Index

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 realitzar 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.

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

Primer copiam els fitxers de la PAC2 a la carpeta que he creat 'pac2' a meu '/home/capa20':
manelmdiaz@manelmdiaz-Desktop-Ubuntu:~/Descargas/PAC2$ scp -P 55000 * capa20@eimtarqso.uoc.edu:/home/capa20/pac2

Creem el fitxer hello_omp.c amb el codi de la PAC2 i el compilem.
[capa20@eimtarqso pac2]$ gcc -fopenmp hello_omp.c -o hello_omp

Creem el script 'omp1.sge' per cridar al programa amb OpenMP SGE

Pregunta 1

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

El node Eimtarqso disposa de 10 nodes de còmput, cadascun amb 4 cores, per tant el número de fils que utilitzaria seria el mateix que el nombre de cores, que son 4 fils.

En termes generals el rendiment de l'execució d'un programa OpenMP, quan s'utilitzen més fluxos que processadors disponibles, és inferior al que s'aconsegueix utilitzant un flux per processador. Per tant utilitzaria el mateix nombre de cores que disposa cada node de còmput.

L'inconvenient d'utilitzar més fluxos que processadors disponibles es degut a la sobrecàrrega en la creació i gestió de fluxos. En el cas contrari d'utilitzar menys processadors no aprofitaríem la execució

paral·lela.

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.

Pregunta 2

2.Com implementaríeu un programa per calcular la suma dels elements d'un vector «a» (sum=a[0]+... +a[N-1]) utilitzant la clàusula reduction?

He creat el fitxer: 'pac2_p2.c' on he definit el vector «a» amb 10 elements amb valor 10 i que llegeixi N=10 valors del vector, sumant els seus valors.

Es compila amb 'gcc -fopenmp pac2_p2.c -o pac2_p2'
Creem el script 'pac2_p2.sge' amb 4 fils.
Executem script 'qsub pac2_p2.sge'

Codi 'pac2_p2.c'

```
[capa20@eimtarqso pac2]$ cat pac2_p2.c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
        int a[]={10,10,10,10,10,10,10,10,10,10};
        int N=10;
        int result=0, i=0;
```

```
        int tid;

        #pragma omp parallel for private(tid, i) reduction (+:result) shared(a)
                for(i=0; i<N; i++)
                {
                result = result + a[i];
                tid = omp_get_thread_num();
                printf("Thread number = %d. Resultat parcial = %d\n",tid, result);
                }
        printf("Resultat final = %d\n",result);
}
```

Script 'pac2_p2.sge'

```
[capa20@eimtarqso pac2]$ cat pac2_p2.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N pac2_p2
#$ -o pac2_p2.out.$JOB_ID
#$ -e pac2_p2.out.$JOB_ID
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./pac2_p2
```

Resultat:

```
[capa20@eimtarqso pac2]$ cat pac2_p2.out.433419
Thread number = 3. Resultat parcial = 10
Thread number = 0. Resultat parcial = 10
Thread number = 0. Resultat parcial = 20
Thread number = 0. Resultat parcial = 30
Thread number = 1. Resultat parcial = 10
Thread number = 1. Resultat parcial = 20
Thread number = 1. Resultat parcial = 30
Thread number = 2. Resultat parcial = 10
Thread number = 2. Resultat parcial = 20
Thread number = 2. Resultat parcial = 30
Resultat final = 100
```

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 complementen la multiplicació de matrius (mm.c i mmc2.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];
(...)
```

mmc2.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 la mida 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 índex dels bucles i, per tant, de com estem accedint a les dades en memòria. Per tant, en aquest cas 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.

Pregunta 3

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.

```
[capa20@eimtarqso pac2]$ cat mm_p3.c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <memory.h>
#include <malloc.h>
#include <omp.h>

int main(int argc, char **argv) {

  int SIZE = atoi(argv[1]);

  float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
  int i,j,k;

/* Begin parallel region */

/*Initialize the Matrix arrays */
#pragma omp parallel private(i)
{
  /* Parallelize the initialization of Matrix arrays */
  /* Indice i is private for each trhead to iterate independly */
  #pragma omp parallel for schedule (runtime)
  for ( i=0; i<SIZE*SIZE; i++ ){
    mresult[0][i] = 0.0;}
```

```
  #pragma omp parallel for schedule (runtime)
  for ( i=0; i<SIZE*SIZE; i++ ){
    matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1; }

