Non Uniform Memory Access

Une machine NUMA est une machine à mémoire commune constituée de nœuds (intégrant des processeurs et de la mémoire) reliés par un réseau. Dans ces machines la latence mémoire dépend de la distance entre le nœud du cœur et le nœud contenant la mémoire cible. Le programme `hwloc-distances` permet d'afficher le facteur NUMA entre différents nœuds et la topologie peut être visualisée via la commande `lstopo`.

Les machines NUMA du CREMI sont celles des salles 008 / 203 ou les serveurs AMD (boursouf, boursouflet et jolicoeur) ou Intel (xeonphi, tesla cocatris). Avant de lancer une commande sur un serveur, vérifier à l'aide de la commande `htop` que celui-ci est libre et évitez de le monopoliser pendant plus de 30 secondes.

1 Latence mémoire sur NUMA

Il s'agit de mesurer l'impact du placement thread / mémoire sur des machines NUMA. Ici nous proposons une expérience pour essayer de quantifier ce facteur NUMA et au passage d'apprécier les latences des différents caches.

Le principe du programme `test-numa coeur noeud` est de fixer un thread sur le cœur donné, d'allouer un tampon sur le nœud NUMA donné ; ensuite on mesure le temps mis pour accéder presque *aléatoirement* au contenu du buffer un nombre constant de fois (ici 1 000 000 de fois). De plus on fait varier la taille du buffer entre 1ko et 256Mo et pour chaque taille du buffer on affiche le temps mis par une itération.

1. Lancez `test-numa 0 0` - ici cœur et mémoire sont sur le même nœud puis lancez `test-numa 0 1` - maintenant cœur et mémoire ne sont plus sur le même nœud.
2. Tracez les courbes à l'aide du script `tracer-latence.r` et essayez d'estimer la taille des différents caches à partir de ces courbes. Utilisez la commande `lstopo` pour obtenir la taille des caches.
3. Estimez le *facteur NUMA* (rapport entre latence d'accès distant et latence d'accès locale).
4. Réessayez sur un serveur NUMA (jolicoeur, boursouf, tesla), constatez que la latence varie selon les positions relatives du processeur et de la mémoire.

2 Faux partage

On va observer les effets de faux partage (False sharing) sur les machines de la salle 008.

La commande `test-line distance coeur1 [coeur2 coeur3 ...]`
lance des threads qui vont de manière concurrente incrémenter des variables différentes : le thread i incrémente la variable `(char *)tab + i*distance`. Le programme affiche le nombre de millions d'incrémentations que chaque thread parvient à faire chaque seconde (plus c'est grand mieux c'est).

Lancez d'abord un seul thread pour obtenir une valeur de référence : le thread tourne alors tout seul, et la variable dans laquelle il accède peut rester en permanence dans le cache.

Lancez maintenant deux threads, sur les cœurs 0 et 1 par exemple. Lorsque les variables confiées aux deux threads sont proches (même si pas confondues !), on a un faux partage, conduisant à un ping-pong de

lignes de cache.

Faîtes varier l'indice de la case confiée au deuxième thread (en veillant à toujours utiliser un multiple de 4 pour conserver tout de même des accès bien alignés en mémoire). Déterminez expérimentalement la taille d'une ligne de cache.

Fixez la distance à 8 et le premier thread sur le processeur 0 et faites maintenant varier le numéro de processeur sur lequel vous lancez le second thread. Que remarquez-vous ?

Toujours avec une distance de 8, comparez l'exécution de 4 threads et ce sur différentes combinaisons (tous sur le même processeur, 2 sur chaque processeur et 3 sur l'un / 1 sur l'autre). Le partage de la ligne de cache vous semble-t-il équitable ?
Testez ce programme sur un (seul) des serveurs AMD.

3 Bande passante

L'intérêt majeur des machine NUMA est de permettre un accès parallèle à la mémoire. Nous allons mesurer l'impact de ce nouveau paramètre sur le noyau transpose du programme `2Dcomp`.
Pour cela vous allez déplacer votre fichier `transpose.c` afin de recopier une version étoffée de ce fichier dans le répertoire du mini-projet chez sur le compte de M. Namyst.
L'utilisation du programme `numactl` qui permet de contrôler grossièrement le placement des threads et de la mémoire. Comparer et interpréter les performances des appels suivants :

```
numactl --cpubind 0 --membind 0 ./2Dcomp -k transpose -v tiled -s 4096 -n -i 10 -g 32
numactl --cpubind 0 --membind 1 ./2Dcomp -k transpose -v tiled -s 4096 -n -i 10 -g 32
numactl --interleave all ./2Dcomp -k transpose -v tiled -s 4096-n -i 10 -g 32
```

La technique dite du *first touch* utilise le fait que les pages physiques sont allouées paresseusement - au dernier moment -par le système d'exploitation pour allouer toute page physique sur le nœud du cœur qui provoque le premier accès à cette page. Cette technique est disponible pour la noyau version `omp_tiled` du noyau `transpose`. L'option `-ft` active la mise en œuvre de cette technique.
En fixant `OMP_NUM_THREADS=12 GOMP_CPU_AFFINITY=$(seq 0 23)`, comparer les performances des appels suivants :

```
./2Dcomp -k transpose -v omp_tiled -s 4096-n -i 10 -g 32
./2Dcomp -k transpose -v omp_tiled -s 4096-n -i 10 -g 32 -ft
numactl --interleave all ./2Dcomp -k transpose -v omp_tiled -s 4096 -n -i 10 -g 32
```

En analysant finement le comportement du noyau, on peut cependant douter de l'optimalité de la technique du first touch pour ce code. Pourquoi ?