Universidad Carlos III de Madrid

Departamento de Informática
Grado en Ingeniería Informática
Arquitectura de Computadores

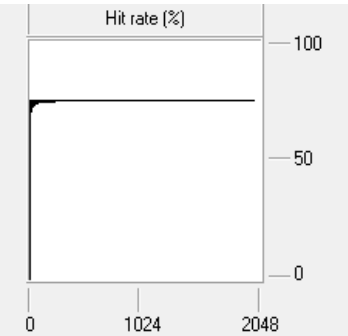
Examen extraordinario
junio de 2011

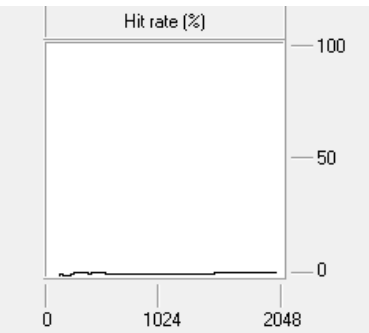
ATENCIÓN:

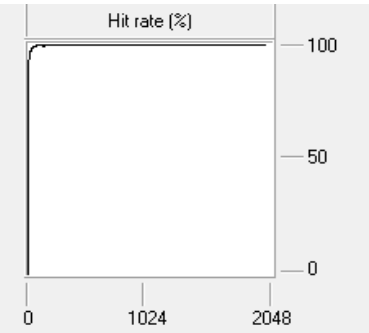
- Lea atentamente todo el enunciado antes de comenzar a contestar.
- Dispone de dos horas y treinta minutos para realizar la prueba.
- No se podrán utilizar libros ni apuntes, ni calculadoras de ningún tipo.
- Los teléfonos móviles deberán permanecer desconectados durante la prueba (apagados, no silenciados).
- Solamente se corregirán los ejercicios contestados con bolígrafo azul o negro. Por favor no utilice lápiz.

NOMBRE:
APELLIDOS:
NIA:

Ejercicio 1 [1 punto]:
Se cuenta con la traza de tres aplicaciones distintas, APP1, APP2 y APP3. Todas las aplicaciones se ejecutan en un computador con una caché de nivel 1.
Se desconoce el patrón de acceso que poseen dichas aplicaciones. Justifique razonadamente si las trazas presentan localidad o no y de qué tipo (espacial o temporal).







SOLUCION

Primer caso: localidad espacial (hay reúso de líneas caché pero no es total, es decir, también hay fallos)
Segundo caso: no hay localidad.
Tercer caso: localidad temporal: hay reúso de líneas caché sin fallos.

Ejercicio 2 [1 punto]:
Se dispone de un multicomputador basado en 8 computadores Quad core (cada uno cuenta con 4 núcleos por procesador) montado sobre una infraestructura de red de alta velocidad. Dados los siguientes argumentos que son necesarios para el despliegue y puesta en marcha de un programa en MPI:

```
mpicc –o arg0 arg1.c
mpdboot –n X –f arg2
mpiexec –n Y ./arg3
mpdallexit
```

Se pide:

1. Explique el significado de los argumentos arg0, arg1, arg2 y arg3.
2. Indique de forma justificada los valores de X e Y para obtener el mayor grado de utilización del multicomputador.

SOLUCION:

Arg0: código ejecutable del programa en mpi escrito en lenguaje C
Arg1: código fuente del programa en lenguaje C
Arg2: lista de ordenadores (nodos) sobre los que se puede ejecutar la aplicación.
Arg3: equivalente a arg0.
X: número de nodos (que no procesos) de la arquitectura
Y: número de procesos de la aplicación paralela.

Ejercicio 3 [2 puntos]:

Dado el siguiente código en ensamblador. Cada instrucción tiene un tiempo de ejecución de un ciclo y una latencia adicional (en ciclos) indicado en la figura. La arquitectura tiene una única vía segmentada (pipeline).

```
Loop:   LD      F2,0(Rx)        % latencia=2
I0:     DIVD    F2,F0,F2        % latencia=10
I1:     LD      F3,0(Ry)        % latencia=2
I2:     MULTD   F3,F0,F3        % latencia=5
I3:     ADDD    F3,F2,F3        % latencia=4
I4:     SD      F4,0(Rx)        % latencia=2
I5:     ADDI    Rx,Rx,#8        % latencia=0
I6:     ADDI    Ry,Ry,#8        % latencia=0
I7:     SUB     R20, R4, Rx     % latencia=1
I8:     BNZ     R20, Loop       % latencia=1
```

Se pide:

1. Identifique todas las dependencias de datos entre las instrucciones del programa.
2. Considérese **emisión en orden**, asumiendo que el salto de realiza. Obtenga el tiempo de ejecución teniendo en cuenta las dependencias de datos, es decir, pudiéndose emitir instrucciones sin dependencias en ciclos consecutivos.
3. Muestre el código con el bucle desenrollado en dos niveles. Obtenga el tiempo de ejecución con una **emisión fuera de orden**.

