



2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores: Exámenes y Controles

Examen Final AC 04/09/2012 resuelto

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita, Julio Ortega

Licencia Creative Commons



1 Enunciado Examen del 04/09/2012

Cuestión 1.(0.5 puntos) Defina e indique la utilidad de las siguientes características o componentes que puede incluir un procesador e indique cuáles se implementan en superescalares, cuáles en VLIW y cuáles en ambos.

- buffer de renombramiento
- buffer de reordenamiento
- ventana de instrucciones
- estación de reserva
- predicción de instrucciones
- predicción de saltos

Cuestión 2.(0.5 puntos). Enuncie la ley de Amdahl en el contexto de procesamiento paralelo y deduzca la expresión que la caracteriza en este contexto. Defina claramente el punto de partida que utilice en la deducción y todas las etiquetas o variables que utilice.

Ejercicio 1. (2 puntos) Se dispone de un multiprocesador CC-NUMA con red estática y 4 nodos (numerados 0, 1, 2 y 3) con un modelo de consistencia que garantiza todos los órdenes menos W→R. Se ha implementado para este multiprocesador el siguiente código para obtener el número de componentes positivos de un vector de enteros:

```
numpos_local = 0;
for (i=ithread ; i< Nmax ; i=i+nthread) {
    if (a[i]>0) numpos_local=numpos_local+1;
}
while (test & set(k)) {};
numpos = numpos + numpos_local;
if (ithread==0) printf("Valores positivos en a[]:%d\n",numpos);
```

Donde k, numpos, Nmax y a[0:Nmax-1] son variables compartidas (k y numpos están inicializadas a 0), el resto son locales: nthread es igual a 4 e ithread es el identificador del thread que ejecuta el código (ithread=0,1,2,3).

Conteste a las siguientes cuestiones:

- ¿Imprime el código el resultado esperado? Rzone su respuesta. Añada lo necesario para que imprima el resultado esperado.
- Suponga que el CC-NUMA no tiene test_&_set(), pero tiene fetch_&_or(). Haga en el código las modificaciones necesarias para que imprima el resultado esperado usando esta segunda instrucción.

- (c) Suponga ahora que el CC-NUMA sólo dispone de `compare_&_swap()`. Haga en el código las modificaciones necesarias para que imprima el resultado esperado (se valorarán las prestaciones del código resultante).
- (d) Suponga ahora que el CC-NUMA dispone de `fetch_&_add()`. Haga en el código las modificaciones necesarias para que imprima el resultado esperado usando esta instrucción (se valorarán las prestaciones del código resultante).
- (e) Razoné si el código que ha escrito en el apartado (a) obtiene el resultado esperado en caso de que el procesador no garantice tampoco W->W. ¿Qué debe proporcionar el modelo de consistencia y dónde se usaría para que se pueda obtenerse el resultado esperado?

Ejercicio 2. (2 puntos) Suponga que en un CC-NUMA de red estática de 4 nodos (N0-N3) se implementa un protocolo MSI basado en directorios sin difusión con dos estados en el directorio (válido e inválido). Cada nodo tiene 16 GBytes de memoria principal y una línea de cache supone 64 Bytes. Considere que el directorio utiliza vector de bits completo. (a) Calcule el tamaño del directorio de uno nodo en bytes. (b) Indique cual sería el contenido del directorio, las transiciones de estados (en cache y en el directorio) y la secuencia de paquetes generados por el protocolo de coherencia en los siguientes accesos sobre una dirección D que se encuentra en la memoria del nodo 3 (initialmente D no está en ninguna cache, el orden de los accesos es el indicado por los números que les preceden):

1. Lectura generada por el procesador del nodo 1
2. Escritura generada por el procesador del nodo 1
3. Lectura generada por el procesador del nodo 2
4. Lectura generada por el procesador del nodo 3
5. Escritura generada por el procesador del nodo 0

Ejercicio 3. (1 puntos) Considere un procesador no segmentado con una arquitectura de tipo LOAD/STORE en la que las operaciones sólo utilizan como operandos registros de la CPU. Para un conjunto de programas representativos de su actividad se tiene que el 50% de las instrucciones son operaciones con la ALU (1 CPI), el 20% LOADs (4 CPI), el 10% STOREs (1 CPI) y el 20% BRANCHs (4 CPI).

Además, un 25% de las operaciones con la ALU utilizan operandos en registros, que no se vuelven a utilizar. ¿Se mejora o se empeoran las prestaciones si, para sustituir ese 25% de operaciones se añaden instrucciones con un dato en un registro y otro en memoria, teniendo en cuenta que para ellas el valor de CPI es 4? ¿En qué tanto por ciento empeora o mejora el tiempo de ejecución del conjunto de programas.

2 Solución Examen del 04/09/2012

Cuestión 1.(0.5 puntos) Defina e indique la utilidad de las siguientes características o componentes que puede incluir un procesador e indique cuáles se implementan en superescalares, cuáles en VLIW y cuáles en ambos.

- buffer de renombramiento
- buffer de reordenamiento
- ventana de instrucciones
- estación de reserva
- predicción de instrucciones
- predicción de saltos

Solución

- Definición: Un buffer de renombramiento es un recurso presente en los procesadores superescalares que permite asignar almacenamiento temporal a los datos asignados a registros del banco de registros de la arquitectura. La asignación a registros de la arquitectura la realiza el compilador o el programador en ensamblador. Utilidad: Mediante la asignación de registros temporales a los registros de la arquitectura se implementa el renombramiento de estos últimos con el fin de eliminar riesgos (dependencias) WAW y WAR. Dónde se implementa: superescalares.
- Definición: Un buffer de reordenamiento es un recurso presente en los procesadores superescalares que permite implementar la finalización ordenada de las instrucciones después de su ejecución. Utilidad: Esto permite implementar consistencia del procesador fuerte en la ejecución de instrucciones con registros y consistencia de memoria fuerte en la ejecución de instrucciones con acceso a memoria. Para implementar consistencia del procesador fuerte las instrucciones con registros terminan, es decir, escriben los resultados en los registros de la arquitectura, en el orden en que se encuentran en el código en memoria (código generado por el compilador o escrito por el programador). Para implementar consistencia de memoria fuerte los accesos a memoria deben terminar exactamente en el orden en que se encuentran en el código que hay en memoria. Dónde se implementa: superescalares.
- Definición: Una ventana de instrucciones es un recurso con almacenamiento al que pasan las instrucciones tras ser decodificadas para ser emitidas desde ahí a la unidad de datos donde se ejecutarán. Se emitirá una instrucción cuando estén disponibles los datos y la unidad que requiere. La emisión desde esa ventana de instrucciones puede ser ordenada o desordenada. Utilidad: Se utiliza para detectar las instrucciones que pueden pasar a ejecutarse por tener los operandos disponibles (evitando, de esta forma, problemas por riesgos RAW). Dónde se implementa: superescalares.
- Definición: Las estaciones de reserva son recursos con almacenamiento presentes en los procesadores superescalares entre los que se distribuye la ventana de instrucciones. Una estación de reserva puede estar asociada a una o a varias unidades funcionales. Las instrucciones decodificadas se envían a una u otra estación de reserva según la unidad funcional (o el tipo de unidad funcional) donde se va a ejecutar la instrucción. Utilidad: Se utiliza para detectar las instrucciones que pueden pasar a ejecutarse por tener los operandos disponibles evitando, de esta forma, problemas por riesgos RAW. Dónde se implementa: superescalares.
- Definición: La predicción de instrucciones es un recurso que se incluye para eliminar ciertas instrucciones de salto condicional de los códigos. Utilidad: contribuye a reducir el número de riesgos de control (saltos) en la ejecución de programas en procesadores segmentados, estos riesgos provocan penalizaciones en ciclos de reloj porque provoca que haya ciclos de reloj en los que no se

ejecutan instrucciones debido a que no se sabe si la instrucción que se debe ejecutar a continuación es la que sigue a la instrucción de salto (en los casos en los que no se salta) en el código o la que se encuentra en la dirección de salto (en los casos en los que se salta). Dónde se implementa: en superescalares y en VLIW

- Definición: La predicción de saltos es un recurso que predice si el salto de una instrucción de salto condicional se va a tomar o no antes de que el procesador tenga evaluada la condición de salto y, por tanto, antes de saber si se va a saltar o no. Cuando se ejecuta una instrucción de salto, en lugar de esperar a que se resuelva la condición asociada al salto, ejecuta las instrucciones de la dirección del salto o la que hay detrás del salto según el resultado de la predicción. Utilidad: se utiliza para disminuir la penalización (en ciclos de reloj) por saltos condicionales. Dónde se implementa: Se utiliza en superescalares y en la arquitectura EPIC de los Itanium de Intel que comparte características con los procesadores VLIW (algunos autores no consideran que los procesadores con predicción de saltos sean VLIW).



Cuestión 2.(0.5 puntos). Enuncie la ley de Amdahl en el contexto de procesamiento paralelo y deduzca la expresión que la caracteriza en este contexto. Defina claramente el punto de partida que utilice en la deducción y todas las etiquetas o variables que utilice.

Solución

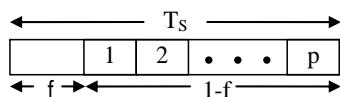
Enunciado

La ganancia en prestaciones que se puede conseguir al añadir procesadores está limitada por la fracción del tiempo de ejecución secuencial que supone la parte del código no parallelizable.

Punto de partida deducción:

Se parte de un modelo de código secuencial (como se representa en la figura de abajo):

- Con un tiempo de ejecución secuencial, T_s , que permanece constante aunque varíe el número de procesadores disponibles p
- Con una parte no paralelizable (la fracción notada por f en la figura) que permanece constante aunque varíe el número de procesadores p disponibles, y una parte $(1-f)$ que se puede paralelizar con un grado de paralelismo ilimitado y, además, repartiéndola por igual entre los p procesadores disponibles.



Las etiquetas de la figura se describen a continuación:

- T_s constante que representa el tiempo de ejecución secuencial (es independiente de p)
- f constante que representa la fracción del tiempo de ejecución secuencial del código que supone la parte no paralelizable
- $(1-f)$ es la fracción del tiempo de ejecución secuencial del código que supone la ejecución de la parte paralelizable
- p variable que representa el número de procesadores disponibles

Deducción:

La ganancia en prestaciones para este modelo de código ideal sería (suponiendo 0 el tiempo de sobrecarga):

$$S(p) = \frac{T_S}{T_P(p)} = \frac{T_S}{f \times T_S + \frac{(1-f) \times T_S}{p}} = \frac{1}{f + \frac{(1-f)}{p}} = \frac{p}{fp + (1-f)} = \frac{p}{1 + f(p-1)}$$

En la práctica la ganancia en prestaciones será menor debido a que (1) el grado de paralelismo estará limitado, (2) puede ser difícil equilibrar la carga de trabajo y (3) la paralelización puede añadir un tiempo de sobrecarga (*overhead*) debido a la necesidad de comunicación/sincronización y a las operaciones extras que puede requerir la paralelización. Teniendo en cuenta esto se utiliza menor o igual en lugar de igual en la expresión que se utiliza para representar la Ley de Amdahl:

$$S(p) \leq \frac{p}{1 + f(p-1)}$$



Ejercicio 1. (2 puntos) Se dispone de un multiprocesador CC-NUMA con red estática y 4 nodos (numerados 0, 1, 2 y 3) con un modelo de consistencia que garantiza todos los órdenes menos W→R. Se ha implementado para este multiprocesador el siguiente código para obtener el número de componentes positivos de un vector de enteros:

```
(1) numpos_local = 0;
(2) for (i=ithread ; i< Nmax ; i=i+nthread) {
(3)     if (a[i]>0) numpos_local=numpos_local+1;
}
(4) while (test_&_set(k)) {};
(5) numpos = numpos + numpos_local;
(6) if (ithread==0) printf("Valores positivos en a[]:%d\n",numpos);
```

Donde `k`, `numpos`, `Nmax` y `a[0:Nmax-1]` son variables compartidas (`k` y `numpos` están inicializadas a 0), el resto son locales: `nthread` es igual a 4 e `ithread` es el identificador del thread que ejecuta el código (`ithread=0,1,2,3`).

Conteste a las siguientes cuestiones:

- (a)** ¿Imprime el código el resultado esperado? Rzone su respuesta. Añada lo necesario para que imprima el resultado esperado.
- (b)** Suponga que el CC-NUMA no tiene `test_&_set()`, pero tiene `fetch_&_or()`. Haga en el código las modificaciones necesarias para que imprima el resultado esperado usando esta segunda instrucción.
- (c)** Suponga ahora que el CC-NUMA sólo dispone de `compare_&_swap()`. Haga en el código las modificaciones necesarias para que imprima el resultado esperado (se valorarán las prestaciones del código resultante).
- (d)** Suponga ahora que el CC-NUMA dispone de `fetch_&_add()`. Haga en el código las modificaciones necesarias para que imprima el resultado esperado usando esta instrucción (se valorarán las prestaciones del código resultante).

- (e) Razona si el código que ha escrito en el apartado (a) obtiene el resultado esperado en caso de que el procesador no garantice tampoco W->W. ¿Qué debe proporcionar el modelo de consistencia y dónde se usaría para que se pueda obtener el resultado esperado?

Solución

(a) Inicialmente la variable compartida `k` está a 0, la primitiva `test_&_set(k)` lo pone a 1. No hay ningún punto en el código donde se vuelva a poner `k` a 0. El flujo de control (*thread*) que realiza el primer acceso a `k`, es decir, el primero que ejecute `test_&_set(k)` obtiene un 0 (`test_&_set(k)` devuelve 0) y pone `k` a 1. Como obtiene un 0 no repite el bucle del `while`, continua la ejecución (el `while` se repite si `test_&_set(k)` devuelve un 1). El resto de flujos de control obtendrán al ejecutar `test_&_set(k)` un 1, por tanto, se quedarán en el `while` ejecutando `test_&_set(k)`, además nunca saldrán del `while` porque no se vuelve a poner `k` a 0 y, por tanto, a ninguno obtendrá al ejecutar `test_&_set(k)` el 0 que les permita dejar el bucle. Si el flujo de control que obtiene un 0 y continúa, por tanto, la ejecución es el 0, se imprime en pantalla los valores positivos que ha contado este flujo, en caso contrario, el código no imprime nada. En cualquier caso se quedan todos los flujos menos uno bloqueados en el `while`.

Para resolver los problemas del código, hay que poner `k` a 0 tras actualizar “`numpos = numpos + numpos_local`” (en calabaza en el código modificado) y añadir una barrera (en azul en el código modificado) para que el flujo de control 0 imprima cuando todos los flujos hayan actualizado la variable compartida `num_pos`. El código modificado es el siguiente:

```
(1) numpos_local = 0;
(2) for (i=ithread ; i< Nmax ; i=i+nthread) {
(3)     if (a[i]>0) numpos_local=numpos_local+1;
    }
(4) while (test_&_set(k)) {};
(5) numpos = numpos + numpos_local;
    k=0;
    bandera_local= !(bandera_local) //se complementa la bandera local
    while (test_&_set(k2)) {};      //lock(k2), k2 inicializada a 0
    bar[id].cont+=1; cont_local = bar[id].cont;   //cont_local es local
    k2=0;                          //unlock(k2)

    if (cont_local ==num_procesos) {
        bar[id].cont=0;           //se hace 0 el contador asociado a la barrera
        bar[id].bandera= bandera_local; // libera procesos en espera
    }
    else while (bar[id].bandera!= bandera_local) {};
(6) if (ithread==0) printf("Valores positivos en a[]:%d\n",numpos);
```

(b) Hay que realizar pocas modificaciones. Donde aparece `test_&_set(variable_cerrojo)` hay que poner `fetch_&_or(variable_cerrojo,1)`:

```
(1) numpos_local = 0;
(2) for (i=ithread ; i< Nmax ; i=i+nthread) {
(3)     if (a[i]>0) numpos_local=numpos_local+1;
    }
(4) while (fetch_&_or(k,1)) {};
(5) numpos = numpos + numpos_local;
    k=0;
    bandera_local= !(bandera_local) //se complementa la bandera local
    while (fetch_&_or(k2,1)) {};      //lock(k2), k2 inicializada a 0
    bar[id].cont+=1; cont_local = bar[id].cont;   //cont_local es local
    k2=0;                          //unlock(k2)

    if (cont_local ==num_procesos) {
        bar[id].cont=0;           //se hace 0 el contador asociado a la barrera
        bar[id].bandera= bandera_local; // libera procesos en espera
    }
```

```

else while (bar[id].bandera!= bandera_local) {};
(6) if (ithread==0) printf("Valores positivos en a[]:%d\n",numpos);

```

(c) En este caso en lugar de usar cerrojos se accederá directamente a las variables compartidas numpos y bar[id].cont con compare_&_swap(), evitando de esta forma tener que añadir (1) dos variables compartidas (variables cerrojos) más a las que acceder y (2) el código para implementar cerrojos. El código resultante es el siguiente (en calabaza se pueden ver las modificaciones con respecto al código del apartado (a)):

```

(1) numpos_local = 0;
(2) for (i=ithread ; i< Nmax ; i=i+nthread) {
(3)     if (a[i]>0) numpos_local=numpos_local+1;
    }
do
a = numpos;
b = a + numpos_local; // a y b son variables locales
compare&swap(a,b,numpos);
while (a!=b);

bandera_local= !(bandera_local) //se complementa la bandera local
do
cont_local = bar[id].cont;
b = cont_local + 1;
compare&swap(cont_local,b,bar[id].cont);
while (cont_local!=b);
if (cont_local == num_procesos-1) {
    bar[id].cont=0; //se hace 0 el contador asociado a la barrera
    bar[id].bandera= bandera_local; // libera procesos en espera
}
else while (bar[id].bandera!= bandera_local) {};
(6) if (ithread==0) printf("Valores positivos en a[]:%d\n",numpos);

```

(d) En este caso también se accederá directamente a las variables compartidas numpos y bar[id].cont, pero esta vez con Fetch_&_add(), evitando igualmente tener que añadir (1) dos variables compartidas (variables cerrojos) más a las que acceder y (2) el código para implementar cerrojos. El código resultante es el siguiente (en calabaza se pueden ver las modificaciones con respecto al código del apartado (a)):

```

(1) numpos_local = 0;
(2) for (i=ithread ; i< Nmax ; i=i+nthread) {
(3)     if (a[i]>0) numpos_local=numpos_local+1;
    }

(5) fetch_&_add(numpos, numpos_local);
bandera_local= !(bandera_local) //se complementa la bandera local
while (fetch_&_or(k2,1)) {}; //lock(k2), k2 inicializada a 0
cont_local = fetch_&_add(bar[id].cont,1); //cont_local es local
if (cont_local == num_procesos-1) {
    bar[id].cont=0; //se hace 0 el contador asociado a la barrera
    bar[id].bandera= bandera_local; // libera procesos en espera
}
else while (bar[id].bandera!= bandera_local) {};
(6) if (ithread==0) printf("Valores positivos en a[]:%d\n",numpos);

```

(e) Si no se garantiza el orden W->W el código no daría siempre el resultado esperado. Se ha añadido al código los accesos a memoria que se realizan de lectura, R, escritura, W, y lectura+escritura atómica, (R,W) en las operaciones de adquisición, liberación y en la sección crítica:

	<pre> (1) numpos_local = 0; (2) for (i=ithread ; i< Nmax ; i=i+nthread) { (3) if (a[i]>0) numpos_local=numpos_local+1; } (R,W) R, W W (R,W) R, W W </pre>
	<pre> (4) while (test_&_set(k)) {}; (5) numpos = numpos + numpos_local; k=0; bandera_local= !(bandera_local) //se complementa la bandera local while (test_&_set(k2)) {}; //lock(k2), k2 inicializada a 0 bar[id].cont+=1; cont_local = bar[id].cont; //cont_local es local k2=0; //unlock(k2) if (cont_local ==num_procesos) { bar[id].cont=0; //se hace 0 el contador asociado a la barrera bar[id].bandera= bandera_local; // libera procesos en espera } else while (bar[id].bandera!= bandera_local) {}; (6) if (ithread==0) printf("Valores positivos en a[]:%d\n",numpos); </pre>

La liberación del cerrojo permite que otro flujo de control acceda a la sección crítica. Las escrituras pueden adelantar a otras escrituras independientes que las precedan en el código porque no se garantizarse orden entre escrituras. Esto permite que las dos liberaciones de cerrojos que hay en el código (escritura en k de 0 y escritura en k2 de 0) puedan adelantar a la escritura en la variable compartida de la sección crítica que las precede. Como consecuencia, más de un flujo de control estarán en la sección crítica a la vez y podrían leer el mismo valor de la variable compartida (`numpos` en un caso y `bar[id].cont` en el otro).

Con respecto a las operaciones de adquisición, la escritura de la variable compartida no puede adelantar a la escritura del cerrojo en la adquisición porque la operación atómica que incluye la adquisición incluye tanto escritura como lectura y no se permite que las escrituras no pueden adelantar lecturas anteriores.



Ejercicio 2. (2 puntos) Suponga que en un CC-NUMA de red estática de 4 nodos (N0-N3) se implementa un protocolo MSI basado en directorios sin difusión con dos estados en el directorio (válido e inválido). Cada nodo tiene 16 GBytes de memoria principal y una línea de cache supone 64 Bytes. Considere que el directorio utiliza vector de bits completo. **(a)** Calcule el tamaño del directorio de un nodo en bytes. **(b)** Indique cual sería el contenido del directorio, las transiciones de estados (en cache y en el directorio) y la secuencia de paquetes generados por el protocolo de coherencia en los siguientes accesos sobre una dirección D que se encuentra en la memoria del nodo 3 (initialmente D no está en ninguna cache, el orden de los accesos es el indicado por los números que les preceden):

1. Lectura generada por el procesador del nodo 1
2. Escritura generada por el procesador del nodo 1
3. Lectura generada por el procesador del nodo 2
4. Lectura generada por el procesador del nodo 3
5. Escritura generada por el procesador del nodo 0

Solución

Datos del ejercicio

- Tamaño Memoria principal por Nodo: 16 GB = TMN
- Tamaño Línea de Cache (bloque de memoria): 64 B = TLC
- 1 bits de estado por bloque en el directorio ya que hay que codificar dos estados (válido, inválido).
- Se accede a una dirección de la memoria del nodo 3 cuyo bloque no se encuentra en ninguna cache.

Como no está en ninguna cache debe estar actualizado en memoria principal; es decir, el estado en el directorio para el bloque es válido.

(a)

$$\text{Tamaño_por_nodo} = \frac{TMN}{TLC} \times (1b + 4b) = \frac{2^{34}B}{2^6B} \times 5b = 2^{28} \times 5b = 2^{20} \times \frac{2^8 \times 5b}{2^3 b/B} = (2^{20} \times 2^5 \times 5)B = 160MB$$

(b)

Intervienen los nodos N0, N1, N2 y N3. V es Válido e I inválido. BD denota el bloque de la dirección D.

ESTADO INICIAL	EVENTO	ACCIÓN	ESTADO SIGUIENTE										
N0) Inválido N1) Inválido N2) Inválido N3) Inválido D) Local <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>V</td><td></td><td></td><td></td><td></td></tr> </table>	V					P1 lee D	1. N1 envía petición de lectura del bloque BD (PtLec(BD)) a N3 2. N3 recibe PtLec(BD). Como tiene el bloque BD en estado Válido, (1) envía paquete de respuesta con el bloque a N1 (RpBloque(BD)) y (2) pone el bit de N1 en el directorio a 1. 3. N1 recibe RpBloque(BD) y pone el bloque en cache en estado Compartido	N0) Inválido N1) Compartido N2) Inválido N3) Inválido D) Válido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>V</td><td></td><td>1</td><td></td><td></td></tr> </table>	V		1		
V													
V		1											
N0) Inválido N1) Compartido N2) Inválido N3) Inválido D) Válido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>V</td><td>1</td><td></td><td></td></tr> </table>	V	1			P1 escribe en D	1. N1 envía petición de acceso exclusivo para BD (PtEx(BD)) a N3. 2. N3, cuando recibe PtEx(BD), (1) pasa BD a estado Inválido y (2) envía paquete de respuesta a N1 confirmando invalidación (RpInv(BD)). 3. N1, recibida la respuesta, modifica el bloque y lo pasa a estado Modificado.	N0) Inválido N1) Modificado N2) Inválido N3) Inválido D) Inválido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>I</td><td>1</td><td></td><td></td></tr> </table>	I	1				
V	1												
I	1												
N0) Inválido N1) Modificado N2) Inválido N3) Inválido D) Inválido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>I</td><td>1</td><td></td><td></td></tr> </table>	I	1			P2 lee D	1. N2 envía PtLec(BD) a N3 porque no tiene BD. 2. N3, como tiene BD en estado Inválido: (1) reenvía (RvPetLec(BD)) la petición al nodo N1 (que según el directorio tiene copia válida del bloque), y (2) pone en la entrada del bloque en el directorio estado pendiente de Válido y activa el bit de N2. 3. N1 recibe RvPetLec(BD) y: (1) envía a N3 un paquete de respuesta con el bloque (RpBloque(BD)), y (2) pasa BD en su cache a Compartido. 4. N3 recibe la respuesta de N1 (RpBloque(BD)) y: (1) responde con el bloque a N2 (RpBloque(BD)) y (2) activa el bit de N2 y pone el estado del bloque en el directorio a Válido 5. N2, cuando recibe RpBloque(BD), introduce el bloque en su cache en estado Compartido	N0) Inválido N1) Compartido N2) Compartido N3) Inválido D) Válido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>V</td><td>1</td><td>1</td><td></td></tr> </table>	V	1	1			
I	1												
V	1	1											
N0) Inválido N1) Compartido N2) Compartido N3) Inválido D) Válido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>V</td><td>1</td><td>1</td><td></td></tr> </table>	V	1	1		P3 lee D	N3 lee BD de su propia memoria, lo mete en su cache en estado Compartido y activa el bit de copia de N3 en el directorio	N0) Inválido N1) Compartido N2) Compartido N3) Compartido D) Válido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>V</td><td>1</td><td>1</td><td>1</td></tr> </table>	V	1	1	1		
V	1	1											
V	1	1	1										
N0) Inválido N1) Compartido N2) Compartido N3) Compartido D) Válido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>V</td><td>1</td><td>1</td><td>1</td></tr> </table>	V	1	1	1	P0 escribe en D	1. N0 envía a N3 petición de lectura de BD con acceso exclusivo PtLecEx(BD) 2. N3 recibe PtLecEx(BD) y, como tiene el bloque en estado Válido: (1) envía los paquetes RvInv(BD) a los nodos que, según el directorio, tienen copia del bloque, (2) pone en el directorio el estado de BD en pendiente de Inválido.	N0) Modificado N1) Inválido N2) Inválido N3) Inválido D) Inválido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>I</td><td>1</td><td></td><td></td></tr> </table>	I	1				
V	1	1	1										
I	1												

		<p>3.N1, N2 y N3, cuando reciben RvInv(BD): (1) invalidan su copia de BD y (2) responden a N3 confirmando la invalidación RplInv(BD).</p> <p>4.N3, cuando recibe todas las RplInv(BD) (espera a todas las invalidaciones para garantizar un orden en los accesos a BD y así garantizar coherencia): (1) envía respuesta con el bloque a NO confirmando invalidación RpBloqueInv y (2) activa el bit de copia de NO en el directorio y pone el estado de BD en el directorio a Inválido</p> <p>5.NO, cuando recibe RpBloqueInv(BD) de N3, introduce el bloque en su cache, escribe en la copia de BD de su cache y lo pone en estado Modificado en el directorio cache</p>
--	--	---



Ejercicio 3. (1 punto) Considere un procesador no segmentado con una arquitectura de tipo LOAD/STORE en la que las operaciones sólo utilizan como operandos registros de la CPU. Para un conjunto de programas representativos de su actividad se tiene que el 50% de las instrucciones son operaciones con la ALU (1 CPI), el 20% LOADs (4 CPI), el 10% STOREs (1 CPI) y el 20% BRANCHs (4 CPI).

Además, un 25% de las operaciones con la ALU utilizan operandos en registros, que no se vuelven a utilizar. ¿Se mejoran o se empeoran las prestaciones si, para sustituir ese 25% de operaciones se añaden instrucciones con un dato en un registro y otro en memoria, teniendo en cuenta que para ellas el valor de CPI es 4? ¿En qué tanto por ciento mejora o empeora el tiempo de ejecución del conjunto de programas.

Solución

Datos del ejercicio:

Alternativa 1 (i: tipo de instrucción en la alternativa 1, puede ser load, store, alu y br (instrucción de salto); CPI_i¹: Ciclos por Instrucción para instrucciones del tipo i en la alternativa 1; NI_i¹: Número de Instrucciones del tipo i en la alternativa 1; NI¹: Número de Instrucciones en total en la alternativa 1):

i=	CPI _i ¹	Fracción NI _i ¹ /NI ¹	NI _i ¹	Comentarios
LOAD	4	0,2	0,2*NI ¹	
STORE	1	0,1	0,1*NI ¹	
ALU	1	0,5	0,5*NI ¹	25 % inst. que usan operados en registros que no se vuelven a utilizar
BR	4	0,2	0,2*NI ¹	
TOTAL		1	NI¹	

Alternativa 2 (i: tipo de instrucción en la alternativa 2, puede ser load, store, alu r-r (operación alu con dos operandos en registro), alu r-m (operación alu con un operando en registro y otro en memoria) y br (instrucción de salto); CPI_i²: Ciclos por Instrucción para instrucciones del tipo i en la alternativa 2; NI_i²: Número de Instrucciones del tipo i en la alternativa 2; NI¹: Número de Instrucciones en total en la alternativa 1; NI²: Número de Instrucciones en total en la alternativa 2):

i=	CPI _i ²	NI _i ² /NI ¹	NI _i ²	Comentarios
LOAD	4	0,2 - 0,25 * 0,5 = 0,2-0,125= 0,075	0,2*NI ¹ - 0,25*NI ¹ *0,5 = 0,075*NI ¹	El 25 % de las instrucciones con la ALU de la alt. 1 operan con un dato en memoria que no hay que cargar con load en un registro en la alt. 2; por tanto, disminuyen las instrucciones load en la alt. 2
STORE	1	0,1	0,1*NI ¹	

ALU r-r	1	0,75 * 0,5 = 0,375	0,75*NI ¹ * 0,5 = 0,375*NI ¹	75% de las instrucciones con la ALU de la alt. 1 siguen estando en la alt. 2
ALU r-m	4	0,25 * 0,5 = 0,125	0,25*NI¹ * 0,5 = 0,125*NI¹	25% de las instrucciones con la ALU de la alt. 1, es decir, el 25% del 50% del total de instrucciones de la alt. 1 son, en la alt. 2, instrucciones con un operando en memoria
BR	4	0,2	0,2*NI ¹	
TOTAL		0,875	NI²=0,875*NI¹	

En primer lugar se calcula el tiempo de CPU para la situación inicial (utilizando el número de instrucciones de cada tipo en la alternativa 1, NI_i¹, que aparece en la cuarta columna de la primera tabla):

$$\begin{aligned} T_{CPU}(1) &= (0,2*NI^1*4+0,1*NI^1*1+0,5*NI^1*1+0,2*NI^1*4) * T_{ciclo} = \\ &(0,8+0,1+0,5+0,8) * NI^1 * T_{ciclo} = 2,2 * NI^1 * T_{ciclo} \end{aligned}$$

En segundo lugar se calcula el tiempo de CPU para la alternativa 2 (utilizando el número de instrucciones de cada tipo en la alternativa 2, NI_i², que aparece en la cuarta columna de la segunda tabla):

$$\begin{aligned} T_{CPU}(2) &= (0,075*NI^1*4+0,1*NI^1*1+0,375*NI^1*1+0,125*NI^1*4+0,2*NI^1*4) * T_{ciclo} = \\ &(0,3+0,1+0,375+0,5+0,8) * NI^1 * T_{ciclo} = 2,075 * NI^1 * T_{ciclo} \end{aligned}$$

Los dos tiempos obtenidos se pueden comparar porque están en función de las mismas variables. Las prestaciones mejoran, puesto que el tiempo se reduce con la alternativa 2. El tiempo se ha reducido un

$$[2,2*NI^1*T_{ciclo} - 2,075*NI^1*T_{ciclo}] * 100 / 2,2*NI^1*T_{ciclo} = 0,125 * 100 / 2,2 \sim 5,68\%$$



2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores: Exámenes y Controles

Examen Final AC 20/06/2012 resuelto

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita, Julio Ortega

Licencia Creative Commons 

1 Enunciado Examen del 20/06/2012

Cuestión 1. (0.5 puntos) ¿Cuál de los siguientes modelos de consistencia permite conseguir mejores tiempos de ejecución? Justifique su respuesta.

- (a) modelo de ordenación débil
- (b) modelo implementado en los procesadores de la línea x86
- (c) modelo de consistencia secuencial

Cuestión 2. (0.5 puntos) Suponga un CC-NUMA con protocolo MSI basado en directorios distribuidos. Suponiendo que un bloque se encuentra en el directorio en estado inválido indique cual sería el contenido del directorio, las transiciones de estados (en cache y en el directorio) y la secuencia de paquetes generados por el protocolo de coherencia si un procesador que no tiene el bloque en su cache escribe en una dirección de dicho bloque. Razona su respuesta.

Ejercicio 1. (1 punto) En un procesador sin segmentación de cauce, determine cuál de estas dos alternativas para realizar un salto condicional es mejor:

- ALT1: Una instrucción COMPARE actualiza un código de condición y es seguida por una instrucción BRANCH que comprueba esa condición.
- ALT2: Una sola instrucción incluye la funcionalidad de las instrucciones COMPARE y BRANCH.

Hay que tener en cuenta que hay un 25% de saltos condicionales en ALT1; que las instrucciones BRANCH en ALT1 y COMPARE+BRANCH en ALT2 necesitan 6 ciclos mientras que todas las demás necesitan sólo 2; y que el ciclo de reloj de la ALT1 es un 80% del ciclo de ALT2, dado que con esta última alternativa la mayor funcionalidad de la instrucción COMPARE+BRANCH ocasiona una mayor complejidad en el procesador.

Ejercicio 2. (0.5 puntos) Implemente un cerrojo simple para un procesador que no garantiza el orden W->R usando Fetch_and_Or.

Ejercicio 3. (1.5 puntos) Considere el bucle:

```
do {  
    b[i]=a[i]*c;  
    c=c+1;  
    if (c>10) then goto etiqueta;  
    while (i<10);  
etiqueta:.....
```

Indique cuál es la penalización efectiva debida a los saltos para c=5 considerando que el procesador utiliza:

- (a) Predicción fija (siempre se considera que se va a producir el salto)

- (b) Predicción estática (si el desplazamiento es negativo se toma y si es positivo no)
- (c) Predicción dinámica con dos bits usando el diagrama de estados visto en clase.

Nota: La penalización por saltos incorrectamente predichos es de 4 ciclos y para los saltos correctamente predichos es 0 ciclos.

Ejercicio 4. (2 puntos) Se quiere realizar un programa paralelo con paso de mensajes que obtenga el valor máximo de los componentes de un vector, $v[]$, con N elementos en coma flotante.

(a) Escriba un programa secuencial con notación algorítmica (podría escribirlo en C) para obtener el máximo. ¿Cuántas operaciones de comparación realiza el código en total?

(b) Escriba ahora un programa paralelo con notación algorítmica (podría escribirlo en C) teniendo en cuenta que: (1) El proceso 0 tiene inicialmente el vector v en su memoria e imprimirá en pantalla el resultado. (2) La herramienta de programación proporciona las funciones `send()`/`receive()` que permiten la comunicación uno-a-uno asíncrona:

```
send(buffer, count, datatype, idproc, group) no bloqueante y  
receive(buffer, count, datatype, idproc, group) bloqueante,
```

donde `group` es el identificador del grupo de procesos que intervienen en la comunicación; `idproc` es el identificador del proceso al que se envía o del que se recibe; `buffer` es la dirección a partir de la que se almacenan los datos que se envían o reciben; `datatype` es el tipo de datos a recibir o enviar; y `count` es el número de datos a transferir del tipo `datatype`.

(c) Dibuje (1) el grafo de dependencias de tareas para este ejemplo a partir del código secuencial y (2) la estructura (grafo) de los procesos para su código paralelo. ¿Con qué tipo de estructuras de las estudiadas en AC se corresponde la estructura de procesos? Razoné su respuesta.

(d) ¿Cuántas operaciones de comparación realiza cada proceso (procesador/core) si el número de procesos p divide a N ? Razoné su respuesta.

(e) Estime el tiempo de ejecución del código secuencial T_s considerando que coincide con el tiempo de cálculo que suponen las operaciones de comparación (llame t al tiempo que supone una comparación).

(f) Estime la ganancia en prestaciones conseguida con la parallelización para un número de procesadores/cores de 10 y $N=10^6$ (1) usando para estimar el tiempo de cálculo paralelo T_c sólo el tiempo que suponen las operaciones de comparación (llame t al tiempo que supone una comparación), (2) usando para estimar el tiempo de sobrecarga (*overhead*) T_o sólo el tiempo de comunicación $T_{c/s}$, (3) considerando el tiempo de trasferencia de un mensaje (comunicación entre procesos) igual a t_1+nt_2 ($t_1=10000t$ y $t_2=2t$ son constantes y n es el número de flotantes del mensaje), (4) considerando que las trasferencias uno-a-uno se realizan en paralelo en la arquitectura sin conflicto entre trasferencias distintas aunque su origen y/o su destino sean el mismo, y (5) considerando que cada proceso se asigna a un procesador/core distinto.

2 Solución Examen del 20/06/2012

Cuestión 1.(0.5 puntos) ¿Cuál de los siguientes modelos de consistencia permite conseguir mejores tiempos de ejecución? Justifique su respuesta.

- a) modelo de ordenación débil
- b) modelo implementado en los procesadores de la línea x86
- c) modelo de consistencia secuencial

Solución

El modelo de consistencia débil relaja todos los órdenes entre operaciones de acceso a memoria independientes (W->R, W->W, R->W,R), el modelo de consistencia secuencial no relaja ningún orden y el de los procesadores x86 sólo relaja en operaciones escalares el orden W->R.

El mejor es el modelo de consistencia débil (la opción (a)) porque, al relajar todos los órdenes en los accesos a memoria, permite mejores tiempos de ejecución que el resto ya que ningún acceso tiene que esperar a que termine otro anterior en el código siempre que los dos sean accesos a distintas posiciones de memoria.



Cuestión 2.(0.5 puntos) Suponga un CC-NUMA con protocolo MSI basado en directorios distribuidos. Suponga un bloque B que se encuentra en memoria principal en estado inválido, indique **(a)** cual sería el contenido de la entrada del directorio para ese bloque en esta situación y **(b)** las transiciones de estados (en cache y en el directorio), la secuencia de paquetes generados por el protocolo de coherencia y el contenido al que pasa la entrada del directorio para ese bloque si un procesador que no tiene el bloque en su cache y no está en el nodo origen del bloque escribe en una dirección de dicho bloque. Razone su respuesta.

Solución

(a) Si el bloque está inválido en memoria principal habrá una copia del bloque válida en un única cache. El contenido en la entrada del bloque del directorio en esa situación tendrá en inválido (a 0) el bit de estado del bloque en memoria y, en su vector de bits de presencia, tendrá un único bit activo, el resto estarán a 0. El bit activo será el que corresponda a la cache que tiene la única copia válida del bloque en el sistema.

Vector de bits de presencia						Estado
0	...	0	1	0	0	

(b) Si un procesador de un nodo que no es el origen del bloque escribe en una dirección del bloque:

1. Se produce un fallo de escritura que provoca la emisión desde este nodo del procesador que escribe (o nodo solicitante S) al nodo origen del bloque (o nodo que tiene el bloque en un módulo de memoria principal) de un paquete de petición de lectura con acceso exclusivo al bloque PtLecEx(B).

2. El nodo origen (O) al recibir este paquete, como tiene el bloque inválido, re-envía la petición al nodo propietario (P) del bloque RvLecEx(B). Obtiene el nodo propietario del directorio, el bit activo de la entrada del directorio para el bloque identifica al único nodo propietario.

3. El nodo propietario (P), al recibir la petición de lectura y acceso exclusivo a B, RvLecEx(B), invalida la copia que tiene del bloque (porque pide acceso exclusivo) y responde con el bloque (porque pide lectura) al origen confirmando acceso exclusivo, es decir, confirma la invalidación del bloque en su cache. Por tanto, envía al origen un paquete de respuesta con el bloque confirmando la invalidación de la copia del bloque en su cache, RpBloqueInv(B).

4. El nodo origen (O), cuando recibe la respuesta la re-envía al nodo solicitante RpBloqueInv (B). Además mantiene el estado del bloque en inválido, porque lo va a modificar el nodo solicitante, y en el vector de bit

de presencia activará el bit que se corresponde con el nodo solicitante (que es donde estará la única copia válida del bloque), el resto de bits de presencia estarán a 0.

5. El nodo solicitante (S), cuando recibe el paquete con el bloque, RpBloqueInv (B), lo introduce en su cache, lo modifica y pone estado modificado en la entrada del bloque en el directorio de su cache (no confundir con el directorio de memoria principal).



Ejercicio 5. (1 punto) En un procesador sin segmentación de cauce, determine cuál de estas dos alternativas para realizar un salto condicional es mejor:

- ALT1: Una instrucción COMPARE actualiza un código de condición y es seguida por una instrucción BRANCH que comprueba esa condición.
- ALT2: Una sola instrucción incluye la funcionalidad de las instrucciones COMPARE y BRANCH.

Hay que tener en cuenta que hay un 25% de saltos condicionales en ALT1; que las instrucciones BRANCH en ALT1 y COMPARE+BRANCH en ALT2 necesitan 6 ciclos mientras que todas las demás necesitan sólo 2; y que el ciclo de reloj de la ALT1 es un 80% del ciclo de ALT2, dado que en con esta última alternativa la mayor funcionalidad de la instrucción COMPARE+BRANCH ocasiona una mayor complejidad en el procesador.

Solución

Datos del ejercicio (considerando 25% de instrucciones BRANCH en ALT1 para el conjunto de programas de prueba):

$$\text{Tiempo de ciclo de ALT1} = 0,8 * \text{Tiempo de ciclo de ALT2} \quad (T_{\text{ciclo}}^1 = 0,8 * T_{\text{ciclo}}^2)$$

Datos para ALT1 (considerando 25% de BRANCH en ALT1; NI_i: nº de instrucciones del tipo i en el conjunto de programas de prueba; NI¹: nº de instrucciones en total en el conjunto de programas de prueba en ALT1):

Instrucciones i	ciclos	NI _i /NI ¹	Proporción (%)	Comentarios
BRANCH en ALT1	6	0,25	25	
RESTO en ALT1	2	0,75	75	Incluye instrucciones COMPARE
TOTAL		1	100	

Datos para ALT2 (considerando 25% de instrucciones BRANCH en ALT1; NI²: número de instrucciones en total en el conjunto de programas de prueba en ALT2):

Instrucciones i	ciclos	NI _i /NI ¹	NI _i /NI ²	Comentarios
COMPARE+BRANCH en ALT2	6	0,25	25/75 = 1/3	
RESTO en ALT2	2	0,75-0,25=0,5	(75-25)/75 = 2/3	Se quitan las instrucciones COMPARE que se han añadido a BRANCH, que son un 25% de NI ¹
TOTAL		0,75	1	NI² = 0,75 * NI¹ (al no tener ALT2 instr. COMPARE aparte)

El tiempo de CPU se puede expresar como:

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Prof. responsables: M. Anguita, J. Ortega



5 / 10

$$T_{CPU} = CPI * NI * T_{CICLO}$$

donde CPI es el número medio de ciclos por instrucción, NI es el número de instrucciones en total del conjunto de programas de prueba, y T_{CICLO} el tiempo de ciclo del procesador.

Para la alternativa ALT1 se tiene:

$$T_{CPU}^1 = CPI^1 * NI^1 * T_{ciclo}^1 = (0.25*6+0.75*2)*NI^1*T_{ciclo}^1 = 3* NI^1*0,8*T_{ciclo}^2 = 2,4 * NI^1 * T_{ciclo}^2$$

Para la alternativa ALT2 se tiene:

$$CPI^2 = (0.25*NI^1*6+0.5*NI^1*2)/(0,75*NI^1) = (0.25*6+0.5*2)/0,75$$

$$T_{CPU}^2 = CPI^2 * NI^2 * T_{ciclo}^2 = [(0.25*6+0.5*2)/0,75] * 0,75 * NI^1 * T_{ciclo}^2 = (0.25*6+0.5*2)*NI^1 * T_{ciclo}^2 = 2,5 * NI^1 * T_{ciclo}^2$$

Los dos tiempos se han calculado en función de las mismas variables. Como se puede ver es menor el tiempo de la alternativa 1: $T_{CPU}^1 < T_{CPU}^2$. Es decir, aunque un repertorio con instrucciones más complejas puede reducir el número de instrucciones de los programas, disminuyendo el número de instrucciones a ejecutar, esto no tiene que suponer una mejora de prestaciones, si al implementar esta opción la mayor complejidad del procesador lo hace algo más lento.



Ejercicio 6. (0.5 puntos) Implemente usando `Fetch_and_Or` un cerrojo simple para un procesador que no garantiza el orden W->R.

Solución

```
Lock(k): while (Fetch_and_Or (k,1)==0) {};
```

Aclaración: el modelo de consistencia garantiza que ningún acceso de lectura y escritura a las variables compartidas (se accede a las variables compartidas después del `lock`) adelantará la escritura del cerrojo `k` porque esta escritura acompaña a una lectura en una operación de lectura-modificación-escritura atómica (`Fetch_and_Or` es atómica) y el modelo de consistencia no permite que se adelante a lecturas anteriores en el código.

```
Unlock (k) : k=0;
```

Aclaración: el modelo de consistencia garantiza que la escritura del cerrojo `k` no adelanta a ningún acceso anterior en el código (se accede a las variables compartidas antes del `unlock`).



Ejercicio 7. (1.5 puntos) Considere el bucle:

```
i=0;  
do {  
    b[i]=a[i]*c;  
    c=c+1;  
    if (c>10) then goto etiqueta;  
    i=i+1;  
    while (i<10);  
etiqueta:.....
```

Indique cuál es la penalización efectiva debida a los saltos para un valor inicial de `c` igual a 5 considerando que el procesador utiliza:

- (a) Predicción fija (siempre se considera que se va a producir el salto)
- (b) Predicción estática (si el desplazamiento es negativo se toma y si es positivo no)



Examen final del 20/06/2012 resuelto



ETSIIT



(c) Predicción dinámica con dos bits usando el diagrama de estados visto en clase (considerar que el estado inicial es 11 para todos los saltos).

Nota: La penalización por saltos incorrectamente predichos es de 4 ciclos y para los saltos correctamente predichos es 0 ciclos.

Solución

Datos del ejercicio:

Hay dos instrucciones de salto. Identificaremos con 1 a la instrucción de salto que hay dentro del bucle (salto condicional hacia delante) e identificaremos con 2 a la que controla el final de las iteraciones del bucle (salto condicional hacia atrás). El bucle se repite mientras i es menor que 10 y mientras c no supere 10. Las dos variables i y c se incrementan en 1 cada iteración. Para $c=5$ la primera condición de salida del bucle que se cumple es la correspondiente al salto que hay dentro del bucle porque c llega a 11 antes de que i llegue a 10. Así que, para $c=5$, la secuencia de Saltos y No Saltos de las instrucciones 1 y 2 es $(N_1 S_2)^5 S_1$ (N significa que no se salta y S que se salta, $()^5$ significa que se repite lo que hay dentro del paréntesis 5 veces). Por tanto, para la instrucción 1 tenemos $N_1 N_1 N_1 N_1 N_1 S_1$ y para la instrucción 2 $S_2 S_2 S_2 S_2 S_2$. Esta ejecución de los dos saltos queda reflejada en la siguiente tabla:

Iteración del bucle	1	2	3	4	5	6
i al final de la iteración del bucle (valor inicial 0)	1	2	3	4	5	6
c al final de la iteración del bucle (valor inicial 5)	6	7	8	9	10	11
Salto del if Ejecución	N	N	N	N	N	S
Salto del while Ejecución	S	S	S	S	S	S

(a) Predicción fija (siempre se salta): hay 5 penalizaciones para la instrucción 1 (20 ciclos), como ilustra la siguiente tabla (P denota penalización):

Iteración del bucle	1	2	3	4	5	6	Total ciclos penalización
Predicción salto if y salto while	S	S	S	S	S	S	
Salto del if Ejecución	N	N	N	N	N	N	
Salto del if Penalización	P	P	P	P	P		$5 * 4 \text{ ciclos} = 20 \text{ ciclos}$
Salto del while Ejecución	S	S	S	S	S	S	
Salto del while Penalización							$0 * 4 \text{ ciclos} = 0 \text{ ciclos}$

(b) Predicción estática: hay una penalización para la instrucción 1 (4 ciclos) como ilustra la siguiente tabla:

Iteración del bucle	1	2	3	4	5	6	Total ciclos penalización
Predicción salto if	N	N	N	N	N	N	
Salto del if Ejecución	N	N	N	N	N	N	
Salto del if Penalización							$P \quad 1 * 4 \text{ ciclos} = 4 \text{ ciclos}$
Predicción salto while	S	S	S	S	S	S	
Salto del while Ejecución	S	S	S	S	S	S	
Salto del while Penalización							$0 * 4 \text{ ciclos} = 0 \text{ ciclos}$

(c) Predicción dinámica: para la instrucción 1 hay tres penalizaciones (los dos primeros N_1 , N_1 y el último S_1), por lo que hay 15 ciclos de penalización. Para la instrucción 2 no hay penalizaciones en este caso. La siguiente tabla ilustra las afirmaciones anteriores:

Iteración del bucle	1	2	3	4	5	6	Total ciclos penalización
Estado para if	1	1	0	0	0	0	
	1	0	1	0	0	0	
Predicción salto if	S	S	N	N	N	N	
Salto del if Ejecución	N	N	N	N	N	N	S
Salto del if Penalización	P	P			P		$3 * 4 \text{ ciclos} = 12 \text{ ciclos}$
Estado para while	1	1	1	1	1	1	
	1	1	1	1	1	1	
Predicción salto while	S	S	S	S	S	S	
Salto del while Ejecución	S	S	S	S	S	S	
Salto del while Penalización							$0 * 4 \text{ ciclos} = 0 \text{ ciclos}$



Ejercicio 8. (2 puntos) Se quiere realizar un programa paralelo con paso de mensajes que obtenga el valor máximo de los componentes de un vector, $V[]$, con N elementos en coma flotante positivos.

- (a) Escriba un programa secuencial con notación algorítmica (podría escribirlo en C) para obtener el máximo. ¿Cuántas operaciones de comparación realiza el código en total?
- (b) Escriba ahora un programa paralelo con notación algorítmica (podría escribirlo en C) teniendo en cuenta que: (1) el proceso 0 tiene inicialmente el vector V en su memoria e imprimirá en pantalla el resultado; (2) el número de procesos (procesadores/cores) p divide a N ; (3) la herramienta de programación proporciona las funciones `send()`/`receive()` que permiten la comunicación uno-a-uno asíncrona:


```
send(buffer, count, datatype, idproc, group) no bloqueante y
receive(buffer, count, datatype, idproc, group) bloqueante,
```

 donde `group` es el identificador del grupo de procesos que intervienen en la comunicación; `idproc` es el identificador del proceso al que se envía o del que se recibe; `buffer` es la dirección a partir de la que se almacenan los datos que se envían o reciben; `datatype` es el tipo de datos a recibir o enviar; y `count` es el número de datos a transferir del tipo `datatype`.
- (c) Dibuje (1) el grafo de dependencias de tareas para este ejemplo (parte del código secuencial) y (2) la estructura (grafo) de los procesos para su código paralelo. ¿Con qué tipo de estructuras de las estudiadas en AC se corresponde la estructura de procesos? Razoné su respuesta.
- (d) ¿Cuántas operaciones de comparación realiza cada proceso (procesador/core) si el número de procesos p divide a N ? Razoné sus respuestas.
- (e) Estime el tiempo de ejecución del código secuencial T_s considerando que coincide con el tiempo de cálculo que suponen las operaciones de comparación (llame t al tiempo que supone una comparación).
- (f) Estime la ganancia en prestaciones conseguida con la parallelización para un número de procesadores/cores de 10 y $N=10^6$ (1) usando para estimar el tiempo de cálculo paralelo T_c sólo el tiempo

que suponen las operaciones de comparación (llame t al tiempo que supone una comparación), (2) usando para estimar el tiempo de sobrecarga (*overhead*) T_o sólo el tiempo de comunicación $T_{c/s}$, (3) considerando el tiempo de trasferencia de un mensaje (comunicación entre procesos) igual a t_1+nt_2 ($t_1=1000t$ y $t_2=2t$ son constantes y n es el número de flotantes del mensaje), (4) considerando que las trasferencias uno-a-uno se realizan en paralelo en la arquitectura sin conflicto entre trasferencias distintas aunque su origen y/o su destino sean el mismo, y (5) considerando que cada proceso se asigna a un procesador/core distinto.

Solución

(a)

Pre-condición

x : contendrá el máximo

N : número de componentes del vector

$V[]$: vector con los N n^o de entrada

Pos-condición

Imprime en pantalla, x , el máximo encontrado en $V[]$

```

Máximo secuencial

x = V[0];
for ( i=1 ; i<N; i++)
    if (V[i]>x) x=V[i];

printf("Máximo = %f \n", x);

```

Comparaciones: El número de instrucciones de comparación, sin contar las instrucciones que controlan la finalización del bucle, es igual a $N-1$, y $2N-1$ contando las instrucciones que controlan la finalización del bucle

(b)

Pre-condición

x : contendrá el máximo, $xaux$: recepción de máximos

N : número de componentes del vector

$V[]$: vector con los N n^o de entrada, todos los procesos tendrán un vector V pero el proceso 0 será el único que al principio de la ejecución tendrá en V los datos, el resto no los tiene (así lo establece el enunciado del ejercicio).

$grupo$: identificador del grupo de procesos que intervienen en la comunicación.

$nproc$: número de procesos en $grupo$.

$idproc$: identificador del proceso (dentro del grupo de $num_procesos$ procesos) que ejecuta el código

$tipo$: tipo de los datos a enviar o recibir

$count$: es $N/nproc$

Pos-condición

Imprime en pantalla, x , el máximo encontrado en $V[]$

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Prof. responsables: M. Anguita, J. Ortega



9 / 10

Máximo paralelo

```
//Difusión V, se supone que nproc divide a N
if (idproc==0)
    for (i=1; i<nproc; i++) send(V[i*count],count, tipo, i, grupo);

else receive(V, count, tipo, 0, grupo);

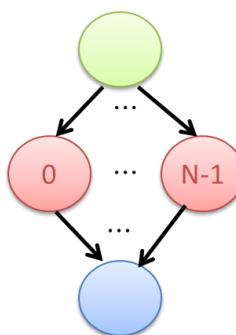
//Cálculo paralelo, asignación estática
x = V[0];
for ( i=1 ; i<count; i++)
    if (V[i]>x) x=V[i];

//Comunicación resultados
if (idproc==0)
    for (i=1; i<nproc) {
        receive(xaux,1,tipo,i,grupo);
        if (xaux>x) x = xaux;
    }
else send(x,1,tipo,0,grupo);

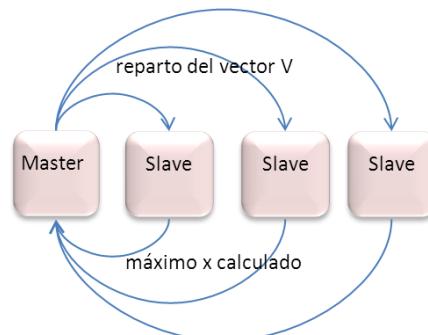
//Proceso 0 imprime resultado
if (idproc==0) printf("Máximo = %f \n", x);
```

(c)

Grafo de tareas



Grafo de procesos



(d) El número de instrucciones de comparación que se realizan en paralelo, sin contar las comparaciones que controlan el final del bucle, es igual a $(N/nproc)-1$. Cada procesador realiza $(N/nproc)-1$ comparaciones ya que el bucle en todos los procesos se ejecuta desde 1 a $(N/nproc)-1$. A cada proceso se trasfiere $N/nproc$ números que se recogen en su vector V . El proceso 0 realiza $nproc-1$ comparaciones más que el resto porque calcula el máximo de los máximos parciales calculados por todos los procesos. Si se cuentan las instrucciones de comparación que se realizan para controlar el final del bucle, el total de comparaciones ascendería a $2(N/nproc)-1$.

(e) $T_s(N) = (N-1) * t$ (teniendo en cuenta sólo las comparaciones con x ; no se tienen en cuenta, por tanto, las comparaciones que controlan el final del bucle)



Examen final del 20/06/2012 resuelto



ETSIIT



(f) Llamando p a n_{proc} y teniendo en cuenta sólo las comparaciones con x (no se tienen en cuenta, por tanto, las comparaciones que controlan el final del bucle):

$$T_p(p, N) = T_c(p, N) + T_{c/s}(N) = [(N/p-1)*t + (p-1)*t] + [10000*t + N/p*2*t + 10000*t + 2*t]$$

El tiempo de comunicación depende del tiempo que supone el reparto de v antes de hacer los cálculos (se envían N/p datos a cada procesador) más el tiempo que supone la trasferencia de los máximos parciales calculados por los procesos (se envía un dato por cada proceso).

$$T_p(p, N) = \{ [(10^6/10-1) + 9] + [2*10^4 + 2*10^6/10 + 2] \} * t$$

$$S(p, N) = (10^6-1)*t / \{ [(10^6/10-1) + 9] + [2*10^4 + 2*10^6/10 + 2] \} * t$$

$$S(p, N) \sim 10^6 / (10^5 + 2*10^4 + 2*10^5)$$

$$S(p, N) \sim 100 / (10 + 2 + 20)$$

$$S(p, N) \sim 100 / 32 = 3,125$$



2º curso / 2º cuatr.

 Grado en
 Ing. Informática

Arquitectura de Computadores: Ejercicios y Cuestiones

Tema 1. Arquitecturas Paralelas: Clasificación y Prestaciones

Material elaborado por los profesores responsables de la asignatura:
 Mancia Anguita, Julio Ortega

Licencia Creative Commons 

1 Ejercicios

Ejercicio 1. En un procesador no segmentado que funciona a 300 MHz, hay un 20% de instrucciones LOAD que necesitan 4 ciclos, un 10% de instrucciones STORE que necesitan 3 ciclos, un 25% de instrucciones con operaciones de enteros que necesitan 6 ciclos, un 15% de instrucciones con operandos en coma flotante que necesitan 8 ciclos por instrucción, y un 30% de instrucciones de salto que necesitan 3 ciclos. **(a)** ¿Cuál es la ganancia que se puede obtener por reducción en el tiempo de las instrucciones con enteros a 3 ciclos? y **(b)** ¿cuál es la ganancia que se puede obtener por reducción en el tiempo de las instrucciones en coma flotante a 3 ciclos?

Solución

Datos del ejercicio:

Tipo i de instr.	CPI _i (c/i)	NI _i	CPI _i ^a	CPI _i ^b
LOAD	4	0.20*NI	4	4
STORE	3	0.10*NI	3	3
ENTEROS	6	0.25*NI	3	6
FP	8	0.15*NI	8	3
BR	3	0.30*NI	3	3
TOTAL		NI		

La frecuencia de reloj no hará falta.