}

/* Matrix-Matrix multiply */

#pragma omp parallel shared(matrixa, matrixb, mresult) private(i,j,k)
{

 /* Parallelize the outer loop which evaluates each row of matrix resultant */
 /* Matrices are shared among threads*/
 /* Indices are privates for each threads to iterate independly */
  #pragma omp for schedule (runtime)
  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];
}

/* End of parallel region */

  printf("Done.\n");
exit(0);
}
```

He inclòs la llibreria ‘omp.h’ i realitzat diverses modificacions comentades al mateix codi ressaltades en Negreta.

Les polítiques de planificació tractades als apunts son FCFS (Firs-come-first-serve), FCFS amb EASY backfilling, però Open MP disposa dels següents tipus de planificació: static, dynamic, guided, auto, runtime, Monotonic, nonmonotonic i simd).

OpenMP disposa de la variable d’entorn OMP_SCHEDULE que inicialitza la variable run-sched-var, la qual es utilitzada quan al codi s’utilitza el tipus de planificació ‘runtime’. Els valors de planificació podem ser static, dynamic, guided o auto, per tant utilitzarem al codi i provarem per als valors disponibles excepte auto on desconeixerem quina política s’utilitza.

Creo un script que cridaria a l’script de cua sge per a cadascuna de les tres polítiques i calculi el temps en nanosegons. Dono permisos d’execució al script i l’executo amb ./mm_p3.sh. He modificar els scripts i el codi per permetre definir al la mida de la matriu al script mm_p3.sh i així veure la diferencia de temps amb matrius més grans.

```
[capa20@eimtarqso pac2]$ cat mm_p3.sh
#!/bin/bash
planPolicies=('static' 'dynamic' 'guided')
for nthreads in 1 2 3 4; do
        for schedule in static dynamic guided; do
                for size in 500 1500 2000 2500; do
                        qsub -v size=$size,nthreads=$nthreads,schedule=$schedule mm_p3.sge
                        echo "$size - $nthreads - $schedule"
```

```
                done
            done
done
```

```
[capa20@eimtarqso pac2]$ cat mm_p3.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N mm_p3_jobname
#$ -o mm_p3_$JOB_ID.out
#$ -e mm_p3_$JOB_ID.err
#$ -pe openmp 4
export OMP_SCHEDULE=$schedule
export OMP_NUM_THREADS=$nthreads
timeIni=$(date +%s%3N)
./mm_p3 $size
timeFin=$(date +%s%3N)
echo "$schedule, $size, $nthreads, $((timeFin-timeIni)) [ms]" >> mm_p3_result.out
```

Compilem i creem l'executable mm_p3

```
[capa20@eimtarqso pac2]$ gcc -fopenmp mm_p3.c -o mm_p3
```

Executem el script mm_p3.sh

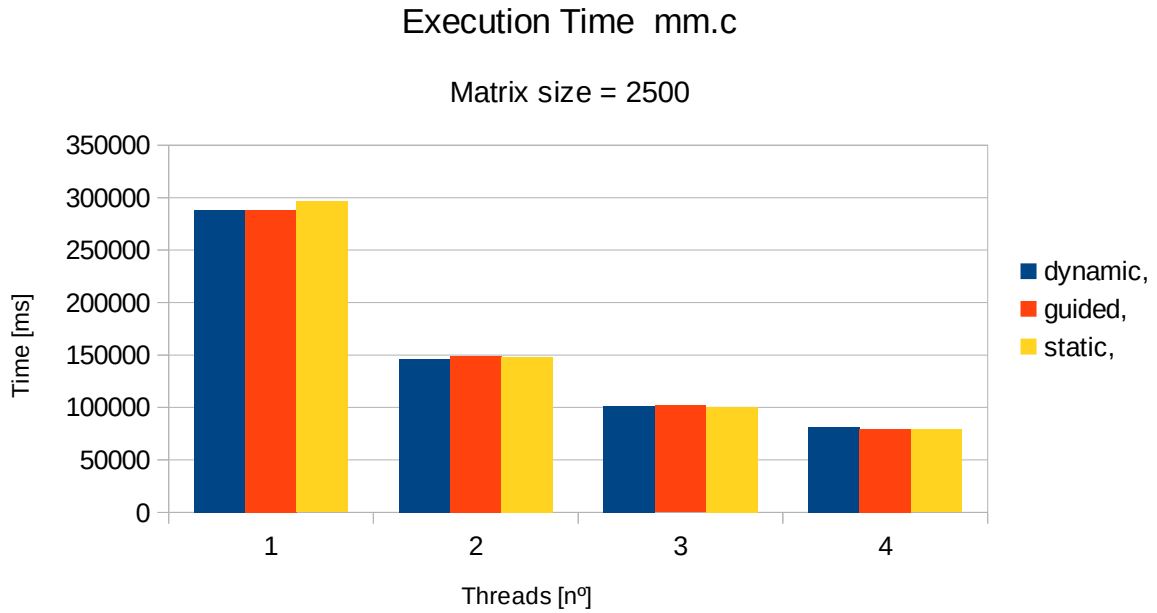
```
[capa20@eimtarqso pac2]$ ./mm_p3.sh
```

Un cop el resultat de la comanda qstat no dona cap treball executant, comprovem que el resultat de tots els treballs ha resultat amb èxit, mostrem per pantalla la sortida dels fitxers de cada job, on ha de sortir Done. per cada fitxer.

