UNIVERSIDAD CARLOS III DE MADRID
DEPARTAMENTO DE INFORMÁTICA
INGENIERÍA EN INFORMÁTICA.
ARQUITECTURA DE COMPUTADORES II
4 de junio de 2009

Para la realización del presente examen se dispondrá de **2 1/2 horas. NO** se podrán utilizar libros ni apuntes. Entregar cada ejercicio en hojas separadas.

Ejercicio 1 (3 puntos)

1.- Responde justificadamente a las siguientes preguntas:

1. Define **eficiencia** como métrica de rendimiento. ¿Qué relación tiene con la escalabilidad de una aplicación paralela?

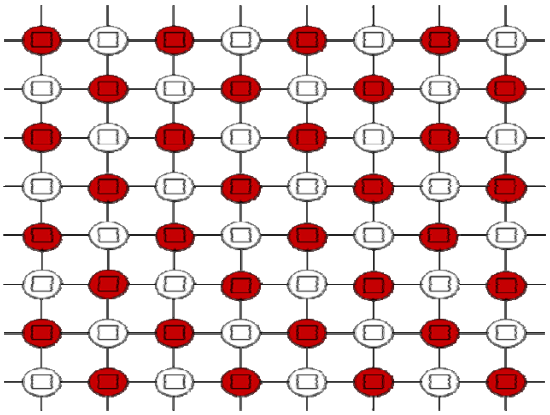
2. ¿En qué consiste la estrategia de descomposición Rojo-Blanco aplicada al OCEAN? ¿Qué característica tiene respecto a las dependencias de datos?

3. Indica **de qué orden (10, 100, 1000, 10000, >10000)** es el máximo número de procesadores que se pueden emplear en una arquitectura UMA, NUMA y un cluster de memoria distribuida. ¿Por qué no se pueden utilizar más procesadores en cada una de estas arquitecturas?

SOLUCIÓN

1. **La eficiencia se define como el speedup dividido por el número de procesadores. Una aplicación es escalable si su eficiencia se mantiene conforme aumenta el tamaño del problema y el número de procesadores.**

2. **Una estrategia Rojo-Blanco consta de dos fases: en la primera, se ejecutan los nodos con índice par y en la segunda con índice impar.**



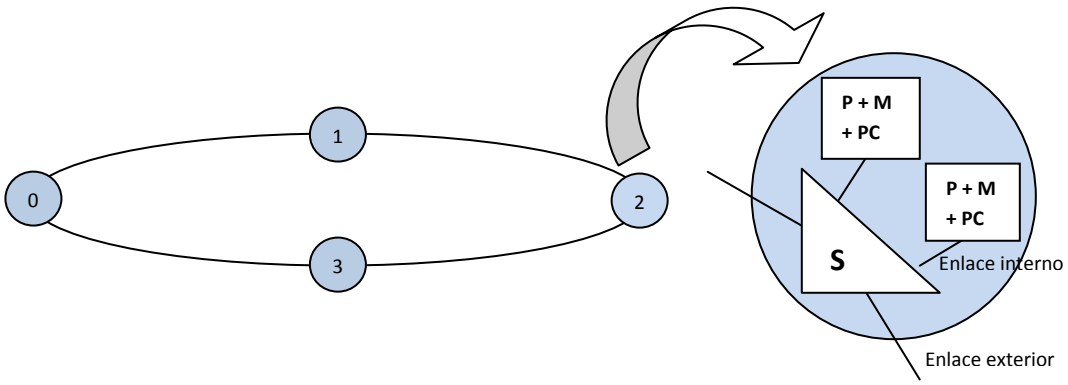
Una Ambas fases están separadas mediante y una operación tipo barrera. Esta estrategia permite obtener un resultado que es independiente del número de procesadores

utilizados, dado que en cada fase siempre se respeta el orden de dependencias de la estrategia.

3. **UMA: ~100. Limitado por el ancho de banda del bus/crossbar.**
   NUMA ~1000. Limitado por el mecanismo de coherencia/consistencia.
   Cluster ~>10000 Limitado por el coste económico y red de comunicaciones.