SOLUCION:

RAW: Loop – I0, I0->I3, I1->I3, I1->I2,I2->I3,I4->I5,I5->I7,I8
WAR:,I1->I2,I2->I3,I1->I3,I4->I5
WAW: LOOP->I0, I2->I3, I1->i2

LOOP: 1-3
I0: 4-14
I1: 5-7
I2: 8-13
I3: 15-19
I4: 16-18
I5: 19
I6: 20
I7: 21-22
I8: 23-24

Ejercicio 4 [2 puntos]:

Sea el siguiente fragmento de código:

```
double a[256][256], b[256][256], c[256][256], d[256][256];
…
for (int i=0;i<256;i++) {
  for (int j=0;j<256;j++) {
    a[i][j] = b[i][j] + c[i][j];
  }
}
for (int i=0;i<256;i++) {
  for (int j=0;j<256;j++) {
    d[i][j] = b[i][j] - c[i][j];
  }
}
```

Se desea ejecutar este código en un computador que tiene una caché de nivel 1 totalmente asociativa de 16KB y política de sustitución LRU con tamaño de línea de 64 bytes. Los fallos de caché de nivel 1 requieren 16 ciclos de reloj. Por otra parte la caché de nivel 2 siempre genera aciertos en este código.
Asuma que los fallos de escritura en la caché de nivel 1 se envían directamente un búfer de escritura y no generan ningún ciclo de espera.

Se pide:

a) Determine la tasa de aciertos del segmento de código, asumiendo que las variables i y j se asignan a registros del procesador y que las matrices a, b, c y d se encuentran totalmente en la caché de nivel 2.
b) Determine el tiempo acceso a memoria asumiendo que los accesos a la caché de nivel 1 requieren un ciclo de reloj.

c) Proponga una transformación de código de las que puede generar un compilador para mejorar la tasa de aciertos, mostrando el código resultante en lenguaje C.
d) Determine la nueva tasa de aciertos y el tiempo medio de acceso a memoria resultantes.

SOLUCION:

a) 64bytes/8bytes=8 doubles por línea

b) Las escrituras no originan fallos porque se escriben en el buffer de escrituras. En las dos lecturas hay un primer fallo por líneas debido a que la caché está vacía. El resto son aciertos. De este modo, para las lecturas hay 1/8 fallos y 7/8 aciertos por acceso. El número total de accesos es 2*2*256*256. El primer 2 es por tratarse de dos bucles independientes (no hay reúso de accesos en L1 por ser pequeña). El segundo 2 es por tratarse de 2 matrices de lectura en cada bucle.

c) Depreciando el coste de escritura a los valores tenemos
Tacc=1*2*2*256*256+16*2*2*256*256*1/8 (todos son aciertos en L2).
Si se considera que en la escritura se carga el valor de a en la caché el tiempo de acceso sería:
Tacc=1*2*3*256*256+16*2*3*256*256*1/8

d) Se puede aplicar la técnica de fusión de bucles.
En este caso, los datos de un bucle y otro se reutilizan en la caché, por lo que los accesos del segundo bucle originarán siempre acierto caché. El tiempo de acceso será (para el caso en que se desprecian las escrituras):

Tacc=1*2*256*256+16*2*256*256*1/8 (todos son aciertos en L2).

Ejercicio 5 [2 puntos]:

Dado el siguiente programa que emplea paso de mensajes y que se ejecuta sobre una arquitectura de memoria distribuida compuesta por cuatro nodos de cómputo. Considérese dos factores en la carga computacional:

1. Las operaciones aritméticas en punto flotante, con un coste de 2ns.
2. Las comunicaciones tienen un coste por cada palabra transmitida de:
   a. Bcast: 0,5ns
   b. Send y Recv: 1ns
   c. Reduce: 10ns
3. Las comunicaciones son síncronas (tanto **`Send`**, **`Recv`**, **`Reduce`** y **`Broadcast`** son síncronos, es decir, bloqueantes).

Se pide:

1. Calcular el tiempo de ejecución de cada uno de los cuatro procesos del programa paralelo.
2. Calcular la aceleración (asumiendo que el programa secuencial ejecuta todo el código sin las operaciones de comunicación).
3. Comente posibles mejoras que contribuyan a:
   a. El aumento del grado de paralelismo y reducción del número de comunicaciones.

b. La mejora del balanceo de carga.

