COMPUTACIÓ D'ALTES PRESTACIONS – PAC 2

Semestre Setembre 2018

- Presentació i objectius
- Enunciat de la pràctica
- Criteris d'avaluació
- Format de lliurament
- Data de lliurament

Presentació i objectius

Objectius

Aquesta pràctica té com a objectius introduir els conceptes bàsics del model de programació OpenMP, i del seu entorn d'execució. Per a la realització es posaran en pràctica alguns dels conceptes presentats en aquesta assignatura i tècniques d'avaluació de rendiment.

Presentació de la pràctica

Cal lliurar els fitxers amb el codi font realitzat i un document amb les respostes a les preguntes formulades i els comentaris que considereu. Tota la codificació es realitza exclusivament en llenguatge C amb les extensions associades a OpenMP.

Restriccions de l'entorn

Tot i que el desenvolupament de la pràctica és molt més interessant utilitzant dotzenes de computadors, s'assumeix que inicialment tindreu accés a un nombre força reduït de computadors amb diversos nuclis per node.

Material per a la realització de la pràctica

En els servidors de la UOC teniu el programari necessari per executar MPI. Els compiladors de C actuals incorporen les extensions per OpenMP per defecte.

S'espera que es produeixi debat en el fòrum de l'assignatura per aprofundir en l'ús dels models de programació i els sistemes paral·lels proporcionats.

Enunciat de la pràctica

1. Fonaments d'OpenMP

La primera part d'aquest treball consisteix en la comprensió de com programar i executar programes OpenMP simples.

El nostre primer programa és de l'estil "hello world", com es mostra a continuació:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
int nthreads, tid;

#pragma omp parallel private(nthreads, tid)
  {

  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }
  }
}
```

Tingueu en compte que aquest exemple segueix la sintaxi bàsica OpenMP:

```
#include "omp.h" int main ()
{
     int var1, var2, var3;
    // Serial code
    ...
    // Beginning of parallel section.
    // Fork a team of threads. Specify variable scoping
    #pragma omp parallel private(var1, var2) shared(var3)
    {
       // Parallel section executed by all threads
       ...
       // All threads join master thread and disband
    }
    // Resume serial code ...
}
```

2. Executant OpenMP

El codi mostrat anteriorment pot ser compilat amb la següent opció:

```
gcc -fopenmp hello_omp.c -o hello_omp
```

S'espera que el binari obtingut s'executi **utilitzant SGE**; No obstant això, el següent exemple mostra com executar un programa OpenMP amb diferents números de fils OpenMP.

```
[ivan@eimtarqso]$ gcc -fopenmp hello_omp.c -o hello_omp

[ivan@eimtarqso]$ ./hello_omp
Hello World from thread = 2
Hello World from thread = 1
Hello World from thread = 3
Hello World from thread = 0
Number of threads = 4
##(by default OpenMP uses: OpenMP threads = CPU cores)

[ivan@eimtarqso]$ export OMP_NUM_THREADS=3
##(we specify the number of threads to be used with the environment variable)

[ivan@eimtarqso]$ ./hello_omp
Hello World from thread = 1
Hello World from thread = 0
Number of threads = 3
Hello World from thread = 2

[ivan@eimtarqso]$ export OMP_NUM_THREADS=2

[ivan@eimtarqso]$ ./hello_omp
Hello World from thread = 1
Hello World from thread = 0
Number of threads = 2
```

Tingueu en compte que el nombre de fils OpenMP pot ser codificat en els seus programes, però ens interessen els codis que utilitzen la variable OMP_NUM_THREADS per especificar el nombre de fils OpenMP a utilitzar.

Els següents scripts il·lustren dues maneres possibles d'executar un programa utilitzant OpenMP SGE:

```
##(omp1.sge)

#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N omp1
#$ -o omp1.out.$JOB_ID
#$ -e omp1.out.$JOB_ID
#$ -pe openmp 3

export OMP_NUM_THREADS=$NSLOTS
./hello_omp
```

En l'exemple anterior definim el nombre de nuclis de CPU que s'assignarà al treball a través de "**#$ -pe openmp 3**" i fem servir $ NSLOTS (que pren el valor proporcionat a "OpenMP -pe") per especificar el nombre de fils OpenMP.

