EXAMEN DE CONCURRENCIA Y PARALELISMO

Bloque II: Paralelismo

APELLIDOS: _____ **NOMBRE:**_____

Convocatoria ordinaria

24 de mayo de 2018

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

[2,5p] 1. **Conceptos de paralelismo.** Responde a las siguientes preguntas razonando las respuestas, y haciéndolo de forma numérica siempre que sea posible.

a) El bucle principal secuencial de una aplicación viene dado por

```
double a[M][N][P], b[M][N][P];
...
while(condicion) {
   ...
    for(i = 1; i < M-1; i++)
      for(j = 1; j < N-1; j++)
        for(k = 1; k < P-1; k++) {
          b[i][j][k] = f(a[i-1][j][k], a[i+1][j][k], a[i][j-1][k],
                         a[i][j+1][k], a[i][j][k-1], a[i][j][k+1]);
        }
    ...
    memcpy(a, b, M * N * P * sizeof(double)); /*copia b a a */
 }
```

donde `f` utiliza todas sus entradas y su coste computacional es independiente de las mismas. Si $M = N = P = 1024$ y dispones de 64 procesadores, ¿Cómo descompondrías la matriz entre los procesos? Especifica el tamaño de bloque y el número de datos que comunicaría en el peor caso un proceso en cada iteración del bucle principal.

b) Idéntica pregunta con 64 procesadores pero considerando $M = N = 1024$ y $P = 64$.

c) Una aplicación con el mismo patrón computacional que la del apartado a) estima las cuotas de pesca marítima a distintas profundidades, representando en ella las matrices `a` y `b` Europa y su entorno, siendo las dimensiones indexadas por `i`, `j`, y `k` la latitud, la longitud y la profundidad. Supón que, lógicamente, sólo hubiese computación en las celdas de mar. ¿Qué tipo de descomposición de tareas y de asignación de tareas a procesos aplicarías?

d) Si en el caso del apartado a) cada evaluación de `f` requiere 7 ciclos, la copia de un elemento de `b` a `a` requiere 1 ciclo, un mensaje entre dos procesos cualesquiera de n elementos requiere $2^{18} + 4 \times n$ ciclos, y los demás costes son despreciables, calcula el speedup y la eficiencia paralela esperables para el bucle paralelizado.

e) Indica en qué procesador y en qué posición local se haya en el caso del apartado a) el elemento `a[700][400][188]`

SOLUCIÓN

a) Al trabajar sobre una matriz con operaciones independientes en cada punto se debe aplicar una descomposición de dominio. Deberíamos maximizar el volumen computacional respecto al coste de las comunicaciones, dado por el tamaño del perímetro (o suma de superficies, ya que este problema tiene 3 dimensiones) del bloque asignado a cada proceso.

Los 64 procesos se organizan de forma natural en una malla de $4 \times 4 \times 4$ procesadores, de forma que como la matriz tiene tamaño $1024 \times 1024 \times 1024$, cada proceso recibiría un bloque de $\frac{1024}{4} \times \frac{1024}{4} \times \frac{1024}{4} = 256 \times 256 \times 256 = 2^{24}$ elementos.

Los procesos que requieren más comunicaciones son los que están en el centro de la malla, pues requerirán intercambiar las 6 caras de su cubo. Así, el número de elementos que tendrán que comunicar será 256×256 elementos/cara $\times 6$ caras $= 3 \times 2^{17} = 393216$ elementos.

b) Si la matriz tiene tamaño $1024 \times 1024 \times 64$ y aplicásemos la misma malla de $4 \times 4 \times 4$ procesadores estaríamos usando una dimensión de 4 procesadores para dividir una dimensión de 64 elementos en 4 porciones de 16. Esto daría lugar a que los procesos centrales hiciesen dos comunicaciones de caras de 256×256 elementos y 4 de caras de 16×256 elementos, lo que sumarían $2 \times 256 \times 256 + 4 \times 16 \times 256 = 147456$ elementos.