4. Escriba el código paralelo resultante (optimizado) y calcule la aceleración.

```
float a[200], b[200]
Rank(&mi_rank);
Size(&mi_size);

if (mi_rank == 0) {
              Broadcast(a, 200, float, root=0);
              for (i=0;i<200;i++)
                     b[i] = i * i;
              Send(b, 100, float, to=1);
              for (i=0;i<200;i++)
                     a[i] = a[i] * a[i];
              Recv(b, 200, float, 1, from=1);
              for (i=0;i<200;i++)
                    Sum = a[i] + b[i];
              Reduce (c, a, 1, float, root=0);
} else if (mi_rank == 1) {
              Broadcast(a, 200, float, root=0);
              Recv(b, 200, float, from=0);
              for (i=0;i<200;i++)
                    b[i] = b[i] / mi_size;
             Send(b, 200, float, to=0);
             Reduce (c, a 1 , float, root=0);
} else if (mi_rank == 2) {
              Broadcast(a, 200, float, root=0);
              for (i=0;i<200;i+=10)
                     a[i] = a[i] * 4;
              Reduce (c, a, 1, float, root=0);
} else if (mi_rank == 3) {
              Broadcast(a, 200, float, root=0);
              for (i=0;i<200;i+=20)
                     a[i] = a[i] * 2;
              Reduce (c, a, 1, float, root=0);
}
```

SOLUCION

1. Coste de cada uno de los procesos

Proceso 0:

Coste de comunicación: (3 * 200 * 0,5ns)Bcast + (200 * 0,5ns)Send + (200 * 0,5ns)Recv +(1 * 10ns)Recuce = 510ns
Coste de proceso: 3 (for) * 200 * 2ns = 1200ns
Coste total: Cc + Cp = 1710ns

Proceso 1:

Coste de comunicación: (200 * 0,5ns)Bcast + (200 * 0,5ns)Recv + (200 * 0,5ns)Send +(1 * 10ns)Recuce = 310ns
Coste de proceso: 200 * 2ns = 400ns
Coste total: Cc + Cp = 910ns

Proceso 2:

Coste de comunicación: (200 * 0,5ns)Bcast + (1 * 10ns)Reduce = 110ns
Coste de proceso: (20 * 2ns) = 40ns
Coste total: Cc + Cp = 250ns

Proceso 3:

Coste de comunicación: (200 * 0,5ns)Bcast + (1 * 10ns)Reduce = 110ns
Coste de proceso: (10 * 2ns)= 20ns
Coste total: Cc + Cp = 230ns

2. Aceleración

Tsec = Cp0 + Cp1 + Cp2 + Cp3 = 1200 + 400 + 40 + 20 = 1660ns
Tparal = Tiempo del proceso más lento = 1710ns

Ace = Tsec/Tparal = 1660 / 1710 = 0,97 < 1

3. Mejoras

1. **La principal mejora consistiría en reducir el tipo de cómputo del proceso 0, y distribuir ese coste entre el resto de procesos (o al menos uno), con la idea de balancear el coste de cómputo. Es preferible usar comunicaciones porque el coste de las comunicaciones es mucho menor que el coste de operación en como flotante.**
2. **Para Aumentar el balenceo de carga se puede asignar trabajo a los procesos que están más desocupados, en este caso, el proceso 2 y 3.**

4. Problema optimizado:

```
if (mi_rank == 0) {
        Broadcast(a, 200, float, root=0);
        for (i=0;i<200;i++)
                b[i] = i * i;
        Send(b, 100, float, to=1);
        Send(a, 100, float, to=2);
        Send(a, 100, float, to=3);
```

```
                    Recv(a, 100, float, from=2);
                    Recv(a, 100, float, from=3);

                Recv(b, 100, float, 1, from=1);
                for (i=0;i<200;i++)
                        Sum = a[i] + b[i];
                Reduce (c, a, 1, float, root=0);
} else if (mi_rank == 1) {
                Broadcast(a, 200, float, root=0);
                Recv(b, 200, float, from=0);
                for (i=0;i<200;i++)
                        b[i] = b[i] / mi_size;
                  Send(b, 200, float, to=0);
                  Reduce (c, a 1 , float, root=0);
} else if (mi_rank == 2) {
                Broadcast(a, 200, float, root=0);
                for (i=0;i<200;i+=10)
                        a[i] = a[i] * 4;
               Recv(a, 100, float, from=0);
                for (i=0;i<100;i++)
                        a[i] = a[i] * a[i];
                  Send(a, 100, float, to=0);
                Reduce (c, a, 1, float, root=0);
} else if (mi_rank == 3) {
                Broadcast(a, 200, float, root=0);
                for (i=0;i<200;i+=20)
                        a[i] = a[i] * 2;
                Recv(a, 100, float, from=0);
                for (i=100;i<200;i++)
                        a[i] = a[i] * a[i];
                  Send(a, 100, float, to=0);
                Reduce (c, a, 1, float, root=0);
}
```