El tiempo total de procesamiento antes de la mejora es igual a:

$$T_{\text{sin_mejora}} = NI \times CPI \times T_{\text{ciclo}} = (NI \times 0.2 \times 4 + NI \times 0.1 \times 3 + NI \times 0.25 \times 6 + NI \times 0.15 \times 8 + NI \times 0.3 \times 3) \times T_{\text{ciclo}} = \\ NI \times (0.2 \times 4 + 0.1 \times 3 + 0.25 \times 6 + 0.15 \times 8 + 0.3 \times 3) \times T_{\text{ciclo}} = \\ NI \times (0.8 + 0.3 + 1.5 + 1.2 + 0.9) \times T_{\text{ciclo}} = NI \times 4.7 \times T_{\text{ciclo}}$$

donde NI es el número de instrucciones y $T_{\text{ciclo}}=1/\text{frecuencia}$

(a) En el caso de mejoras en la ejecución de enteros, el tiempo sería:

$$T_{\text{mejora_ent}} = NI \times CPI \times T_{\text{ciclo}} = NI \times (0.2 \times 4 + 0.1 \times 3 + 0.25 \times 3 + 0.15 \times 8 + 0.3 \times 3) \times T_{\text{ciclo}} = \\ NI \times (0.8 + 0.3 + 0.75 + 1.2 + 0.9) \times T_{\text{ciclo}} = NI \times 3.95 \times T_{\text{ciclo}}$$

y por tanto la ganancia sería:

$$S_{\text{mejora_ent}} = T_{\text{mejora_ent}} / T_{\text{sin_mejora}} = NI \times 4.7 \times T_{\text{ciclo}} / NI \times 3.95 \times T_{\text{ciclo}} = 4.7 / 3.95 = 1,18987$$

$$S_{\text{mejora_ent}} = 1.19$$

(b) En el caso de mejoras en operaciones de coma flotante, el tiempo sería:

$$T_{\text{mejora_fp}} = NI \times CPI \times T_{\text{ciclo}} = NI \times (0.2 \times 4 + 0.1 \times 3 + 0.25 \times 6 + 0.15 \times 3 + 0.3 \times 3) \times T_{\text{ciclo}} = NI \times (0.8 + 0.3 + 1.5 + 0.45 + 0.9) \times T_{\text{ciclo}} = NI \times 3.95 \times T_{\text{ciclo}}$$

y por tanto la ganancia sería:

$$S_{\text{mejora_ent}} = T_{\text{mejora_ent}} / T_{\text{sin_mejora}} = NI \times 4.7 \times T_{\text{ciclo}} / NI \times 3.95 \times T_{\text{ciclo}} = 4.7 / 3.95 = 1.18987$$

$$S_{\text{mejora_fp}} = 1.19$$

En ambos casos la ganancia conseguida es la misma.

Ejercicio 2. Un circuito que implementaba una operación en $T_{\text{op}}=450$ ns. se ha segmentado mediante un cauce lineal con cuatro etapas de duración $T_1=100$ ns., $T_2=125$ ns., $T_3=125$ ns., y $T_4=100$ ns. respectivamente, separadas por un registro de acople que introduce un retardo de 25 ns. (a) ¿Cuál es la máxima ganancia de velocidad posible? ¿Cuál es la productividad máxima del cauce? (b) ¿A partir de qué número de operaciones ejecutadas se consigue una productividad igual al 90% de la productividad máxima?

Solución

Datos del ejercicio:



Sin segmentar



Con segmentación

(a) La ganancia en velocidad para n entradas se obtiene dividiendo en tiempo que tardan en ejecutarse n entradas en el circuito sin segmentar entre el tiempo que supone la ejecución en el circuito segmentado:

$$S(n) = \frac{T^{ss}(n)}{T^{cs}(n)}$$

El tiempo de ejecución de n entradas en el circuito sin segmentar es:

$$T^{ss}(n) = T_{\text{op}} \times n = 450 \text{ ns} \times n$$

El circuito se ha segmentado en cuatro etapas separadas por un registro de acople con un retardo de d=25 ns. El tiempo de ciclo del cauce, t, se obtiene a partir de la expresión:

$$t = \max \{T_1, T_2, T_3, T_4\} + d = \max \{100, 125, 125, 100\} + 25 \text{ ns} = 150 \text{ ns}$$

En ese cauce, una operación tarda un tiempo $TLI = 4 \times 150$ ns. (tiempo de latencia de inicio) en pasar por las cuatro etapas del mismo. El tiempo de ejecución de las n entradas en el circuito segmentado es:

$$T^{cs}(n) = TLI + (n-1) \times t = 4 \times 150 \text{ ns} + (n-1) \times 150 \text{ ns}$$

La ganancia en prestaciones conseguida al segmentar sería:

$$S(n) = \frac{T^{ss}(n)}{T^{cs}(n)} = \frac{450 \text{ ns} \times n}{4 \times 150 \text{ ns} + (n-1) \times 150 \text{ ns}} = \frac{3 \times n}{3+n}$$

El valor máximo de la ganancia de velocidad se obtiene aplicando el límite cuando n tiende a infinito:

$$S(n) = \lim_{n \rightarrow \infty} \frac{3 \times n}{3+n} = 3$$

La productividad del cauce es:

$$P(n) = \frac{n}{T^{cs}(n)} = \frac{n}{4 \times 150 \text{ ns} + (n-1) \times 150 \text{ ns}} = \frac{n}{(3+n) \times 150 \text{ ns}}$$

La productividad máxima es:

$$P(n) = \lim_{n \rightarrow \infty} \frac{n}{(3+n) \times 150 \text{ ns}} = \frac{1}{150 \text{ ns}} = 6.67 \text{ Mop./s}$$

(c) El valor de n para el que se consigue una productividad igual al 90% de la productividad máxima es:

$$P(n) = \frac{n}{(3+n) \times 150 \text{ ns}} = 0.9 \times \frac{1}{150 \text{ ns}} \Rightarrow \frac{n}{3+n} = 0.9 \Rightarrow n = 2.7 + 0.9 \times n \Rightarrow n = \frac{2.7}{0.1} = 27 \text{ op.}$$

Con 27 operaciones se alcanza el 90% de la productividad máxima. La productividad aumentará conforme se incremente n.

Ejercicio 3. En un procesador sin segmentación de cauce, determine cuál de estas dos alternativas para realizar un salto condicional es mejor:

- ALT1: Una instrucción COMPARE actualiza un código de condición y es seguida por una instrucción BRANCH que comprueba esa condición.
- ALT2: Una sola instrucción incluye la funcionalidad de las instrucciones COMPARE y BRANCH.

Hay que tener en cuenta que para ALT1, en el conjunto de programas de prueba, el 20% de las instrucciones son instrucciones BRANCH asociadas a salto condicional; que las instrucciones BRANCH en ALT1 y COMPARE+BRANCH en ALT2 necesitan 4 ciclos mientras que todas las demás necesitan sólo 3; y que el ciclo de reloj de la ALT1 es un 25% menor que el de la ALT2, dado que en éste caso la mayor funcionalidad de la instrucción COMPARE+BRANCH ocasiona una mayor complejidad en el procesador.

Solución

Datos del ejercicio:

$$T^1_{\text{ciclo}} = T^2_{\text{ciclo}} - 0.25 \times T^2_{\text{ciclo}} = 0.75 \times T^2_{\text{ciclo}}$$

ALT1

Tipo i de instr.	ciclos	NI _i	Proporción (%)	Comentarios
BRANCH en ALT1	4	0,2xNI ¹	20	
RESTO en ALT1	3	0,8xNI ¹	80	Incluye instrucciones COMPARE
TOTAL		NI¹	100	NI¹ : nº instr. en ALT1

ALT2

Tipo i de instr.	ciclos	NI _i	Proporción (%)	Comentarios
COMPARE+BRANCH en ALT2	4	0,2×NI ¹	20×100/80 = 25	ALT2 no tendrá aparte las instrucciones COMPARE, que son un 20% en ALT1
RESTO en ALT2	3	0,6×NI ¹	(80-20) ×100/80 = 75	ALT2 no tendrá aparte las instrucciones COMPARE, que son un 20% en ALT1
TOTAL		NI²=0,8×NI¹	100	Se reducen el nº de instr. en ALT2, NI ² , al no tener instr. COMPARE aparte

El tiempo de CPU se puede expresar como:

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$

donde CPI es el número medio de ciclos por instrucción, NI es el número de instrucciones, y T_{ciclo} el tiempo de ciclo del procesador.

Para realizar la comparativa se va a calcular T^1_{CPU} y T^2_{CPU} en función de las mismas variables:

- Para la alternativa primera, *ALT1*, se tiene:

$$T^1_{CPU} = NI^1 \times CPI^1 \times T^1_{ciclo} = NI^1 \times (0.2 \times 4 + 0.8 \times 3) \times T^1_{ciclo} = NI^1 \times 3.2 \times 0.75 \times T^2_{ciclo} = NI^1 \times 2.4 \times T^2_{ciclo}$$

(CPI para ALT1 es CPI¹=0.2×4+0.8×3=3.2)

- Para la alternativa segunda, *ALT2*, se tiene:

$$T^2_{CPU} = NI^2 \times CPI^2 \times T^2_{ciclo} = NI^2 \times (0.2 \times 4 + 0.6 \times 3) \times T^2_{ciclo} = NI^2 \times 2.6 \times T^2_{ciclo}$$

(CPI para ALT2 es CPI² = $\frac{0.2 \times NI^1 \times 4 + 0.6 \times NI^1 \times 3}{NI^2}$ = $(0.2 \times 4 + 0.6 \times 3) \times \frac{NI^1}{NI^2} = \frac{0.8 + 1.8}{0.8} = \frac{2.6}{0.8} = 3.25$)

Los tiempos de CPU para *ALT2* y para *ALT1* se han expresado en función de las mismas variables (número de instrucciones de *ALT1*, NI¹, y tiempo de ciclo de *ALT2*, T²_{ciclo}) para poder comparar entre ambos. Como se puede ver $T^1_{CPU} < T^2_{CPU}$ y, por tanto, se tiene que *ALT2* no mejora a *ALT1*. Es decir, aunque un repertorio con instrucciones más complejas puede reducir el número de instrucciones de los programas, reduciendo el número de instrucciones a ejecutar, esto no tiene que suponer una mejora de prestaciones si al implementar esta opción la mayor complejidad del procesador lo hace algo más lento.

Ejercicio 4. ¿Qué ocurriría en el problema anterior si el ciclo de reloj fuese únicamente un 10% mayor para la *ALT2*?

Solución

En este caso $T^2_{ciclo} = 1.10 \times T^1_{ciclo}$; por tanto:

$$T^1_{CPU} = NI^1 \times CPI^1 \times T^1_{ciclo} = NI^1 \times (0.2 \times 4 + 0.8 \times 3) \times T^1_{ciclo} = NI^1 \times 3.2 \times T^1_{ciclo}$$

$$T^2_{CPU} = NI^2 \times CPI^2 \times T^2_{ciclo} = NI^2 \times (0.2 \times 4 + 0.6 \times 3) \times 1.1 \times T^1_{ciclo} = NI^2 \times 2.6 \times 1.1 \times T^1_{ciclo}$$

Así, ahora $T^2_{CPU} < T^1_{CPU}$ y por lo tanto, la segunda alternativa sí que es mejor que la primera. Es decir, el mismo planteamiento que antes no mejoraba la situación de partida ahora sí lo consigue. Sólo ha sido necesario que el aumento del ciclo de reloj que se produce al hacer un diseño más complejo del procesador sea algo menor.

La situación reflejada en estos dos ejercicios (3 y 4) pone de manifiesto la necesidad de estudiar la arquitectura del computador desde un enfoque cuantitativo, y no sólo teniendo en cuenta razonamientos que pueden ser correctos, pero que pueden llevar a resultados finales distintos según el valor de las magnitudes implicadas.

Ejercicio 5. Considere un procesador no segmentado con una arquitectura de tipo LOAD/STORE en la que las operaciones sólo utilizan como operandos registros de la CPU. Para un conjunto de programas representativos de su actividad se tiene que el 43% de las instrucciones son operaciones con la ALU (3 CPI), el 21% LOADs (4 CPI), el 12% STOREs (4 CPI) y el 24% BRANCHs (4 CPI).

Se ha podido comprobar que un 25% de las operaciones con la ALU utilizan operandos en registros que no se vuelven a utilizar. Compruebe si mejorarían las prestaciones si, para sustituir ese 25% de operaciones, se añaden instrucciones con un dato en un registro y otro en memoria. Tengan en cuenta en la comprobación que para estas nuevas instrucciones el valor de CPI es 4 y que añadirlas ocasiona un incremento de un ciclo en el CPI de los BRANCHs, pero no afectan al ciclo de reloj.

Solución

Datos del ejercicio:

Alternativa 1

I _i ¹	CPI _i ¹	NI _i ¹	Comentarios
LOAD	4	0,21×NI ¹	
STORE	4	0,12×NI ¹	
ALU	3	0,43×NI ¹	25 % inst. que usan operandos en registros que no se vuelven a utilizar
BR	4	0,24×NI ¹	
TOTAL		NI¹	

Alternativa 2

I _i ²	CPI _i ²	NI _i ²	Comentarios
LOAD	4	(0,21-0,25×0,43)×NI ¹ = 0,1025×NI ¹	El 25 % de 43% desaparece al usarse ese mismo número de operaciones con la ALU que acceden a memoria
STORE	4	0,12×NI ¹	
ALU r-r	3	0,75×0,43×NI ¹ = 0,3225×NI ¹	
ALU r-m	4	0,25×0,43×NI ¹ = 0,1075×NI ¹	25% de las operaciones con la ALU; es decir el 25% del 43%
BR	5	0,24×NI ¹	
TOTAL		NI²=0,8925×NI¹	Es decir, NI² es igual a 0,8925×NI¹

En la Tabla anterior se muestra que, al introducir las nuevas instrucciones de operación con la ALU con uno de los operandos en memoria:

- Se reduce el 25% de las $0.43 \times NI^1$ instrucciones de operación con la ALU y operandos en registros a las que las nuevas instrucciones sustituyen, donde NI^1 es el nº de instrucciones para la arquitectura original.
- Se reduce en $0.25 \times 0.43 \times NI^1$ el número de LOADs, ya que según se indica en el enunciado, esos LOADs sólo están en el programa para cargar uno de los operandos de las operaciones con la ALU, que no se vuelven a utilizar nunca más.
- Hay que contabilizar las $0.25 \times 0.43 \times NI^1$ nuevas instrucciones de operación con la ALU que se introducen en el repertorio (y que sustituyen a las operaciones con la ALU y operandos en registros).
- Como resultado, al sumar todas las instrucciones que se tienen en la nueva situación, el número total de instrucciones se reduce (lógicamente, ya que se han ahorrado instrucciones LOADs), siendo igual a $NI^2=0.8925 \times NI^1$.

En primer lugar se calcula el tiempo de CPU para la situación inicial:

$$T_{CPU}^1 = CPI^1 \times NI^1 \times T_{ciclo} = (0.43 \times 3 + 0.21 \times 4 + 0.12 \times 4 + 0.24 \times 4) \times NI^1 \times T_{ciclo} = (1.29 + 2.28) \times NI^1 \times T_{ciclo} = 3.57 \times NI^1 \times T_{ciclo}$$

(CPI para ALT1 es $CPI^1=0.43 \times 3 + 0.21 \times 4 + 0.12 \times 4 + 0.24 \times 4 = 1.29 + 0.57 \times 4 = 1.29 + 2.28 = 3.57$)

En segundo lugar se calcula el tiempo de CPU para la alternativa 2:

$$T_{CPU}^2 = CPI^2 \times NI^2 \times T_{ciclo} = [0.3225 \times 3 + (0.12 + 0.1025 + 0.1075) \times 4 + 5 \times 0.24] \times NI^1 \times T_{ciclo} = \\ (0.9675 + 0.33 \times 4 + 0.24 \times 5) \times NI^1 \times T_{ciclo} = 3.4875 \times NI^1 \times T_{ciclo}$$

Como se puede ver, en este caso, $T_{CPU}^1 > T_{CPU}^2$, y por lo tanto se mejoran las prestaciones, pero si el porcentaje de instrucciones sustituidas fuese un 20% en lugar de un 25%, en ese caso, se puede ver que

$$T_{CPU}^2 = CPI^2 \times NI^2 \times T_{ciclo} = 3.59 \times NI^1 \times T_{ciclo}$$

Ahora, en cambio, la segunda opción no mejora la primera. Como conclusión, se puede indicar que una determinada decisión de diseño puede suponer una mejora en el rendimiento del computador correspondiente según sean las características de las distribuciones de instrucciones en los programas que constituyen la carga de trabajo característica del computador.

Por tanto, queda clara también la importancia que tiene el proceso de definición de conjuntos de *benchmarks* para evaluar las prestaciones de los computadores, y las dificultades que pueden surgir para que los fabricantes se pongan de acuerdo en aceptar un conjunto de *benchmarks* estándar.

Ejercicio 6. Se ha diseñado un compilador para la máquina LOAD/STORE del problema anterior. Ese compilador puede reducir en un 50% el número de operaciones con la ALU, pero no reduce el número de LOADs, STOREs, y BRANCHs. Suponiendo que la frecuencia de reloj es de 50 Mhz. ¿Cuál es el número de MIPS y el tiempo de ejecución que se consigue con el código optimizado? Compárelos con los correspondientes del código no optimizado.

Solución

Datos del ejercicio:

Alternativa 1

I_i^1	CPI_i^1	NI_i^1/NI^1	Comentarios
LOAD	4	0,21	
STORE	4	0,12	
ALU	3	0,43	
BR	4	0,24	
TOTAL		1	

Alternativa 2

I_i^2	CPI_i^2	NI_i^2/NI^1	Comentarios
LOAD	4	0,21	
STORE	4	0,12	
ALU r-r	3	$0.5 * 0.43 = 0.215$	Se reducen las instrucciones que usan la ALU en un 50%
BR	4	0,24	
TOTAL		0,785	Es decir, $NI^2 = 0,785 * NI^1$

En la situación inicial del problema anterior se tenía que

$$T_{CPU}^1 = CPI * NI^1 * T_{ciclo} = 3.57 * NI^1 * T_{ciclo}$$

y, por lo tanto

$$\text{MIPS}^1 = \frac{NI^1}{T_{CPU}^1 * 10^6} = \frac{NI^1}{CPI * NI^1 * T_{ciclo} * 10^6} = \frac{f(\text{hz})}{CPI * 10^6} = \frac{50 * 10^6 \text{ c/s}}{3.57 \text{ c/i} * 10^6} = \mathbf{14.005 \text{ MIPS}}$$

El tiempo de CPU para la alternativa 2 es:

$$T_{CPU}^2 = CPI^2 * NI^2 * T_{ciclo} = [0.215*3 + (0.21+0.12+0.24)*4] * NI^1 * T_{ciclo} = (0.645+0.57*4) * NI^1 * T_{ciclo} = (0.645+2.28) * NI^1 * T_{ciclo} = 2.925 * NI^1 * T_{ciclo}$$

$$T_{CPU}^2 = 2.925 * \frac{NI^1}{NI^2} * NI^2 * T_{ciclo} = 2.925 * \frac{NI^1}{NI^2} * NI^2 * T_{ciclo} = 3.726 * NI^2 * T_{ciclo}$$

$$MIPS^2 = \frac{NI^2}{T_{CPU}^2 * 10^6} = \frac{NI^2}{CPI * NI^2 * T_{ciclo} * 10^6} = \frac{f(hz)}{CPI * 10^6} = \frac{50 * 10^6 \text{ c/s}}{3.726 \text{ c/i} * 10^6} = 13.42 \text{ MIPS}$$

Como se puede ver, se consigue una reducción de tiempo de ejecución (T_{CPU}^2 es menor que T_{CPU}^1), pero sin embargo, el número de MIPS para la segunda opción es menor. Se tiene aquí un ejemplo en el que los MIPS dan una información inversamente proporcional a las prestaciones. La razón de esta situación, en este caso, es que se ha reducido el número de las instrucciones que tenían un menor valor de CPI. Así, se incrementan las proporciones de las instrucciones más lentas en el segundo caso (por eso crece el valor de CPI), y claro, el valor de los MIPS se reduce. No obstante, hay que tener en cuenta que aunque las instrucciones que se ejecutan son más lentas, hay que ejecutar un número menor de instrucciones, y al final, el tiempo se reduce.

Ejercicio 7. En un programa que se ejecutan en un procesador no segmentado que funciona a 100 MHz, hay un 20% de instrucciones LOAD que necesitan 4 ciclos, un 15% de instrucciones STORE que necesitan 3 ciclos, un 40% de instrucciones con operaciones en la ALU que necesitan 6 ciclos, y un 25% de instrucciones de salto que necesitan 3 ciclos. **(a)** Si en las instrucciones que usan la ALU el tiempo en la ALU supone 4 ciclos, determine cuál es la máxima ganancia que se puede obtener si se mejora el diseño de la ALU de forma que se reduce su tiempo de ejecución a la mitad de ciclos. **(b)** ¿Con qué porcentaje de instrucciones con operaciones en la ALU se podría haber obtenido en los cálculos del apartado (a) una ganancia mayor que 2? Razona su respuesta.

Solución

Datos del ejercicio:

Alternativa 1

I _i ¹	CPI _i ¹	NI _i /NI	Comentarios
LOAD	4	0,2	
STORE	3	0,15	
ALU	6	0,4	4 ciclos en la ALU
BR	3	0,25	
TOTAL	1		

Alternativa 2

I _i ²	CPI _i ²	NI _i /NI	Comentarios
LOAD	4	0,2	
STORE	3	0,15	
ALU	2+2=4	0,4	Se reducen en 2 ciclos lo que consumen en la ALU
BR	3	0,25	
TOTAL	1		

(a) El tiempo de ejecución sin la mejora sería igual a

$$T_{CPU}^1 = NI * (0.2*4 + 0.15*3 + 0.4*6 + 0.25*3) * T_c = NI * 4.4 * T_c$$

donde f es la frecuencia y NI el número de instrucciones promedio de los programas. Puesto que la mejora consiste en que las operaciones con la ALU necesitan la mitad de tiempo, dado que dicha operación

consume 4 ciclos, tras la mejora pasaría a necesitar 2 ciclos, y por tanto, las instrucciones con la ALU necesitarían $2+2=4$ ciclos. Por tanto, el tiempo con la mejora sería:

$$T_{CPU}^2 = NI * (0.2 * 4 + 0.15 * 3 + 0.4 * 4 + 0.25 * 3) * T_c = NI * 3.6 * T_c$$

De esta forma, la ganancia de velocidad sería:

$$S = T_{CPU}^1 / T_{CPU}^2 = 4.4 / 3.6 = 1.22$$

(b) La ganancia en prestaciones conseguida en una instrucción de la ALU es de 6 ciclos/4 ciclos=1.5. Aunque todas las instrucciones fuesen instrucciones de la ALU la ganancia nunca podría llegar a 2, llegaría a 1.5, que es lo que han mejorado las instrucciones de la ALU. Supongamos que todas las instrucciones son de la ALU, entonces:

$$T_{CPU}^2 = 4 \text{ ciclos} * NI$$

$$T_{CPU}^1 = 6 \text{ ciclos} * NI$$

$$S = T_{CPU}^1 / T_{CPU}^2 = 1.5$$

Ejercicio 8. Suponga que, en los programas que constituyen la carga de trabajo habitual de un procesador, las instrucciones de coma flotante consumen un promedio del 13% del tiempo del procesador.

(a) Ha aparecido en el mercado una nueva versión del procesador en la que la única mejora con respecto a la versión anterior es una nueva unidad de coma flotante que permite reducir el tiempo de las instrucciones de coma flotante a tres cuartas partes del tiempo que consumían antes. ¿Cuál es la máxima ganancia de velocidad que puede esperarse en los programas si se sustituye el procesador de la versión antigua por el nuevo?

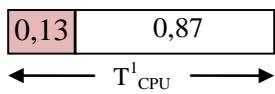
(b) ¿Cuál es la máxima ganancia de velocidad que, en promedio, puede esperarse en los programas debido a mejoras en la velocidad de las operaciones en coma flotante?

(c) ¿Cuál debería ser el porcentaje de tiempo de cálculo con datos en coma flotante en los programas que ejecuta el procesador para esperar una ganancia máxima de 4?

(d) En la situación anterior, ¿cuánto debería reducirse el tiempo de las operaciones en coma flotante con respecto a la situación inicial para que la ganancia sea 2?

Solución

Datos del ejercicio:



(a) Resolución 1:

$$T_{CPU}^2 = 0.13 \times \frac{3}{4} \times T_{CPU}^1 + 0.87 \times T_{CPU}^1 = 0.0975 \times T_{CPU}^1 + 0.87 \times T_{CPU}^1 = 0.9675 \times T_{CPU}^1$$

$$S \leq \frac{T_{CPU}^1}{T_{CPU}^2} = \frac{T_{CPU}^1}{0.9675 \times T_{CPU}^1} \cong 1.0336$$

(a) Resolución 2 (se usa la expresión que caracteriza la ley de Amdahl):

Como el tiempo de las instrucciones de coma flotante se reduce tres cuartas partes, la mejora de velocidad es $p=4/3$ (la inversa de la reducción del tiempo).

Como el porcentaje de instrucciones de coma flotante es el 13%, la fracción a la que se puede aplicar la mejora es:

$$(1-f) = 0.13, \text{ y por tanto, } f = 0.87$$

Sustituyendo estos valores en la expresión de la ley de Amdahl se tiene:

$$S \leq \frac{p}{1 + f \times (p - 1)} = \frac{4/3}{1 + 0.87 \times (4/3 - 1)} = \frac{4/3}{1 + 0.87 \times 1/3} = \frac{4}{3 + 0.87} \cong 1.0336$$

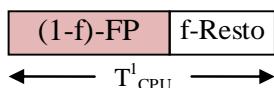
(b) Resolución 1: La ganancia será mejor cuanto menor sea el tiempo que supongan las operaciones FP. La mayor ganancia se conseguiría haciendo ese tiempo despreciable, es decir, si la mejora de las unidades FP se hace muy muy grande, infinito. En este caso:

$$T_{\text{CPU}}^2 = 0 + 0.87 \times T_{\text{CPU}}^1 = 0.87 \times T_{\text{CPU}}^1 \quad S \leq \frac{T_{\text{CPU}}^1}{T_{\text{CPU}}^2} = \frac{T_{\text{CPU}}^1}{0.87 \times T_{\text{CPU}}^1} = \frac{1}{0.87} \cong 1.149$$

(b) Resolución 2 (se usa la expresión que caracteriza la ley de Amdahl): Para determinar el valor de la ganancia máxima que se pide, se obtiene el límite cuando la mejora del recurso (instrucciones en coma flotante) tiende a infinito:

$$S_{\max(f=\text{cte})} \leq \lim_{p \rightarrow \infty} \frac{p}{1 + f \times (p - 1)} = \frac{1}{f} = \frac{1}{0.87} \cong 1.149$$

(c) Resolución 1:



Si se cambia la fracción de código con FP (la notamos por $(1-f)$) y tenemos en cuenta que para obtener la mayor ganancia se debe hacer el tiempo de ejecución de las FP igual a 0, entonces:

$$T_{\text{CPU}}^2 = 0 + f \times T_{\text{CPU}}^1 \quad S \leq \frac{T_{\text{CPU}}^1}{T_{\text{CPU}}^2} = \frac{T_{\text{CPU}}^1}{f \times T_{\text{CPU}}^1} = \frac{1}{f} = 4 \Rightarrow f = 0.25 \text{ y } (1-f) = 0.75$$

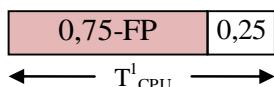
f debería ser al menos 0,25 y, por tanto, $1-f=0,75$; es decir, las operaciones FP deberían ser un 75%.

(c) Resolución 2 (se usa la expresión que caracteriza la ley de Amdahl): Para que la ganancia máxima (es decir cuando la mejora, p , tiende a infinito) sea igual a 4, la fracción del tiempo inicial que no se reduce con la mejora se obtendría a partir de:

$$S_{\max(f=\text{cte})} \leq \frac{1}{f} = 4$$

Por lo que $f=1/4=0.25$, y la fracción de tiempo que tendría que poder reducirse con la mejora tendría que ser $(1-f)=1-0.25=0.75$.

(d) Resolución 1:



$$T_{\text{CPU}}^2 = 0.75 \times \frac{1}{p} \times T_{\text{CPU}}^1 + 0.25 \times T_{\text{CPU}}^1$$

$$S \leq \frac{T_{\text{CPU}}^1}{T_{\text{CPU}}^2} = \frac{T_{\text{CPU}}^1}{0.75 \times \frac{1}{p} \times T_{\text{CPU}}^1 + 0.25 \times T_{\text{CPU}}^1} = \frac{1}{0.75 \times \frac{1}{p} + 0.25} = 2 \Rightarrow p = \frac{1.5}{0.5} = 3$$

El tiempo se debería reducir a 1/3 del tiempo original. Es decir, las instrucciones de coma flotante deberían hacerse tres veces más rápidas.

(d) Resolución 2 (se usa la expresión que caracteriza la ley de Amdahl): Si, al utilizar el valor de f anterior se deseara obtener una mejora de velocidad de 2 en los programas, la mejora de velocidad en las instrucciones de coma flotante (es decir p) se puede obtener a partir de la ley de Amdahl:

$$S \leq 2 = \frac{p}{1 + f \times (p - 1)} = \frac{1}{f + \frac{f-1}{p}} = \frac{1}{25 + \frac{0.75}{p}} \Rightarrow p = \frac{1.5}{0.5} = 3$$

Es decir, las instrucciones de coma flotante deberían hacerse tres veces más rápidas.



Ejercicio 9. Suponga que, en el código siguiente, $a[]$ es un array de números de 32 bits en coma flotante y b un número de 32 bits en coma flotante, y que debería ejecutarse en menos de 0,5 segundos para $N=10^9$:

```
for (i=0; i<N; i++) a[i+2]=(a[i+2]+a[i+1]+a[i]) *b;
```

(a) ¿Cuántos GFLOPS se necesitan para poder ejecutar el código en menos de 0,5 segundos?

(b) Suponiendo que este código en ensamblador tiene $7N$ instrucciones y que se ha ejecutado en un procesador de 32 bits a 2 GHz. ¿Cuál es el número medio de instrucciones que el procesador tiene que poder completar por ciclo para poder ejecutar el código en menos de 0,5 segundos?

(c) Estimando que el programa pasa el 75% de su tiempo de ejecución realizando operaciones en coma flotante, ¿cuánto disminuiría como mucho el tiempo de ejecución si se redujesen un 75% los tiempos de las unidades de coma flotante?

Solución

(a) Puesto que hay tres operaciones en coma flotante por iteración (dos sumas y un producto, con igual coste para el producto y la suma) el número de operaciones en coma flotante a realizar tras las N iteraciones es $N_{\text{float}} = 3 \times N = 3 \times 10^9$.

Por lo tanto, $\text{GFLOPS} = N_{\text{float}} / (t \times 10^9)$ donde t es el tiempo en segundos. Así:

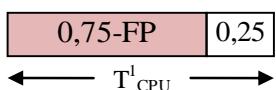
$$\text{GFLOPS} = N_{\text{float}} / (t \times 10^9) = 3 \times 10^9 / 0.5 \times 10^9 = 6 \text{ GFLOPS}$$

(b) Si el programa tiene $NI = 7N = 7 \times 10^9$ instrucciones, teniendo en cuenta que

$$T = NI \times CPI \times T_{\text{ciclo}} = NI \times CPI / f = (7 \times 10^9 \text{ instrucciones}) \times CPI / (2 \times 10^9 \text{ ciclos/s}) < 0.5 \text{ s}$$

Al despejar $CPI < 0.5 / 3.5$, y el número de instrucciones por ciclo $IPC = 1/CPI > 7$

(c) Resolución 1:



$$T_{\text{CPU}}^2 = 0.25 \times T_{\text{CPU}}^1 + 0.25 \times 0.75 \times T_{\text{CPU}}^1 = 0.25 \times T_{\text{CPU}}^1 \times (1 + 0.75)$$

$$S \leq \frac{T_{\text{CPU}}^1}{T_{\text{CPU}}^1} - \frac{T_{\text{CPU}}^2}{T_{\text{CPU}}^1} = 1 - 0.25 \times 1.75 = 1 - 0.4375 = 0.5625 \Rightarrow 56.25\%$$

Por tanto, el tiempo se reduce un 56.25%

Resolución 2: Para resolver la última cuestión se puede recurrir a la ley de Amdahl. En este caso $f=0.25$ es la fracción de tiempo en el que no se puede aprovechar la mejora; el incremento de velocidad es $p=1/0.25$ (si el tiempo de las operaciones en coma flotante era t , el tiempo de las operaciones en coma flotante con la mejora es $0.25t$, dado que se ha reducido un 75%). La mejora de velocidad que se observaría se puede obtener aplicando la ley de Amdahl, sustituyendo convenientemente:

$$S \leq p / (1+f(p-1)) = 4 / (1+0.25 \times 3) = 2.286$$

Como $S = T_{\text{sin_mejora}}/T_{\text{con_mejora}} \leq 2.286$ se tiene que $T_{\text{sin_mejora}}/2.286 = 0.4375 * T_{\text{sin_mejora}} \leq T_{\text{con_mejora}}$. Es decir, el tiempo con la mejora podría pasar a ser el 43.75% del tiempo sin la mejora y, por tanto, se reduce el tiempo en un 56.75%.

Ejercicio 10. Un compilador ha generado un código máquina optimizado para el siguiente programa

```
par=0; impar=0;
for (i=0;i<N;i++)
    if ((i%2) == 0)
        par=par+c*x[i];
    else
        impar=impar-c*x[i];
```

sin utilizar instrucciones de salto dentro de las iteraciones del bucle (porque se ha usado la técnica de desenrollado el bucle que veremos en el Seminario 4): el código tiene un número de iteraciones de $N/2$, 7 instrucciones fuera del bucle (2 de almacenamiento en memoria, 5 instrucciones para inicializar registros), 10 instrucciones dentro del bucle (4 instrucciones para implementar el bucle `for`: incremento de la variable de control `i`, comparación, salto condicional y un salto incondicional; 4 instrucciones coma flotante y 2 instrucciones de carga desde memoria a registro -se leen dos componentes de `x`). El computador donde se ejecuta dispone de:

- Un procesador superescalar de 32 bits a 2 GHz capaz de terminar dos instrucciones de coma flotante por ciclo y dos instrucciones de cualquier otro tipo por ciclo, excepto instrucciones de carga, cuyo tiempo depende de si hay o no fallo de cache (si no hay fallo de cache suponen 1 ciclo), y las instrucciones de almacenamiento que suponen 1 ciclo.
- Dos caches integradas en el chip de procesamiento (una para datos y otra para instrucciones) de 512 KBytes cada una, mapeo directo, política de actualización de postescritura, líneas de 32 bytes, y latencia de un ciclo de reloj.
- Una memoria principal con latencia de 30 ns y ciclos burst 6-1-1-1 a través de un bus de memoria de 200 MHz con 64 bits.

Conteste a las siguientes cuestiones: **(a)** ¿Cuál es la velocidad pico del procesador (en GFLOPS)? **(b)** Estime el tiempo que tarda en ejecutarse el programa para $N=2^{11}$ teniendo en cuenta los datos aportados por el ejercicio. **(c)** Estime los MFLOPS que alcanza el programa a partir del resultado obtenido en el apartado (b).

(NOTA: Considere que el vector `x` se almacena en memoria en una dirección múltiplo del tamaño de una línea de cache y que ningún componente está en cache cuando se referencia; `N`, `i` estarán en registros de enteros, `par`, `impar`, `c`, y `x[]` son números de 32 bits en coma flotante, dentro del bucle `c`, `par` e `impar` estarán en registros).

Solución

Datos del ejercicio:

Tipo i	1/CPI _i	CPI _i
LOAD	1 instr./ciclo	1 si NO hay fallo de cache
STORE	1 instr./ciclo	1
Resto	2 instr./ciclo	0.5
burst	6 - 1 - 1 - 1	Frecuencia del bus = 200 MHz => ciclo de 1/200 MHz = 5 ns => 6 ciclos = 30ns
Flanco reloj	↑ ↑ ↑ ↑	Bus de 64 bits (4B): el bus trasfiere 8B (dos palabras de 4B) en paralelo en cada flanco de disparo del reloj del bus
Bytes transferidos	8B 8B 8B 8B = 32B	Línea de cache de 32B: se supone entonces que un <i>burst</i> obtiene una línea
Ciclos procesador	60 10 10 10 ciclos	Ciclos procesador por ciclos bus: $\frac{2 \text{ GHz proc.}}{0.2 \text{ GHz bus}} = 10 \text{ ciclos}$
Ciclos componente entre MP y cache	60 70 80 90 ciclos... x[0] x[2] x[4] x[6] x[1] x[3] x[5] x[7]	Se supone que x comienza en un múltiplo de una línea de cache. Al acceder a x[0] hay un fallo de cache. Tras 60 ciclos estarán en la cache x[0] y x[1] (en 61 ciclos estará x[0] en el procesador y un ciclo más tarde estará x[1]), tras 10 ciclos más (70 ciclos) estarán x[2] y x[3] en la cache, tras otros 10 ciclos (80 ciclos) estarán x[4] y x[5], tras otros 10 ciclos (90 ciclos) x[6] y x[7]. A continuación, el acceso a x[8] y x[9] supone otro fallo de cache de 60 ciclos. Entonces, el acceso a una línea de cache completa supone 90 ciclos.

Código

Código optimizado	Instrucciones máquina	Ciclos (ver primera tabla del ejercicio)
par=0; impar=0;	5 instr. inicialización de registros (par, impar, i, N, c)	5 instr. * 0.5 ciclos/instr. = 2.5 ciclos
for (i=0;i<N;i=i+2)	ADD i,2 CMP i,N BRCND si i>=N salto a STORE	0.5 ciclos 0.5 ciclos 0.5 ciclos
 par=par+c*x[i];	2 LOAD (x[i],x[i+1])	1.5 ciclos
 impar=impar-c*x[i+1];	2 MULFP, 1 ADDFP, 1SUBFP	2i * 1c/i = 2 ciclos 4 i * 0.5c/i = 2 ciclos
	BR a ADD	0.5 ciclos
	2 STORE (par, impar)	6 o 10 ciclos MP cache
		6 ciclos sin fallos de cache
	10* N/2 + 7 instr. en total	2 instr. * 1 ciclo/instr. = 2 ciclos

En la tabla, ADD nota instrucción de suma, CMP instrucción de comparación, BR instrucción de salto y BRCND instrucción de salto condicional. El bucle for se implementa en ensamblador con 4 instrucciones: 3 instrucciones (incremento variable de control *i*, ADD, comparación variable de control con *i*, CMP, salto condicional si *i* no alcanza *N*, BRCND) antes del código que ejecutan las iteraciones del bucle y una instrucción detrás de estas instrucciones (salto incondicional, BR, a la instrucción de suma, ADD, que incrementa *i*).

(a) Velocidad pico del Procesador:

$$V_{\text{pico}} = 2 \text{ inst/ciclo} \times 2.10^9 \text{ ciclos/sg} = 4.10^9 \text{ inst/sg} = 4 \text{ GFLOPS}$$

(b) El número de iteraciones es la mitad del número de iteraciones del bucle original:

$$\text{Iteraciones} = N / 2 = 2^{11}/2 = 2^{10}$$

El número de instrucciones ejecutadas es (ver primera tabla de datos del ejercicio):

$$N\text{Instrucciones} = 9*2^{10} + 7 = 9223$$

El número de fallos de cache será de:

$$\text{Fallos de cache} = \text{nº de líneas de cache en } x =$$

$$\frac{2^{11} \text{ comp.} \times 2^2 B/\text{comp.}}{2^5 B/LC} = 2^8 \text{ Líneas de cache}$$

Tiempo mínimo de procesamiento (ver primera tabla del ejercicio):

$$\text{Ciclos} = 5 \text{ instr. * 0.5 ciclos/instr. (iniciar 5 registros)} + \text{ciclos del bucle} + 2 \text{ instr. * 0.5 ciclos/instr. (2 STORE)}$$

$$\text{Ciclos} = 2.5 \text{ ciclos} + \text{ciclos del bucle} + 1 \text{ ciclos} = 3.5 \text{ ciclos} + \text{ciclos del bucle}$$

Ciclos que supone la 1ª iteración del bucle:

$$\text{Ciclos 1ª iteración del bucle} = 3i (\text{ADD, CMP, BRCND}) * 0.5c/i + 60 \text{ c (fallo de cache)} + 1 \text{ c (load de cache)} + 2i*0.5c/i (\text{2 op. FP}) + 1 \text{ c (load de cache)} + 2*0.5c/i (\text{2 op. FP}) + 1i*0.5c/i (\text{BR: salto incondicional})$$

Ciclos que supone la ejecución de instrucciones entre la instr. de carga de $x[0]$ y la instr. de carga de $x[2]$:

$$\begin{aligned} \text{Ciclos entre instr. carga de } x[0] \text{ e instr. carga de } x[2] &= (\text{LOAD de } x[0] + 4 \text{ instr. FP de la iteración 1} \\ &+ \text{instr. LOAD de } x[1] \text{ iteración 1} + \text{instr. BR iteración 1} + \text{ADD, CMP y BRCND de la iteración 2}) = 1 \text{ c (load de cache)} \\ &+ 2i*0.5c/i (\text{2 op. FP}) + 1 \text{ c (load de cache)} + 2*0.5c/i (\text{2 op. FP}) + 1i*0.5c/i (\text{BR: salto incondicional}) \\ &+ 3i (\text{ADD, CMP, BRCND}) * 0.5c/i = 1 \text{ c} + 1 \text{ c} + 1 \text{ c} + 1 \text{ c} + 0.5 \text{ c} + 1.5 \text{ c} = 6 \text{ c} \end{aligned}$$

Hay 6 ciclos, pero $x[2]$ no estará disponible en cache hasta pasados 10 ciclos desde que $x[0]$ y $x[1]$ llegaron a cache. Luego la carga y, por consiguiente, el resto de instrucciones de la 2º iteración no podrán ejecutarse hasta que pasen 10 ciclos, no 6 ciclos. Igual ocurre con la 3ª iteración. En la 4ª iteración hay que esperar más, ya que se produce un fallo de cache, lo que supone 60 ciclos.

Teniendo todo esto en cuenta, los ciclos del bucle serían:

Ciclos del bucle =

$$\begin{aligned} &3i (\text{ADD, CMP, BRCND iteración 1}) * 0.5c/i + 60 \text{ c (fallo de cache } x[0]) + \max(6c, 10c) * 3 \text{ (línea de cache 1)} + \\ &\sum_{i=1}^{2^8-1} (\max(6c, 60c) + \max(6c, 10c) * 3) (2^8 - 1 \text{ líneas de cache}) \\ &+ 1 \text{ c (load de cache iteración } N/2) + 2i*0.5c/i (\text{2 op. FP}) + 1 \text{ c (load de cache)} + 2*0.5c/i (\text{2 op. FP}) + \\ &1i*0.5c/i (\text{BR: salto incondicional}) + 3i (\text{ADD, CMP, BRCND}) * 0.5c/i \end{aligned}$$

$$\text{Ciclos del bucle} = 1.5c + (60c + 30c) + (2^{8-1}) * 90c + 6c = 7.5c + 2^{8-1} * 90c = 7.5c + 23040c = 23047.5c$$

Los ciclos en total serían:

$$\text{Ciclos} = 2.5 \text{ ciclos} + \text{ciclos del bucle} + 1 \text{ ciclos} = 3.5 \text{ ciclos} + \text{ciclos del bucle} = 23051c$$

El tiempo en segundos sería:

$$T = \frac{23051 \text{ c}}{2 \times 10^9 \text{ c/s}} = 11524.5 \text{ ns}$$

(c) MFLOPS que alcanza el programa (una operación FP de multiplicación y otra de resta por iteración, no se van a tener en cuenta en el cálculo las operaciones de acceso a memoria):

$$\text{MFLOPS} = \frac{2 \text{ op./iter.} \times 2^{11} \text{ iter.}}{11524.5 \times 10^{-9} \text{ s} \times 10^6} \cong 355.42$$



2 Cuestiones

Cuestión 1. Indique cómo se podría aprovechar un computador MISD para acelerar la determinación de si n números son primos o no. Considere que se conocen los M números primos entre 1 y el máximo valor que puede tener un número de entrada.



Cuestión 2. Indique cuál es la diferencia fundamental entre una arquitectura CC-NUMA y una arquitectura SMP.

Solución

Ambos, SMP y CC-NUMA, son multiprocesadores, es decir, comparten el espacio de direcciones físico. También en ambos se mantiene coherencia entre caches de distintos procesadores (*CC-Cache-Coherence*).

En un SMP (*Symmetric Multiprocessor*) la memoria se encuentra centralizada en el sistema a igual distancia (latencia) de todos los procesadores mientras que, en un CC-NUMA (*Cache-Coherence Non Uniform Memory Access*), cada procesador tiene físicamente cerca un conjunto de sus direcciones de memoria porque los módulos de memoria están físicamente distribuidos entre los procesadores y, por tanto, los módulos no están a igual distancia (latencia) de todos los procesadores.

La memoria centralizada del SMP hace que el tiempo de acceso a una dirección de memoria en un SMP sea igual para todos los procesadores y el tiempo de acceso a memoria de un procesador sea el mismo independientemente de la dirección a la que se esté accediendo; por estos motivos se denomina multiprocesador simétrico (*Symmetric Multiprocessor*) o UMA (*Uniform Memory Access*).

Los módulos de memoria físicamente distribuidos entre los procesadores de un CC-NUMA hacen que el tiempo de acceso a memoria dependa de si el procesador accede a una dirección de memoria que está en la memoria física que se encuentra en el nodo de dicho procesador (cercana, por tanto, al procesador que accede) o en la memoria física de otro nodo. Por este motivo el acceso no es uniforme o simétrico y recibe el nombre de NUMA.



Cuestión 3. ¿Cuándo diría que un computador es un multiprocesador y cuándo que es un multicomputador?

Solución. Será un multiprocesador si todos los procesadores comparten el mismo espacio de direcciones físico y será un multicomputador si cada procesador tiene su espacio de direcciones físico propio.



Cuestión 4. ¿Un CC-NUMA escala más que un SMP? ¿Por qué?

Solución. Sí, un CC-NUMA escala más que un SMP, porque al añadir procesadores al cálculo el incremento en el tiempo de acceso a memoria medio aumenta menos que en un SMP si el sistema operativo, el compilador y/o el programador se esfuerzan en ubicar en la memoria cercana a un procesador la carga de trabajo que va a procesar. La menor latencia media se debe a un menor número de conflictos en la red en el acceso a datos y a la menor distancia con el módulo de memoria físico al que se accede. Al aumentar menos la latencia se puede conseguir un tiempo de respuesta mejor en un CC-NUMA que en un SMP al ejecutar un código paralelo o múltiples códigos secuenciales a la vez.



Cuestión 5. Indique qué niveles de paralelismo implícito en una aplicación puede aprovechar un PC con un procesador de 4 cores, teniendo en cuenta que cada core tiene unidades funcionales SIMD (también llamadas unidades multimedia) y una microarquitectura segmentada y superscalar. Razona su respuesta.

Solución

Paralelismo a nivel de operación. Al ser una arquitectura con paralelismo a nivel de instrucción (ejecuta instrucciones en paralelo) por tener una arquitectura segmentada y superescalar, puede ejecutar operaciones en paralelo, luego puede aprovechar el paralelismo a nivel de operación.

Paralelismo a nivel de bucle (paralelismo de datos). Las operaciones realizadas en las iteraciones del bucle sobre vectores y matrices se podrían implementar en las unidades SIMD de los cores, lo que reduciría el número de iteraciones de los bucles. Además las iteraciones de los bucles, si son independientes, se podrían repartir entre los 4 cores.

Paralelismo a nivel de función. Las funciones independientes se podrían asignar a cores distintos para que se puedan ejecutar a la vez.

Paralelismo a nivel de programa. Los programas del mismo o distinto usuario se pueden asignar a distintos cores para así ejecutarlos al mismo tiempo.

Cuestión 6. Si le dicen que un ordenador es de 20 GIPS ¿puede estar seguro que ejecutará cualquier programa de 20000 instrucciones en un microsegundo?

Solución. Los MIPS se definen como el número de instrucciones que puede ejecutar un procesador (en millones) divididos por el tiempo que tardan en ejecutarse.

$$\text{MIPS} = \frac{\text{Número_de_Instrucciones}}{\text{tiempo(sg.)} * 10^6}$$

Los MIPS suelen utilizarse como medida de las prestaciones de un procesador, pero realmente sólo permiten estimar la velocidad pico del procesador, que solo permitiría comparar procesadores con el mismo repertorio de instrucciones en cuanto a sus velocidades pico. Esto se debe a que esta medida

- No tiene en cuenta las características del repertorio de instrucciones de procesador. Se da el mismo valor a una instrucción que realice una operación compleja que a una instrucción sencilla.
- Pueden tenerse valores de MIPS inversamente proporcionales a la velocidad del procesador. Si un procesador tiene un repertorio de instrucciones de tipo CISC que permite codificar un algoritmo con, por ejemplo, 1000 instrucciones, y otro con un repertorio de tipo RISC lo codifica con 2000, si el primero tarda 1 microsegundo y el segundo 1.5 microsegundos, tendremos que el primer procesador tiene 1000 MIPS y el segundo 1333 MIPS. Por tanto, si nos fijamos en los MIPS, el segundo procesador será mejor que el primero, aun precisando un 50% más de tiempo que el primero, para resolver el mismo problema.

De ahí que incluso se haya dicho que MIPS significa *Meaningless Information of Processor Speed* (información sin sentido de la velocidad del procesador).

En relación a la pregunta que se plantea, la respuesta puede tener en cuenta dos aspectos:

- El número de instrucciones que constituyen un programa (número estático de instrucciones) puede ser distinto del número de instrucciones que ejecuta el procesador finalmente (número dinámico de instrucciones), ya que puede haber instrucciones de salto, bucles, etc. que hacen que ciertas instrucciones del código se ejecuten más de una vez, y otras no se ejecuten nunca. Si la pregunta, al hacer referencia al número de instrucciones, se refiere al número estático, la respuesta es que NO se

puede estar seguro puesto que puede que al final no se ejecuten 20000 millones de instrucciones.

- Si el repertorio de instrucciones contiene instrucciones que tardan en ejecutarse tiempos diferentes, para que el programa tardase el mismo tiempo es preciso que se ejecutase el mismo número de instrucciones y del mismo tipo (en cuanto a tiempo de ejecución de cada una). En este caso, también, la respuesta es NO.



Cuestión 7. ¿Aceptaría financiar/embarcarse en un proyecto en el que se plantease el diseño e implementación de un computador de propósito general con arquitectura MISD? (Justifique su respuesta).

Solución: El tipo de procesamiento que realiza un procesador MISD puede implementarse en un procesador MIMD con la sincronización correspondiente entre los procesadores del computador MIMD para que los datos vayan pasando adecuadamente de un procesador a otro (definiendo un flujo único de datos que pasan de procesador a procesador). Por esta razón un computador MISD **no tiene mucho sentido** como computador de propósito general. Sin embargo, puede ser útil un diseño MISD para un dispositivo de propósito específico que sólo tiene que ejecutar una aplicación que implica un procesamiento en el que los datos tienen que pasar por distintas etapas en las que sufren ciertas transformaciones que pueden programarse (en cada uno de los procesadores). La especificidad del diseño puede permitir generar diseños muy eficientes desde el punto de vista del consumo, de la complejidad hardware, etc.



Cuestión 8. Deduzca la ley de Amdahl para la mejora de la velocidad de un procesador suponiendo que hay una probabilidad f de no utilizar un recurso del procesador cuya velocidad se incrementa en un factor p .

Solución: El tiempo necesario para ejecutar un programa en el procesador sin la mejora es igual a:

$$T_{sin_mejora} = f \times T_{sin_mejora} + (1-f) \times T_{sin_mejora}$$

donde $f \times T_{sin_mejora}$ es el tiempo que no podría reducirse al aplicar la mejora, y $(1-f) \times T_{sin_mejora}$ es el tiempo que podría reducirse como máximo en un factor igual a p al aplicar la mejora. Por tanto, el tiempo al aplicar la mejora sería:

$$T_{con_mejora} \geq f \times T_{sin_mejora} + ((1-f) \times T_{sin_mejora})/p$$

El signo “ \geq ” se usa porque al aplicar la mejora es posible añadir algún tiempo de sobrecarga extra. Según esto, la ganancia de velocidad que se podría conseguir sería:

$$S = T_{sin_mejora}/T_{con_mejora} \leq T_{sin_mejora}/(f \times T_{sin_mejora} + ((1-f) \times T_{sin_mejora})/p) = 1/(f + (1-f)/p) = p/(1+f(p-1))$$

y así llegamos a la expresión que se utiliza para describir la Ley de Amdahl: $S \leq p/(1+f(p-1))$



Cuestión 9. ¿Es cierto que para una determinada mejora realizada en un recurso se observa experimentalmente que, al aumentar el factor de mejora, llega un momento en que se satura el incremento de velocidad que se consigue? (Justifique la respuesta)

Solución: Para responder a esta pregunta recurrimos a la ley de Amdahl, que marca un límite superior a la mejora de velocidad. Como se tiene que $S \leq p/(1+f(p-1))$, donde p es el factor de mejora y f la fracción de tiempo sin la mejora en el que dicha mejora no puede aplicarse, se puede calcular la derivada de $p/(1+f(p-1))$ con respecto a p , y ver qué pasa cuando cambia p . Así:

$$\frac{d(p/(1+f(p-1)))/dp}{dp} = ((1+f(p-1))-pf)/(1+f(p-1))^2 = (1-f)/(1+f(p-1))^2$$

y como puede verse, a medida que aumenta p el denominador se va haciendo mayor, y como $(1-f)$ se mantiene fijo, la derivada tiende a cero. Eso significa que la pendiente de la curva que describe la variación de $p/(1+f(p-1))$ con p va haciéndose igual a cero y por lo tanto no aumenta la ganancia de velocidad: se satura, o lo que es lo mismo, llega a una asíntota que estará situada en $1/f$ (que es al valor al que tiende $p/(1+f(p-1))$ cuando p tiende a infinito).

Cuestión 10. ¿Es cierto que la cota para el incremento de velocidad que establece la ley de Amdahl crece a medida que aumenta el valor del factor de mejora aplicado al recurso? (Justifique la respuesta).

Solución: La respuesta a esta pregunta se deduce de la misma expresión de la derivada de la cota que establece la ley de Amdahl, calculada al responder la cuestión 4:

$$\frac{d(p/(1+f(p-1)))/dp}{dp} = ((1+f(p-1))-pf)/(1+f(p-1))^2 = (1-f)/(1+f(p-1))^2$$

dado que $(1-f)$ es positivo (solo en el peor de los casos, si no se pudiera aplicar el factor de ganancia a ninguna parte del programa, tendríamos que $(1-f)=0$ y no se obtendría ninguna ganancia de velocidad), y puesto que $(1+f(p-1))^2$ siempre es positivo, la derivada es positiva. Eso significa que $1+f(p-1)$ crece al crecer p . Lo que ocurre es que, tal y como se ha visto en la cuestión 4, este crecimiento es cada vez menor (tiende a 0).

Cuestión 11. ¿Qué es mejor, un procesador superescalar capaz de emitir cuatro instrucciones por ciclo, o un procesador vectorial cuyo repertorio permite codificar 8 operaciones por instrucción y emite una instrucción por ciclo? (Justifique su respuesta).

Solución. Considérese una aplicación que se codifica con N instrucciones de un repertorio de instrucciones escalar (cada instrucción codifica una operación). La cota inferior para el tiempo que un procesador superescalar podría tardar en ejecutar ese programa sería

$$T_{superescalar} = (N \times T_{ciclo_superesc})/4$$

Un procesador vectorial ejecutaría un número de instrucciones menor puesto que su repertorio de instrucciones permitiría codificar hasta ocho operaciones iguales (sobre datos diferentes) en una instrucción. Si suponemos que la aplicación permite que todas las operaciones que hay que ejecutar se puedan "empaquetar" en instrucciones vectoriales, el procesador vectorial tendría que ejecutar $N/8$ instrucciones. Por tanto, considerando también que el procesador vectorial consigue terminar instrucciones según su velocidad pico (como supusimos en el caso del superescalar), la cota inferior para el tiempo que tarda la ejecución del programa en el procesador vectorial sería:

$$T_{vectorial} = (N/8) \times T_{ciclo_vectorial}/1$$

Por tanto

$$T_{superescalar} / T_{vectorial} = 8 \times T_{ciclo_superesc} / 4 \times T_{ciclo_vectorial} = 2 \times T_{ciclo_superesc} / T_{ciclo_vectorial}$$

Según esto, si los dos procesadores funcionan a la misma frecuencia, el procesador vectorial podría ser mejor (siempre y cuando las hipótesis que se están considerando de que están procesando instrucciones según su máximo rendimiento se puedan considerar suficientemente aproximada a la realidad).

No obstante, hay que tener en cuenta que usualmente, los procesadores superescalares han sido los que más rápidamente han incorporado las mejoras tecnológicas, y normalmente, los procesadores superescalares funcionan a frecuencias más altas que los procesadores vectoriales contemporáneos. En el ejemplo, si la frecuencia del superescalar fuera más del doble de la del vectorial, sería mejor utilizar un procesador superescalar.

Además, hay que tener en cuenta que para que los procesadores vectoriales sean eficientes (se puedan admitir las hipótesis en cuanto a codificación de las operaciones y funcionamiento según la velocidad pico) es necesario que el programa tenga un alto grado de paralelismo de datos (son procesadores más específicos que los superescalares). Por eso, para dar una respuesta más precisa habría que conocer más detalles de la carga de trabajo que se piensa ejecutar en los procesadores.

Cuestión 12. En la Lección 2 de AC se han presentado diferentes criterios de clasificación de computadores y en el Seminario 0 de prácticas se ha presentado atcgrid. Clasifique atcgrid, sus nodos, sus encapsulados y sus núcleos dentro de la clasificación de Flynn y dentro de la clasificación que usa como criterio el sistema de memoria. Razone su respuesta.

Solución

Los nodos de atcgrid, atcgrid1, atcgrid2 y atcgrid3, están conectados entre sí mediante un conmutador Ethernet, esto supone que por hardware un nodo no puede acceder a la memoria que se encuentra en otros nodos. El software del sistema de comunicación ofrece funciones para copiar datos de la memoria de un nodo a la memoria de otro nodo. Por tanto, en el nivel de empaquetamiento/conexión de sistema, atcgrid es una arquitectura MIMD (clasificación de Flynn) y, en particular, un multicomputador (clasificación del sistema de memoria) o arquitectura NORMA.

Los dos nodos de atcgrid tienen una placa con dos socket (encapsulados) que comparten memoria, aunque cada socket tiene acceso mediante un enlace (conexión punto-a-punto dedicada exclusivamente a la comunicación entre esos puntos) a un conjunto de direcciones de memoria del sistema de memoria compartido. Por tanto, en el nivel de placa, atcgrid es una arquitectura MIMD (clasificación de Flynn) y, en particular, un multiprocesador CC-NUMA (clasificación del sistema de memoria). Es NUMA por tener el espacio de memoria compartido físicamente distribuido entre los sockets (encapsulados), lo que hace que un núcleo tarde menos en acceder a la memoria que está directamente conectada con un enlace al encapsulado en el que se encuentra que a la memoria conectada al otro encapsulado de la placa. Además el hardware mantiene coherencia entre las caches de último nivel de los encapsulados, de ahí que aparezca CC (*Cache-Coherent*) en el nombre.

