

Metodología de la Programación Paralela 2017-2018

Prueba práctica, 1 junio 2018

Estructura de la prueba:

- Entre 10:00 y 11:30, se realizará una prueba escrita 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 11:30 a 13:00, 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 13:00.
- Antes de las 13:15 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. Para cada cuestión hay que indicar un único número de envío que será el que se evalúe. Para las cuestiones que no se tenga envíos aceptados, el alumno puede indicar el envío que considere que tiene más cercano a una solución correcta.

Se puede trabajar en el laboratorio, pero hay que hacer los envíos finales a calisto.inf.um.es.

Se trabajará con el concurso 2018Junio.

El profesor dejará en un anuncio en el aula virtual los enunciados de los problemas junto con los esquemas y los ficheros con las soluciones secuenciales.

Se muestra la puntuación máxima de cada cuestión. Para cada una de ellas, la mitad de la puntuación corresponde a la parte escrita y la otra a la parte sobre ordenador.

La puntuación máxima es 10. Esta puntuación se transformará a una puntuación sobre 3 o 4 según se siga la evaluación continua o la no continua.

El número de hilos o procesos de los problemas paralelos debe ser general y no depender del tamaño del problema, salvo en los casos en que en el enunciado se indique un número fijo.

PROBLEMA 1: Ordenación de filas y columnas y suma.

Se trabaja sobre una matriz de números reales, A de dimensión $N \times N$. Se genera una matriz F , $N \times N$, con cada fila la fila correspondiente de A con los datos ordenados de menor a mayor, y otra matriz C , $N \times N$, con cada columna la columna correspondiente de A con los datos ordenados de menor a mayor. Finalmente se actualiza $A = F + C$.

Se resuelven varios problemas. Para cada problema la función a paralelizar tiene parámetros:

- Input parameter:
int N : número de filas y columnas
- Input/Output parameter:
double $*A$: matriz de dimensión $N \times N$

La entrada tiene en la primera línea el número de problemas. Para cada problema hay una línea con cinco valores: N , la semilla para la generación aleatoria de datos, los valores mínimo y máximo para la generación, y un valor que indica cada cuantos elementos se escribe un elemento a la salida.

CUESTIÓN 1-a (1 punto)

Hacer una implementación paralela con OpenMP en la que se paralelicen las tres funciones (ordenación de filas, de columnas y suma) por separado (Problema A en mooshak).

CUESTIÓN 1-b (1 punto)

Hacer una implementación paralela con OpenMP en la que la ordenación de filas y la de columnas se hagan simultáneamente, un hilo haciendo las ordenaciones de filas y otro las de columnas (Problema B).

CUESTIÓN 1-c (1.5 punto)

Hacer una implementación OpenMP con paralelismo de dos niveles. En un primer nivel trabajarán dos hilos, uno con la ordenación de filas y otro con la de columnas (esquema de la CUESTIÓN 1-b), y cada uno de estos hilos pondrá en marcha otros hilos para realizar sus ordenaciones (Problema C).

CUESTIÓN 1-d (1 punto)

Hacer una implementación MPI con dos procesos, con un proceso para realizar las ordenaciones de filas y otro las de columnas (Problema D).

CUESTIÓN 1-e (1.5 punto)

Hacer una implementación MPI+OpenMP combinando cualesquiera de los esquemas MPI y OpenMP anteriores o con una implementación distinta (Problema E).

PROBLEMA 2: Camino máximo de primera fila a última.

Sobre una matriz de números enteros positivos, A de dimensión $N \times N$, se quiere obtener el camino de longitud máxima desde la primera a la última fila de la matriz. Los posibles caminos empiezan en cualquier elemento de la primera fila, y desde una casilla se avanza a una vecina en la fila siguiente, en la misma columna o en la columna anterior o siguiente. La longitud de un camino viene dada por la suma de los valores en las casillas por las que pasa. El problema se puede resolver de forma más eficiente que con el esquema de backtracking que se proporciona, pero hay que realizar las paralelizaciones que se piden utilizando el código secuencial que se proporciona.

Se resuelven varios problemas. Para cada problema la función a paralelizar devuelve la longitud de camino máximo, y tiene parámetros:

- Input parameter:

int N : número de filas y columnas

double $*A$: matriz de dimensión $N \times N$

La entrada tiene en la primera línea el número de problemas. Para cada problema hay una línea con tres valores: N , la semilla para la generación aleatoria, el valor máximo para la generación (el mínimo es uno).

CUESTIÓN 2-a (1.5 puntos)

Hacer una implementación paralela con OpenMP (Problema F en mooshak).

CUESTIÓN 2-b (1.5 puntos)

Hacer una implementación paralela con MPI (Problema G).

CUESTIÓN 2-c (1 punto)

Hacer una implementación MPI+OpenMP (Problema H).