Parece un desperdicio utilizar una dimensión de malla de procesadores para dividir una dimensión de matriz que solo tiene 64 elementos cuando en las otras dos dimensiones cada bloque de cada procesador individual tiene 256 elementos incluso después de haber particionado la dimensión. Probemos por tanto a usar los 64 procesadores para dividir solo las dos dimensiones más grandes del problema, dejando sin dividir la de tamaño 64. Para ellos los organizaríamos en una malla de $8 \times 8 \times 1$ procesadores, de forma que como la matriz tiene tamaño $1024 \times 1024 \times 64$, cada proceso recibiría un bloque de $\frac{1024}{8} \times \frac{1024}{8} \times \frac{64}{1} = 128 \times 128 \times 64$ elementos. En la tercera dimensión ahora no se requerirían comunicaciones, y las caras de las otras dos dimensiones, que en total requerirían 4 comunicaciones en los procesos centrales, constarían de 128×64 elementos. Así pues, en el peor caso un proceso transmitiría $4 \times 128 \times 64 = 32768$ elementos, que es claramente mejor que la primera posibilidad analizada.

c) Al igual que en el apartado a), al trabajar sobre una matriz con operaciones independientes en cada punto se debe aplicar una descomposición de dominio. La diferencia esencial es que en este caso el coste computacional se concentra en ciertas áreas de la matriz (donde hay mar), siendo nulo en otras (donde hay tierra). Por ello en lugar de un reparto con un solo bloque por procesador, como en los apartados anteriores, sería deseable un reparto cíclico de bloques que equilibrase la carga de trabajo entre los procesadores. Hay que tener en cuenta que, dado el patrón computacional observado en a), este reparto daría lugar a un aumento de los costes de las comunicaciones. Por ello debería contrastarse si el coste de las comunicaciones adicionales no anulase la ganancia de rendimiento obtenida gracias al mayor equilibrio de carga.

d) Dado que Speedup$= \frac{T.\ secuencial}{T.\ paralelo} = \frac{T.\ secuencial}{T.\ comp.\ paralelo + T.\ comm.\ paralelo}$, calculamos que

El código secuencial requeriría $1024 \times 1024 \times 1024$ elementos $\times (7 + 1)$ ciclos $= 2^{33} = 8589934592$ ciclos.

En el código paralelo, en el peor caso un proceso operaría sobre un bloque de $256 \times 256 \times 256$ elementos con comunicaciones por las 6 caras, lo que requeriría

$T.\ comp.\ paralelo = 256 \times 256 \times 256$ elementos $\times\ (7 + 1)$ ciclos $= 2^{27} = 134217728$ ciclos

$T.\ comm.\ paralelo = (2^{18} + 4 \times 256 \times 256)$ ciclos/mensaje $\times\ 6$ mensajes $= (2^{18} + 2^{18}) \times 6 = 2^{19} \times 6 = 3 \times 2^{20} = 3145728$ ciclos

Con lo que Speedup$= \frac{8589934592}{134217728 + 3145728} = 62{,}53$ ó $\frac{2^{33}}{2^{27} + 3 \times 2^{20}} = \frac{2^{33}}{2^{20} \times (2^7 + 3)} = \frac{2^{13}}{2^7 + 3} = \frac{8192}{131} = 62{,}53$