Los encapsulados de atcgrid constan de un único dado de silicio con 6 cores/núcleos que comparten el último nivel de cache. Por tanto, en el nivel de encapsulado, atcgrid es una arquitectura MIMD (clasificación de Flynn) y, en particular, un multiprocesador UMA o SMP en un chip (clasificación del sistema de memoria), también llamado en la bibliografía científica CMP (*Chip MultiProcessor*). Es UMA porque la latencia de acceso al último nivel de memoria cache del encapsulado es igual para todos los cores.

Los cores/núcleos de atcgrid tiene repertorio de instrucciones vectoriales porque tiene unidades funcionales multimedia que permiten ejecutar en paralelo múltiples operaciones escalares (suma, and, multiplicación, etc.). Las unidades funcionales multimedia implementan una arquitectura SIMD (la memoria en la que están los datos con los que se opera es un registro). Además, cada núcleo de atcgrid ejecuta dos flujos de control en paralelo (dos threads del SO en paralelo) debido a la implementación de multithread simultáneo (*HyperThreading*). Desde el punto de vista del SO un núcleo de atcgrid son dos procesadores independientes (es como si el núcleo tuviera dos procesadores lógicos). Por tanto, desde el punto de vista del SO un núcleo tiene dos procesadores de una arquitectura MIIMD, aunque en realidad físicamente los dos flujos de control de un núcleo comparten una unidad de control y una unidad de procesamiento (como se verá en el Tema 3).

Cuestión 13. En la Lección 1 de AC se han presentado diferentes criterios de clasificación del paralelismo implícito en una aplicación y en el Seminario 0 de prácticas se ha presentado atcgrid. ¿Qué tipos de paralelismo aprovecha atcgrid? Razona su respuesta.

Solución

❖ Paralelismo a nivel de operación:

- Al ser una arquitectura con paralelismo a nivel de instrucción, por tener una arquitectura segmentada y superescalar, puede ejecutar operaciones en paralelo, luego puede aprovechar el paralelismo a nivel de operación.

❖ Paralelismo a nivel de bucle (paralelismo de datos):

- Las operaciones realizadas en las iteraciones del bucle sobre vectores y matrices se podrían implementar en las unidades SIMD de los cores de atcgrid, lo que reduciría el número de iteraciones de los bucles del código máquina.
- Las iteraciones de los bucles se podrían repartir entre threads del SO. Los cores/nucleos de atcgrid pueden ejecutar dos threads del SO en paralelo debido a la implementación de multithread simultánea (*HyperThreading*). Los threads de un mismo código también se pueden repartir entre los 6 cores de un encapsulado de atcgrid (6 cores * 2 threads/core = 12 threads) y entre los cores de los dos encapsulados de una placa atcgrid (2 encapsulados * 6 cores/encapsulado * 2 threads/core * = 24 threads). Por tanto, las iteraciones de un bucle se podrían repartir entre 24 threads del SO que se pueden ejecutar en paralelo en un nodo de atcgrid.
- Si las iteraciones de los bucles se reparten además entre procesos del SO se podrían aprovechar los cores de los dos nodos de atcgrid (2 nodos * 2 encapsulados/nodo * 6 cores/encapsulado * 2 threads/core * = 48 threads repartidos en dos procesos). En total se podrían ejecutar en paralelo 48 flujos de control repartidos entre un mínimo de dos procesos (uno por nodo).

❖ Paralelismo a nivel de función (paralelismo de tareas):

- Las funciones independientes se podrían ejecutar en paralelo usando los dos threads de los cores, usando todos los cores de los dos encapsulados (24 flujos de control en paralelo como máximo), e incluso, si el paralelismo se hace explícito a nivel de proceso, se podrían usar los dos nodos (48 flujos de control en paralelo como máximo).

❖ Paralelismo a nivel de programa:

- Los programas del mismo o distinto usuario se pueden asignar a distintos cores de atcgrid estén o no en el mismo nodo para así ejecutarlos al mismo tiempo (48 flujos de control en paralelo como máximo en atcgrid). El paralelismo a nivel de programa sólo se puede hacer explícito a nivel de proceso del SO.

PREGUNTAS AC TEMA 1 – CURSO 2019/2020

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos p puede ser mayor que 1.

RESPUESTA: V

2. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos, f es la fracción del tiempo antes de la mejora en la que se utiliza el recurso mejorado.

RESPUESTA: F

3. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos, p es el factor de incremento de prestaciones del recurso que se mejora.

RESPUESTA: V

4. Un multiprocesador puede funcionar como computador MISD con la sincronización adecuada entre sus procesadores.

RESPUESTA: V

5. En la secuencia de instrucciones:

- (a) $\text{add } r1,r2,r3 ; r1 \leftarrow r2 + r3$
- (b) $\text{sub } r1,r1,r4 ; r1 \leftarrow r1 - r4$

Hay dependencia WAW entre las instrucciones debido al registro r1.

RESPUESTA: V

6. En la secuencia de instrucciones:

- (a) $\text{add } r1,r2,r3 ; r1 \leftarrow r2 + r3$
- (b) $\text{sub } r1,r1,r4 ; r1 \leftarrow r1 - r4$

NO hay dependencia WAR entre las instrucciones debido al registro r1.

RESPUESTA: V

7. En la secuencia de instrucciones:

- (a) $\text{add } r1,r2,r3 ; r1 \leftarrow r2 + r3$
- (b) $\text{sub } r1,r1,r4 ; r1 \leftarrow r1 - r4$

Sólo hay dependencia RAW entre las instrucciones debido al registro r1.

RESPUESTA: F

8. En la secuencia de instrucciones que aparecen en el orden indicado en un código:

- (a) $\text{add } r1,r2,r4 ; r1 \leftarrow r2 + r4$
- (b) $\text{add } r4,r2,r3 ; r4 \leftarrow r2 + r3$
- (c) $\text{sub } r1,r1,r4 ; r1 \leftarrow r1 - r4$

Hay dependencia WAR entre las instrucciones i1 e i2 debido al registro r4.

RESPUESTA: V

9. En la secuencia de instrucciones que aparecen en el orden indicado en un código:

- (a) add r1,r2,r4 ; r1 ← r2 + r4
- (b) add r4,r2,r3 ; r4 ← r2 + r3
- (c) sub r1,r1,r4 ; r1 ← r1 – r4

Hay dependencia RAW entre las instrucciones i2 e i3 debido al registro r4.

RESPUESTA: V

10. En la secuencia de instrucciones que aparecen en el orden indicado en un código:

- (a) add r1,r2,r3 ; r1 ← r2 + r3
- (b) sub r1,r2,r4 ; r1 ← r2 - r4
- (c) add r3,r2,r1 ; r3 ← r2 + r1

El registro r1 solo genera una dependencia RAW

RESPUESTA: F

11. En la secuencia de instrucciones que aparecen en el orden indicado en un código:

- (a) add r1,r2,r3 ; r1 ← r2 + r3
- (b) sub r1,r2,r4 ; r1 ← r2 - r4
- (c) add r3,r2,r1 ; r3 ← r2 + r1

No hay dependencias debido al uso del registro r2

RESPUESTA: V

12. En la secuencia de instrucciones que aparecen en el orden indicado en un código:

- (a) add r1,r2,r3 ; r1 ← r2 + r3
- (b) sub r1,r2,r4 ; r1 ← r2 - r4
- (c) add r3,r2,r1 ; r3 ← r2 + r1

El registro r3 genera una dependencia WAW

RESPUESTA: F

13. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo. ¿Cuál es la velocidad pico (en GFLOPS) de un microprocesador con 4 núcleos Sunday Bridge que funciona a una frecuencia de reloj de 2GHz?

RESPUESTA: 64

14. Un computador NUMA, es un multiprocesador donde la memoria está físicamente distribuida.

RESPUESTA: V

15. En un computador NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida.

RESPUESTA: V

16. Si el bucle siguiente: **for i=1 to N do a(i) = b(i) * c;** se ejecuta en 2 segundos y N=10^11, siendo c, a(), y b() datos en coma flotante, ¿cuánto GFLOPS alcanza la máquina al ejecutar el código?

RESPUESTA: 50

17. Dado el bucle **for i=1 to N do a(i) = b(i) * c(i)** son números en coma flotante, ¿cuántos GFLOPS consigue un computador que lo ejecuta en 2 segundos cuando N=10^10?

RESPUESTA: 5

18. Dado el bucle **for i=1 to N do** $a(i) = b(i) * c(i)$ son números en coma flotante, ¿cuántos GFLOPS consigue un computador que lo ejecuta en 2 segundos cuando $N=10^{12}$?

RESPUESTA: 500

$$1(\text{operacion_flot}) * 10^{12} / ((2 \text{ s}) * 10^9) = 1000/2 = 500 \text{ GFLOPS}$$

19. Un cluster de computadores es un computador NUMA.

RESPUESTA: F

20. Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse entre sí.

RESPUESTA: F

21. En un procesador superescalar el valor de CPI puede ser menor que 1.

RESPUESTA: V

22. El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos.

RESPUESTA: V

23. Los multicomputadores son máquinas MIMD y los multiprocesadores SIMD.

RESPUESTA: F

24. En un computador de tipo NORMA tanto los accesos a memoria local como los de acceso a memoria remota se realizan a través de instrucciones de carga y almacenamiento de datos en memoria.

RESPUESTA: F

25. ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta dos instrucciones por ciclo y funciona a una frecuencia de reloj de 1GHz?

RESPUESTA: 2000

$$2 (\text{inst/ciclo}) * 1 * 10^9 (\text{ciclos/s}) * (1/10^6) = 2000 \text{ MIPS}$$

26. Un programa tiene 1000 millones de instrucciones y se ejecuta en un computador que tiene cinco tipos de instrucciones. Las del tipo 1 necesitan 6 ciclos, las del tipo 2 necesitan 4 ciclos, las del tipo 3 necesitan 3 ciclos, y las del tipo 4 necesitan 5 ciclos y las del tipo 5 necesitan 2. Si entre las instrucciones ejecutadas por el programa hay un 20% de instrucciones de cada uno de los tipos. ¿Cuántos segundos tarda el programa en ejecutarse en el computador si utiliza un reloj de 2GHz?

RESPUESTA: 2

$$\text{CPI} = 0.20 * (6+4+3+5+2) = (1/5) * 20 = 4 \text{ (ciclos / instrucción)}$$

$$\text{T}_{\text{(CPU)}} = \text{NI} * \text{CPI} * \text{Tciclo} = 10^9 \text{ (instrucciones)} * 4 \text{ (ciclos/instrucción)} * (1/2) * 10^9 \text{ (s/ciclo)} = 2 \text{ s}$$

27. Un procesador puede terminar hasta 4 operaciones en coma flotante por ciclo. ¿Cuál es su velocidad pico (en GFLOPS) si funciona a una frecuencia de reloj de 2GHz?

RESPUESTA: 8

$$\text{GFLOPS} = 4 \text{ op_float/ciclo} * (2*10^9) \text{ ciclos/s} * (1/10^9) = 8$$

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



28. Según la ley de Amdahl, la ganancia máxima de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/f$ (f fracción del tiempo de procesamiento en el computador base durante el que NO se puede aprovechar la mejora):

RESPUESTA: V

29. La comunicación entre procesadores en un computador UMA se realiza a través de escrituras y lecturas en la memoria compartida, igual que en un computador NUMA:

RESPUESTA: V

30. Escriba la expresión del tiempo de CPU (Tcpu) en términos del número de instrucciones ejecutadas (NI), el número medio de ciclos por instrucción (CPI) y la frecuencia de reloj (F):

RESPUESTA: $T_{cpu} = NI \cdot CPI / F$

31. ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta cuatro instrucciones por ciclo y funciona a una frecuencia de reloj de 3 GHz?

RESPUESTA: 12000

$$\text{MIPS} = 4 \text{ int/ciclo} * (3 * 10^9) \text{ ciclos/s} * (1/10^6) = 12000$$



EJERCICIO 1. Al extraer paralelismo para una aplicación se ha llegado a un grafo de tareas segmentado para la aplicación de 4 etapas. La primera, segunda y tercera suponen cada una 2s, y la cuarta, 1s. Teniendo en cuenta que las etapas no se pueden subdividir en tareas independientes y que la sobrecarga es despreciable. ¿Cuánto tardaría el cauce en ejecutar 10 entradas en paralelo? ¿Con qué número mínimo de procesadores obtendría la ganancia máxima en velocidad (ganancia para un número elevado de entradas al cauce)? [¿Ganancia máxima en velocidad?]

SOLUCIÓN: Disponemos de los siguientes datos:

- Duración de las etapas [1,2,3] ---> 2s
- Duración de la etapa [4] ---> 1s
- Tenemos 10 entradas en paralelo

El tiempo de procesamiento secuencial será igual a la suma del tiempo total que se tarda en ejecutar todas las etapas multiplicado por el número de entradas que tenemos, esto es:

$$T_s(n) = n \cdot \sum_{i=1}^4 T_i = n \cdot (2 + 2 + 2 + 1) = 7n$$

El tiempo de procesamiento en paralelo atiende al tiempo que se tarda en ejecutar en total la primera entrada más la suma del tiempo máximo de ejecución del conjunto de etapas que tenemos multiplicado por (n-1) entradas. En nuestro ejemplo, esto es:

$$T_p(n) = 7 + 2 \cdot (n - 1) = 7 + 2n - 2 = 5 + 2n$$

El tiempo de procesamiento en paralelo para 10 entradas sería:

$$T_p(10) = 7 + 2 \cdot (10 - 1) = 7 + 2 \cdot 10 - 2 = 5 + 2 \cdot 10 = 25s$$

La ganancia máxima en velocidad viene expresada por el límite cuando n (número de procesadores) tiende a infinito del cociente del tiempo secuencial y el tiempo paralelo.

$$S(n) = \frac{T_s(n)}{T_p(n)} = \frac{7n}{5+2n} \Rightarrow \lim_{n \rightarrow \infty} S(n) = \frac{7}{2}$$

EJERCICIO 2. Al extraer paralelismo para una aplicación se ha llegado a un grafo de tareas segmentado para la aplicación de 4 etapas. La primera, segunda y tercera suponen cada una 1s, y la cuarta, 2s. Teniendo en cuenta que las etapas no se pueden subdividir en tareas independientes y que la sobrecarga es despreciable. ¿Cuánto tardaría el cauce en ejecutar 10 entradas en paralelo? ¿Con qué número mínimo de procesadores obtendría la ganancia máxima en velocidad (ganancia para un número elevado de entradas al cauce)? [¿Ganancia máxima en velocidad?]

SOLUCIÓN: Disponemos de los siguientes datos:

- Duración de las etapas [1,2,3] ---> 1s
- Duración de la etapa [4] ---> 2s
- Tenemos 10 entradas en paralelo

El tiempo de procesamiento secuencial será igual a la suma del tiempo total que se tarda en ejecutar todas las etapas multiplicado por el número de entradas que tenemos, esto es:

$$T_s(n) = n \cdot \sum_{i=1}^4 T_i = n \cdot (1 + 1 + 1 + 2) = 5n$$

El tiempo de procesamiento en paralelo atiende al tiempo que se tarda en ejecutar en total la primera entrada más la suma del tiempo máximo de ejecución del conjunto de etapas que tenemos multiplicado por (n-1) entradas. En nuestro ejemplo, esto es:

$$T_p(n) = 5 + 2 \cdot (n - 1) = 5 + 2n - 2 = 3 + 2n$$

El tiempo de procesamiento en paralelo para 10 entradas sería:

$$T_p(10) = 23s$$

La ganancia máxima en velocidad viene expresada por el límite cuando n (número de procesadores) tiende a infinito del cociente del tiempo secuencial y el tiempo paralelo.

$$S(n) = \frac{T_s(n)}{T_p(n)} = \frac{5n}{3+2n} \Rightarrow \lim_{n \rightarrow \infty} S(n) = \frac{5}{2}$$

EJERCICIO 3. Al extraer paralelismo para una aplicación se ha llegado a un grafo para la aplicación en forma de árbol binario que tiene un grado de paralelismo de 8. Teniendo en cuenta que cada comunicación entre tareas supone un tiempo de 0.25s y que se puede hacer en paralelo a cualquier otra. ¿Cuánto supone en segundos el tiempo de comunicación en el tiempo de ejecución paralelo total?

SOLUCIÓN: Disponemos de los siguientes datos:

- Grado de paralelismo = 8
- Cada tarea es de 0.25s.
- Nos preguntan por T_p .

Tenemos $k=3$ niveles de comunicación, con $k=\log_2(p)$. Y por tanto, en paralelo podemos comunicar desde una tarea a cualquier otra:

$$T_p = 3 \cdot 0.25s = 0.75s$$

EJERCICIO 4. Al extraer paralelismo para una aplicación se ha llegado a un grafo de tareas para la aplicación en forma de árbol binario que tiene un grado de paralelismo de 8. Teniendo en cuenta que cada tarea del grafo supone 1s y que se desprecia sobrecarga. ¿Cuánto tiempo supone la ejecución en paralelo si se dispone de 4 procesadores?

SOLUCIÓN: Disponemos de los siguientes datos:

- Grado de paralelismo = 8
- Cada tarea es de 1s.
- Nos preguntan por $T_p(4)$.

Sabemos que el tiempo en ejecución en paralelo de las tres primeras etapas es:

$$T_p^{1,2,3} = 3 \cdot 1s = 3s$$

Como en la etapa 4, grado par > procesadores. Entonces $8/4 = 2$ tareas/procesador y en conclusión:

$$T_p^4 = 2 \cdot 1s$$

Finalmente, la ejecución total es: $3s + 2s = 5s$

EJERCICIO 5. Al extraer paralelismo para una aplicación se ha llegado a un grafo de tareas para la aplicación en forma de árbol binario que tiene un grado de paralelismo de 8. Teniendo en cuenta que cada tarea del grafo supone 2s y que se desprecia la sobrecarga. ¿Cuánto tiempo supone la ejecución en paralelo si se dispone de 8 procesadores?

SOLUCIÓN: Disponemos de los siguientes datos:

- Grado de paralelismo = 8
- Cada tarea es de 1s.
- Nos preguntan por $T_p(8)$.

El tiempo de ejecución en paralelo es:

$$T_p^{1,2,3} = 4 \cdot 2s = 8s$$

EJERCICIO 6. Un programa tarda 10s en ejecutarse en un procesador. Se ha comprobado que durante un 50% de ese tiempo se puede ejecutar en 8 procesadores, durante un 40% de ese tiempo se puede ejecutar en 4 procesadores, y durante el 10% restante en uno. ¿Cuál es la eficiencia obtenida al paralelizar? ¿Cuál es la ganancia en velocidad obtenida?

SOLUCIÓN: Disponemos de los siguientes datos:

(Llamando T_s al tiempo secuencial y T_p al tiempo en paralelo de ejecución)

- En 1 procesador se tarda 10 segundos en ejecutar todo el código.
- En 8 procesadores -> 50% del código.
- En 4 procesadores -> 40% del código.
- En 1 procesador -> 10% del código.

El último dato nos indica que hay un 10% del código que no es paralelizable. Para obtener el tiempo que se tarda en paralelo simplemente calculamos:

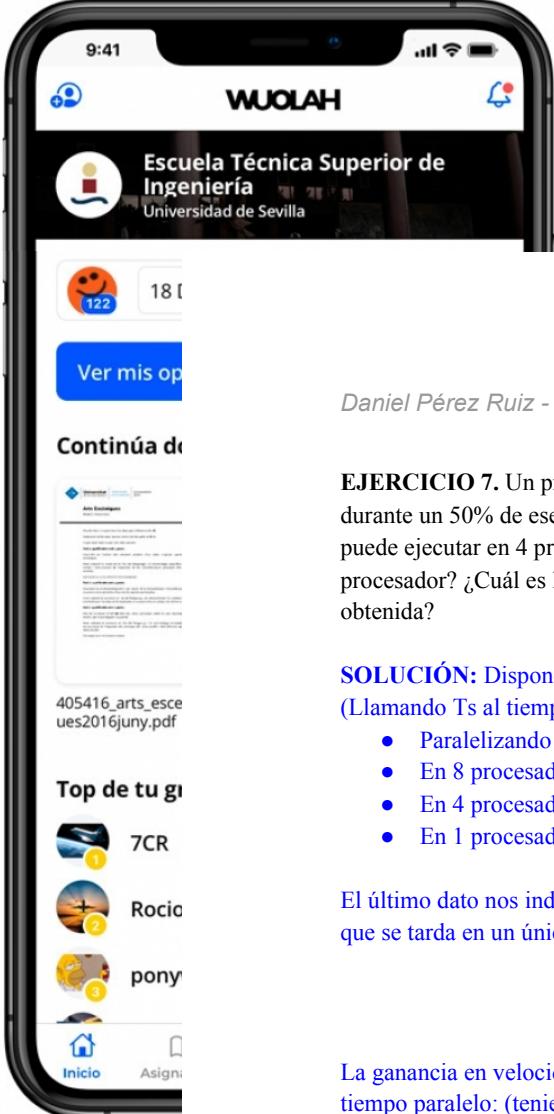
$$T_p = \frac{0.5 \cdot T_s}{8} + \frac{0.4 \cdot T_s}{4} + 0.1 \cdot T_s = 10 \cdot \left(\frac{0.5}{8} + \frac{0.4}{4} + 0.1 \right) = 2.625s$$

La ganancia en velocidad obtenida viene expresada por el cociente del tiempo secuencial entre el tiempo paralelo: (teniendo en cuenta que el número de procesadores que tenemos es 8)

$$S(8) = \frac{T_s}{T_p} = \frac{10}{2.625} \approx 3.81$$

Para finalizar, la eficiencia viene dada por el cociente de la ganancia en velocidad entre el número de procesadores.

$$E(8) = \frac{S(8)}{8} = \frac{3.81}{8} \approx 0.48$$



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Daniel Pérez Ruiz - EJERCICIOS T2 - ARQUITECTURA DE COMPUTADORES

EJERCICIO 7. Un programa tarda 10s en ejecutarse en un multiprocesador. Se ha comprobado que durante un 50% de ese tiempo se puede ejecutar en 8 procesadores, durante un 40% de ese tiempo se puede ejecutar en 4 procesadores, y durante el 10% restante en uno. ¿Tiempo en ejecutarse en un sólo procesador? ¿Cuál es la eficiencia obtenida al paralelizar? ¿Cuál es la ganancia en velocidad obtenida?

SOLUCIÓN: Disponemos de los siguientes datos:

(Llamando T_s al tiempo secuencial y T_p al tiempo en paralelo de ejecución)

- Paralelizando el código se tarda 10s.
- En 8 procesadores \rightarrow 50% del código.
- En 4 procesadores \rightarrow 40% del código.
- En 1 procesador \rightarrow 10% del código.

El último dato nos indica que hay un 10% del código que no es paralelizable. Para obtener el tiempo que se tarda en un único procesador, calculamos:

$$T_s = T_p \cdot (0.5 \cdot 8 + 0.4 \cdot 4 + 0.1) = 10 \cdot 5.7 = 57$$

La ganancia en velocidad obtenida viene expresada por el cociente del tiempo secuencial entre el tiempo paralelo: (teniendo en cuenta que el número de procesadores que tenemos es 8)

$$S(8) = \frac{T_s}{T_p} = \frac{57}{10} \approx 5.7$$

Para finalizar, la eficiencia viene dada por el cociente de la ganancia en velocidad entre el número de procesadores.

$$E(8) = \frac{S(8)}{8} = \frac{5.7}{8} \approx 0.7125$$

EJERCICIO 8. Un programa tarda 10s en ejecutarse en un multiprocesador. Se ha comprobado que durante un 40% de ese tiempo se puede ejecutar en 5 procesadores, durante un 50% de ese tiempo se puede ejecutar en 3 procesadores, y durante el 10% restante en uno. ¿Tiempo en ejecutarse en un sólo procesador? ¿Cuál es la eficiencia obtenida al paralelizar? ¿Cuál es la ganancia en velocidad obtenida?

SOLUCIÓN: Disponemos de los siguientes datos:

(Llamando T_s al tiempo secuencial y T_p al tiempo en paralelo de ejecución)

- Paralelizando el código se tarda 10s.
- En 5 procesadores \rightarrow 50% del código.
- En 3 procesadores \rightarrow 40% del código.
- En 1 procesador \rightarrow 10% del código.

El último dato nos indica que hay un 10% del código que no es paralelizable. Para obtener el tiempo que se tarda en un único procesador, calculamos:

$$T_s = T_p \cdot (0.5 \cdot 5 + 0.4 \cdot 3 + 0.1) = 10 \cdot 3.6 = 36s$$

La ganancia en velocidad obtenida viene expresada por el cociente del tiempo secuencial entre el tiempo paralelo: (teniendo en cuenta que el número de procesadores que tenemos es 5)

$$S(5) = \frac{T_s}{T_p} = \frac{36}{10} \approx 3.6$$

Para finalizar, la eficiencia viene dada por el cociente de la ganancia en velocidad entre el número de procesadores.

$$E(5) = \frac{S(5)}{5} = \frac{3.6}{5} \approx 0.72$$

EJERCICIO 9. Se dispone de un computador con 3 procesadores P1,P2,P3. El tiempo secuencial de un programa es de 1s con P1, 2s con P2 y 3s con P3. Calcular qué fracción de código le tiene que asignar a cada uno de los procesadores teniendo en cuenta que la sobrecarga es despreciable y que el código se puede partir sin limitaciones. Escribir la fracción para P2.

SOLUCIÓN: Disponemos de los siguientes datos:

- Tiempo secuencial de ejecución en P1: 1s.
- Tiempo secuencial de ejecución en P2: 2s.
- Tiempo secuencial de ejecución en P3: 3s.

Para asignar debidamente qué trozo de código le corresponde a cada procesador lo que tenemos que tener en cuenta es que:

$$T_p^{P1} = T_p^{P2} = T_p^{P3}$$

De donde deducimos en primer lugar, para P1 y P2 que:

$$1 \cdot f = (1 - f) \cdot 2 \rightarrow f = 2 - 2f \rightarrow f = \frac{2}{3}$$

Y por tanto:

$$P1 \rightarrow 2/3 \cdot 1s = 2/3s$$

$$P1 \rightarrow 1/3 \cdot 2s = 2/3s$$

Ahora, con respecto P3:

$$T_p^{P1, P2} \cdot f = T_p^{P3} \cdot (1 - f) \rightarrow \frac{2}{3}f = 3 \cdot (1 - f) \rightarrow f = \frac{9}{11}$$

Y por tanto:

$$T_p^{P1} = 1s \cdot 2/3 \cdot 9/11 = 6/11s \rightarrow f_{P1} = 6/11$$

$$T_p^{P2} = 2s \cdot 1/3 \cdot 9/11 = 6/11s \rightarrow f_{P2} = 3/11$$

$$T_p^{P3} = 3s \cdot 2/11 = 6/11s \rightarrow f_{P3} = 2/11$$

EJERCICIO 10. Se dispone de un computador con 2 procesadores P1, P2. El tiempo secuencial de un programa es de 1s con P1 y 0.5s con P2. Calcular qué fracción de código le tiene que asignar a cada uno de los procesadores teniendo en cuenta que la sobrecarga es despreciable y que el código se puede partir sin limitaciones. Escribir la fracción para P1.

SOLUCIÓN: Disponemos de los siguientes datos:

- Tiempo secuencial de ejecución en P1: 1s.
- Tiempo secuencial de ejecución en P2: 0.5s.

Para obtener la fracción assignable a cada uno de los procesadores montamos la siguiente ecuación en la que la incógnita “x” será la fracción buscada:

$$x \cdot P1 = (1 - x) \cdot P2$$

$$x \cdot P1 = P2 - P2x \Leftrightarrow P2x + x = P2 \Leftrightarrow x \cdot (P2 + 1) = P2$$

$$x = \frac{P2}{P2+1} = \frac{0.5}{1.5} = \frac{1}{3}$$

EJERCICIO 11. Un 10% de un programa no se puede parallelizar, el resto se puede distribuir por igual entre cualquier número de procesadores. ¿A partir de cuál número de procesadores se podrían conseguir ganancias mayores o iguales que 4?

SOLUCIÓN: Se nos pregunta el número de procesadores (p) necesarios tal que la ganancia producida por ejecutar el código en paralelo sea mayor o igual que cuatro. En primer lugar, hay que calcular el tiempo en paralelo de ejecución sabiendo que el 10% no es paralelizable.

$$T_p = 0.1 \cdot T_s + 0.9 \cdot \frac{T_s}{p} \rightarrow S(p) = \frac{T_s}{T_p} = \frac{T_s}{0.1 \cdot T_s + 0.9 \cdot \frac{T_s}{p}} = \frac{1}{0.1 + \frac{0.9}{p}}$$

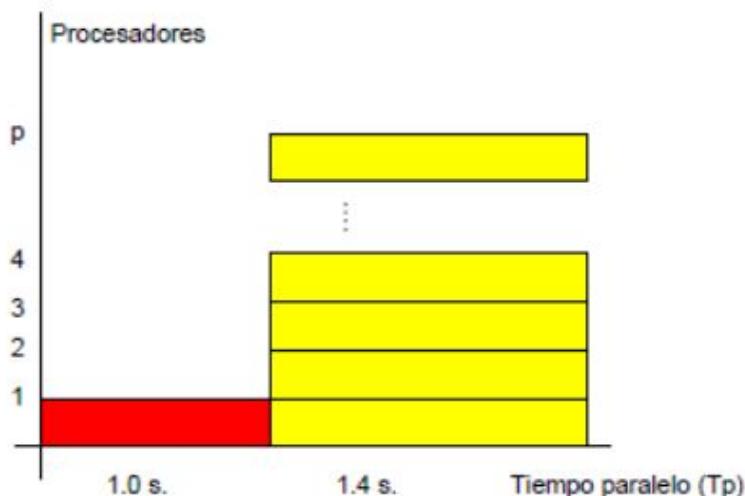
$$\text{Si } S(p) \geq 4 \Leftrightarrow 1 \geq 0.4 + \frac{3.6}{p} \Leftrightarrow 0.6p \geq 3.6 \Leftrightarrow p \geq 6$$

En conclusión, el número mínimo de procesadores para obtener una ganancia igual o superior a 4 es 6.

EJERCICIO 12. Escriba la expresión de la Ley de Gustafson en términos de los parámetros f y p :

SOLUCIÓN: $S_p = f + (1-f)p$

EJERCICIO 13. Teniendo en cuenta la siguiente figura:



¿Qué valor tiene el parámetro f en la ley de Gustafson?

SOLUCIÓN: $f_g = 1.0 / 2.4$

Escriba el valor del parámetro f en la ley de Amdahl:

SOLUCIÓN: $f_a = 1.0 / (1.0 + 1.4p)$



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play



122

l

Ver mis op

Daniel Pérez Ruiz - EJERCICIOS T2 - ARQUITECTURA DE COMPUTADORES

Continúa d



405416_arts_esce_ues2016juniy.pdf

Top de tu g



7CR



Rocio



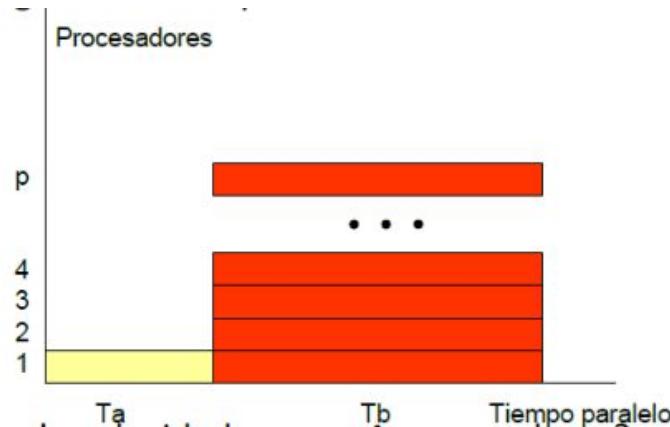
pony



Inicio



Asigni



¿Qué valor tiene la ganancia de velocidad para $p=4$ procesadores?

SOLUCIÓN: Primero calculamos el tiempo secuencial:

$$T_s = T_a + p * T_b = 10 + 4 * 30 = 130; T_p = 10 + 30 = 40s$$

$$S = T_s/T_p = 13/4$$

¿Cuál es el valor de la f de la ley de Gustafson?

SOLUCIÓN: $f = T_a/(T_a + T_b) = 10/(10 + 30) = 1/4$

1. Se ha conseguido dividir un trozo de código en 5 tareas independientes que se pueden ejecutar en paralelo de 1s, 2s, 3s, 4s, y 5s. ¿Cuál sería el menor tiempo de ejecución paralela de este trozo de código usando 5 procesadores? (se desprecia sobrecarga)

RESPUESTA : 5

2. Se ha conseguido dividir un trozo de código en 5 tareas independientes que se pueden ejecutar en paralelo de 1s, 2s, 3s, 4s y 5s. ¿Qué número mínimo de procesadores utilizaría para conseguir un equilibrado de la carga perfecto? (se desprecia sobrecarga)

RESPUESTA : 3



3. La siguiente expresión se usa para representar la Ley de Gustafson (ganancia escalable): $f + (1-f)p$ donde f es la fracción del tiempo de ejecución secuencial que supone la parte no paralelizable y p es el número de procesadores.

RESPUESTA: FALSO

4. OpenMP es una herramienta de programación paralela basada en paso de mensajes.

RESPUESTA : FALSO

5. OpenMP es una herramienta de programación paralela basada en directivas del compilador y funciones de una biblioteca.

RESPUESTA : VERDADERO

6. OpenMP permite aprovechar el paralelismo de datos usando directivas.

RESPUESTA : VERDADERO

7. OpenMP permite aprovechar el paralelismo a nivel de bucle usando directivas.

RESPUESTA : VERDADERO

8. OpenMP permite aprovechar el paralelismo a nivel de tarea usando directivas.

RESPUESTA : VERDADERO

9. OpenMP es una herramienta de programación paralela basada en variables compartidas.

RESPUESTA : VERDADERO

10. OpenMP tiene implementada con una cláusula la función de comunicación colectiva de reducción.

RESPUESTA : VERDADERO

11. La expresión que caracteriza la Ley de Amdahl es: $p / (1+f(p-1))$ donde f es la fracción del tiempo de ejecución secuencial que supone la parte no paralelizable y p es el número de procesadores.

RESPUESTA : VERDADERO

12. La expresión que caracteriza la Ley de Amdahl es $p/(1+f(1-p))$ donde f es la fracción del tiempo de ejecución secuencial que supone la parte no paralelizable y p es la ganancia máxima que se podría obtener si se pudiera paralelizar todo el código distribuyendo por igual las cargas entre los procesadores disponibles.

RESPUESTA : VERDADERO

13. La siguiente expresión se usa para representar la Ley de Gustafson (ganancia escalable): $n(1-x) + n$ donde x es la fracción del tiempo de ejecución paralelo que tarda en ejecutarse la parte no paralelizable y n es el número de procesadores.

RESPUESTA : FALSO

14. ¿Qué función de comunicación colectiva se puede utilizar para calcular el producto escalar de dos vectores? (Escriba una palabra)

RESPUESTA: Reducción

15. Reducción es una función de comunicación colectiva en la que los n flujos de instrucciones que colaboran en la ejecución paralela, F_j ($j=0, \dots, n-1$), envía datos (x_j para F_i) y un único flujo de instrucciones, F_i , recibe el resultado de reducir los datos enviados (x_0, x_1, \dots, x_{n-1}) a un único valor que usando una operación conmutativa y asociativa.

RESPUESTA : VERDADERO

16. Reducción es una función de comunicación colectiva en la que los n flujos de instrucciones que colaboran en la ejecución paralela F_j ($j=0, \dots, n-1$) envían datos x_j y un único flujo de instrucciones F_i , recibe concatenados en memoria los datos enviados (x_0, x_1, \dots, x_{n-1}).

RESPUESTA: FALSO

17. Para deducir la expresión que representa la ganancia escalable (o Ley de Gustafson) se usa un modelo de código secuencial en el que hay una parte no paralelizable y otra paralelizable que se puede repartir entre los procesadores disponibles de forma equilibrada y cuyo tiempo de ejecución secuencial se mantiene constante conforme se incrementa el número de procesadores.

RESPUESTA: FALSO

18. Para deducir la expresión que representa la ganancia escalable (o Ley de Gustafson) se usa un modelo de código paralelo en el que hay (1) una parte no paralelizable y (2) otra paralelizable que se reparte entre los procesadores de forma equilibrada y cuyo tiempo de ejecución paralelo no va a cambiar conforme se incrementa el número de procesadores.

RESPUESTA: VERDADERO

19. Todos dispersan es una función de comunicación colectiva en la que un flujo de instrucciones F_j reparte datos (x_0, x_1, \dots, x_{n-1}) entre los n flujos de instrucciones que colaboran en la ejecución paralela, de forma que al final x_i acaba en F_i .

RESPUESTA: VERDADERO

20. Con asignación estática, el coste en tiempo de la penalización (sobrecarga) es menor que con una asignación dinámica.

RESPUESTA: VERDADERO

21. Con asignación estática la asignación de tareas a flujos de instrucciones puede cambiar en distintas ejecuciones, aunque no varíe el número de procesadores ni el de tareas.

RESPUESTA: FALSO

22. Con asignación dinámica la asignación de tareas a flujos de instrucciones puede cambiar en distintas ejecuciones, aunque no varíe el número de procesadores ni el de tareas.

RESPUESTA: VERDADERO

23. Una herramienta de programación paralela podría realizar la asignación de la carga de trabajo a los flujos de instrucciones usando una asignación estática.

RESPUESTA: VERDADERO

24. La eficiencia permite evaluar en qué medida las prestaciones que se consiguen al paralelizar usando p procesadores se acercan a las prestaciones máximas que cabría esperar con p procesadores.

RESPUESTA: VERDADERO

25. Se dispone de un computador con dos procesadores distintos, P1 Y P2. El tiempo secuencial de un programa es de 1s si se usa P1 y 0.5s si se usa P2. Calcular qué fracción de código se tiene que asignar a cada uno de los procesadores para obtener el menor tiempo de ejecución teniendo en cuenta que la sobrecarga es despreciable, y que el código se puede partir sin limitaciones. Escribir la fracción para P1 (Formato: numerador/denominador, sin espacios en blanco al principio, al final, ni entre los números y “/”) [PREGUNTA CON CÁLCULOS EN FOLIO]

RESPUESTA : 1/3

TP1 = 1s ---> TP1 = 2TP2 ---> x·0.5 = (1-x) ---> 1.5·x = 1 ---> x = 2/3 para P2.

TP2 = 0.5s

33% a P1 – 66% a P2

26. La difusión (broadcast) implica comunicación colectiva de todos-con-todos.

RESPUESTA: FALSO

27. La difusión (broadcast) implica comunicación colectiva de todos-a-todos.

RESPUESTA: FALSO

28. La dispersión (scatter) implica comunicación colectiva todos-con-todos.

RESPUESTA: FALSO

29. La dispersión (scatter) implica comunicación colectiva de todos-a-uno.

RESPUESTA: FALSO

30. La reducción implica comunicación colectiva todos-a-uno.

RESPUESTA: VERDADERO

31. La acumulación (gather) implica comunicación colectiva todos-con-todos.

RESPUESTA: FALSO

32. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, Cosa que también OCURRE en la comunicación gossiping.

RESPUESTA: VERDADERO

33. En la comunicación colectiva de tipo gossiping todos los procesadores envían información, pero no todos los procesadores reciben.

RESPUESTA: FALSO

34. OpenMP es una biblioteca que permite hacer programas paralelos con el paso de mensajes.

RESPUESTA: FALSO

35. MPI es una biblioteca de paso de mensajes.

RESPUESTA: VERDADERO

36. El tiempo de sincronización entre procesos forma parte del overhead de un programa paralelo.

RESPUESTA: VERDADERO

37. El tiempo de comunicación entre procesos forma parte del overhead de un programa paralelo.

RESPUESTA: VERDADERO

38. La asignación de carga dinámica no tiene nunca ningún coste en el momento de la ejecución.

RESPUESTA: FALSO

39. La asignación de carga dinámica AFECTA al tiempo de overhead del programa paralelo.

RESPUESTA: VERDADERO

40. En la asignación de carga estática se asigna el trabajo que va a realizar cada procesador, antes de la ejecución.

RESPUESTA: VERDADERO

41. Para equilibrar la carga asociada a los procesadores interesa asignar más carga a los procesadores más rápidos.

RESPUESTA: VERDADERO

42. En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) prefijo paralelo, el procesador P2 recibe información de los procesadores P0, P1, y del propio P2 (aparte de otras posibles comunicaciones).

RESPUESTA: VERDADERO

43. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a 0 es igual a p para $p < n$.

RESPUESTA: VERDADERO

44. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a p es $T_s / ((T_s/n) + n)$, para $p = n$.

RESPUESTA: VERDADERO

PANDURA



MOTIVACIÓN

+



CONCENTRACIÓN

+



MEMORIA

PANDURA



"Sin Pandora,
juegas en
desventaja."



"Memorizo un
50% más en el
mismo tiempo."



MELEENA DE LEÓN



BACOPA MONNIERI



CAMELLIA SINENSIS



GINGKO BILOBA



HUPERZIA SERRATA



COFFEA ARABICA

Recarga tu energía sobre el fondo de colores negros.

Daniel Pérez Ruiz - PREGUNTAS T2 - ARQUITECTURA DE COMPUTADORES

45. La falta de equilibrado de la carga es una de las causas de que haya tiempo de sobrecarga u overhead en los programas paralelos.

RESPUESTA: VERDADERO

46. La expresión para la ley de Gustafson es $S=(1-f)+p*f$, donde f es la fracción no paralelizable del tiempo de ejecución paralelo y p es el número de procesadores que intervienen.

RESPUESTA: FALSO

47. En un multicomputador con 4 procesadores (P0 a P3), mediante la permutación de rotación, el procesador P0 envía información al procesador P1 y recibe del P2 (aparte de otras posibles comunicaciones).

RESPUESTA: FALSO

48. Un programa secuencial tarda 40 ns en ejecutarse en un procesador y durante 10 ns de esos 40 ns el programa no es paralelizable. El valor de la f de la ley de Amdahl para ese programa es igual a 0.75.

RESPUESTA: FALSO

49. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo Ts en ejecutarse en un procesador, con una fracción no paralela de Ts igual a f, un grado de paralelismo ilimitado y un tiempo de overhead igual a 0 es $p/(1+f(p-1))$.

RESPUESTA: VERDADERO

50. Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es 4.

RESPUESTA: FALSO

51. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo Ts en ejecutarse en un procesador, con una fracción no paralela de Ts igual a 0, un grado de paralelismo ilimitado y un tiempo de overhead igual a p^2 es $Ts/((Ts/p)+p^2)$.

RESPUESTA: VERDADERO

Nota: A partir de aquí es posible que haya alguna pregunta repetida.

52. OpenMP es una biblioteca que permite hacer programas paralelos con paso de mensajes.

RESPUESTA: FALSO

53. MPI es una biblioteca de paso de mensajes.

RESPUESTA: VERDADERO

54. El tiempo de comunicación entre procesos forma parte del overhead de un programa paralelo.

RESPUESTA: VERDADERO

55. El tiempo de sobrecarga u overhead es un componente del tiempo de procesamiento paralelo junto con el tiempo de comunicación

RESPUESTA: FALSO

56. La asignación de carga dinámica afecta al tiempo de overhead del programa paralelo

RESPUESTA: VERDADERO

57. La asignación de carga dinámica se realiza antes de la ejecución del programa paralelo

RESPUESTA: FALSO

58. La asignación de carga dinámica no tiene ningún coste en el momento de la ejecución

RESPUESTA: FALSO

59. En la asignación de carga estática se asigna el trabajo que ca a realizar cada procesador, antes de la ejecución

RESPUESTA: VERDADERO

60. Para equilibrar la carga asignada a los procesadores interesa asignar más carga a los procesadores más rápidos.

RESPUESTA: VERDADERO

61. La falta de equilibrado de la carga es una de las causas de que haya tiempo de sobrecarga u overhead en los programas paralelos

RESPUESTA: VERDADERO

62. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, cosa que no ocurre en la comunicación gossiping

RESPUESTA: FALSO

63. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, cosa que también ocurre en la comunicación gossiping

RESPUESTA: VERDADERO

64. En la comunicación colectiva de tipo gossiping todos los procesadores envían información, pero no todos los procesadores reciben

RESPUESTA: FALSO

65. La acumulación (gather) implica comunicación colectiva de todos-con-todos

RESPUESTA: FALSO

66. La acumulación (gather) es un modo de comunicación colectiva en el que todos los procesadores envían información a uno de ellos

RESPUESTA: VERDADERO

67. La difusión (broadcast) implica comunicación colectiva de todos-con-todos

RESPUESTA: FALSO

68. La dispersión (scatter) implica comunicación colectiva todos-con-todos

RESPUESTA: FALSO

69. La dispersión (scatter) implica comunicación colectiva todos-a-uno

RESPUESTA: FALSO

70. Tanto la difusión (broadcast) como la dispersión (scatter) implican comunicación de un procesador a todos los demás

RESPUESTA: VERDADERO

71. La reducción implica comunicación colectiva todos-a-uno

RESPUESTA: VERDADERO

72. En un multicomputador con 4 procesadores (P0 a P3), mediante la permutación de rotación, el procesador P0 envía información al procesador P1 y recibe del P2 (aparte de otras posibles comunicaciones)

RESPUESTA: FALSO

73. En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) prefijo paralelo, el procesador P2 recibe información de los procesadores P0 , y del propio P2 (aparte de otras posibles comunicaciones)

RESPUESTA: VERDADERO

74. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo Ts en ejecutarse en un procesador, con una fracción no paralela de Ts igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a 0 es igual a p/n

RESPUESTA: VERDADERO

75. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo Ts en ejecutarse en un procesador, con una fracción no paralela de Ts igual a f, un grado de paralelismo ilimitado y un tiempo de overhead igual a 0 es $p/(1+f(p-1))$

RESPUESTA: VERDADERO

76. Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de cargados). El valor de la f de la ley de Gustafson es 0.5.

RESPUESTA: VERDADERO

77. La expresión para la ley de Gustafson es $S=f+p*(1-f)$, donde f es la fracción no paralelizable del tiempo de ejecución paralelo y p es el número de procesadores que intervienen.

RESPUESTA: VERDADERO

78. La expresión para la ley de Gustafson es $S=(1-f)+p+f$, donde f es la fracción no paralelizable del tiempo de ejecución paralelo y p es el número de procesadores que intervienen.

RESPUESTA: FALSO

79. Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de

cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es 3.

RESPUESTA: VERDADERO

80. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo ilimitado y un tiempo de overhead igual a p^2 es $T_s/((T_s/p)+p^2)$.

RESPUESTA: VERDADERO

81. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a p es $T_s/((T_s/n)+n)$, para $p=n$.

RESPUESTA: VERDADERO

82. Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es 4

RESPUESTA: FALSO

83. Un programa secuencial tarda 40 ns en ejecutarse en un procesador y durante 10 ns de esos 40 ns el programa no es paralelizable. El valor de la f de la ley de Amdahl para ese programa es igual a 0.75.

RESPUESTA: FALSO

84. En la expresión de la ganancia de velocidad, $S=T_s/T_P$, el tiempo de computación paralelo, T_P , se obtiene sumando el tiempo de cálculo paralelo más el tiempo de sobrecarga u overhead, más el tiempo de comunicación.

RESPUESTA: FALSO

85. Un programa paralelo tarda 50 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 40 ns intervienen 6 procesadores (todos ellos igual de cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es 5

RESPUESTA: VERDADERO

86. En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) sufijo paralelo, el procesador P2 envía información los procesadores P0, P1, y al propio P2 (aparte de otras posibles comunicaciones)

RESPUESTA: VERDADERO

87. La expresión para la ley de Gustafson es $S=f+p*(1-f)$, donde f es la fracción no paralelizable del tiempo de ejecución secuencial y p es el número de procesadores que intervienen.

RESPUESTA: FALSO

88. Un programa paralelo tarda 40 ns en ejecutarse en un procesador y durante 10 ns de esos 40 ns el programa no es paralelizable, mientras que en el resto del tiempo paralelo intervienen cinco procesadores cargados por igual. El valor de la f de la ley de Gustafson para ese programa es igual a 0.25.

RESPUESTA: VERDADERO

PANDURA



MOTIVACIÓN



CONCENTRACIÓN



MEMORIA

PANDURA



"Sin Pandora,
juegas en
desventaja."

Daniel Pérez Ruiz - PREGUNTAS T2 - ARQUITECTURA DE COMPUTADORES

89. En un multicomputador con 4 procesadores (P0 a P3), mediante la permutación de rotación el procesador P3 envía información al procesador P0 y recibe del P1 (aparte de otras posibles comunicaciones)

RESPUESTA: FALSO



"Memorizo un
50% más en el
mismo tiempo."



MELENA DE LEÓN



BACOPA MONNIERI



CAMELLIA SINENSIS



GINGKO BILOBA



HUPERZIA SERRATA



COFFEA ARABICA

WUOLAH

EJERCICIO 1: En un multiprocesador NUMA con 16 nodos, 4GBytes por nodo, y líneas de cache de 64Bytes. ¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la coherencia de cache en un protocolo MSI sin difusión?

SOLUCIÓN: $2^2 * 2^{30} / 2^6 = 2^{26}$ entradas

EJERCICIO 2: En el multiprocesador NUMA descrito en la pregunta anterior ¿Cuántos bits tiene cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache?

SOLUCIÓN: $16+1=17$ bits

EJERCICIO 3: En un multiprocesador NUMA con 8 nodos, 16GBytes por nodo, y líneas de cache de 64Bytes. ¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la coherencia de cache en un protocolo MSI sin difusión?

SOLUCIÓN: $16 \text{ GBytes} = 2^{34} \text{ Bytes} ; 2^{34} / 2^6 = 2^{28}$ líneas

EJERCICIO 4: En el multiprocesador NUMA descrito en la pregunta anterior, ¿Cuántos bits tienen cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache en un protocolo MSI con directorio?

SOLUCIÓN: $8+1 = 9$ bits

EJERCICIO 5: En el mismo multiprocesador NUMA anterior con el mismo protocolo de coherencia de cache, se puede tener el estado 1 1 0 ... 0 V (1: hay copia del bloque en la cache del nodo correspondiente al bit; 0: no hay copia en la cache del nodo correspondiente al bit; V:bloque válido en memoria principal) en alguna de las entradas de alguno de los directorios.

RESPUESTA: VERDADERO

EJERCICIO 6: En un multiprocesador NUMA con 8 nodos, 8GBytes por nodo, y líneas de cache de 128Bytes. ¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la coherencia de cache en un protocolo MSI sin difusión?

SOLUCIÓN: $8\text{GBytes} = 2^{33} \text{ Bytes} ; 2^{33} / 2^7 = 2^{26}$ líneas

EJERCICIO 7: En el multiprocesador NUMA descrito en la pregunta anterior, ¿Cuántos bits tienen cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache en un protocolo MSI con directorio?

SOLUCIÓN: $8 + 1 = 9$ bits

EJERCICIO 8: En el mismo multiprocesador NUMA anterior con el mismo protocolo de coherencia de cache, se puede tener el estado 1 1 0 ... 0 I (1: hay copia del bloque en la cache del nodo correspondiente al bit; 0: no hay copia en la cache del nodo correspondiente al bit; I:bloque no válido en memoria principal).

RESPUESTA: FALSO

EJERCICIO 9: En un multiprocesador NUMA con 64 nodos, 8GBytes por nodo, y líneas de cache de 128Bytes. ¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la coherencia de cache en un protocolo MSI sin difusión?

SOLUCIÓN: $2^3 * 2^{30} / 2^7 = 2^{26}$ entradas

EJERCICIO 10: En el multiprocesador NUMA descrito en la pregunta anterior ¿Cuántos bits tiene cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache?

SOLUCIÓN: $64 + 1 = 65$ (un bit por nodo más un bit de validez)

EJERCICIO 11: En un multiprocesador NUMA con 32 nodos, 16GBytes por nodo, y líneas de cache de 128Bytes. ¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la coherencia de cache en un protocolo MSI sin difusión?

SOLUCIÓN: $2^4 * 2^{30} / 2^7 = 2^{27}$ entradas

EJERCICIO 12: En el multiprocesador NUMA descrito en la pregunta anterior ¿Cuántos bits tiene cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache?

SOLUCIÓN: $32 + 1 = 33$ (un bit por nodo más un bit de validez)

EJERCICIO 13: ¿Qué valores se puede observar en R si el modelo de consistencia de memoria del computador donde están los procesadores que ejecutan estos códigos no respeta el orden W->W (sí respeta los demás) e inicialmente X=Y=0?

P1: X=2 Y=1	P2: R=1; if (Y==1) R=X;
---------------------	---------------------------------

SOLUCIÓN: R=0; R=1; R=2

Si respeta el order W->W : R=1; R=2

EJERCICIO 14: ¿Qué valor deben r1,r2 y r3 para que la secuencia de instrucciones siguiente implemente un cerrojo lock(k) en el que k=1 significa que el cerrojo está cerrado y 0 que está abierto?

```
b=1;  
do  
    compare&swap(r2,b,k); // si r2==k k y b se intercambian  
    while (b==r3);
```

SOLUCIÓN: r1=1; r2=0; r3=1

EJERCICIO 15: Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden R->W (SÍ respeta todos los demás), e inicialmente X=Y=Z=r1=0 (donde X,Y,Z son variables en memoria compartida y r1 es un registro de P1), al final se podría tener Y=0.

P1: (1) while (Z==0) {}; (2) r1=X; (3) Y=r1;	P2: (a) X=1; (b) Y=2; (c) Z=1;
---	---

SOLUCIÓN: En P2 siempre se producen escrituras mientras que en P1 se produce un RAW entre (2) y (3). Como se respeta W->W, entonces las escrituras de P2 se hacen en orden, por tanto X no puede valer nunca 0, y en consecuencia, sólo podría darse que X=1.

RESPUESTA: FALSO

EJERCICIO 16: Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden R->W (SÍ respeta todos los demás), e inicialmente X=Y=Z=r1=0 (donde X,Y,Z son variables en memoria compartida y r1 es un registro de P1), al final SÓLO se podría tener Y=1.

P1:	P2:
(1) while (Z==0) {};	(a) X=1;
(2) r1=X;	(b) Y=2;
(3) Y=r1;	(c) Z=1;

SOLUCIÓN: Sigue igual que en el ejercicio anterior.

RESPUESTA: VERDADERO

EJERCICIO 17: ¿Qué valores se puede observar en R si el modelo de consistencia de memoria del computador donde están los procesadores que ejecutan estos códigos tienen un modelo de consistencia secuencial, e inicialmente X=Y=0? (R es un registro del procesador donde se ejecuta P2 y X e Y son direcciones de memoria compartida).

P1:	P2:
X=2	R=1;
Y=1	if(Y==1) R=X;

SOLUCIÓN: R=1 ó R=2.

EJERCICIO 18: ¿Qué valores se observarían en el registro R del problema anterior si no se garantiza el orden W->R?

SOLUCIÓN: Lo mismo (no cambia el orden de las escrituras en P1) R=1 ó R=2

EJERCICIO 19: ¿Qué valores se observarían en el registro R del problema anterior si no se garantiza el orden W->W?

SOLUCIÓN: R=1; R=2 ó R=0.

EJERCICIO 20: ¿Qué valor pondría en a para que la secuencia de instrucciones siguiente implemente un cerrojo lock(k) en el que k=1 significa que el cerrojo está cerrado y 0 que está abierto?

```
b=1;  
//... colocar a  
do  
    compare&swap(a,b,k); //lect-mod-escritura atómica  
    while(b==1);
```

SOLUCIÓN: a=0;

EJERCICIO 21: ¿Cómo implementaría un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y 0 que está abierto con una primitiva del tipo test_&_set?

SOLUCIÓN: while(test_&_set(k)==1){}; //k se inicializa a 0.

ENCENDER TU LLAMA CUESTA MUY POCO



Daniel Pérez Ruiz - EJERCICIOS T3 - ARQUITECTURA DE COMPUTADORES

EJERCICIO 22: ¿Qué valor deben tener r1, r2 y r3 tener para que la secuencia de instrucciones siguiente implemente un cerrojo lock(k) en el que k=1 significa que el cerrojo está cerrado y 0 que está abierto?

```
b=r1;  
do  
    compare&swap(r2,b,k); //Si r2==k -> k y b se intercambian  
    while(b==r3);
```

SOLUCIÓN: r1=1; r2=0; r3=1.

BURN.COM

#StudyOnFire

BURN
ENERGY DRINK

WUOLAH

1. Cuando se utilizan instrucciones del tipo LL/SC (lectura enlazada/escritura condicional) para implementar un cerrojo, los recursos hardware asociados a dicha técnica impiden que, entre la ejecución de la lectura (LL) y la ejecución de la escritura (SC) a la dirección de memoria del cerrojo, ningún otro procesador pueda acceder a dicha dirección de memoria del cerrojo.

EXPLICACIÓN: *Si, se permite el acceso a la memoria, pero se modifica un bit de marca para tenerlo en cuenta en la instrucción SC posterior.*

RESPUESTA: FALSO

2. Cuando se utilizan instrucciones del tipo LL/SC (lectura enlazada/escritura condicional) para implementar un cerrojo, los recursos hardware asociados a dicha técnica permiten detectar si, entre la ejecución de la lectura enlazada (LL) y la ejecución de la escritura condicional (SC) a la dirección de memoria del cerrojo, algún otro procesador ha accedido a dicha dirección.

EXPLICACIÓN: *Precisamente en eso consiste la técnica. Gracias al uso de una marca en un registro auxiliar.*

RESPUESTA: VERDADERO

3. El código siguiente permite implementar un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y k=0 que está abierto:

```
b=0; k=1;  
while (fetch_and_add(k,b)==1) {};
```

EXPLICACIÓN: *b debería ser igual a 1 y utilizar fetch_and_or(k,b)*

RESPUESTA: FALSO

4. En el protocolo MESI para mantener la coherencia de cache, una línea puede estar en el estado E (exclusivo) solo en una cache del multiprocesador.

EXPLICACIÓN: *De ahí, precisamente el nombre del estado, aunque ocurre lo mismo con una línea en el estado M.*

RESPUESTA: VERDADERO

5. En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado M (Modificado), y ese nodo detecta que el nodo N2 intenta escribir en un dato del mismo bloque B, dicho bloque pasará al estado M (Modificado) en la caché del nodo N2, y se mantendrá en el estado M en la caché del nodo N1 tras actualizarse en la memoria principal.

EXPLICACIÓN: *El bloque en N1 se invalida*

RESPUESTA: FALSO

6. En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado S (Compartido), y ese nodo detecta que el nodo N2 intenta escribir un dato en el mismo bloque B (que no está en la caché de N2), dicho bloque pasará al estado E (exclusivo) en la caché del nodo N2, y se mantendrá en el estado S en la caché del nodo N1

EXPLICACIÓN: *Pasará al estado I en el nodo N1 y al estado M en el nodo N2.*

RESPUESTA: FALSO

7. En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado S (Compartido), y ese nodo detecta que el nodo N2 intenta leer un dato del mismo bloque B, dicho bloque pasará al estado S (Compartido) en la caché del nodo N2, y se mantendrá en el estado S en la caché del N1.

EXPLICACIÓN: *Eso es precisamente lo que ocurriría según el protocolo MESI.*

RESPUESTA: VERDADERO

8. En el protocolo MSI de espionaje para la coherencia, si en la cache de un nodo N1 hay un bloque B en estado M (Modificado) y detecta que el nodo N2 intenta escribir en un dato del mismo bloque B, dicho bloque pasará al estado M (Modificado) en la caché del nodo N2, y al estado I en la caché del nodo N1 tras actualizarse en la memoria principal.