Se incluyen los códigos secuenciales de los dos problemas:

PROBLEMA 1:

```
/*
  CPP_CONTEST=2018Junio
  CPP_PROBLEM=A
  CPP_LANG=CPP+OPENMP
  CPP_PROCESSES_PER_NODE=calisto 1
*/

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

extern void escribir(double *,int,int);

void copiar(int n,double *vd,int ldvd,double *vo,int ldvo)
//se copia el vector vo en el vd
//como los datos pueden no estar contiguos se accede a ellos con los ld
{
  int i;
  for(i=0;i<n;i++)
    vd[i*ldvd]=vo[i*ldvo];
}

void ordenar(int n,double *m)
{
  int i,j;
```

```

double t;
for(i=0;i<n;i++)
    for(j=i+1;j<n;j++)
    {
        if(m[i]>m[j])
        {
            t=m[i];
            m[i]=m[j];
            m[j]=t;
        }
    }
}

void ordenarfilas(int n,double *A,double *F)
{
    int i;
    for(i=0;i<n;i++)
    {
        copiar(n,&F[i*n],1,&A[i*n],1); //se copia la fila de A en la correspondiente de F
        ordenar(n,&F[i*n]); //y se ordena la fila en F
    }
}

void trasponer(int n,double *m)
{
    int i,j;
    double t;
    for(i=0;i<n;i++)
        for(j=i+1;j<n;j++)
        {
            t=m[i*n+j];
            m[i*n+j]=m[j*n+i];
            m[j*n+i]=t;
        }
}

void ordenarcolumnas(int n,double *A,double *C)
{
    int i;
    for(i=0;i<n;i++)
    {
        copiar(n,&C[i*n],1,&A[i],n); //se copia la columna de A en la fila correspondiente de C
        ordenar(n,&C[i*n]); //y se ordena la fila en C
    }
    trasponer(n,C); //se pasan las filas de C, que tienen las columnas ordenadas de A, a las columnas
}

void sumar(int n,double *vd,double *vo1,double *vo2)
{
    int i;
    for(i=0;i<n*n;i++)
        vd[i]=vo1[i]+vo2[i];
}

void sec(int n,double *A)
{
    double *F=new double[n*n];
}

```

```

double *C=new double[n*n];

ordenarfilas(n,A,F);
#ifdef DEBUG
    escribir(F,n,n);
#endif
ordenarcolumnas(n,A,C);
#ifdef DEBUG
    escribir(C,n,n);
#endif
sumar(n,A,F,C);
#ifdef DEBUG
    escribir(A,n,n);
#endif
delete[] F;
delete[] C;
}

```

PROBLEMA 2:

```

/*
  CPP_CONTEST=2018Junio
  CPP_PROBLEM=F
  CPP_LANG=CPP+OPENMP
  CPP_PROCESSES_PER_NODE=calisto 1
*/

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

extern void escribir(int *,int,int);

void copiar(int n,int *vd,int *vo)
//se copia el vector vo en el vd
{
    int i;
    for(i=0;i<n;i++)
        vd[i]=vo[i];
}

int back(int n,int *A,int fila,int columna)
{
    int valor;
    if(fila==n-1)
        return A[fila*n+columna];
    else
    {
        int maximo=0;
        if(columna!=0)
        {
            valor=A[fila*n+columna]+back(n,A,fila+1,columna-1);
            if(valor>maximo)
                maximo=valor;
        }
        if(columna!=n-1)
        {

```

```

        valor=A[filan+columna]+back(n,A,filan+1,columna+1);
        if(valor>maximo)
            maximo=valor;
    }
    valor=A[filan+columna]+back(n,A,filan+1,columna);
    if(valor>maximo)
        maximo=valor;
    return maximo;
}
}

int  sec(int n,int *A)
{
    int VOA=0; //valor óptimo actual

    for(int i=0;i<n;i++) //se calculan los óptimos empezando en cada una de las columnas de la primera
    {
        int valor=0; //para almacenar el valor calculado
        valor=back(n,A,0,i); //se pasa la fila y columna por lo que se sigue haciendo el backtracking
        if(valor>VOA)
        {
            VOA=valor;
        }
    }
#ifdef DEBUG
    printf("columna %d , valor %d , optimo %d\n",i,valor,VOA);
#endif
}
return VOA;
}

```

SOLUCIONES parte escrita

CUESTIÓN 1-a

La paralelización en los tres casos se consigue simplemente paralelizando los bucles con `#pragma omp parallel for private(i)`

CUESTIÓN 1-b

Se utilizan secciones para lanzar una sección por cada uno de los dos grupos de ordenaciones. La suma se sigue haciendo como antes:

```

#pragma omp parallel sections
{
#pragma omp section
    ordenarfilas(n,A,F);
#pragma omp section
    ordenarcolumnas(n,A,C);
}
sumar(n,A,F,C);

```

CUESTIÓN 1-c

Es una combinación obvia de los dos casos anteriores: se lanzan dos hilos con secciones, como en la cuestión anterior, y se paralelizan los bucles como en la cuestión 1. Sólo hay que asegurarse de que está habilitado el paralelismo anidado, lo que se puede hacer ejecutando al principio `omp_set_nested(1)`. Para establecer dos hilos para el primer nivel y otros dos para el segundo, se utiliza `omp_set_num_threads` antes de la `sections` y de los bucles, con el número de hilos que se quiere usar en cada caso.

