EXAMEN DE CONCURRENCIA Y PARALELISMO

Bloque II: Paralelismo

APELLIDOS: _____ **NOMBRE:** _____

Convocatoria ordinaria

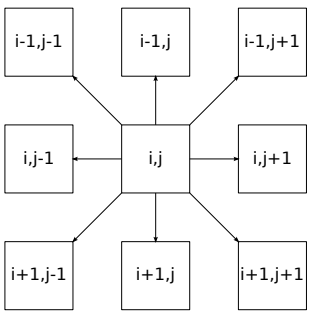
30 de mayo de 2016

AVISO: *No mezcléis en el mismo folio preguntas del bloque de concurrencia y del bloque de paralelismo. Las respuestas a cada bloque se entregarán por separado.*

[2,5p] 1. **Ejercicio de paralelismo** Se dispone de un motor de resolución de ecuaciones en diferencias finitas en dos dimensiones. Dicho motor calcula, de forma iterativa, expresiones de la forma:

$$a_{i,j}^{k+1} = f(a_{i-1,j-1}; a_{i-1,j}; a_{i-1,j+1}; a_{i,j-1}; a_{i,j}; a_{i,j+1}; a_{i+1,j-1}; a_{i+1,j}; a_{i+1,j+1})$$

donde f es una función lineal configurable por el usuario, cuyo resultado es la suma de todas las entradas multiplicadas cada una de ellas por una constante (cuyo valor puede ser cero). La entrada al sistema consiste en una matriz de tamaño $N \times N$ (nótese que en la expresión anterior $a_{i,j}$ representa la posición (i, j) de esta matriz, como se ilustra en la Figura 1). El motor se ejecuta iterativamente sobre ella, aplicando la función f sobre cada punto, hasta que se alcanza la convergencia del sistema. El criterio de convergencia especifica que el cálculo termina cuando la suma de todas las diferencias cuadráticas de cada punto (i, j) de la matriz calculada en una iteración con respecto a la iteración anterior no rebasa un determinado umbral.




```
while( diff > umbral ) {
    diff = 0
    for( i = 1; i < N-1; ++i ) {
        for( j = 1; j < N-1; ++j ) {
            new_a[i][j] = f( a[i-1][j-1], ... );
            diff += pow( new_a[i][j] - a[i][j], 2 );
        }
    }
}
```

Se desea paralelizar dicho motor, empleando para ello una estrategia de descomposición de dominio. Responde de forma razonada a las siguientes preguntas:

a) ¿Cómo asignarías las tareas a los procesos disponibles? ¿Depende esta asignación de la función f a aplicar en cada problema concreto?

b) Un problema concreto presenta como entrada una matriz de tamaño 1024×1024. El valor de cada posición (i, j) en la iteración $(k+1)$ depende únicamente de su vecino inmediatamente superior $(i-1, j)$ y de su vecino inmediatamente inferior $(i+1, j)$ en la iteración (k). ¿Cómo realizarías la asignación de tareas a 9 procesos? Si el tiempo de calcular una posición en cada iteración es de un ciclo y el tiempo de enviar un mensaje es de k ciclos, donde k corresponde al número de elementos del mensaje, ¿cuál es el speedup esperable? ¿Y la eficiencia paralela?

c) En el supuesto del apartado anterior, ¿a qué proceso se asigna la posición con índice global $(114, 464)$? ¿Cuál es su índice local?

d) Otro problema presenta como entrada una matriz de tamaño 510×510. El valor de cada posición (i, j) en la iteración $(k + 1)$ depende de sus vecinos superior $(i - 1, j)$, inferior $(i + 1, j)$, izquierdo $(i, j - 1)$ y derecho $(i, j + 1)$. ¿Cómo asignarías en este caso las tareas a los 9 procesos? ¿A qué proceso se asigna la posición con índice global $(114, 464)$? ¿Cuál es su índice local?

e) En el supuesto del apartado anterior, considera que el criterio de convergencia cambia, de modo que la matriz se divide en regiones de tamaño 30×30, y cada una de ellas pasa a ser invariable desde el momento en que la suma de todas las diferencias cuadráticas de los puntos en ella contenidos no rebasa un determinado umbral entre dos iteraciones consecutivas. ¿Cómo cambiarías la asignación elegida? ¿A qué proceso se asignaría en este caso la posición con índice global $(114, 464)$? ¿Cuál es su índice local?

