Metodología de la Programación Paralela 2017-2018
Práctica 2 de Esquemas Algorítmicos
Evaluación CONTINUA
jueves 14 de diciembre 2017, puntuación sobre 0.5

Esta práctica se realizará como un ensayo para el examen de programación el 15 de enero. Se realizará de forma individual, y con estructura similar a la que se usará en el examen:

- Durante la primera hora, entre 17:45 y 18:45, se realizará una prueba escrita, en la que los alumnos trabajarán respondiendo las cuestiones que se plantean a continuación, pudiendo utilizar material bibliográfico pero sin trabajar con el ordenador. Se entregarán las soluciones antes de empezar la prueba sobre el ordenador. Cada alumno se puede quedar con esquemas de las soluciones que proponen.

- De 18:45 a 19:45, los alumnos trabajarán resolviendo las cuestiones sobre el ordenador, pudiendo acceder a los recursos de internet que consideren, pero sin copiar. Se tendrán en cuenta sólo soluciones que implementen las soluciones propuestas por el alumno en la primera parte. El sistema se cerrará automáticamente a las 19:45.

- Antes de las 20:00 se entregará un documento escrito en el que se indicarán los cambios realizados sobre la versión presentada en la parte escrita y los números de envío en los que se implementan.

El profesor se pondrá en contacto con los alumnos para acordar la revisión de prácticas, que será entre el 15 y 21 de diciembre. Se puede trabajar en el laboratorio, pero hay que hacer los envíos finales al cluster Heterosolar a través de Mooshak. Si hay problemas con el uso de Heterosolar se puede realizar los envíos a calisto.inf.um.es.

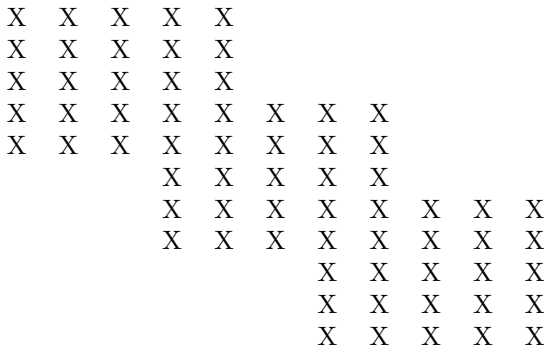
Se trabajará con el problema A del concurso cpp2015, "cuadrados sucesivos de submatrices". El problema A es una versión MPI+OpenMP, por lo que las versiones OpenMP deben ejecutarse con un único proceso MPI y dentro de él lanzar hilos, y las versiones MPI serán con varios procesos MPI y dentro de cada uno un único hilo.

CUESTIÓN 1 (0.1)
Hacer una implementación que explote el paralelismo de hilos en memoria compartida con OpenMP.
CUESTIÓN 2 (0.1)
Hacer una implementación OpenMP que use tareas.
CUESTIÓN 3 (0.1)
Hacer una implementación que explote el paralelismo de paso de mensajes con MPI.
CUESTIÓN 4 (0.1)
Hacer una implementación que explote el paralelismo de paso de mensajes con MPI siguiendo el paradigma *pipeline*.
CUESTIÓN 5 (0.1)
Hay que hacer una implementación MPI+OpenMP.

Se incluyen el enunciado del problema y el código secuencial que se proporciona:

Problema de Successive squares of submatrices - MPI+OpenMP version

We have a square matrix A of size $N \times N$ of double precision numbers, and consider submatrices of size $M \times M$ in the diagonal of A (the submatrix beginning at position (i, i) consists of M rows and columns beginning in row i and column i: $A[i : i + M - 1, i : i + M - 1]$). The elements in the last L rows and columns of a submatrix overlap with those in the initial L rows and columns of the following submatrix. Given N, M and L, the submatrices are those beginning in the first row and column, row and column $M - L$, row and colum $2 * M - 2 * L$..., and only submatrices with all the data in A are considered (the final submatrix will be of size $M \times M$). For example, if $N = 11$, $M = 5$ and $L = 2$, the structure of the submatrices is:



The squares of the matrices are calculated in the order: first matrix beginning in the first row and column is substituted by its square, then the matrix beginning in row and column $M - L$ is substituted by its square, and so on until the square of all the submatrices is calculated. The whole process is performed S times.