```
[capa20@eimtarqso pac2]$ cat mm_p3_4345*
Done.
```

Consultem els resultats desats al fitxer 'mm_p3_result.out' i els guardem en una fulla de càlcul, fitxer 'PAC2_p3.ods' per sumaritzar els resultats i mostrar-ho en una gràfica.

```
[capa20@eimtarqso pac2]$ cat mm_p3_result.out
dynamic, 500, 1, 1734 [ms]
static, 500, 1, 1899 [ms]
guided, 500, 1, 1900 [ms]
static, 1500, 1, 62223 [ms]
dynamic, 1500, 1, 62204 [ms]
static, 500, 2, 1075 [ms]
guided, 1500, 1, 62228 [ms]
```



Fem el mateix per al mm2.c, per el qual he afegit la paral·lelització després del for per [k] i [j] on es multiplica la matriu per cada [i] en paral·lel.

```
[capa20@eimtarqso pac2]$ cat mm2.c
        #include <stdlib.h>
        #include <stdio.h>
        #include <unistd.h>
        #include <errno.h>
        #include <sys/types.h>
        #include <memory.h>
        #include <malloc.h>
        #include <omp.h>

        int main(int argc, char **argv) {

          int SIZE = atoi(argv[1]);

          float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
          int i,j,k;

        /* Begin parallel region */

        /*Initialize the Matrix arrays */
        #pragma omp parallel private(i)
        {
          /* Parallelize the initialization of Matrix arrays */
          /* Indice i is private for each trhead to iterate independly */
          #pragma omp parallel for schedule (runtime)
          for ( i=0; i<SIZE*SIZE; i++ ){
            mresult[0][i] = 0.0;}
          #pragma omp parallel for schedule (runtime)
          for ( i=0; i<SIZE*SIZE; i++ ){
```

```
    matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1; }

}

/* Matrix-Matrix multiply */

#pragma omp parallel shared(matrixa, matrixb, mresult) private(i,j,k)
{

 /* Parallelize the inner loop to evaluate the matrix resultant */
 /* Matrices are shared among threads*/
 /* Indices are privates for each threads to iterate independly */
  for (k=0;k<SIZE;k++)
   for(j=0;j<SIZE;j++)
    #pragma omp for schedule (runtime)
     for(i=0;i<SIZE;i++)
       mresult[i][j]=mresult[i][j] + matrixa[i][k]*matrixb[k][j];
}