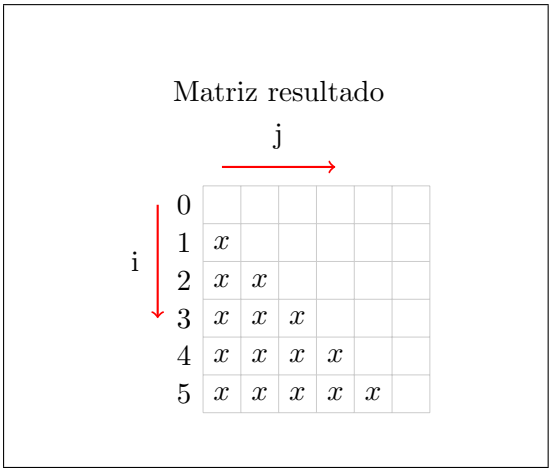
Y la eficiencia$= \frac{Speedup}{Num\ procs} = \frac{62{,}53}{64} = 0{,}977$

e) El procesador asociado sería $\left(\lfloor\frac{700}{256}\rfloor, \lfloor\frac{400}{256}\rfloor, \lfloor\frac{188}{256}\rfloor\right) = (2, 1, 0)$
La posición local sería $(700 \text{ mód } 256, 400 \text{ mód } 256, 188 \text{ mód } 256) = (188, 144, 188)$.

[2,5p] 2. **Diseño de algoritmos paralelos.** Un grupo de investigadores holandeses ha estado recopilando en una base de datos información sobre 100.000 gatos de Ámsterdam y sus alrededores. Para cada gato se han extraído y almacenado alrededor de 1.500 características. Su objetivo es comparar cada par de gatos según una función $f : \mathbb{G}^2 \to \mathbb{R}$ que les permita clasificar los gatos según un determinado criterio, donde $\mathbb{G}$ es el conjunto de gatos incluidos en la base de datos. La función f es necesariamente simétrica (es decir, $f(g_i, g_j) = f(g_j, g_i)$), e incorpora un subconjunto de estas características. Esta función tiene un coste fijo (independiente de sus parámetros).

Para evaluar cada función han diseñado el siguiente algoritmo, que recibe como entrada un array de N elementos de tipo *gato_t*, donde N es el número de gatos y *gato_t* es un tipo de dato que incluye todas las características que han recopilado los investigadores, y devuelve una matriz triangular inferior de $N \times N$ con el resultado de aplicar una función f a cada par de datos de entrada:

```
gato_t gatos[N];
double res[N][N];

// inicializa el array de gatos
lee(gatos);

// aplica la función f
for (i=0 ; i<N ; ++i)
  for (j=0 ; j<i ; ++j)
    res[i][j] = f(gatos[i],gatos[j]);

// escribe el resultado
escribe(res);
```



El problema con el que se encuentran los investigadores es que el número de comparaciones que se debe hacer es considerablemente alto ($N(N-1)/2$), y con su ordenador de sobremesa tardarían semanas en aplicar cada función al array de entrada, por lo que necesitan idear una solución paralela que les permita obtener resultados en el menor tiempo posible.

Asumimos que tanto el número de gatos N como el número de comparaciones $N(N-1)/2$ son múltiplos del número de procesos que trabajarán en paralelo. La entrada/salida (funciones *lee*(...) y *escribe*(...)) debe realizarla un único proceso. Consideraremos que en el sistema paralelo el tiempo de comunicación es siempre despreciable en comparación con el tiempo de computación.

a) ¿Qué tipo de descomposición de tareas debemos hacer? ¿Qué tipo de asignación de tareas?

b) Esboza un pseudocódigo MPI que paralelice la solución propuesta. A continuación tienes las firmas de las funciones MPI que puedes utilizar. Puedes utilizar un tipo de dato predefinido al que hemos denominado *MPI_GATO*, que incorpora la información contenida en el tipo *gato_t*.