A number of problems is solved. For each problem the function to parallelize has:

- Input parameters:
  - int N: the size of matrix A, $N \times N$
  - int M: the size of the submatrices, $M \times M$
  - int L: the number of rows and columns where the submatrices overlap
  - int S: the number of computations
- Input-output parameter:

  double $*A$: the matrix $N \times N$ of double precision numbers. It is updated by dividing each element by 10, so that the numbers do not become very high after the multiplications
- Parallelism parameters:
  - int *node*: identification of the MPI process
  - int *np*: total number of MPI processes

sec.cpp

```
/*
  CPP_CONTEST=2015
  CPP_PROBLEM=A
  CPP_LANG=CPP+OPENMP+MPI
  CPP_PROCESSES_PER_NODE=marte 1
*/

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

void multiplication(double *a,int n,int ld)
{
  double *r=new double[n*n];

  for(int i=0;i<n;i++)
  {
    for(int j=0;j<n;j++)
    {
       double s=0.;
       for(int k=0;k<n;k++)
```

```
      {
        s+=a[i*ld+k]*a[k*ld+j];
      }
      r[i*n+j]=s;
    }
  }

  for(int i=0;i<n;i++)
  {
    for(int j=0;j<n;j++)
    {
       a[i*ld+j]=r[i*n+j];
    }
  }

  delete [] r;
}

void multiplications(int N,int M,int L,double *A)
{
  for(int pos=0;pos<=N-M;pos+=M-L)
  {
    multiplication(&A[pos*N+pos],M,N);
  }
}

void update(int n,double *a,double d)
{
  for(int i=0;i<n*n;i++)
  {
    a[i]/=d;
  }
}

void  sec(int N,int M,int L,int S,double *A,int nodo,int np)
{
  update(N,A,10.);
  for(int step=0;step<S;step++)
  {
     multiplications(N,M,L,A);
  }
}
```

SOLUCIONES

Parte escrita

CUESTIÓN 1

Vamos a analizar dónde se puede incluir paralelismo de bucles con OpenMP, empezando por los bucles más externos para intentar hacer paralelismo del mayor grano posible.

No se puede paralelizar el bucle con los pasos en la función `sec`, ya que los valores de un paso dependen de los de los pasos anteriores.

En cada paso, dentro de la función `multiplications` se hacen cuadrados de submatrices en la diagonal de la matriz A. Como las submatrices se solapan y los cuadrados de las submatrices se hacen empezando en la superior izquierda y bajando hasta la inferior derecha, tampoco se puede paralelizar el bucle de `multiplications`.

La única opción que queda es paralelizar dentro de la función `multiplication`, lo que se hace como una multiplicación de matrices, incluyendo `#pragma omp parallel for private(i)` antes del bucle en `i`. No hace falta indicar que `j`, `k` y `s` son privadas ya que se declaran dentro de la región paralela. Para implementar paralelismo de mayor grado se puede incluir la cláusula `collapse(2)`.

CUESTIÓN 3

Respondemos a la CUESTIÓN 3 antes que a la 2 pues una forma básica de introducir paralelismo MPI es paralelizando cada llamada a `multiplication`, tal como hacemos en la cuestión 1.

En el caso de multiplicar dos matrices A y B para obtener la matriz $C = A * B$, se puede mandar la matriz B a todos los procesos (con un broadcast), y dividir la matriz A por filas, asignando a cada proceso $\frac{M}{p}$ (p es el número de procesos) filas de la submatriz que se esté multiplicando, obteniendo cada proceso $\frac{M}{p}$ filas de la matriz C.

Las diferencias con el caso que nos ocupa son:

- Si A_i es la submatriz de tamaño $M \times M$ que estamos multiplicando por sí misma, con enviar A_i a cada proceso (con broadcast) es suficiente, y cada proceso multiplicará las filas de A_i que le correspondan por toda A_i.
- Cuando se vaya a hacer el broadcast de la matriz A_i hay que tener en cuenta que los datos no están contiguos en memoria, pues la matriz A_i de tamaño $M \times M$ está dentro de la matriz A, de dimensión $N \times N$. Por tanto, antes de enviar la matriz A_i hay que compiarla en una matriz temporal T de dimensión $M \times M$.
- Por la misma razón, cuando el proceso P_0 reciba del resto de los procesos (con `Gatherv` por ejemplo) los trozos de la matriz resultado, los recibirá en una matriz T, y de ahí los copiará a las zonas correspondientes de la matriz A.
- Hay que determinar el número de filas que calcula cada proceso. Como todas las multiplicaciones son del mismo tamaño, ese cálculo se puede hacer una única vez al principio. El número de filas de cada proceso será $\frac{M}{p}$, y se asignará una fila más a los procesos P_i con $i < M\%p$. Por ejemplo, si $M = 10$ y $p = 3$, se asignan tres filas a cada proceso, y a P_0 se le asigna una más pues $0 < 10\%3 = 1$.

CUESTIÓN 5

La combinación MPI+OpenMP es obvia. Se paraleliza con MPI cada multiplicación, como se ha indicado para la cuestión 3, y la multiplicación en cada proceso de las filas de A_i que le corresponden por la matriz A_i se hace paralelizando con OpenMP el bucle más externo de la multiplicación, como en la cuestión 1.

