École « Programmation hybride : une étape vers le many-coeurs ? »
Placement des threads et aspects mémoire

Labs Mémoire

Dans la suite de ce document, le terme PPN signifie « Process Per Node » et se réfère au nombre de processus MPI par nœud de calculs. Quant au terme OMP, il se réfère au nombre de threads OpenMP par processus séquentiel ou MPI suivant le contexte ; autrement dit, `OMP_NUM_THREADS`=OMP.

Vous pouvez récupérer l'archive `Labs_memoire.tar.gz` contenant l'ensemble des codes d'intérêt pour ce TP à l'emplacement `/home_nfs/saugel/Labs_memoire.tar.gz`.

Ce TP est composé de quatre grandes parties :

1. Bande passante mémoire des nœuds de calculs
2. Placement hybride
3. Matrice des distances
4. Mesures de débits vers les GPUs et effet NUMA IOs

Bande passante mémoire des nœuds de calculs

Avant de commencer, nous allons estimer la bande passante mémoire des nœuds de calcul mise à notre disposition. Afin d'estimer ceci, nous allons utiliser le code de McCalpin « stream ».

Compilez tout d'abord le code à l'aide des compilateurs Intel®. On considèrera un tableau de plus de 4 GB (`N=200000000`) afin d'obtenir des valeurs significatives.

Compilation
```
# module add intel/compilers/11.1.069
# cd STREAM
# icc -o stream_intel -O2 -ip -openmp -mcmodel=medium src/stream.c -DN=200000000
```

N'oubliez pas le flag `-mcmodel=medium`. Sans celui-ci vous obtiendrez une erreur à la compilation du type « `relocation truncated to fit` » (essayez !).

Débit par nœud (machine)

Exécutez « stream » au travers de slurm via la commande `srun` avec OMP=8 (machine pleine).

Execution « as-is »
```
# export OMP_NUM_THREADS=8
# srun --exclusive -N1 ./stream_intel
```

Relancez le test plusieurs fois. Il se peut que vous obteniez de la variabilité dans les mesures. Dans ce cas (et dans tous les autres !) il est grand temps d'optimiser le placement des processus et de la mémoire. Nous allons

- d'abord placer et attacher les processus à l'aide de `KMP_AFFINITY`,
- en profiter pour demander à la runtime Intel de nous afficher ce qu'elle fait réellement (ajout du modificateur `verbose` à `KMP_AFFINITY`),
- forcer la politique de gestion mémoire à « local ».

Placements optimisés
```
# export OMP_NUM_THREADS=8
# export KMP_AFFINITY=verbose,compact
# srun --exclusive -N1 numactl -l ./stream_intel
```

Relevez les débits « stream » nominaux par nœud.

Copy	_____ MB/sec
Scale	_____ MB/sec
Add	_____ MB/sec
Triad	_____ MB/sec

Débit par socket

Exécutez « stream » sur une seule socket (OMP=4). Vous prendrez garde à placer correctement les processus et d'assurer les transferts depuis le contrôleur mémoire local. Reproduisez l'expérience pour les deux sockets.

Lancement « débit par socket »

```
# export OMP_NUM_THREADS=4
# ____________________________________________________
# srun --exclusive -N1 _____________________________________
```

	Socket 0	Socket 2
Copy	_____ MB/sec	_____ MB/sec
Scale	_____ MB/sec	_____ MB/sec
Add	_____ MB/sec	_____ MB/sec
Triad	_____ MB/sec	_____ MB/sec

Sachant que les processeurs utilisés sont des Xeon ® 5560[1] et que le contrôleur mémoire de ces sockets supporte les barrettes à 800, 1066 et 1333 GT/sec (3 canaux par socket), que les barrettes utilisées sont à 1333 GT/sec, estimez l'efficacité du contrôleur mémoire. Pour ce calcul, on considère généralement le résultat de Triads.

Efficacité (estimée) de l'IMC : _____ %

Détermination du facteur NUMA (débit)

Toujours à l'aide de « stream », estimez le facteur NUMA en débit. Vous vous servirez des mesures précédentes et de nouvelles en croisant les domaines de placement « CPUs » et « mémoire » (utilisez `numactl`).

