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

Labs Mémoire

CORRIGÉ

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
```

```
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       35888.1486       0.0897       0.0892       0.0907
Scale:      34972.2963       0.0919       0.0915       0.0922
Add:        34213.6143       0.1406       0.1403       0.1411
Triad:      33934.0408       0.1419       0.1415       0.1422
```

J'ai noté personnellement peu de variabilité. À part peut-être le nœud nova4 qui montrait des performances moins bonnes que les autres. Je conseille dans ce cas d'exclure ce nœud. Pour ce faire, utiliser l'option « `-x` » de `slurm/srun` :

```
srun -xnova4 --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	36106 MB/sec
Scale	35131 MB/sec
Add	34362 MB/sec
Triad	33950 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 -xnova4 numactl --cpunodebind=0 --membind=0 ./stream_intel
```

	Socket 0	Socket 2
Copy	18227 MB/sec	18288 MB/sec
Scale	17737 MB/sec	17752 MB/sec
Add	17258 MB/sec	17305 MB/sec
Triad	17073 MB/sec	17066 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 MT/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.

Le débit théorique vaut: `1333 MTransferts/s x (64 bits /Transfert/canal) x 3 canaux = 255936 Mbits/sec = 31.9 Gbytes/sec.`

Efficacité (estimée) de l'IMC : 54 %

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

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	8920 MB/sec	8972 MB/sec	1 :2
Scale	8771 MB/sec	8802 MB/sec	1 :2
Add	8952 MB/sec	9015 MB/sec	1 :2
Triad	9078 MB/sec	9093 MB/sec	1 :2

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).

```
srun -xnova4 --exclusive -N1 numactl --cpunodebind=1 --membind=0 tools/numadiff
./stream_intel
-------------------------------------------------------------
STREAM version $Revision: 5.9 $
-------------------------------------------------------------
…
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:        8971.1496       0.3574       0.3567       0.3584
Scale:       8794.0580       0.3646       0.3639       0.3663
Add:         9020.8412       0.5346       0.5321       0.5384
Triad:       9103.8274       0.5299       0.5273       0.5341
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------

numastat:
----------------------------------------
                node0    node1
numa_hit        1173704  2454
numa_miss       0        0
numa_foreign    0        0
interleave_hit  17       15
local_node      1190     2415
other_node      1172514  39
```

Avec GCC

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

```
cc -o stream_cc -O3 -fopenmp -mcmodel=medium src/stream.c -DN=200000000
```

	Par nœud	Par Socket
Copy	24897 MB/sec	12619 MB/sec
Scale	25592 MB/sec	12851 MB/sec
Add	27397 MB/sec	13914 MB/sec
Triad	27505 MB/sec	13968 MB/sec

Conclusion ? A la différence du compilateur intel, gcc n'optimise pas les opérations d'écriture à l'aide des streaming stores. Du coup, le nombre d'opérations mémoire supposé fait dans le code stream est faux, ce

qui fausse le calcul des débits (d'un facteur 2/3 pour copy scale et ¾ pour Add/Triad). Il suffit en fait de modifier le code de stream en modifiant les lignes :

```
static double   bytes[4] = {
    2 * sizeof(double) * N,
    2 * sizeof(double) * N,
    3 * sizeof(double) * N,
    3 * sizeof(double) * N
    };
```

par

```
static double   bytes[4] = {
    3 * sizeof(double) * N,
    3 * sizeof(double) * N,
    4 * sizeof(double) * N,
    4 * sizeof(double) * N
    };
```

Dans ce cas, les mesures machines pleine donnent :

```
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       37447.3318       0.1286       0.1282       0.1293
Scale:      37976.7627       0.1268       0.1264       0.1273
Add:        36659.4408       0.1753       0.1746       0.1763
Triad:      36789.2158       0.1744       0.1740       0.1751
-------------------------------------------------------------
```

Dans le cas Intel, on modifie aussi les mêmes lignes de codes et on ajoute à la compilation `-opt-streaming-stores never`. On obtient :

```
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       53988.4186       0.0893       0.0889       0.0900
Scale:      37998.4093       0.1265       0.1263       0.1269
Add:        36955.5430       0.1735       0.1732       0.1742
Triad:      37029.7683       0.1730       0.1728       0.1735
-------------------------------------------------------------
```

Pour copy, le compilateurs optimise cette opération via une fonction `memcpy_intel` optimisée, ne tenez pas compte de ce résultat dans un premier temps (Il faut en fait conserver le 36106 MB/sec de tout à l'heure). Par compte, pour Scale, Add et Triad, les résultats sont identiques dans les deux cas (Intel et gcc). Tout est OK ! Dans ce cas, l'efficacité augmente légèrement, on arrive à 59% !

L'utilisation des compteurs hardware sur la bande passante confirme ces mesures.

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	36106 MB/sec	10695 MB/sec
Scale	35131 MB/sec	14460 MB/sec
Add	34362 MB/sec	14830 MB/sec
Triad	33950 MB/sec	14830 MB/sec

Recommencez l'expérience en mode « interleave ».

	stream (interleave)	fstream (interleave)
Copy	19712 MB/sec	19746 MB/sec
Scale	25847 MB/sec	25883 MB/sec
Add	27688 MB/sec	27646 MB/sec
Triad	27462 MB/sec	27394 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	39.168 sec
PPN=1 OMP=8	42.885 sec
PPN=2 OMP=4 (même domaine CPU et mémoire)	39.880 sec

Compilation/Execution