/* End of parallel region */

  printf("Done.\n");
exit(0);
}
```

Compilem i creem l'executable mm_p3

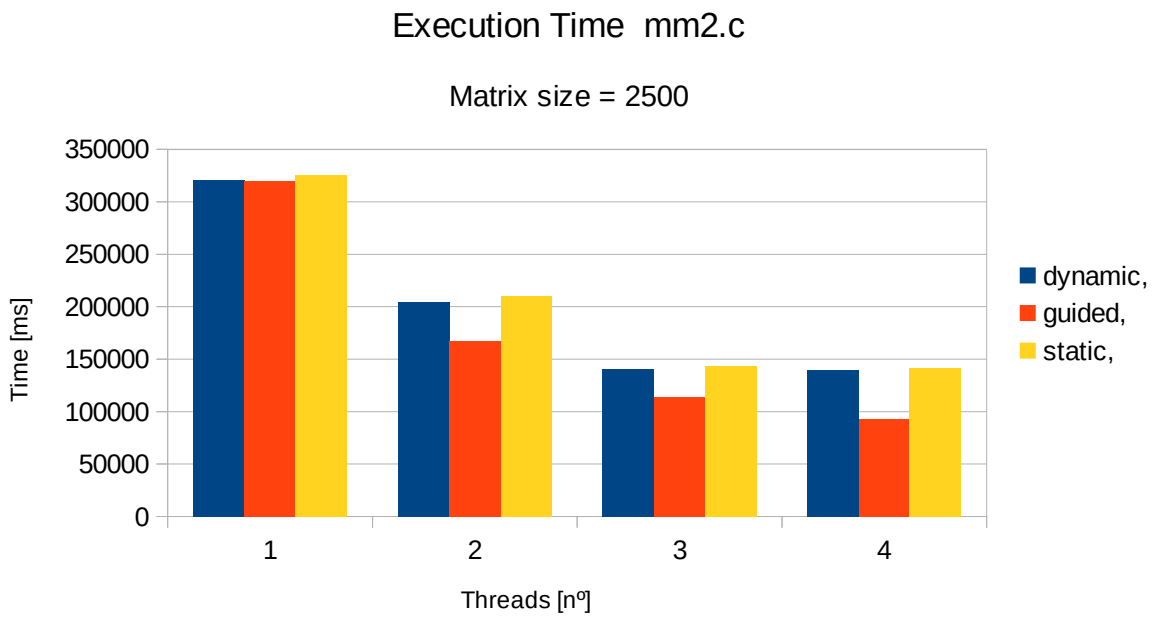
[capa20@eimtarqso pac2]$ gcc -fopenmp mm2_p3.c -o mm2_p3

Executem el script mm2_p3.sh que crida al nou executable.

[capa20@eimtarqso pac2]$ ./mm2_p3.sh

Consultem els resultats desats al fitxer 'mm2_p3_result.out' i els guardem al fitxer 'PAC2_p3.ods' per sumaritzar els resultats i mostrar-ho en una gràfica.

[capa20@eimtarqso pac2]$ cat mm2_p3_result.out
static, 500, 1, 1929 [ms]



Per estudiar el comportament d'aquests programes i l'impacte en la utilització dels recursos utilitzarem PAPI (Performance Application Programmig 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 pòpia cerca d'informació i exemples. Podeu començar a través del link 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 dels 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.

Compilem els programes flops.c i flops2.c

gcc -I/export/apps/papi/include/ flops.c /export/apps/papi/lib/libpapi.a -o flops
gcc -I/export/apps/papi/include/ flops2.c /export/apps/papi/lib/libpapi.a -o flops2

Creem un script per cridar a flops i un altre per flops2.

[capa20@eimtarqso pac2]$ cat flops.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N flops
#$ -o flops.out.$JOB_ID

```
#$ -e flops.out.$JOB_ID
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./flops

[capa20@eimtarqso pac2]$ cat flops2.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N flops2
#$ -o flops2.out.$JOB_ID
#$ -e flops2.out.$JOB_ID
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./flops2

[capa20@eimtarqso pac2]$ qsub flops.sge
Your job 436373 ("flops") has been submitted

[capa20@eimtarqso pac2]$ cat *4363743
Real_time:        11.251243
Proc_time:        11.224405
Total flpins:     2000941725
MFLOPS:           178.267059

[capa20@eimtarqso pac2]$ qsub flops2.sge
Your job 436374 ("flops2") has been submitted

[capa20@eimtarqso pac2]$ cat *436374*
Real_time:        12.984845
Proc_time:        12.974654
Total flpins:     2003194599
MFLOPS:           154.392914
flops2.c  PASSED
```

Pregunta 4

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

Primer cal tenir tenir present com s'emmagatzema a memòria un array, que en llenguatge C els elements d'un array, s'emmagatzemen en l'ordre de les files de forma seqüencial i adjacent en la memòria. L'ordre del bucle del nostre programa marcarà en quin ordre s'accedeix a les dades dels diferents arrays però com la mida de la memòria dels diferents nivells pot ser diferent, totes les dades poden no ubicar-se a la cache de nivell1 i part s'emmagatzemi al nivell 2 i 3 ,repercutint en un augment de temps d'accés al ser més lentes a cada nivell que augmenta.

En el codi flops.c l'ordre del bucle es ijk, on el valor de la variable k es el que més sovint canvia al estar més endins de bucle. A mesura que k augmenta, s'accedirà al següent element de la fila de 'matrixa' i el següent element de la columna de 'matrixb'.