SOLUCIÓN

a) El coste computacional de la función f es aproximadamente el mismo independientemente de la celda de matriz a calcular, por lo que el problema es adecuado para realizar una distribución estática en bloques de las posiciones a calcular de la matriz. En todo caso, la función f determina las dependencias existentes entre las casillas vecinas, por lo que el tipo y tamaño de los bloques dependerá de ésta. En problemas que tengan dependencias en ambas dimensiones de la matriz lo más adecuado será hacer un reparto en bloques bidimensionales, mientras que en problemas con dependencias en una única dimensión lo ideal será utilizar bloques unidimensionales que eliminen la necesidad de comunicaciones para resolver las dependencias entre tareas. En cualquier caso, el tamaño de bloque debería ser máximo para minimizar las comunicaciones (efecto volumen-superficie).

b) Dado que únicamente existen dependencias verticales entre las posiciones de la matriz, el reparto ideal será en bloques de columnas, ya que eliminará la necesidad de realizar comunicaciones para resolver dependencias entre casillas.

$$m_{by} = \left\lceil \frac{1024}{9} \right\rceil = 114$$

$$\hat{m}_{by} = 1024 - 8 \times 114 = 112$$

En cuanto al *speedup*:

$$sp(9) = \frac{T_{sec}}{T_{par}(9)} = \frac{T_{sec}}{T_{comp}(9) + T_{comm}(9)}$$

El tiempo secuencial es igual a la suma del tiempo de cómputo de todas las casilla, es decir, $T_{sec} = 1024^2$. En cuanto al tiempo paralelo, será el máximo de los tiempos de cómputo de los 9 procesos, y se divide en tiempo de cómputo y tiempo de comunicaciones. En cuanto al cómputo, todos los procesos deben calcular un bloque de tamaño 114×1024, excepto P_8 que tiene un bloque de tamaño 112×1024. En lo tocante a las comunicaciones, se han eliminado todas las necesarias para resolver dependencias entre las casillas de la matriz. Es necesario realizar una reducción de los valores locales de la variable `diff` que controla la convergencia del cálculo, pero asumimos que este tiempo ($O(log_2(p))$ es despreciable en el contexto de este cálculo. Por tanto:

$$sp(9) = \frac{T_{sec}}{T_{par}(9)} = \frac{1024^2}{114 \times 1024} = 8{,}98$$

$$eff(9) = \frac{sp(9)}{9} = 0{,}99$$

c)

$$p = \left\lfloor \frac{464}{114} \right\rfloor = 4$$

$$i_l = (114, 464\,\%114) = (114, 8)$$

d) En este caso, al existir dependencias en ambas direcciones, la mejor solución consistirá en realizar un reparto bidimensional de la matriz sobre una malla bidimensional de 3×3 procesos:

$$m_b = \left(\left\lceil \frac{510}{3} \right\rceil, \left\lceil \frac{510}{3} \right\rceil \right) = (170, 170)$$

En cuanto al elemento $i = (114, 464)$:

$$p = \left(\left\lfloor \frac{114}{170} \right\rfloor, \left\lfloor \frac{464}{170} \right\rfloor \right) = (0, 2)$$

$$i_l = (114\,\%170, 464\,\%170) = (114, 124)$$

e) Al modificarse el criterio de convergencia es posible que ciertas partes de la matriz dejen de formar parte del cálculo progresivamente. Si esto ocurriese, y dependendiendo de su distribución espacial, podría ocurrir que un reparto por bloques bidimensionales desbalancease mucho la carga (si los bloques que han convergido en un momento determinado se reparten de forma desigual entre los procesadores). Para evitarlo, es necesario que todos los procesos participen en el cálculo de cada bloque 30×30 de la matriz. Para ello utilizamos un reparto cíclico por bloques con tamaño de bloque $m_b = (10, 10)$.
En cuanto al elemento $i = (114, 464)$:

$$b = \left(\left\lfloor \frac{114}{10} \right\rfloor, \left\lfloor \frac{464}{10} \right\rfloor \right) = (11, 46)$$

$$p = (11\,\%3, 46\,\%3) = (2, 1)$$

$$i_l = \left(\left\lfloor \frac{11}{3} \right\rfloor \times m_{bx} + 114\,\%m_{bx}, \left\lfloor \frac{46}{3} \right\rfloor \times m_{by} + 464\,\%m_{by} \right) = (34, 154)$$

[2,5p] 2. **Diseño de algoritmos paralelos.** Un computador de memoria distribuida consta de 17 procesadores P_i, $0 \leq i \leq 16$. El procesador 0 no tiene hardware para operaciones en punto flotante, pero a cambio es el único que puede hacer E/S. Todos los procesadores pueden comunicarse por una red que sólo permite un mensaje en cada momento, es decir, no pueden solaparse dos comunicaciones. El sistema tiene una versión limitada de MPI que no soporta ni comunicaciones asíncronas ni operaciones colectivas, sólo comunicaciones bloqueantes punto a punto.

