# EXAMEN DE CONCURRENCIA Y PARALELISMO

## Bloque II: Paralelismo

**APELLIDOS:** _____ **NOMBRE:** _____

**Convocatoria ordinaria**

**26 de mayo de 2017**

**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. **Conceptos de paralelismo.** El código de la Figura 1 es la versión paralelizada usando el paradigma SPMD de un código secuencial con las siguientes características:

- El tamaño de la variable `data` es $L = 2^{32}$ elementos.
- La entrada de datos tiene un tiempo de ejecución de $2^{10}$ segundos.
- El procesamiento secuencial en el código original tiene un tiempo de ejecución de $2^{32}$ segundos.
- La paralelización del segmento de procesamiento (a partir de la línea 16) tiene una eficiencia del 100 %.
- Hay un único procesador capaz de acceder al sistema de almacenamiento.
- El sistema cuenta con una red totalmente conexa con un tiempo de envío de $2^{-20}$ $segundos/byte$.

Contesta de forma razonada a las siguientes preguntas:

*a*) ¿Cuál es la aceleración máxima **teórica** del problema original?

```
1         char data[L];
2         ...
3         MPI_Init( &argsc, &argsv );
4         MPI_Comm_rank( MPI_COMM_WORLD, &rank );
5         MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
6
7         if( rank == 0 ) {
8                 data = leer_datos();
9                 for( int i = 1; i < nprocs; ++i ) {
10                        MPI_Send( data, L, MPI_BYTE, i, 0, MPI_COMM_WORLD );
11                }
12        } else {
13                MPI_Recv( data, L, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
                     MPI_STATUS_IGNORE );
14        }
15
16        // Procesamiento
17        ...
```

Figura 1

*b*) ¿Cuál es la aceleración (*speedup*) obtenida por el código paralelo con $nprocs = 16$ procesos? ¿Y la eficiencia?

*c*) ¿A partir de qué número de procesos el coste de las comunicaciones hace que la paralelización no obtenga ninguna mejora en tiempo de ejecución con respecto a la versión secuencial?

*d*) ¿Qué operación colectiva MPI se puede utilizar para sustituir a las líneas 9–14? ¿Cuál sería la implementación más eficiente sobre el sistema descrito? ¿Qué aceleración se obtendría usando esta implementación con el número de procesos calculado en el apartado anterior?

*e*) ¿Dirías que esta paralelización presenta escalabilidad fuerte?

[SOLUCIÓN]

*a*) El problema tiene una parte intrínsecamente secuencial con duración $2^{10}$ segundos y otra perfectamente paralelizable con duración $2^{32}$ segundos. Aplicando la Ley de Amdahl:

$$sp(p \to \infty) = \frac{T_{sec}}{T_{par}(p \to \infty)} = \lim_{p \to \infty} \frac{2^{10} + 2^{32}}{2^{10} + {}^{2^{32}}/_{p}} = \frac{2^{10} + 2^{32}}{2^{10}} \simeq 4 \times 10^6$$

*b*) El tiempo de ejecución del código secuencial original es igual al tiempo de entrada de datos más el tiempo de procesamiento:

$$T_{sec} = T_{read} + T_{comp}(1) = 2^{10} + 2^{32}\ s.$$

En cuanto al tiempo de ejecución paralelo, será igual a a la suma de: (i) el tiempo de entrada de datos, que no varía con respecto a la ejecución secuencial; (ii) el tiempo de reparto de los datos entre los procesos (líneas 9–14); y (iii) el tiempo de procesamiento.

$$T_{par}(p) = T_{read} + T_{comms} + T_{comp}(p)$$

Trabajando con 16 procesos:

$$T_{comms} = 2^{-20} \times 2^{32} \times 15 = 61440\ s.$$

$$T_{comp}(16) = 2^{32}/16 = 2^{28}\ s$$

$$sp(16) = \frac{T_{sec}}{T_{par}(16)} = \frac{2^{10} + 2^{32}}{2^{10} + 61440 + 2^{28}} = 15{,}99$$

$$eff(16) = \frac{sp(16)}{16} = \frac{15{,}99}{16} = 0{,}99$$

*c*) Debemos encontrar cuándo el tiempo de computación secuencial se iguala al tiempo de computación paralelo más el tiempo de comunicaciones:

$$T_{comm}(1) = T_{comms} + T_{comp}(p) \Rightarrow$$
$$2^{32} = 2^{-20} \times 2^{32} \times (p-1) + {}^{2^{32}}/_{p} \Rightarrow$$
$$1 = 2^{-20} \times (p-1) + {}^{1}/_{p} \Rightarrow$$
$$2^{-20} \times p \times (p-1) = p - 1 \Rightarrow$$
$$\begin{cases} p = 1 \\ 2^{-20} \times p = 1 \Rightarrow p = 2^{20} \end{cases}$$