```
/* Matrix-Matrix multiply */
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];
```

En el codi flops2.c on l'ordre del bucle es kji i al bucle més intern a 'matrixa' s'accedix per fila de forma seqüencial, però no per a 'matrixb' s'accedeix per columna i cal un salt per accedir a la dada.

```
 /* Matrix-Matrix multiply */
 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];
```

Tenint en compte el tipus d'accés a les dades, per fila o columna, l'accés per columna implica salts en les dades més grans i major possibilitat que hi hagin errors i l'accés a la dades sigui en nivells de cache més lentes. Per tant, el codi flops2.c es susceptible que requereix més temps d'execució perquè accedeix a dues matrius per columna i el codi flops.c només accedeix per columna a una matriu. Adjunto resum del tipus d'accés al bucle interior segons l'ordre del bucle:

	mresult	matrixa	matrixb
Bucle ijk	Fixe	fila	columna
Bucle kji	columna	columna	fixe

Respecte als MFLOPs, que son el número d'operacions en coma flotant per segon, també depen del temps en que les instruccions i dades estan disponibles per executar-se i en el codi flops2.c on l'accés a dades es pitjor també es redueix els MFLOPs.

En l'última versió dels exemples proporcionats (counter.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
```

[capa20@eimtarqso pac2]$ gcc -I/export/apps/papi/include/ counters.c /export/apps/papi/lib/libpapi.a -o counters

[capa20@eimtarqso pac2]$ qsub counters.sge
Your job 436375 ("counters") has been submitted

[capa20@eimtarqso pac2]$ cat *436375*
Total hardware flops = 9219
Total instructions 33073987255

[capa20@eimtarqso pac2]$ gcc -I/export/apps/papi/include/ counters2.c /export/apps/papi/lib/libpapi.a -o counters2

[capa20@eimtarqso pac2]$ qsub counters2.sge
Your job 436376 ("counters2") has been submitted

[capa20@eimtarqso pac2]$ cat *436376*
Total hardware flops = 9219
Total instructions 33073992554

Pregunta 5

5. Quins comptadors hardware faríeu servir per estudiar les diferències entres les dues implementacions? Per què?

A nivell general per analitzar la eficiència del codi, calcularen número d'instruccions realitzades per cicle,

anomenat IPC (Instructions per cycle). Quan més gran sigui el IPC, més eficient es la execució del codi per l'arquitectura on s'executa. Això ens permet comparar el rendiment d'execució de diferents codis. S'utilitzen els comptadors:

- PAPI_TOT_CYC 0x8000003b Yes No Total cycles
- PAPI_TOT_INS 0x80000032 Yes No Instructions completed

Tenint en compte que l'intercanvi dels índexs dels bucles està relacionat en la forma d'accedir als elements dels arrays i per tant com s'accedeix a la memòria, em focalitzaré en els comptadors relacionats amb l'accés a memòria cache i la traducció d'adreces (Translation looaside buffers). Al accedir a memòria per cercar les instruccions i dades, els errors en l'accés o traducció suposen una penalització en temps, per tant analitzarem els comptadors on es quantifiquen el número d'accessos i errors ('misses') per als diferents nivells de cache i així tindre el rati d'error. Proposaria utilitzar els següents:

- PAPI_L1_DCA 0x80000040 No No Level 1 data cache accesses
- PAPI_L2_DCA 0x80000041 Yes No Level 2 data cache accesses
- PAPI_L3_DCA 0x80000042 Yes Yes Level 3 data cache accesses
- PAPI_L1_DCM 0x80000000 Yes No Level 1 data cache misses
- PAPI_L2_DCM 0x80000002 Yes Yes Level 2 data cache misses
- PAPI_L3_DCM 0x80000004 No No Level 3 data cache misses
- PAPI_L1_ICA 0x8000004c Yes No Level 1 instruction cache accesses
- PAPI_L2_ICA 0x8000004d Yes No Level 2 instruction cache accesses
- PAPI_L3_ICA 0x8000004e Yes No Level 3 instruction cache accesses
- PAPI_L1_ICM 0x80000001 Yes No Level 1 instruction cache misses
- PAPI_L2_ICM 0x80000003 Yes No Level 2 instruction cache misses
- PAPI_L3_ICM 0x80000005 No No Level 3 instruction cache misses
- PAPI_TLB_IM 0x80000015 Yes No Instruction translation lookaside buffer misses
- PAPI_TLB_DM 0x80000014 Yes No Data translation lookaside buffer misses
- PAPI_TLB_TL 0x80000016 Yes Yes Total translation lookaside buffer misses
- PAPI_TLB_SD 0x8000001e No No Translation lookaside buffer shootdowns