EXPLICACIÓN: *Es lo que ocurriría*

RESPUESTA: VERDADERO

9. En el protocolo MSI de espionaje para la coherencia, si en la cache de un nodo N1 hay un bloque B en estado M (Modificado) y detecta que el nodo N2 intenta escribir en un dato del mismo bloque B, dicho bloque pasará al estado S (Compartido) en las cachés de los nodos N1 y N2 tras actualizarse en la memoria principal.

EXPLICACIÓN: *Pasa a M en N2 y se invalida en N1*

RESPUESTA: FALSO

10. En el protocolo MSI de espionaje para la coherencia, si en la cache de un nodo N1 hay un bloque B en estado M (Modificado) y detecta que el nodo N2 intenta leer un dato del mismo bloque B, dicho bloque pasará al estado S (Compartido) en las cachés de los nodos N1 y N2 tras actualizarse en la memoria principal.

EXPLICACIÓN: *Efectivamente es lo que ocurre según el protocolo MSI*

RESPUESTA: VERDADERO

11. En un microprocesador SMT (multihebra simultánea) se pueden enviar a ejecutar instrucciones de hebras diferentes en cada ciclo.

EXPLICACIÓN: *Eso ocurre*

RESPUESTA: VERDADERO

12. En un microprocesador SMT (multihebra simultánea), se procesan varias hebras concurrentemente y en un instante determinado solo se pueden enviar a ejecutar instrucciones de una misma hebra.

EXPLICACIÓN: *Se pueden enviar a ejecutar instrucciones de hebras diferentes*

RESPUESTA: FALSO

13. En un multiprocesador NUMA con 16 nodos, 4 GBytes por nodo, y líneas de cache de 128 Bytes, el directorio de memoria utilizado en cada nodo para mantener la cache en un protocolo MSI sin difusión tiene 2^{25} (2 elevado a 25) entradas

EXPLICACIÓN: *$2^{32} \text{ Bytes} / 2^7 (\text{Bytes/linea}) = 2^{25} \text{ líneas}$ (el número de entradas en el directorio coincide con el número de bloques o líneas la memoria del nodo.)*

RESPUESTA: VERDADERO

14. En un multiprocesador NUMA con 16 nodos, 4 GBytes por nodo, y líneas de cache de 128 Bytes, el número de bits que tiene cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache en un protocolo MSI con directorio y codificación de bit completo es igual a 9

EXPLICACIÓN: *Son 17. Un bit por cada nodo (16) más otro de válido/no válido.*

RESPUESTA: FALSO

15. En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo NO puede haber una entrada en uno de los directorios con todos sus bits de copias en caches a cero (no hay una ninguna copia del bloque correspondiente en las caches de la máquina) y el bit de estado del bloque en memoria igual a 0 (estado No Válido en memoria)

EXPLICACIÓN: *Si el bloque no se ha llevado a ninguna caché debe estar en estado válido en la memoria principal.*

RESPUESTA: VERDADERO

16. En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo NO puede haber una entrada en uno de los directorios con un único bit a uno (hay una copia del bloque correspondiente en una cache de la máquina) y el bit de estado del bloque en memoria igual a 0 (estado No válido en memoria)

EXPLICACIÓN: *Si podría haberlo. El bloque estaría en estado modificado en la caché y no sería coherente con la memoria*

RESPUESTA: FALSO

17. En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo NO puede haber una entrada en uno de los directorios con un único bit a uno (hay una copia del bloque correspondiente en una cache de la máquina) y el bit de estado del bloque en memoria igual a 0 (estado No válido en memoria).

EXPLICACIÓN: *Si podría haberlo. El bloque podría estar en estado compartido en la correspondiente caché y sería coherente con la copia en memoria.*

RESPUESTA: FALSO

18. En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo NO puede haber una entrada en uno de los directorios con varios bits a uno (hay copias del bloque correspondiente en varias caches de la máquina) y el bit de estado del bloque en memoria igual a 0 (estado No válido en memoria)

EXPLICACIÓN: *Si hay varias copias deben estar en estado compartido y ser coherentes con la memoria principal del nodo correspondiente. Por lo tanto, el bit de estado del bloque en memoria debe estar en estado válido.*

RESPUESTA: VERDADERO

19. En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo puede haber una entrada en uno de los directorios con un único bit a uno (hay una copia del bloque correspondiente en una cache de la máquina) y el bit de estado del bloque en memoria igual a 1 (estado Válido en memoria)

EXPLICACIÓN: *El bloque estaría en estado compartido en la caché y sería coherente con la memoria.*

RESPUESTA: VERDADERO

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Daniel Pérez Ruiz - PREGUNTAS T3 - ARQUITECTURA DE COMPUTADORES

20. En un multiprocesador, el procesador P1 ejecuta las instrucciones

- (1) while ($Z==0$) { };
- (2) $r1=Y$;

en paralelo con las instrucciones que ejecuta el procesador P2:

- (a) $X=1$;
- (b) $Y=2$;
- (c) $Z=1$;

Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden $W \rightarrow R$ (SÍ respeta todos los demás), e inicialmente $X=Y=Z=r1=0$ (donde X,Y,y Z son variables en memoria compartida y r1 es un registro de P1), al final se podría tener $r1=2$.

EXPLICACIÓN: (1) estaría ejecutándose hasta que (c) se haya ejecutado y r1 se cargaría con el último valor almacenado en Y, que es 2. El orden $W \rightarrow W$ se respeta.

RESPUESTA: VERDADERO

21. En un multiprocesador, el procesador P1 ejecuta las instrucciones

- (1) while ($Z==0$) { };
- (2) $r1=Y$;

en paralelo con las instrucciones que ejecuta el procesador P2:

- (a) $X=1$;
- (b) $Y=2$;
- (c) $Z=1$;

Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden $W \rightarrow W$ (SÍ respeta todos los demás), e inicialmente $X=Y=Z=r1=0$ (donde X,Y,y Z son variables en memoria compartida y r1 es un registro de P1), al final se podría tener $r1=0$.

EXPLICACIÓN: (1) estaría ejecutándose hasta que (c) se haya ejecutado, pero (c) puede adelantar a (a) y (b) y r1 se cargaría con el último valor almacenado en Y, que podría haberse mantenido a 0 si no se ha ejecutado (b). Hay que tener en cuenta que el orden $W \rightarrow W$ no se respeta.

RESPUESTA: VERDADERO



22. En un multiprocesador, el procesador P1 ejecuta las instrucciones

- (1) while (Z==0) { };
- (2) r1=Y;

en paralelo con las instrucciones que ejecuta el procesador P2:

- (a) X=1;
- (b) Y=2;
- (c) Z=1;

Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden W→W (SÍ respeta todos los demás), e inicialmente X=Y=Z=r1=0 (donde X,Y,y Z son variables en memoria compartida y r1 es un registro de P1), al final SOLO se podría tener r1=2.

EXPLICACIÓN: (1) estaría ejecutándose hasta que (c) se haya ejecutado, pero (c) puede adelantar a (a) y (b) y r1 se cargaría con el último valor almacenado en Y, que, por ejemplo, podría haberse mantenido a 0 si no se ha ejecutado (b). Hay que tener en cuenta que el orden W→W no se respeta.

RESPUESTA: FALSO

23. Los comercialmente denominados procesadores HT de Intel (hyper-threading) son procesadores multihebra simultánea (SMT).

EXPLICACIÓN: Hyper-Threading (HT) es la denominación comercial de Intel para sus microprocesadores multihebra simultánea.

RESPUESTA: VERDADERO

24. Los microprocesadores SMT (multihebra simultánea) pueden ejecutar varias instrucciones de una misma hebra en paralelo.

EXPLICACIÓN: Precisamente, pueden enviar varias instrucciones a ser ejecutadas en un ciclo. Incluso esas instrucciones pueden pertenecer a hebras diferentes.

RESPUESTA: VERDADERO

25. Si en la secuencia de instrucciones siguiente se tiene que r1=1, r2=0, r3=0, dicha secuencia implementa un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y k=0 que está abierto.

```
b=r1;  
do  
    compare&swap(r2,b,k); // si r2==k, k y b se intercambian  
    while (b==r3);
```

EXPLICACIÓN: r3 debe ser igual a 1.

RESPUESTA: FALSO

26. Si en la secuencia de instrucciones siguiente se tiene que r1=1, r2=0, r3=1,dicha secuencia implementa un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y k=0 que está abierto.

```
b=r1;  
do  
    compare&swap(r2,b,k); // si r2==k, k y b se intercambian  
    while (b==r3);
```

EXPLICACIÓN: Es un lock() con espera activa.

RESPUESTA: VERDADERO

27. Si una línea de la cache del nodo N1 está en el estado M del protocolo MSI para mantener la coherencia de caché, el contenido de esa línea es coherente con su contenido en memoria principal.

EXPLICACIÓN: No es coherente porque ha podido ser modificada.

RESPUESTA: FALSO

28. Un procesador multinúcleo no incluye memoria cache.

RESPUESTA: FALSO

29. En un microprocesador SMT (multihebra simultánea), en un instante determinado se pueden enviar a ejecutar instrucciones de hebras diferentes.

RESPUESTA: VERDADERO

30. En un microprocesador SMT (multihebra simultánea) se pueden enviar a ejecutar varias instrucciones de una misma hebra en un instante determinado.

RESPUESTA: VERDADERO

31. En un microprocesador SMT (multihebra simultánea), se procesan varias hebras concurrentemente, aunque en un instante determinado solo se pueden enviar a ejecutar instrucciones de la misma hebra.

RESPUESTA: FALSO

32. En el protocolo MESI para mantener la coherencia de cache una línea puede estar en el estado S solo en una de las caches del multiprocesador.

RESPUESTA: FALSO

33. En el protocolo MESI para mantener la coherencia de cache, una línea puede estar en el estado E en varias caches del multiprocesador.

RESPUESTA: FALSO

34. En el protocolo MESI para mantener la coherencia de cache, una línea puede estar en el estado E sólo en una caché del multiprocesador.

RESPUESTA: VERDADERO

35. En el protocolo MESI para mantener la coherencia de cache, una línea puede estar en el estado M solo en una cache del multiprocesador.

RESPUESTA: VERDADERO

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Daniel Pérez Ruiz - PREGUNTAS T3 - ARQUITECTURA DE COMPUTADORES

36. En el protocolo MESI para mantener la coherencia de caché, una línea dada de memoria puede estar, en un momento dado, en el estado E(exclusivo) en una caché y en el estado S(compartido) en otras cachés.

RESPUESTA: FALSO

37. En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado E(Exclusivo), y ese nodo detecta que otro procesador en el nodo N2 intenta leer un dato que está en el bloque B, dicho bloque pasa al estado S(Compartido) en las caches N1 y N2.

RESPUESTA: VERDADERO

38. En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado S(Compartido), y ese nodo detecta que otro procesador en el nodo N2 intenta leer un dato que está en el bloque B, dicho bloque pasa al estado S(Compartido) en las caches N1 y N2.

RESPUESTA: VERDADERO

39. En el protocolo MSI, si en la cache de un nodo N1 hay un bloque B en estado M(Modificado), y ese nodo detecta que otro procesador en el nodo N2 intenta escribir un dato que está en el bloque B, dicho bloque pasa al estado I(no válido) en la cache del N1 y a M(modificado) en N2

RESPUESTA: VERDADERO

40. En el protocolo MSI, si en la cache de un nodo N1 hay un bloque B en estado M(Modificado), y ese nodo detecta que otro procesador en el nodo N2 intenta leer un dato que está en el bloque B, dicho bloque pasa al estado I(no válido) en la cache del N1 y a M(modificado) en N2

RESPUESTA: FALSO

41. En el protocolo MSI, si en la cache de un nodo N1 hay un bloque en estado M(Modificado), y ese nodo detecta que otro procesador intenta leer un dato que está en ese bloque, el bloque pasa al estado I(Inválido) en el nodo N1.

RESPUESTA: FALSO



TABLA 1: El procesador PROCEXA de 32 bits, puede captar TRES instrucciones por ciclo, DECODIFICAR TRES instrucciones por ciclo, y dispone de una ventana de instrucciones centralizada desde donde se produce la EMISIÓN DESORDENADA de hasta DOS instrucciones por ciclo.

También dispone de un buffer de reordenamiento (ROB) donde implementa el renombramiento y la finalización ordenada, pudiendo retirarse desde el ROB, TRES instrucciones por ciclo para escribir sus resultados en la etapa WB del cauce. El procesador incluye además DOS unidades de Suma/Resta de UN ciclo; UN multiplicador de TRES ciclos, y UNA unidad de Carga/Almacenamiento de memoria de DOS ciclos.

Por tanto, las etapas del cauce del procesador son: IF (captación de instrucciones), ID (Decodificación de instrucciones), EX (ejecución en unidad funcional de la operación codificada por la instrucción), y WB (retirada de la instrucción del ROB y escritura de resultado en los registros del procesador).

No se considera etapa ROB explícita en el cauce porque se supone que en el último ciclo de ejecución de las unidades funcionales, el resultado de la operación queda almacenado en el ROB.

Para este procesador y la secuencia de instrucciones siguiente

- (1) ld r1,0(r3) // r1=M(r3)
- (2) ld r2,8(r3) // r2 = M(r3+8)
- (3) add r2, r1, r2 // r2=r1+r2
- (4) mul r4, r1, r2 // r4= r1*r2
- (5) sub r4, r1, r4 // r4=r1-r4
- (6) sub r5, r1, r3 // r5=r1-r3

TABLA SOLUCIONADA:

		1	2	3	4	5	6	7	8	9	10	11	12
(1)	ld r1,0(r3) ; r1=M(r3)	IF	ID	EX	EX	WB							
(2)	ld r2,8(r3) ; r2=M(r3+8)	IF	ID			EX	EX	WB					
(3)	add r2,r1,r2 ; r2=r1+r2	IF	ID					EX	WB				
(4)	mul r4,r1,r2 ; r4=r1*r2	IF	ID						EX	EX	EX	WB	
(5)	sub r4,r1,r4 ; r4=r1-r4	IF	ID								EX	WB	
(6)	sub r5,r1,r3 ; r5=r1-r3	IF	ID		EX								WB

Para el procesador PROCEXA de la **TABLA 1**:

EJERCICIO 1: ¿En qué ciclo se empieza a ejecutar la instrucción (6)? (los ciclos se numeran empezando en 1).

SOLUCIÓN: Ciclo 5.

EJERCICIO 2: ¿En qué ciclo se empieza a ejecutar la instrucción (2)? (los ciclos se numeran empezando en 1).

SOLUCIÓN: Ciclo 5.

EJERCICIO 3: ¿En qué ciclo se empieza a ejecutar la instrucción (4)? (los ciclos se numeran empezando en 1).

SOLUCIÓN: Ciclo 8.

EJERCICIO 4: ¿Cuántos ciclos tarda en procesarse la secuencia de seis instrucciones indicada?

SOLUCIÓN: 12 ciclos.

TABLA 2: El cauce de un superescalar tiene las siguientes etapas:

- a) IF (1 ciclo = 1 c para cada instrucción) capaz de procesar 4 instrucciones por ciclo (i/c).
- b) ID (1 c) capaz de procesar 4 i/c.
- c) EX (de 1c a 4c) dependiendo de la unidad.
- d) WB (1c) capaz de retirar del buffer de reorden (ROB) 4 i/c.

Dispone de las siguientes unidades:

- A) 1 unidad para carga de memoria segmentada en dos etapas de 1c cada una.
- B) 1 unidad de almacenamiento en memoria (1 c).
- C) 2 unidades para *addq* y *cmpl* (1 c).
- D) 1 unidad para *addsd* (1 c).
- E) 1 unidad para *mulsd* segmentada en 4 etapas de 1c cada una.
- F) 1 unidad para saltos (1c).

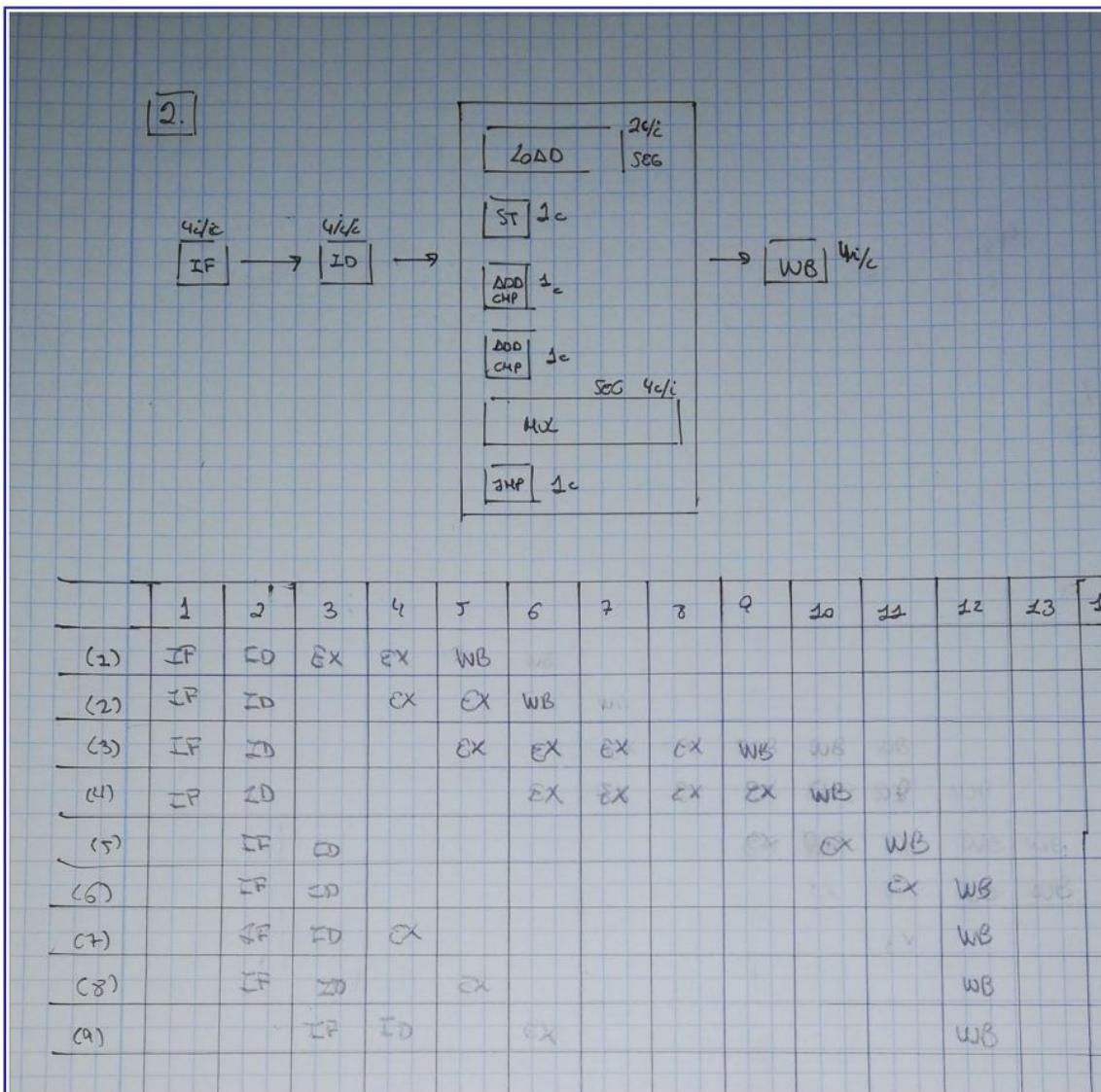
Tenga en cuenta que no hay límite en las entradas del ROB y de la ventana de instrucciones centralizada y que se pueden emitir un máximo de 4 i/c. ¿El siguiente código tarda en procesarse 14 ciclos en el cauce descrito?

.L6

(1)	movsd (%r12,%rax,8), %xmm2	;	xmm2=M[r12+rax*8]
(2)	movsd 0(%rbp,%rax,8), %xmm4	;	xmm4=M[rbp+rax*8]
(3)	mulsd %xmm1, %xmm2	;	xmm2=xmm2*xmm1
(4)	mulsd %xmm3, %xmm4	;	xmm4=xmm4*xmm3
(5)	addsd %xmm4, %xmm2	;	xmm2=xmm2+xmm4
(6)	movsd %xmm2, 0(%r13,%rax,8)	;	M[r13+rax*8]=xmm2
(7)	addq \$1, %rax	;	rax=rax+1
(8)	cmpl %eax, %ebx	;	eax-ebx
(9)	jg .L6	;	Salto si eax-ebx>0

RESPUESTA: FALSO

SOLUCIÓN TABLA:



EJERCICIO 5: Suponga que en un mismo ciclo se decodifican las cuatro instrucciones siguientes.

- (i) add r2,r2,r1 //r2=r2+r1
- (i+1) mul r3,r4,r1 //r3=r4*r1
- (i+2) add r4,r4,r1 //r4=r4+r1
- (i+3) sub r5,r2,r1 //r5=r2-r1

(entre paréntesis se indica el orden en el que están en el código), y pasan a una ventana de instrucciones única desde la que se emiten a las distintas unidades funcionales. El procesador puede emitir (con emisión DESORDENADA) CUATRO instrucciones por ciclo y tiene DOS unidades de suma/resta (con un retardo de 1 ciclo) y UN multiplicador (con un retardo de 3 ciclos).

Las instrucciones (i) e (i+3) se pueden emitir en el mismo ciclo.

RESPUESTA: FALSO. Hay dependencia RAW entre ellas.

EJERCICIO 6: En la misma situación de la pregunta anterior para la misma secuencia de instrucciones. Las cuatro instrucciones se pueden emitir en un único ciclo.

RESPUESTA: FALSO. Hay dependencia RAW entre la primera y la última.

EJERCICIO 7: Un procesador de 32 bits (4 bytes) NO podría adelantar la instrucción $i+1$ a la i (i precede a $i+1$ en el código) aunque implemente adelantamiento de LOADS a STORES ESPECULATIVO.

- (i) sw 0(r6), r2 // $M(r6) \leftarrow r2$
- ($i+1$) lw r4, 8(r6) // $r4 \leftarrow M(r6+8)$

RESPUESTA: FALSO. El dato que se escribe y el que se lee no se solapan y no hay riesgo RAW. No hace falta que el procesador tenga adelantamiento especulativo.

EJERCICIO 8: Un procesador de 32 bits (4 bytes) que implementa adelantamiento de loads a stores ESPECULATIVO podría adelantar la instrucción $i+1$ a la i (i precede a $i+1$ en el código)

- (i) sw 0(r6), r2 // $M(r6) \leftarrow r2$
- ($i+1$) lw r4, 8(r6) // $r4 \leftarrow M(r6+8)$

RESPUESTA: VERDADERO. El dato que se escribe y el que se lee no se solapan y no hay riesgo RAW. No hace falta que el procesador tenga adelantamiento especulativo.

EJERCICIO 9: Un procesador de 32 bits (4 bytes) que NO implementa adelantamiento de loads a stores ESPECULATIVO podría adelantar la instrucción $i+1$ a la i (i precede a $i+1$ en el código)

- (i) sw 0(r6), r2 // $M(r6) \leftarrow r2$
- ($i+1$) lw r4, 8(r6) // $r4 \leftarrow M(r6+8)$

RESPUESTA: VERDADERO. El dato que se escribe y el que se lee no se solapan y no hay riesgo RAW. No hace falta que el procesador tenga adelantamiento especulativo.

EJERCICIO 10: Un procesador podría adelantar la instrucción $i+1$ a la i (i precede a $i+1$ en el código) si implementa adelantamiento de LOADS a STORES ESPECULATIVO

- (i) sw 0(r5), r2 // $M(r5) \leftarrow r2$
- ($i+1$) lw r4, 0(r6) // $r4 \leftarrow M(r6)$

RESPUESTA: VERDADERO. No se puede saber si r5 y r6 apuntan a direcciones solapadas hasta que no se obtengan los correspondientes valores al ejecutar el código. Por eso para que se pueda producir el adelantamiento, el procesador debe implementar adelantamiento especulativo.

1. El aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador VLIW necesita estructuras hardware como el ROB

EXPLICACIÓN: *Precisamente se intenta reducir la complejidad hardware del procesador eliminando estructuras como el ROB y dejando que sea el compilador el principal responsable de la planificación de las instrucciones.*

RESPUESTA: FALSO

2. El aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador VLIW NO requiere el uso del ROB.

EXPLICACIÓN: *Precisamente se intenta reducir la complejidad hardware del procesador eliminando estructuras como el ROB y dejando que sea el compilador el principal responsable de la planificación de las instrucciones.*

RESPUESTA: VERDADERO

3. El buffer de reordenamiento (ROB) permite implementar la finalización ordenada en un procesador superescalar.

EXPLICACIÓN: *Efectivamente, el ROB facilita implementar de la finalización ordenada en los superescalares.*

RESPUESTA: VERDADERO

4. El principal responsable del aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador superescalar es el compilador.

EXPLICACIÓN: *Precisamente es el aumento de la complejidad del hardware del superescalar el que ha permitido aumentar la eficiencia con la que se aprovecha el paralelismo ILP en estos procesadores.*

RESPUESTA: FALSO

5. En la predicción dinámica de dos bits, el sentido de la predicción cambia (de saltar a no saltar o de no saltar a saltar) solo cuando el predictor falla dos veces seguidas.

EXPLICACIÓN: *No siempre. Depende del estado en que esté, el cambio de sentido puede necesitar dos fallos o puede producirse con un único fallo.*

RESPUESTA: FALSO

6. En la predicción dinámica de dos bits, el sentido de la predicción puede no cambiar (de saltar a no saltar o de no saltar a saltar) cuando el predictor falla.

EXPLICACIÓN: *Esto ocurriría si se encontrase en el estado 11 ó en el 00. Depende del estado en que esté, el cambio de sentido puede necesitar dos fallos o puede producirse con un único fallo.*

RESPUESTA: VERDADERO

7. En la predicción dinámica de dos bits, el sentido de la predicción siempre cambia (de saltar a no saltar o de no saltar a saltar) cuando el predictor falla.

EXPLICACIÓN: *No siempre cambia el sentido de la predicción. Depende del estado en que esté puede necesitar hasta dos fallos para cambiar el sentido de la predicción.*

RESPUESTA: FALSO

8. En la predicción dinámica de un bit el sentido de la predicción no cambia (de saltar a no saltar o de no saltar a saltar) mientras el predictor no falle

EXPLICACIÓN: *Es lo que ocurriría*

RESPUESTA: VERDADERO

9. En la predicción dinámica de un bit el sentido de la predicción cambia (de saltar a no saltar o de no saltar a saltar) si el predictor falla.

EXPLICACIÓN: *Es lo que ocurriría*

RESPUESTA: VERDADERO

10. La segmentación software es una técnica puramente software.

RESPUESTA: VERDADERO

11. La segmentación software no se puede aplicar en el caso de los procesadores superescalares.

EXPLICACIÓN: *En una técnica software que puede utilizarse para cualquier procesador, independientemente de su microarquitectura, aunque tiene más sentido utilizarla para ciertos procesadores.*

RESPUESTA: FALSO

12. El buffer de reorden se usa para eliminar dependencias WAW:

RESPUESTA: VERDADERO

13. En un VLIW, una instrucción decodificada que no disponga de unidad para su ejecución está ocupando una entrada de una estación de reserva.

RESPUESTA: FALSO

14. Hasta que una instrucción decodificada no disponga de los operandos para su ejecución permanecerá en una ventana de instrucciones de un VLIW.

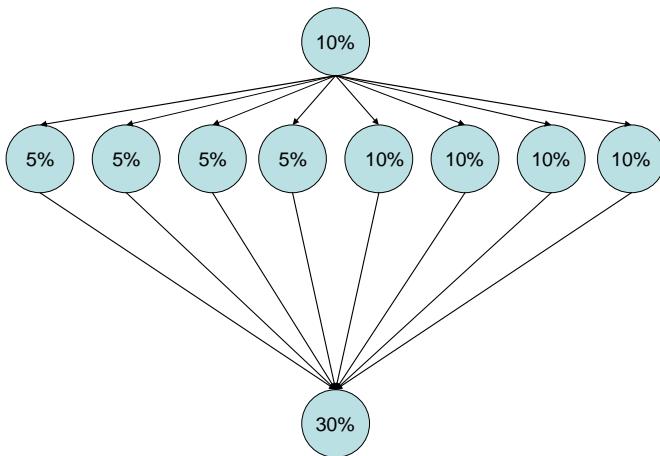
RESPUESTA: FALSO

ARQUITECTURA DE COMPUTADORES

Benchmarks 2014.

Problema. En la Figura se muestra el grafo de dependencia entre tareas para una aplicación, indicando en cada nodo la fracción del tiempo de ejecución secuencial que la aplicación tarda en ejecutar cada grupo de tareas del grafo. Suponiendo un tiempo de ejecución secuencial de 200 s, que las tareas no se pueden dividir en tareas de menor granularidad y un tiempo de comunicación despreciable:

- ¿Cuál es el valor de f en la ley de Amdahl?
- ¿Cuál es el número mínimo de procesadores para el que se obtiene la máxima ganancia de velocidad?
- ¿Cuál es esa ganancia máxima?



Solución:

$$f = (\text{parte del tiempo secuencial no paralelizable}) / (\text{tiempo secuencial}) = 200 * (0.1 + 0.3) / 200 = 0.4$$

Número mínimo para el que se consigue la máxima ganancia: **6 procesadores (cuatro procesadores ejecutarían las tareas paralelas del 10% y dos procesadores podrían ejecutar en el mismo tiempo las cuatro del 5%: si se meten más procesadores para hacer las tareas del 5%, el tiempo no se reduciría dado que tendría que esperar a que terminen los que ejecutan las tareas del 10%)**

$$S = T_1 / T_p = 200s / (200s * (0.1 + 0.1 + 0.3)) = 1 / 0.5 = 2$$

Problema. En un multiprocesador SMP con 4 procesadores o nodos (N1-N4) basado en un bus, que implementa el protocolo MESI para mantener la coherencia de cache, cada procesador dispone de una cache de datos de 512 Kbytes con marcos de bloque (también llamados líneas) de 32 bytes.

En el multiprocesador se están ejecutando en paralelo cuatro hebras, cada una en un procesador, que acceden a los elementos de dos arrays X[] e Y[] de 16 elementos de 32 bits, y se encuentran almacenados en posiciones consecutivas de la memoria principal (primero los de X[] y luego los de Y[]) a partir de un inicio de bloque.

Indique los estados de este bloque en las caches ante la siguiente secuencia de eventos:

- Lectura generada por el procesador 1 a X[0]
- Lectura generada por el procesador 2 a Y[2]

- Escritura generada por el procesador 1 a X[4]
- Escritura generada por el procesador 2 a Y[0]
- Escritura generada por el procesador 3 a X[10]
- Lectura generada por el procesador 4 a Y[1]

NOTA: Suponga que bloques distintos se almacenan en la cache de cada procesador en marcos de bloque (líneas) diferentes.

Solución:

Cada bloque de cache tiene 32 Bytes, como cada dato tiene 4 Bytes (son datos de 32 bits), en un bloque de cache hay espacio para $32/4=8$ datos. Por lo tanto los datos en memoria se ubicarían de la siguiente manera:

B(1)	X[0] X[7]
B(2)	X[0] X[15]
B(3)	Y[0] Y[7]
B(4)	Y[8] Y[15]

Los estados en la cache son los siguientes:

Acceso	Bloque	C1	C2	C3	C4
R1(X[0])	R1(B1)	(B1,E)			
R2(Y[2])	R2(B3)	(B1,E)	(B3,E)		
W1(X[4])	W1(B1)	(B1,M)	(B3,E)		
W2(Y[0])	W2(B3)	(B1,M)	(B3,M)		
W3(X[10])	W3(B2)	(B1,M)	(B3,M)	(B2,M)	
R4(Y[1])	R4(B3)	(B1,M)	(B3,S)	(B2,M)	(B3,S)

Problema. En un multiprocesador SMP con 4 procesadores basado en un bus, que implementa el protocolo MESI para mantener la coherencia, supongamos una dirección de memoria incluida en un bloque que no se encuentra en ninguna cache. Indique los estados de este bloque en las caches ante la siguiente secuencia de eventos para dicha dirección:

- Escritura generada por el procesador 1
- Lectura generada por el procesador 2
- Escritura generada por el procesador 1
- Escritura generada por el procesador 2
- Escritura generada por el procesador 3
- Lectura generada por el procesador 1

¿Por qué estados se pasaría si se utilizase el protocolo MSI?

Acceso	C1	C2	C3	C4
W1	M			
R2	S	S		
W1	M	I		
W2	I	M		
W3	I	I	M	
R1	S	I	S	

Para el MSI se pasa por los mismos estados: no se utiliza el estado E en el caso anterior para MESI

Problema. Suponga que un programa representativo del uso de un computador con un procesador no segmentado contiene un 20% de instrucciones de carga de memoria (LOAD) que consumen 5 ciclos; un 15% de escrituras en memoria (STORES) que consumen 4 ciclos; un 40% de operaciones en coma flotante que precisan 7 ciclos; un 15% de instrucciones de operaciones con enteros que necesitan 5 ciclos. El resto de instrucciones necesitan 4 ciclos. (a) ¿Qué dato o datos necesita para conocer el tiempo que tarda en ejecutarse el programa (en segundos)? (b) Indique el valor de f para la ley de Amdahl para la mejora de prestaciones en las instrucciones en coma flotante, y el valor de f para la mejora en las instrucciones de operaciones con enteros. (c) ¿Qué intentaría mejorar, las instrucciones en coma flotante o las de operaciones con enteros? ¿por qué?

Solución:

- Como $T_{cpu} = NI * CPI * T_{ciclo}$ hacen falta el número de instrucciones, NI, y el tiempo de ciclo, T_{ciclo} (o lo que es lo mismo, la frecuencia de reloj).
- Los valores de f para la ley de Amdahl se obtienen a partir de la fracción del tiempo secuencial en el que no se puede aplicar la mejora (o lo que es lo mismo, 1 menos la fracción en la que se puede aplicar la mejora)

$$f(FP) = 1 - \frac{(NI * 0.4 * 7 * T_{ciclo})}{(NI * (0.2 * 5 + 0.15 * 4 + 0.4 * 7 + 0.15 * 5 + 0.1 * 4) * T_{ciclo})} \\ = 1 - \frac{(0.4 * 7) / (0.2 * 5 + 0.15 * 4 + 0.4 * 7 + 0.15 * 5 + 0.1 * 4)}$$

$$f(INT) = 1 - \frac{(NI * 0.15 * 5 * T_{ciclo})}{(NI * (0.2 * 5 + 0.15 * 4 + 0.4 * 7 + 0.15 * 5 + 0.1 * 4) * T_{ciclo})} \\ = 1 - \frac{(0.15 * 5) / (0.2 * 5 + 0.15 * 4 + 0.4 * 7 + 0.15 * 5 + 0.1 * 4)}$$

ENCENDER TU LLAMA CUESTA MUY POCO



- c) Como $f(FP) < f(INT)$ es mejor **reducir el tiempo de las instrucciones en coma flotante** porque la parte del tiempo en el que no se podría aplicar esa mejora es menor.

BURN.COM

#StudyOnFire

BURN
ENERGY DRINK

WUOLAH

Preguntas Pruebas del Tema 4

El aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador VLIW necesita estructuras hardware como el ROB

Precisamente se intenta reducir la complejidad hardware del procesador eliminando estructuras como el ROB y dejando que sea el compilador el principal responsable de la planificación de las instrucciones

(F)

El aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador VLIW NO requiere el uso del ROB

Precisamente se intenta reducir la complejidad hardware del procesador eliminando estructuras como el ROB y dejando que sea el compilador el principal responsable de la planificación de las instrucciones

(V)

El buffer de reordenamiento (ROB) permite implementar la finalización ordenada en un procesador superescalar

Efectivamente, el ROB facilita implementar la finalización ordenada en los superescalares.

(V)

El desenrollado de bucles necesita un hardware especial de apoyo

Se consigue escribiendo adecuadamente el bucle (menos iteraciones y cuerpo del bucle con más instrucciones)

(F)

El principal responsable del aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador superescalar es el compilador.

Precisamente es el aumento de la complejidad del hardware del superescalar el que ha permitido aumentar la eficiencia con la que se aprovecha el paralelismo ILP en estos procesadores.

(F)

El procesador PROCEXA de 32 bits, puede captar TRES instrucciones por ciclo, DECODIFICAR TRES instrucciones por ciclo, y dispone de una ventana de instrucciones centralizada desde donde se produce la EMISIÓN

DESORDENADA de hasta DOS instrucciones por ciclo. También dispone de un buffer de reordenamiento (ROB) donde implementa el renombramiento y la finalización ordenada, pudiendo retirarse desde el ROB, TRES instrucciones por ciclo para escribir sus resultados en la etapa WB del cauce. El procesador incluye además DOS unidades de Suma/Resta de UN ciclo; UN multiplicador de TRES ciclos, y UNA unidad de Carga/Almacenamiento de memoria de DOS ciclos.

Por tanto, las etapas del cauce del procesador son: IF (captación de instrucciones), ID (Decodificación de instrucciones), EX (ejecución en unidad funcional de la operación codificada por la instrucción), y WB (retirada de la instrucción del ROB y escritura de resultado en los registros del procesador).

No se considera etapa ROB explícita en el cauce porque se supone que en el último ciclo de ejecución de las unidades funcionales, el resultado de la operación queda almacenado en el ROB

Para este procesador y la secuencia de instrucciones siguiente

- (1) ld r1,0(r3) // r1=M(r3)
- (2) ld r2,8(r3) // r2 = M(r3+8)
- (3) add r2, r1, r2 // r2=r1+r2
- (4) mul r4, r1, r2 // r4= r1*r2
- (5) sub r4, r1, r4 // r4=r1-r4
- (6) sub r5, r1, r3 // r5=r1-r3

¿En qué ciclo se empieza a ejecutar la instrucción (6)? (los ciclos se numeran desde 1)

Ciclo 5. La respuesta se obtiene del diagrama temporal de procesamiento de las instrucciones siguiente

		1	2	3	4	5	6	7	8	9	10	11	12
(1)	ld r1,0(r3) ; r1=M(r3)	IF	ID	EX	EX	WB							
(2)	ld r2,8(r3) ; r2=M(r3+8)	IF	ID			EX	EX	WB					
(3)	add r2,r1,r2 ; r2=r1+r2	IF	ID					EX	WB				
(4)	mul r4,r1,r2 ; r4=r1*r2		IF	ID					EX	EX	EX	WB	
(5)	sub r4,r1,r4 ; r4=r1-r4		IF	ID								EX	WB
(6)	sub r5,r1,r3 ; r5=r1-r3		IF	ID		EX							WB

Para el procesador PROCEXA y la secuencia de instrucciones que se proporciona:

¿En qué ciclo empieza a ejecutarse la instrucción (2)? (los ciclos se numeran desde 1)

Ciclo 5. La respuesta se obtiene de la tabla de procesamiento de la secuencia de instrucciones

(5)

Para el procesador PROCEXA y la secuencia de instrucciones que se proporciona:

¿En qué ciclo empieza a ejecutarse la instrucción (4)? (los ciclos se numeran desde 1)

Ciclo 8. La respuesta se obtiene a partir de la tabla de procesamiento de la secuencia de instrucciones

(8)

Para el procesador PROCEXA y la secuencia de seis instrucciones que se proporciona:

¿Cuántos ciclos tarda en procesarse la secuencia de seis instrucciones indicada?

12 ciclos. La respuesta se obtiene a partir de la tabla de procesamiento de la secuencia de instrucciones

(12)

En la predicción dinámica de dos bits, el sentido de la predicción cambia (de saltar a no saltar o de no saltar a saltar) solo cuando el predictor falla dos veces seguidas.

No siempre. Depende del estado en que esté, el cambio de sentido puede necesitar dos fallos o puede producirse con un único fallo.

(F)

En la predicción dinámica de dos bits, el sentido de la predicción puede no cambiar (de saltar a no saltar o de no saltar a saltar) cuando el predictor falla. Esto ocurriría si se encontrase en el estado 11 ó en el 00. Depende del estado en que esté, el cambio de sentido puede necesitar dos fallos o puede producirse con un único fallo.

(V)

En la predicción dinámica de dos bits, el sentido de la predicción siempre cambia (de saltar a no saltar o de no saltar a saltar) cuando el predictor falla. No siempre cambia el sentido de la predicción. Depende del estado en que esté puede necesitar hasta dos fallos para cambiar el sentido de la predicción

(F)

En la predicción dinámica de un bit el sentido de la predicción no cambia (de saltar a no saltar o de no saltar a saltar) mientras el predictor no falle

Efectivamente, se comporta de esa manera

(V)

En la predicción dinámica de un bit el sentido de la predicción cambia (de saltar a no saltar o de no saltar a saltar) si el predictor falla

Efectivamente, se comporta de esta manera

(V)

La segmentación software es una técnica puramente software

(V)

La segmentación software no se puede aplicar en el caso de los procesadores superescalares

En una técnica software que puede utilizarse para cualquier procesador, independientemente de su microarquitectura, aunque tiene más sentido utilizarla para ciertos procesadores

(F) Suponga que en un mismo ciclo se decodifican las cuatro instrucciones siguientes

```
(i) add r2, r2, r1      // r2 = r2+r1  
(i+1) mul r3, r4, r1    // r3 = r4*r1  
(i+2) add r4, r4, r1    // r4 = r4+r1  
(i+3) sub r5, r2, r1    // r5 = r2- r1
```

(entre paréntesis se indica el orden en el que están en el código), y pasan a una ventana de instrucciones única desde la que se emiten a las distintas unidades funcionales. El procesador puede emitir (con emisión DESORDENADA) CUATRO instrucciones por ciclo y tiene DOS unidades de suma/resta (con un retardo de 1 ciclo) y UN multiplicador (con un retardo de 3

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



ciclos).

Las instrucciones (i) e (i+3) se pueden emitir en el mismo ciclo

Hay dependencia RAW entre ellas
(F)

En la misma situación de la pregunta anterior para la misma secuencia de instrucciones.

Las cuatro instrucciones se pueden emitir en un único ciclo

Hay dependencia RAW entre la primera y la última.
(F)

Un procesador de 32 bits (4 bytes) NO podría adelantar la instrucción i+1 a la i (i precede a i+1 en el código) aunque implemente adelantamiento de LOADs a STORES ESPECULATIVO

(i) sw 0(r6), r2 // M(r6)<-- r2
(i+1) lw r4, 8(r6) // r4 <--M(r6+8)

El dato que se escribe y el que se lee no se solapan y no hay riesgo RAW. No hace falta que el procesador tenga adelantamiento especulativo.

(F)

Un procesador de 32 bits (4 bytes) que implementa adelantamiento de loads a stores ESPECULATIVO podría adelantar la instrucción i+1 a la i (i precede a i+1 en el código)

(i) sw 0(r5), r2 // M(r5)<-- r2
(i+1) lw r4, 8(r5) // r4 <--M(r5+8)

El dato que se escribe y el que se lee no se solapan y no hay riesgo RAW. No hace falta que el procesador sea tenga adelantamiento especulativo

(V)

Un procesador de 32 bits (4 bytes) que NO implementa adelantamiento de loads a stores ESPECULATIVO podría adelantar la instrucción i+1 a la i (i precede a i+1 en el código)

(i) sw 0(r6), r2 // M(r6)<-- r2
(i+1) lw r4, 8(r6) // r4 <--M(r6+8)

El dato que se escribe y el que se lee no se solapan y no hay riesgo RAW. No hace falta que el procesador tenga adelantamiento especulativo

(V)

Un procesador podría adelantar la instrucción i+1 a la i (i precede a i+1 en el código) si implementa adelantamiento de LOADs a STORES ESPECULATIVO

(i) sw 0(r5), r2 // M(r5)<-- r2
(i+1) lw r4, 0(r6) // r4 <--M(r6)

No se puede saber si r5 y r6 apuntan a direcciones solapadas hasta que no se obtengan los correspondientes valores al ejecutar el código. Por eso para que se pueda producir el adelantamiento, el procesador debe implementar adelantamiento especulativo.

(V)



WUOLAH

ARQUITECTURA DE COMPUTADORES. BENCHMARK del TEMA 1.

Apellidos y nombre:

- Escriba la expresión de la ley de Amdahl en términos de p (ganancia de velocidad del recurso que se ha mejorado) y de f (fracción del tiempo de procesamiento en el computador base durante el que NO se puede aprovechar la mejora):

$$S \leq p / (1 + f \times (p - 1))$$

- Según la ley de Amdahl, la máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/f$ (f definido como en la pregunta 1)

(V)

- Escriba la expresión del tiempo de CPU (T_{CPU}) en términos del número de instrucciones ejecutadas (NI), el número medio de ciclos por instrucción (CPI) y la frecuencia de reloj (F)

$$T_{CPU} = NI \times CPI / F$$

- ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta cuatro instrucciones por ciclo y funciona a una frecuencia de reloj de 3 GHz?

$$MIPS = 4 \text{ int/ciclo} \times 3 \times 10^9 \text{ ciclos/s} \times (1/10^6) = 12000$$

- La comunicación entre procesadores en un computador UMA se realiza a través de escrituras y lecturas en la memoria compartida, igual que en un computador NUMA

(V)

- Un procesador puede terminar hasta 4 operaciones en coma flotante por ciclo. ¿Cuál es su velocidad pico (en GFLOPS) si funciona a una frecuencia de reloj de 2 GHz?

$$GFLOPS = 4 \text{ op_float/ciclo} \times 2 \times 10^9 \text{ ciclos/s} \times (1/10^6) = 8$$

- El bucle $for i=1 to N do a(i)=c\times(a(i)+b(i));$ con $N=10^{14}$, se ejecuta en 10 segundos, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?

$$GFLOPS = (2 \times 10^{14} \text{ op_float}) / (10 \text{ s} \times 10^9) = 2 \times 10^4 = 20,000$$

- En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; r1 \leftarrow r2 + r3
- (b) sub r1, r2, r4 ; r1 \leftarrow r2 - r4
- (c) add r3, r2, r1 ; r3 \leftarrow r2 + r1

- El registro r1 solo genera una dependencia RAW

(F)

- No hay dependencias debido al uso del registro r2

(V)

- El registro r3 genera una dependencia WAW

(F)

ARQUITECTURA DE COMPUTADORES. BENCHMARK del TEMA 1.

Apellidos y nombre:

- Escriba la expresión de la ley de Amdahl en términos de p (ganancia de velocidad del recurso que se ha mejorado) y de f (fracción del tiempo de procesamiento en el computador base durante el que NO se puede aprovechar la mejora del recurso):

$$S \leq p / (1 + f \times (p - 1))$$

- Según la ley de Amdahl, la máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso considerado es $1/f$ (f definido como en la pregunta 1)

(V)

- Escriba la expresión del tiempo de CPU (T_{CPU}) en términos del número de instrucciones ejecutadas (NI), el número medio de instrucciones por ciclo (IPC) y el tiempo de ciclo del reloj (T_{ciclo})

$$T_{CPU} = NI \times T_{ciclo} / IPC$$

- ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta tres instrucciones por ciclo y funciona a una frecuencia de reloj de 2 GHz?

$$MIPS = 3 \text{ int/ciclo} \times 2 \times 10^9 \text{ ciclos/s} \times (1/10^6) = 6000$$

- En un computador NUMA la memoria principal está físicamente distribuida, igual que en un computador NORMA

(V)

- Un procesador puede terminar hasta 2 operaciones en coma flotante por ciclo. ¿Cuál es su velocidad pico (en GFLOPS) si funciona a una frecuencia de reloj de 2.5 GHz?

$$GFLOPS = 2 \text{ op_float/ciclo} \times 2.5 \times 10^9 \text{ ciclos/s} \times (1/10^9) = 5$$

- El bucle $for i=1 to N do a(i)=c\times(a(i)+b(i));$ con $N=10^{16}$, se ejecuta en 10 segundos, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?

$$GFLOPS = (2 \times 10^{16} \text{ op_float}) / (10 \text{ s} \times 10^9) = 2 \times 10^6 = 2,000,000$$

- En la secuencia de instrucciones:

- (a) add r1, r1, r2 ; r1 \leftarrow r1 + r2
- (b) sub r2, r2, r1 ; r2 \leftarrow r2 - r1
- (c) add r3, r3, r1 ; r3 \leftarrow r3 + r1

- El registro r1 solo genera dependencias RAW

(V)

- No hay dependencias debido al uso del registro r3

(V)

- El registro r2 genera una dependencia WAW

(F)

ARQUITECTURA DE COMPUTADORES. BENCHMARK del TEMA 1.

Apellidos y nombre:

- Escriba la expresión de la ley de Amdahl en términos de p (ganancia de velocidad del recurso que se ha mejorado) y de f (fracción del tiempo de procesamiento en el computador base durante el que NO se puede aprovechar la mejora del recurso):

$$S \leq p / (1 + f \times (p - 1))$$

- Según la ley de Amdahl, la máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso considerado es $1/f$ (f definido como en la pregunta 1)

(V)

- Escriba la expresión del tiempo de CPU (T_{CPU}) en términos del número de instrucciones ejecutadas (NI), el número medio de instrucciones por ciclo (IPC) y el tiempo de ciclo del reloj (T_{ciclo})

$$T_{CPU} = NI \times T_{ciclo} / IPC$$

- ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta tres instrucciones por ciclo y funciona a una frecuencia de reloj de 2 GHz?

$$MIPS = 3 \text{ int/ciclo} \times 2 \times 10^9 \text{ ciclos/s} \times (1/10^6) = 6000$$

- En un computador NUMA la memoria principal está físicamente distribuida, igual que en un computador NORMA

(V)

- Un procesador puede terminar hasta 2 operaciones en coma flotante por ciclo. ¿Cuál es su velocidad pico (en GFLOPS) si funciona a una frecuencia de reloj de 2.5 GHz?

$$GFLOPS = 2 \text{ op_float/ciclo} \times 2.5 \times 10^9 \text{ ciclos/s} \times (1/10^9) = 5$$

- El bucle $for i=1 to N do a(i)=cx(a(i)+b(i));$ con $N=10^{16}$, se ejecuta en 10 segundos, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?

$$GFLOPS = (2 \times 10^{16} \text{ op_float}) / (10 \text{ s} \times 10^9) = 2 \times 10^6 = 2,000,000$$

- En la secuencia de instrucciones:

- (a) add r1, r1, r2 ; r1 \leftarrow r1 + r2
- (b) sub r2, r2, r1 ; r2 \leftarrow r2 - r1
- (c) add r3, r3, r1 ; r3 \leftarrow r3 + r1

- El registro r1 solo genera dependencias RAW

(V)

- No hay dependencias debido al uso del registro r3

(V)

- El registro r2 genera una dependencia WAW

(F)

ARQUITECTURA DE COMPUTADORES
GRUPO A. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $S_p \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- p es el factor de incremento de prestaciones del recurso que se mejora (V)
- f es la fracción del tiempo antes de la mejora en el que no se utiliza el recurso mejorado (V)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/(1-f)$ (F)
- f puede ser mayor que 1 (F)

2. En un procesador superescalar a pleno rendimiento, el número de ciclos por instrucción (CPI) es menor que 1 (responda Verdadero, V, o Falso, F)

(V)

3. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo.

¿Cuál es la máxima velocidad (en GFLOPS) de un núcleo con dicha arquitectura que funciona a una frecuencia de reloj de 2.5 GHz?

$$8 \text{ FLOP/ciclo} * 2.5 \text{ (Gciclos/s)} = 20 \text{ GFLOPS}$$

4. Responda Verdadero (V) o Falso (F):

- En un computador NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida (V)
- Un multicomputador también se denomina computador UMA (F)

5. Si el bucle siguiente: for i=1 to N do a(i)=b(i)*c; se ejecuta en 5 segundos y $N=10^{12}$, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$1*10^{12} \text{ (FLOP)}/(5s*10^9)=1000/5 \text{ GFLOPS} = 200 \text{ GFLOPS}$$

6. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso comparten la memoria asignada al proceso, los registros, la pila y el contador de programa (F)
- Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse (F)
- Un multiprocesador puede funcionar como un computador MISD. (V)

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; $r1 \leftarrow r2 + r3$
- (b) sub r1, r1, r4 ; $r1 \leftarrow r1 - r4$

- Hay dependencia RAW entre las instrucciones debido al registro r1 (V)
- Hay dependencia WAR entre las instrucciones debido al registro r1 (F)

ARQUITECTURA DE COMPUTADORES
GRUPO A. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- p es el factor de incremento de prestaciones del recurso que se mejora (V)
- f es la fracción del tiempo antes de la mejora en el que no se utiliza el recurso mejorado (V)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/(1-f)$ (F)
- f puede ser mayor que 1 (F)

2. En un procesador superescalar a pleno rendimiento, el número de ciclos por instrucción (CPI) es menor que 1 (responda Verdadero, V, o Falso, F)

(V)

3. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo.

¿Cuál es la máxima velocidad (en GFLOPS) de un núcleo con dicha arquitectura que funciona a una frecuencia de reloj de 2.5 GHz?

$$8 \text{ FLOP/ciclo} * 2.5 \text{ (Gciclos/s)} = 20 \text{ GFLOPS}$$

4. Responda Verdadero (V) o Falso (F):

- En un computador NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida (V)
- Un multicomputador también se denomina computador UMA (F)

5. Si el bucle siguiente: for i=1 to N do a(i)=b(i)*c; se ejecuta en 5 segundos y $N=10^{12}$, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$1*10^{12} \text{ (FLOP)}/(5s*10^9)=1000/5 \text{ GFLOPS} = 200 \text{ GFLOPS}$$

6. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso comparten la memoria asignada al proceso, los registros, la pila y el contador de programa (F)
- Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse (F)
- Un multiprocesador puede funcionar como un computador MISD. (V)

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; $r1 \leftarrow r2 + r3$
- (b) sub r1, r1, r4 ; $r1 \leftarrow r1 - r4$

- Hay dependencia RAW entre las instrucciones debido al registro r1 (V)
- Hay dependencia WAR entre las instrucciones debido al registro r1 (F)

ARQUITECTURA DE COMPUTADORES
GRUPO B. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $S_p \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- p es el factor de incremento de prestaciones del recurso que se mejora (V)
- f es la fracción del tiempo antes de la mejora en la que se utiliza el recurso mejorado (F)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/f$ (V)
- p puede ser mayor que 1 (V)

2. En un procesador segmentado a pleno rendimiento, el número de ciclos por instrucción (CPI) es (estrictamente) menor que 1 (responda Verdadero, V, o Falso, F)

(F)

3. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo.

¿Cuál es la máxima velocidad (en GFLOPS) de un microprocesador con 4 núcleos Sunday Bridge que funciona a una frecuencia de reloj de 2 GHz?

$$8 \text{ FLOP}/(\text{núcleo*ciclo}) * 2 \text{ Ciclos/s} * 4 \text{ núcleos} = 64 \text{ GFLOPS}$$

4. Responda Verdadero (V) o Falso (F):

- Un computador UMA, es un multiprocesador donde la memoria está físicamente distribuida. (F)
- Un multicomputador también se denomina computador NORMA (V)

5. Si el bucle siguiente: for i=1 to N do $a(i)=b(i)*c$; se ejecuta en 2 segundos y $N=10^{11}$, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$1*10^{11} \text{ FLOP} / 2 \text{ s} *10^9 = 100/2 \text{ GFLOPS} = 50 \text{ GFLOPS}$$

6. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse (F)
- El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos (V)
- Un multiprocesador puede funcionar como computador MISD con la correspondiente sincronización entre sus procesadores (V)

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; $r1 \leftarrow r2 + r3$
- (b) sub r1, r1, r4 ; $r1 \leftarrow r1 - r4$

- Hay dependencia WAW entre las instrucciones debido al registro r1 (V)
- No hay dependencia WAR entre las instrucciones debido al registro r1 (V)

ARQUITECTURA DE COMPUTADORES
GRUPO B. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- p es el factor de incremento de prestaciones del recurso que se mejora (V)
- f es la fracción del tiempo antes de la mejora en la que se utiliza el recurso mejorado (F)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/f$ (V)
- p puede ser mayor que 1 (V)

2. En un procesador segmentado a pleno rendimiento, el número de ciclos por instrucción (CPI) es (estrictamente) menor que 1 (responda Verdadero, V, o Falso, F)

(F)

3. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo.