Ejercicio 2 (2.5 puntos)

Sea un multiprocesador de memoria distribuida con una red de interconexión con una topología de anillo tal y como se muestra en la figura. Sobre esta arquitectura se ejecuta el siguiente programa en MPI con 8 procesos en el que el proceso 0 envía un vector de 100 palabras (de 4 bytes cada palabra) a cada uno de los 7 procesos restantes. **Notar que el envío/recepción es síncrono.**




```
int x[100];
int mi_rank,i;
MPI_Status status;

MPI_Comm_rank(MPI_COMM_WORLD, &mi_rank);
if (mi_rank==0) {
      for(i=1;i<8;i++){
            MPI_Send(x, 100, MPI_INT, i, 0, MPI_COMM_WORLD);
      }
}
else{
      MPI_Recv(x, 100, MPI_INT, 0, 0, MPI_COMM_WORLD,&status);
}
```

Las características de la arquitectura son las siguientes:

- Cada nodo tiene un *switch* de comunicaciones con dos enlaces exteriores y dos internos, conectados estos últimos a dos módulos cómputo. Cada módulo tiene su procesador, memoria y procesador de comunicaciones.
- Los enlaces internos no tienen *routing delay* y tienen un ancho de banda infinito.
- Los enlaces exteriores tienen un ancho de banda es 1 Mbits/seg y un *routing delay* (retardo de encaminamiento del *switch*) de 2 ms.

- El retardo de envío y recepción del procesador de comunicaciones es de 1 ms.
- En la red de comunicaciones, no existe restricción en el tamaño de paquete (cualquier conjunto de datos puede ser enviado en un único paquete).

Se pide:

1. El tiempo de ejecución del programa empleando el protocolo de encaminamiento de **conmutación de paquetes** (*store and fordward).*
2. Calcular el tiempo de ejecución del programa empleando el protocolo de **encaminamiento de conmutación de circuitos** (*cut throught*).
3. Definir y calcular el diámetro de la red y su bisección.

SOLUCIÓN

Conmutación de paquetes:

Tamaño de paquete: 4*8*100 bits = 3200 bits.

Tiempo de transferencia=3200/1000000=3.2mseg.

Ta= Coste de envío entre nodo 0 y 0: Retardo Envío + T recepción = 1+1=2ms.

Tb=Coste de envío entre nodo 0 y 1 o 3: Retardo Envío + T enrutamiento + T transferencia + T recepción = 1 + 2 + 3.2 + 1 = 7.2ms.

Tc=Coste de envío entre nodo 0 y 2: Retardo Envío + T enrutamiento + T transferencia+ T enrutamiento + T transferencia + T recepción = 1 + 2 + 3.2 + 2 + 3.2 + 1 = 12.4ms.

El tiempo total será: Ta + 4Tb + 2Tc.

Conmutación de circuitos:

Tamaño de paquete: 4*8*10000 bits = 320000 bits.

Ta= Coste de envío entre nodo 0 y 0: Retardo Envío + T recepción = 1+1=2ms.

Tb=Coste de envío entre nodo 0 y 1 o 3: Retardo Envío + T enrutamiento + T transferencia + T recepción = 1 + 2 + 320000/1000000 + 1 = 7.2ms.

Tc=Coste de envío entre nodo 0 y 2: Retardo Envío + T enrutamiento + T transferencia+ T enrutamiento + T transferencia + T recepción = 1 + 2 + 3.2 + 2 + 1 = 9.2mseg.

El tiempo total será: Ta + 4Tb + 2Tc.

Diámetro: Distancia entre los nodos más alejados: entre el 0 y el 1.

Bisección: número de enlaces que hay que eliminar para dividir la red en dos mitades iguales: en este caso es 2.

Ejercicio 3 (2.5 puntos)

Dado el siguiente código ejecutado sobre una arquitectura de memoria compartida sobre la que ejecuta el siguiente programa empleando 8 hilos.

```
#pragma omp parallel private(tmp,i), shared(b)
{
      myrank=omp_get_thread_num()

      if(myrank==0){
           for(i=0;i<20;i++){
               b[2*i]= 16;
           }
      }
      else{
           sleep(1000);
           for(i=0;i<20;i++){
               tmp = tmp + b[2*i+1];
           }
      }
}
```

La arquitectura tiene las siguientes características:

- 8 procesadores conectados a través de un bus y un único banco de memoria. Cada procesador tiene una memoria caché local.
- Inicialmente todos los procesadores tienen una copia del vector **b** en su caché
- El tamaño de una palabra es de 4 bytes y el tamaño de bloque caché es de 16 bytes.
- El coste asociado a un fallo caché de lectura es de 4 bytes para notificar el fallo y 16 bytes para traer el dato a la memoria caché.
- Existen tres posibles implementaciones de esta arquitectura:
    1. Mediante un protocolo de invalidación **sin *read broadcast***. El tráfico asociado a la operación de invalidación es de 2 bytes.
    2. Mediante un protocolo de invalidación **con *read broadcast***. El tráfico asociado a la operación de invalidación es de 2 bytes.
    3. Mediante un protocolo de actualización. El coste de una actualización es de 16 bytes.