```
make -C Src MPIF90=mpiifort clean
make -C Src MPIF90=mpiifort
export OMP_PROC_BIND=true

export OMP_NUM_THREADS=1
srun -xnova4 --exclusive -N 1 -n 8 ./Src/hydro ./Input/input_sedov_noio_250x250.nml

export OMP_NUM_THREADS=8
srun -xnova4 --exclusive -N 1 -n 1 ./Src/hydro ./Input/input_sedov_noio_250x250.nml

export OMP_NUM_THREADS=4
srun -xnova4 --exclusive --cpu_bind=mask_cpu:0xF,0xF0 -N 1 -n 1 ./Src/hydro
./Input/input_sedov_noio_250x250.nml
```

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)	39.876 sec

On vérifie de binding et le wrapper à l'aide du HelloWorld hybride et de `numactl --show` :

```
srun --cpu_bind=none --exclusive -N 1 -n 2 ./wrapper.sh numactl --show

policy: bind
preferred node: 1
physcpubind: 0 1 2 3
cpubind: 0
nodebind: 0
membind: 1
policy: bind
preferred node: 0
physcpubind: 4 5 6 7
cpubind: 1
nodebind: 1
membind: 0
```

Dans le résultat de la commande précédente, clairement on voit que les cœurs logiques 0 1 2 3 (nœud numa 0) « binderont » leur mémoire sur le nœud mémoire numa 1 (membind: 1) et inversement. Le croisement est bien en place.

```
srun --cpu_bind=none --exclusive -N 1 -n 2 ./wrapper.sh $HOME/
/Labs_memoire/HelloWorld/hw-omp
processor per node (check on rank 0): 8
proc   1/  2 says Hello,World! (from node nova4) 4 5 6 7
proc   0/  2 says Hello,World! (from node nova4) 0 1 2 3
```

```
hello, was launched by process (0,0) and I am running on logical core: 0
hello, was launched by process (1,0) and I am running on logical core: 4
hello, was launched by process (0,1) and I am running on logical core: 1
hello, was launched by process (1,1) and I am running on logical core: 5
hello, was launched by process (1,3) and I am running on logical core: 7
hello, was launched by process (1,2) and I am running on logical core: 6
hello, was launched by process (0,2) and I am running on logical core: 2
hello, was launched by process (0,3) and I am running on logical core: 3
```

```
srun --cpu_bind=none --exclusive -N 1 -n 2 ./wrapper.sh ./Src/hydro
./Input/input_sedov_noio_250x250.nml
```

Qu'en concluez-vous ? Le code avec ce cas test est très peu dépendant de la bande passante de la machine.

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)	(c) 97 sec	(a) 127 sec	31%
PPN=2 OMP=4 (domaines croisés)	(d) 130 sec	(b) 180 sec	38%

```
(a) srun -xnova4 --cpu_bind=verbose,mask_cpu:0xF,0xF0 --exclusive -N 1 -n 2 ./Src/hydro
    ./Input/input_sedov_noio_10000x10000.nml

(b) srun -xnova4 --cpu_bind=verbose,none --exclusive -N 1 -n 2 ./wrapper.sh ./Src/hydro
    ./Input/input_sedov_noio_10000x10000.nml

(c) srun -xnova4 --cpu_bind=verbose,mask_cpu:0xF,0xF0 --exclusive -N 1 -n 2
    ./Src/hydro_orig ./Input/input_sedov_noio_10000x10000.nml

(d) srun -xnova4 --cpu_bind=verbose,none --exclusive -N 1 -n 2 ./wrapper.sh
    ./Src/hydro_orig ./Input/input_sedov_noio_10000x10000.nml
```

Qu'en concluez-vous ? Ce cas test montre une dépendance plus importante au niveau de la bande passante que le précédent (malgré le fait que ce soit le même code, le même binaire). En gros 30 à 40% du temps

d'exécution est dépendant de la bande passante. De plus, l'option `-opt-streaming-stores always` tend à augmenter fortement le trafic mémoire et pénalise les performances (30-40%).

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	157.1 cycles	251.1 cycles
1	250.5 cycles	157.4 cycles

```
srun --exclusive -p basic -N 1 -n 1 numactl --hardware
available: 2 nodes (0-1)
node 0 size: 12092 MB
node 0 free: 1114 MB
node 1 size: 12120 MB
node 1 free: 10300 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

Normalisée :

	0	1
0	10 (numactl : 10)	16 (numactl : 20)
1	16 (numactl : 20)	10 (numactl : 10)

Les distances fournies par la commande `numactl` ne sont qu'indicatives et ne sont pas forcément égales à la latence tel qu'on peut la mesurer avec un benchmark adapté.

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: 2 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	5599 MB/s	5558 MB/s	5450 MB/s	5310 MB/s
GPU device 1	5577 MB/s	5550 MB/s	5450 MB/s	5310 MB/s

Félicitations ! et …