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm)
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm, MPI_Status *status)
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
              int root, MPI_Comm comm)
int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int recvcnt, MPI_Datatype recvtype,
                int root, MPI_Comm comm)
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
               void *recvbuf, int recvcount, MPI_Datatype recvtype,
               int root, MPI_Comm comm)
int MPI_Reduce(void *sendbuf, void *recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op,
               int root, MPI_Comm comm)
```

c) Suponiendo que la ejecución de la función f tarda 1 segundo, calcula el speedup y la eficiencia paralela de la implementación anterior para $N = 12$ y 3 procesos.

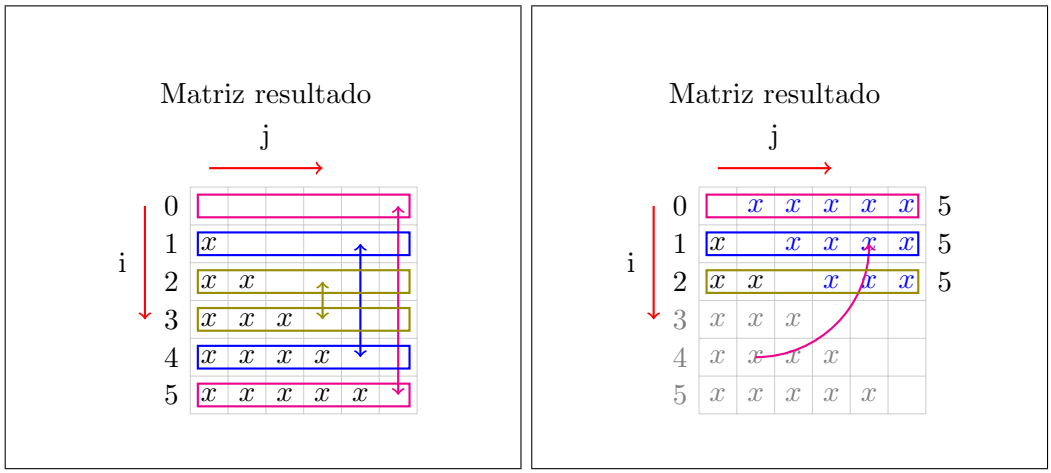
d) Queremos obtener en todos los procesos el máximo valor de la matriz resultado antes de mandarla al proceso 0. Para ello cada proceso primero obtiene el máximo local de la porción que él ha calculado y luego debe intercambiar mensajes con los demás procesos de forma que todos obtengan el máximo global. Indica con un grafo de dependencias estáticas la forma de realizar esta operación en el mínimo número de pasos en el caso específico de que tengamos 4 procesos. ¿Qué comunicaciones se llevan a cabo entre procesos?

SOLUCIÓN

a) Estamos hablando de una **descomposición de dominio sobre el cálculo de la matriz resultado** (ojo, ¡no sobre el vector de gatos!), que podemos hacer de 2 maneras. La forma más sencilla es asignar una tarea por fila, lo que produciría un total de N tareas de carga desigual pero conocida de antemano: la iteración i hace 'i' comparaciones. En este caso, la asignación eficaz más sencilla sería **estática y cíclica**, aunque sin más consideraciones provocará un cierto desbalanceo (diferencia de carga entre los procesos con más y menos trabajo) de $\lceil N * (p-1)/p \rceil$.

