TD 6 Programmation PVM

La documentation complète : http ://www.csm.ornl.gov/pvm/man/manpages.html.
Les programmes d'exemples sont disponibles dans `$PVM_ROOT/examples`.

Prise en main

1. Vérifier que la variable `$PVM_ROOT` est bien positionnée à `/opt/pvm3`
2. Créer votre environnement : dans votre home
   ```
   mkdir .ssh
   cd .ssh
   ssh-keygen -t dsa
   cat id_dsa.pub >>authorized_keys
   ```
3. Créer (avec `mkdir -p`)les répertoires
   - `$HOME/pvm3/bin/$PVM_ARCH`. Tous les exécutables utilisant pvm **doivent** y être placés.
   - `$HOME/pvm3/src/$PVM_ARCH`. Pour vos futurs programmes source.
4. Lancer pvm par la commande `pvm`.
   - Dans la console, `help`.
   - Sous un shell, `ls -l /tmp |grep pvm`.
5. Ajouter une machine, puis :
   - commande console `conf` ;
   - comparer l'effet des commandes console `quit` et `halt` en les exécutant, puis en relançant la console.
6. Copier `germain/Pub/Makefile` dans `$HOME/pvm3/src/$PVM_ARCH` et l'étudier.

Pour plus d'information sur la configuration, on peut lire `$PVM_ROOT/Readme`.

Syntaxe

1. Modifier le programme `hello_other.c` pour qu'il s'exécute obligatoirement sur un autre processeur que la console :
2. Modifier les programmes `hello.c` et `hello_other.c` pour que hello_other communique en plus un tableau de réels et un tableau d'entiers (initialisés), et que `hello` affiche aussi ces tableaux.
3. Modifier les programmes précédents pour que l'envoi et la réception ne concernent qu'un élément sur 3.
4. Modifier les programmes `hello.c` et `hello_other.c` pour qu'ils communiquent de façon continue jusqu'à ce que l'esclave décide d'arrêter l'échange.

5. Sur le modèle précédent, créer deux programmes où hello_other envoie des messages de taille aléatoire entre 1 et 10 entiers, et hello n'ouvre que les messages de taille supérieure à 5. Pour générer un nombre aléatoire (avec un qualité d'aléa possiblement très mauvaise), utiliser 1+rand()%10.

Programmation SPMD : Anneau à jeton

Le programme `spmd.c` simule le passage d'un jeton dans un anneau de P tâches : la tâche 0 émet un jeton et attend qu'il lui revienne. Modifier le programme pour les tâches effectuent un calcul "utile" (incrémentation d'un compteur flottant) en attendant le jeton. Le processeur 0 récupèrera la valeur cumulée des compteurs.

Tableaux distribués

On considère un tableau $N \times N$, partitionné par lignes en bandes de taille $M = N/P$.

1. Etudier le programme `germain/Pub/decal.c`. Quelle est la distribution utilisée ?
2. Modifier ce programme pour qu'il effectue un décalage vers le haut, et où le voisin de 0 et $P-1$.
3. Modifier ce programme pour une distribution block-2D à 16 processeurs.
4. Modifier ce programme pour qu'il effectue un stencil 5 points (moyenne avec les 4 voisins).

Performances

1. Etudier les programmes `timing.c` et `timing_slave.c`
2. Exécuter ces programmes en faisant varier `ENCODING` et la localisation des exécutions. Que peut-on en conclure ?

Programmation maître-esclave et répartition de charge

On considère un programme maître-esclave, avec la structure suivante : le maître initialise aléatoirement un tableau de N entiers, et crée P esclaves Chacun des esclaves doit effectuer une tâche paramétrée par un des entiers du tableau. Lorsque l'esclave a fini une tâche, il le signale au maître. Dans le TD, la tâche sera simplement un `sleep`.

1. Réaliser les programmes avec une distribution block : l'esclave i exécute un bloc contigu de tâches Donner les durées d'exécution de chaque esclave vues du maître et la durée d'exécution totale.
2. Mêmes questions avec une distribution cyclique : l'esclave i exécute les tâches $i\%P$.