EXAMEN DE CONCURRENCIA Y PARALELISMO

Bloque II: Paralelismo

APELLIDOS: _____ **NOMBRE:** _____

Convocatoria ordinaria

19 de junio de 2020

[1p] 1. **Conceptos de paralelismo.** Partiendo de una matriz bidimensional A con M filas y N columnas queremos calcular una matriz B, también con M filas y N columnas, con el valor al cubo de cada elemento de A, de forma que tras ejecutar nuestro programa todos los elementos $b_{i,j}$ con $0 \leq i < M$ y $0 \leq j < N$ se cumple que $b_{i,j} = a_{i,j} * a_{i,j} * a_{i,j}$. Este programa tiene tres fases: lectura de la matriz de un fichero de entrada, elevación al cubo de toda la matriz y escritura de la matriz resultante de fichero.

Ejecutamos este programa con una malla también bidimensional de PxQ procesos en una arquitectura donde la primera y la última fase, que trabajan con el sistema de ficheros, solo puede llevarlas a cabo el proceso 0. El tiempo de lectura de cada elemento de la matriz es igual al de escritura (T_{es}) y el tiempo de realizar una simple multiplicación de dos floats es T_{prod}. La distribución de los datos de las matrices A y B entre los procesos es estática bloque pura en dos dimensiones. Aunque se necesitan comunicaciones para repartir A entre los procesos y recoger los resultados de B, estas son tan rápidas que se puede decir que su coste es 0.

De cara a contestar las preguntas que figuran a continuación, los datos concretos serán diferentes para cada alumno, dependiendo de su número de DNI (pasaporte para alumnos sin nacionalidad española). Si tu DNI es $C_1C_2C_3C_4C_5C_6C_7C_8L$ (en caso de alumnos sin nacionalidad española $C_1C_2C_3C_4C_5C_6C_7$ son las siete cifras que aparecen en el número de pasaporte). Teniendo en cuenta lo anterior, los valores numéricos concretos son:

- $M = 500 + 10 * C_1 + C_2$
- $N = 300 + 10 * C_1 + C_2$
- $P = 2 + C_3$
- $Q = 2 + C_4$
- $T_{es} = 1 + C_5$ segundos
- $T_{prod} = 1 + C_6$ segundos

(a) ¿Qué cantidad de elementos será asignado a cada uno de los procesos? Proporciona el valor numérico pero razona tu respuesta (una respuesta sin razonamiento correcto no será válida aunque el resultado numérico sea correcto). [0,5p]

(b) ¿A qué proceso será asignado el elemento (100, 100)? Proporciona el valor numérico pero razona tu respuesta (una respuesta sin razonamiento correcto no será válida aunque el resultado numérico sea correcto). [0,25p]

(c) ¿Cuál será el speedup al ejecutar con estos PxQ procesos? Proporciona el valor numérico pero razona tu respuesta (una respuesta sin razonamiento correcto no será válida aunque el resultado numérico sea correcto). [0,25p]

SOLUCIÓN

(a)
- La mayoría de procesos tendrán $\lceil \frac{M}{P} \rceil$ filas y $\lceil \frac{N}{Q} \rceil$ columnas.
- Los procesos de la última fila de la malla tendrán $M - (P-1) * \lceil \frac{M}{P} \rceil$ filas y $\lceil \frac{N}{Q} \rceil$ columnas.
- Los procesos de la última columna de la malla tendrán $\lceil \frac{M}{P} \rceil$ filas y $N - (Q-1) * \lceil \frac{N}{Q} \rceil$ columnas.
- El proceso $P * Q - 1$ tendrá $M - (P-1) * \lceil \frac{M}{P} \rceil$ filas y $N - (Q-1) * \lceil \frac{N}{Q} \rceil$ columnas.

(b) Representando los procesos en dos dimensiones está en el proceso $\lfloor \frac{100}{\lceil \frac{M}{P} \rceil} \rfloor$X$\lfloor \frac{100}{\lceil \frac{N}{Q} \rceil} \rfloor$. Si lo representamos en una dimensión está en el proceso $\lfloor \frac{100}{\lceil \frac{M}{P} \rceil} \rfloor * Q + \lfloor \frac{100}{\lceil \frac{N}{Q} \rceil} \rfloor$.

(c) El tiempo de acceso al sistema de ficheros (lectura y escritura) es: $2 * T_{es} * M * N$, que es inherentemente secuencial. El tiempo de cálculo secuencial es $2 * T_{prod} * M * N$. Así que el tiempo secuencial es $2 * M * N * (T_{es} + T_{prod})$.

El tiempo de lectura y escritura no varía en el caso de tener varios procesos. El tiempo de cálculo es igual al tiempo del proceso que más tarda, que es cualquiera con los bloques enteros ($\lceil \frac{M}{P} \rceil$ filas y $\lceil \frac{N}{Q} \rceil$ columnas). Por tanto el tiempo paralelo es $2 * (T_{es} * M * N + T_{prod} * \lceil \frac{M}{P} \rceil * \lceil \frac{N}{Q} \rceil$. El speedup sale de dividir el tiempo secuencial entre el tiempo paralelo.