```
##(omp2.sge)

#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N omp1
#$ -o omp1.out.$JOB_ID
#$ -e omp1.out.$JOB_ID
#$ -pe openmp 3

./hello_omp
```

Aquest segon exemple se suposa que la variable OMP_NUM_THREADS es defineix externament. La següent comanda es pot utilitzar per enviar la tasca:

```
[ivan@eimtarqso]$ qsub -v OMP_NUM_THREADS='3' h.sge
```

Preguntes:

1. **Quants fils OpenMP faries servir al clúster UOC? Per què? Explicar els inconvenients de la utilització d'altres opcions.**

3. Paral.lelisme de dades

A l'exemple que es mostra a continuació mostra l'ús de les clàusules "parallel" i "for" de dues maneres diferents però equivalents.

```
#pragma omp parallel
{
     #pragma omp for
     {
          for(i=0; i<N; i++){
               c[i] = b[i]+a[i];
          }
     }
}


##(we will target this second way moving forward)

#pragma omp parallel for
{
     for(i=0; i<N; i++){
          c[i] = b[i]+a[i];
     }
}
```

L'exemple mostrat anteriorment realitza la suma de dos vectors.

Preguntes:

2. **Com implementarieu un programa per calcular la suma dels elements d'un vector "a" (sum=a[0]+…+a[N-1]) utilitzant la clausula reduction?**

4. Paral·lelisme funcional

En aquesta PAC també pot utilitzar el paral·lelisme de tasques, si és necessari (no és obligatori). OpenMP suporta tasques, que, a diferència de paral·lelisme de dades on s'aplica la mateixa operació per a tots els elements, permet que diferents seccions del codi pugin executar diferents operacions sobre diferents elements de dades. Una tasca OpenMP té un codi per executar, un entorn de dades, i una thread assignat que executa el codi i utilitza les dades.

El següent codi il·lustra l'ús de seccions OpenMP (veure més detalls en els materials proporcionats al campus de la UOC).

```
#pragma omp parallel shared(n,a,b,c,d) private(i)
{
     #pragma omp sections nowait
          #pragma omp section
               for (i=0; i<n; i++)
                    d[i] = 1.0/c[i];
          #pragma omp section
               for (i=0; i<n-1; i++)
                    b[i] = (a[i] + a[i+1])/2;
     } /*-- End of sections --*/
} /*-- End of parallel region
```

Tingueu en compte que aquest codi executarà en paral·lel dos bucles diferents (seqüencialment) sobre diferents matrius usant un thread OpenMP per a cadascun d'ells.

5. Avaluació de rendiment (OpenMP)

En aquesta PAC veurem com utilitzar eines bàsiques d'avaluació de rendiment per a aplicacions de memòria compartida com ara OpenMP.

Utilitzarem un exemple il·lustratiu basat en una multiplicació de matrius sense cap optimització. Trobareu amb aquest enunciat dos programes diferents que implementen la multiplicació de matrius (mm.c i mm2.c). La diferència entre aquests dos és que en la segona versió hem intercanviat els índexs dels bucles exterior i més interior com es mostra a continuació:

mm.c

```
(...)
for (i=0;i<SIZE;i++)
 for(j=0;j<SIZE;j++)
  for(k=0;k<SIZE;k++)
    mresult[i][j]=mresult[i][j] + matrixa[i][k]*matrixb[k][j];
(...)
```

mm2.c

```
(...)
for (k=0;k<SIZE;k++)
 for(j=0;j<SIZE;j++)
  for(i=0;i<SIZE;i++)
    mresult[i][j]=mresult[i][j] + matrixa[i][k]*matrixb[k][j];
(...)
```

Per compilar aquests exemples només us cal utilitzar

