INF7235 : Laboratoire #7
Utilisation de MPI sur une grappe de calcul

13 mars 2017

Introduction

L'objectif de ce labo est de vous familiariser avec l'utilisation de MPI sur une grappe de calcul — sur un *cluster*.

La grappe que vous allez utiliser est une machine du LAMISS (acquise à l'hiver 2014) : `turing.hpc.uqam.ca`.

Cette machine possède 31 noeuds pour le calcul distribué — processeurs Intel Xeon X5650 2.67 GHz.

#SERVEUR	role	Dénomination	nb CPU	nb cœurs	cap.RAM	adresse IP	
1	head-node	TURING	2	16	24 Go	132.208.127.81	
2	OpenMP	HADRON1	4	64	512 Go	192.168.20.1	192.168.21.1
3	Hyperviseur	**Hadron2**	4	64	512 Go	192.168.20.2	192.168.21.2
	Nœuds	QUARK20	4		32Go	192.168.20.3	192.168.21.3
		QUARK21	4		32Go	192.168.20.4	192.168.21.4
		QUARK22	4		32Go	192.168.20.5	192.168.21.5
		QUARK23	4		32Go	192.168.20.6	192.168.21.6
		QUARK24	4		32Go	192.168.20.7	192.168.21.7
		QUARK25	4		32Go	192.168.20.8	192.168.21.8
		QUARK26	4		32Go	192.168.20.9	192.168.21.9
		QUARK27	4		32Go	192.168.20.10	192.168.21.10
		QUARK28	4		32Go	192.168.20.11	192.168.21.11
		QUARK29	4		32Go	192.168.20.12	192.168.21.12
4	Hyperviseur	**Hadron3**	4	64	512 Go	192.168.20.13	192.168.21.13
	Nœuds	QUARK30	4		32Go	192.168.20.14	192.168.21.14
		QUARK31	4		32Go	192.168.20.15	192.168.21.15
		QUARK32	4		32Go	192.168.20.16	192.168.21.16
		QUARK33	4		32Go	192.168.20.17	192.168.21.17
		QUARK34	4		32Go	192.168.20.18	192.168.21.18
		QUARK35	4		32Go	192.168.20.19	192.168.21.19
		QUARK36	4		32Go	192.168.20.20	192.168.21.20
		QUARK37	4		32Go	192.168.20.21	192.168.21.21
		QUARK38	4		32Go	192.168.20.22	192.168.21.22
		QUARK39	4		32Go	192.168.20.23	192.168.21.23
5	Hyperviseur	**Hadron4**	4	64	512 Go	192.168.20.24	192.168.21.24
	Nœuds	QUARK40	4		32Go	192.168.20.25	192.168.21.25
		QUARK41	4		32Go	192.168.20.26	192.168.21.26
		QUARK42	4		32Go	192.168.20.27	192.168.21.27
		QUARK43	4		32Go	192.168.20.28	192.168.21.28
		QUARK44	4		32Go	192.168.20.29	192.168.21.29
		QUARK45	4		32Go	192.168.20.30	192.168.21.30
		QUARK46	4		32Go	192.168.20.31	192.168.21.31
		QUARK47	4		32Go	192.168.20.32	192.168.21.32
		QUARK48	4		32Go	192.168.20.33	192.168.21.33
		QUARK49	4		32Go	192.168.20.34	192.168.21.34

Ce qu'il faut faire

1. Dans un premier temps, vous devez vous connecter à votre compte sur `turing` :

   ```
   $ ssh nomUsager@turing.hpc.uqam.ca
   ```

   Votre nom d'usager et code permanent sont comme suit :

   - *nomUsager* = `nomDeFamille_p`
     Où `p` dénote l'initiale ou les initiales de votre prénom.
     Exemple : `tremblay_g`
   - Mot de passe = Code permanent
     Exemple : `TREGjjmmaa99` où `jj` = date de naissance, `mm` = mois de naissance, `aa` = année de naissance, `99` = numéro unique d'identification.