Per aquesta PAC el hardware del clúster es incompatible amb diversos comptadors, que calculen els accessos a dades de nivell1, els errors d'instruccions i dades de nivell3. Per tant, no podríem calcular el rati d'erros per a tots els nivells. Existeixen comptadors generals que no diferencien dades d'instruccions però el clúster es incompatible amb el comptador d'accessos a la cache nivell1. Com alternatives tenim:

- Comparar el número d'errors cache per instruccions i dades de nivell 1 i 2.
  - PAPI_L1_DCM 0x80000000 Yes No Level 1 data cache misses
  - PAPI_L2_DCM 0x80000002 Yes Yes Level 2 data cache misses
  - PAPI_L1_ICM 0x80000001 Yes No Level 1 instruction cache misses
  - PAPI_L2_ICM 0x80000003 Yes No Level 2 instruction cache misses
- Comparar el número total d'erros cache de nivell 1,2 i 3.
  - 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

Pregunta 6

6. Proporcioneu el codi on feu servir comptadors hardware per quantificar les diferències entre els dos exemples proporcionats. Proporcioneu els resultats obtinguts.

A continuació el codi de counters.c i counter2.c amb comptadors hardware. Ressaltat en negreta el codi modificat.

[capa20@eimtarqso pac2]$ cat counters_papi.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <memory.h>
#include <malloc.h>
#include <papi.h>

#define SIZE 1000

int main(int argc, char **argv) {

  float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
  int i,j,k;
  int events[3] = {PAPI_L1_TCM, PAPI_L2_TCM,PAPI_L3_TCM}, ret;
  long long values[3];

  if (PAPI_num_counters() < 3) {
          fprintf(stderr, "No hardware counters here, or PAPI not supported.\n");
          exit(1);
  }

  if ((ret = PAPI_start_counters(events, 3)) != PAPI_OK) {
          fprintf(stderr, "PAPI failed to start counters: %s\n", PAPI_strerror(ret));
          exit(1);
  }

  /* Initialize the Matrix arrays */
  for ( i=0; i<SIZE*SIZE; i++ ){
    mresult[0][i] = 0.0;
    matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1; }

  /* Matrix-Matrix multiply */
  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];

  if ((ret = PAPI_read_counters(values, 3)) != PAPI_OK) {
          fprintf(stderr, "PAPI failed to read counters: %s\n", PAPI_strerror(ret));
          exit(1);
  }
  printf("Cache Misses Level1: %lld\n",values[0]);
  printf("Cache Misses Level2: %lld\n",values[1]);
  printf("Cache Misses Level3: %lld\n",values[2]);
  exit(0);
}

[capa20@eimtarqso pac2]$ cat counters2_papi.c
#include <stdlib.h>
#include <stdio.h>
```

```
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <memory.h>
#include <malloc.h>
#include <papi.h>

#define SIZE 1000

int main(int argc, char **argv) {

  float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
  int i,j,k;
  int events[3] = {PAPI_L1_TCM, PAPI_L2_TCM,PAPI_L3_TCM}, ret;
  long long values[3];

  if (PAPI_num_counters() < 3) {
          fprintf(stderr, "No hardware counters here, or PAPI not supported.\n");
          exit(1);
  }

  if ((ret = PAPI_start_counters(events, 3)) != PAPI_OK) {
          fprintf(stderr, "PAPI failed to start counters: %s\n", PAPI_strerror(ret));
          exit(1);
  }

  /* Initialize the Matrix arrays */
  for ( i=0; i<SIZE*SIZE; i++ ){
    mresult[0][i] = 0.0;
    matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1; }

  /* Matrix-Matrix multiply */
  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];

  if ((ret = PAPI_read_counters(values, 3)) != PAPI_OK) {
          fprintf(stderr, "PAPI failed to read counters: %s\n", PAPI_strerror(ret));
          exit(1);
  }

  printf("Cache Misses Level1: %lld\n",values[0]);
  printf("Cache Misses Level2: %lld\n",values[1]);
  printf("Cache Misses Level3: %lld\n",values[2]);
  exit(0);
}
```

He compilat els programes:

```
[capa20@eimtarqso pac2]$ gcc -I/export/apps/papi/include/ counters_papi.c
/export/apps/papi/lib/libpapi.a -o counters_papi
```

```
[capa20@eimtarqso pac2]$ gcc -I/export/apps/papi/include/ counters2_papi.c
/export/apps/papi/lib/libpapi.a -o counters2_papi
```

He creat un script SGE per a cada programa i els he cridat amb el script SGE corresponent.

