machine parallèle et l'option « `-hostfile composition_cluster` » indique la liste des machines à utiliser pour déployer ces nœuds.

ATTENTION

Si les hôtes indiqués dans le fichier « *composition_cluster* » sont éteints, l'exécution échouera.

xterm
```
$ mpirun -np 2 -host fst-o-i-211-07,fst-o-i-211-06 mon_programme
```

Vous indiquerez « manuellement *» les hôtes **allumés** à utiliser...*

Vous devez également vous y **connecter une par une en ssh** pour les autorisations initiales.

1 – Vous rentrerez le programme suivant afin de tester votre cluster :

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Vous vérifierez que la fonction `MPI_Get_processor_name` *affiche bien les différents noms de votre cluster.*

2 – Vous récupérerez les fichiers :

- `makefile`
- `send_recv.c`
- `ping_pong.c`
- `ring.c`

Disponibles sur :

`https://github.com/wesleykendall/mpitutorial/tree/gh-pages/tutorials/mpi-send-and-receive/code`

Attention : cette URL étant longue, vous devrez la copier/coller dans votre navigateur plutôt que de cliquer dans le document PDF lui-même...

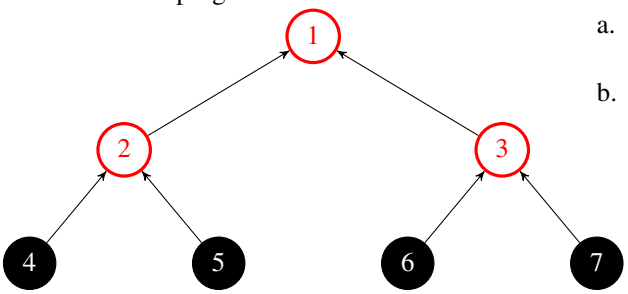
Questions :

a. Pour le programme `send_recv`, **combien** de nœuds vont travailler ?

b. Dans le programme `ping_pong`, comment sont choisis les « partenaires » de ping-pong ?
Est-ce que le programme peut **bloquer** ?
Est-il possible d'ajouter une **étiquette** différente par ping-pong ?

c. Pour le programme `ring`, comment est créer « *l'anneau* » ?
Pourquoi doit-on modifier le **comportement** du nœud de rang 0 ?
Pouvez vous ajouter un « *token* » qui s'incrémente à chaque envoi d'un nœud à l'autre ?
Est-ce que le nœud 0 peut connaitre le **nombre de nœuds** par la valeur du token qu'il reçoit au final ?

3 – Vous écrirez un programme réalisant la **somme d'entiers en utilisant une structure d'arbre** :



a. Comment chaque nœud va connaitre le rang de son parent ?

b. les nœuds 4,5,6 et 7 vont chacun envoyer un entier à leur parent respectif 2 et 3 qui va en faire la somme, avant de l'envoyer au nœud 1.

Écrivez un programme MPI réalisant ce travail en utilisant le rang du nœud feuille comme valeur entière à transmettre.

c. Pouvez vous rendre votre code le plus « *générique* » possible :
- définir le rôle de chaque nœud : feuille, intermédiaire et racine ;
- permettre la « *scalabilité* » : s'adapter automatiquement à l'augmentation de la taille de l'arbre.

Écrivez la version « *scalable* » de votre programme MPI qui s'adaptera au nombre de nœuds utilisés.