COMPUTACIÓ D'ALTES PRESTACIONS – PAC 2

Semestre Setembre 2017

- 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 des models de programació MPI i OpenMP, i del seu entorn d'execució. Per a la realització es posaran en pràctica alguns dels conceptes presentats en aquesta assignatura.

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 MPI i 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.

De tota manera, si voleu fer el vostre desenvolupament i proves en el vostre sistema local haureu de tenir instal·lat algun programari que implementi MPI. Podeu utilitzar el que millor us sembli però us aconsellem OpenMPI o MPICH2, que és una distribució lliure, portable i molt popular. Podeu descarregar-la i trobar més informació sobre la instal·lació / utilització en la següent adreça:

http://www.mcs.anl.gov/research/projects/mpich2/

En qualsevol cas s'espera que realitzeu la vostra investigació i si cal que es debati en el fòrum de l'assignatura.

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:

2.1 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:

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

3.2 Com implementarieu el programa sol·licitat en 3.1 usant una clàusula de reducció?

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. Polítiques de planificació

En aquesta secció de la PAC es demana que implementeu quan versió paral·lela del programa proporcionat ("mm_omp.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:

5.1 Proporcioneu la versió paral·lela del codi (mm_omp.c) utilitzant les diferents polítiques de planificació. Expliqueu les decisions d'implementació que heu pres.

6. Entorn d'execució MPI

La aquesta part consisteix a familiaritzar-se amb l'entorn i ser capaç de compilar i executar programes MPI.

Utilitzarem el següent programa MPI que implementa el típic programa que retorna "hello world" (hello.c):

```
#include "mpi.h"
#include <stdio.h>

int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);

  printf("Hello World!\n");

  MPI_Finalize();
}
```

podeu veure que cal incloure "`mpi.h`" i que és imprescindible començar el vostre programa MPI amb `MPI_Init` i al final acabar-lo amb `MPI_Finalize`. Aquestes crides s'encarreguen de treballar amb el runtime de MPI per a la creació i destrucció de processos MPI.

A continuació es presenten els passos necessaris per compilar i executar un simple "hello world" utilitzant MPI:

- Compilar el programa `hello.c` amb mpicc (més detalls a la documentació).
- Executar el programa MPI hello world mitjançant SGE. Un script d'exemple per a l'execució del programa hello utilitzant 8 processos MPI el podeu trobar a continuació:

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N hello
#$ -o hello.out.$JOB_ID
#$ -e hello.err.$JOB_ID
#$ -pe orte 8

mpirun -np 8 ./hello
```

L'opció –pe orte 8 li indica a SGE que necessiteu 8 nuclis (que s'assignaran en més d'un node al clúster de la UOC). La comanda mpirun s'encarrega de cridar al runtime de MPI per realitzar l'execució del programa MPI. L'opció –np 8 li indica al runtime de MPI que utilitzi 8 processos per a l'execució del programa MPI i al final indiquem el nom del binari del programa. Podem estudiar altres opcions de mpirun.

Preguntes:

6.1 Quin és el resultat de l'execució del programa MPI? Per què?

6.2 Expliqueu què passa si utilitzeu l'opció –np 4 en canvi de –np 8 en el mpirun.

7. Processos MPI

A continuació introduïm un programa MPI que inclou crides bàsiques que trobaren en qualsevol programa MPI. Noteu que la variable "rank" és bàsica en l'execució de programes MPI ja que permet identificar el número de procés dins d'un comunicador (en aquest cas MPI_COMM_WORLD). Si us plau, consulteu els apunts per més detalls.

```
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  int rank, numprocs;
  char hostname[256];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);

  gethostname(hostname,255);

  printf("Hello world! I am process number %d of %d MPI processes on host %s\n", rank,
numprocs, hostname);

  MPI_Finalize();

  return 0;
}
```

7.1 Quin és el resultat de l'execució del programa MPI? Per què?

8. Pas de missatges punt a punt

En aquest apartat us demanem el primer exercici de programació. Us facilitem un altre exemple d'aplicació MPI anomenat "hellompi.c" juntament amb l'anunciat de la PAC. Aquest programa està dissenyat per ser executat només amb 2 processos MPI i bàsicament s'encarrega de fer de pas de missatges entre els dos processos MPI.

8.1 Quina és la mida del missatge que es transfereix entre processos?

Es demana que realitzeu una variant del programa "hellompi.c" que faci que cadascun dels processos MPI faci l'enviament d'un missatge MPI a la resta de processos.

Un exemple de sortida amb 3 processos MPI es mostra a continuació:

```
Proc #1 sending message to Proc #0
Proc #1 sending message to Proc #2
Proc #1 received message from Proc #0
Proc #1 received message from Proc #2
Proc #0 sending message to Proc #1
Proc #0 sending message to Proc #2
Proc #0 received message from Proc #1
Proc #0 received message from Proc #2
Proc #2 sending message to Proc #0
Proc #2 sending message to Proc #1
Proc #2 received message from Proc #0
Proc #2 received message from Proc #1
```

8.2 Proporcioneu el codi del vostre programa.

9. Ordre de l'escriptura de la sortida estàndard

En aquest apartat es demana que implementeu un programa MPI que realitzi pas de missatges entre processos MPI en forma d'anell, és a dir, que cada procés MPI faci l'enviament d'un missatge al següent rank i en rebi un del rank anterior (excepte el primer i l'últim que han de tancar l'anell).

Així doncs, donats N processos MPI, s'espera que el procés MPI amb rank 0 faci l'enviament d'un missatge al procés amb rank 1, el procés amb rank 1 rebi el missatge el procés amb rank 0 i faci l'enviament d'un missatge al procés amb rank 2, i així fins arribar al procés amb rank N-1 que rebrà un missatge del procés N-2 i farà l'enviament d'un missatge al procés amb rank 0.

Podeu utilitzar com a referència la següent línia com a exemple per a la sortida del vostre programa.

```
Proc 2: received message from proc 1 sending to 3
```

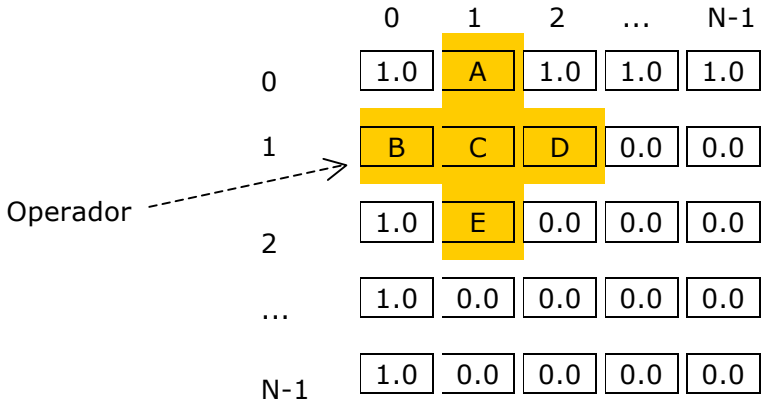
9.1. Proporcioneu el codi del vostre programa.

9.2. En quin orde apareixen els missatges de sortida? Per què?

10. Implementació d'un solver mitjançant MPI

La última part d'aquesta PAC consisteix en implementar la versió paral·lela mitjançant MPI d'un programa seqüencial que implementa un solver. En primer lloc, el programa inicialitza una matriu (de tamany NxN) i després aplica un operador (també conegut com a *stencil*) sobre tota la matriu durant un determinat nombre d'iteracions (ITERS). Per a cada element, l'operador aplica la fórmula:

C '= 0,35 * (C + A + B + D + E), on C ' és el valor de C en la propera iteració.



Es proporciona un programa seqüencial de referència que implementa el solver (solver.c)

Nota: Per implementar aquest problema es recomana definir la matriu de tamany (N+2)x(N+2) per tal de poder aplicar l'operador sense sortir de rang - aquesta tècnica consisteix en envoltar la matriu (amb les també anomenades *halo zones*) i es discutirà en les properes setmanes. També es proporciona un exemple addicional (stencil.c) que utilitza aquesta tècnica en un programa que implementa la propagació d'energia en un espai 2D. Podeu fer algunes proves i visualitzar el fitxer de sortida (heat.svg), per exemple:

```
./stencil 100 10 50
./stencil 100 10 150
./stencil 100 10 500
```

10.1 Implementeu el solver.c mitjançant MPI. Expliqueu les vostres decisions/observacions (per exemple, si heu fet servir crides MPI síncrones o asíncrones/per què).

10.2 Avalueu l'execució del vostre programa MPI en relació a la versió seqüencial. En particular, es demana que estudieu (i propocioneu gràfiques) el speedup respecte el número de nuclis utilitzats, respecte el tamany de la matriu i respecte el número d'iteracions. Nota: podeu parametritzar tamany/iteracions.

Criteris d'avaluació

Es valorarà el correcte ús del model de programació MPI. També es tindrà en compte la correcta programació, estructura i funcionament dels programes i la discussió dels mateixos.

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

Diumenge 12 de Novembre de 2017.