```
[capa20@eimtarqso pac2]$ cat counters_papi.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N counters_papi
#$ -o counters_papi.out.$JOB_ID
#$ -e counters_papi.out.$JOB_ID
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./counters_papi

[capa20@eimtarqso pac2]$ cat counters2_papi.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N counters2_papi
#$ -o counters2_papi.out.$JOB_ID
#$ -e counters2_papi.out.$JOB_ID
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./counters2_papi

[capa20@eimtarqso pac2]$ qsub counters_papi.sge
Your job 436687 ("counters_papi") has been submitted

[capa20@eimtarqso pac2]$ qsub counters2_papi.sge
Your job 436691 ("counters2_papi") has been submitted
```

El resultat obtingut l'obtenim del fitxer de sortida:

```
[capa20@eimtarqso pac2]$ cat counters_papi.out.436687
Cache Misses Level1: 1873502921
Cache Misses Level2: 73628420
Cache Misses Level3: 4918572

[capa20@eimtarqso pac2]$ cat counters2_papi.out.436691
Cache Misses Level1: 3044424197
Cache Misses Level2: 204881213
Cache Misses Level3: 13770611
```

Si comparem el resultat de counters i counters2 veiem que hi ha un increment de 'Cache misses' i per tant es el motiu de la diferència de rendiment.

	Counters_papi	Counters2_papi	Increment [%]
Cache Misses Level1	1873502921	3044424197	38,46 %
Cache Misses Level2	73628420	204881213	64,06 %
Cache Misses Level3	4918572	13770611	64,28 %

Pregunta 7

7. Proporcioneu versions paral·leles (OpenMP) i els resultats obtinguts utilitzant PAPI pels dos exemples proporcionats.

A continuació el codi de counters.c i counter2.c amb comptadors hardware. Ressaltat en negreta el codi modificat.

```
[capa20@eimtarqso pac2]$ cat counters_omp.c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <memory.h>
#include <malloc.h>
#include <papi.h>
#include <omp.h>

#define SIZE 1000

int main(int argc, char **argv) {

  float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
  int i,j,k;
  int events[3] = {PAPI_L1_TCM, PAPI_L2_TCM,PAPI_L3_TCM}, ret;
  long long values[3];

  if (PAPI_num_counters() < 3) {
          fprintf(stderr, "No hardware counters here, or PAPI not supported.\n");
          exit(1);
  }

  if ((ret = PAPI_start_counters(events, 3)) != PAPI_OK) {
          fprintf(stderr, "PAPI failed to start counters: %s\n", PAPI_strerror(ret));
          exit(1);
  }

/* Begin parallel region */

  /* Initialize the Matrix arrays */
  #pragma omp parallel private(i)
  {
```

```
   #pragma omp sections nowait
   {
   #pragma omp section
  for ( i=0; i<SIZE*SIZE; i++ ){
    mresult[0][i] = 0.0;}
   #pragma omp section
   for ( i =0;i<SIZE*SIZE; i++ ){
    matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1; }
   /* End of sections */
   }
 }
 /* End of parallel region */

 /* Matrix-Matrix multiply */
 #pragma omp parallel shared(matrixa, matrixb, mresult) private(i,j,k)
 {
   #pragma omp for
  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];
 }
 /* End of parallel region */

 if ((ret = PAPI_read_counters(values, 3)) != PAPI_OK) {
         fprintf(stderr, "PAPI failed to read counters: %s\n", PAPI_strerror(ret));
         exit(1);
 }

 printf("Cache Misses Level1: %lld\n",values[0]);
 printf("Cache Misses Level2: %lld\n",values[1]);
 printf("Cache Misses Level3: %lld\n",values[2]);
 exit(0);
}

[capa20@eimtarqso pac2]$ cat counters2_omp.c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <memory.h>
#include <malloc.h>
#include <papi.h>
#include <omp.h>

#define SIZE 1000

int main(int argc, char **argv) {

 float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
 int i,j,k;
```

```
 int events[3] = {PAPI_L1_TCM, PAPI_L2_TCM,PAPI_L3_TCM}, ret;
 long long values[3];

 if (PAPI_num_counters() < 3) {
         fprintf(stderr, "No hardware counters here, or PAPI not supported.\n");
         exit(1);
 }

 if ((ret = PAPI_start_counters(events, 3)) != PAPI_OK) {
         fprintf(stderr, "PAPI failed to start counters: %s\n", PAPI_strerror(ret));
         exit(1);
 }