¿Cuál es la máxima velocidad (en GFLOPS) de un microprocesador con 4 núcleos Sunday Bridge que funciona a una frecuencia de reloj de 2 GHz?

$$8 \text{ FLOP}/(\text{núcleo*ciclo}) * 2 \text{ Ciclos/s} * 4 \text{ núcleos} = 64 \text{ GFLOPS}$$

4. Responda Verdadero (V) o Falso (F):

- Un computador UMA, es un multiprocesador donde la memoria está físicamente distribuida. (F)
- Un multicomputador también se denomina computador NORMA (V)

5. Si el bucle siguiente: for i=1 to N do $a(i)=b(i)*c$; se ejecuta en 2 segundos y $N=10^{11}$, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$1*10^{11} \text{ FLOP} / 2 \text{ s} *10^9 = 100/2 \text{ GFLOPS} = 50 \text{ GFLOPS}$$

6. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse (F)
- El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos (V)
- Un multiprocesador puede funcionar como computador MISD con la correspondiente sincronización entre sus procesadores (V)

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; $r1 \leftarrow r2 + r3$
- (b) sub r1, r1, r4 ; $r1 \leftarrow r1 - r4$

- Hay dependencia WAW entre las instrucciones debido al registro r1 (V)
- No hay dependencia WAR entre las instrucciones debido al registro r1 (V)

ARQUITECTURA DE COMPUTADORES

GRUPO IM. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- f es la fracción del tiempo antes de aplicar la mejora en la que se utiliza el recurso mejorado (F)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/(1-f)$ (F)
- p no puede ser mayor que 1 (F)

2. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo. ¿Cuál es la máxima velocidad (en GFLOPS) de un procesador de 6 núcleos con dicha arquitectura que funciona a una frecuencia de reloj de 2.5 GHz?

$$8 \text{ FLOP}/(\text{ciclo*núcleo}) * 2.5 \text{ (Gciclos/s)} * 6 \text{ núcleos} = 120 \text{ GFLOPS}$$

3. Responda Verdadero (V) o Falso (F):

- En un computador NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida (V)
- Un computador UMA es un tipo de multiprocesador (V)
- En un computador de tipo NORMA los accesos a memoria local y remota se realizan a través de instrucciones de acceso a memoria (carga y almacenamiento de datos en memoria) (F)

4. Si el bucle siguiente: for i=1 to N do a(i)=b(i)+c; y $N=10^{15}$, se ejecuta en 2 segundos en un computador, siendo c, a(), y b() datos en coma flotante. ¿Cuántos TFLOPS alcanza la máquina al ejecutar el código?.

$$1 * 10^{15} \text{ FLOP} / 2 \text{ s} * 10^{12} = 1000/2 \text{ TFLOPS} = 500 \text{ TFLOPS}$$

5. Escriba la expresión de los MIPS en términos del número de ciclos por instrucción (CPI) del procesador, y de su frecuencia de reloj (F). $\text{MIPS} = F/\text{CPI} * 10^6$

8. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso no necesitan recurrir a llamadas al sistema operativo para comunicarse (V)
- El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos (V)
- Los multicomputadores son máquinas MIMD y los multiprocesadores SIMD. (F)

9. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; r1 \leftarrow r2 + r3
- (b) sub r2, r1, r4 ; r2 \leftarrow r1 - r4

- Hay dependencia RAW entre las instrucciones debido al registro r1 (V)
- No hay ninguna dependencia WAR entre las instrucciones (F)
- No hay ninguna dependencia WAW entre las instrucciones (V)

ARQUITECTURA DE COMPUTADORES

GRUPO IM. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- f es la fracción del tiempo antes de aplicar la mejora en la que se utiliza el recurso mejorado (F)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/(1-f)$ (F)
- p no puede ser mayor que 1 (F)

2. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo. ¿Cuál es la máxima velocidad (en GFLOPS) de un procesador de 6 núcleos con dicha arquitectura que funciona a una frecuencia de reloj de 2.5 GHz?

$$8 \text{ FLOP}/(\text{ciclo*núcleo}) * 2.5 \text{ (Gciclos/s)} * 6 \text{ núcleos} = 120 \text{ GFLOPS}$$

3. Responda Verdadero (V) o Falso (F):

- En un computador NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida (V)
- Un computador UMA es un tipo de multiprocesador (V)
- En un computador de tipo NORMA los accesos a memoria local y remota se realizan a través de instrucciones de acceso a memoria (carga y almacenamiento de datos en memoria) (F)

4. Si el bucle siguiente: for i=1 to N do a(i)=b(i)+c; y $N=10^{15}$, se ejecuta en 2 segundos en un computador, siendo c, a(), y b() datos en coma flotante. ¿Cuántos TFLOPS alcanza la máquina al ejecutar el código?.

$$1 * 10^{15} \text{ FLOP} / 2 \text{ s} * 10^{12} = 1000/2 \text{ TFLOPS} = 500 \text{ TFLOPS}$$

5. Escriba la expresión de los MIPS en términos del número de ciclos por instrucción (CPI) del procesador, y de su frecuencia de reloj (F). $\text{MIPS} = F/\text{CPI} * 10^6$

8. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso no necesitan recurrir a llamadas al sistema operativo para comunicarse (V)
- El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos (V)
- Los multicomputadores son máquinas MIMD y los multiprocesadores SIMD. (F)

9. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; r1 \leftarrow r2 + r3
- (b) sub r2, r1, r4 ; r2 \leftarrow r1 - r4

- Hay dependencia RAW entre las instrucciones debido al registro r1 (V)
- No hay ninguna dependencia WAR entre las instrucciones (F)
- No hay ninguna dependencia WAW entre las instrucciones (V)

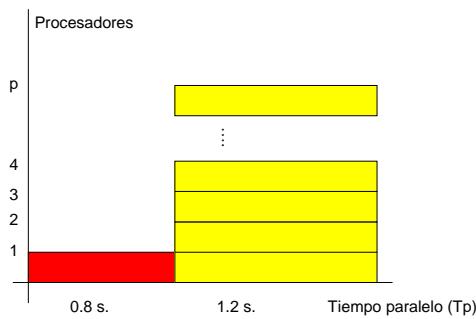
ARQUITECTURA DE COMPUTADORES

GRUPO A. BENCHMARK del TEMA 2

Estudiante:

- Escriba la expresión de la ley de Gustafson en términos de los parámetros f y p:

$$S_p = f + (1-f)p$$



- Teniendo en cuenta la figura anterior

- ¿Qué valor tiene el parámetro f en la ley de Gustafson:

$$f_g = 0.8/2.0$$

- Escriba el valor del parámetro f en la ley de Amdahl (en función del número de procesadores p)
 $f_a = 0.8/(0.8+1.2p)$

- Complete la siguiente Tabla de Ganancias de Velocidad:

Fracción no paralela en T_s	Grado de Parallelismo	Overhead	Ganancia
0	ilimitado	0	p
f	ilimitado	0	$p/(1+f(p-1))$
f	n	0	$p/(1+f(p-1)) \text{ (} p \leq n \text{) y } n/(1+f(n-1)) \text{ (} p > n \text{)}$
f	ilimitado	$T_o(p)=p$	$1/(f+(1-f)/p+(p/T_s))$

- Responda Verdadero (V) o Falso (F):

- La reducción implica comunicación colectiva todos-a-uno (V)
- La acumulación (gather) implica comunicación colectiva todos-con-todos (F)
- MPI es una biblioteca de paso de mensajes (V)
- En la asignación de carga estática se asigna el trabajo que va a realizar cada procesador, antes de la ejecución (V)
- El tiempo de sincronización entre procesos forma parte del overhead de un programa paralelo (V)

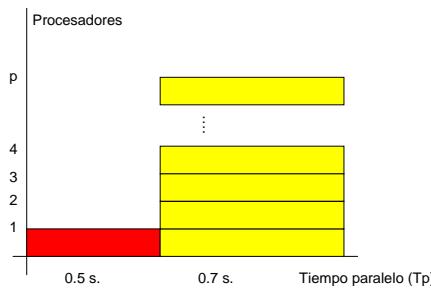
ARQUITECTURA DE COMPUTADORES

GRUPO B. BENCHMARK del TEMA 2

Estudiante:

- Escriba la expresión de la ley de Gustafson en términos de los parámetros f y p:

$$S_p = f + (1-f)p$$



- Teniendo en cuenta la figura anterior

- ¿Qué valor tiene el parámetro f en la ley de Gustafson:

$$f_g = 0.5 / 1.2$$

- Escriba el valor del parámetro f en la ley de Amdahl (en función del número de procesadores p)

$$f_a = 0.5 / (0.5 + 0.7p)$$

- Complete la siguiente Tabla de Ganancias de Velocidad:

Fracción no paralela en T_s	Grado de Paralelismo	Overhead	Ganancia
0	ilimitado	0	p
f	ilimitado	0	$p / (1 + f(p-1))$
f	n	0	$p / (1 + f(p-1)) \text{ (} p \leq n \text{)} \text{ y } n / (1 + f(n-1)) \text{ (} p > n \text{)}$
f	ilimitado	$T_o(p) = p$	$1 / (f + (1-f)/p + (p/T_s))$

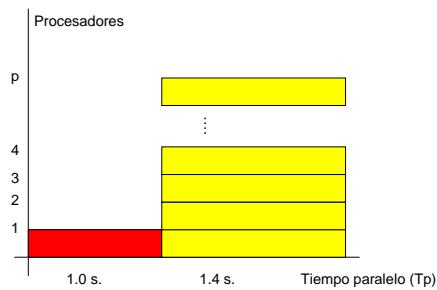
- Responda Verdadero (V) o Falso (F):

- La difusión (broadcast) implica comunicación colectiva de todos-con-todos (F)
- La dispersión (scatter) implica comunicación colectiva todos-con-todos (F)
- OpenMP es una biblioteca que permite hacer programas paralelos con paso de mensajes (F)
- El tiempo de comunicación entre procesos forma parte del overhead de un programa paralelo (V)
- La asignación de carga dinámica se realiza antes de la ejecución del programa paralelo (F)

ARQUITECTURA DE COMPUTADORES
GRUPO IM. BENCHMARK del TEMA 2
Estudiante:

1. Escriba la expresión de la ley de Gustafson en términos de los parámetros f y p:

$$S_p = f + (1-f)p$$



2. Teniendo en cuenta la figura anterior

- ¿Qué valor tiene el parámetro f en la ley de Gustafson:

$$f_g = 1.0 / 2.4$$

- Escriba el valor del parámetro f en la ley de Amdahl (en función del número de procesadores p)

$$f_a = 1.0 / (1.0 + 1.4p)$$

3. Complete la siguiente Tabla de Ganancias de Velocidad:

Fracción no paralela en T_s	Grado de Paralelismo	Overhead	Ganancia
0	ilimitado	0	p
f	ilimitado	0	$p / (1 + f(p-1))$
f	n	0	$p / (1 + f(p-1)) \text{ (} p \leq n \text{)} \text{ y } n / (1 + f(n-1)) \text{ (} p > n \text{)}$
f	ilimitado	$T_o(p) = p$	$1 / (f + (1-f)/p + (p/T_s))$

4. Responda Verdadero (V) o Falso (F):

- La difusión (broadcast) implica comunicación colectiva de todos-a-todos (F)
- La dispersión (scatter) implica comunicación colectiva todos-a-uno (F)
- La asignación de carga dinámica no tiene nunca ningún coste en el momento de la ejecución (F)
- Para equilibrar la carga asignada a los procesadores interesa asignar más carga a los procesadores más rápidos (V)
- El tiempo de comunicación entre procesos forma parte del overhead de un programa paralelo (V)

1. OpenMP es una biblioteca que permite hacer programas paralelos con paso de mensajes
= (F)
2. MPI es una biblioteca de paso de mensajes
= (V)
3. El tiempo de comunicación entre procesos forma parte del overhead de un programa paralelo = (V)
4. La asignación de carga dinámica afecta al tiempo de overhead del programa paralelo
= (V)
5. La asignación de carga dinámica se realiza antes de la ejecución del programa paralelo
= (F)
6. La asignación de carga dinámica no tiene ningún coste en el momento de la ejecución
= (F)
7. En la asignación de carga estática se asigna el trabajo que ca a realizar cada procesador, antes de la ejecución
= (V)
8. Para equilibrar la carga asignada a los procesadores interesa asignar más carga a los procesadores más rápidos
= (V)
9. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, cosa que no ocurre en la comunicación gossiping
= (F)
10. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, cosa que también ocurre en la comunicación gossiping
= (V)
11. En la comunicación colectiva de tipo gossiping todos los procesadores envían información, pero no todos los procesadores reciben
= (F)
12. La acumulación (gather) implica comunicación colectiva de todos-con-todos
= (F)
13. La difusión (broadcast) implica comunicación colectiva de todos-con-todos
= (F)
14. La dispersión (scatter) implica comunicación colectiva todos-con-todos
= (F)
15. La dispersión (scatter) implica comunicación colectiva todos-a-uno
= (F)
16. La reducción implica comunicación colectiva todos-a-uno
= (V)

1. OpenMP es una biblioteca que permite hacer programas paralelos con paso de mensajes
= (F)
2. MPI es una biblioteca de paso de mensajes
= (V)
3. El tiempo de comunicación entre procesos forma parte del overhead de un programa paralelo = (V)
4. El tiempo de sobrecarga u overhead es un componente del tiempo de procesamiento paralelo junto con el tiempo de comunicación
=(F)
5. La asignación de carga dinámica afecta al tiempo de overhead del programa paralelo
= (V)
6. La asignación de carga dinámica se realiza antes de la ejecución del programa paralelo
= (F)
7. La asignación de carga dinámica no tiene ningún coste en el momento de la ejecución
= (F)
8. En la asignación de carga estática se asigna el trabajo que ca a realizar cada procesador, antes de la ejecución
= (V)
9. Para equilibrar la carga asignada a los procesadores interesa asignar más carga a los procesadores más rápidos
= (V)
10. La falta de equilibrado de la carga es una de las causas de que haya tiempo de sobrecarga u overhead en los programas paralelos
=(V)
11. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, cosa que no ocurre en la comunicación gossiping
= (F)
12. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, cosa que también ocurre en la comunicación gossiping
= (V)
13. En la comunicación colectiva de tipo gossiping todos los procesadores envían información, pero no todos los procesadores reciben
= (F)
14. La acumulación (gather) implica comunicación colectiva de todos-con-todos
= (F)
15. La acumulación (gather) es un modo de comunicación colectiva en el que todos los procesadores envían información a uno de ellos
=(V)
16. La difusión (broadcast) implica comunicación colectiva de todos-con-todos
= (F)
17. La dispersión (scatter) implica comunicación colectiva todos-con-todos
= (F)
18. La dispersión (scatter) implica comunicación colectiva todos-a-uno
= (F)
19. Tanto la difusión (broadcast) como la dispersión (scatter) implican comunicación de un procesador a todos los demás
=(V)
20. La reducción implica comunicación colectiva todos-a-uno
= (V)

21. En un multicomputador con 4 procesadores (P0 a P3), mediante la permutación de rotación, el procesador P0 envía información al procesador P1 y recibe del P2 (aparte de otras posibles comunicaciones)
=(F)
22. En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) prefijo paralelo, el procesador P2 recibe información de los procesadores P0, y del propio P2 (aparte de otras posibles comunicaciones)
=(V)
23. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a 0 es igual a p para $p < n$
=(V)
24. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a f, un grado de paralelismo ilimitado y un tiempo de overhead igual a 0 es $p/(1+f(p-1))$
=(V)
25. Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de cargados). El valor de la f de la ley de Gustafson es 0.5
=(V)
26. La expresión para la ley de Gustafson es $S=f+p*(1-f)$, donde f es la fracción no paralelizable del tiempo de ejecución paralelo y p es el número de procesadores que intervienen.
=(V)
27. La expresión para la ley de Gustafson es $S=(1-f)+p+f$, donde f es la fracción no paralelizable del tiempo de ejecución paralelo y p es el número de procesadores que intervienen.
=(F)
28. Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es 3
=(V)
29. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo ilimitado y un tiempo de overhead igual a p^2 es $T_s/((Ts/p)+p^2)$
=(V)
30. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a p es $T_s/((Ts/n)+p)$, para $p=n$
=(V)
31. Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es 4
=(F)
32. Un programa secuencial tarda 40 ns en ejecutarse en un procesador y durante 10 ns de esos 40 ns el programa no es paralelizable. El valor de la f de la ley de Amdahl para ese programa es igual a 0.75
=(F)
33. En la expresión de la ganancia de velocidad, $S=T_s/T_P$, el tiempo de computación paralelo, T_P , se obtiene sumando el tiempo de cálculo paralelo más el tiempo de sobrecarga u overhead, más el tiempo de comunicación

=**(F)**

34. Un programa paralelo tarda 50 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 40 ns intervienen 6 procesadores (todos ellos igual de cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es 5
=**(V)**
35. En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) sufijo paralelo, el procesador P2 envía información los procesadores P0, P1, y al propio P2 (aparte de otras posibles comunicaciones)
=**(V)**
36. La expresión para la ley de Gustafson es $S=f+p*(1-f)$, donde f es la fracción no parallelizable del tiempo de ejecución secuencial y p es el número de procesadores que intervienen.
=**(F)**
37. Un programa paralelo tarda 40 ns en ejecutarse en un procesador y durante 10 ns de esos 40 ns el programa no es parallelizable, mientras que en el resto del tiempo paralelo intervienen cinco procesadores cargados por igual. El valor de la f de la ley de Gustafson para ese programa es igual a 0.25
=**(V)**
38. En un multicomputador con 4 procesadores (P0 a P3), mediante la permutación de rotación el procesador P3 envía información al procesador P0 y recibe del P1 (aparte de otras posibles comunicaciones)
=**(F)**

1.- En un microprocesador SMT (multihebra simultánea) se pueden enviar a ejecutar varias instrucciones de una misma hebra en un instante determinado. V

2.- En el protocolo MESI para mantener la coherencia de cache, una línea dada de memoria puede estar, en un momento dado, en el estado E(exclusivo) en una cache y en el estado S(compartido) en otras caches. F

3.- En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado E(Exclusivo), y ese nodo detecta que otro procesador en el nodo N2 intenta leer un dato que está en el bloque B, dicho bloque pasa al estado S(Compartido) en las caches del N1 y N2. V

4.- En el protocolo MSI, si en la cache de un nodo N1 hay un bloque B en estado M(Modificado), y ese nodo detecta que otro procesador en el nodo N2 intenta escribir un dato que está en el bloque B, dicho bloque pasa al estado I(no válido) en la cache del N1 y a M(Modificado) en N2. V

5.- En un multiprocesador NUMA con 8 nodos, 16 GBytes por nodo, y líneas de caché de 64 Bytes. ¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la coherencia de cache en un protocolo MSI sin difusión? $16Gb = 2^{34}$ Bytes. $2^{34}/2^6 = 2^{28}$ líneas.

6.- En un multiprocesador NUMA descrito en la pregunta anterior ¿Cuántos bits tienen cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache en un protocolo MSI con directorio? $8+1 = 9$ bits.

7.- En el mismo multiprocesador NUMA anterior con el mismo protocolo de coherencia de cache, se puede tener el estado 1 1 0 ... 0 V (1:hay copia del bloque en la cache del nodo correspondiente al bit; 0: no hay copia en la cache del nodo correspondiente al bit; V: bloque válido en memoria principal) en alguna de las entradas de alguno de los directorios. V

8.- Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden R->W (Sí respeta todos los demás), e inicialmente $X=Y=Z=r1=0$ (donde X,Y y Z son variables en memoria compartida y r1 es un registro de P1), al final se podría tener $Y = 0$. F

P1:

```
while(Z==0){};  
r1=X;  
Y=r1;
```

P2:

```
X=1;  
Y=2;  
Z=1;
```

9.- Cuando se utilizan instrucciones del tipo LL/SC (lectura enlazada/escritura condicional) para implementar un cerrojo, los recursos hardware asociados a dicha técnica permiten detectar si, entre la ejecución de la lectura enlazada (LL) y la ejecución de la escritura condicional (SC) a la dirección de memoria del cerrojo, algún otro procesador ha accedido a dicha dirección. V

10.- Si en la secuencia de instrucciones siguiente se tiene que $r1=1$, $r2=0$, $r3=0$, dicha secuencia implementa un cerrojo (lock(k)) en el que $k=1$ significa que el cerrojo está cerrado y $k=0$ que está abierto. F

```
b=r1;  
do  
    compare&swap(r2,b,k); //Si r2==k, k y b se intercambian  
    while(b==r3);
```

11.- En un microprocesador SMT(multihebra simultánea), se procesan varias hebras concurrentemente y en un instante determinado solo se pueden enviar a ejecutar instrucciones de una misma hebra. F

12.- En el protocolo MESI para mantener la coherencia de cache, una línea puede estar en el estado E(Exclusivo) solo en la cache del multiprocesador. V

13.- En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado S(Compartido), y ese nodo detecta que otro procesador en el nodo N2 intenta leer un dato que está en el bloque B, dicho bloque pasa al estado S(Compartido) en las caches del N1 y N2. V

14.- En el protocolo MSI, si en la cache de un nodo N1 hay un bloque B en estado M(Modificado), y ese nodo detecta que el procesador del nodo N2 intenta leer un dato que está en el bloque B, dicho bloque pasa al estado I(no válido) en la cache del N1y a M(Modificado) en N2.

15.- En un multiprocesador NUMA con 8 nodos, 8GBytes por nodo, y líneas de caché de 128 Bytes. ¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la cache en un protocolo MSI sin difusión? $8\text{Gbytes} = 2^{33}$ Bytes. $2^{33}/2^7 = 2^{26}$ entradas

16.- En el multiprocesador NUMA descrito en la pregunta anterior ¿Cuántos bits tiene cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache en un protocolo MSI con directorio? $8+1=9$

17.- En el mismo multiprocesador NUMA anterior con el protocolo MSI de coherencia de cache, es posible que alguna de las entradas de alguno de los directorios esté en el estado 1 1 0 ... 0 I (1:hay copia del bloque en la cache del nodo correspondiente al bit; 0: no hay copia en la cache del nodo correspondiente al bit; I: bloque no válido en memoria principal). F

18.- Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden R->W (Sí respeta todos los demás), e inicialmente X=Y=Z=r1=0 (donde X,Y y Z son variables en memoria compartida y r1 es un registro de P1), al final SOLO podría tener Y = 1. V

P1:	P2:
while(Z==0){};	X=1;
r1=X;	Y=2;
Y=r1;	Z=1;

19.- Cuando se utilizan instrucciones del tipo LL/SC (lectura enlazada/escritura condicional) para implementar un cerrojo, los recursos hardware asociados a dicha técnica impiden que, entre la ejecución de la lectura (LL) y la ejecución de la escritura (SC) a la dirección de memoria del cerrojo, ningún otro procesador pueda acceder a dicha dirección de memoria del cerrojo. F

20.- Si en la secuencia de instrucciones siguiente se tiene que r1=1, r2=0, r3=1, dicha secuencia implementa un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y k=0 que está abierto. V

```
b=r1;
do
    compare&swap(r2,b,j);
while(b==r3);
```

21.- Un microprocesador multinúcleo no incluye memoria caché. F

22.- En un microprocesador SMT (Multihebra simultánea), en un instante determinado se pueden enviar a ejecutar instrucciones de hebras diferentes. V

23.- En un microprocesador SMT (Multihebra simultánea), se procesan varias hebras concurrentemente aunque en un instante determinado solo se pueden enviar a ejecutar instrucciones de la misma hebra. F

24.- En el protocolo MESI para mantener coherencia de cache, una línea puede estar en el estado E en varias caches del multiprocesador. F

25.- En el protocolo MSI, si en la cache de un nodo N1 hay un bloque en estado M(Modificado), y ese nodo detecta que otro procesador intenta leer un dato que está en ese bloque, el bloque pasa al estado I (no válido) en el nodo N1. F

26.- En un multiprocesador NUMA con 16 nodos, 4GBytes por nodo, y líneas de cache de 64 Bytes. ¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la coherencia de cache en un protocolo MSI sin difusión? $2^2 * 2^{30} / 2^6 = 2^{26}$ entradas

27.- En el multiprocesador NUMA descrito en la pregunta anterior ¿Cuántos bits tiene cada una de las entradas del directorio que se utiliza para mantener

coherencia de cache? $16 + 1 = 17$

28.- ¿Qué valores se puede observar en R si el modelo de consistencia de memoria del computador donde están los procesador que ejecutan estos códigos no respeta el orden W->W(sí respeta los demás) e inicialmente X=Y=0?

P1:

```
X = 2;  
Y = 1;
```

P2:

```
R=1;  
if(Y==1)R=X;
```

R = 0; R = 1; R = 2;
Sí, respeta el orden W->W: R=1; R=2

29.- ¿Qué valor deben r1, r2, y r3 tener para que la secuencia de instrucciones siguiente implemente un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y 0 que está abierto?

```
b=r1;  
do  
    compare&swap(r2, b, k); //si r2==k k y b se intercambian  
    while(b==r3)  
  
r1=1, r2=0, r3=1
```

1.- Suponga que en un mismo ciclo se decodifican las cuatro instrucciones siguientes

```
(i) add r2, r2, r1 // r2 = r2 + r1  
(i + 1) mul r3, r4, r1 // r3 = r4 * r1  
(i + 2) add r4, r4, r1 // r4 = r4 + r1  
(i + 3) sub r5, r2, r1 // r5 = r2 - r1
```

(entre paréntesis se indica el orden en el que están en el código), y pasan a una ventana de instrucciones única desde la que se emiten distintas unidades funcionales. El procesador puede emitir (con emisión DESORDENADA) TRES instrucciones por ciclo y tiene DOS unidades de suma/resta (con un retardo de 1 ciclo) y UN multiplicador (con un retardo de 3 ciclos).

Las instrucciones (i) e (i+3) se pueden emitir en el mismo ciclo, F

2.- En la predicción dinámica de un bit el sentido de la predicción no cambia (de saltar a no saltar o de no saltar a saltar) mientras el predictor no falle. V

3.- Un procesador de 32 bits (4 bytes) que implementa adelantamiento de loads a stores ESPECULATIVO podría adelantar la instrucción i+1 a la i (i precede a i+1 en el código)

```
(i) sw 0(r5), r2 // M(r5)<-- r2  
(i + 1) lw r4, 8(r5) //r4<--M(r5+8)
```

V

4.- Para el procesador PROCEXA y la secuencia de seis instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿Cuántos ciclos tarde en procesarse la secuencia de seis instrucciones indicada? 8

5.- El aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador VLIW necesita el uso del ROB. F

6.- El desenrollado de bucles necesita un hardware especial de apoyo. F

7.- Para el procesador PROCEXA y la secuencia de instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿En qué ciclo empieza a ejecutarse la instrucción (2)? (los ciclos se numeran desde 1). 5

8- Para el procesador PROCEXA y la secuencia de instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿En qué ciclo empieza a ejecutarse la instrucción (4)? (los ciclos se numeran desde 1). 8

9.- En la predicción dinámica de dos bits, el sentido de la predicción siempre cambia (de saltar a no saltar o de no saltar a saltar) cuando el predictor falla. F

10.- El procesador PROCEXA de 32 bits, puede captar TRES instrucciones por ciclo, DECODIFICAR TRES instrucciones por ciclo, y dispone de una ventana de instrucciones centralizada desde donde se produce la EMISIÓN DESORDENADA de hasta DOS instrucciones por ciclo. También dispone de un buffer de reordenamiento (ROB) donde implementa el renombramiento y la finalización ordenada, pudiendo retirarse desde el ROB, TRES instrucciones por ciclo para escribir sus resultados en la etapa WB del cauce. El procesador incluye además DOS unidades de Suma/Resta de UN ciclo; UN multiplicador de TRES ciclos, y UNA unidad de Carga/Almacenamiento de memoria de DOS ciclos.

Por tanto, las etapas del cauce del procesador son IF(capatación de instrucciones), ID(Decodificación de instrucciones), EX(ejecución en unidad funcional de la operación codificada por la instrucción), y WB(retirada de la instrucción del ROB y escritura de resultado en los registros del procesador). No se considera etapa ROB explícita en el cauce porque se supone que en el último ciclo de ejecución de las unidades funcionales, el resultado de la operación queda almacenado en el ROB

Para este procesador y la secuencia de instrucciones siguiente

```
(1) ld r1,0(r3) //r1=M(r3)
```

```
(2) ld r2,8(r3) //r2=M(r3+8)
(3) add r2, r1, r2 //r2=r1+r2
(4) mul r4, r1, r2 //r4=r1*r2
(5) sub r4, r1, r4 //r4=r1-r4
(6) sub r5, r1, r3 //r5=r1-r3
```

¿En qué ciclo se empieza a ejecutar la instrucción (6)? (los ciclos se numeran desde 1). 5

11.- La segmentación software es una técnica puramente software. V

12.- El aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador VLIW NO requiere el uso del ROB. V

13.- Un procesador de 32 bits (4 bytes) que implementa adelantamiento de loads a stores, aunque no implementa el adelantamiento ESPECULATIVO, podría adelantar la instrucción $i+1$ a la i (i precede a $i+1$ en el código)

```
(i) sw 0(R6), r2 //r2=r2+r1
(i+1) lw r4, 8(r6) //r4<-M(r6+8)
```

V

15.- Suponga que en un mismo ciclo se decodifican las cuatro instrucciones siguientes

```
(i) add r2, r2, r1 // r2 = r2 + r1
(i + 1) mul r3, r4, r1 // r3 = r4 * r1
(i + 2) add r4, r4, r1 // r4 = r4 + r1
(i + 3) sub r5, r2, r1 // r5 = r2 - r1
```

(entre paréntesis se indica el orden en el que están en el código), y pasan a una ventana de instrucciones única desde la que se emiten distintas unidades funcionales. El procesador puede emitir (con emisión DESORDENADA) CUATRO instrucciones por ciclo y tiene TRES unidades de suma/resta (con un retardo de 1 ciclo) y UN multiplicador (con un retardo de 3 ciclos).

La instrucción (i) NO se puede emitir en el mismo ciclo que la (i+3), V

16.- Para el procesador PROCEXB y la secuencia de seis instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿En qué ciclo empieza a ejecutarse la instrucción (4)? (los ciclos se numeran desde 1). 8

17.- En la predicción dinámica de dos bits, el sentido de la predicción siempre cambia (de saltar a no saltar o de no saltar a saltar) cuando el predictor falla. F

18.- El procesador PROCEXB de 32 bits, puede captar TRES instrucciones por ciclo, DECODIFICAR TRES instrucciones por ciclo, y dispone de una ventana de instrucciones centralizada desde donde se produce la EMISIÓN DESORDENADA de hasta DOS instrucciones por ciclo. También dispone de un buffer de reordenamiento (ROB) donde implementa el renombramiento y la finalización ordenada, pudiendo retirarse desde el ROB, TRES instrucciones por ciclo para escribir sus resultados en la etapa WB del cauce. El procesador incluye además DOS unidades de Suma/Resta de UN ciclo; UN multiplicador de TRES ciclos, y UNA unidad de Carga/Almacenamiento de memoria de DOS ciclos.

Por tanto, las etapas del cauce del procesador son IF(capatación de instrucciones), ID(Decodificación de instrucciones), EX(ejecución en unidad funcional de la operación codificada por la instrucción), y WB(retirada de la instrucción del ROB y escritura de resultado en los registros del procesador). No se considera etapa ROB explícita en el cauce porque se supone que en el último ciclo de ejecución de las unidades funcionales, el resultado de la operación queda almacenado en el ROB

Para este procesador y la secuencia de instrucciones siguiente

```
(1) ld r1,0(r3) //r1=M(r3)
(2) ld r2,8(r3) //r2=M(r3+8)
(3) add r2, r1, r2 //r2=r1+r2
(4) mul r4, r1, r2 //r4=r1*r2
```

ENCENDER TU LLAMA CUESTA MUY POCO



(5) sub r4, r1, r4 //r4=r1-r4
(6) sub r5, r1, r3 //r5=r1-r3

¿En qué ciclo se empieza a ejecutar la instrucción (6)? (los ciclos se numeran desde 1). 5

19.- Para el procesador PROCEXB y la secuencia de seis instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿En qué ciclo empieza a ejecutarse la instrucción (2)?(los ciclos se numeran desde 1). 5

20.- Para el procesador PROCEXB y la secuencia de seis instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿Cuántos ciclos tarde en procesarse la secuencia de seis instrucciones indicada? 12

BURN.COM

#StudyOnFire

BURN
ENERGY DRINK

WUOLAH

1.- Suponga que en un mismo ciclo se decodifican las cuatro instrucciones siguientes

```
(i) add r2, r2, r1 // r2 = r2 + r1  
(i + 1) mul r3, r4, r1 // r3 = r4 * r1  
(i + 2) add r4, r4, r1 // r4 = r4 + r1  
(i + 3) sub r5, r2, r1 // r5 = r2 - r1
```

(entre paréntesis se indica el orden en el que están en el código), y pasan a una ventana de instrucciones única desde la que se emiten distintas unidades funcionales. El procesador puede emitir (con emisión DESORDENADA) TRES instrucciones por ciclo y tiene DOS unidades de suma/resta (con un retardo de 1 ciclo) y UN multiplicador (con un retardo de 3 ciclos).

Las instrucciones (i) e (i+3) se pueden emitir en el mismo ciclo, F

2.- En la predicción dinámica de un bit el sentido de la predicción no cambia (de saltar a no saltar o de no saltar a saltar) mientras el predictor no falle. V

3.- Un procesador de 32 bits (4 bytes) que implementa adelantamiento de loads a stores ESPECULATIVO podría adelantar la instrucción i+1 a la i (i precede a i+1 en el código)

```
(i) sw 0(r5), r2 // M(r5)<-- r2  
(i + 1) lw r4, 8(r5) //r4<--M(r5+8)
```

V

4.- Para el procesador PROCEXA y la secuencia de seis instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿Cuántos ciclos tarde en procesarse la secuencia de seis instrucciones indicada? 8

5.- El aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador VLIW necesita el uso del ROB. F

6.- El desenrollado de bucles necesita un hardware especial de apoyo. F

7.- Para el procesador PROCEXA y la secuencia de instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿En qué ciclo empieza a ejecutarse la instrucción (2)? (los ciclos se numeran desde 1). 5

8- Para el procesador PROCEXA y la secuencia de instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿En qué ciclo empieza a ejecutarse la instrucción (4)? (los ciclos se numeran desde 1). 8

9.- En la predicción dinámica de dos bits, el sentido de la predicción siempre cambia (de saltar a no saltar o de no saltar a saltar) cuando el predictor falla. F

10.- El procesador PROCEXA de 32 bits, puede captar TRES instrucciones por ciclo, DECODIFICAR TRES instrucciones por ciclo, y dispone de una ventana de instrucciones centralizada desde donde se produce la EMISIÓN DESORDENADA de hasta DOS instrucciones por ciclo. También dispone de un buffer de reordenamiento (ROB) donde implementa el renombramiento y la finalización ordenada, pudiendo retirarse desde el ROB, TRES instrucciones por ciclo para escribir sus resultados en la etapa WB del cauce. El procesador incluye además DOS unidades de Suma/Resta de UN ciclo; UN multiplicador de TRES ciclos, y UNA unidad de Carga/Almacenamiento de memoria de DOS ciclos.

Por tanto, las etapas del cauce del procesador son IF(capatación de instrucciones), ID(Decodificación de instrucciones), EX(ejecución en unidad funcional de la operación codificada por la instrucción), y WB(retirada de la instrucción del ROB y escritura de resultado en los registros del procesador). No se considera etapa ROB explícita en el cauce porque se supone que en el último ciclo de ejecución de las unidades funcionales, el resultado de la operación queda almacenado en el ROB

Para este procesador y la secuencia de instrucciones siguiente

(1) ld r1,0(r3) //r1=M(r3)

(2) ld r2,8(r3) //r2=M(r3+8)
(3) add r2, r1, r2 //r2=r1+r2
(4) mul r4, r1, r2 //r4=r1*r2
(5) sub r4, r1, r4 //r4=r1-r4
(6) sub r5, r1, r3 //r5=r1-r3

¿En qué ciclo se empieza a ejecutar la instrucción (6)? (los ciclos se numeran desde 1). 5

11.- La segmentación software es una técnica puramente software. V

12.- El aprovechamiento eficiente del paralelismo entre instrucciones (ILP) en un procesador VLIW NO requiere el uso del ROB. V

13.- Un procesador de 32 bits (4 bytes) que implementa adelantamiento de loads a stores, aunque no implementa el adelantamiento ESPECULATIVO, podría adelantar la instrucción i+1 a la i (i precede a i+1 en el código)

(i) sw 0(R6), r2 //r2=r2+r1
(i+1) lw r4, 8(r6) //r4<--M(r6+8)

V

15.- Suponga que en un mismo ciclo se decodifican las cuatro instrucciones siguientes

(i) add r2, r2, r1 // r2 = r2 + r1
(i + 1) mul r3, r4, r1 // r3 = r4 * r1
(i + 2) add r4, r4, r1 // r4 = r4 + r1
(i + 3) sub r5, r2, r1 // r5 = r2 - r1

(entre paréntesis se indica el orden en el que están en el código), y pasan a una ventana de instrucciones única desde la que se emiten distintas unidades funcionales. El procesador puede emitir (con emisión DESORDENADA) CUATRO instrucciones por ciclo y tiene TRES unidades de suma/resta (con un retardo de 1 ciclo) y UN multiplicador (con un retardo de 3 ciclos).

La instrucción (i) NO se puede emitir en el mismo ciclo que la (i+3), V

16.- Para el procesador PROCEXB y la secuencia de seis instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿En qué ciclo empieza a ejecutarse la instrucción (4)?(los ciclos se numeran desde 1). 8

17.- En la predicción dinámica de dos bits, el sentido de la predicción siempre cambia (de saltar a no saltar o de no saltar a saltar) cuando el predictor falla. F

18.- El procesador PROCEXB de 32 bits, puede captar TRES instrucciones por ciclo, DECODIFICAR TRES instrucciones por ciclo, y dispone de una ventana de instrucciones centralizada desde donde se produce la EMISIÓN DESORDENADA de hasta DOS instrucciones por ciclo. También dispone de un buffer de reordenamiento (ROB) donde implementa el renombramiento y la finalización ordenada, pudiendo retirarse desde el ROB, TRES instrucciones por ciclo para escribir sus resultados en la etapa WB del cauce. El procesador incluye además DOS unidades de Suma/Resta de UN ciclo; UN multiplicador de TRES ciclos, y UNA unidad de Carga/Almacenamiento de memoria de DOS ciclos.

Por tanto, las etapas del cauce del procesador son IF(capatación de instrucciones), ID(Decodificación de instrucciones), EX(ejecución en unidad funcional de la operación codificada por la instrucción), y WB(retirada de la instrucción del ROB y escritura de resultado en los registros del procesador). No se considera etapa ROB explícita en el cauce porque se supone que en el último ciclo de ejecución de las unidades funcionales, el resultado de la operación queda almacenado en el ROB

Para este procesador y la secuencia de instrucciones siguiente

(1) ld r1,0(r3) //r1=M(r3)
(2) ld r2,8(r3) //r2=M(r3+8)
(3) add r2, r1, r2 //r2=r1+r2
(4) mul r4, r1, r2 //r4=r1*r2

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



(5) sub r4, r1, r4 //r4=r1-r4
(6) sub r5, r1, r3 //r5=r1-r3

¿En qué ciclo se empieza a ejecutar la instrucción (6)? (los ciclos se numeran desde 1). 5

19.- Para el procesador PROCEXB y la secuencia de seis instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿En qué ciclo empieza a ejecutarse la instrucción (2)?(los ciclos se numeran desde 1). 5

20.- Para el procesador PROCEXB y la secuencia de seis instrucciones que se proporcionan en una de las preguntas de esta prueba: ¿Cuántos ciclos tarde en procesarse la secuencia de seis instrucciones indicada? 12



ARQUITECTURA DE COMPUTADORES
GRUPO A. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- p es el factor de incremento de prestaciones del recurso que se mejora (V)
- f es la fracción del tiempo antes de la mejora en el que no se utiliza el recurso mejorado (V)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/(1-f)$ (F)
- f puede ser mayor que 1 (F)

2. En un procesador superescalar a pleno rendimiento, el número de ciclos por instrucción (CPI) es menor que 1 (responda Verdadero, V, o Falso, F)

(V)

3. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo.

¿Cuál es la máxima velocidad (en GFLOPS) de un núcleo con dicha arquitectura que funciona a una frecuencia de reloj de 2.5 GHz?

$$8 \text{ FLOP/ciclo} * 2.5 \text{ (Gciclos/s)} = 20 \text{ GFLOPS}$$

4. Responda Verdadero (V) o Falso (F):

- En un computador NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida (V)
- Un multicomputador también se denomina computador UMA (F)

5. Si el bucle siguiente: for i=1 to N do a(i)=b(i)*c; se ejecuta en 5 segundos y $N=10^{12}$, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$1*10^{12} \text{ (FLOP)}/(5s*10^9)=1000/5 \text{ GFLOPS} = 200 \text{ GFLOPS}$$

6. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso comparten la memoria asignada al proceso, los registros, la pila y el contador de programa (F)
- Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse (F)
- Un multiprocesador puede funcionar como un computador MISD. (V)

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; $r1 \leftarrow r2 + r3$
- (b) sub r1, r1, r4 ; $r1 \leftarrow r1 - r4$

- Hay dependencia RAW entre las instrucciones debido al registro r1 (V)
- Hay dependencia WAR entre las instrucciones debido al registro r1 (F)

ARQUITECTURA DE COMPUTADORES

BENCHMARK del TEMA 1.

1. En la expresión de la ley de Amdahl, $S_p \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- S_p no puede ser nunca mayor que p según esa ley (V)

- f es la fracción del tiempo antes de la mejora, en la que se utiliza el recurso mejorado
Es la fracción de tiempo en la que NO se utiliza el recurso (F)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/f$ (V)

2. Un procesador con una frecuencia de reloj de 4 GHz ejecuta sus programas en menos tiempo que otro a una frecuencia de 2 GHz

El tiempo de CPU, además de depender de la frecuencia depende de NI y de CPI (F)

3. ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta tres instrucciones por ciclo y funciona a una frecuencia de reloj de 2 GHz?

$$3 \text{ (inst/ciclo)} * 2 * 10^9 \text{ (ciclos/s)} * (1/10^6) = 6000 \text{ MIPS}$$

4. La comunicación entre procesadores en un computador UMA se realiza a través de escrituras y lecturas en la memoria compartida, igual que en un computador NUMA

(V)

5. Un núcleo de procesamiento puede terminar hasta 4 operaciones en coma flotante (FLOP) por ciclo. ¿Cuál es su máxima velocidad (en GFLOPS) si funciona a una frecuencia de reloj de 2 GHz?

$$4 \text{ (op_float/ciclo)} * 2 * 10^9 \text{ (ciclos/s)} * (1/10^9) = 8 \text{ GFLOPS}$$

6. Si el bucle siguiente: for i=1 to N do a(i)=b(i)*c+a(i); se ejecuta en 10 segundos y $N=10^{14}$, siendo c, a(), y b() datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$2 * 10^{14} \text{ (op_float)} / (10 \text{ (seg.)}) * 10^9 = 20000 \text{ GFLOPS} = 20 \text{ TFLOPS}$$

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; r1 \leftarrow r2 + r3
- (b) sub r1, r2, r4 ; r1 \leftarrow r2 - r4
- (c) add r3, r2, r1 ; r3 \leftarrow r2 + r1

- Solo hay dependencias WAW y RAW debido al registro r1

También hay dependencias WAR debidas al registro r3 (F)

- No hay dependencias debido al uso del registro r2

(V)

ARQUITECTURA DE COMPUTADORES

BENCHMARK del TEMA 1.

1. En la expresión de la ley de Amdahl, $S_p \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- S_p no puede ser mayor que p según esa ley

En el mejor de los casos ($f=0$), $S_p \leq p$ (V)

- f es la fracción del tiempo antes de la mejora en la que no se utiliza el recurso mejorado

(V)

2. Escriba la expresión para el tiempo de CPU en términos del número de instrucciones (NI), el número de ciclos por instrucción (CPI), y la frecuencia de reloj (F).

$$T_{CPU} = (NI \times CPI) / F$$

3. Un procesador con una frecuencia de reloj de 3 GHz ejecuta sus programas en menos tiempo que otro a una frecuencia de 2.5 GHz

El tiempo de CPU depende de NI y de CPI, además de depender de la frecuencia (F)

4. Dado un programa en C, dos procesadores que tengan el mismo valor de IPC y frecuencia de reloj no necesariamente tardarán lo mismo en ejecutarlo

El T_{CPU} también depende del repertorio de instrucciones (a través de NI) (V)

5. ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta 4 instrucciones por ciclo y funciona a una frecuencia de reloj de 1 GHz?

$$4 \text{ (instrucciones/ciclo)} * 10^9 \text{ (ciclos/seg)} * (1/10^6) = 4000 \text{ MIPS}$$

6. El acrónimo NORMA significa No ORdered Memory Access ya que en este tipo de computador los procesadores necesitan sincronizarse para garantizar el acceso ordenado a la memoria compartida

NORMA significa NO Remote Memory Access (F)

7. Un núcleo de procesamiento puede terminar hasta 4 operaciones en coma flotante (FLOP) por ciclo. ¿Cuál es la máxima velocidad (en GFLOPS) de un microprocesador con dos núcleos de este tipo que funcionan a una frecuencia de reloj de 2 GHz?

$$4 \text{ (op_float/ciclo)} * 2 * 10^9 \text{ (ciclos/seg)} * (1/10^9) = 8 \text{ GFLOPS}$$

$$8 \text{ (GFLOPS/núcleo)} * 2 \text{ núcleos} = 16 \text{ GFLOPS}$$

8. Si el bucle siguiente: for i=1 to N do a(i)=b(i)*c+a(i); se ejecuta en 20 segundos y $N=10^{12}$, siendo c, a(), y b() datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$(2 * 10^{12} \text{ Op_float}) / (20 \text{ seg} * 10^9) = 100 \text{ GFLOPS}$$

9. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; r1 \leftarrow r2 + r3
- (b) sub r1, r2, r4 ; r1 \leftarrow r2 - r4
- (c) add r3, r2, r1 ; r3 \leftarrow r2 + r1

- Hay dependencia RAW entre las instrucciones debido al registro r2

El registro r2 solo se lee y por lo tanto no da lugar a dependencias (F)

ARQUITECTURA DE COMPUTADORES
GRUPO B. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- p es el factor de incremento de prestaciones del recurso que se mejora (V)
- f es la fracción del tiempo antes de la mejora en la que se utiliza el recurso mejorado (F)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/f$ (V)
- p puede ser mayor que 1 (V)

2. En un procesador segmentado a pleno rendimiento, el número de ciclos por instrucción (CPI) es (estrictamente) menor que 1 (responda Verdadero, V, o Falso, F)

(F)

3. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo.

¿Cuál es la máxima velocidad (en GFLOPS) de un microprocesador con 4 núcleos Sunday Bridge que funciona a una frecuencia de reloj de 2 GHz?

$$8 \text{ FLOP}/(\text{núcleo*ciclo}) * 2 \text{ Ciclos/s} * 4 \text{ núcleos} = 64 \text{ GFLOPS}$$

4. Responda Verdadero (V) o Falso (F):

- Un computador UMA, es un multiprocesador donde la memoria está físicamente distribuida. (F)
- Un multicomputador también se denomina computador NORMA (V)

5. Si el bucle siguiente: for i=1 to N do $a(i)=b(i)*c$; se ejecuta en 2 segundos y $N=10^{11}$, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$1*10^{11} \text{ FLOP} / 2 \text{ s} *10^9 = 100/2 \text{ GFLOPS} = 50 \text{ GFLOPS}$$

6. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse (F)
- El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos (V)
- Un multiprocesador puede funcionar como computador MISD con la correspondiente sincronización entre sus procesadores (V)

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; $r1 \leftarrow r2 + r3$
- (b) sub r1, r1, r4 ; $r1 \leftarrow r1 - r4$

- Hay dependencia WAW entre las instrucciones debido al registro r1 (V)
- No hay dependencia WAR entre las instrucciones debido al registro r1 (V)

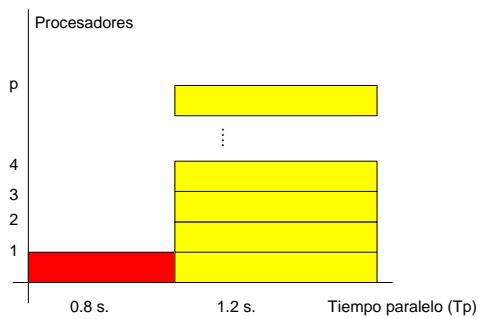
ARQUITECTURA DE COMPUTADORES

GRUPO A. BENCHMARK del TEMA 2

Estudiante:

- Escriba la expresión de la ley de Gustafson en términos de los parámetros f y p:

$$S_p = f + (1-f)p$$



- Teniendo en cuenta la figura anterior

- ¿Qué valor tiene el parámetro f en la ley de Gustafson:

$$f_g = 0.8 / 2.0$$

- Escriba el valor del parámetro f en la ley de Amdahl (en función del número de procesadores p)

$$f_a = 0.8 / (0.8 + 1.2p)$$

- Complete la siguiente Tabla de Ganancias de Velocidad:

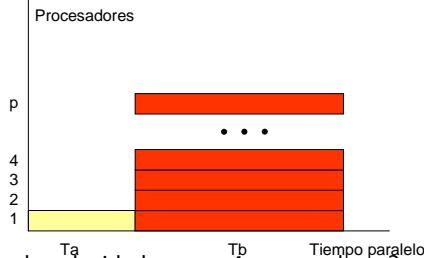
Fracción no paralela en T_s	Grado de Parallelismo	Overhead	Ganancia
0	ilimitado	0	p
f	ilimitado	0	$p / (1 + f(p-1))$
f	n	0	$p / (1 + f(p-1)) \text{ (} p \leq n \text{) y } n / (1 + f(n-1)) \text{ (} p > n \text{)}$
f	ilimitado	$T_o(p) = p$	$1 / (f + (1-f)/p + (p/T_s))$

- Responda Verdadero (V) o Falso (F):

- La reducción implica comunicación colectiva todos-a-uno (V)
- La acumulación (gather) implica comunicación colectiva todos-con-todos (F)
- MPI es una biblioteca de paso de mensajes (V)
- En la asignación de carga estática se asigna el trabajo que va a realizar cada procesador, antes de la ejecución (V)
- El tiempo de sincronización entre procesos forma parte del overhead de un programa paralelo (V)

ARQUITECTURA DE COMPUTADORES
BENCHMARK del TEMA 2

1. Suponiendo que en la figura $T_a=10$ s. y $T_b=30$ s.



¿Qué valor tiene la ganancia de velocidad para $p=4$ procesadores?

$$Ts = Ta + p * Tb = 10 + 4 * 30 = 130; Tp = 10 + 30 = 40$$

$$S = Ts / Tp = 130 / 40 = 13 / 4$$

¿Cuál es el valor de la f de la ley de Gustafson? $f = Ta / (Ta + Tb) = 10 / (10 + 30) = 1/4 = 0.25$

2. Complete la siguiente Tabla de Ganancias de Velocidad (Ts =tiempo secuencial):

Fracción no paralela en Ts	Grado de Paralelismo	Overhead	Ganancia para p procesadores (con $p > n$)	Ganancia para $p \rightarrow \infty$
0	ilimitado	$T_o(p) = p$	$1 / ((1/p) + (p/Ts))$ (también he dado por bueno si se supone $Ts=1$)	0
f	n	0	$1 / (f + ((1-f)/n))$	$1 / (f + ((1-f)/n))$
f	ilimitado	0	$1 / (f + ((1-f)/p))$	
0	n	$T_o(p) = p$	$1 / ((1/n) + (p/Ts))$ (también he dado por bueno si se supone $Ts=1$ y/o se utiliza n en el overhead)	

3. Responda Verdadero (V) o Falso (F):

- En la comunicación colectiva *all-scatter* todos los procesadores reciben información de todos, cosa que no ocurre en la comunicación *gossiping* (F)
- La asignación de carga dinámica afecta al tiempo de overhead del programa paralelo (V)
- En la comunicación colectiva *all-scatter* todos los procesadores reciben información de todos, cosa que también ocurre en la comunicación *gossiping* (V)
- La asignación de carga dinámica no afecta al tiempo de overhead del programa paralelo (F)
- En la comunicación colectiva de tipo *gossiping* todos los procesadores envían información, pero no todos los procesadores reciben (F)

Arquitectura de Computadores. Curso 2015/2016. Prueba de Teoría del Tema 4

Considere que en un mismo ciclo se decodifican las siguientes cuatro instrucciones (el índice i, i+1,... indica el orden en el que están en el código) y pasan a una ventana de instrucciones centralizada desde la que se emiten a las unidades funcionales. El procesador dispone de un buffer de reordenamiento (ROB) que también implementa el renombramiento. Además, tiene DOS unidades de suma, UN multiplicador, y una unidad de CARGA de memoria, y puede emitir CUATRO instrucciones en el mismo ciclo, con emisión DESORDENADA:

Nº inst	Instrucción	Significado	1	2	3	4	5	6	7	8	9
i	ld r2, 0(r3)	r2 ← m(r3+0)	EX	EX	ROB	WB					
i+1	mul r4,r2,r1	r4 ← r2 × r1			EX	EX	EX	EX	ROB	WB	
i+2	add r1,r1,r2	r1 ← r1+r2			EX	ROB				WB	
i+3	add r4,r3,r5	r4 ← r3+r5	EX	ROB							WB
Pregunta 5											
Pregunta 6											

1. Las instrucciones i e i+3 se podrían emitir en el mismo ciclo (V)
2. Las instrucciones i+1 e i+2 no se podrían emitir en el mismo ciclo porque entre ellas hay riesgo de tipo WAR
Los riesgos WAR desaparecen con el renombramiento que se hace en el ROB (F)
3. El compilador puede evitar el riesgo WAW entre las instrucciones i+1 e i+3 utilizando el registro r6 en lugar de r4 en la instrucción i+1 (V)
4. En el procesador que se ha descrito al comienzo, aunque la emisión es desordenada, la finalización de instrucciones es ordenada.
Hay un ROB que precisamente permite que la finalización sea ordenada (V)
5. Si en el procesador descrito la SUMA tiene un retardo de un ciclo, la CARGA de memoria un retardo de dos ciclos, y la MULTIPLICACIÓN un retardo de cuatro ciclos ¿Cuántos ciclos tardaría en EJECUTARSE la secuencia de cuatro instrucciones anterior?. (Contando desde el ciclo en que termina de ejecutarse la primera de las cuatro instrucciones)
4 ciclos (ver la tabla arriba)
6. Suponiendo que el ROB puede retirar dos instrucciones por ciclo. ¿En cuántos ciclos se retirarían las cuatro instrucciones anteriores (contando desde el ciclo en el que se retira la primera de las cuatro instrucciones)?
5 ciclos (ver la tabla arriba)
7. Las instrucciones de movimiento condicional de datos permiten reducir el número de instrucciones de salto condicional en los códigos. (V)
8. La predicción de saltos dinámica implícita para una instrucción de salto condicional, i, utiliza el resultado (salto o no salto) de la ejecución previa de dicha instrucción de salto condicional, i, para hacer la predicción (V)
9. Los procesadores WLIW no tienen buffers de renombramiento porque la planificación de instrucciones la realiza el compilador. (V)

10. Considere las dos instrucciones siguientes (la instrucción i precede a la i+1 en el código)

(i) sw 0(r5),r2 // m(r5+0)←r2

(i+1) ld r4,0(r6) // r4←m(r6+0)

Un procesador que NO implemente adelantamiento ESPECULATIVO de LOADs a STORES podría adelantar la ejecución de i+1 a la de i

Debe haber recursos para el adelantamiento especulativo porque podría ocurrir que r5 sea igual a r6 y se violaría el riesgo RAW si se produce el adelantamiento (F)

ARQUITECTURA DE COMPUTADORES

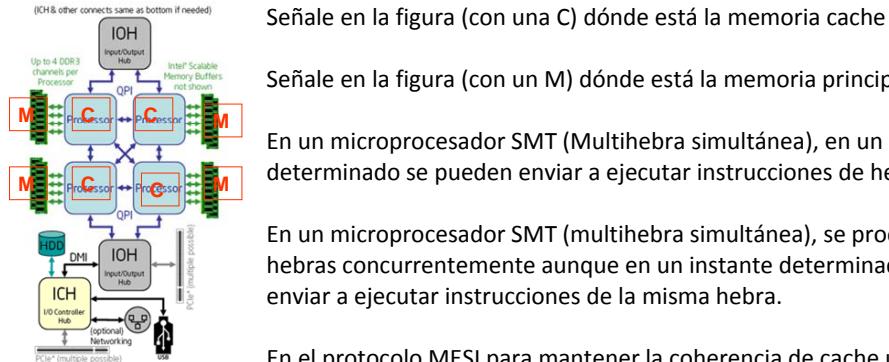
BENCHMARK del TEMA 3

Un microprocesador multinúcleo no incluye memoria cache

(F)

El servidor de la figura tiene una arquitectura de tipo UMA

(F)



(V)

En el protocolo MESI para mantener la coherencia de cache una línea puede estar en el estado S solo en una de las caches del multiprocesador

(F)

En el protocolo MESI para mantener la coherencia de cache, una línea puede estar en el estado E en varias caches del multiprocesador

(F)

En el protocolo MSI, si en la cache de un nodo N1 hay un bloque en estado M (Modificado), y ese nodo detecta que otro procesador intenta leer un dato que está en ese bloque, el bloque pasa al estado I (Inválido) en el nodo N1

(F)

En un multiprocesador NUMA con 16 nodos, 4 GBytes por nodo, y líneas de cache de 64 Bytes.

¿Cuántas entradas tiene el directorio de memoria utilizado en cada nodo para mantener la coherencia de cache en un protocolo MSI sin difusión?

$$2^2 \cdot 2^{30} / 2^6 = 2^{26} \text{ entradas}$$

En el multiprocesador NUMA descrito en la pregunta anterior ¿Cuántos bits tiene cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache?

$$16+1 = 17$$

¿Qué valores se puede observar en R si el modelo de consistencia de memoria del computador donde están los procesadores que ejecutan estos códigos no respeta el orden W→W (sí respeta los demás), e inicialmente X=Y=0?

P1: X=2 Y=1	P2: R=1; if (Y==1) R=X;
---------------------	---------------------------------

R=0; R=1; R=2

Si respeta el orden W→W: R=1; R=2

¿Qué valor deben r1, r2, y r3 para que la secuencia de instrucciones siguiente implemente un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y 0 que está abierto.

```
b=r1;
do
    compare&swap(r2,b,k); // si r2==k b se intercambian
    while (b==r3);
```

r1=1

r2=0

r3=1

Arquitectura de Computadores

Benchmark de Teoría del Tema 4

Suponga que en un mismo ciclo se decodifican estas cuatro instrucciones (el número entre paréntesis indica en orden en el que están en el código), y pasan a una ventana de instrucciones única desde la que se emiten a las distintas unidades funcionales. El procesador tiene un buffer de reorden (ROB) para la finalización ordenada donde también se implementa el renombrado.

(i)	add r2, r2, r1 // r2 = r2+r1
(i+1)	mul r3, r4, r1 // r3 = r4*r1
(i+2)	add r4, r3, r1 // r4 = r3+r1
(i+3)	add r1, r2, r1 // r1 = r2+r1

- Si el procesador tiene TRES unidades de suma y UN multiplicador y puede emitir cuatro instrucciones por ciclo con emisión DESORDENADA, se podrían emitir las cuatro instrucciones en el mismo ciclo. **Hay dependencias RAW** (F)
- Entre la instrucción (i+1) y la (i+2) SOLO hay un riesgo de tipo WAR. **También hay dependencia RAW en r3 aparte de la WAR en r4** (F)
- Si el procesador descrito anteriormente puede emitir (con emisión DESORDENADA) DOS instrucciones por ciclo y tiene DOS unidades de suma (con un retardo de 2 ciclos) y UN multiplicador (con un retardo de 5 ciclos). ¿Cuántos ciclos tardan en emitirse las cuatro instrucciones (cuente a partir del ciclo de la decodificación, sin incluir este)?

	1	2	3	4	5	6	7	8	9
(i)	EX	EX	ROB	WB					
(i+1)	EX	EX	EX	EX	EX	ROB	WB		
(i+2)						EX	EX	ROB	WB
(i+3)			EX	EX	ROB				WB

Tardan 5 ciclos en emitirse las instrucciones y 9 en retirarse todas (suponiendo que se pueden retirar más de dos instrucciones por ciclo)

- En la predicción de salto dinámica implícita, cada vez que llega una instrucción de salto condicional se predice que NO se va a producir el salto (**es dinámica, luego debe cambiar según lo que ocurre en la ejecución del código. En este caso, se hace lo que se hizo en la última ejecución**) (F)
- Indique cómo expresaría sin instrucciones de salto utilizando un repertorio típico de instrucciones con predicado en el que TODA instrucción se puede predicar (suponga que LOS PREDICADOS ESTÁN INICIALIZADOS A 0):
 $\text{if } (r2==0) \text{ then } r2=r2+r3 \text{ else } r2=r2-r3;$

(p1)	p1,p2 cmp.eq r2,0
(p1)	add r2,r2,r3
(p2)	sub r2,r2,r3
- NO existen instrucciones de ejecución condicional en los repertorios de instrucciones de procesadores superescalares
- La segmentación software NO necesita un hardware especial de apoyo (F)

¿BUSCAS LIBROS O EBOOKS SOBRE ENFERMERÍA?

AXON librería especializada en Ciencias de la Salud.

AXON
axon.es



Encuentra todos los libros y eBooks para tu especialización, además del material complementario que necesites.