*d*) Las líneas 9–14 se pueden sustituir por un *broadcast*, que en el entorno de computación descrito se beneficiaría de una implementación binomial. Ahora el tiempo de ejecución:

$$T_{par}(p) = 2^{10} + 2^{-20} \times 2^{32} \times \lceil log_2(p) \rceil + {}^{2^{32}}/_{p}$$
$$T_{par}(2^{20}) = 2^{10} + 2^{12} \times 20 + 2^{12}$$
$$sp(2^{20}) = \frac{2^{10} + 2^{32}}{2^{10} + 2^{12} \times 21} = 49344{,}76$$

*e*) La paralelización presenta escalabilidad fuerte mientras el tiempo de comunicaciones sea sensiblemente inferior al tiempo de computación, es decir, mientras:

$${}^{2^{32}}/_{p} >>> 2^{12} \times log_2(p)$$

Para un número de procesos muy superior a $p = 2^{10}$, la eficiencia empieza a resentirse sensiblemente.

**[2,5p]** 2. **Diseño de algoritmos paralelos.** Algunas de las aplicaciones bioinformáticas más utilizadas en la actualidad consisten en alineadores de secuencias de ADN. Estos programas reciben como entrada dos ficheros de texto, cada uno con la siguiente información:

- Un genoma de referencia, que consiste en una lista muy larga de caracteres (varios millones).
- Un conjunto de muchas secuencias de ADN. Cada una de ellas es otra lista de caracteres pero de un tamaño más reducido (alrededor de cien caracteres).

El objetivo de algunos de estos alineadores es proporcionar, para cada secuencia de ADN, la primera posición dentro del genoma donde se pueden encontrar exactamente los mismos caracteres que en dicha secuencia. Se puede ver un ejemplo en la Figura 2. Es decir, todas las secuencias se compararán con el genoma de forma independiente y se proporcionará la posición de inicio del alineamiento para cada una de ellas.

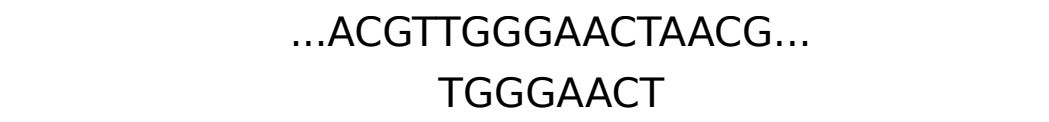


Figura 2: Ejemplo de alineamiento de una secuencia a un genoma

Partiendo del pseudocódigo secuencial que se muestra a continuación, se desea paralelizar un determinado alineador en un computador de memoria distribuida en el que solo uno de los nodos (el que estará asociado siempre al Proceso 0) tiene acceso al sistema de ficheros y además no existe soporte para solapar computación y comunicación. Se pide:

```
char genoma[L];
char secuencias[N*M];
int res[N];

// Lee las entradas de fichero
lee_genoma(genoma);
lee_secuencias(N, secuencias);

// Alinea todas las secuencias
for(i=0; i<N; i++){
  res[i] = alinea_secuencia(genoma, secuencias[i*M]);
}

imprime_alineamientos(res);
```

*a*) Diseña una paralelización del problema para $P$ nodos de cómputo asumiendo que el coste de alinear cada secuencia es homogéneo (se tarda exactamente el mismo tiempo en alinear cualquiera de las secuencias). Detalla la descomposición en tareas y la asignación a procesos. Justifica todas las decisiones de diseño.

*b*) Esboza un pseudocódigo MPI que paralelice la solución propuesta. Céntrate en las comunicaciones, que deben ser implementadas exclusivamente con rutinas $MPI\_Send()$ y $MPI\_Recv$. Asume que la longitud del genoma ($L$), el número de secuencias ($N$) y la longitud ($M$) de cada una de las secuencias los conoces inicialmente, y que $N$ es múltiplo de $P$. Deja por escrito todas las asunciones que consideres necesarias.

*c*) Explica qué cambios deberías hacer en el pseudocódigo anterior si $N$ no fuese siempre múltiplo de $P$ (sin desarrollar un nuevo pseudocódigo).

*d*) Modifica el pseudocódigo del apartado *b*) utilizando colectivas en lugar de comunicaciones punto a punto siempre que sea posible (considera de nuevo que $N$ es múltiplo de $P$).