Proceso 0:

Coste de comunicación: (3 * 200 * 0,5ns)Bcast + (200 * 0,5ns)Send + (200 * 0,5ns)Recv +(1 * 10ns)Recuce + 2*(100 * 0,5ns)Send + 2*(100 * 0,5ns)Recv = 710ns
Coste de proceso: 1 (for) * 200 * 2ns = 400ns
Coste total: Cc + Cp = 1110ns

Proceso 1:

Coste de comunicación: (200 * 0,5ns)Bcast + (200 * 0,5ns)Recv + (200 * 0,5ns)Send +(1 * 10ns)Recuce = 310ns
Coste de proceso: 200 * 2ns = 400ns
Coste total: Cc + Cp = 910ns

Proceso 2:

Coste de comunicación: (200 * 0,5ns)Bcast + (1 * 10ns)Reduce + (100 0,5ns)Send + (100 * 0,5ns) = 210ns
Coste de proceso: (20 * 2ns) + (100 * 2ns) = 240ns
Coste total: Cc + Cp = 450ns

Proceso 3:

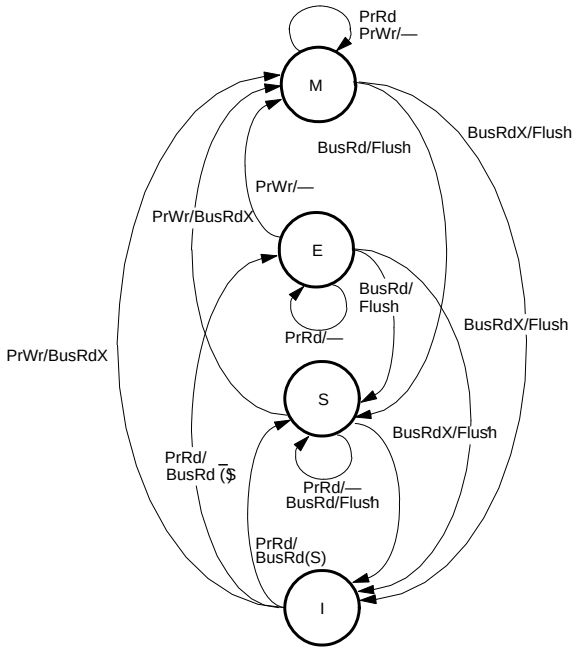
Coste de comunicación: (200 * 0,5ns)Bcast + (1 * 10ns)Reduce + 100 0,5ns)Send + (100 * 0,5ns) = 210ns
Coste de proceso: (10 * 2ns) + (100 * 2ns)= 220ns
Coste total: Cc + Cp = 430ns

Tsec = Cp0 + Cp1 + Cp2 + Cp3 = 400 + 400 + 240 + 220 = 1660ns = primera versón
Tparal = Tiempo del proceso más lento = 1110ns

Ace = Tsec/Tparal = 1660 / 1110 = 1,49

Ejercicio 6 [2 puntos]:
Un computador dispone de tres procesadores conectados por un bus a un sistema de memoria compartida. El computador utiliza el protocolo de coherencia de caché MESI.
La siguiente tabla muestra las instrucciones ejecutadas en cada procesador, que se asumen atómicas. La instrucción Print es equivalente a una lectura.

Procesador 1	Procesador 2	Procesador 3
Print x	x = 1	Print x x = 2



Se pide:

1. Explique el uso de la señal S del bus en el protocolo MESI.
2. Indique todos los posibles órdenes globales del programa y todas las posibles impresiones generadas, asumiendo que la variable x vale inicialmente 0.
3. Para el caso en que primero se ejecutan todas las instrucciones del procesador 1, después todas las del procesador 2 y finalmente todas las del procesador 3 indique todas las transiciones ligadas al protocolo MESI y todas las acciones del bus, rellenando la tabla adjunta. Asuma que inicialmente las cachés están vacías.

Instrucción	Transición P1	Transición P2	Transición P3	Acción en bus
1:				

2:				
3:				
4:				

SOLUCION:

a) La señal S indica si el bloque solicitado está o no replicado en alguna otra memoria caché. En el caso de estarlo, la transición se realiza al estado S. En caso contrario, sería al estado E.

b) Debido a que no se indica si existe consistencia secuencial se asume el caso más general por lo que se puede dar cualquier combinación instrucciones, incluyendo la ejecución fuera de orden.

c)

Instrucción	Transición P1	Transición P2	Transición P3	Acción en bus
1:	I -> E			BusRd(^S)
2:	E -> I	I->M		BusRdX Flush
3:		M->S	I->S	BusRd Flush
4:		S->I	S->M	BusRdX

d)