/* Begin parallel region */

 /* Initialize the Matrix arrays */
 #pragma omp parallel private(i)
 {
   #pragma omp sections nowait
   {
   #pragma omp section
  for ( i=0; i<SIZE*SIZE; i++ ){
    mresult[0][i] = 0.0;}
   #pragma omp section
   for ( i =0;i<SIZE*SIZE; i++ ){
    matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1; }
  /* End of sections */
   }
 }
 /* End of parallel region */

 /* Matrix-Matrix multiply */
 #pragma omp parallel shared(matrixa, matrixb, mresult) private(i,j,k)
 {
   #pragma omp for
  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];
 }
 /* End of parallel region */

 if ((ret = PAPI_read_counters(values, 3)) != PAPI_OK) {
         fprintf(stderr, "PAPI failed to read counters: %s\n", PAPI_strerror(ret));
         exit(1);
 }

 printf("Cache Misses Level1: %lld\n",values[0]);
 printf("Cache Misses Level2: %lld\n",values[1]);
 printf("Cache Misses Level3: %lld\n",values[2]);
 exit(0);
}
```

He compilat els programes:

```
[capa20@eimtarqso pac2]$ gcc -I/export/apps/papi/include/ counters_omp.c /export/apps/papi/lib/libpapi.a -o counters_omp

[capa20@eimtarqso pac2]$ gcc -I/export/apps/papi/include/ counters2_omp.c /export/apps/papi/lib/libpapi.a -o counters2_omp
```

He creat un script SGE per a cada programa i els he cridat amb el script SGE corresponent.

```
[capa20@eimtarqso pac2]$ cat counters_omp.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N counters_omp
#$ -o counters_omp.out.$JOB_ID
#$ -e counters_omp.out.$JOB_ID
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./counters_omp

[capa20@eimtarqso pac2]$ cat counters2_omp.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N counters2_omp
#$ -o counters2_omp.out.$JOB_ID
#$ -e counters2_omp.out.$JOB_ID
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./counters2_omp

[capa20@eimtarqso pac2]$ qsub counters_omp.sge
Your job 436817 ("counters_omp") has been submitted

[capa20@eimtarqso pac2]$ qsub counters2_omp.sge
Your job 436818 ("counters2_omp") has been submitted
```

El resultat obtingut l'obtenim del fitxer de sortida:

```
[capa20@eimtarqso pac2]$ cat counters_omp.out.436817
Cache Misses Level1: 1874392653
Cache Misses Level2: 73647462
Cache Misses Level3: 5594552

[capa20@eimtarqso pac2]$ cat counters2_omp.out.436818
Cache Misses Level1: 3075977094
Cache Misses Level2: 271259944
Cache Misses Level3: 20890550
```

La versió paralelitzada del codi obté valors més grans de 'Cache misses',

	Counters_omp	Counters2_omp	Increment [%]
Cache Misses Level1	1874392653	3075977094	39,06 %
Cache Misses Level2	73647462	271259944	72,85 %
Cache Misses Level3	5594552	20890550	73,22 %

Addicionalment a les preguntes de la PAC, per tal de validar l'impacte del intercanvi en l'accés a les dades, he creat una versió del codi counter_omp.c amb intercanviant l'ordre del bucle de ijk a ikj i s'observa una reducció molt significativa en 'Cache Misses'.

```
[capa20@eimtarqso pac2]$ cat *436827*
Cache Misses Level1: 62596789
Cache Misses Level2: 4067684
Cache Misses Level3: 405358
```

	Counters_omp ijk	Counters_omp ikj	Increment [%]
Cache Misses Level1	1874392653	62596789	-2894,39 %
Cache Misses Level2	73647462	4067684	-1710,55 %
Cache Misses Level3	5594552	405358	-1280,15 %

Bibliografía

http://developer.amd.com/wordpress/media/2012/10/Introduction_to_CodeAnalyst.pdf
http://perfsuite.ncsa.illinois.edu/psprocess/metrics.shtml
http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2013/Lect08-counters.pdf
https://www.computer.org/csdl/proceedings/iacc/2017/1560/00/07976754.pdf
http://www.abc-lib.org/MyHTML/Lectures/NU/SCSC/Mat-Mat_1_2017.pdf