```
gcc mm.c –o mm
```

Amb el tamany de matriu per defecte (SIZE) igual a 1000 obtindreu un temps d'execució ràpid però que us permetrà observar diferències entre les dues versions de la multiplicació de matrius.

Utilitzant la comanda `time`, (per exemple, `time ./mm`) podreu observar una diferència en el temps d'execució d'un 15% aproximadament.

Nota:

Recordeu que s'espera que feu les vostres execucions mitjançant el sistema de cues, si no ho feu així pot passar que tingueu degradació de rendiment totalment aleatori degut a la compartició dels recursos (per exemple degut a contenció de memòria).

És evident que hi ha un impacte en el rendiment de l'execució de la multiplicació pel fet de modificar els índexs dels bucles i, per tant, de com estem accedint a les dades en memòria. Per tant, en aquest cap l'eina del sistema time no és suficient per entendre (o comprovar quantitativament) aquesta situació.

Nota:

No optimitzeu la compilació del codi proporcionat (NO utilitzeu opcions com ara –O3) ja que les optimitzacions automàtiques reduiran els temps d'execucions i faran modificacions típiques (fer alignment, loop unrolling, etc.) que no us permetran seguir aquesta part de la PAC. Recordeu que estem treballant un exemple il·lustratiu.

Polítiques de planificació

En aquesta secció de la PAC es demana que implementeu la versió paral·lela dels programes proporcionats ("mm.c" i "mm2.c") usant OpenMP.

Una vegada que s'hagi paral·lelitzat l'aplicació es demana que feu versions per a diferent polítiques de planificació (static, dynamic, guided). Compareu el temps d'execució amb les diferents polítiques. Tingueu en compte que caldrà que el problema sigui suficientment gran com per poder apreciar diferències.

Preguntes:

3. **Proporcioneu la versió paral·lela dels codis (mm.c i mm2.c) utilitzant les diferents polítiques de planificació. Expliqueu les decisions d'implementació que heu pres.**

Per estudiar el comportament d'aquests programes i l'impacte en la utilització dels recursos utilitzarem PAPI (Performance Application Programming Interface). PAPI és una interfície molt útil que permet utilitzar comptador hardware que hi ha disponibles en la majoria de microprocessadors moderns.

Us proporcionem un parell d'exemples de com utilitzar PAPI però s'espera que feu la vostra pròpia cerca d'informació i exemples. Podeu començar a través del següent link online:

http://icl.cs.utk.edu/papi/

PAPI està disponible al clúster de la UOC en el següent directori: `/export/apps/papi/`

En l'exemple `flops.c` i `flops2.c` que es proporcionen amb l'enunciat d'aquesta PAC es mostra un exemple d'utilització de PAPI. Aquests exemples utilitzen una interfície de PAPI que proporciona els MFLOPS obtinguts en l'execució del programes. També us proporciona el temps d'execució de forma més acurada i el temps de processador.

Per tal de compilar-los podeu fer servir la següent comanda:

```
gcc -I/export/apps/papi/include/ flops.c /export/apps/papi/lib/libpapi.a -o flops
```

Veureu que observem una diferència significativa en el FLOPS obtinguts per a les dues versions de la multiplicació de matrius.

Preguntes

4. **Per què penseu que és aquesta diferència en el temps d'execució i en els MFLOPs?**

En la última versió dels exemples proporcionats (`counters.c` i `counters2.c`) s'utilitzen comptadors hardware de forma explícita. Per tal de compilar-los només heu de fer com anteriorment:

```
gcc -I/export/apps/papi/include/ counters.c /export/apps/papi/lib/libpapi.a -o counters
```

Veureu que la sortida de l'execució d'aquests us proporciona la quantitat total d'instruccions (PAPI_TOT_INS) i operacions en coma flotant (PAPI_FP_OPS).

Podreu observar que la sortida us indica que els dos exemples estan executant el mateix número d'operacions en coma flotant (tot i que podeu veure una certa variabilitat en el número total d'instruccions). Clarament s'han analitzar altres recursos relacionats amb l'accés a memòria per entendre millor l'impacte de l'intercanvi dels índexs dels bucles.

En aquesta PAC es demana que utilitzeu altres comptadors hardware per estudiar els exemples proporcionats. Podeu consultar el significat dels comptadors utilitzats en els exemples de la PAC i també el conjunt total de comptadors hardware disponibles als processadors del clúster de la UOC i els events disponibles per a consultar mitjançant la següent comanda:

`/export/apps/papi/bin/papi_avail`

Obtindreu el següent resultat (parcial - la sortida està tallada):