Se desea calcular $r = A \times B \times v$ donde A y B son matrices de $N \times N$ elementos, y v y r vectores de N elementos, siendo N del orden de millones y divisible por 16. Todos los datos son de punto flotante de doble precisión y residen inicialmente en P_0, deseándose que al final del algoritmo r también resida en P_0. Sabiendo que tanto una operación en punto flotante como la transmisión de un elemento por la red requiere un ciclo:

a) Calcula razonadamente cuánto se tardaría en calcular $r = (A \times B) \times v$ y $r = A \times (B \times v)$ usando un solo procesador para los cálculos. Recuerda que P_0 tiene los datos iniciales y debe guardar r al finalizar.

b) Teniendo en cuenta las restricciones del MPI de este sistema, escribe el pseudocódigo MPI de una función `myAllGather(const double *input, double *output, int elemsPerProc)` diseñada para ser invocada simultáneamente por los procesadores P_i, $1 \leq i \leq 16$, (P_0 nunca participa en ella) que haga que (1) cada procesador P_i le envíe los `elemsPerProc` elementos que están almacenados a partir de `input` a cada uno de los otros 15 procesadores, y (2) cada procesador guarde el resultado en el buffer `output`, de forma que `output[`$(i-1)*$`elemsPerProc]` a `output[`$i*$`elemsPerProc`-1] contenga los datos aportados por P_i.

c) Diseña una paralelización para el cálculo de r en el sistema descrito. Detalla la descomposición en tareas y la asignación a procesos.

d) Escribe un pseudocódigo que implemente tu propuesta. Puedes usar tanto las funciones MPI soportadas en el sistema como `myAllGather` si te es útil.

e) Calcula razonadamente la aceleración de tu solución paralela respecto a la mejor solución secuencial apoyando tu cálculo en una representación gráfica. Dado que N es muy grande, en los cálculos de los tiempos quédate sólo con el monomio de mayor grado, es decir, el término donde N esté elevado al mayor exponente posible, pues los restantes términos tendrán un peso despreciable. Además puedes suponer que los mensajes se intercambiarán en el orden que dé lugar al menor tiempo de ejecución.

SOLUCIÓN

a) Para enviar A, B y v a un procesador y recibir r siempre requeriremos $2N^2 + N$ y N ciclos respectivamente, que suman $2N^2 + 2N$ ciclos.

El cálculo $(A \times B) \times v$ requiere un producto de matrices ($2N^3$ operaciones) y un producto matriz-vector ($2N^2$) operaciones, totalizando $2N^3 + 2N^2 = 2N^2(N+1)$ operaciones. Si les sumamos los debidos a comunicaciones, totalizan $2N^2(N+2) + 2N$ ciclos.

El cálculo $A \times (B \times v)$ requiere dos productos matriz-vector en secuencia, es decir, $2 \times 2N^2 = 4N^2$ operaciones. Si les sumamos los debidos a comunicaciones, totalizan $6N^2 + 2N$ ciclos.

Por tanto la segunda versión es la más rápida.

b) Pseudocódigo para `myAllGather` :

```
void myAllGather(const double *input, double *output, int elemsPerProc) {
  int i, rank;
  MPI_Status status;

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  for(i = 1; i < rank; i++) {
    MPI_Recv(output + (i - 1) * elemsPerProc, elemsPerProc, MPI_DOUBLE,
             i, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    MPI_Send(input, elemsPerProc, MPI_DOUBLE,
             i, 101, MPI_COMM_WORLD, &status);
  }
  for(i = rank+1; i <= 16; i++) {
    MPI_Send(input, elemsPerProc, MPI_DOUBLE,
             i, 101, MPI_COMM_WORLD, &status);
    MPI_Recv(output + (i - 1)* elemsPerProc, elemsPerProc, MPI_DOUBLE,
             i, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
  }
  memcpy(output + (rank - 1) * elemsPerProc, input, elemsPerProc * sizeof(double));
}
```

c) Hemos visto en la implementación secuencial que el algoritmo basado en el orden $A \times (B \times v)$ es mucho más rápido que el de $(A \times B) \times v$. Además para implementar en paralelo productos matriz vector, basta distribuir las matrices por bloques de filas y replicar los vectores, mientras los productos de matrices paralelos requieren replicar o comunicar muchos más datos. Por tanto, paralelizaría $A \times (B \times v)$, primero calculando $w = B \times v$ y luego $r = A \times w$, siguiendo este esquema, que me permite solapar las computaciones con casi todas las comunicaciones:

1) P_0 repartiría B por filas, dando un bloque de $N/16$ a cada procesador P_i, $1 \leq i \leq 16$, junto con una copia de v.
2) Cada procesador multiplicaría su bloque de B por v, generando $N/16$ elementos de w.
3) Cada procesador recibiría un bloque de $N/16$ filas de A, le enviaría su bloque de w a los demás, y recibiría los bloques de los demás para tener una copia entera de w.
4) Cada procesador multiplicaría su bloque de A por w, generando $N/16$ elementos de r.
5) Cada procesador computacional enviaría su bloque de r a P_0.