CUESTIÓN 1-d

Se establecen dos procesos con `calisto 2`.

El proceso 0 envía al 1 la matriz (previamente tiene que mandar su tamaño para que el 1 reserve espacio para los datos). La comunicación puede ser punto a punto, pues sólo tenemos dos procesos, pero también se puede usar broadcast.

El proceso 0 hace las ordenaciones por filas y el 1 las de columnas.

El 1 envía al 0 sus resultados, y el 0 los suma.

CUESTIÓN 1-e

La combinación más inmediata es la de lanzar dos procesos MPI como en la cuestión anterior y paralelizar los bucles como en la cuestión 1-a.

CUESTIÓN 2-a

Se paraleliza el bucle principal con `parallel for`. De esta forma se divide el trabajo desde columnas distintas de la primera fila entre los hilos. Hay que asegurarse el acceso en exclusión mutua a la evaluación y actualización del valor óptimo actual, lo que se hace con la cláusula `critical`.

El backtracking no está optimizado, pues no se eliminan nodos del árbol de búsqueda que se sepa que no pueden mejorar la solución actual. Si se hiciera así podría ser mejor una distribución cíclica del trabajo, para propiciar que se distribuyan equitativamente entre los hilos las zonas más podadas.

CUESTIÓN 2-b

El proceso cero envía los datos a los demás procesos con broadcast (antes se ha enviado el tamaño de la matriz para que se pueda reservar espacio para los datos).

Cada proceso hace el backtracking desde un grupo de columnas, para lo que se cambia el bucle `for` a:

```
for(int i=nodo;i<n;i+=np)
```

Los procesos de 1 al `np-1` envían su valor óptimo al proceso 0, que obtiene el óptimo global.

Si se hubiera incluido poda, tal como se indica al final de la cuestión anterior, el distribuir el trabajo entre los procesos tal como se hace puede propiciar que se realicen menos podas, pues no se utiliza un VOA global. De esta forma, puede que empeoraran las prestaciones. La alternativa sería realizar un envío del valor óptimo local cada vez que se actualice, pero eso conllevaría más comunicaciones y sería necesario programar esas comunicaciones, posiblemente con un proceso para gestionarlas.

CUESTIÓN 2-c

Se lanzan los procesos MPI como en la cuestión anterior, y el trabajo de cada proceso se paraleliza como en la cuestión 2-a, ahora con

```
#pragma omp parallel for
for(int i=nodo;i<n;i+=np)
```

SOLUCIONES parte sobre ordenador

Todas las pruebas se han hecho en calisto.inf.um.es.

CUESTIÓN 1-a

La versión secuencia da un tiempo de 30056, y la OpenMP con 4 hilos (calisto tiene 4 cores) 7677. El speed-up es 3.92, por lo que está cerca del óptimo.

CUESTIÓN 1-b

El tiempo paralelo es 14517, con speed-up de 1.94, lo que es aceptable ya que utilizamos sólo dos hilos en la parte más costosa de la computación, las dos ordenaciones, que tienen coste $n^2 \log n$, mientras la suma tiene coste n^2 .

CUESTIÓN 1-c

Con dos hilos en el primer nivel de las ordenaciones y dos en los bucles de las ordenaciones, y con cuatro hilos para la suma, el tiempo es 7760, con 3.66 de speed-up.

CUESTIÓN 1-d

El tiempo es 14920, un poco peor que en el caso de dos hilos (cuestión 2), debido al mayor coste de las comunicaciones y a que la suma no se paraleliza.

CUESTIÓN 1-e

Se lanzan los dos procesos MPI, y se paralelizan los bucles sin indicar el número de hilos, con lo que usa el valor por defecto que es 4, el número de cores en el sistema. En total se utilizan 8 hilos en las zonas de

ordenación, 4 trabajando en ordenaciones de filas y 4 de columnas. El tiempo es 7699, con lo que la sobrecarga por usar más hilos que cores no es grande.

CUESTIÓN 2-a

La versión secuencial tarda 10262 y la OpenMP con 4 hilos 3277. El speed-up es 3.14, que es aceptable.

CUESTIÓN 2-b

Con 4 procesos tarda 4184. Las prestaciones son algo peores que con OpenMP debido a las comunicaciones.

CUESTIÓN 2-c

El tiempo con 2 procesos MPI y sin establecer el número de hilos OpenMP (se utiliza el valor por defecto, 4) es de 2991. Es algo mejor que el de OpenMP usando 4 hilos, lo que puede deberse a una mejor distribución del trabajo al usar un total de 8 hilos, sobrepasando el número de cores del sistema.