```
[@eimtarqso]$ /export/apps/papi/bin/papi_avail
Available PAPI preset and user defined events plus hardware information.
--------------------------------------------------------------------------------
PAPI Version             : 5.5.0.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Intel(R) Xeon(R) CPU           E5603  @ 1.60GHz (44)
CPU Revision             : 2.000000
CPUID Info               : Family: 6  Model: 44  Stepping: 2
CPU Max Megahertz        : 1600
CPU Min Megahertz        : 1200
Hdw Threads per core     : 1
Cores per Socket         : 4
Sockets                  : 1
NUMA Nodes               : 1
CPUs per Node            : 4
Total CPUs               : 4
Running in a VM          : no
Number Hardware Counters : 7
Max Multiplex Counters   : 32
--------------------------------------------------------------------------------
================================================================================
  PAPI Preset Events
================================================================================
    Name         Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   No    Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No    Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   Yes   Level 2 data cache misses
PAPI_L2_ICM  0x80000003  Yes   No    Level 2 instruction cache misses
PAPI_L3_DCM  0x80000004  No    No    Level 3 data cache misses
PAPI_L3_ICM  0x80000005  No    No    Level 3 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes   Yes   Level 1 cache misses
PAPI_L2_TCM  0x80000007  Yes   No    Level 2 cache misses
PAPI_L3_TCM  0x80000008  Yes   No    Level 3 cache misses
PAPI_CA_SNP  0x80000009  No    No    Requests for a snoop
PAPI_CA_SHR  0x8000000a  No    No    Requests for exclusive access to shared cache line
PAPI_CA_CLN  0x8000000b  No    No    Requests for exclusive access to clean cache line
PAPI_CA_INV  0x8000000c  No    No    Requests for cache line invalidation
PAPI_CA_ITV  0x8000000d  No    No    Requests for cache line intervention
PAPI_L3_LDM  0x8000000e  Yes   No    Level 3 load misses
PAPI_L3_STM  0x8000000f  No    No    Level 3 store misses
(...)
--------------------------------------------------------------------------------
Of 108 possible events, 58 are available, of which 14 are derived.

avail.c                                  PASSED
```

Preguntes

5. **Quins comptadors hardware faríeu servir per estudiar les diferències entre les dues implementacions? Per què?**
6. **Proporcioneu el codi on feu servir comptadors hardware per quantificar les diferències entre els dos exemples proporcionats. Proporcioneu els resultats obtinguts.**
7. **Proporcioneu versions paral·leles (OpenMP) i els resultats obtinguts utilitzant PAPI pels dos exemples proporcionats.**

Criteris d'avaluació

Es valorarà el correcte ús del model de programació OpenMP. També es tindrà en compte la correcta programació, estructura i funcionament dels programes i la discussió de l'avaluació de rendiment.

Format de lliurament

Es crearà un fitxer en format PDF amb tota la informació.

Si els scripts o codis són llarg per afegir-los al document de l'entrega (no s'esperen que siguin massa sofisticats), podeu adjuntar-los amb la comanda següent:

```
$ tar cvf tot.tar fitxer1 fitxer2 ...
```

es crearà el fitxer "tot.tar" on s'hauran emmagatzemat els fitxers "fitxer1", fitxer2" i . . .

Per llistar la informació d'un fitxer `tar` es pot utilitzar la comanda següent:

```
$ tar tvf tot.tar
```

Per extraure la informació d'un fitxer `tar` es pot utilitzar:

```
$ tar xvf tot.tar
```

El nom del fitxer tindrà el format següent: "Cognom1Cognom2PAC2.pdf" (o *.tar). Els cognoms s'escriuran sense accents. Per exemple, l'estudiant Marta Vallès i Marfany utilitzarà el nom de fitxer següent: `VallesMarfanyPAC2.pdf (o *.tar)`

Data de lliurament

Dimarts 6 de Novembre de 2018.