	CPU domain : socket 0 Memory domain : socket 1	CPU domain : socket 1 Memory domain : socket 0	Facteur NUMA en débit
Copy	_____ MB/sec	_____ MB/sec	_____
Scale	_____ MB/sec	_____ MB/sec	_____
Add	_____ MB/sec	_____ MB/sec	_____
Triad	_____ MB/sec	_____ MB/sec	_____

Vous trouverez dans le répertoire `STREAM/tools` un script `numadiff`. Il s'utilise comme un wrapper shell et fournit en fin d'exécution les statistiques NUMA. Testez-le à l'occasion dans diverses configurations (l'unité est la page soit 4k).

[1] Voir les caractéristiques: http://ark.intel.com/products/37109/Intel-Xeon-Processor-X5560-8M-Cache-2_80-GHz-6_40-GTs-Intel-QPI

Avec GCC

Recompilez « stream » avec GCC et relevez les débits par nœud et par socket.

	Par nœud	Par Socket
Copy	_____ MB/sec	_____ MB/sec
Scale	_____ MB/sec	_____ MB/sec
Add	_____ MB/sec	_____ MB/sec
Triad	_____ MB/sec	_____ MB/sec

Conclusion ? _____

Facultatif : first touch

Comparez les sources de « fstream » et « stream ». Compilez les deux codes dans les mêmes conditions (compilateur et flags).
Exécutez-les dans les mêmes conditions (OMP=8, machine pleine et politique NUMA « locale »). Comparez les résultats.

	stream	fstream
Copy	_____ MB/sec	_____ MB/sec
Scale	_____ MB/sec	_____ MB/sec
Add	_____ MB/sec	_____ MB/sec
Triad	_____ MB/sec	_____ MB/sec

Recommencez l'expérience en mode « interleave ».

	stream (interleave)	stream (interleave)
Copy	_____ MB/sec	_____ MB/sec
Scale	_____ MB/sec	_____ MB/sec
Add	_____ MB/sec	_____ MB/sec
Triad	_____ MB/sec	_____ MB/sec

Placement hybride

Par la suite, vous utiliserez au choix Intel ® MPI ou bullxmpi (« OpenMPI based »). Vous utiliserez les compilateurs Intel®.

Warm-up : « Hello,World ! » hybride

Avant de commencer avec `hydro`, compilez les deux codes fournis dans le répertoire `HelloWorld` à l'aide du makefile.

Compilation
```
# module add intel/compilers/12.1.8.273
# module add intel/mpi/4.0.3.008
# cd HelloWorld
# make CC=mpiicc all
```

Vous devrez indiquer à Intel ® MPI la bonne bibliothèque pmi à utiliser (sinon crash et insultes assurés au `srun` …) :

Environnement
```
# export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
```

Testez le binaire (et l'environnement):

Execution
```
# srun --exclusive -N 2 -n 16 ./hw
# OMP_NUM_THREADS=4 \
  OMP_PROC_BIND=true \
  srun --exclusive --cpu_bind=mask_cpu:0xF,0xF0 -N 2 -n 4 ./hw-omp
```

Si vous ne rencontrez pas d'erreur à ce stade, vous pouvez continuer ! Sinon criez « aux secours » !

Hydro hybride, cas test 250x250-noio

Pour ces tests, vous utiliserez la version `F90/Hybride/MPI_OMPCG2DSync` du code hybride.
Le but de cette partie est d'évaluer l'adhérence du cas test (et pas seulement du code) à la bande passante mémoire.

Pour ce faire, vous exécuterez le cas test sur un seul nœud, dans différentes configurations. Vous relèverez le temps d'exécution à colonne « `Process Elapsed Time (s)` », ligne « `Average` » dans le profiling de fin d'exécution de l'application. Prenez soin aux placements CPU et mémoire.

configurations	Average Process Elapsed Time
PPN=8 OMP=1	_____ sec
PPN=1 OMP=8	_____ sec
PPN=2 OMP=4 (même domaine CPU et mémoire)	_____ sec

Vous allez maintenant reproduire la dernière expérience (PPN=2 OMP=4) dans une configuration des domaines mémoire et CPU « croisés », obligeant un processus (threads) attaché à une socket à aller ces données dans la mémoire de l'autre socket. C'est un cas *a priori* extrêmement défavorable, voire le plus défavorable …
Pour se faire, vous allez écrire le wrapper bash suivant :

```
#!/bin/bash

CPUNODE=$SLURM_LOCALID
MEMNODE=$(($((CPUNODE+1))%2))

exec numactl --membind=$MEMNODE --cpunodebind=$CPUNODE $@
```

Exécutez-le sur le code hydro, relevez une nouvelle fois le temps.