   Notes :

   - Contrairement à `japet`, il est possible de se connecter directement à `turing` avec `x2go` (donc sans passer par `malt`).
     Au préalable, il faut sélectionner «**Gnome**» comme gestionnaire de fenêtres, et non pas KDE qui est sélectionné par défaut!
   - La première chose à faire après vous être connecté est **de modifier votre mot de passe** — commande `passwd`.

2. Ensuite, il faut obtenir les fichiers du labo :

   ```
   $ git clone http://www.labunix.uqam.ca/~tremblay/git/Labo1-MPI.git
   ```

Remarques : Pour compiler et exécuter un programme MPI/C sur une machine multi-ordinateurs avec OpenMPI, deux étapes simples sont requises :

1. On compile le programme — avec `mpicc`.
2. On lance l'exécution du programme en spécifiant le nombre de «processeurs» qu'on désire utiliser — avec `mpirun`.

Pour ces diverses tâches, des cibles `make` appropriées ont été définies.

Programme `hello.c`

1. Compilez et exécutez le programme `hello.c` et observez son comportement.

   Voici les principales commandes pour ce faire :

   - `make compile1` : Compile le fichier, mais sans l'exécuter.
   - `make run1` : Exécute le programme, en le compilant au préalable si nécessaire.

2. Le commentaire au début du fichier `hello.c` décrit ce qu'est supposé faire ce programme. **Complétez le fichier `hello.c` pour qu'il ait le comportement désiré.**

Programme `somme-vecteurs.c`

Le programme `somme-vecteurs.c` fait la somme de deux vecteurs, de façon parallèle, en utilisant une distribution statique des éléments des vecteurs (distribution par groupe d'éléments adjacents).

Complétez le corps du programme principal — plusieurs procédures auxiliaires vous sont fournies.

Voici les principales commandes :

- `make compile2` : Compile le fichier, mais sans l'exécuter.
- `make run2` : Exécute le programme, en le compilant au préalable si nécessaire, et en utilisant les paramètres par défaut (8 processeurs, 10000 éléments).
- `make run2 NP=`*`np`* `N=`*`n`* : Exécute le programme avec les paramètres d'exécution indiqués, où n indique la taille des vecteurs.

Programme `min-max.c`

Le programme `min-max.c` met en oeuvre, en MPI, deux des exemples Ruby vus en cours pour calculer le minimum et le maximum d'une valeur locale conservée par des processus, et ce en échangeant des messages de différentes façons.

- `minMaxCentralise` : Solution centralisée.

  Chaque processus transmet sa valeur au processus maître, qui calcule le minimum et le maximum, puis retransmet ces résultats aux autres processus.

- `minMaxAllreduce` : Solution avec réduction collective.

  En MPI, il est aussi possible d'effectuer la même tâche, mais en utilisant plus simplement une opération collective de réduction — ici, `MPI_Allreduce`, puisqu'on désire que chaque processus connaisse le minimum et le maximum.

- `minMaxAnneau` : Solution avec anneau virtuel.

  Durant le premier tour, le minimum et le maximum sont mis à jour petit à petit par chacun des processus visités autour de l'anneau. À la fin du premier tour, le premier processus connait alors les valeurs minimum et maximum, mais pas les autres processus. Le premier processus amorce alors une deuxième vague de transmission de messages autour de l'anneau qui va permettre à chacun des autres processus de connaitre les valeurs minimum et maximum résultantes.

1. Complétez les deux premières versions des fonctions pour calculer le minimum et le maximum — `minMaxCentralise` et `minMaxAllreduce`.

2. Comparez les temps d'exécution requis par chacune de ces deux approches, et ce en faisant varier le nombre de processus — notamment, utilisant aussi des processeurs virtuels, i.e., un plus grand nombre de processus que de processeurs **physiques**. Qu'est-ce qui semble le plus rapide? De façon significative? Pour tous les nombres de processeurs?

3. Complétez la troisième version et faites à nouveau des mesures de temps d'exécution.

Note : Pour compiler et exécuter : «`make compile3`» et «`make run3`».