8. La responsabilidad del aprovechamiento del paralelismo entre instrucciones (ILP) en un procesador VLIW recae fundamentalmente en el compilador (V)
9. Los procesadores superescalares no pueden tener un repertorio de instrucciones CISC, por eso todos los núcleos de Intel son VLIW (V)
10. Considere las dos instrucciones siguientes (i precede a i+1 en el código) (F)

(i)	sw 0(r5), r2 // M(r5) ← r2
(i+1)	lw r4, 32(r5) // r4 ← M(r5+32)

Para que un procesador adelante la ejecución de (i+1) a la de (i), DEBE IMPLEMENTAR adelantamiento de loads a stores ESPECULATIVO (**la dirección a la que se hace el almacenamiento, r5+0, siempre va a ser distinta de la dirección desde donde se lee, r5+32**)

(F)

Test Tema 2 Curso 2019/2020 - AC

1. En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) prefijo paralelo, el procesador P2 recibe información de los procesadores P0, P1, y del propio P2 (aparte de otras posibles comunicaciones)

V

2. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a 0 es igual a p para $p < n$

V

3. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a p es $T_s / ((T_s/n) + n)$, para $p = n$

V

4. La falta de equilibrado de la carga es una de las causas de que haya tiempo de sobrecarga u overhead en los programas paralelos

V

5. La expresión para la ley de Gustafson es $S = (1-f) + p*f$, donde f es la fracción no paralelizable del tiempo de ejecución paralelo y p es el número de procesadores que intervienen

F

6. En un multicomputador con 4 procesadores (P0 a P3), mediante la permutación de rotación, el procesador P0 envía información al procesador P1 y recibe del P2 (aparte de otras posibles comunicaciones)

F

7. Un programa secuencial tarda 40 ns en ejecutarse en un procesador y durante 10 ns de esos 40 ns el programa no es paralelizable. El valor de la f de la ley de Amdahl para ese programa es igual a 0.75

F

8. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a f, un grado de paralelismo ilimitado y un tiempo de overhead igual a 0 es $p/(1+f(p-1))$

V

9. Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de

cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es 4

F

10. La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo ilimitado y un tiempo de overhead igual a p^2 es $T_s/((T_s/p)+p^2)$

V

1-En un multiprocesador, el procesador P1 ejecuta las instrucciones

- (1) while (Z==0) { };
- (2) r1=Y;

en paralelo con las instrucciones que ejecuta el procesador P2:

- (a) X=1;
- (b) Y=2;
- (c) Z=1;

Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden W→R (SÍ respeta todos los demás), e inicialmente X=Y=Z=r1=0 (donde X,Y,y Z son variables en memoria compartida y r1 es un registro de P1), al final se podría tener r1=2

V

2-En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo puede haber una entrada en uno de los directorios con un único bit a uno (hay una copia del bloque correspondiente en una cache de la máquina) y el bit de estado del bloque en memoria igual a 1 (estado Válido en memoria)

V

3-En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo NO puede haber una entrada en uno de los directorios con varios bits a uno (hay copias del bloque correspondiente en varias caches de la máquina) y el bit de estado del bloque en memoria igual a 0 (estado No válido en memoria)

V

4-En un multiprocesador NUMA con 8 nodos, 8 GBytes por nodo, y líneas de cache de 128 Bytes, el número de bits que tiene cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache en un protocolo MSI con directorio y codificación de bit completo es igual a 5

F

5-Cuando se utilizan instrucciones del tipo LL/SC (lectura enlazada/escritura condicional) para implementar un cerrojo, los recursos hardware asociados a dicha técnica permiten detectar si, entre la ejecución de la lectura enlazada (LL) y la ejecución de la escritura condicional (SC) a la dirección de memoria del cerrojo, algún otro procesador ha accedido a dicha dirección

V

6-El código siguiente permite implementar un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y k=0 que está abierto:

```
b=0; k=1;  
while (fetch_and_or(k,b)==1) {};
```

F

7-En el protocolo MSI de espionaje para la coherencia, si en la cache de un nodo N1 hay un bloque B en estado M (Modificado) y detecta que el nodo N2 intenta escribir en un dato del mismo bloque B, dicho bloque pasará al estado M (Modificado) en la caché del nodo N2, y al estado I en la caché del nodo N1 tras actualizarse en la memoria principal.

V

8-En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado S (Compartido), y ese nodo detecta que el nodo N2 intenta leer un dato del mismo bloque B, dicho bloque pasará al estado S (Compartido) en la caché del nodo N2, y se mantendrá en el estado S en la caché del nodo N1

V

9-En un multiprocesador NUMA con 8 nodos, 8 GBytes por nodo, y líneas de cache de 128 Bytes, el directorio de memoria utilizado en cada nodo para mantener la cache en un protocolo MSI sin difusión tiene 2^{26} (2 elevado a 26) entradas

V

10-En un microprocesador SMT (multihebra simultánea), se procesan varias hebras concurrentemente y en un instante determinado solo se pueden enviar a ejecutar instrucciones de una misma hebra.

F

11-Los microprocesadores SMT(multihebra simultánea), pueden ejecutar varias instrucciones de una misma hebra en paralelo

V

12-En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo puede haber una entrada en uno de los directorios con un único bit a uno (hay una copia del bloque correspondiente en una cache de la máquina) y el bit de estado del bloque en memoria igual a 0 (estado No válido en memoria)

V

13- En un multiprocesador, el procesador P1 ejecuta las instrucciones

- (1) while (Z==0) { };
- (2) r1=Y;

en paralelo con las instrucciones que ejecuta el procesador P2:

- (a) X=1;
- (b) Y=2;
- (c) Z=1;

Si el modelo de consistencia de memoria de un multiprocesador NO respeta el orden W→R (SÍ respeta todos los demás), e inicialmente X=Y=Z=r1=0 (donde X,Y,y Z son variables en memoria compartida y r1 es un registro de P1), al final se podría tener r1=0

V

14- El código siguiente permite implementar un cerrojo (lock(k)) en el que k=1 significa que el cerrojo está cerrado y k=0 que está abierto:

```
b=0; k=1;  
while (fetch_and_add(k,b)==1) {};
```

F

15- En el protocolo MESI, si en la cache de un nodo N1 hay un bloque B en estado S (Compartido), y ese nodo detecta que el nodo N2 intenta escribir un dato del mismo bloque B, dicho bloque pasará al estado E (Exclusivo) en la caché del nodo N2, y se mantendrá en el estado S en la caché del nodo N1

F

16- En un multiprocesador NUMA con protocolo MSI basado en directorios de vector de bits completo NO puede haber una entrada en uno de los directorios con todos sus bits de copias a cero (hay una copia del bloque correspondiente en una cache de la máquina) y el bit de estado del bloque en memoria igual a 1 (estado No válido en memoria)

F

17- En un multiprocesador NUMA con 16 nodos, 4 GBytes por nodo, y líneas de cache de 64 Bytes, el directorio de memoria utilizado en cada nodo para mantener la cache en un protocolo MSI sin difusión tiene 2^{25} (2 elevado a 25) entradas

F

18- Cuando se utilizan instrucciones del tipo LL/SC (lectura enlazada/escritura condicional) para implementar un cerrojo, los recursos hardware asociados a dicha técnica impiden que, entre la ejecución de la lectura enlazada (LL) y la ejecución de la escritura condicional (SC) a la dirección de memoria del cerrojo, ningun otro procesador pueda acceder a dicha dirección de memoria del cerrojo

F

19- Si una linea de la caché del nodo N1 está en el estado M del protocolo MSI para mantener la coherencia de caché, el contenido de esa línea es coherente con su contenido en memoria principal.

F

20- En un multiprocesador NUMA con 16 nodos, 8 GBytes por nodo, y líneas de cache de 64 Bytes, el número de bits que tiene cada una de las entradas del directorio que se utiliza para mantener la coherencia de cache en un protocolo MSI con directorio y codificación de bit completo es igual a 9

F

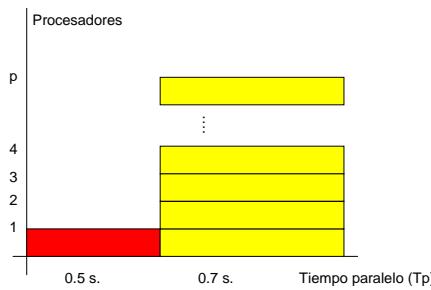
ARQUITECTURA DE COMPUTADORES

GRUPO B. BENCHMARK del TEMA 2

Estudiante:

- Escriba la expresión de la ley de Gustafson en términos de los parámetros f y p:

$$S_p = f + (1-f)p$$



- Teniendo en cuenta la figura anterior

- ¿Qué valor tiene el parámetro f en la ley de Gustafson:

$$f_g = 0.5 / 1.2$$

- Escriba el valor del parámetro f en la ley de Amdahl (en función del número de procesadores p)

$$f_a = 0.5 / (0.5 + 0.7p)$$

- Complete la siguiente Tabla de Ganancias de Velocidad:

Fracción no paralela en T_s	Grado de Paralelismo	Overhead	Ganancia
0	ilimitado	0	p
f	ilimitado	0	$p / (1 + f(p-1))$
f	n	0	$p / (1 + f(p-1)) \text{ (} p \leq n \text{)} \text{ y } n / (1 + f(n-1)) \text{ (} p > n \text{)}$
f	ilimitado	$T_o(p) = p$	$1 / (f + (1-f)/p + (p/T_s))$

- Responda Verdadero (V) o Falso (F):

- La difusión (broadcast) implica comunicación colectiva de todos-con-todos (F)
- La dispersión (scatter) implica comunicación colectiva todos-con-todos (F)
- OpenMP es una biblioteca que permite hacer programas paralelos con paso de mensajes (F)
- El tiempo de comunicación entre procesos forma parte del overhead de un programa paralelo (V)
- La asignación de carga dinámica se realiza antes de la ejecución del programa paralelo (F)



2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores: Exámenes y Controles

Examen de Prácticas 04/09/2012 resuelto

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita, Julio Ortega

Licencia Creative Commons



1 Enunciado Examen de Prácticas del 04/09/2012

Cuestión 1.(1 punto) Conteste a las siguientes cuestiones sobre el código examen1sep.c de la siguiente figura (considere que la variable de control `dyn_var` está a `false`):

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int n, ck, i, nth, ith, b[32];
    if(argc < 4) { printf("[ERROR]-Faltan parámetros\n"); exit(-1); }
    n = atoi(argv[1]); ck = atoi(argv[2]); nth = atoi(argv[3]);
    if(n>32) n=32; omp_set_num_threads(nth);
    #pragma omp parallel if(n>8) private(ith)
    { int a = 0; ith=omp_get_thread_num();
        #pragma omp single copyprivate(a)
        { printf("\nIntroduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("\nSingle ejecutada por el thread %d\n", ith);
        }
        #pragma omp for schedule(static,ck)
        for (i=0; i<n; i++) b[i] = ith + a
    }
    for (i=0; i<n; i++) printf(b[%d]=%d, i, b[i]);
    printf("\n");
}
```

- (a) (0.2) Indique qué hace el código y la orden que usaría para compilar desde una ventana de comandos (*Shell* o intérprete de comandos) si el ejecutable se quiere llamar `examen1sep` (utilice en la compilación alguna de las opciones de optimización que ha utilizado en la práctica de optimización de código).
- (b) (0.4) Razone qué `printf` de los que hay en el código se ejecutan, qué threads ejecutan cada uno de ellos y qué imprime el programa en cada uno de estos `printf` si el usuario ejecuta: `examen1sep 20 2 4.`
- (c) (0.2) Razone qué imprime el programa en cada uno de los `printf` si el usuario ejecuta: `examen1sep 8 2 4.`

- (d) (0.2) Razone qué imprime el programa en cada uno de los `printf` si el usuario ejecuta: `examen1sep 13 4 3.`

Cuestión 2. (1 punto) Conteste a las siguientes cuestiones sobre el código `examen2sep.c` de la siguiente figura (considere que la variable de control `dyn_var` está a `false`):

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n, a[n], suma=0, sumap=0;
    if (argc < 2) { fprintf(stderr, "\nFalta iteraciones\n"); exit(-1); }
    n = atoi(argv[1]); if (n>20) n=20;
    for (i=0; i<n; i++) a[i] = i;
    #pragma omp parallel
    {
        #pragma omp for schedule(static,1)
        for (i=0; i<n; i++) sumap+= a[i];
        #pragma omp atomic
        suma += sumap;
    }
    printf("Fuera de 'parallel' suma=%d sumap=%d\n",suma, sumap); return(0);
}
```

- (a) (0.2) Razone qué imprime el programa en el último `printf` si el usuario ejecuta `examen2sep 20`.
- (b) (0.3) Añada lo necesario al código (sin eliminar nada) para que al imprimir la variable `suma` se obtenga la suma de los `n` componentes del vector `a`. Razone las modificaciones realizadas. Razone qué valor numérico se obtiene en la pantalla cuando se imprimen `suma` y `sumap` si se ejecuta en las aulas de prácticas en una ventana de comandos lo siguiente:

```
> export OMP_NUM_THREADS=2
> examen2sep 10
```

(`examen2sep` se ha generado con el código fuente de este apartado (b))

- (c) (0.2) Aplique desenrollado de bucles al código resultante del apartado (b). Escriba el código con las modificaciones realizadas.
- (d) (0.3) Si se usa la cláusula `reduction` en la versión de código resultante del apartado (c), ¿qué eliminaría del código y qué añadiría al código y por qué? Escriba el código con las modificaciones descritas.

2 Solución Examen de Prácticas del 04/09/2012

Cuestión 1.(1 punto) Conteste a las siguientes cuestiones sobre el código `examen1sep.c` de la siguiente figura (considere que la variable de control `dyn_var` está a `false`):

```

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
(1) int n, ck, i, nth, ith, b[32];
(2) if(argc < 4) { printf("[ERROR]-Faltan parámetros\n"); exit(-1); }
(3) n = atoi(argv[1]); ck = atoi(argv[2]); nth = atoi(argv[3]);
(4) if(n>32) n=32; omp_set_num_threads(nth);
(5) #pragma omp parallel if(n>8) private(ith)
(6) { int a = 0; ith=omp_get_thread_num();
(7) #pragma omp single copyprivate(a)
(8) { printf("\nIntroduce valor de inicialización a: "); scanf("%d", &a );
(9) printf("\nSingle ejecutada por el thread %d\n", ith);
(10) }
(11) #pragma omp for schedule(static,ck)
(12) for (i=0; i<n; i++) b[i] = ith + a
(13)
(14) for (i=0; i<n; i++) printf(b[%d]=%d, ,i, b[i]);
(15) printf("\n");
}

```

- (a) (0.2) Indique qué hace el código y la orden que usaría para compilar desde una ventana de comandos (*Shell* o intérprete de comandos) si el ejecutable se quiere llamar `examen1sep` (utilice en la compilación alguna de las opciones de optimización que ha utilizado en la práctica de optimización de código).
- (b) (0.4) Razone qué `printf` de los que hay en el código se ejecutan, qué threads ejecutan cada uno de ellos y qué imprime el programa en cada uno de estos `printf` si el usuario ejecuta: `examen1sep 20 2 4.`
- (c) (0.2) Razone qué imprime el programa en cada uno de los `printf` si el usuario ejecuta: `examen1sep 8 2 4.`
- (d) (0.2) Razone qué imprime el programa en cada uno de los `printf` si el usuario ejecuta: `examen1sep 13 4 3.`

Solución

(a) El código inicializa en paralelo los `n` componentes de un vector `b` (líneas (11) (12)) a partir de un valor `a` que se pide al usuario que introduzca (líneas (9)-(10)) y del identificador de los `nth` threads (`ith=0,1,...nth-1`) que ejecutan en paralelo la inicialización. La componente `i` de `b` se inicializa con `a+ith` (líneas (12)). Por tanto, el valor de `b[i]` va a depender del thread que ejecute la iteración `i` del bucle. Debido a la cláusula `if (n>8)` que acompaña a la directiva `parallel`, si `n` es menor o igual que 8 inicializa `b` sólo un thread, el máster (en este caso `ith=0` para todos los componentes de `b`). Los valores de `b` tras la inicialización se imprimen en pantalla.

Para compilar se usaría: `gcc -O2 -fopenmp -o examen1sep examen1sep.c`

(b) Hay un total de 5 `printf` en el código, uno antes de la construcción `parallel` (línea (2)), dos en la construcción `parallel` (líneas (8) (9)), que están dentro de la construcción `single`, y dos después de la construcción `parallel` (líneas (14) (15)). Para la ejecución `examen1sep 20 2 4` se ejecutan todos los `printf` excepto el primero:

- ❖ El primero (línea (2)) se ejecuta cuando el número de parámetros o argumentos en el comando que ejecuta el programa es menor que tres. En la ejecución de este apartado hay tres parámetros, luego no se ejecuta este primer `printf`.
- ❖ Los dos `printf` de la construcción `single` (líneas (8) (9)) los ejecutará sólo un thread de los que ejecutan la región `parallel`, el primero que llegue a la región `single`. El primer `printf` imprime, tras un salto de línea “`\n`”, el siguiente aviso al usuario “`Introduce valor de inicialización a:`”. El segundo (“`\nSingle ejecutada por el thread %d\n`”, `ith`) imprime un salto de línea, el texto “`Single ejecutada por el thread`”, el identificador del thread, `ith`, que ejecuta el código de la construcción `single` y otro salto de línea.
- ❖ Los dos `printf` que hay después de la construcción `parallel` (líneas (14) (15)) los ejecuta el thread 0, el master. El segundo `printf` imprimen un salto de línea. El primero está dentro de un `for` que imprime el contenido del vector `b` que se ha inicializado en paralelo dentro de la región `parallel`. El valor que se imprime para `b[i] (=a+ith)` va a depender del thread, `ith`, que ejecute la iteración `i` del bucle. El código que hay dentro de la construcción `parallel` fuera de la construcción `single` lo ejecutará además del thread 0 todos los threads que se crean por la directiva `parallel`. Según la cláusula `schedule(static, ck)` de la directiva `for`, la asignación de las iteraciones del bucle `for` de la construcción `parallel` se realiza con una planificación estática (`static`) que reparte unidades de trabajo de `ck` iteraciones consecutivas del bucle en turno rotatorio (round-robin) entre los `nth` threads. Dado el código “`n = atoi(argv[1]); ck = atoi(argv[2]); nth = atoi(argv[3]);`” (línea (3)) tendríamos que :
 - El primero de los argumentos, `argv[1]`, se almacena en la variable `n`, que se usa como número de componentes del vector `b` que se van a utilizar en el código y es el número de iteraciones de los dos bucles, en particular, del `for` de la construcción `parallel`.
 - El segundo de los argumentos, `argv[2]`, se almacena en la variable `ck`, que se usa como el número de iteraciones del bucle que contienen las unidades de código que se van a usar en la asignación de trabajo a los threads. Si `ck` no divide a `n` (número de iteraciones del bucle `for`) entonces habrá un trozo con menos de `ck` iteraciones.
 - El tercero de los argumentos, `argv[3]`, se almacena en la variable `nth`, que se usa como número de threads que van a ejecutar la región paralela del código.

Teniendo en cuenta el comando utilizado para ejecutar el programa, `examen1sep 20 2 4`, los valores de `n`, `ck` y `nth` son 20, 2 y 4 respectivamente; por tanto:

- Como `n=20` es mayor que 8 (cláusula `if` de la directiva `parallel`) la región paralela la ejecutan los 4 threads (`nth=4`) especificados en el último parámetro de entrada.
- Las 20 iteraciones del bucle se dividen en unidades de trabajo de 2 (`ck=2`) iteraciones cada una.
- Como hay 20 iteraciones (`n=20`), se tienen $20/2 = 10$ unidades.

La asignación a los 4 threads de las 20 iteraciones del bucle y, por tanto, las asignaciones de las componentes de `b` a threads es la mostrada en la Figura 1.

<code>ith =</code>	0	1	2	3	0	1	2	3	0	1
<code>i =</code>	0 1	2 3	4 5	6 7	8 9	10 11	12 13	14 15	16 17	18 19

Figura 1. Reparto de las iteraciones (`i=0,1,...19`) del bucle en unidades de dos iteraciones y asignación de unidades a los `nth=4` threads (`ith=0,1,2 y 3`)

Según esta asignación el penúltimo `printf` del código (línea (14)) imprimiría lo siguiente si, por ejemplo, `a` es 0:

`b[0]=0, b[1]=0, b[2]=1, b[3]=1, b[4]=2, b[5]=2, b[6]=3, b[7]=3, b[8]=0, b[9]=0, b[10]=1, b[11]=1, b[12]=2, b[13]=2, b[14]=3, b[15]=3, b[16]=0, b[17]=0, b[18]=1, b[19]=1.`

Para `a=10` imprimiría:

`b[0]=10, b[1]=10, b[2]=11, b[3]=11, b[4]=12, b[5]=12, b[6]=13, b[7]=13, b[8]=10, b[9]=10, b[10]=11, b[11]=11, b[12]=12, b[13]=12, b[14]=13, b[15]=13, b[16]=10, b[17]=10, b[18]=11, b[19]=11.`

(c) En este caso se aplica lo comentado en el apartado anterior (b) excepto en lo que respecta a los valores concretos que imprime el penúltimo `printf` (el que imprime los componentes del vector `b`).

Teniendo en cuenta el comando utilizado para ejecutar el programa, `examen1sep 8 2 4`, los valores de `n`, `ck` y `nth` son 8, 2 y 4 respectivamente:

- Como `n=8` NO es mayor que 8, debido a la cláusula `if` de la directiva `parallel` (línea (5)), la región paralela la ejecutará sólo el thread 0, el *master*.

Todas las iteraciones se asignarán, por tanto, al thread 0. Según esta asignación el penúltimo `printf` del código imprime lo siguiente si, por ejemplo, `a` es igual a 10:

`b[0]=10, b[1]=10, b[2]=10, b[3]=10, b[4]=10, b[5]=10, b[6]=10, b[7]=10, b[8]=10, b[9]=10, b[10]=10, b[11]=10, b[12]=10, b[13]=10, b[14]=10, b[15]=10, b[16]=10, b[17]=10, b[18]=10, b[19]=10.`

(d) En este caso también se aplica lo comentado en el apartado anterior (b) excepto en lo que respecta a los valores concretos que imprime el penúltimo `printf` (el que imprime los componentes del vector `b`).

Teniendo en cuenta el comando utilizado para ejecutar el programa, `examen1sep 13 4 3`, los valores de `n`, `ck` y `nth` son 13, 4 y 3 respectivamente, por tanto:

- Como `n=13` es mayor que 8 (cláusula `if` de la directiva `parallel`) la región paralela la ejecutan los 3 threads (`nth=3`) fijados con el último parámetro.
- Las 13 iteraciones del bucle se dividen en unidades de trabajo de 4 (`ck=4`) iteraciones cada una, excepto una de las unidades que tendrá menos iteraciones debido a que 4 no divide a 13.
- Como hay 13 iteraciones (`n=13`), se tienen $13/4 = 3$ unidades de 4 iteraciones y 1 unidad de 1 ($=13 \bmod 4$) iteración.

La asignación a los 3 threads de las 13 iteraciones del bucle y, por tanto, las asignaciones de los componentes de `b` a threads es la mostrada en la Figura 2.

<code>ith =</code>	0	1	2	0
<code>i =</code>	0 1 2 3	4 5 6 7	8 9 10 11	12

Figura 2. Reparto de las iteraciones (`i=0,1,...12`) del bucle en unidades de cuatro iteraciones y asignación de unidades a los `nth=3` threads (`ith=0,1,2`)

Según esta asignación el penúltimo `printf` del código imprime lo siguiente si, por ejemplo, `a` es 10:

`b[0]=10, b[1]=10, b[2]=10, b[3]=10, b[4]=11, b[5]=11, b[6]=11, b[7]=11, b[8]=12, b[9]=12, b[10]=12, b[11]=12, b[12]=10.`



Cuestión 2.(1 punto) Conteste a las siguientes cuestiones sobre el código examen2sep.c de la siguiente figura (considere que la variable de control `dyn_var` está a `false`):

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n, a[n], suma=0, sumap=0;
    if (argc < 2) { fprintf(stderr, "\nFalta iteraciones\n"); exit(-1); }
    n = atoi(argv[1]); if (n>20) n=20;
    for (i=0; i<n; i++) a[i] = i;
    #pragma omp parallel
    {
        #pragma omp for schedule(static,1)
        for (i=0; i<n; i++) sumap+= a[i];
        #pragma omp atomic
        suma += sumap;
    }
    printf("Fuera de 'parallel' suma=%d sumap=%d\n", suma, sumap);
    return(0);
}
```

- (a) (0.2) Razone qué imprime el programa en el último `printf` si el usuario ejecuta `examen2sep 20`.
 (b) (0.3) Añada lo necesario al código (sin eliminar nada) para que al imprimir la variable `suma` se obtenga la suma de los `n` componentes del vector `a`. Razone las modificaciones realizadas. Razone qué valor numérico se obtiene en la pantalla cuando se imprimen `suma` y `sumap` si se ejecuta en las aulas de prácticas en una ventana de comandos lo siguiente:

```
> export OMP_NUM_THREADS=2
> examen2sep 10
```

(examen2sep se ha generado con el código fuente de este apartado (b))

- (c) (0.2) Aplique desenrollado de bucles al código resultante del apartado (b). Escriba el código con las modificaciones realizadas.
 (d) (0.3) Si se usa la cláusula `reduction` en la versión de código resultante del apartado (c), ¿qué eliminaría del código y qué añadiría al código y por qué? Escriba el código con las modificaciones descritas.

Solución

(a) El código imprime la variable compartida `suma` que contendrá un valor igual al número de thread que ejecutan la región paralela multiplicado por el valor que tenga la variable compartida `sumap` al terminar la ejecución paralela del bucle `for` de la región paralela. La directiva `for` añade una barrera implícita. También imprime la variable compartida `sumap`, su valor no es determinístico; es decir, la ejecución del código varias veces con los mismos parámetros de entrada no imprime siempre lo mismo. Esto ocurre porque los threads acceden a `sumap` en paralelo y todos la leen, modifican y escriben (R-M-W): `sumap+=a[i]`. Si los threads accedieran secuencialmente (es decir, un thread detrás de otro) a

`sumap+=a[i]` (R-M-W) al imprimir `sumap` se imprimiría la suma de todos los componentes del vector `a`.

(b) Para que, al imprimir `suma`, se obtenga la suma de `n` los componentes de `a`, la variable `sumap` debe ser privada a los threads y debe estar inicializada a 0 en todos los threads. Para conseguirlo, sin eliminar código, podemos usar una cláusula `firstprivate(sumap)` en la directiva `parallel`. Con esta opción se sustituiría el código “`#pragma omp parallel`” por este otro “`#pragma omp parallel firstprivate(sumap)`”. Con `firstprivate` además de crear una variable privada `sumap` en todos los threads estas variables privadas se inicializan con el valor que tiene la variable global `sumap`, que en este caso es 0.

Si se ejecuta en una ventana de comandos:

```
> export OMP_NUM_THREADS=2
> examen2sep 10
```

se imprime lo siguiente:

```
> Fueras de 'parallel' suma=45 sumap=0
```

porque:

1. Al imprimir `suma` se imprime un valor que coincide con la suma de los componentes del vector `a`. Las 10 componentes de `a` se inicializa de forma que `a[i]=i`; por tanto, la suma de todas ellas será igual a $10*(10-1)/2=45$.
2. Al imprimir `sumap` se imprime 0, que es el valor al que se inicializó la variable compartida `sumap`.

(c) Se va a aplicar, por ejemplo, un desenrollado de 2 iteraciones del bucle, es decir, en cada iteración del nuevo bucle se va a realizar dos iteraciones del bucle original:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n=20, a[n], suma=0, sumap=0, sumap2=0;
    if (argc < 2) {
        fprintf(stderr, "\nFalta iteraciones\n"); exit(-1);
    }
    n = atoi(argv[1]); if (n % 2) n+=1; if (n>20) n=20;
    for (i=0; i<n; i++) a[i] = i;
    #pragma omp parallel firstprivate(sumap, sumap2)
    {
        #pragma omp for schedule(static,1)
        for (i=0; i<n; i+=2) {
            sumap+= a[i]; sumap2+= a[i+1];
        }
        sumap += sumap2;
        #pragma omp atomic
        suma += sumap;
    }
    printf("Fueras de 'parallel' suma=%d sumap=%d\n", suma, sumap); return(0);
}
```

(d) Cambios a realizar en el código:

- Se elimina la cláusula `firstprivate(sumap, sumap2)` de la directiva `parallel` y se añade a la directiva `for` una cláusula de reducción con el operador “+” y una lista de dos variables `sumap` y `sumap2: reduction(+:sumap,sumap2)`.

- Ya no es necesario usar una directiva `atomic` para que cada thread añada la suma parcial que han calculado a la variable compartida `suma`. Se eliminaría, por tanto, el siguiente código:

```
    sumap += sumap2;
    #pragma omp atomic
    suma += sumap;
```

- Se debe añadir código tras la construcción `parallel` que sume los contenidos de todas las variables compartidas que aparecen en la lista de la cláusula `reduction`, en este caso, son dos. Se añadiría después de la construcción `parallel`: `suma = sumap+sumap2;`

El código resultante sería el siguiente:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n=20, a[n], suma=0, sumap=0, sumap2=0;
    if (argc < 2)  {
        fprintf(stderr, "\nFalta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n % 2) n+=1; if (n>20) n=20;
    for (i=0; i<n; i++) a[i] = i;
    #pragma omp parallel
    {
        #pragma omp for schedule(static,1) reduction(+:sumap,sumap2)
        for (i=0; i<n; i+=2) {
            sumap+= a[i];
            sumap2+= a[i+1];
        }
    }
    suma = sumap+sumap2;
    printf("Fuera de 'parallel' suma=%d sumap=%d\n", suma, sumap);
    return(0);
}
```





2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores: Exámenes y Controles

Examen de Prácticas 20/06/2012 resuelto

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita, Julio Ortega

Licencia Creative Commons



1 Enunciado Examen de Prácticas del 20/06/2012

Cuestión 1.(1 punto) Considere el programa de la siguiente figura:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv) {
    int i, n=20, tid, a[n], suma=0,sumalocal;
    if(argc < 2) {
        fprintf(stderr, "\nFalta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;

    for (i=0; i<n; i++) a[i] = i;

#pragma omp parallel private(sumalocal,tid)
{ sumalocal=0;
    tid=omp_get_thread_num();
    #pragma omp for schedule(static)
    for (i=0; i<n; i++)
    { sumalocal += a[i];
        printf(" thread %d suma de a[%d]=%d sumalocal=%d \n", tid,i,a[i],sumalocal);
    }
    suma += sumalocal;
    #pragma omp barrier
    #pragma omp master
    printf("thread master=%d imprime suma=%d\n", tid,suma);
}
```

- (a) ¿Permite calcular correctamente la suma de todos los elementos del array `a[]`? (Indique cómo solucionaría el problema en el caso de que lo hubiera).
- (b) ¿Qué se pretende al incluir `#pragma omp master` en este programa?
- (c) ¿Se puede prescindir de `#pragma omp barrier`? (Justifique su respuesta).
- (d) ¿Cuál es la utilidad de la cláusula `schedule(static)` en la directiva `#pragma omp for` `schedule(static)`?

- (e) Qué diferencias habría en lo que imprime en pantalla el programa si se sustituyera `master` por `single` en el programa? Razoné su respuesta. ¿Para qué sirve la función `omp_get_thread_num()`?

Cuestión 2. (1 punto) Considere el programa de la siguiente figura

```
#include <stdio.h>
#include <stdlib.h>
#ifndef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

main(int argc, char **argv) {
    int i, n = 7, chunk, a[n], suma=0;

    if(argc< 2) {
        fprintf(stderr,"\\nFalta chunk \\n");
        exit(-1);
    }
    chunk = atoi(argv[1]);

    for (i=0; i<n; i++) a[i] = i;

#pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(static,chunk)
    for (i=0; i<n; i++)
    { suma = suma + a[i];
        printf(" thread %d suma a[%d] suma=%d \\n",
               omp_get_thread_num(),i,suma);
    }

    printf("Fuera de 'parallel for' suma=%d\\n",suma);
}
```

- (a) ¿Permite calcular la suma de los componentes del vector `a []`? Justifique su respuesta.
- (b) ¿Qué imprime el código cada vez que se ejecuta la función `printf` del bucle?
- (c) ¿Qué imprime el código cuando se ejecuta la segunda función `printf`?
- (d) ¿Para qué sirve el parámetro `chunk` en la construcción `#pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(static,chunk)`?
- (e) ¿Para qué sirven las cláusulas `firstprivate(suma)` y `lastprivate(suma)`?
- (f) ¿Qué imprime el programa si se eliminan `firstprivate(suma)` y `lastprivate(suma)` y se incluye `reduction(+:suma)` en la construcción `#pragma omp parallel for`?

2 Solución Examen de Prácticas del 20/06/2012

Cuestión 1.(1 punto) Considere el programa de la siguiente figura:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv) {
    int i, n=20, tid, a[n], suma=0,sumalocal;
    if(argc < 2) {
        fprintf(stderr,"\\nFalta iteraciones\\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;

    for (i=0; i<n; i++) a[i] = i;

#pragma omp parallel private(sumalocal,tid)
{ sumalocal=0;
    tid=omp_get_thread_num();
#pragma omp for schedule(static)
    for (i=0; i<n; i++)
    { sumalocal += a[i];
        printf(" thread %d suma de a[%d]=%d sumalocal=%d \\n", tid,i,a[i],sumalocal);
    }
    suma += sumalocal;
#pragma omp barrier
#pragma omp master
    printf("thread master=%d imprime suma=%d\\n", tid,suma);
}
}
```

- (a) ¿Permite calcular correctamente la suma de todos los elementos del array `a[]`? (Indique cómo solucionaría el problema en el caso de que lo hubiera).
- (b) ¿Qué se pretende al incluir `#pragma omp master` en este programa?
- (c) ¿Se puede prescindir de `#pragma omp barrier`? (Justifique su respuesta).
- (d) ¿Cuál es la utilidad de la cláusula `schedule(static)` en la directiva `#pragma omp for schedule(static)`?
- (e) Qué diferencias habría en lo que imprime en pantalla el programa si se sustituyera `master` por `single` en el programa? Razona su respuesta. ¿Para qué sirve la función `omp_get_thread_num()`?

Solución

(a) No, porque los threads acceden a la variable compartida `suma` en paralelo y todos leen, modifican y escriben en la variable (R-M-W). Tendrían que acceder un thread detrás de otro, es decir, los threads tendrían que acceder secuencialmente a realizar la operación `suma+=sumalocal` (R-M-W) para evitar que más de un thread puedan leer el mismo valor de `suma`. Para resolver el problema se podría, por ejemplo, utilizar la directiva `atomic` o la directiva `critical` (esta última es menos eficiente):

atomic	critical
#pragma omp atomic suma += sumalocal;	#pragma omp atomic suma += sumalocal;

(b) Se pretende que el thread 0 (el master) ejecute la función `printf` que hay justo después de esta directiva (en el bloque estructurado de la directiva). Esta función imprime `tid`, que es el identificador del thread (0 en este caso) que ejecuta `printf` y el contenido de la variable compartida `suma`. Como se ha reflexionado previamente, `suma` puede que no contenga la suma de todos los componentes del vector debido al acceso sin exclusión mutua que realizan los threads previamente en el código.

(c) No. Con esta directiva los threads se esperan en el punto del código donde se encuentra, cuando todos han llegado a ese punto, continúan la ejecución. Es necesario mantener esta barrera para que el thread 0 imprima el contenido de la variable `suma` cuando todos los threads han acumulado en esta variable el resultado parcial de `suma` que han almacenado en su variable local `sumalocal`. Si se elimina, el thread 0 podría imprimir la suma parcial que él mismo ha calculado o la suma de los valores calculados por alguno de los threads incluido el mismo, no habría, por tanto, garantía de haber acumulado en `suma` todas las sumas parciales calculadas por los threads.

(d) Esta cláusula fija qué tipo de asignación de iteraciones del bucle (planificación) se va a realizar. En este caso fija que se realice una planificación por parte del compilador (estática). Como no se especifica el tamaño de los trozos (es decir, el número de iteraciones consecutivas del bucle) que se van a asignar a los threads en turno rotatorio, se tomará el que fije por defecto la implementación particular de OpenMP que se use.

(e) Como se ha comentado en (b) la función `printf` imprime `tid`, que contiene el identificador del thread que ejecuta el `printf`. Si se usa `single`, la función `printf` la ejecutará el thread que llegue en primer lugar a ese punto del código, podría ser el 0 o cualquier otro. Por tanto, el valor que se imprime como `tid` ahora, usando `single`, puede ser 0, 1,... `nthread-1`, donde `nthread` es el identificador del thread que ejecuta el código. La variable `tid` contiene el identificador del thread que ejecuta el código porque se inicializa con el valor que devuelve la función de la biblioteca OpenMP `omp_get_thread_num()`.



Cuestión 2.(1 punto) Considere el programa de la siguiente figura

```
#include <stdio.h>
#include <stdlib.h>
#ifndef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

main(int argc, char **argv) {
    int i, n = 7, chunk, a[n], suma=0;

    if(argc< 2) {
        fprintf(stderr, "\nFalta chunk \n");
        exit(-1);
    }
    chunk = atoi(argv[1]);

    for (i=0; i<n; i++) a[i] = i;

    #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(static,chunk)
    for (i=0; i<n; i++)
    { suma = suma + a[i];
        printf(" thread %d suma a[%d] suma=%d \n",
               omp_get_thread_num(),i,suma);
    }

    printf("Fuera de 'parallel for' suma=%d\n",suma);
}
```

- (a) ¿Permite calcular la suma de los componentes del vector `a []`? Justifique su respuesta.
- (b) ¿Qué imprime el código cada vez que se ejecuta la función `printf` del bucle?
- (c) ¿Qué imprime el código cuando se ejecuta la segunda función `printf`?
- (d) ¿Para qué sirve el parámetro `chunk` en la construcción `#pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(static,chunk)`?
- (e) ¿Para qué sirven las cláusulas `firstprivate(suma)` y `lastprivate(suma)`?
- (f) ¿Qué imprime el programa si se eliminan `firstprivate(suma)` y `lastprivate(suma)` y se incluye `reduction(+:suma)` en la construcción `#pragma omp parallel for`?

Solución

(a) No. La cláusula `firstprivate(suma)` fuerza a que haya una variable `suma` privada a cada thread y a que esta variable se inicialice al valor que tiene la variable compartida `suma` declarada en el thread master. Por tanto, el valor de la variable compartida `suma` declarada en el thread master se copia a todas las variables privadas `suma` de los threads que ejecutan la región paralela. En ningún momento en el código se suman las sumas parciales obtenidas por los threads en sus respectivas variables privadas `suma` con el fin de obtener la suma total.

(b) El `printf` que se ejecuta cada iteración del bucle imprime el identificador del thread que ejecuta la iteración del bucle (porque es el valor que devuelve `omp_get_thread_num()`), la iteración que se ejecuta (valor de `i`), y el valor de la variable privada del thread `suma` en esa iteración. Los threads imprimen en pantalla el valor de su variable privada `suma` cada vez que añade un nuevo componente del vector a dicha variable.

(c) Imprime el contenido de la variable compartida `suma` que se ha obtenido en la última iteración del bucle, independientemente de cual sea el thread que ha ejecutado esa iteración.

(d) Se utiliza `chunk` para fijar el número de iteraciones consecutivas del bucle que va a contener los trozos de código que se van a usar como unidades de asignación a los threads. Estos trozos se asignan por turno rotatorio. Si, por ejemplo, `chunk` fuese 1, entonces la unidad de asignación sería una iteración y se asignaría entonces al thread 0, las iteraciones 0, `nthreads`, `2nthreads`, ..., al thread 1 las iteraciones 1, `nthreads+1`, `2nthreads+1`, ..., y así sucesivamente. Si, por ejemplo, fuese 2, entonces se asignaría al thread `i` las iteraciones, $i, i+1, 2nthreads+2i, 2nthreads+2i+1, 4nthreads+2i, 4nthreads+2i+1, \dots$

(e) La cláusula `firstprivate(suma)` fuerzasustituye a que haya una variable `suma` privada a cada thread y a que esta variable se inicialice al valor que tiene la variable compartida `suma` declarada en el thread master. Por tanto, el valor de la variable compartida `suma` declarada en el thread master se copia a todas las variables privadas `suma` de los threads que ejecutan la región paralela.

La cláusula `lastprivate(suma)` fuerza a que haya una variable `suma` privada a cada thread (lo fuerza también `firstprivate`) y a que al valor que tiene la variable privada `suma` en la última iteración del bucle se copie a la variable compartida `suma` declarada en el thread master. En este caso se devuelve el valor de la variable `suma` del thread que ejecute la iteración $n-1=6$.

(e) La cláusula `reduction(+:suma)` fuerza a que haya una variable `suma` privada a cada thread inicializada a 0 y a que los threads, una vez ejecutadas las iteraciones que tienen asignados, sumen (por usar "+") en exclusión mutua el contenido de su variable privada `suma` a la variable compartida `suma` declarada en el thread master. Por tanto, si se sustien `firstprivate(suma)` y `lastprivate(suma)` por `reduction(+:suma)`, la suma se va a calcular correctamente y el programa imprimirá en el segundo `printf` la suma de todos los componentes del vector. El vector de 7 componentes se ha inicializado a 0, 1, 2, 3, 4, 5, 6 ($a[i]=i$); por tanto, se imprimirá 21.



2º curso / 2º cuatr.

 Grado en
 Ing. Informática

Arquitectura de Computadores. Ejercicios resueltos

Tema 3. Arquitecturas con paralelismo a nivel de thread (TLP)

Material elaborado por los profesores responsables de la asignatura:
 Mancia Anguita, Julio Ortega

1 Ejercicios

Ejercicio 1. En un multiprocesador SMP con 4 procesadores o nodos (N0-N3) basado en un bus, que implementa el protocolo MESI para mantener la coherencia, supongamos una dirección de memoria incluida en un bloque que no se encuentra en ninguna cache. Indique los estados de este bloque en las caches y las acciones que se producen en el sistema ante la siguiente secuencia de eventos para dicha dirección:

1. Lectura generada por el procesador 1
2. Lectura generada por el procesador 2
3. Escritura generada por el procesador 1
4. Escritura generada por el procesador 2
5. Escritura generada por el procesador 3

Solución

Datos del ejercicio

Se accede a una dirección de memoria cuyo bloque k no se encuentra en ninguna cache, luego debe estar actualizado en memoria principal y el estado en las caches se considera inválido.

Estado del bloque en las caches y acciones generadas ante los eventos que se refiere a dicho bloque

Hay 4 nodos con cache y procesador (N0-N3). Intervienen N1, N2 y N3. En la tabla se van a utilizar las siguientes siglas y acrónimos:

MP: Memoria Principal.

PtLec(k): paquete de petición de lectura del bloque k.

PtLecEx(k): paquete de petición de lectura del bloque k y de petición de acceso exclusivo al bloque k.

RpBloque(k): paquete de respuesta con el bloque k.

Se va a suponer que no existe en el sistema paquete de petición de acceso exclusivo a un bloque sin lectura (no existe PtEx).

ESTADO INICIAL	EVENTO	ACCIÓN	ESTADO SIGUIENTE
N1) Inválido			N1) Exclusivo
N2) Inválido	P1 lee k	1.- N1 (el controlador de cache de N1) genera y deposita en el bus una petición de lectura del bloque k (PtLec(k)) porque no lo tiene en su caché válido	N2) Inválido
N3) Inválido		2.-MP (el controlador de memoria de MP), al observar PtLec(k) en el bus, genera la respuesta con el bloque (RpBloque(k)).	N3) Inválido

		3.- N1 (el controlador de cache de N1) recoge del bus la respuesta depositada por la memoria principal ($RpBloque(k)$), el bloque entra en la cache de N1 en estado exclusivo ya que no hay copia en otra cache del bloque (es decir, la salida de la OR cableada con entradas procedentes de todas las caches es 0).
N1) Exclusivo N2) Inválido N3) Inválido	P2 lee k	<p>1.- N2 genera y deposita en el bus una $PtLec(k)$ porque no tiene k en su caché en estado válido</p> <p>2.- N1 observa $PtLec(k)$ en el bus y, como tiene el bloque en estado exclusivo, lo pasa a compartido (la copia que tiene ya no es la única válida en caches). MP, al observar $PtLec(k)$ en el bus, genera la respuesta con el bloque ($RpBloque(k)$).</p> <p>3.- N2 recoge $RpBloque(k)$ que ha depositado la memoria, el bloque entra en estado compartido en la cache de N2 (la salida de la OR cableada será 1).</p>
N1) Compartido N2) Compartido N3) Inválido	P1 escribe en k	<p>1.- N1 genera petición de lectura con acceso exclusivo del bloque k ($PtLecEx(k)$) (suponemos que no hay petición de acceso exclusivo sin lectura, no hay $PtEx$). N1 modifica la copia de k que tiene en su cache y lo pasa a estado modificado.</p> <p>2.- N2 observa $PtLecEx(k)$ y, como la petición incluye acceso exclusivo (Ex) a un bloque que tiene en su cache en estado compartido, pasa su copia a estado inválido. MP genera $RpBloque(k)$ porque observa en el bus una petición de k con lectura (Lec), pero esta respuesta no se va a recoger del bus. N1 no recoge $RpBloque(k)$ depositada por la memoria porque tiene el bloque válido.</p>
N1) Modificado N2) Inválido N3) Inválido	P2 escribe en k	<p>1.- N2 genera petición de lectura con acceso exclusivo de k ($PtLecEx(k)$)</p> <p>2.- N1 observa $PtLecEx(k)$ y, como tiene el bloque en estado modificado (es la única copia válida en todo el sistema), inhibe la respuesta de MP y genera respuesta con el bloque $RpBloque(k)$, y además, como el paquete pide acceso exclusivo a k (Ex), invalida su copia del bloque k.</p> <p>3.- N2 recoge $RpBloque(k)$, introduce k en su cache, lo modifica y lo pone en estado modificado</p>
N1) Inválido N2) Modificado N3) Inválido	P3 escribe en k	<p>1.- N3 genera petición de lectura con acceso exclusivo de k ($PtLecEx(k)$)</p> <p>2.- N2 observa $PtLecEx(k)$ y, como tiene el bloque en estado modificado, inhibe la respuesta de MP y genera respuesta con el bloque $RpBloque(k)$, y además, como el paquete pide acceso exclusivo a k (Ex), invalida su copia de k.</p> <p>3.- N3 recoge $RpBloque(k)$, introduce el k en su cache, lo modifica y lo pone en estado modificado</p>

Ejercicio 2. Para un multiprocesador de memoria distribuida con 8 nodos se quiere implementar un protocolo para mantenimiento de coherencia basado en directorios. Suponiendo que se necesitan un bit de

estado para un bloque en el directorio de memoria principal y que el tamaño de una línea de cache es de 64 bytes, calcular el porcentaje del tamaño de memoria principal que supone el tamaño del directorio de vector de bits completo.

Solución

Datos del ejercicio:

- Tamaño Línea de Cache (bloque de memoria): 64 bytes = TLC
- 1 bit de estado por bloque en el directorio

Directorio de vector de bits completo

$$\text{Tamaño_por_nodo} = \text{Nº_de_bloques_memoria_nodo} \times \text{Espacio_por_bloque}$$

TMN: Tamaño Memoria principal por Nodo

Teniendo en cuenta que hay 8 nodos y un bit de estado, se necesitan 9 bits por entrada del directorio.

$$\text{Tamaño_directorio_por_nodo} = \frac{\text{TMN}}{\text{TLC}} \times (8b + 1b)$$

$$\% \text{ de TMN} = \frac{\frac{\text{TMN}}{\text{TLC}} \times 9b}{\text{TMN}} \times 100 = \frac{9b}{\text{TLC}} \times 100 = \frac{9b}{64 \times 8b} \times 100 \approx 1,76\%$$

Ejercicio 3. Suponga que en un CC-NUMA de red estática de 4 nodos (N0-N3) se implementa un protocolo MSI basado en directorios sin difusión con dos estados en el directorio (válido e inválido). Cada nodo tiene 8 GBytes de memoria y una línea de cache supone 64 Bytes. Considere que el directorio utiliza vector de bits completo. **(a)** Calcule el tamaño del directorio en bytes. **(b)** Indique cual sería el contenido del directorio, las transiciones de estados (en cache y en el directorio) y la secuencia de paquetes generados por el protocolo de coherencia en los siguientes accesos sobre una dirección D que se encuentra en la memoria del nodo 3 (initialmente D no está en ninguna cache):

1. Lectura generada por el procesador del nodo 1
2. Escritura generada por el procesador del nodo 1
3. Lectura generada por el procesador del nodo 2
4. Lectura generada por el procesador del nodo 3
5. Escritura generada por el procesador del nodo 0

Solución

Datos del ejercicio

- Tamaño Memoria principal por Nodo: 8GB = TMN
- Tamaño Línea de Cache (bloque de memoria): 64 B = TLC
- 1 bits de estado por bloque en el directorio ya que hay que codificar dos estados (válido, inválido).
- Se accede a una dirección de la memoria del nodo 3 cuyo bloque no se encuentra en ninguna cache, luego debe estar actualizado en memoria principal.

(a)

$$\text{Tamaño_por_nodo} = \frac{\text{TMN}}{\text{TLC}} \times (1b + 4b) = \frac{2^{33}B}{2^6B} \times 5b = 2^{27} \times 5b = 2^{20} \times \frac{2^7 \times 5b}{2^3b/B} = (2^{20} \times 2^4 \times 5)B = 80MB$$

$$\text{Tamaño_por_nodo} = \frac{\text{TMN}}{\text{TLC}} \times (2b + 4b) = \frac{2^{33}B}{2^6B} \times 6b = 2^{27} \times 6b = 2^{20} \times \frac{2^7 \times 6b}{2^3b/B} = (2^{20} \times 2^4 \times 6)B = 96MB$$

$$\text{Tamaño}_{\text{directorio}} = \text{Tamaño}_{\text{por_nodo}} \times 4 = 384MB$$

(b)

Intervienen los nodos N0, N1, N2 y N3. V es Válido e I inválido. BD denota el bloque de la dirección D.

ESTADO INICIAL	EVENTO	ACCIÓN	ESTADO SIGUIENTE								
N0) Inválido N1) Inválido N2) Inválido N3) Inválido D) Válido <table border="1"> <tr><td>V</td><td></td><td></td><td></td></tr> </table>	V				P1 lee D	1. N1 envía petición de lectura del bloque BD (PtLec(BD)) a N3 2. N3 recibe PtLec(BD). Como tiene el bloque BD en estado Válido, (1) envía paquete de respuesta con el bloque a N1 (RpBloque(BD)) y (2) pone el bit de N1 en el directorio a 1. 3. N1 recibe RpBloque(BD) y pone el bloque en cache en estado Compartido	N0) Inválido N1) Compartido N2) Inválido N3) Inválido D) Válido <table border="1"> <tr><td>V</td><td>1</td><td></td><td></td></tr> </table>	V	1		
V											
V	1										
N0) Inválido N1) Compartido N2) Inválido N3) Inválido D) Válido <table border="1"> <tr><td>V</td><td>1</td><td></td><td></td></tr> </table>	V	1			P1 escribe en D	1. N1 envía petición de acceso exclusivo para BD (PtEx(BD)) a N3. 2. N3, cuando recibe PtEx(BD), (1) pasa BD a estado Inválido y (2) envía paquete de respuesta a N1 confirmando invalidación (RpInv(BD)). 3. N1, recibida la respuesta, modifica el bloque y lo pasa a estado Modificado.	N0) Inválido N1) Modificado N2) Inválido N3) Inválido D) Inválido <table border="1"> <tr><td>I</td><td>1</td><td></td><td></td></tr> </table>	I	1		
V	1										
I	1										
N0) Inválido N1) Modificado N2) Inválido N3) Inválido D) Inválido <table border="1"> <tr><td>I</td><td>1</td><td></td><td></td></tr> </table>	I	1			P2 lee D	1. N2 envía PtLec(BD) a N3 porque no tiene BD. 2. N3, como tiene BD en estado Inválido: (1) reenvía (RvLec(BD)) la petición al nodo N1 (que según el directorio tiene copia válida del bloque), y (2) pone en la entrada del bloque en el directorio estado pendiente de Válido y activa el bit de N2. 3. N1 recibe RvLec(BD) y: (1) envía a N3 un paquete de respuesta con el bloque (RpBloque(BD)), y (2) pasa BD en su cache a Compartido. 4. N3 recibe la respuesta de N1 (RpBloque(BD)) y: (1) responde con el bloque a N2 (RpBloque(BD)) y (2) escribe BD en MP y pone el estado del bloque en el directorio a Válido 5. N2, cuando recibe RpBloque(BD), introduce el bloque en su cache en estado Compartido	N0) Inválido N1) Compartido N2) Compartido N3) Inválido D) Válido <table border="1"> <tr><td>V</td><td>1</td><td>1</td><td></td></tr> </table>	V	1	1	
I	1										
V	1	1									
N0) Inválido N1) Compartido N2) Compartido N3) Inválido D) Válido <table border="1"> <tr><td>V</td><td>1</td><td>1</td><td></td></tr> </table>	V	1	1		P3 lee D	N3 lee BD de su propia memoria, lo introduce en su cache en estado Compartido y activa el bit de copia en la entrada de BD en el directorio de N3	N0) Inválido N1) Compartido N2) Compartido N3) Compartido D) Válido <table border="1"> <tr><td>V</td><td>1</td><td>1</td><td>1</td></tr> </table>	V	1	1	1
V	1	1									
V	1	1	1								
N0) Inválido N1) Compartido N2) Compartido N3) Compartido D) Válido <table border="1"> <tr><td>V</td><td>1</td><td>1</td><td>1</td></tr> </table>	V	1	1	1	P0 escribe en D	1. N0 envía a N3 petición de lectura de BD con acceso exclusivo PtLecEx(BD) 2. N3 recibe PtLecEx(BD) y, como tiene el bloque en estado Válido: (1) pone en el directorio el estado de BD en pendiente de Inválido, (2) envía los paquetes RvInv(BD) a los nodos que, según el directorio, tienen copia del bloque, (3) desactiva los bits de presencia 1, 2 y 3, activa bit de presencia 0 (de N0). 3. N1, N2 y N3, cuando reciben RvInv(BD): (1) invalidan su copia de BD y (2) responden a N3 confirmando la invalidación	N0) Modificado N1) Inválido N2) Inválido N3) Inválido D) Inválido <table border="1"> <tr><td>I</td><td>1</td><td></td><td></td></tr> </table>	I	1		
V	1	1	1								
I	1										



	<p>RplInv(BD).</p> <p>4.N3, cuando recibe todas las RplInv(BD): (1) envía respuesta con el bloque a N0 confirmando invalidación RpBloqueInv (espera a todas las invalidaciones para garantizar un orden en los accesos a BD y así garantizar coherencia) y (2) pone el estado de BD en el directorio a Inválido</p> <p>5.N0 recibe RpBloqueInv de N3, escribe BD en su cache, modifica BD y lo pone en estado Modificado</p>	
--	--	--

Ejercicio 4. Supongamos que se va a ejecutar en paralelo el siguiente código (initialmente x e y son 0):

P1	P2
x=1;	y=1;
x=2;	y=2;
print y ;	print x ;

Qué resultados se pueden imprimir si (considere que el compilador no altera el código):

- (a) Se ejecutan P1 y P2 en un multiprocesador con consistencia secuencial.
- (b) Se ejecutan en un multiprocesador basado en un bus que garantiza todos los órdenes excepto el orden W→R. Esto es debido a que los procesadores tienen buffer de escritura, permitiendo el procesador que las lecturas en el código que ejecuta adelantan a las escrituras que tiene su buffer. Obsérvese que hay varios posibles resultados.

Solución

El compilador no altera ningún orden garantizado ya que se supone, según el enunciado, que no altera el código.

(a) Si P1 es el primero que imprime puede imprimir 0, 1 o 2, pero P2 podrá imprimir sólo 2. Esto es así porque se mantiene orden secuencial (el hardware parece ejecutar los accesos a memoria del código que ejecuta un procesador en el orden en el que están en dicho código) y, por tanto, cuando P1 lee "y" (instrucción 1.3 en el código, esta instrucción lee "y" para imprimir su contenido), ha asignado ya a "x" un 2 (punto 1.2 en el código) ya que esta asignación está antes en el código que la lectura de "y".

P1	P2
(1.1) x=1;	(2.1) y=1;
(1.2) x=2;	(2.2) y=2;
(1.3) print y ;	(2.3) print x ;

Si P2 es el primero que imprime podrá imprimir 0, 1 o 2, pero entonces P1 sólo puede imprimir 2. Esto es así porque se mantiene orden secuencial y, por tanto, cuando P2 lee "x" (punto 2.3 en el código), ha asignado ya a "y" un 2 (punto 2.2 en el código) ya que esta asignación está antes en el código que la lectura de "x" y se mantiene orden secuencial en los accesos a memoria, es decir, los accesos parecen completarse en el orden en el que se encuentran en el código.

Se puede obtener como resultado de la ejecución las combinaciones que hay en cada una de las líneas:

P1 P2
0 2 (en este caso P1 imprime 0 y P2 imprime 2)
1 2
2 2
2 0
2 1

(b) Si no se mantiene el orden W→R además de los resultados anteriores, los dos procesos pueden imprimir:

P1 P2
1 1 (en este caso P1 imprime 1 y P2 imprime 1)
0 1

1	0
0	0

Se pueden imprimir también las combinaciones anteriores porque no se asegura que cuando un procesador ejecute la lectura de la variable que imprime `print` (puntos 1.3 y 2.3 en los códigos) haya ejecutado las instrucciones anteriores que escriben en `x` (P1 en los puntos 1.1 y 1.2 del código) o en `y` (P2 en los puntos 2.1 y 2.2). Esto es así porque no se garantiza el orden W->R y, por tanto, una lectura puede adelantar a escrituras que estén antes en el código secuencial. P1 puede leer `y` (1.3) antes de escribir en `x` 2 (1.2) o incluso antes de escribir en `x` 1 (1.1). Igualmente P2 puede leer `x` (2.3) antes de escribir en `y` 2 (2.2) o antes de escribir en `y` 1 (2.1).

Teniendo esto en cuenta P1 puede imprimir 1 o 2 o 0, y P2 1 o 2 o 0. Todas las combinaciones son posibles.

Ejercicio 5. Supongamos que se va a ejecutar en paralelo el siguiente código (initialmente `x` e `y` son 0):

P1	P2
<code>x=30;</code>	<code>while (flag==0) {};</code>
<code>y=40;</code>	<code>r1=x;</code>
<code>flag=1;</code>	<code>r2=y;</code>

Qué datos puede obtener P2 en `r1` y `r2` si (considere que el compilador no altera el código):

- (a) Se ejecutan P1 y P2 en un multiprocesador con consistencia secuencial.
- (b) Se ejecutan en un multiprocesador con consistencia de liberación. Razone su respuesta.

Solución

El compilador no altera ningún orden garantizado ya que se supone, según el enunciado, que no altera el código.

- (a) Si se implementa consistencia secuencial en `r1` se almacena 30 y en `r2` 40.

RAZONAMIENTO:

Esto es así porque se garantizan los órdenes de acceso a memoria W->W y R->R que aparecen en el código que ejecuta un proceso:

1) En P1, al garantizarse el orden W->W, la escritura de 1 en `flag` no se realiza hasta que no se han realizado las escrituras en `x` e `y` que preceden a la escritura de `flag` en el código de P1.

2) Hasta que P2 no encuentra en `flag` un 1 no almacena en `r1` el contenido de `x` ni en `r2` el contenido de `y`. Al mantenerse el orden R->R, en P2 no se adelanta la lectura de `x` ni la lectura de `y` a la lectura de `flag` aunque se permita la lectura especulativa (ejecutar instrucciones cuya ejecución depende de una condición de salto antes de verificar que se cumple la condición).

Por tanto, como se garantiza W->W y además se garantiza R->R, si P2 ve en `flag` un 1 y, por tanto, sale del bucle, va a ver al leer en `x` 30 y en `y` 40.

- (b) Si se implementa consistencia de liberación se pueden dar los siguientes resultados

`r1 r2`

---- ----

0 0

0 40

30 0

30 40

RAZONAMIENTO:

1) Al no garantizarse el orden entre accesos de escritura (W->W), P1 podría escribir en `flag` un 1 antes de escribir 30 en `x` o 40 en `y` (obsérvese que la escritura `y=40` podría también adelantar a `x=30`). Por lo que P2

podría leer en flag un 1 y, por tanto, salir del bucle, antes de que en x o en y pudiera ver los valores que escribe P1 en estas variables (vería entonces 0).

2) Al no garantizarse el orden entre accesos de lectura (R->R), si se permite ejecutar lecturas cuya ejecución depende de una condición de salto antes de verificar que se cumple la condición (ejecución especulativa), en P2 se podría, en la última iteración del bucle, adelantar la lectura de x o la lectura de y o ambas a la de flag. En este caso P2 podría entonces leer de forma *especulativa los valores de x e y que tienen en su cache antes de que P1 los modifique y modifique flag*. En estas circunstancias podría obtenerse en r1 o en r2 o en ambos un 0. Otras combinaciones serían también posibles.

Ejercicio 6. Se quiere implementar un cerrojo simple en un multiprocesador SMP basado en procesadores de la línea x86 de Intel, en particular, procesadores Intel Core. **(a)** Teniendo en cuenta el modelo de consistencia de memoria que ofrece el hardware de este multiprocesador ¿podríamos implementar la función de liberación del cerrojo simple usando “`mov k, 0`”, siendo k la variable cerrojo? Razona tu respuesta. **(b)** ¿Cómo se debería implementar la función de liberación de un cerrojo simple si se usan procesadores Itanium? Razona tu respuesta.

Solución

- (a)** La función de liberación se utiliza después de acceder a las variables compartidas para permitir que, después de un acceso por parte de un flujo de control a estas variables, otros flujos de control puedan acceder. Los procesadores de la línea x86 sólo relajan W->R. Se podría implementar puesto que en el multiprocesador no se permite que una escritura adelante a una lectura o escritura anterior; por tanto, la liberación del cerrojo no va a realizarse antes de haber terminado los accesos a las variables compartidas.
- (a)** Los Itanium implementan un modelo de ordenación que relaja todos los órdenes. No obstante, proporcionan instrucciones para garantizar órdenes apropiados cuando resulta necesario; en particular, proporcionan una escritura con liberación que garantiza que no se escribe hasta que no hayan terminado los accesos a memoria que preceden a la instrucción de liberación. Para implementar la función de liberación de un cerrojo simple bastaría usar una instrucción de almacenamiento en memoria de liberación (`st.rel`)

Ejercicio 7. Se ha ejecutado el siguiente código en un multiprocesador con un modelo de consistencia que no garantiza ni W->R ni W->W (garantiza el resto de órdenes):