Se pide:

1. Calcular de forma justificada el tráfico de bus en cada uno de las tres posibles implementaciones anteriores cuando el código de ejecuta empleando 4 hilos en total. **Nótese que el hilo con myrank=0 se ejecuta y finaliza antes que el resto de los hilos.**
2. ¿Cómo mejorarías el rendimiento del programa?

SOLUCIÓN

Cada bloque contiene 4 palabras, de las cuales 2 son accedidas por el hilo rank=0 (escritura) y 2 por el resto de hilos rank>0 (lecturas).

Para cada bloque el rank=0 realiza dos escrituras y el resto de hilos realiza 2 lecturas. En total, serán 20/2=10 bloques accedidos por cada hilo.

Sin read broadcast:

Hilo rank=0. La primera escritura en cada bloque produce una invalidación. La segunda escritura se realiza inmediatamente después de la primera escritura por lo que no es necesario invalidar de nuevo (write once). En este caso, se realiza una única invalidación en cada bloque. Tráfico por cada bloque: 2B.

Tráfico total del padre=2B*210

Resto de los hilos: realizan dos lecturas en cada bloque. Se genera un único fallo, dado que el segundo acceso origina un acierto. Tráfico por cada bloque: 4B + 16B=20B

Debido a que no existe read broadcast, cada hilo que lee generada un fallo debido a la invalidación. El tráfico total será:

Tráfico asociado: 2B*10 + 20B*10*3

Con read broadcast:

Es lo mismo que en el caso anterior pero ahora, cuando el primer hilo genera un fallo, el bloque es enviado a todos los hilos, los cuales no generarán futuros fallos.

Tráfico asociado: 2B*10 + 20B*10

Actualización:

Cada escritura genera una actualización. Tráfico por cada bloque: 16B.

Tráfico asociado: 16B*20

El rendimiento se mejoraría espaciando los datos del vector b en 4 posiciones. De este modo, cada uno estaría en un bloque distinto y no habría invalidaciones.

Ejercicio 4 (2 puntos)

Dado el siguiente código ejecutado en una arquitectura de memoria compartida:

```
double A[N+2][N+2];
for (i=1;i<=N;i++){
{
   for (j=i;j<=N;j++){
         A[i][j]=(A[i-1][j]+ A[i][j]+A[i+1][j])/3;
```

```
    }
}
```

Se pide:

1. Identificar las dependencias de datos entre las distintas iteraciones.
2. Desarrolla un esquema de paralelización para memoria compartida que ignore (no respete) las dependencias de datos y que alcance alta eficiencia.
    - Describe las cuatro etapas del proceso de paralelización.
    - Escribe el código paralelo en OpenMP.
    - Indica qué estrategia de *scheduling* es la más adecuada y justifica por qué.
3. Asumiendo que el contenido del cuerpo del lazo cambia por el siguiente:

```
A[i][j]=(A[i][j-1]+A[i][j]+A[i][j+1])/3;
```

Justificar si la estrategia de paralelización del apartado anterior respeta el orden de las depèndencias.

SOLUCIÓN

1. **La iteración i depende de la i-1 e i+1.**
2. **Descomposición: una iteración del lazo exterior.**
   Asignación: en función del scheduling (dynámico).
   Orquestación: no es necesaria.
   Mapeo: un proceso a cada procesador.
3.

```
1: double A[N+2][N+2];
#omp parallel for schedule (dynamic,1)
2: for (i=1;i<=N;i++)
3:    for (j=i;j<=N;j++)
4:         A[i][j] = (A[i-1][j] + A[i][j] + A[i+1][j]) / 3;
```

   El *scheduling* dinámico es el más apropiado debido a que tiene en consideración la carga de cada proceso.
4. **No afecta a las dependencias del programa. Se respetan.**