Una distribución **dinámica maestro-esclavo**, dado que el coste de las iteraciones es incremental, nos llevaría a un reparto puramente cíclico, pero con las desventajas de que tenemos más comunicaciones y un proceso menos trabajando, además de que la implementación del algoritmo sería mucho más compleja. Para conseguir un buen balanceo con esta estrategia tendremos que repartir las filas en orden inverso (carga computacional decreciente).

Si quisiésemos hacer un reparto balanceado podemos considerar que N es múltiplo de $2p$. Si tomamos conjuntamente la primera fila y la última, la segunda y la penúltima, etc., veremos que la carga de trabajo siempre es $N-1$. En general, agrupamos la fila i y la fila $N-i-1$, lo que denominaremos aquí reparto **bloque simétrico**. Por tanto, de ese modo tendremos $N/2$ iteraciones (tareas) de carga de trabajo homogénea que podemos asignar estáticamente de forma consecutiva a los procesos:



b) Existen muchas formas de conseguir un buen balanceo de carga sin incurrir en un exceso de comunicaciones, pero las más sencillas de implementar son la opción de reparto **cíclico** y **bloque simétrico**, descritas en el apartado anterior. Es importante tener en cuenta que se indica que N es múltiplo del número de procesos, por lo que no tenemos que preocuparnos por iteraciones sobrantes en el reparto.

b.1) Reparto cíclico

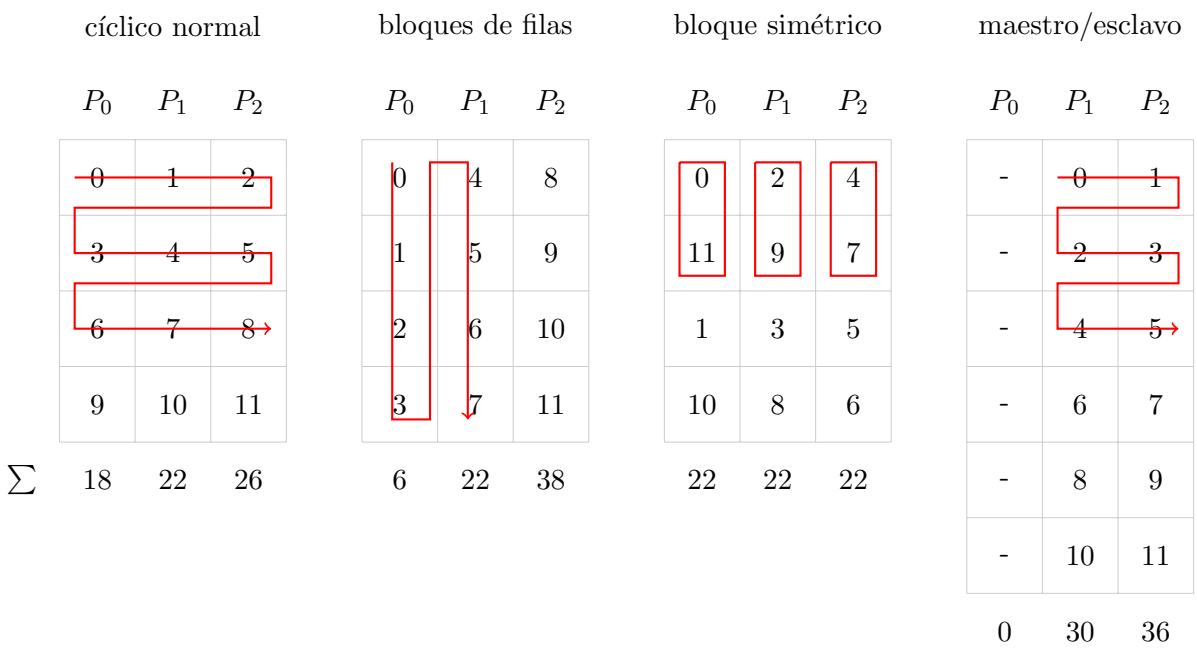
```
int block_size = N/n_procs;
double res_local[block_size][N];