CUESTIÓN 4

Respondemos a la cuestión 4 antes que a la dos pues se van a hacer de forma similar. Los cálculos asociados a cada submatriz A_i se asocian en MPI a un proceso, y en OpenMP a una tarea.

Consideramos que tenemos p procesos, cada proceso es P_i, con $0 \leq i \leq p-1$, y que el proceso P_i opera sobre la submatriz A_i. El problema es que no conocemos el número de submatrices hasta que no tenemos la entrada del problema. Se pueden lanzar procesos de más y en el código determinar el número de procesos que se necesita, y el resto que no haga nada. Otra posibilidad es lanzar un único proceso, y que este arranque en tiempo de ejecución el resto de procesos. Esto se puede hacer en MPI con `MPI_Comm_spawn`, pero esta función (disponible en MPI2) no la hemos estudiado.

Por tanto, consideramos que el número de procesos se pone en marcha de forma estática con `mpirun -np X`, y lanzaremos un número elevado de procesos para internamente descartar los que no tienen que trabajar. Hay que identificar procesos que trabajan de forma distinta. P_0 puede empezar a trabajar calculando el cuadrado de la matriz A_0, y enviando el trozo que solapa A_0 con A_1 al proceso P_1. En el resto de iteraciones tiene que esperar el trozo que solapan del proceso P_1, hacer su cálculo y enviar a P_1. Los procesos intermedios, de P_1 a P_{p-2} trabajan de otra forma, y el P_{p-1} de otra distinta. Un esquema algorítmico puede ser:

En cada P_i, $i = 0, \ldots, p-1$:
  si $i = 0$
    para $i = 1, 2, \ldots, p-1$
      enviar A_i a P_i //antes de enviar hay que copiar A_i a una matriz temporal
  else
    recibir en A, A_i de P_0

//una vez que reciben la matriz hacen los S pasos sobre ella
si $i = 0$
//paso 1
calcular $A = A * A$
enviar submatriz $L \times L$ inferior-derecha a P_1 //antes hay que copiarla a una matriz temporal de tamaño $L \times L$
para $paso = 2, 3, \ldots, S$ //resto de pasos
recibir de P_1 submatriz $L \times L$ y copiarla en parte superior-izquierda de A
calcular $A = A * A$
enviar submatriz $L \times L$ inferior-derecha a P_1
en otro caso si $i = p - 1$ //en último proceso
para $paso = 1, 2, \ldots, S - 1$//primeros pasos
recibir de P_{p-2} submatriz $L \times L$ y copiarla en parte superior-izquierda de A
calcular $A = A * A$
enviar submatriz $L \times L$ superior-izquierda a P_{p-2}
//paso S
recibir de P_{p-2} submatriz $L \times L$ y copiarla en parte superior-izquierda de A
calcular $A = A * A$
en otro caso //procesos intermedios
//paso 1
recibir de P_{i-1} submatriz $L \times L$ y copiarla en parte superior-izquierda de A
calcular $A = A * A$
enviar submatriz $L \times L$ superior-izquierda a P_{i-1}
enviar submatriz $L \times L$ inferior-derecha a P_{i+1}
para $paso = 2, \ldots, S - 1$ //resto de pasos hasta el penúltimo
recibir de P_{i-1} submatriz $L \times L$ y copiarla en parte superior-izquierda de A
recibir de P_{i+1} submatriz $L \times L$ y copiarla en parte inferior-derecha de A
calcular $A = A * A$
enviar submatriz $L \times L$ superior-izquierda a P_{i-1}
enviar submatriz $L \times L$ inferior-derecha a P_{i+1}
//paso S
recibir de P_{i-1} submatriz $L \times L$ y copiarla en parte superior-izquierda de A
recibir de P_{i+1} submatriz $L \times L$ y copiarla en parte inferior-derecha de A
calcular $A = A * A$
enviar submatriz $L \times L$ inferior-derecha a P_{i+1}
//al acabar todos los pasos se mandan las submatrices a P_0, que las deja en su sitio dentro de A
si $i = 0$
para $i = 1, 2, \ldots, p - 1$
recibir matriz de P_i y copiarla en posiciones de A_i
else
enviar A a P_0