Configurations	Average Process Elapsed Time
PPN=2 OMP=4 (domaines croisés)	_____ sec

Qu'en concluez-vous ? _____

Hydro hybride, cas test 10000x10000-noio

Dans cette section, nous allons utiliser deux binaires. Cette partie pourra être faite à deux, un membre du binôme travaillant avec le binaire original, l'autre avec le nouveau binaire.

Le nouveau binaire sera produit en ajoutant l'option « `-opt-streaming-stores always` » au compilateur Intel® Fortran. N'oubliez pas de renommer l'ancien binaire avant qu'il ne soit écrasé lors de la nouvelle compilation !

Production des binaires

```
# module add intel/compilers/12.1.8.273
# module add intel/mpi/4.0.3.008

# cd $HOME/HYDRO/F90/Hybride/MPI_OMPCG2DSync
# mv Src/hydro Src/hydro_orig
# make -C Src MPIF90=mpiifort FFLAGS="-O3 -openmp -opt-streaming-stores always"
```

Pour chacun des binaires, vous allez mesurer les temps de restitution suivant la même méthode qu'à la section précédente, uniquement dans la configuration PPN=2,OMP=4, dans les cas d'accès purement locaux (favorable) et dans le cas d'accès purement distant (défavorable).

Configurations	Average Process Elapsed Time		Dégradation
	`hydro_orig`	`hydro`	
PPN=2 OMP=4 (même domaine CPU et mémoire)	_____ sec	_____ sec	_____
PPN=2 OMP=4 (domaines croisés)	_____ sec	_____ sec	_____

Qu'en concluez-vous ? _____

Matrice des distances

Dans le répertoire `LATENCES` vous trouverez un code simplifié qui donne le temps d'accès en cycles processeur d'un tableau dont les données sont accédées linéairement avec de larges « strides ». Ceci permet d'estimer les latences d'accès des divers éléments constituant la hiérarchie mémoire de la machine (caches, RAM locale voire distante…).
Le but de cette section est de reconstituer la matrice des distances NUMA telle qu'elle peut être donnée par la commande `numactl` entre autre.

Compilez le code (deux fichiers .c) à l'aide du `makefile`.
Exécutez dans la foulée le code à l'aide du script `matrix` fourni dans le répertoire `tools`. Si le temps et le cœur vous en dit, disséquez ce script.

Compilation/Execution

```
# cd LATENCES
# make all
# srun --exclusive -N 1 tools/matrix
```

Par définition, la distance intra-nœud NUMA vaut 10. Normalisez la matrice des latences et comparez à la matrice des distances donnée par `numactl -hardware` .

	0	1
0	_____ cycles	_____ cycles
1	_____ cycles	_____ cycles

Normalisée :

	0	1
0	_____ (numactl : _____)	_____ (numactl : _____)
1	_____ (numactl : _____)	_____ (numactl : _____)

Mesures de débits vers les GPUs et effet NUMA IOs

L'exercice ici est simple. Tout comme dans le cas de la mémoire avec « stream », nous allons par l'expérience déterminer le facteur NUMA IOs vers/depuis les cartes GPUs. Ceci va nous permettre d'ébaucher l'architecture « PCIe » des nœuds de calculs.

Warm-up: compilation et détermination du nombre de GPUs.

Les codes d'exemples relatifs à cette partie sont disponibles dans le répertoire `GPU` (original non ?).
Les codes d'intérêts (`deviceQuery` et `bandwidthTest`) se compilent à l'aide du même `makefile` et les binaires produits se trouvent sous `bin/linux/release/`

Compilation
```
# module add cuda/4.0
# cd GPU
# make clean
# make
```

Execution
```
# srun --exclusive -N 1 ./bin/linux/release/deviceQuery
```

Nombre de GPUs par nœuds: _____ GPU(s)

Détermination de la matrice des débits

Prenez exemple sur la ligne de commande suivante pour construire la matrice des débits « host-to-target » et « target-to-host ». Déduisez-en un schéma de principe de l'architecture du nœud de calcul (CPUs, chipset, GPUs, et liens QPI/PCIe).

Execution
```
# srun --exclusive -N 1 numactl --physcpubind=0 --membind=0 \
  ./bin/linux/release/bandwidthTest --memory=pinned --device=0
```

	host-to-target		target-to-host	
	Socket 0	**Socket 1**	**Socket 0**	**Socket 1**
GPU device 0	_____ MB/s	_____ MB/s	_____ MB/s	_____ MB/s
GPU device 1	_____ MB/s	_____ MB/s	_____ MB/s	_____ MB/s

Félicitations ! et …