if (root)
  lee(gatos);

MPI_Bcast(gatos, N, MPI_GATO, 0, MPI_COMM_WORLD);

/* el reparto del bucle principal es similar al utilizado
 * en la primera práctica de la asignatura para el cálculo
 * de pi */
int i_local = 0;
for (i=rank ; i<N ; i+=n_procs)
{
  for (j=0 ; j<i ; ++j)
    res_local[i_local][j] = f(gatos[i], gatos[j]);
  ++i_local;
}

if (!root)
{
  // enviamos el bloque calculado al proceso root
  MPI_Send(res_local, N*block_size, MPI_DOUBLE,
           0, 1, MPI_COMM_WORLD);
}
else
{
  for (proc = 0; proc < n_procs; proc++)
  {
    // recibimos cada bloque excepto el nuestro propio
    if (proc > 0)
      MPI_Recv(res_local, N*block_size, MPI_DOUBLE, proc,
               MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    // y lo ordenamos en la matriz resultado
    for (i=0; i<block_size; i++)
    {
      memcpy(res[proc + i*n_procs],
             res_local[i],
             N * sizeof(double));
    }
  }
  escribe(res);
}
```

b.2) Reparto bloque simétrico

```
int block_size = N/n_procs;
int half_block = block_size/2;
double res_local[block_size][N];

if (root)
  lee(gatos);

MPI_Bcast(gatos, N, MPI_GATO, 0, MPI_COMM_WORLD);

/* el reparto del bucle principal es similar al utilizado
 * en la segunda práctica de la asignatura para el cálculo
 * del fractal */
int i_local = 0;
for (i=half_block*rank ; i<half_block*(rank+1) ; i++)
{
  // calculamos tanto la fila i como la fila N-i-1
  int symm_i = N-i-1;
  for (j=0 ; j<i ; ++j)
    res_local[i_local][j] = f(gatos[i], gatos[j]);
  for (j=0 ; j<symm_i ; ++j)
    res_local[block_size - i_local - 1][j] =
      f(gatos[symm_i], gatos[j]);
  ++i_local;
}

if (!root)
{
  // enviamos los bloques calculados al proceso root
  MPI_Send(res_local, N*block_size, MPI_DOUBLE,
           0, 1, MPI_COMM_WORLD);
}
else
{
  for (proc = 0; proc < n_procs; proc++)
  {
    // recibimos cada bloque excepto el nuestro propio
    if (proc > 0)
      MPI_Recv(res_local, N*block_size, MPI_DOUBLE, proc,
               MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    // copiamos la primera mitad en su posición
    memcpy(res [ half_block * proc ],
           res_local,
           half_block * N * sizeof ( double ));
    // y la segunda mitad en su posición simétrica
    memcpy(res [ N - half_block*(proc+1) ] ,
           res_local[ half_block ] ,
           half_block * N * sizeof ( double ));
  }
  escribe(res);
}
```

c) Recordamos que el speedup es el tiempo de ejecución secuencial dividido por el tiempo de ejecución paralelo. Con los datos que conocemos, que la ejecución de f tarda un segundo, el tiempo de ejecución secuencial será $N(N-1)/2$, que es el número de comparaciones. Como $N = 12$, entonces $T_s = 12 * 11/2 = 66$ segundos. El tiempo paralelo es el **tiempo de ejecución del proceso que más tarde**, y dependerá de la estrategia que hayamos escogido. Para calcularlo de forma fácil, como tenemos un número pequeño de iteraciones y además sabemos que la carga computacional de cada iteración coincide con su índice (es decir, la fila i realiza i comparaciones de un segundo), podemos dibujar cómo se asignan las iteraciones a los procesos, y de ese modo sabremos cuanto tarda en total cada uno de ellos.

Por último, la eficiencia paralela se calcula dividiendo el speedup entre el número total de procesos que intervienen.



- Con un reparto cíclico por filas, el speedup es $\frac{66}{26} = 2{,}54$ y la eficiencia paralela $\frac{2{,}54}{3} = 0{,}85$.
- Con reparto bloque, $\frac{66}{38} = 1{,}74$ y la eficiencia paralela $\frac{1{,}74}{3} = 0{,}58$.
- Con reparto bloque simétrico, $\frac{66}{22} = 3$ y la eficiencia paralela $\frac{3}{3} = 1$.
- Con maestro/esclavo, $\frac{66}{36} = 1{,}83$ y la eficiencia paralela $\frac{1{,}83}{3} = 0{,}61$.

d) La solución es la propuesta en las transparencias del tema 3.4 en la sección de asignación de grafos de dependencias estáticas para el ejemplo de reducción con replicación. La solución se reduce a 2 pasos ($log_2(n\_procs)$), y en cada paso los procesos intercambian y actualizan valores de máximo local.