[1p] 2. **Diseño de algoritmos paralelos.** Un candidato cuya ideología consiste en que a la gente se le debe dar lo que pide se va a presentar a unas elecciones. A fin de confeccionar su programa electoral, su equipo determina cuáles son las N cuestiones $Q_0, \ldots Q_{N-1}$ que más preocupan al electorado, así como el conjunto de posturas posible P_i en torno a cada cuestión Q_i, denominando $p_i = |P_i|$ al número de posturas posibles para la cuestión Q_i. Tanto todo p_i como el número total de combinaciones posibles de posturas que podría adoptar una persona $T = \prod_{i=0}^{N-1} p_i$ se pueden representar usando el tipo `int`. El equipo hace entonces una encuesta a M personas y rellena con ella una tabla `Pos[M][N]` donde `Pos[i][j]` es un entero entre 0 y $p_j - 1$ que codifica la postura de la persona i en torno a la cuestión Q_j. El equipo intenta determinar entonces el programa electoral del candidato aplicando el siguiente algoritmo:

```
void main(int argc, char *argv[]) {
    int Support[T];
    int Pos[M][N];
    int best = 0;
    int max = 0;

    ReadPoll(Pos);

    for(int i = 0; i < T; i++)
      Support[i] = 0;

    for(int i = 0; i < M; i++)
      Support[cod(Pos[i])]++;

    for(int i = 0; i < T; i++)
      if(Support[i] > max) {
        best = i;
        max = Support[i];
      }

    decod(best);
}
```

en donde `cod` recibe un puntero a una fila de la matriz `Pos` y devuelve un entero entre 0 y $T-1$ que codifica de forma unívoca la combinación de posturas reflejadas en la encuesta representada en esa fila, `Support[i]` mide el número de apoyos a la combinación de posturas codificada con el entero `i`, y `decod(best)` imprime por pantalla las posturas que vienen codificadas por el código entero de entrada `best`. La función `cod` es con diferencia la más pesada, haciendo que el bucle donde se usa domine ampliamente el tiempo de computación.

La encuesta llega el último día en que se puede presentar el programa electoral y hay que paralelizar el algoritmo para poder presentarlo en plazo. Para ello se dispone de un sistema con P procesadores en el que sólo P_0 puede hacer entrada/salida, tanto de disco como de pantalla. Se sabe que M es del orden de decenas de miles, N de decenas, y $T \approx 3.5N$.

(a) Diseña una paralelización del problema asumiendo que todas las iteraciones del mismo bucle tienen el mismo coste. Indica el tipo de descomposición y de asignación de tareas, justificando tu respuesta. [0,25p]

(b) Esboza un pseudocódigo MPI que paralelice la solución propuesta. A continuación tienes las firmas de las funciones MPI que puedes utilizar. Asume que M, N y T son conocidos de antemano y que todos ellos son múltiplos de P, el cual sólo se conoce en tiempo de ejecución. Utiliza colectivas siempre que sea posible. [0,5p]

```
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) Explica brevemente cómo cambiarías el diseño, de ser necesario, si el coste de la función `cod` puede variar mucho dependiendo de la entrada y no se puede determinar de antemano. Es decir, la codificación de algunas filas puede necesitar más tiempo que la de otras y antes de ejecutar no se puede estimar cuánto tiempo va a requerir cada una. [0,25p]

SOLUCIÓN

(a) Nos centramos en el bucle que usa `cod` para diseñar la paralelización, ya que es el más costoso con diferencia. En él el coste de cada iteración, la cual usa una fila entera de la matriz `Pos`, cuyo tamaño es conocido de antemano, es constante. Así pues, conviene aplicar una descomposición de dominio donde cada tarea procese un número lo más similar posible de filas. Como el enunciado indica que M es divisible por P, lo idóneo sería construir una tarea por procesador de forma que cada una procesaría M/P filas, pero vale cualquier número de tareas y filas por tarea que equilibre el trabajo entre los procesadores.

La asignación será estática, asignando el mismo número de tareas, y por tanto de filas, a cada proceso. Lo más sencillo sería aplicar una distribución bloque pura para minimizar el número de mensajes necesarios inicialmente en la distribución de las filas desde el Proceso 0. Como no se aportan detalles acerca del rendimiento de la red y de si es mejor agrupar comunicaciones, es válido aplicar cualquier tamaño de bloque mientras se asigne el mismo número de filas o encuestas a cada proceso.

(b)

```
int Support[T];
int P, MP, miID, best = 0, max = 0, *Pos;

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &P);
MPI_Comm_rank(MPI_COMM_WORLD, &miId);

MP = M/P;

if(!myid){
  Pos = (int *)malloc(sizeof(int) * M * N);
  ReadPoll(Pos);
} else {
  Pos = (int *)malloc(sizeof(int) * MP * N);
}

for(int i = 0; i < T; i++)
  Support[i] = 0;

MPI_Scatter(Pos, MP*N, MPI_INT,
            myId ? Pos : MPI_IN_PLACE, MP*N, MPI_INT, 0, MPI_COMM_WORLD);

for(int i = 0; i < MP; i++)
  Support[cod(Pos + i * N)]++;

MPI_Reduce(myId ? Support : MPI_IN_PLACE, Support, T, MPI_INT,
           MPI_SUM, 0, MPI_COMM_WORLD);

if(!myid){
  for(int i = 0; i < T; i++)
    if(Support[i] > max) {
      best = i;
      max = Support[i];
    }

    decod(best);
}

MPI_Finalize();
```

(c) La fuerte variabilidad no predecible a priori del coste de las iteraciones del bucle más pesado podría dar lugar a desequilibrios de carga importantes entre los procesadores si seguimos realizando una asignación estática. Para evitar este problema tendríamos que recurrir a generar más de una tarea por proceso, por ejemplo una por fila, y a aplicar una asignación dinámica en la que se las fuese asignando a los procesos a medida que estos fuesen terminando las que se les hubiesen asignado previamente. Esto podría lograrse por ejemplo con un esquema maestro-esclavo, aunque también podría aplicarse un esquema descentralizado tras hacer una distribución inicial de las filas entre los procesos.