*e*) Explica brevemente cómo cambiarías el diseño, de ser necesario, si el coste de alinear diferentes secuencias no es constante y además no se puede determinar de antemano. Es decir, algunas secuencias pueden necesitar más tiempo que otras para ser alineadas y antes de ejecutar no se puede estimar cuánto tiempo va a tardar cada secuencia.

[SOLUCIÓN]

*a*) Como el coste de alinear cada secuencia es constante y el número de secuencias se conoce de antemano se utilizará una descomposición de dominio donde cada tarea consista en alinear el mismo número de secuencias. Es válido que elijan una tarea para el alineamiento de cada secuencia, o que agrupen varios alineamientos en la misma tarea. Lo fundamental es que cada tarea involucre al mismo número de secuencias y que haya al menos $P$ tareas.

La asignación será estática, asignando el mismo número de tareas (y, por tanto, el mismo número de secuencias) a cada proceso. Yo usaría una distribución bloque pura para minimizar el número de mensajes necesarios inicialmente en la distribución de las secuencias desde el Proceso 0. Sin embargo, como no se aportan detalles acerca del rendimiento de la red y de si es mejor agrupar comunicaciones, daré por bueno que usen cualquier tamaño de bloque (incluso distribución cíclica de las secuencias) mientras asignen el mismo número de tareas a cada proceso.

b)

```
 ...
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&P);
MPI_Comm_rank(MPI_COMM_WORLD,&miId);
...
int miN = N/P;
if(!myid){
  lee_genoma(genoma);
  lee_secuencias(N, secuencias);

  // Envia el genoma y el bloque de secuencias a cada proceso
  memcpy(misSecuencias, secuencias, M*miN*sizeof(char));
  for(i=1; i<P; i++){
    MPI_Send(genoma, L, MPI_CHAR, i, 0, MPI_COMM_WORLD);
    MPI_Send(&secuencias[i*miN], M*miN, MPI_CHAR, i, 0, MPI_COMM_WORLD);
  }
} else { // Recibe el genoma y el bloque de secuencias
  MPI_Recv(genoma, L, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
  MPI_Recv(misSecuencias, M*miN, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
}

// Todos los procesos operan sobre sus secuencias
for(i=0; i<miN; i++){
  miRes[i] = alinea_secuencia(genoma, &misSecuencias[i*M]);
}

// Todos los procesos mandan al 0
if(!miId){
  memcpy(res, miRes, miN*sizeof(int));
  for(i=1; i<P; i++){
    // Optimizado para recepcion fuera de orden
    MPI_Recv(buffer, miN, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD, &status);
    memcpy(res[miN*status.source], buffer, miN*sizeof(int));
  }

  imprime_alineamientos(res);
} else {
  MPI_Send(miRes, miN, MPI_INT, 0, 1, MPI_COMM_WORLD);
}

MPI_Finalize();
```

c) Todos los procesos a excepción del último tendrán asociadas $b = \lceil \frac{N}{P} \rceil$ secuencias, mientras que el proceso con id $P-1$ tendrá $N - ((P-1) \times b)$ filas. Hay que tener cuidado a la hora de hacer los envíos de las secuencias y la recepción de los resultados tanto por el número de elementos a enviar como por la posición en la que se empieza con cada mensaje.

*d*)

```
 ...
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&P);
MPI_Comm_rank(MPI_COMM_WORLD,&miId);
...
int miN = N/P;
if(!myid){
  lee_genoma(genoma);
  lee_secuencias(N, secuencias);
}

// Envia el genoma y el bloque de secuencias a cada proceso
MPI_Bcast(genoma, L, MPI_CHAR, 0, MPI_COMM_WORLD);
MPI_Scatter(secuencias, M*miN, MPI_CHAR, misSecuencias,
            M*miN, MPI_CHAR, 0, MPI_COMM_WORLD);

// Todos los procesos operan sobre sus secuencias
for(i=0; i<miN; i++){
  miRes[i] = alinea_secuencia(genoma, &misSecuencias[i*M], M);
}

// Todos los procesos mandan al 0
MPI_Gather(miRes, miN, MPI_INT, res, miN, MPI_INT, 0, MPI_COMM_WORLD);

if(!miId)
  imprime_alineamientos(res);

MPI_Finalize();
```

*e*) En este caso una asignación estática puede generar desbalanceo de carga ya que, aunque asignemos el mismo número de secuencias a cada proceso, el coste computacional puede ser distinto. lo correcto sería utilizar una aproximación maestro-esclavo o completamente distribuida en que las tareas (conjuntos de secuencias) se vayan reasignando a los procesos que se queden ocioso (que probablemente hayan trabajado con secuencias más fáciles de alinear).