CUESTIÓN 2

Podemos lanzar una tarea por cada submatriz y sincronizarlas como en la cuestión 4 se sincronizan los procesos, pero en vez de con mensajes con un array de p enteros al que se accede en exclusión mutua y cada tarea apunta el número de paso que ha hecho. Por tanto, no se hacen envíos a la tarea anterior y siguiente, sino que se apunta en el array de *pasos* y las recepciones se sustituyen por lecturas en el array hasta que está el número de paso que necesitamos. Así, el esquema anterior se modifica:

pasos $\leftarrow 0$ //se inicializa el array de pasos (dimensión igual al número de submatrices, que coincide con el de tareas) a 0
# pragma omp parallel //se ponen en marcha los hilos
{
#pragma omp single nowait //un hilo pone en marcha las tareas
{
para $i = 0, 1, \ldots, numerotareas$
#pragma omp task //inicia una tarea para cada submatriz
multiplicar A_i

}

La función multiplicar A_i:

//cada tarea i hace los S pasos sobre la matriz A_i

si $i = 0$

//paso 1

calcular $A_i = A_i * A_i$

#pragma omp critical //acceso en exclusión mutua al array de pasos

$pasos[0] = 1$

para $paso = 2, 3, \ldots, S$ //resto de pasos

acceso en exclusión mutua a $pasos[1]$ hasta que valga $paso - 1$ //para que se pueda hacer un paso de A_0 tiene que estar hecho el paso anterior de A_1

calcular $A_i = A_i * A_i$

en exclusión mutua poner $pasos[0] = paso$

en otro caso si $i = p - 1$

para $paso = 1, 2, \ldots, S$

acceso en exclusión mutua a $pasos[p - 2]$ hasta que valga $paso$ //puede hacer ese paso cuando la tarea de la matriz aanterior lo ha hecho ya

calcular $A_i = A_i * A_i$

en exclusión mutua poner $pasos[p - 1] = paso$

en otro caso //tareas intermedias

//paso 1

acceso en exclusión mutua a $pasos[i - 1]$ hasta que valga 1

calcular $A_i = A_i * A_i$

en exclusión mutua poner $pasos[i] = 1$

para $paso = 2, \ldots, S$ //resto de pasos hasta el último

acceso en exclusión mutua a $pasos[i - 1]$ hasta que valga $paso$ //el anterior tiene que haber hecho este paso

acceso en exclusión mutua a $pasos[i + 1]$ hasta que valga $paso - 1$ //y el siguiente el paso anterior

calcular $A_i = A_i * A_i$

en exclusión mutua poner $pasos[i] = paso$

}

Parte con el ordenador

Se usa la cuenta `Domingo` en HETEROSOLAR. Se acompañan los códigos.

CUESTIÓN 1

Hacemos un primer envío secuencial para poder comparar con él las versiones paralelas que hagamos. Es el envío 532, con tiempo 7135. Se hace un envío de la versión OpenMP sin indicar el número de hilos, con lo que por defecto usa 6, ya que marte tiene 6 cores. Es el número 534, con tiempo 1616.

CUESTIÓN 3

El envío 541 es con 6 procesos en marte y 6 en mercurio, con un tiempo 3026, peor que el de OpenMP que sólo usa marte. Se podría hacer experimentos variando el número de procesos para ver con el que se obtiene menor tiempo, pues la dimensión reducida de las submatrices cuya multiplicación se está paralelizando y el alto coste de las comunicaciones puede hacer que al aumentar el número de procesos empeore el tiempo.

CUESTIÓN 5

Se paraleliza con MPI como en la cuestión 3, y dentro de la multiplicación de matrices se divide el trabajo entre los hilos con `parallel for`, como en la cuestión 1, pero en este caso el índice del bucle en cada proceso va desde la posición de inicio de sus filas en la submatriz hasta su posición de inicio más el número de filas que tiene que calcular. El envío 542 corresponde a un proceso en marte y otro en mercurio, y cada proceso usa 6 hilos, con lo que se utilizan los 12 cores. El tiempo es 2387, que es mejor que el del envío de la cuestión 3, al tener ahora menos comunicaciones por usar dos procesos en vez de 12.

CUESTIÓN 4

El envío 561, con tiempo 3640, corresponde a 3 procesos en marte y 3 en mercurio, y en cada proceso se arrancan 2 hilos, de forma que se tiene 6 hilos en cada nodo, lo que coincide con el número de cores. El envío 560, con tiempo 2048, corresponde a 3 procesos en marte y 3 en mercurio y 6 hilos en cada proceso. De esta

forma se lanzan más hilos que cores, pero al estar en cada pasada algunos procesos inactivos se hace un mejor uso de los cores en el sistema.

CUESTIÓN 2

El envío 563 se hace con 6 hilos para trabajar con las tareas que se lancen. Para explotar más los cores en el sistema, se usa paralelismo anidado y se lanzan 6 hilos en el segundo nivel de paralelismo para realizar las multiplicaciones matriciales. El tiempo es 1946. Se podrían realizar experimentos para ver cuál es la mejor combinación del número de hilos en los dos niveles.