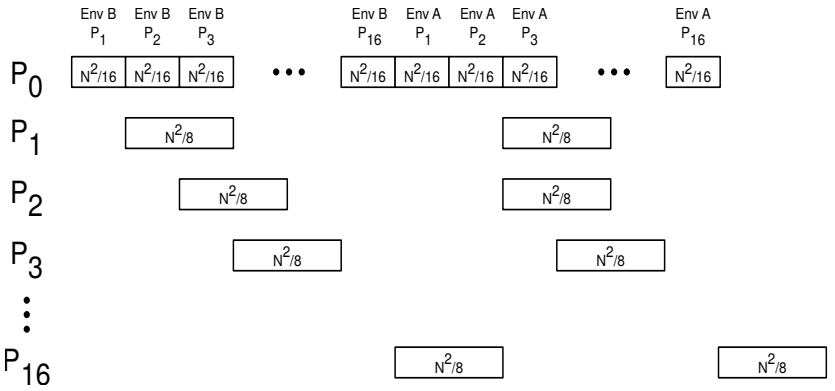
d) Pseudocódigo suponiendo que los nombres de las variables son punteros a buffers de suficiente tamaño ya reservados:

```
if(rank = 0) {
  for(int i=1; i <= 16; i++) {
    MPI_Send(B+((N/16)*N*(i-1)), (N/16)*N, MPI_DOUBLE, i);
    MPI_Send(v, N, MPI_DOUBLE, i);
  }
  for(int i=1; i <= 16; i++) {
    MPI_Send(A+((N/16)*N*(i-1)), (N/16)*N, MPI_DOUBLE, i);
  }
  for(int i=1; i <= 16; i++) {
    MPI_Recv(r+((N/16)*(i-1)), N/16, MPI_DOUBLE, i);
  }
} else {
  MPI_Recv(B, (N/16)*N, MPI_DOUBLE, 0);
  MPI_Recv(v, N, MPI_DOUBLE, 0);
  for(int i=0; i < N/16; i++) {
    wtmp[i] = 0.;
    for(int j=0; j < N; j++)
      wtmp[i] = wtmp[i] + B[i][j] * v[j];
  }
  MPI_Recv(A, (N/16)*N, MPI_DOUBLE, 0);
  myAllGather(wtmp, w, N/16); /* Podría reusar v */
  for(int i=0; i < N/16; i++) {
    r[i] = 0.;
    for(int j=0; j < N; j++)
      r[i] = r[i] + A[i][j] * w[j];
  }
  MPI_Send(r, N/16, MPI_DOUBLE, 0);
}
```

e) Podemos estimar el tiempo de la ejecución paralela haciendo una gráfica de la ejecución. En ella los términos con más peso serán

1) los envíos de submatrices de $N/16$ filas de las matrices, que al constar que N columnas contendrán $N^2/16$ elementos, requiriendo ese número de ciclos para enviarse.
2) los productos matriz-vector de estas submatrices, que requerirán de $2 \times N^2/16 = N^2/8$ operaciones en punto flotante, y por tanto ese número de ciclos para computarse.

los otros términos serían mensajes de envíos de N elementos del vector v o de $N/16$ elementos de los vectores w y r, que son despreciables respecto a los anteriores al tener un tamaño aproximadamente N veces inferior. El diagrama de ejecución resultante sería el siguiente:



Así pues el tiempo de ejecución vendría dado por el envío secuencial por la red de los $2N^2$

elementos de las matrices A Y B a los procesadores esclavos, los cuales podrían realizar sus operaciones en paralelo con dichos envíos, excepto las últimas submatrices de A, que requerirían $N^2/8$ ciclos más. Por tanto, el tiempo de ejecución aproximado sería $(2 + 1/8)N^2 = 2{,}125N^2$ ciclos.

Como el mejor tiempo secuencial era $6N^2+2N$ ciclos, la aceleración aproximada es $(6N^2)/(2{,}125N^2) = 2{,}8$.