```
(1) sump = 0;
(2) for (i=ithread ; i<8 ; i=i+nthread) {
(3)     sump = sump + a[i];
}
(4) while (Fetch_&_Or(k,1)==1) {};
(5) sum = sum + sump;
(6) k=0;
```

Conteste a las siguientes preguntas (considere que el compilador no altera el código):

- (a)** Indique qué se puede obtener en sum si se suma la lista `a={1,2,3,4,5,6,7,8}`. k y sum son variables compartidas que están inicialmente a 0 (el resto de variables son privadas), `nthread = 3` y `ithread` es el identificador del thread en el grupo (0,1,2). Si hay varios posibles resultados, se tienen que dar todos ellos. Justifique su respuesta.
- (b)** ¿Qué resultados se pueden obtener si lo único que no garantiza el modelo de consistencia es el orden W->R? Justifique su respuesta.

Solución

No hay problemas con `Fetch_&_Or(k, 1)` porque es una operación atómica que contiene R (también tiene W) y ni las R ni las W pueden adelantar a R anteriores en el orden del programa. Pero, al no garantizarse el orden W->W, la liberación del cerrojo, es decir la asignación de 0 a k puede adelantar los accesos a la variable compartida anteriores, en particular, puede adelantar a la escritura de `sum` o a la escritura y la lectura. Si esto ocurre más de un flujo de control puede leer el mismo valor en `sum`, acumular a ese valor su variable local `sum_p` y almacenar el resultado. En la variable `sum` acaba acumulado entonces, tras la escritura de los threads que han leído lo mismo de `sum`, sólo el contenido de `sum_p` de uno de esos threads, en particular, del último que ha escrito.

Para obtener los posibles resultados, primero hay que obtener lo que calcula cada thread en `sum_p`. Teniendo en cuenta que el bucle `for` asigna iteraciones consecutivas a distintos threads (turno rotatorio) el thread 0 suma $1+4+7=12$, el 1 suma $2+5+8=15$ y el 2 suma $3+6=9$.

	resultado	comentario
Si leen distinto valor	$12+15+9=36$	Si no hay problemas debido a que la escritura de liberación adelante a los accesos a <code>sum</code> anteriores
Si todos leen 0 en sum	12 15 9	<p>Si los tres flujos de control llegan a leer el valor 0 inicial de <code>sum</code> y el último que escribe en <code>sum</code> tras actualizar su valor es el thread 0</p> <p>Si los tres flujos de control llegan a leer el valor 0 inicial de <code>sum</code> y el último que escribe en <code>sum</code> es el thread 1</p> <p>Si los tres flujos de control llegan a leer el valor 0 inicial de <code>sum</code> y el último que escribe en <code>sum</code> es el thread 2</p>
Si dos leen el mismo valor en sum, y el otro un valor distinto	$12+15=27$ ó $12+9=21$	<ul style="list-style-type: none"> - Si el flujo de control 0 ha logrado acumular primero sin problemas y el 1 y el 2 leen el valor que tiene <code>sum</code> tras la acumulación de 0. Si esto ocurre entonces 1 y 2 acceden al mismo valor, 12, le acumulan cada uno lo que han calculado y escriben el resultado de la acumulación en <code>sum</code>. El resultado será 27 si el último que escribe es 1 y 21 si el último que escribe es 2. - Si los flujos 1 y 2 leen 0 los dos y acumulan su resultado parcial. Si de los dos escribe el último 1, entonces 0 sumará 12 a 15 obteniéndose 27. Si escribe el último 2, entonces 0 sumará 12 a 9, obteniéndose 21.
	$15+12=27$ ó $15+9=24$	<ul style="list-style-type: none"> - Si el flujo de control 1 ha logrado acumular primero sin problemas y el 0 y el 2 acceden al valor que tiene <code>sum</code> justo tras la acumulación de 1. Si esto ocurre entonces 0 y 2 acceden al mismo valor, 15, le acumulan cada uno lo que han calculado y escriben el resultado de la acumulación en <code>sum</code>. El resultado será 27 si el último que escribe es 0 y 24 si el último que escribe es 2. - Si los flujos 0 y 2 leen 0 los dos y acumulan su resultado parcial. Si de los dos escribe el último 0, entonces 1 sumará 15 a 12 obteniéndose 27. Si escribe el último 2, entonces 1 sumará 15 a 9, obteniéndose 24.
	$9+12=21$ ó $9+15=24$	<ul style="list-style-type: none"> - Si el flujo de control 2 ha logrado acumular primero sin problemas y el 0 y el 1 acceden al valor que tiene <code>sum</code> justo tras la acumulación de 2. Si esto ocurre entonces 0 y 2 acceden al mismo valor, 9, le acumulan cada uno lo que han calculado y escriben el resultado de la acumulación en <code>sum</code>. El resultado será 21 si el último que escribe es 0 y 24 si el último que escribe es 1. - Si los flujos 0 y 1 leen 0 los dos y acumulan su resultado parcial. Si de los dos escribe el último 0, entonces 2 sumará 9 a 12 obteniéndose 21. Si escribe el



último 1, entonces 2 sumará 9 a 15, obteniéndose 24.

- (b) En este caso la liberación del cerrojo no puede adelantar nunca a los accesos a la variable compartida `sum` porque la liberación es una escritura y no se admiten que las escrituras adelanten a accesos anteriores en el código. Por tanto, se hace el acceso a `en` en exclusión mutua por parte de los tres flujos. El único resultado posible sería la suma de todos los componentes de la lista porque se acumula correctamente en `sum` la suma parcial calculada en `sump` por todos los flujos de control: $12+15+9=36$.

Ejercicio 8. ¿Qué ocurre si en el segundo código para implementar barreras visto en clase eliminamos la variable local, `cont_local`, sustituyéndola en los puntos del código donde aparece por el contador compartido asociado a la barrera `bar[id].cont`?

Solución

Pueden surgir problemas pudiendo incluso no funcionar bien como barrera.

Si se usa el contador global en el `if` que comprueba si contador es ya igual al número de procesos que se sincronizan con la barrera, un proceso puede encontrar, cuando llegue al `if`, que el contador es igual al número de procesos sin ser el último proceso que ha incrementado el contador. Esto puede provocar el siguiente problema:

Además del proceso que ha hecho el contador igual al número de procesos, otros que lo han incrementado antes pueden encontrar la condición del `if` verdadera y escribir, por tanto, en contador global y en la bandera global. Todas estas escrituras provocan accesos a través de la red para transferir el bloque de memoria que contiene la información sobre la barrera cuando una trasferencia por cada variable a modificar sería suficiente. Estas transferencias adicionales simplemente devuelven prestaciones, nada más. El problema aparece si la barrera se vuelve a reutilizar por los mismos procesos y el SO suspende a uno de los procesos que encuentran contador igual a `num_procesos` antes de escribir un 0 en el contador global. Puede ocurrir que, mientras este proceso está bloqueado, el resto de procesos salgan de la barrera y vuelvan a reutilizarla. Conforme los procesos van ejecutando de nuevo la barrera, incrementan el contador global y se quedan esperando a que éste llegue a ser igual a `num_procesos`. Pero nunca llegará a alcanzar este valor porque cuando el proceso suspendido vuelve a ejecutarse pondrá a 0 el contador global perdiéndose la cuenta del número de procesos que están ya esperando en la barrera.

Ejercicio 9. Suponiendo que la arquitectura dispone de instrucciones Fetch&Add, simplifique el segundo código para barreras visto en clase.

Solución

En la siguiente figura se destaca en rojo el cambio realizado. Se decrementa uno a `num_procesos` en el `if` porque Fetch&Add devuelve el valor antes de la modificación, no después de la modificación. Por tanto, cuando llega el último proceso a la barrera se devuelve `num_procesos-1`.

Barrera sense-reversing

```
Barrera(id, num_procesos)
{
    bandera_local= !(bandera_local) //se complementa la bandera local
    cont_local = Fecht&Add (bar[id].cont,1); //cont_local es una variable privada
    if (cont_local ==num_procesos-1) {
        bar[id].cont=0; //se hace 0 el contador asociado a la barrera
        bar[id].bandera= bandera_local; //para liberar los procesos en espera
    }
    else while (bar[id].bandera!= bandera_local) {}; //espera ocupada
}
```



Ejercicio 10. Se quiere parallelizar el siguiente ciclo de forma que la asignación de iteraciones a los procesadores disponibles se realice en tiempo de ejecución (dinámicamente):

```
For (i=0; i<100; i++) {
    Código que usa i
}
```

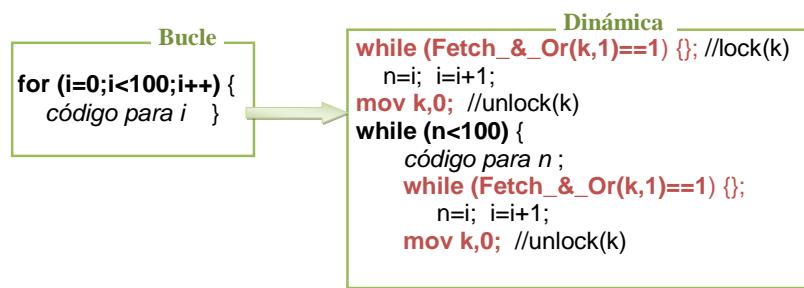
Nota: Considerar que las iteraciones del ciclo son independientes, que el único orden no garantizado por el sistema de memoria es W->R, que las primitivas atómicas garantizan que sus accesos a memoria se realizan antes que los accesos posteriores y que el compilador no altera el código.

- (a) Parallelizar el ciclo para su ejecución en un multiprocesador que implementa la primitiva Fetch&Or para garantizar exclusión mutua.
- (b) Parallelizar el anterior ciclo en un multiprocesador que además tiene la primitiva Fetch&Add .

Solución

Se debe tener en cuenta que el único orden que no garantiza el hardware es el orden W->R.

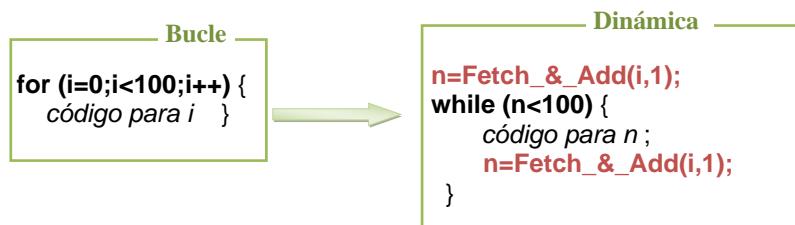
- (a) Parallelizar el ciclo para su ejecución en un multiprocesador que implementa la primitiva Fetch&Or para garantizar exclusión mutua.



Nota: la variable i estaría inicializada a 0 (por ejemplo, se puede iniciar cuando se declara)

Se supone que el compilador no cambia de sitio "mov k, 0".

- (b) Parallelizar el anterior ciclo en un multiprocesador que además tiene la primitiva Fetch&Add .



Nota: la variable i estaría inicializada a 0 (por ejemplo, se puede iniciar cuando se declara)



Ejercicio 11. Un programador está usando el siguiente código para barreras (bar es un vector compartido, k es una variable compartida, el resto son variables locales, Fetch_&_Or(k,1) realiza sus accesos a memoria antes de que puedan realizarse los accesos posteriores):

```
Barrera(id, num_procesos)
{
    band_local= !(band_local)
```

```

while (fetch_&_or(k,1)==1) {};
    cont_local = ++bar[id].cont;
k=0;
if (cont_local == num_procesos) {
    bar[id].cont=0;
    bar[id].band=band_local;
}
else while (bar[id].band != band_local) {};
}

```

Conteste a las siguientes cuestiones (considere que el compilador no altera el código):

(a) ¿Funciona bien este código como barrera en un multiprocesador en el que lo único que no garantiza su modelo de consistencia es el orden W->R? Razone por qué.

(b) Funciona bien este código como barrera en un multiprocesador con modelo de consistencia de memoria de ordenación débil? Razone por qué.

Solución

```

Barrera(id, num_procesos)
{
    band_local= !(band_local)
    while (fetch_&_or(k,1)==1) {};
        cont_local = ++bar[id].cont;
    k=0;
    if (cont_local == num_procesos) {
        bar[id].cont=0;
        bar[id].band=band_local;
    }
    else while (bar[id].band != band_local) {};
}

```

**(R,W) atómica
R seguida de W (RAW)
W**

Habrá problemas si dos o más threads pueden estar al mismo tiempo ejecutando accesos a variables compartidas de la sección crítica; es decir, en este caso, accediendo a la variable compartida `bar[id].cont`. Esto podría ocurrir si:

1. La apertura del cerrojo o escritura en `k` de 0 adelanta a los accesos anteriores a la variable compartida `bar[id].cont`, porque un proceso podría encontrar el cerrojo abierto estando otro proceso aún en la sección crítica, o
2. El acceso a la variable compartida (lectura) adelanta a la operación atómica de escritura y lectura `fetch_&_or`, porque el proceso que la adelanta accedería a la variable compartida pudiendo estar otro proceso accediendo a ella. El enunciado del ejercicio dice que esto no puede ocurrir.

(a) Sí, funciona bien como barrera. La escritura en `k` de 0 no puede adelantar a los accesos anteriores a la variable compartida `bar[id].cont`, puesto que la arquitectura no admite que una escritura pueda adelantar a lecturas o escrituras anteriores en el orden del programa. Por tanto, un thread asigna un 0 a `k` cuando ya ha escrito en la variable compartida.

(b) No, no funciona bien como barrera. La escritura en `k` de 0 puede adelantar a los accesos anteriores a la variable compartida `bar[id].cont`, puesto que la arquitectura permite que una escritura pueda adelantar a lecturas o escrituras anteriores en el orden del programa. Por tanto, un thread puede asignar un 0 a `k` cuando aún no ha escrito en la variable compartida y, por tanto, otro thread podría entrar en la sección crítica por encontrarse el cerrojo abierto y podría leer el mismo valor de `bar[id].cont` que el thread que le abrió el cerrojo.

Ejercicio 12. Se quiere implementar un programa que calcule en paralelo la siguiente expresión en un multiprocesador en el que sólo se relaja el orden W->R y en el que sólo se dispone de primitiva de sincronización test_&_set:

$$d = \frac{1}{N} \sum_{i=1}^N x_i^2 - \bar{x}^2, \text{ donde } \bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

Un programador ha implementado el código de abajo. Tenga en cuenta lo siguiente: el código lo ejecutan `nthread` threads en paralelo; `i` es una variable local que nota el identificador del thread; `i`, `med1` y `varil` son variables locales; `i`, `med`, `vari`, el vector `x` y `N` son variables compartidas; inicialmente `med`, `vari`, `med1` y `varil` son 0.

```
(1) for (i=ithread;i<N;i=i+nthread) {
(2)   med1=med1+x[i];
(3)   varil=varil+x[i]*x[i];
(4) }
(5) med = med + med1/N; vari = vari + varil/N;
(6) vari= vari - med*med;
(7) if (ithread==0) printf("varianza = %f", vari); //imprime en pantalla
```

Conteste a las siguientes cuestiones (considere que el compilador no altera el código):

- (a) Se ha ejecutado este código usando varias hebras y se ha visto que, aunque `N` y el vector `x` no varían, no siempre se imprime lo mismo. ¿Por qué ocurre esto?
- (b) Añada lo mínimo necesario para solucionar el problema teniendo en cuenta que sólo se dispone para implementar sincronización de `test_&_set` (tampoco se dispone de primitivas software de sincronización). Indique qué variables son ahora compartidas y cuáles locales.
- (c) Escriba el programa suponiendo que el multiprocesador además tiene primitivas de sincronización `fetch_&_add` (se puntuará según prestaciones). Indique qué variables son compartidas y cuáles locales.
- (d) Escriba el programa ahora suponiendo que el multiprocesador sólo tiene primitivas de sincronización `compare_&_swap` (se puntuará según prestaciones). Indique qué variables son compartidas y cuáles locales.

NOTA: En todos los apartados puede añadir o quitar variables si lo estima conveniente.

Solución

- (a) Hay dos tipos de errores en el código:

1. No se accede en exclusión mutua a las variables compartidas `med` y `vari` por parte de los diferentes threads (línea de código 5). Esto permite que puedan intentar varios threads a la vez acumular el resultado parcial que han calculado (carrera). Por ejemplo, varias pueden cargar el mismo valor en `med`, acumular a ese valor su variable local `med1` y almacenar el resultado en `med`. El resultado de `med` acumulado en `med` será entonces el cargado por todas ellas más el contenido de `med1` de sólo una de ellas, de la última que ha almacenado. Por lo que no se acumularía lo calculado por todas ellas.
2. Las operaciones de la línea (6) leen y modifican la variable compartida `vari`. Tal y como está el código, esa operación la realizan todos los threads lo que puede llevar a restar varias veces a `vari` el resultado de elevar al cuadrado `med`. Para evitar este problema se puede escribir en `varil` (e imprimir esta variable de 0 en el `printf`) o meter (6) dentro del `if` que acompaña al `printf`
3. El thread 0 obtiene el valor definitivo de `vari` a partir de `med` y `vari` (línea de código 6) e imprime (línea de código 7) sin esperar a que todos los threads acumulen en las variables compartidas `med` y `vari` los resultados parciales que han obtenido en el bucle en las variables locales `med1` y `varil`. Esto supone que cuando imprime pueden haber intentado acumular su resultado parcial el thread 0 y una combinación del resto de threads en un número de 0 a `nthread-1`. Por este motivo, aunque se accediera en exclusión mutua a las variables compartidas, se podrían imprimir distintos resultados en distintas ejecuciones.



(b) Se accederá en exclusión mutua a `med` y `vari` implementando un cerrojo simple con una variable compartida `k1` (ver código en calabaza), se tiene que añadir una barrera antes de que imprima el thread 0 (ver código en azul) y se introduce (6) dentro del if. La barrera se debe implementar también usando un cerrojo simple (`k2`):

```
(1)  for (i=ithread;i<N;i=i+nthread) {
(2)    medl=medl+x[i];
(3)    varil=varil+x[i]*x[i];
(4)  }
(5)  medl=medl/N; varil=varil/N;
      while (test_&_set(k1)) {}; //lock(k1)
      med = med + medl; vari = vari + varil;
      k1=0;                      //unlock(k1)

      bandera_local= !(bandera_local) //se complementa la bandera local
      while (test_&_set(k2)) {};     //lock(k2)
      bar[id].cont+=1; cont_local = bar[id].cont; //cont_local es local
      k2=0;                         //unlock(k2)
      if (cont_local ==num_procesos) {
          bar[id].cont=0;           //se hace 0 el contador asociado a la barrera
          bar[id].bandera= bandera_local; // libera procesos en espera
      }
      else while (bar[id].bandera!= bandera_local) {};

(6)  if (ithread==0) { vari= vari - med*med;
(7)      printf("varianza = %f", vari);} //imprime en pantalla
```

`k1`, `k2`, `bandera_local`, `cont_local`, `ithread`, `i`, `medl` y `varil` son variables locales; el vector `bar`, `med`, `vari`, el vector `x` y `N` son variables compartidas; inicialmente `k1`, `k2`, `med`, `vari`, `medl` y `varil` son 0.

(c) Con `fetch_&_add()`, para el acceso en exclusión mutua no es necesario usar variables compartidas extras como cerrojos:

```
(1)  for (i=ithread;i<N;i=i+nthread) {
(2)    medl=medl+x[i];
(3)    varil=varil+x[i]*x[i];
(4)  }
(5)  fetch_&_add(med,medl/N); fetch_&_add(vari,varl/N);

      bandera_local= !(bandera_local) //se complementa la bandera local
      cont_local = fetch_&_add(bar[id].cont,1); //cont_local es local
      if (cont_local==num_procesos-1) {
          bar[id].cont=0;           //se hace 0 el contador asociado a la barrera
          bar[id].bandera= bandera_local; // libera procesos en espera
      }
      else while (bar[id].bandera!= bandera_local) {};

(6)  if (ithread==0) { vari= vari - med*med;
(7)      printf("varianza = %f", vari);} //imprime en pantalla
```

`bandera_local`, `cont_local`, `ithread`, `i`, `medl` y `varil` son variables locales; el vector `bar`, `med`, `vari`, el vector `x` y `N` son variables compartidas; inicialmente `med`, `vari`, `medl` y `varil` son 0.

(d) Con `compare_&_swap()`, para el acceso en exclusión mutua no es necesario usar variables compartidas extras como cerrojos:

```

(1) for (i=ithread;i<N;i=i+nthread) {
(2)     medl=medl+x[i];
(3)     varil=varil+x[i]*x[i];
(4) }
(5) medl=medl/N; varil=varil/N;
do
    a = med;
    b = a + medl; //a y b son variables locales
    compare&swap(a,b,med);
while (a!=b);
do
    a = vari;
    b = a + varil;
    compare&swap(a,b,vari);
while (a!=b);

bandera_local= !(bandera_local)//se complementa la bandera local
do
    cont_local = bar[id].cont;
    b = cont_local + 1;
    compare&swap(cont_local,b,bar[id].cont);
while (cont_local!=b);
if (cont_local==num_procesos-1) {
    bar[id].cont=0; //se hace 0 el contador asociado a la barrera
    bar[id].bandera= bandera_local; // libera procesos en espera
}
else while (bar[id].bandera!= bandera_local) {};
(6) if (ithread==0) { vari= vari - med*med;
(7)                         printf("varianza = %f", vari);} //imprime en pantalla

```

a, b, bandera_local, cont_local, ithread, i, med1 y varil son variables locales; el vector bar, med, vari, el vector x y N son variables compartidas; inicialmente med, vari, med1 y varil son 0.

Ejercicio 13. Se ha extraído la siguiente implementación de cerrojo (spin-lock) para x86 del kernel de Linux (<http://lxr.free-electrons.com/source/arch/x86/include/asm/spinlock.h>):

```
typedef struct {
    unsigned int slock;
} raw_spinlock_t;
...
/*Para un n mero de procesadores menor que 256=2^8
-#if (NR_CPUS < 256)
...
-static __always_inline void __ticket_spin_lock(raw_spinlock_t *lock)
-{
-    short inc = 0x0100;
-
-    asm volatile (
-        "lock xaddw %w0, %l\n" /*w: se queda con los 16 bits menos significativos*/
-        "1: \t" /*t: se queda con el byte menos significativo*/
-        "cmpb %h0, %b0 \n\t" /*h: coge el byte que sigue al menos significativo*/
-        "je 2f \n\t" /*f: forward */
-        "rep ; nop \n\t" /*retardo, es equivalente a pause*/
-        "movb %1, %b0 \n\t"
-        /* don't need lfence here, because loads are in-order */
-        "jmp 1b \n" /*b: backward */
-        "2:"
```

```

-      : "+Q" (inc), "+m" (lock->slock) /*%0 es inc, %1 es lock->slock */
- /*Q asigna cualquier registro al que se pueda acceder con rh: a, b, c y d; ej. ah, bh ...
*/
-
-      :
-      : "memory", "cc");
-}
-
static __always_inline void __ticket_spin_unlock(raw_spinlock_t *lock)
{
    asm volatile( "incb %0" /*%0 es lock->slock */
    : "+m" (lock->slock)
    :
    : "memory", "cc");
}

```

Conteste a las siguientes preguntas:

- (a) Utiliza una implementación de cerrojo con etiquetas ¿Cuál es el contador de adquisición y cuál es el contador de liberación?
- (b) Describa qué hace `xaddw %w0, %1` ¿opera con el contador de adquisición, con el de liberación o con los dos? ¿qué operaciones hace con ellos?
- (c) Describa qué hace `cmpb %h0, %b0` ¿opera con el contador de adquisición, con el de liberación o con los dos? ¿qué operaciones hace con ellos?
- (d) ¿Por qué cree que se usa el prefijo `lock` delante de la instrucción `xaddw`?

NOTAS: (1) Puede consultar las instrucciones en el manual de Intel con el repertorio de instrucciones (Volumen 2 o volúmenes 2A, 2B y 2C) que puede encontrar aquí <http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html>.

(2) Si no recuerda la interfaz entre C/C++ y ensamblador en `gcc` (se ha presentado en Estructura de Computadores), consulte el manual de `gcc` aquí <http://gcc.gnu.org/onlinedocs/gcc-4.6.2/gcc/Extended-Asm.html#Extended-Asm> (<http://gcc.gnu.org/onlinedocs/>)

Solución

- (b) `lock->slock` contiene el contador de liberación en los bits de 0 a 7 liberación (`lock->slock[7...0]`) y el de adquisición en los bits de 8 a 15 (`lock->slock[15...8]`).
- (c) `xaddw %w0, %1` almacena en los 16 bits menos significativos del registro al que se ha asigna `inc` (%0) los 16 bits (sufijo w) menos significativos de `lock->slock` (%1) y asigna a `lock->slock` (contador de adquisición y contador de liberación) el resultado de sumarlo con `inc`. Como consecuencia: **(1)** incrementa en uno el contador de adquisición (`lock->slock[15...8]`) dado que `inc` tiene un 1 en el bit 8 (`inc` contiene 0x0100) y **(2)** almacena en `inc[15...8]` (%h0) el valor de este contador antes de la modificación y en `inc[7...0]` (%b0) el valor del contador de liberación (`lock->slock[7...0]`).
- (d) `cmpb %h0, %b0` compara el valor actual del contador de liberación `inc[7...0]` (%b0) y el de adquisición `inc[15...8]` (%h0); es decir, resta ambos contadores modificando sólo el registro de estado. En las instrucciones posteriores se usa el resultado de la comparación (los bits de estado resultantes de la comparación). Si son iguales ambos contadores (bit z del registro de estado a 1), abandona la función `lock` del cerrojo, y si son distintos actualiza el valor del contador de liberación cargando lo que hay en `lock->slock[7...0]` en `inc[7...0]` (%b0)
- (e) Se requiere el prefijo `lock` para que la lectura y escritura en memoria que realiza la instrucción `xaddw` se hagan de forma atómica. Si `xaddw` no fuese atómica dos flujos de control podrían leer el mismo valor del contador de adquisición y, como consecuencia, más de un flujo podría entrar a la vez en una sección crítica.

2 Cuestiones

Cuestión 1. Diferencias entre núcleos con multithread temporal y núcleos con multithread simultánea.

Solución

Un núcleo con multithread simultánea es un núcleo superescalar que puede emitir a unidades funcionales instrucciones de múltiples threads distintos por lo que puede ejecutar operaciones de distintos threads en paralelo. Mientras que un núcleo con multithread temporal es un núcleo segmentado, superescalar o VLIW que ejecuta varios threads concurrentemente, pero no en paralelo (sólo emite instrucciones de un thread a unidades funcionales), es decir, multiplexando en el tiempo el uso de las etapas del cauce entre los threads. Para la multiplexación tiene hardware que se encarga de conmutar entre threads.

Cuestión 2. Diferencias entre núcleos con multithread temporal de grano fino y núcleos con multithread temporal de grano grueso.

Solución

En un FGMT el hardware puede conmutar de thread cada ciclo de reloj, con una penalización de 0 ciclos, por turno rotatorio o por un evento (dependencia funcional, fallo en cache de nivel 1, operación con CPI de varios ciclos, salto no predecible) combinado con alguna técnica de planificación (p. ej. Thread que lleva menos sin ejecutarse), mientras que en un CGMT se conmuta tras varios ciclos de reloj, con una penalización que puede ser distinta de 0, tras un nº de ciclos prestablecido o por un evento estático (ejecución de una instrucción añadida al repertorio o ya incorporada -carga/almacenamiento, salto) o dinámico (como el fallo de la cache de último nivel o una interrupción).

Cuestión 3. Suponga un multiprocesador con protocolo MESI de espionaje. Si un controlador de cache observa en el bus un paquete de petición de lectura exclusiva de un bloque que tiene en estado S, debe (indique cuál sería la respuesta correcta y razoné por qué es la respuesta correcta):

- Generar un paquete de respuesta con el bloque y pasar el bloque a estado I.
- Pasar el bloque a estado I.
- Generar un paquete de respuesta con el bloque y pasar el bloque a estado E.
- No tiene que hacer nada

Solución

(b) Pasar el bloque a estado I.

Al estar su copia del bloque en estado Compartido no necesita entregar copia del bloque, porque lo tiene válido la memoria y será ella quien generará un paquete de respuesta con el bloque. Al ser la petición de acceso exclusivo al bloque, significa que quien ha enviado el paquete va a escribir en el bloque, por tanto, la copia que tiene el nodo ya no será válida, de ahí que pase a estado Inválido.

Cuestión 4. Suponga un multiprocesador con protocolo MESI de espionaje. Si un nodo observa en el bus un paquete de petición de lectura exclusiva de un bloque que tiene en estado M, ¿qué debe hacer? Razoné su respuesta.

Solución

- Generar un paquete de respuesta con el bloque, porque el paquete de petición incluye lectura del bloque y la memoria no tiene copia válida, él tiene la única copia válida en todo el sistema.



- (2) Pasar el bloque a estado I, porque si se solicita un acceso exclusivo al bloque es porque quien ha enviado el paquete va a escribir en su cache en la copia que va a recibir, por tanto, la copia que está en estado M ya no estará actualizada, será inválida.

Cuestión 5. Suponga un multiprocesador con el protocolo MESI de espionaje. Si el procesador de un nodo escribe en un bloque que tiene en su cache en estado I, debe (indique cuál sería la respuesta correcta y razoné por qué es la respuesta correcta):

- a) Generar paquete de petición de acceso Exclusivo al bloque y pasar el bloque a M
- b) Generar paquete de petición de acceso Exclusivo al bloque y pasar el bloque a estado E
- c) Generar paquete de petición de acceso Exclusivo al bloque con lectura y pasar el bloque a estado E
- d) Generar paquete de petición de acceso Exclusivo al bloque con lectura y pasar el bloque a M

Solución

- (d) Generar paquete de petición de acceso Exclusivo al bloque con lectura y pasar el bloque a M

Razonamiento:

(1) Paquete con lectura: como la cache del nodo no tiene copia válida del bloque (debido al estado inválido), el controlador de cache del nodo tiene que generar un paquete que incluya lectura.

(2) Paquete con acceso exclusivo: como se va a escribir en la copia que se reciba del bloque, el controlador de cache del nodo tiene que pedir acceso exclusivo para que las copias del bloque en otras caches sean invalidadas. Esto es necesario porque las siguientes lecturas del bloque que se realicen en otros nodos deberán acceder a la última modificación del mismo que es la que se va a realizar en este nodo en lugar de acceder a copias no actualizadas del bloque que estén en sus caches.

(3) Pasar a estado modificado: como el nodo va a escribir en el bloque cuando lo reciba será la única copia válida del bloque en el sistema, por tanto, deberá estar en estado modificado para que el controlador de cache sepa que debe responder con el bloque si se recibe alguna petición que incluya lectura del bloque.

Cuestión 6. ¿Cuál de los siguientes modelos de consistencia permite mejores tiempos de ejecución?

Justifique su respuesta.

- a) modelo de ordenación débil
- b) modelo implementado en los procesadores de la línea x86
- c) modelo de consistencia secuencial
- d) modelo de consistencia de liberación

Solución

- (d) modelo de consistencia de liberación

Tanto el modelo de ordenación débil como el de consistencia de liberación relajan todos los órdenes entre operaciones de acceso a memoria (W->R, W->W, R->W,R) por lo que permitirán mejores tiempos de ejecución que el resto porque ningún acceso tiene que esperar a otro.

Pero el de liberación ofrece mejores prestaciones que el de ordenación débil porque:

(1) El de ordenación débil ofrece instrucciones máquina que permiten que, si S es una operación de sincronización, se garanticen los siguientes órdenes:

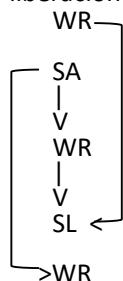
- S->WR (hasta que no termina la operación de sincronización no puede empezar ningún acceso a memoria posterior)
- WR->S (hasta que no terminen los accesos a memoria que hay antes de una operación de sincronización no puede empezar la operación de sincronización)

Con estos órdenes los accesos a memoria que hay detrás de una sincronización no pueden empezar hasta que no hayan acabado todos los accesos que hay delante de la sincronización. Por tanto, se garantiza el siguiente orden entre operaciones de acceso a memoria y operaciones de sincronización de adquisición SA y de liberación SL: WR->SA->WR->SL->WR. No hay ningún tipo de paralelismo entre operaciones de acceso a memoria antes y después de operaciones de sincronización

(2) Mientras que el modelo de liberación ofrece instrucciones máquina menos restrictivas que permiten garantizar los siguientes órdenes entre accesos a memoria de lectura L y escritura W y operaciones de sincronización de adquisición SA y liberación SL:

- SA->WR (hasta que no termina la operación de sincronización de adquisición no puede empezar ningún acceso a memoria posterior)
- WR->SL (hasta que no terminen los accesos a memoria que hay antes de una operación de sincronización de liberación no puede empezar la operación de sincronización de liberación)

Con estos órdenes los accesos a memoria que hay detrás de una sincronización de liberación pueden empezar aunque no hayan terminado los que hay delante, y los que hay delante de una operación de adquisición pueden terminar después de las que hay detrás. O sea, las operaciones de acceso a memoria que hay antes de la operación de adquisición se pueden realizar en paralelo a las operaciones que hay detrás de la adquisición y antes de la liberación, y las operaciones de acceso a memoria que hay detrás de la operación de liberación se pueden realizar en paralelo a las que hay entre la operación de adquisición y la de liberación:



Cuestión 7. Indique qué expresión no se corresponde con la serie (justifique su respuesta):

- Lock
- Fetch_and_Or
- Compare_and_Swap
- Test_and_Set

Solución

Hay una función software de cerrojos, lock, y tres instrucciones máquina utilizadas para mejorar prestaciones en la sincronización de flujos de control: Fetch_and_Or, Compare_and_Swap y Test_and_Set.

Luego lock no se corresponde con la serie.

Arquitectura de Computadores (AC)

Examen de Prácticas. 24 de junio de 2014.

Puntuación: 2 puntos

Duración: 1 hora

Identificación: DNI

Cuestión 1.(0.5 puntos) Se necesita ejecutar un código OpenMP con 8 threads. Comente todas las alternativas que puede usar con OpenMP para fijar el número de threads a 8 y ordénelas en función de su prioridad (menor a mayor).

Respuesta:

Las alternativas para modificar el número de threads son 3:

- Variable de entorno OMP_NUM_THREADS
Para fijar el número de threads a 8 se usaría la siguiente sentencia en la terminal:
export OMP_NUM_THREADS=8
- Función omp_set_num_threads()
Para fijar el número de threads a 8 se usaría la siguiente sentencia dentro del programa antes de la región paralela:
Omp_set_num_threads(8)
- Clausula num_threads
Para fijar el número de threads a 8 se añadiría la siguiente cláusula a la cabecera de la región paralela:
num_threads(8)
ejemplo de uso: *#pragma omp parallel num_threads(8){...}*

Cuestión 2. (1 puntos) (a) (0.5) Implemente un código paralelo OpenMP que calcule el producto escalar de dos vectores, $x[]$ e $y[]$ de $N=1000$ componentes (declare los componentes como variables globales, implemente en paralelo también la inicialización de los vectores): $z = \sum_{i=0}^{N-1} x(i) \bullet y(i)$.

(b) (0.25) Comente para qué ha usado cada una de las directivas y cláusulas que ha incluido en el código.

(c) (0.05) Indique qué orden o comando usaría para compilar el código desde una ventana de comandos (terminal) si el ejecutable se quiere llamar *pescalar* (utilice en la compilación la opción de optimización con la que ha obtenido mejores tiempos en la práctica de optimización de código).

(d) (0.2) Suponga que debe ejecutar en atcgrid el fichero ejecutable *pescalar* que tiene en el PC del aula de prácticas (o en su portátil), ¿qué haría para ejecutarlo en atcgrid (tenga en cuenta que está en el PC del aula o en su portátil)? ¿Cómo sabría que ya ha terminado la ejecución? ¿dónde podría consultar los resultados de la ejecución?

Respuesta:

a)

Versión 1:

```
#include <stdio.h>
#include <stdlib.h>
#ifndef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

struct tipo{
    int x;
```

Versión 2:

```
#include <stdio.h>
#include <stdlib.h>
#ifndef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

const int N=1000;
int resultado=0,x[N],y[N];
```

<pre> int y; }; const int N=1000; int resultado=0; tipo estructura[N]; int main(int argc, char **argv) { #pragma omp parallel { #pragma omp for for(int i=0; i<N; i++){ estructura[i].x=i; estructura[i].y=i; } #pragma omp for reduction(+:resultado) for(int i=0; i<N; i++){ resultado+=estructura[i].x*estructura[i].y; } } printf("Fuera de 'parallel for' resultado=%d\n",resultado); return 0; } </pre>	<pre> int main(int argc, char **argv) { #pragma omp parallel { #pragma omp for for(int i=0; i<N; i++){ x[i]=i; } #pragma omp for for(int i=0; i<N; i++){ y[i]=i; } #pragma omp for reduction(+:resultado) for(int i=0; i<N; i++){ resultado+=x[i]*y[i]; } printf("Fuera de 'parallel for' resultado=%d\n",resultado); return 0; } } </pre>
--	--

b)

he utilizado la primera directiva “`#pragma omp parallel`” para crear las hebras y no hacerlo reiteradamente más adelante. Anidadas dentro de ella están las demás directivas.

Versión 1: la segunda directiva “`#pragma omp for`” la he utilizado para repartir las iteraciones del bucle entre las distintas hebras de forma equitativa (por defecto es static con chunk 1). Esta se encarga de inicializar el vector.

Versión 2: la segunda y la tercera directiva “`#pragma omp for`” la he utilizado para repartir las iteraciones del bucle entre las distintas hebras de forma equitativa (por defecto es static con chunk 1). Estas se encargan de inicializar el vector.

La ultima directiva con la cláusula reduction “`#pragma omp for reduction(+:resultado)`” la he utilizado para repartir las iteraciones del bucle entre las distintas hebras de forma equitativa. La he utilizado con la cláusula reduction para que en cada hebra se genere una variable privada llamada resultado donde cada hebra va a ir almacenando la suma del producto de cada una de sus iteraciones y al final, cuando todas las hebras terminen la cláusula se encargara de unificar todas las variables privadas creadas anteriormente en la variable resultado global usando el operador “+” que es el que se ha especificado en la cláusula, realizando correctamente el producto escalar de los vectores.

c)

`g++ -fopenmp -O2 pescalar.c -o pescalar`

d)

paso 1: conectar con ssh y sftp al front-end de atcgrid

paso2: compilar en pc-local y enviarlo al frontend (en sftp: “`put pescalar`”)

paso3: encolar el ejecutable en la cola ac de torque con el comando “`echo './pescalar' | qsub -q ac`” en ssh

paso 4: (una vez haya terminado la ejecución) copiar los resultados al pc-local (en sftp: “`get STDIN*`”)

paso 5: visualizar el fichero en el pc-local (en terminal con `cat` o en entorno gráfico con `gedit`)

paso 6: eliminar los ficheros generados en el front-end (en ssh: “`rm STDIN*`”)

para comprobar si la ejecución ha terminado o no se usa el comando `qstat` el cual te devuelve una tabla en la que se

¿BUSCAS LIBROS O EBOOKS SOBRE ENFERMERÍA?

AXON
axon.es

AXON librería especializada en Ciencias de la Salud.



Encuentra todos los libros y eBooks para tu especialización, además del material complementario que necesites.

pueden ver los procesos de la cola y algunas características. Una de ellas es el estado (columna S) si tiene valor C es que el proceso ya ha terminado.

Los resultados de la ejecución se generan en dos ficheros:

STDIN.o<identificador>: en este fichero se almacena la salida estándar de la ejecución.

STDIN.e<identificador>: en este fichero se almacena la salida estándar de error de la ejecución.

Cuestión 3. (0.5 puntos) (a) (0.25) ¿Cuál de los siguientes códigos para C/C++ ofrece mejores prestaciones? ¿Por qué?

double m1[n][n], m2[n][n], mr[n][n] ; ... for (i=0; i<n; i++){ for (j=0; j<n; j++){ for(k=0; k<n; k+=4){ mr[i][j] += m1[i][k] * m2[k][j]; mr[i][j] += m1[i][k+1] * m2[k+1][j]; mr[i][j] += m1[i][k+2] * m2[k+2][j]; mr[i][j] += m1[i][k+3] * m2[k+3][j]; } } }	double m1[n][n], m2[n][n], mr[n][n] ; ... for (i=0; i<n; i++){ for (j=0; j<n; j+=4){ for(k=0; k<n; k++){ mr[i][j] += m1[i][k] * m2[k][j]; mr[i][j+1] += m1[i][k] * m2[k][j+1]; mr[i][j+2] += m1[i][k] * m2[k][j+2]; mr[i][j+3] += m1[i][k] * m2[k][j+3]; } } }
---	---

(b) (0.25) ¿Cuál de los siguientes códigos para C/C++ ofrece mejores prestaciones? ¿Por qué?

struct { int a; int b; } s[5000]; main() { ... for (ii=1; ii<=40000;ii++) { for(i=0;i<5000;i++) X1+=2*s[i].a+ii; for(i=0;i<5000;i++) X2+=3*s[i].b-ii; ... //instrucciones que usan X1 y X2 } }	struct { int a; int b; } s[5000]; main() { ... for (ii=1; ii<=40000;ii++) { for(i=0;i<5000;i++) { X1+=2*s[i].a+ii; X2+=3*s[i].b-ii; } ... //instrucciones que usan X1 y X2 } }
---	--

Respuesta:

- a) el código de la derecha es más eficiente debido a la localidad en memoria.

Si nos fijamos, en el código de la izquierda hay dependencia raw en $mr[i][j]$, además, se produce un fallo de cache en cada línea dentro del bucle for, ya que el segundo operando en cada línea accede a una fila distinta de la matriz. Estos dos problemas se solucionan en el código de la derecha haciéndolo más eficiente.

- b) El código de la derecha es más eficiente debido a la localidad en memoria.

Si nos fijamos, en el código de la izquierda se accede a memoria pegando saltitos debido a que las estructuras almacenan sus campos unos contiguos a los otros al igual que los vectores, entonces las posiciones $s[0].a$ y $s[0].b$ estarán contiguas, pero $s[0].a$ y $s[1].a$ no. A demás el código de la izquierda tiene dependencia raw tanto en $X1$ y $X2$. Por lo tanto, el código de la derecha es mucho mejor, ya que resuelve los problemas mencionados anteriormente



1. En un computador de tipo NORMA tanto los accesos a memoria local como los de acceso a memoria remota se realizan a través de instrucciones de carga y almacenamiento de datos en memoria

FALSO

2. Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse entre si

FALSO

3. Un multiprocesador puede funcionar como computador MISD con la sincronización adecuada entre sus procesadores

VERDADERO

4. Un programa tiene 1000 millones de instrucciones y se ejecuta en un computador que tiene cinco tipos de instrucciones. Las del tipo 1 necesitan 6 ciclos, las del tipo 2 necesitan 4 ciclos, las del tipo 3 necesitan 3 ciclos, las del tipo 4 necesitan 5 ciclos y las del tipo 5 necesitan 2. Si entre las instrucciones ejecutadas por el programa hay un 20% de instrucciones de cada uno de los tipos. ¿Cuántos segundos tarda el programa en ejecutarse en el computador si utiliza un reloj de 2 GHz (indique un número entero)?

2

5. Un cluster de computadores es un computador NUMA

FALSO

6. El paralelismo entre hebras permite aprovechar una granularidad más fina que el paralelismo entre procesos

VERDADERO

7. ¿Cuál es el número de GIPS que puede alcanzar un núcleo superescalar que funciona a 2GHz y es capaz de terminar 4 instrucciones por ciclo (introduzca un número entero)?

8

8. En la secuencia de instrucciones que aparecen en el orden indicado en un código:

- (i1) add r1, r2, r4 ; r1 ← r2 + r4
- (i2) add r4, r2, r3 ; r4 ← r2 + r3
- (i3) sub r1, r1, r4 ; r1 ← r1 - r4

Hay dependencia RAW entre las instrucciones i2 e i3 debido al registro r4

VERDADERO

9. En la expresión de la ley de Amdahl, $S_p \leq p/(1+f(p-1))$ para la ganancia de velocidad de un computador al mejorar uno de sus recursos, p es el factor de incremento de prestaciones del recurso que se mejora

VERDADERO

10. Dado el bucle $i=1$ to N do $a(i)=b(i)+c(i)$, en el que $a()$, $b()$, y $c()$ son números en coma flotante, ¿cuántos GFLOPS consigue un computador que lo ejecuta en 2 segundos cuando $N=10^{12}$ (introduzca un número entero)?

500

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos p puede ser mayor que 1

Verdadero

toxico TOXICO TOXIC

2. Un multiprocesador puede funcionar como computador MISD con la sincronización adecuada entre sus procesadores

Verdadero

3. En la expresión de la ley de Amdahl, $Sp \leq p/(1 + f(p-1))$ para la ganancia de velocidad de un computador al mejorar uno de sus recursos, p es el factor de incremento de prestaciones del recurso que se mejora

Verdadero

4. En la secuencia de instrucciones:

- (a) add r1,r2, r3 ; r1 <- r2 + r3
- (b) sub r1,r1, r4 ; r1 <- r1 - r4

Hay dependencia WAW entre las instrucciones debido al registro r1

Verdadero

5. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo. ¿Cuál es la velocidad pico (en GFLOPS) de un microprocesador con 4 núcleos Sunday Bridge que funciona a una frecuencia de reloj de 2 GHz?

64

6. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$ para la ganancia de velocidad de un computador al mejorar uno de sus recursos, f es la fracción de tiempo ant de la mejora en la que se utiliza el recurso mejorado

Falso

7. Un computador NUMA, es un multiprocesador donde la memoria está físicamente distribuida. Verdadero

8. Si el bucle siguiente: for i=1 to N do a(i)=b(i)*c; se ejecuta en 2 segundos y N=10^11, siendo c, a(), y b() datos en coma flotante, ¿cuántos GFLOPS alcanza la máquina al ejecutar el código? **50**

9. Un cluster de computadores es un computador NUMA. **Falso**

10. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; r1 ← r2 + r3
- (b) sub r1, r1, r4 ; r1 ← r1 - r4

No hay dependencia WAR entre las instrucciones debido al registro r1.

Verdadero

TEST 2:

1. Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse entre sí

Falso

2. Un programa tiene 1000 millones de instrucciones y se ejecuta en un computador que tiene cuatro tipos de instrucciones. Las del tipo 1 necesitan 6 ciclos, las del tipo 2 necesitan 5 ciclos, las del tipo 3 necesitan 3 (Wos, y las del tipo necesitan 2 ciclos. Si entre las instrucciones ejecutadas por el programa hay un 25% de instrucciones de cada uno de los tipos. ¿Cuántos segundos tarda el programa en ejecutarse en el computador si utiliza un reloj de 1 GHz?

Sin respuesta

3. En un procesador superescalar el valor de CPI puede ser menor que 1

Verdadero

4. ¿Cuál es el número de GIPS que puede alcanzar un núcleo superescalar que funciona a 2GHz y es capaz de terminar 4 instrucciones por ciclo?

8

5. En la secuencia de instrucciones que aparecen en el orden indicado en un código:

(i1) add r1, r2, r3 ; r1 <- r2 + r4

(i2) add r4, r2, r3 ; r1 <- r2 + r3

(i3) sub r1, r1, r4 ; r1 <- r1 - r4

Hay dependencia WAR entre las instrucciones i1 e i2 debido al registro r4

Verdadero

6. En la secuencia de instrucciones que aparecen en el orden indicado en un código:

(i1) add r1, r2, r3 ; r1 <- r2 + r4

(i2) add r4, r2, r3 ; r1 <- r2 + r3

(i3) sub r1, r1, r4 ; r1 <- r1 - r4

Hay dependencia RAW entre las instrucciones i2 e i3 debido al registro r4

Verdadero

1. En la secuencia de instrucciones:

(a) add r1, r2, r3; r1 <- r2 + r3

(b) sub r1, r1, r4; r1 <- r1 - r4

No hay dependencia WAR entre las instrucciones debido al registro r1

VERDADERO

2. En la secuencia de instrucciones:

(a) add r1, r2, r3; r1 <- r2 + r3

(b) sub r1, r1, r4; r1 <- r1 - r4

Solo hay dependencia RAW entre las instrucciones debido al registro r1

FALSO

3. Dado el bucle for i=1 to N do a(i)=b(i)+c(i), en el que a(), b(), y c() son números en coma flotante, ¿Cuántos GFLOPS consigue un computador que lo ejecuta en 2 segundos cuando N=10^12?

500

4. En la expresión de la ley de Amdahl, $SP \leq p(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos, f es la parte del tiempo antes de la mejora en la que se utiliza el recurso mejorado.

FALSO

5. En la expresión de la ley de Amdahl, $SP \leq p(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos, p y f pueden ser mayor que 1.

FALSO

7. Dado el bucle for i=1 to N do $a(i)=k*b(i)+c(i)$, en el que a(),b(),c(), y k son números en flotante, ¿cuántos GFLOPS consigue un computador que lo ejecutan en 4 segundos cuando $N=10^{10}$?

5

8. En una computadora NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida

Verdadera

9. En la expresión de la ley de Amdahl, $S_p \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos, p no puede ser nunca mayor que 1

Falso

10. En la expresión de la ley de Amdahl, $S_p \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos, la máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso, es $1/(1-f)$

Falso

6. ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta dos instrucciones por ciclo y funciona a una frecuencia de reloj de 1 GHz?

2000

11. Los multicomputadores son máquinas MIMD y los multiprocesadores SIMD

FALSO

12. En un computador de tipo NORMA tanto los accesos a memoria local como los de acceso a memoria remota se realizan a través de instrucciones de carga y almacenamiento de datos de memoria

FALSO

13. Un programa tiene 1000 millones de instrucciones y se ejecuta en un computador que tiene cinco tipos de instrucciones. Las del tipo 1 necesitan 6 ciclos, las del tipo 2 necesitan 4 ciclos, las del tipo 3 necesitan 3 ciclos, y las del tipo 4 necesitan 5 ciclos y las del tipo 5 necesitan 2. Si entre las instrucciones ejecutadas por el programa hay un 20% de instrucciones de cada uno de los tipos. ¿Cuántos segundos tarda el programa en ejecutarse en el computador si utiliza un reloj de 2 GHz?

2

14. El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos

VERDADERO

¿Qué devuelve clock_gettime() tal y como se usa en el ejemplo SumaVectores.c?

- a) Devuelve el tiempo transcurrido en la ejecución de un trozo de código.
- b) Ninguna de las otras respuestas es correcta
- c) Devuelve el tiempo transcurrido desde las cero horas del 1 de enero de 1970.
- d) Devuelve el tiempo transcurrido en el ejecución del programa.

Añadir la opción -n1 a srun permite asegurar

- a) que el programa que va a ejecutar usa solamente un nodo de cómputo de atcgrid
- b) que el programa que va a ejecutar usa solamente un núcleo lógico de uno de los nodos de cómputo de atcgrid
- c) que el programa que va a ejecutar usa solamente un núcleo físico de uno de los nodos de cómputo de atcgrid
- d) que se crea solamente un proceso del programa que va a ejecutar

¿Qué opción añadiría a srun para que use solamente un núcleo lógico por cada núcleo físico que se asigne a la ejecución de su programa?

- a) --cpus-per-task=1
- b) --exclusive
- c) --hint=nomultithread
- d) -n1

Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid: sbatch -p ac --account=ac -n1 --cpus-per-task=6 -hint=nomultithread script.sh
Teniendo en cuenta esta orden va a utilizar un máximo de núcleos físicos de

- a) 6
- b) 3
- c) 24
- d) 12

¿Cuál de las siguientes órdenes usaría para que se cree un único proceso con el código paralelo HelloOMP que use 12 hebras. de forma que cada hebra se ejecute en un núcleo físico distinto de un nodo de cómputo de atcgrid?

- a) srun -p ac --account=ac -n1 --cpus-per-task=8 HelloOMP
- b) srun -p ac --account=ac -n1 HelloOMP
- c) srun -p ac --account=ac -n1 --cpus-per-task=24 HelloOMP
- d) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread HelloOMP

Se ha usado la siguiente orden ejecutar script.sh en un nodo de cómputo de atcgrid: sbatch -p ac --account=ac -n1 --cpus-per-task=6 script.sh

Teniendo en cuenta esta orden y el proceder por defecto de slurm en atcgrid va a utilizar un máximo núcleos físicos de

- a) 24
- b) 6
- c) 3
- d) 12

ARQUITECTURA DE COMPUTADORES
GRUPO A. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $S_p \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- p es el factor de incremento de prestaciones del recurso que se mejora (V)
- f es la fracción del tiempo antes de la mejora en el que no se utiliza el recurso mejorado (V)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/(1-f)$ (F)
- f puede ser mayor que 1 (F)

2. En un procesador superescalar a pleno rendimiento, el número de ciclos por instrucción (CPI) es menor que 1 (responda Verdadero, V, o Falso, F)

(V)

3. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo.

¿Cuál es la máxima velocidad (en GFLOPS) de un núcleo con dicha arquitectura que funciona a una frecuencia de reloj de 2.5 GHz?

$$8 \text{ FLOP/ciclo} * 2.5 \text{ (Gciclos/s)} = 20 \text{ GFLOPS}$$

4. Responda Verdadero (V) o Falso (F):

- En un computador NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida (V)
- Un multicomputador también se denomina computador UMA (F)

5. Si el bucle siguiente: for i=1 to N do a(i)=b(i)*c; se ejecuta en 5 segundos y $N=10^{12}$, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$1*10^{12} \text{ (FLOP)}/(5s*10^9)=1000/5 \text{ GFLOPS} = 200 \text{ GFLOPS}$$

6. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso comparten la memoria asignada al proceso, los registros, la pila y el contador de programa (F)
- Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse (F)
- Un multiprocesador puede funcionar como un computador MISD. (V)

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; $r1 \leftarrow r2 + r3$
- (b) sub r1, r1, r4 ; $r1 \leftarrow r1 - r4$

- Hay dependencia RAW entre las instrucciones debido al registro r1 (V)
- Hay dependencia WAR entre las instrucciones debido al registro r1 (F)

ARQUITECTURA DE COMPUTADORES
GRUPO B. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- p es el factor de incremento de prestaciones del recurso que se mejora (V)
- f es la fracción del tiempo antes de la mejora en la que se utiliza el recurso mejorado (F)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/f$ (V)
- p puede ser mayor que 1 (V)

2. En un procesador segmentado a pleno rendimiento, el número de ciclos por instrucción (CPI) es (estrictamente) menor que 1 (responda Verdadero, V, o Falso, F)

(F)

3. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo.

¿Cuál es la máxima velocidad (en GFLOPS) de un microprocesador con 4 núcleos Sunday Bridge que funciona a una frecuencia de reloj de 2 GHz?

$$8 \text{ FLOP}/(\text{núcleo*ciclo}) * 2 \text{ Ciclos/s} * 4 \text{ núcleos} = 64 \text{ GFLOPS}$$

4. Responda Verdadero (V) o Falso (F):

- Un computador UMA, es un multiprocesador donde la memoria está físicamente distribuida. (F)
- Un multicomputador también se denomina computador NORMA (V)

5. Si el bucle siguiente: for $i=1$ to N do $a(i)=b(i)*c$; se ejecuta en 2 segundos y $N=10^{11}$, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?.

$$1*10^{11} \text{ FLOP} / 2 \text{ s} *10^9 = 100/2 \text{ GFLOPS} = 50 \text{ GFLOPS}$$

6. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso necesitan recurrir a llamadas al sistema operativo para comunicarse (F)
- El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos (V)
- Un multiprocesador puede funcionar como computador MISD con la correspondiente sincronización entre sus procesadores (V)

7. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; $r1 \leftarrow r2 + r3$
- (b) sub r1, r1, r4 ; $r1 \leftarrow r1 - r4$

- Hay dependencia WAW entre las instrucciones debido al registro r1 (V)
- No hay dependencia WAR entre las instrucciones debido al registro r1 (V)

ARQUITECTURA DE COMPUTADORES

GRUPO IM. BENCHMARK del TEMA 1

Estudiante:

1. En la expresión de la ley de Amdahl, $Sp \leq p/(1+f(p-1))$, para la ganancia de velocidad de un computador al mejorar uno de sus recursos (Responda verdadero (V) o falso (F)):

- f es la fracción del tiempo antes de aplicar la mejora en la que se utiliza el recurso mejorado (F)
- La máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/(1-f)$ (F)
- p no puede ser mayor que 1 (F)

2. Los núcleos de la arquitectura Sunday Bridge de Intel pueden terminar hasta 8 operaciones en coma flotante (FLOP) por ciclo. ¿Cuál es la máxima velocidad (en GFLOPS) de un procesador de 6 núcleos con dicha arquitectura que funciona a una frecuencia de reloj de 2.5 GHz?

$$8 \text{ FLOP}/(\text{ciclo*núcleo}) * 2.5 \text{ (Gciclos/s)} * 6 \text{ núcleos} = 120 \text{ GFLOPS}$$

3. Responda Verdadero (V) o Falso (F):

- En un computador NUMA, la memoria está físicamente distribuida aunque utiliza un modelo de programación de memoria compartida (V)
- Un computador UMA es un tipo de multiprocesador (V)
- En un computador de tipo NORMA los accesos a memoria local y remota se realizan a través de instrucciones de acceso a memoria (carga y almacenamiento de datos en memoria) (F)

4. Si el bucle siguiente: for i=1 to N do a(i)=b(i)+c; y $N=10^{15}$, se ejecuta en 2 segundos en un computador, siendo c, a(), y b() datos en coma flotante. ¿Cuántos TFLOPS alcanza la máquina al ejecutar el código?.

$$1 * 10^{15} \text{ FLOP} / 2 \text{ s} * 10^{12} = 1000/2 \text{ TFLOPS} = 500 \text{ TFLOPS}$$

5. Escriba la expresión de los MIPS en términos del número de ciclos por instrucción (CPI) del procesador, y de su frecuencia de reloj (F). $\text{MIPS} = F/\text{CPI} * 10^6$

8. Responda Verdadero (V) o Falso (F):

- Las hebras de un proceso no necesitan recurrir a llamadas al sistema operativo para comunicarse (V)
- El paralelismo entre hebras permite aprovechar una granularidad menor que el paralelismo entre procesos (V)
- Los multicomputadores son máquinas MIMD y los multiprocesadores SIMD. (F)

9. En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; r1 \leftarrow r2 + r3
- (b) sub r2, r1, r4 ; r2 \leftarrow r1 - r4

- Hay dependencia RAW entre las instrucciones debido al registro r1 (V)
- No hay ninguna dependencia WAR entre las instrucciones (F)
- No hay ninguna dependencia WAW entre las instrucciones (V)

Se ha usado la siguiente orden para script.sh en un nodo de cómputo de atcgrid[1-3]: sbatch -p ac --account=ac -n1 --cpus-per-task=1 --hint=nomultithread script.sh

Teniendo en cuenta esta orden y el proceder por defecto de la instalación slurm de atcgrid, va a utilizar un máximo núcleos lógicos de

- a) 1
- b) 2
- c) Ninguna otra respuesta es correcta
- d) 3

¿Qué orden puedo utilizar para conectarme al front-end de atcgrid para enviar/recibir ficheros?

- a) sftp usuario@atcgrid.ugr.es
- b) ssh usuario@atcgrid.ugr.es
- c) telnet usuario@ugr.es
- d) sbatch -p ac script. sh

¿Con qué comando SLURM se pueden ver todos los trabajos en ejecución y los que están encolados?

- a) sjobs
- b) sinfo
- C) snumjobs
- d) squeue

¿En qué zona de la memoria se alojan los vectores dinámicos de un programa en C?

- a) heap
- b) data
- c) Ninguna otra respuesta es correcta
- d) stack

En un script para SLURM, ¿qué indica la línea #SBATCH --job-name=helloOMP?

- a) El nombre asignado al trabajo
- b) El usuario que manda el trabajo
- c) El directorio donde está el ejecutable
- d) La cola a la que se manda el trabajo

¿Cuál de las siguientes órdenes usaría para que se cree un único proceso con el código paralelo HelloOMP que use 12 hebras, de forma cada hebra distinto de un nodo de cómputo de atcgrid[1-3]?

- a) srun -p ac --account=ac -n1 HelloOMP
- b) srun -p ac --account=ac -n1 --cpus-per-task=8 HelloOMP
- c) srun -p ac --account=ac -n1 --cpus-per-task=24 HelloOMP
- d) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread HelloOMP

¿Cuántos zócalos (sockets) tiene el cluster atcgrid en total sin contar el host o front-end?

- a) 2
- b) 4
- c) 8
- d) 12

¿Cuántos cores lógicos tiene cada microprocesador Intel Xeon Silver 4216?

- a) 64
- b) 12
- c) 32
- d) 16

¿Qué opción de sbatch evita que haya otros trabajos ejecutándose simultáneamente en el mismo nodo al que se envía su trabajo?

- a) **--exclusive**
- b) Ninguna otra respuesta es correcta
- c) --hint=nomultithread
- d) -p ac -n1

¿Cuántos cores lógicos tiene cada nodo de cómputo de atcgrid[1-3]?

- a) 6
- b) 12
- c) 3
- d) **24**



Examen BP0

Universidad de Granada - Doble Grado en Ingeniería Informática y
Matemáticas
Arquitectura de Computadores



Estudiante: 75578698 de la Hera Luis, Alejandro



Inicio: Hoy, miércoles, 09:50:06

Final: Hoy, miércoles, 09:59:11

Preguntas: 10

Respuestas

válidas:



Puntuación:



Nota:



1 ¿Cuántos zócalos (sockets) tiene el cluster atcgrid en total sin contar el host o front-end?

Elección única

Usuario Profesores

- a) 8
- b) 12
- c) 2
- d) 4

2 ¿Cuál es el número de microprocesadores que tiene cada nodo de cómputo de atcgrid?

Elección única

Usuario Profesores

- a) 1
- b) 2
- c) 4
- d) 8

3 Indica los flags correctos para que srun ejecute el programa HelloOMP de forma que se use una sola hebra en cada uno de los cores físicos de un nodo de cómputo de atcgrid[1-3]

Elección única

Usuario Profesores

- a) -n12
- b) -n1 --cpus-per-task=12
- c) -n1 --cpus-per-task=24
- d) -n1 --cpus-per-task=12 --hint=nomultithread

4 Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid[1-3]:

Elección única `sbatch -p ac --account=ac -n1 --cpus-per-task=6 script.sh`

Teniendo en cuenta esta orden y el proceder por defecto de slurm en atcgrid, va a utilizar un máximo núcleos físicos de

Usuario Profesores

- a) 6
- b) 12
- c) 24
- d) 3

5 ¿Con cuántos nodos de cómputo cuenta atcgrid?

Elección única Usuario Profesores

- a) 5
- b) 3
- c) 2
- d) 4

6 En un script para SLURM, ¿qué contiene la variable \$SLURM_JOBID?

Elección única Usuario Profesores

- a) El identificador del usuario que ha mandado el trabajo a la cola
- b) El nombre del trabajo en la cola
- c) El nombre del usuario que ha mandado el trabajo a la cola
- d) El identificador del trabajo en la cola

7 ¿En qué zona de la memoria se alojan los vectores dinámicos de un programa en C?

Elección única Usuario Profesores

- a) heap
- b) Ninguna otra respuesta es correcta
- c) stack
- d) data

8 Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid[1-3]:

Elección única `sbatch -p ac --account=ac -n1 --cpus-per-task=1 script.sh`

Teniendo en cuenta esta orden y el proceder por defecto de la instalación slurm de atcgrid, va a utilizar un máximo de núcleos lógicos de

Usuario Profesores

- a) 6
- b) 1
- c) 2
- d) 12

9 Al ejecutar `srun` seguido con el argumento `-n3`, se está indicando...

Elección única Usuario Profesores

- a) La creación de 3 procesos
- b) El identificador del trabajo
- c) La asignación del trabajo a una CPU específica

**10**

Elección única

¿Con qué comando SLURM se pueden ver todos los trabajos en ejecución y los que están encolados?

Usuario Profesores

- a) `squeue`
- b) `sjobs`
- c) `sinfo`
- d) `snumjobs`



Jose Luis	33"
Rico R...	42"
Salvador	
Molina ...	
Eva	
Rueda ...	
Pablo	
Fuente...	
Javier	
Garrue...	
...	

Sistema

Actividades

Proyectos

Convocatorias

Test

Exámenes

Juegos

Encuestas

?

Examen BP0



Universidad de Granada - Doble Grado en Ingeniería Informática y Matemáticas
Arquitectura de Computadores



Estudiante: 77692362 Molina Plaza, Salvador



Examen de prácticas BP0

10 preguntas

Nota máxima 10

Examen BP0

10 preguntas

1 ¿Con cuántos nodos de cómputo cuenta atcgrid?

Elección
única

- a) 3
- b) 5
- c) 4
- d) 2

2 ¿Cuál de las siguientes órdenes usaría para que se ejecute el código paralelo OpenMP en 12 núcleos lógicos de atcgrid en cualquiera de los nodos atcgrid[1-3]?

Elección
única

- a) srun -p ac --account=ac -n12 codigo
- b) srun -p ac --account=ac -n2 --cpus-per-task=6 --hint=nomultithread codigo
- c) srun -p ac --account=ac -n1 --cpus-per-task=12 codigo
- d) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread codigo

3 ¿Cuál de las siguientes órdenes cree que es más apropiada usar para que su código paralelo OpenMP esté en las mejores condiciones de conseguir un menor tiempo de ejecución?

Elección
única

- a) srun -p ac --account=ac -n1 --hint=nomultithread --exclusive executableOpenMP
- b) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread --exclusive executableOpenMP
- c) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread executableOpenMP
- d) srun -p ac --account=ac -n1 --cpus-per-task=12 --exclusive executableOpenMP

4 Si jobid indica el identificador de un trabajo en la cola de SLURM y jobname su nombre, indica cómo eliminar dicho trabajo de la cola

Elección
única

- a) sdelete jobid
- b) scancel jobid
- c) scancel jobname
- d) sdelete jobname

5 Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid[1-3]:

sbatch -p ac --account=ac -n1 script.sh

Elección
única

Teniendo en cuenta esta orden y el proceder por defecto de la instalación slurm de atcgrid va a utilizar un máximo de núcleos lógicos de

- a) 12
- b) 2
- c) 6
- d) 1

6 ¿Qué indica la variable de entorno SLURM_JOB_NUM_NODES del sistema de gestión de colas de atcgrid?

- a) La lista de nodos asignados al trabajo

Elección
única

- b) Asigna la cola del trabajo
- c) El número de nodos asignados a un trabajo
- d) Especifica un nombre para el trabajo

7Elección
única

En prácticas se recomienda usar un script para ejecutar los trabajos en un nodo de cómputo de atcgrid, además se recomienda utilizar una ejecución en *segundo plano*. Teniendo en cuenta estas recomendaciones, ¿qué usaría para enviar a ejecutar pmvOpenMP a un nodo de cómputo de atcgrid[1-3] utilizando una hebra por cada núcleo físico? (script.sh ejecuta pmvOpenMP):

- a) sbatch -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread pmvOpenMP
- b) sbatch -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread script.sh
- c) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread script.sh
- d) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread pmvOpenMP

8Elección
única

Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid[1-3]:

```
sbatch -p ac --account=ac -n1 --cpus-per-task=6 script.sh
```

Teniendo en cuenta esta orden y el proceder por defecto de slurm en atcgrid, va a utilizar un máximo núcleos físicos de

- a) 24
- b) 3
- c) 6
- d) 12

9Elección
única

Si queremos que el programa HelloOMP se ejecute específicamente en el nodo atcgrid1 utilizando la cola -p ac, se deberá usar la orden:

- a) srun -p ac -n1 ./HelloOMP
- b) No es posible puesto que el gestor de colas es quien decide el nodo al que se mandará la tarea
- c) srun -p ac --node=1 ./HelloOMP
- d) srun -p ac ./HelloOMP

10Elección
única

¿Cuántos cores lógicos tiene cada microprocesador Intel Xeon Silver 4216?

- a) 12
- b) 64
- c) 32
- d) 16

He terminado

Información**DocumentaciónUGR****Community****Software libre Android****iOS**

¿Qué es SWAD? [ES]	Manual breve [ES]	Condiciones legales	Twitter	Source code	SWADroid	Google Play	iSWAD App Store
What is SWAD? [EN]	Brief manual [EN]	Protección de datos	Facebook	Download	SWADroid Blog	iSWAD	Twitter
Publicaciones	Guía usuario [ES]	Twitter	SWAD UGR	Wikipedia	Install	SWADroid Twitter	iSWAD GitHub
Funcionalidad	User guide [EN]	Estadísticas		Google+	Database	SWADroid Google+	
Difusión	Presentaciones	Póster		YouTube	Translation	SWADroid GitHub	
Prensa	Videotutoriales	Servidor		alternativeTo	API	SWADroid Open HUB	
		Logos	Encuentro	startupRANKING	Changelog		
				Capterra	Roadmap		
				SourceForge	Authors		
				GitHub	Implementación		
				Open HUB			



Universidad de Granada

Consultas y problemas: swad@ugr.es

Acera de SWAD 20.30 (2021-02-11) Página generada en 36 ms y enviada en 389 µs



31 de AC

3 profesores

	Mancia	1'19"
	Anguita...	
	Julio	2'23"
	Ortega ...	
	Juan J...	9'38"
	Escoba...	
28 estudiantes		
	Juan M...	41"
	Rodríg...	
	Salvador	43"
	Molina ...	
	Ricardo	45"
	López ...	
	Jose Luis	53"
	Rico R...	
	María d...	53"
	Izquier...	
	Eva	1'06"
	Rueda ...	
	Pablo	1'07"
	Fuente...	

...

Sistema Actividades Proyectos Convocatorias Test Exámenes Juegos Encuestas



Examen BP0



Universidad de Granada - [REDACTED]
Arquitectura de Computadores



Estudiante:



Examen de prácticas BP0

10 preguntas

Nota máxima 10

Examen BP0

10 preguntas

1 ¿Cuántos bits ocupa un double?

- Elección única
- a) 4
 - b) 64
 - c) 8
 - d) 32

2 ¿Cuál de las siguientes órdenes usaría para que se cree un único proceso con el código paralelo HelloOMP que use 12 hebras, de forma que cada hebra se ejecute en un núcleo físico distinto

- Elección única** de un nodo de cómputo de atcgrid[1-3]?
 a) srun -p ac --account=ac -n1 --cpus-per-task=8 HelloOMP
 b) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread HelloOMP
 c) srun -p ac --account=ac -n1 --cpus-per-task=24 HelloOMP
 d) srun -p ac --account=ac -n1 HelloOMP
- 3** ¿Qué opción añadiría a `srun` para que use solamente un núcleo lógico por cada núcleo físico que se asigne a la ejecución de su programa?
 a) --cpus-per-task=1
 b) --hint=nomultithread
 c) -n1
 d) --exclusive
- 4** ¿Cuántos zócalos (sockets) tiene el cluster atcgrid en total sin contar el host o front-end?
 a) 2
 b) 12
 c) 4
 d) 8
- 5** ¿Cuántos cores físicos tiene cada nodo de cómputo de atcgrid[1-3]?
 a) 6
 b) 12
 c) 24
 d) 2
- 6** ¿Qué devuelve la función `clock_gettime()` tal como se usa en el código `SumaVectores.c`?
 a) El tiempo transcurrido desde la última llamada a la misma función
 b) El tiempo transcurrido desde las 00:00 del 1 de enero de 1970 (EPOCH)
 c) El tiempo entre el instante en que se ejecuta y el instante que recibe como parámetro
 d) El tiempo transcurrido desde que comenzó la ejecución del programa
- 7** ¿Con qué comando SLURM se pueden ver todos los trabajos en ejecución y los que están encolados?
 a) `sjobs`
 b) `squeue`
 c) `snumjobs`
 d) `sinfo`
- 8** Indica los flags correctos para que `srun` ejecute el programa `HelloOMP` de forma que se use una sola hebra en cada uno de los cores físicos de un nodo de cómputo de atcgrid[1-3]
 a) -n1 --cpus-per-task=24
 b) -n1 --cpus-per-task=12 --hint=nomultithread
 c) -n1 --cpus-per-task=12
 d) -n12
- 9** Se ha usado la siguiente orden para ejecutar `script.sh` en un nodo de cómputo de atcgrid[1-3]:
`sbatch -p ac --account=ac -n1 --cpus-per-task=6 script.sh`
Teniendo en cuenta esta orden y el proceder por defecto de slurm en atcgrid, va a utilizar un máximo núcleos físicos de
 a) 24
 b) 6

a) 0

b) 12

c) 3

10

Elección
única

En un script para SLURM, ¿qué indica la línea `#SBATCH --job-name=helloOMP?`

a) El directorio donde está el ejecutable

b) El nombre asignado al trabajo

c) El usuario que manda el trabajo

d) La cola a la que se manda el trabajo

He terminado

Información DocumentaciUGR

Community Software libAndroid

iOS

¿Qué es SWAD? Manual breve [E]	Condiciones legal Twitter
What is SWAD? [I]	Brief manual [EN] Protección de datos Facebook
Publicaciones	Guía usuario [E] Twitter SWAD UGWikiPedia
Funcionalidad	User guide [EN] Estadísticas
Difusión	Presentaciones Póster
Prensa	Videotutoriales Servidor Logos Encuentro

Source code	SWADroid
Download	Google App Store
Install	SWADroid Blog
Database	iSWAD Twitter
Translation	SWADroid GitHub
API	SWADroid Google+
alternativeTo	SWADroid GitHub
startup	SWADroid Open HUB
RANKIN	
Changelog	
Capterra	
Roadmap	
SourceForge	
Authors	
GitHub	
Implementación	
Open HUB	



Universidad de Granada

Consultas y problemas: swad@ugr.es

Acerca de SWAD 20.30 (2021-02-11)

Página generada en 37 ms y enviada en 409 µs

Examen BP0

Universidad de Granada - Doble Grado en Ingeniería Informática y Matemáticas
Arquitectura de Computadores



Estudiante: 20100264 Luque Salguero, Pablo



Inicio: Hoy, miércoles, 09:50:10

Final: Hoy, miércoles, 09:56:43

Preguntas: 10

Respuestas
válidas:



Puntuación:



Nota:



1 Asumiendo que el programa HelloOMP está en el mismo directorio que el script que se usará para ejecutarlo en el cluster, ¿cuál es la opción correcta para indicar dentro de dicho script que se encole la ejecución de HelloOMP en un nodo de cómputo de atcgrid?

Elección única

Usuario Profesores

- a) sbatch HelloOMP
- b) HelloOMP
- c) ./HelloOMP
- d) srun ./HelloOMP

2 ¿Qué orden utilizo para conectarme al *front-end* de atcgrid para ejecutar comandos?

Elección única

Usuario Profesores

- a) sbatch -p ac script.sh
- b) telnet usuario@ugr.es
- c) ssh usuario@atcgrid.ugr.es
- d) sftp usuario@atcgrid.ugr.es

3 Si jobid indica el identificador de un trabajo en la cola de SLURM y jobname su nombre, indica cómo eliminar dicho trabajo de la cola

Elección única

Usuario Profesores

- a) sdelete jobname
- b) scancel jobid
- c) sdelete jobid
- d) scancel jobname

4 Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid[1-3]:

Elección única

sbatch -p ac --account=ac -n1 --cpus-per-task=1 -hint=nomultithread script.sh

Teniendo en cuenta esta orden y el proceder por defecto de la instalación slurm de atcgrid, va a utilizar un máximo de núcleos lógicos de

Usuario Profesores

- a) Ninguna otra respuesta es correcta.
- b) 3
- c) 2
- d) 1

5 ¿Qué indica la variable de entorno SLURM_JOB_NUM_NODES del sistema de gestión de colas de atcgrid?

Elección única

Usuario Profesores

- a) Especifica un nombre para el trabajo
- b) El número de nodos asignados a un trabajo
- c) La lista de nodos asignados al trabajo
- d) Asigna la cola del trabajo

6 Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid[1-3]:

Elección única

sbatch -p ac --account=ac -n1 --cpus-per-task=1 script.sh

Teniendo en cuenta esta orden y el proceder por defecto de la instalación slurm de atcgrid, va a utilizar un máximo de núcleos lógicos de

Usuario Profesores

- a) 1
- b) 12
- c) 2
- d) 6

7 ¿Con qué comando SLURM se pueden ver todos los trabajos en ejecución y los que están encolados?

Elección única Usuario Profesores

- a) sjobs
- b) squeue
- c) snumjobs
- d) sinfo

8 ¿Cuál de las siguientes órdenes cree que es más apropiada usar para que su código paralelo OpenMP esté en las mejores condiciones de conseguir un menor tiempo de ejecución?

Elección única Usuario Profesores

- a) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread executableOpenMP
- b) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread --exclusive executableOpenMP
- c) srun -p ac --account=ac -n1 --hint=nomultithread --exclusive executableOpenMP
- d) srun -p ac --account=ac -n1 --cpus-per-task=12 --exclusive executableOpenMP

9 ¿Qué opción añadiría a srun para que use solamente un núcleo lógico por cada núcleo físico que se asigne a la ejecución de su programa?

Elección única Usuario Profesores

- a) --hint=nomultithread
- b) --cpus-per-task=1
- c) -n1
- d) --exclusive

10 En un script para SLURM, ¿qué contiene la variable \$SLURM_JOBID?

Elección única Usuario Profesores

- a) El identificador del trabajo en la cola
- b) El nombre del usuario que ha mandado el trabajo a la cola
- c) El identificador del usuario que ha mandado el trabajo a la cola
- d) El nombre del trabajo en la cola



31 de AC

3 profesores



Mancia	1'45"
Anguita...	
Julio	2'49"
Ortega ...	
Juan J...	10'04"
Escoba...	



28 estudiantes



Ricardo	1'11"
López ...	
María d...	1'19"
Izquier...	
Jose Luis	1'19"
Rico R...	
Salvador	1'23"
Molina ...	
Eva	1'32"
Rueda ...	
Pablo	1'33"
Fuente...	
Javier	1'35"
Garrue...	



- d) La cola a la que se manda el trabajo
- 2** ¿Qué indica la variable de entorno SLURM_JOB_NUM_NODES del sistema de gestión de colas de atcgrid?
 a) La lista de nodos asignados al trabajo
 b) Especifica un nombre para el trabajo
 c) Asigna la cola del trabajo
 d) El número de nodos asignados a un trabajo
- 3** ¿Cuántos cores lógicos tiene cada microprocesador Intel Xeon E5645?
 a) 12
 b) 2
 c) 3
 d) 6
- 4** Indica los flags correctos para que srun ejecute el programa HelloOMP de forma que se use una sola hebra en cada uno de los cores físicos de un nodo de cómputo de atcgrid[1-3]
 a) -n12
 b) -n1 --cpus-per-task=24
 c) -n1 --cpus-per-task=12
 d) -n1 --cpus-per-task=12 --hint=nomultithread
- 5** ¿Cuántos cores lógicos tiene cada nodo de cómputo de atcgrid[1-3]?
 a) 6
 b) 12
 c) 3
 d) 24
- 6** ¿Qué opción añadiría a srun para que use solamente un núcleo lógico por cada núcleo físico que se asigne a la ejecución de su programa?
 a) --hint=nomultithread
 b) --cpus-per-task=1
 c) --exclusive
 d) -n1
- 7** ¿Qué orden puedo utilizar para conectarme al *front-end* de atcgrid para enviar/recibir ficheros?
 a) ssh usuario@atcgrid.ugr.es
 b) sftp usuario@atcgrid.ugr.es
 c) sbatch -p ac script.sh
 d) telnet usuario@ugr.es
- 8** ¿Con qué orden podemos compilar el código fuente de HelloOMP.c?
 a) gcc -fopenmp HelloOMP.c -o HelloOMP
 b) Ninguna otra respuesta es correcta.
 c) gcc -fopenmp -o HelloOMP
 d) gcc HelloOMP.c -o HelloOMP
- 9** ¿Cuál es el número de microprocesadores que tiene cada nodo de cómputo de

 Elección
única

- atcgrid?
 a) 8
 b) 1
 c) 4
 d) 2

10

Elección
única

¿Cuántos cores lógicos tiene cada microprocesador Intel Xeon Silver 4216?

- a) 32
 b) 64
 c) 16
 d) 12

He terminado

Información Documental UGR

Community Software

iOS

¿Qué es SWADManual breve |Condiciones legTwitter

Source code SWADroid GoogliSWAD App St

What is SWAD?Brief manual [EProtección de dFacebook

Download SWADroid Blog iSWAD Twitter

Publicaciones Guía usuario [ITwitter SWAD LWikipedia

Install SWADroid TwitteiSWAD GitHub

Funcionalidad User guide [ENEstadísticas

Google+ Database

SWADroid Goog

Difusión PresentacionePóster

YouTube Translation

SWADroid GitHub

Prensa VideotutorialesServidor

alternativeTo API

SWADroid Open HUB

Logos Encuentro

startupRANKIChangelog

Capterra Roadmap

SourceForge Authors

GitHub Implementación

Open HUB



Universidad de Granada

Consultas y problemas: swad@ugr.es

Acerca de SWAD 20.30 (2021-02-11) Página generada en 36 ms y enviada en 400 µs



Examen BP0 Grupo 1



Universidad de Granada - Doble Grado en Ingeniería Informática y Matemáticas
Arquitectura de Computadores



Estudiante: 77558355 Martínez Sáez, Néstor



Inicio: Hoy, jueves, 08:40:03

Final: Hoy, jueves, 08:44:59

Preguntas: 10

Respuestas válidas:



Puntuación:



Nota:



1 ¿Cuál de las siguientes órdenes usaría para que se cree un único proceso con el código paralelo HelloOMP que use 32 hebras, de forma que cada hebra se ejecute en un núcleo físico distinto del nodo de cómputo atcgrid4?

Elección única Usuario Profesores

- a) srun -p ac --account=ac -n1 --cpus-per-task=32 --hint=nomultithread HelloOMP
- b) srun -p ac4 --account=ac -n1 --cpus-per-task=16 HelloOMP
- c) srun -p ac4 --account=ac -n1 --cpus-per-task=32 HelloOMP
- d) srun -p ac4 --account=ac -n1 --cpus-per-task=32 --hint=nomultithread HelloOMP

2 Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid[1-3]:

sbatch -p ac --account=ac -n1 --cpus-per-task=6 script.sh

Teniendo en cuenta esta orden y el proceder por defecto de slurm en atcgrid, va a utilizar un máximo núcleos físicos de

Usuario Profesores

- a) 3
- b) 12
- c) 6
- d) 24

3 Si jobid indica el identificador de un trabajo en la cola de SLURM y jobname su nombre, indica cómo eliminar dicho trabajo de la cola

Elección única Usuario Profesores

- a) scancel jobname
- b) sdelete jobid
- c) sdelete jobname
- d) scancel jobid

4 ¿Qué opción de sbatch evita que haya otros trabajos ejecutándose simultáneamente en el mismo nodo al que se envía su trabajo?

Elección única Usuario Profesores

- a) Ninguna otra respuesta es correcta
- b) -p ac -n1
- c) --hint=nomultithread
- d) --exclusive

5 En un script para SLURM, ¿qué indica la línea #SBATCH --job-name=helloOMP?

Elección única Usuario Profesores

- a) La cola a la que se manda el trabajo
- b) El usuario que manda el trabajo
- c) El nombre asignado al trabajo
- d) El directorio donde está el ejecutable

6 En prácticas se recomienda usar un script para ejecutar los trabajos en un nodo de cómputo de atcgrid, además se recomienda utilizar una ejecución en segundo plano. Teniendo en cuenta estas recomendaciones, ¿qué usuario

Elección única

Se recomienda utilizar una ejecución en *segundo plano*. Teniendo en cuenta estas recomendaciones, ¿qué usaria para enviar a ejecutar pmvOpenMP a un nodo de cómputo de atcgrid[1-3] utilizando una hebra por cada núcleo físico? (script.sh ejecuta pmvOpenMP):

Usuario Profesores

- a) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread script.sh
- b) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread pmvOpenMP
- c) sbatch -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread script.sh
- d) sbatch -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread pmvOpenMP

7

¿Por qué a partir de un determinado número de componentes en los vectores locales se obtiene un error de fallo de segmentación?

Elección única

Usuario Profesores

- a) Porque los nodos atcgrid[1-3] no tienen suficiente memoria y la tarea debió mandarse a atcgrid4
- b) El código fuente no ha sido compilado con la opción `--mem unlimited`
- c) No se obtiene ningún error durante la ejecución
- d) Porque se ha superado el tamaño máximo permitido del stack

8

Elección única

Al compilar un programa se han obtenido mensajes del compilador y no se ha generado el ejecutable debido al tamaño de las variables que utiliza el programa. Como el error es al compilar sabemos que el problema lo causan variables:

Usuario Profesores

- a) dinámicas
- b) ninguna otra respuesta es correcta
- c) globales
- d) locales

9

Elección única

Se ha usado la siguiente orden para ejecutar script.sh en un nodo de cómputo de atcgrid[1-3]:

```
sbatch -p ac --account=ac -n1 --cpus-per-task=1 -hint=nomultithread script.sh
```

Teniendo en cuenta esta orden y el proceder por defecto de la instalación slurm de atcgrid, va a utilizar un máximo de núcleos lógicos de

Usuario Profesores

- a) 2
- b) 3
- c) Ninguna otra respuesta es correcta.
- d) 1

10

Elección única

¿En qué zona de la memoria se alojan los vectores locales de un programa en C?

Usuario Profesores

- a) stack
- b) Ninguna de las anteriores
- c) data
- d) heap



Examen BP0 Grupo 1

Universidad de Granada - Doble Grado en Ingeniería Informática y Matemáticas
Arquitectura de Computadores



Estudiante: 77393052 Rosado García, Miguel



Inicio: Hoy, jueves, 08:40:09

Final: Hoy, jueves, 08:47:43

Preguntas: 10

Respuestas
válidas:



Puntuación:



Nota:



1 Indica los flags correctos para que srun ejecute el programa HelloOMP de forma que se use una sola hebra en cada uno de los cores físicos de un nodo de cómputo de atcgrid[1-3]

Elección única

- Usuario Profesores
- a) -n1 --cpus-per-task=12 --hint=nomultithread
 - b) -n1 --cpus-per-task=12
 - c) -n1 --cpus-per-task=24
 - d) -n12

2 Añadir la opción -n1 a srun permite asegurar

Elección única

- Usuario Profesores
- a) que el programa que va a ejecutar usa solamente un núcleo físico de uno de los nodos de cómputo de atcgrid
 - b) que el programa que va a ejecutar usa solamente un nodo de cómputo de atcgrid
 - c) que el programa que va a ejecutar usa solamente un núcleo lógico de uno de los nodos de cómputo de atcgrid
 - d) que se crea solamente un proceso del programa que va a ejecutar

3 ¿Con qué precisión devuelve el tiempo clock_gettime()?

Elección única

- Usuario Profesores
- a) nanosegundos
 - b) segundos
 - c) milisegundos
 - d) microsegundos

4 ¿Qué orden puedo utilizar para conectarme al front-end de atcgrid para enviar/recibir ficheros?

Elección única

- Usuario Profesores
- a) telnet usuario@ugr.es
 - b) ssh usuario@atcgrid.ugr.es
 - c) sbatch -p ac script.sh
 - d) sftp usuario@atcgrid.ugr.es

5 ¿Con qué comando SLURM se pueden ver todos los trabajos en ejecución y los que están encolados?

Elección única

- Usuario Profesores
- a) squeue
 - b) sinfo
 - c) snumjobs
 - d) sjobs

6 ¿Cuál de las siguientes órdenes usaría para que se cree un único proceso con el código paralelo HelloOMP que use 32 hebras, de forma que cada hebra se ejecute en un núcleo físico distinto del nodo de cómputo atcgrid4?

Elección única

Usuario Profesores

- a) srun -p ac4 --account=ac -n1 --cpus-per-task=32 HelloOMP
- b) srun -p ac --account=ac -n1 --cpus-per-task=32 --hint=nomultithread HelloOMP
- c) srun -p ac4 --account=ac -n1 --cpus-per-task=32 --hint=nomultithread HelloOMP
- d) srun -p ac4 --account=ac -n1 --cpus-per-task=16 HelloOMP

7 Asumiendo que el programa HelloOMP está en el mismo directorio que el script que se usará para ejecutarlo en el cluster, ¿cuál es la opción correcta para indicar dentro de dicho script que se encole la ejecución de HelloOMP en un nodo de cómputo de atcgrid?

Elección única

Usuario Profesores

- a) HelloOMP
- b) sbatch HelloOMP
- c) srun ./HelloOMP
- d) ./HelloOMP

8 ¿Por qué a partir de un determinado número de componentes en los vectores locales se obtiene un error de fallo de segmentación?

Elección única

Usuario Profesores

- a) Porque se ha superado el tamaño máximo permitido del stack
- b) El código fuente no ha sido compilado con la opción --mem unlimited
- c) Porque los nodos atcgrid[1-3] no tienen suficiente memoria y la tarea debió mandarse a atcgrid4
- d) No se obtiene ningún error durante la ejecución

9 ¿Qué opción añadiría a srun para que use solamente un núcleo lógico por cada núcleo físico que se asigne a la ejecución de su programa?

Elección única

Usuario Profesores

- a) --exclusive
- b) --hint=nomultithread
- c) --cpus-per-task=1
- d) -n1

10 ¿Cuál de las siguientes órdenes usaría para que se cree un único proceso con el código paralelo HelloOMP que use 12 hebras, de forma que cada hebra se ejecute en un núcleo físico distinto de un nodo de cómputo de atcgrid[1-3]?

Elección única

Usuario Profesores

- a) srun -p ac --account=ac -n1 --cpus-per-task=24 HelloOMP
- b) srun -p ac --account=ac -n1 --cpus-per-task=12 --hint=nomultithread HelloOMP
- c) srun -p ac --account=ac -n1 HelloOMP
- d) srun -p ac --account=ac -n1 --cpus-per-task=8 HelloOMP



Examen BP0

Universidad de Granada - Doble Grado en Ingeniería Informática y Matemáticas
Arquitectura de Computadores



Estudiante: 75925602

75925602A Luque Framit, Rafael



Inicio: Hoy, viernes, 09:40:05

Final: Hoy, viernes, 09:48:35

Preguntas: 10

Respuestas
válidas:



Puntuación:



Nota:



1

En un script para SLURM, ¿qué indica la línea `#SBATCH --job-name=HelloOMP`?

Elección única

- a) El usuario que manda el trabajo
- b) La cola a la que se manda el trabajo
- c) El nombre asignado al trabajo
- d) El directorio donde está el ejecutable

2

Si `jobid` indica el identificador de un trabajo en la cola de SLURM y `jobname` su nombre, indica cómo eliminar dicho trabajo de la cola

Elección única

- a) `sdelete jobname`
- b) `scancel jobname`
- c) `sdelete jobid`
- d) `scancel jobid`

3

¿Cuántos cores lógicos tiene el nodo de cómputo atcgrid4?

Elección única

- a) 64
- b) 16
- c) 32
- d) 12

4

En un script para SLURM, ¿qué contiene la variable `$SLURM_JOBID`?

Elección única

- a) El nombre del usuario que ha mandado el trabajo a la cola
- b) El nombre del trabajo en la cola
- c) El identificador del usuario que ha mandado el trabajo a la cola
- d) El identificador del trabajo en la cola

5

Asumiendo que el programa `HelloOMP` está en el mismo directorio que el script que se usará para ejecutarlo en el cluster, ¿cuál es la opción correcta para indicar dentro de dicho script que se encole la ejecución de `HelloOMP` en un nodo de cómputo de atcgrid?

Elección única

- a) `srun ./HelloOMP`
- b) `./HelloOMP`
- c) `HelloOMP`
- d) `sbatch HelloOMP`

6

¿Cuántos cores lógicos tiene cada microprocesador Intel Xeon Silver 4216?

- Elección única** Usuario Profesores
- a) 16
 - b) 32
 - c) 64
 - d) 12
- 7** ¿Por qué a partir de un determinado número de componentes en los vectores locales se obtiene un error de fallo de segmentación?
- Elección única** Usuario Profesores
- a) El código fuente no ha sido compilado con la opción `--mem unlimited`
 - b) Porque se ha superado el tamaño máximo permitido del stack
 - c) Porque los nodos atcgrid[1-3] no tienen suficiente memoria y la tarea debió mandarse a atcgrid4
 - d) No se obtiene ningún error durante la ejecución
- 8** ¿Cuántos bits ocupa un double?
- Elección única** Usuario Profesores
- a) 32
 - b) 4
 - c) 8
 - d) 64
- 9** ¿Cuál de las siguientes órdenes usaría para que se cree un único proceso con el código paralelo HelloOMP que use 32 hebras, de forma que cada hebra se ejecute en un núcleo físico distinto del nodo de cómputo atcgrid4?
- Elección única** Usuario Profesores
- a) `srun -p ac4 --account=ac -n1 --cpus-per-task=16 HelloOMP`
 - b) `srun -p ac4 --account=ac -n1 --cpus-per-task=32 HelloOMP`
 - c) `srun -p ac --account=ac -n1 --cpus-per-task=32 --hint=nomultithread HelloOMP`
 - d) `srun -p ac4 --account=ac -n1 --cpus-per-task=32 --hint=nomultithread HelloOMP`
- 10** ¿Con qué precisión devuelve el tiempo `clock_gettime()`?
- Elección única** Usuario Profesores
- a) segundos
 - b) nanosegundos
 - c) milisegundos
 - d) microsegundos

ARQUITECTURA DE COMPUTADORES. BENCHMARK del TEMA 1.

Apellidos y nombre:

- Escriba la expresión de la ley de Amdahl en términos de p (ganancia de velocidad del recurso que se ha mejorado) y de f (fracción del tiempo de procesamiento en el computador base durante el que NO se puede aprovechar la mejora):

$$S \leq p / (1 + f \times (p - 1))$$

- Según la ley de Amdahl, la máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso es $1/f$ (f definido como en la pregunta 1)

(V)

- Escriba la expresión del tiempo de CPU (T_{CPU}) en términos del número de instrucciones ejecutadas (NI), el número medio de ciclos por instrucción (CPI) y la frecuencia de reloj (F)

$$T_{CPU} = NI \times CPI / F$$

- ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta cuatro instrucciones por ciclo y funciona a una frecuencia de reloj de 3 GHz?

$$MIPS = 4 \text{ int/ciclo} \times 3 \times 10^9 \text{ ciclos/s} \times (1/10^6) = 12000$$

- La comunicación entre procesadores en un computador UMA se realiza a través de escrituras y lecturas en la memoria compartida, igual que en un computador NUMA

(V)

- Un procesador puede terminar hasta 4 operaciones en coma flotante por ciclo. ¿Cuál es su velocidad pico (en GFLOPS) si funciona a una frecuencia de reloj de 2 GHz?

$$GFLOPS = 4 \text{ op_float/ciclo} \times 2 \times 10^9 \text{ ciclos/s} \times (1/10^9) = 8$$

- El bucle $for i=1 \text{ to } N \text{ do } a(i)=c \times (a(i)+b(i));$ con $N=10^{14}$, se ejecuta en 10 segundos, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?

$$GFLOPS = (2 \times 10^{14} \text{ op_float}) / (10 \text{ s} \times 10^9) = 2 \times 10^4 = 20,000$$

- En la secuencia de instrucciones:

- (a) add r1, r2, r3 ; r1 \leftarrow r2 + r3
- (b) sub r1, r2, r4 ; r1 \leftarrow r2 - r4
- (c) add r3, r2, r1 ; r3 \leftarrow r2 + r1

- El registro r1 solo genera una dependencia RAW

(F)

- No hay dependencias debido al uso del registro r2

(V)

- El registro r3 genera una dependencia WAW

(F)

ARQUITECTURA DE COMPUTADORES. BENCHMARK del TEMA 1.

Apellidos y nombre:

- Escriba la expresión de la ley de Amdahl en términos de p (ganancia de velocidad del recurso que se ha mejorado) y de f (fracción del tiempo de procesamiento en el computador base durante el que NO se puede aprovechar la mejora del recurso):

$$S \leq p / (1 + f \times (p - 1))$$

- Según la ley de Amdahl, la máxima ganancia de velocidad que se puede conseguir, por mucho que se mejore el recurso considerado es $1/f$ (f definido como en la pregunta 1)

(V)

- Escriba la expresión del tiempo de CPU (T_{CPU}) en términos del número de instrucciones ejecutadas (NI), el número medio de instrucciones por ciclo (IPC) y el tiempo de ciclo del reloj (T_{ciclo})

$$T_{CPU} = NI \times T_{ciclo} / IPC$$

- ¿Cuál es la velocidad pico en MIPS de un procesador que puede terminar hasta tres instrucciones por ciclo y funciona a una frecuencia de reloj de 2 GHz?

$$MIPS = 3 \text{ int/ciclo} \times 2 \times 10^9 \text{ ciclos/s} \times (1/10^6) = 6000$$

- En un computador NUMA la memoria principal está físicamente distribuida, igual que en un computador NORMA

(V)

- Un procesador puede terminar hasta 2 operaciones en coma flotante por ciclo. ¿Cuál es su velocidad pico (en GFLOPS) si funciona a una frecuencia de reloj de 2.5 GHz?

$$GFLOPS = 2 \text{ op_float/ciclo} \times 2.5 \times 10^9 \text{ ciclos/s} \times (1/10^9) = 5$$

- El bucle $for i=1 to N do a(i)=cx(a(i)+b(i));$ con $N=10^{16}$, se ejecuta en 10 segundos, siendo c , $a()$, y $b()$ datos en coma flotante. ¿Cuántos GFLOPS alcanza la máquina al ejecutar el código?

$$GFLOPS = (2 \times 10^{16} \text{ op_float}) / (10 \text{ s} \times 10^9) = 2 \times 10^6 = 2,000,000$$

- En la secuencia de instrucciones:

- (a) add r1, r1, r2 ; r1 \leftarrow r1 + r2
- (b) sub r2, r2, r1 ; r2 \leftarrow r2 - r1
- (c) add r3, r3, r1 ; r3 \leftarrow r3 + r1

- El registro r1 solo genera dependencias RAW

(V)

- No hay dependencias debido al uso del registro r3

(V)

- El registro r2 genera una dependencia WAW

(F)

5. ¿Qué paradigma de programación paralela implementa OpenMP?

- a) Paso de mensajes mediante procesos
- b) Memoria compartida mediante hebras**
- c) SIMD
- d) Memoria compartida mediante procesos

6. La API de programación de OpenMP está formada por...

- a) Directivas del compilador funciones y variables de entorno**
- b) Variables del compilador y funciones para medición de tiempo en programas con varias hebras
- c) Ninguna de las respuestas es correcta
- d) Variables de entorno definidas en consola y funciones de paso de mensajes

7. Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
```

```
{  
    sumalocal=0;  
    #pragma omp for  
    for (i=0; i<n; i++)  
        sumalocal += a[i];  
    #pragma omp barrier  
    #pragma omp critical  
        suma = suma + sumalocal;  
    #pragma omp barrier  
    #pragma omp single  
        printf("La suma es=%d\n", suma);  
}
```

- a) El valor de suma que se imprime es correcto**
- b) Todas las demás respuestas son incorrectas
- c) Tendríamos el mismo comportamiento si eliminamos los dos #pragma omp barrier
- d) Tendríamos el mismo comportamiento si cambiamos critical por atomic

8. Cuando ejecutamos el código que aparece a continuación, obtenemos la salida

Ejecutada por: 0. b[3]=1. ¿qué afirmación es correcta?

```
a=0;  
for (i=0; i<10;i++) b[i]=-1;  
#pragma omp parallel{  
    #pragma omp single{  
        a=1;  
        printf("Ejecutada por: %d.", omp_get_thread_num());  
    }  
    #pragma omp for  
    for (i=0; i<10;i++) b[i]=a;  
}  
printf("b[3]= %d.", b[3]);
```

- a) Todas las respuestas son correctas
- b) Podríamos obtener la misma salida si cambiamos single por master
- c) La hebra master ejecutó la instrucción a=1;
- d) Podríamos obtener la salida Ejecutada por: 0. b[3]=0. si cambiamos single por master

9. ¿Cuántas hebras ejecutarán una directiva sections con 4 secciones (section) en una plataforma con 8 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 3? (considere que no se usan en el código funciones OpenMP)

- a) 1
- b) Ninguna de las otras respuestas es correcta
- c) 3
- d) 2

10. ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=3?

```
#pragam omp parallel
{
    #pragma omp single
    {
        printf("x");
    }
}
```

- a) x
- b) xx
- c) xxx
- d) Indeterminado porque existe conducción de carrera

11. ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=3?

```
#pragma omp parallel
{
    #pragma omp critical
    {
        printf("x");
    }
}
```

- a) XXXX
- b) X
- c) XX
- d) XXX

6. Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
{
    sumalocal=0;
    #pragma omp for
    for (i=0; i<n; i++)
        sumalocal += a[i];
    #pragma omp barrier      → ESTE SOBRA
    #pragma omp critical
        suma = suma + sumalocal;
    #pragma omp barrier
    #pragma omp single
        printf("La suma es=%d\n", suma);
}
```

- a) El valor de suma que se imprime sería correcto si cambiamos private(sumalocal) por private(sumalocal, suma)
- b) Uno de los #pragma omp barrier es innecesario
- c) Todas las demás respuestas son incorrectas
- d) El valor de suma que se imprime no es siempre correcto

7. Cuando se mide el tiempo de ejecución de un programa mediante la orden del sistema time, la suma de los tiempos de usuario y sistema

- a) Para programas secuenciales, es siempre menor o igual que el tiempo de ejecución
- b) Para programas paralelos, es siempre menor o igual que el tiempo de ejecución
- c) Es siempre igual que el tiempo de ejecución
- d) Es siempre menor o igual que el tiempo de ejecución

5. ¿Qué paradigma de programación paralela implementa OpenMP?

- a) Paso de mensajes mediante procesos
- b) Memoria compartida mediante hebras**
- c) SIMD
- d) Memoria compartida mediante procesos

6. La API de programación de OpenMP está formada por...

- a) Directivas del compilador funciones y variables de entorno**
- b) Variables del compilador y funciones para medición de tiempo en programas con varias hebras
- c) Ninguna de las respuestas es correcta
- d) Variables de entorno definidas en consola y funciones de paso de mensajes

7. Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
```

```
{  
    sumalocal=0;  
    #pragma omp for  
    for (i=0; i<n; i++)  
        sumalocal += a[i];  
    #pragma omp barrier  
    #pragma omp critical  
        suma = suma + sumalocal;  
    #pragma omp barrier  
    #pragma omp single  
        printf("La suma es=%d\n", suma);  
}
```

- a) El valor de suma que se imprime es correcto**
- b) Todas las demás respuestas son incorrectas
- c) Tendríamos el mismo comportamiento si eliminamos los dos #pragma omp barrier
- d) Tendríamos el mismo comportamiento si cambiamos critical por atomic

8. Cuando ejecutamos el código que aparece a continuación, obtenemos la salida

Ejecutada por: 0. b[3]=1. ¿qué afirmación es correcta?

```
a=0;  
for (i=0; i<10;i++) b[i]=-1;  
#pragma omp parallel{  
    #pragma omp single{  
        a=1;  
        printf("Ejecutada por: %d.", omp_get_thread_num());  
    }  
    #pragma omp for  
    for (i=0; i<10;i++) b[i]=a;  
}  
printf("b[3]= %d.", b[3]);
```

- a) Todas las respuestas son correctas
- b) Podríamos obtener la misma salida si cambiamos single por master
- c) La hebra master ejecutó la instrucción a=1;
- d) Podríamos obtener la salida Ejecutada por: 0. b[3]=0. si cambiamos single por master

9. ¿Cuántas hebras ejecutarán una directiva sections con 4 secciones (section) en una plataforma con 8 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 3? (considere que no se usan en el código funciones OpenMP)

- a) 1
- b) Ninguna de las otras respuestas es correcta
- c) 3
- d) 2

10. ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=3?

```
#pragam omp parallel
{
    #pragama omp single
    {
        printf("x");
    }
}
```

- a) x
- b) xx
- c) xxx
- d) Indeterminado porque existe conducción de carrera

11. ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=3?

```
#pragma omp parallel
{
    #pragma omp critical
    {
        printf("x");
    }
}
```

- a) XXXX
- b) X
- c) XX
- d) XXX

6. Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
{
    sumalocal=0;
    #pragma omp for
    for (i=0; i<n; i++)
        sumalocal += a[i];
    #pragma omp barrier      → ESTE SOBRA
    #pragma omp critical
        suma = suma + sumalocal;
    #pragma omp barrier
    #pragma omp single
        printf("La suma es=%d\n", suma);
}
```

- a) El valor de suma que se imprime sería correcto si cambiamos private(sumalocal) por private(sumalocal, suma)
- b) Uno de los #pragma omp barrier es innecesario**
- c) Todas las demás respuestas son incorrectas
- d) El valor de suma que se imprime no es siempre correcto

7. Cuando se mide el tiempo de ejecución de un programa mediante la orden del sistema time, la suma de los tiempos de usuario y sistema

- a) Para programas secuenciales, es siempre menor o igual que el tiempo de ejecución**
- b) Para programas paralelos, es siempre menor o igual que el tiempo de ejecución
- c) Es siempre igual que el tiempo de ejecución
- d) Es siempre menor o igual que el tiempo de ejecución

3. ¿Para qué sirve la directiva barrier?

- a) Para fijar en el código un punto de sincronización de todas las hebras**
- b) Para evitar las condiciones de carrera
- c) Para proteger el acceso a una variable compartida
- d) Para que todas las hebras terminen su ejecución en ese punto

4. ¿Cuáles de las siguientes directivas no incorpora una barrera implícita al final?

- a) atomic**
- b) parallel
- c) for
- d) sections

5. ¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva critical en una plataforma con 3 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2?

- a) 3
- b) 1
- c) 2**
- d) 4



Examen BP1



Universidad de Granada - Doble Grado en Ingeniería Informática y
Matemáticas
Arquitectura de Computadores



Estudiante: 79112789 Tirado Guzmán, Miguel



Inicio: Hoy, miércoles, 09:40:04

Final: Hoy, miércoles, 09:54:16

Preguntas: 12

Respuestas

válidas:

Puntuación:

Nota:

1 ¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva `critical` en una plataforma con 4 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 2?

Elección única Usuario Profesores

- a) 3
- b) 4
- c) 2
- d) 1

2 ¿Qué parámetro hace que gcc genere código ensamblador?

Elección única Usuario Profesores

- a) -O
- b) -S
- c) -asm
- d) -lrt

3 ¿Qué resultado muestra por pantalla la ejecución del siguiente código suponiendo que `OMP_NUM_THREADS=4`?

Elección única

```
int main () {
    # pragma omp parallel
    cout << 'x' ;
```

}

Usuario Profesores

- a) xxx
- b) x
- c) xx
- d) xxxx

4

¿Qué componentes define la API de programación de OpenMP?

Elección única

Usuario Profesores

- a) Ninguna de las respuestas anteriores es correcta
- b) Variables de entorno y funciones de paso de mensajes
- c) Directivas del compilador, funciones y variables de entorno
- d) Directivas del compilador y variables de sincronización entre hebras

5

Cuando ejecutamos el código que aparece a continuación, obtenemos la salida Ejecutada por: 0. b[3]=1., ¿qué afirmación es correcta?

Elección única

a=0;

```
for (i=0; i<10;i++) b[i]=-1;
#pragma omp parallel{
    #pragma omp single{
        a=1;
        printf("Ejecutada por: %d.", omp_get_thread_num());
    }
    #pragma omp for
    for (i=0; i<10;i++) b[i]=a;
}
printf("b[3]= %d.", b[3]);
```

Usuario Profesores

- a) Todas las respuestas son correctas
- b) La hebra master ejecutó la instrucción a=1;
- c) Podríamos obtener la salida Ejecutada por: 0. b[3]=0. si cambiamos single por master
- d) Podríamos obtener la misma salida si cambiamos single por master

6

¿Qué directiva usarías para que las hebras ejecuten el siguiente código en exclusión mutua?

Elección única

```
{b -= 2; }
```

Usuario Profesores

- a) Solo atomic
- b) Solo critical
- c) critical o atomic
- d) exclusive

7

¿Qué paradigma de programación paralela implementa OpenMP?

Elección única

Usuario Profesores

- a) Memoria compartida mediante hebras
- b) SIMD

8
Elección única

¿Cuántas hebras ejecutará una directiva `single` en una plataforma con 4 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 3?

Usuario Profesores

- a) 3
- b) 4
- c) 2
- d) 1

9
Elección única

¿Cuándo existe una condición de carrera?

Usuario Profesores

- a) Cuando el resultado de un programa paralelo depende del orden de ejecución de sus hebras
- b) Siempre que diferentes hebras comparten una variable
- c) Solamente si se garantiza el acceso en exclusión mutua a un recurso compartido
- d) Siempre que haya más de una hebra en ejecución

10
Elección única

La API de programación de OpenMP está formada por ...

Usuario Profesores

- a) Ninguna de las otras respuestas es correcta
- b) Variables de entorno definidas en consola y funciones de paso de mensajes
- c) Directivas del compilador, funciones y variables de entorno
- d) Variables del compilador y funciones para medición de tiempo en programas con varias hebras

11
Elección única

¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que `OMP_NUM_THREADS=4`?

`#pragma omp parallel`

`printf("x");`

Usuario Profesores

- a) xxxx
- b) xxx
- c) x
- d) Ninguna otra respuesta es correcta

12
Elección única

¿Qué identificador tiene la hebra máster en la ejecución de un programa paralelo?

Usuario Profesores

- a) El valor que defina el usuario mediante la función OpenMP `omp_set_master_num()`
- b) El 0
- c) Un valor numérico arbitrario, dependiente de los identificadores que se usaron en la ejecución anterior
- d) Un valor numérico arbitrario, dependiente del entorno de ejecución



Un valor numérico decimal, dependiente de la ejecución



Examen BP1



Universidad de Granada - Doble Grado en Ingeniería Informática y
Matemáticas
Arquitectura de Computadores



Estudiante: 20080849
20080849D Rico Ramos, Jose Luis



Inicio: Hoy, miércoles, 09:40:03

Final: Hoy, miércoles, 09:53:50

Preguntas: 12

Respuestas
válidas:

Puntuación:

Nota:

1 Elección única El número de MIPS de un programa...

Usuario Profesores

- a) siempre será mayor o igual que su número de MFLOPS
- b) será mayor que su número de MFLOPS sólo si el programa usa datos en coma flotante
- c) puede ser menor que su número de MFLOPS si el programa es de cálculo intensivo
- d) no guarda ninguna relación con su número de MFLOPS

2 Elección única Cuando ejecutamos el código que aparece a continuación, obtenemos la salida Ejecutada por: 0. b[3]=1.., ¿qué afirmación es correcta?

```
a=0;  
for (i=0; i<10;i++) b[i]=-1;  
#pragma omp parallel{  
    #pragma omp single{  
        a=1;  
        printf("Ejecutada por: %d.", omp_get_thread_num());  
    }  
    #pragma omp for  
    for (i=0; i<10;i++) b[i]=a;  
}  
printf("b[3]= %d.", b[3]);
```

Usuario Profesores

- a) Todas las respuestas son correctas
- b) Podríamos obtener la salida Ejecutada por: 0. b[3]=0.

		<p>si cambiamos <code>single</code> por <code>master</code></p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> c) Podríamos obtener la misma salida si cambiamos <code>single</code> por <code>master</code> <input checked="" type="checkbox"/> d) La hebra master ejecutó la instrucción <code>a=1</code>;
3 Elección única	¿Qué directiva OpenMP se encarga de crear hebras para ejecutar en paralelo su bloque estructurado?	Usuario Profesores
		<ul style="list-style-type: none"> <input checked="" type="checkbox"/> a) <code>atomic</code> <input checked="" type="checkbox"/> b) <code>parallel</code> <input checked="" type="checkbox"/> c) <code>for</code> <input checked="" type="checkbox"/> d) <code>section</code>
4 Elección única	¿Qué directiva es más adecuada para ejecutar en paralelo un bloque estructurado?	Usuario Profesores
		<ul style="list-style-type: none"> <input checked="" type="checkbox"/> a) <code>atomic</code> <input checked="" type="checkbox"/> b) <code>section</code> <input checked="" type="checkbox"/> c) <code>for</code> <input checked="" type="checkbox"/> d) <code>parallel</code>
5 Elección única	¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que <code>OMP_NUM_THREADS=4</code> ?	Usuario Profesores
	<pre>#pragma omp parallel printf("x");</pre>	<ul style="list-style-type: none"> <input checked="" type="checkbox"/> a) Ninguna otra respuesta es correcta <input checked="" type="checkbox"/> b) xxxx <input checked="" type="checkbox"/> c) x <input checked="" type="checkbox"/> d) xxx
6 Elección única	¿Cuál de las siguientes directivas no incorpora una barrera implícita al final?	Usuario Profesores
		<ul style="list-style-type: none"> <input checked="" type="checkbox"/> a) <code>atomic</code> <input checked="" type="checkbox"/> b) <code>for</code> <input checked="" type="checkbox"/> c) <code>sections</code> <input checked="" type="checkbox"/> d) <code>parallel</code>
7 Elección única	¿Qué directivas admiten una forma combinada?	Usuario Profesores
		<ul style="list-style-type: none"> <input checked="" type="checkbox"/> a) <code>parallel</code> y <code>for</code> <input checked="" type="checkbox"/> b) <code>parallel</code> y <code>reduction</code> <input checked="" type="checkbox"/> c) <code>parallel</code> y <code>schedule</code> <input checked="" type="checkbox"/> d) Todas las respuestas son correctas
8 Elección única	Cuando se mide el tiempo de ejecución de un programa mediante la orden del sistema <code>time</code> , la suma de los tiempos de usuario y sistema	Usuario Profesores
		<ul style="list-style-type: none"> <input checked="" type="checkbox"/> a) Es siempre menor o igual que el tiempo de ejecución <input checked="" type="checkbox"/> b) Para programas secuenciales, es siempre menor o igual que el tiempo de ejecución

- 9** ¿Qué componentes define la API de programación de OpenMP?
Elección única
Usuario Profesores
- a) Directivas del compilador y variables de sincronización entre hebras
 - b) Ninguna de las respuestas anteriores es correcta
 - c) Variables de entorno y funciones de paso de mensajes
 - d) Directivas del compilador, funciones y variables de entorno
- 10** ¿Cuándo existe una condición de carrera?
Elección única
Usuario Profesores
- a) Solamente si se garantiza el acceso en exclusión mutua a un recurso compartido
 - b) Siempre que diferentes hebras comparten una variable
 - c) Siempre que haya más de una hebra en ejecución
 - d) Cuando el resultado de un programa paralelo depende del orden de ejecución de sus hebras
- 11** ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que `OMP_NUM_THREADS=3`?
Elección única
`#pragma omp parallel`
{
 #pragma omp critical
 {
 printf("x");
 }
}
- Usuario Profesores
- a) x
 - b) xxxx
 - c) xxx
 - d) xx
- 12** ¿Cuál de las siguientes no es una directiva de OpenMP?
Elección única
Usuario Profesores
- a) `exclusive`
 - b) `for`
 - c) `single`
 - d) `master`



Examen BP1



Universidad de Granada - Doble Grado en Ingeniería Informática y
Matemáticas
Arquitectura de Computadores



Estudiante: 77692362 Molina Plaza, Salvador



Inicio: Hoy, miércoles, 09:40:05

Final: Hoy, miércoles, 09:52:25

Preguntas: 12

Respuestas

válidas:



Puntuación:



Nota:



1

¿Qué directivas admiten una forma combinada?

Usuario Profesores

Elección única



a) parallel y schedule



b) parallel y reduction



c) Todas las respuestas son correctas

-



d) parallel y for

2

¿Cuándo existe una condición de carrera?

Usuario Profesores

Elección única



a) Solamente si se garantiza el acceso en exclusión mutua a un recurso compartido

-



b) Cuando el resultado de un programa paralelo depende del orden de ejecución de sus hebras



c) Siempre que diferentes hebras comparten una variable



d) Siempre que haya más de una hebra en ejecución

3

¿Qué directivas no admiten una forma combinada?

Usuario Profesores

Elección única

-



a) parallel y single



b) parallel y sections



c) Ninguna otra respuesta es correcta



d) parallel y for

4

¿Qué identificador tiene la hebra máster en la ejecución de un programa

Elección única

paralelo?
Usuario Profesores

- a) Un valor numérico arbitrario, dependiente del entorno de ejecución
- b) El valor que defina el usuario mediante la función OpenMP `omp_set_master_num()`
- c) Un valor numérico arbitrario, dependiente de los identificadores que se usaron en la ejecución anterior
- d) El 0

5

Elección única

¿Cuál de las siguientes afirmaciones acerca de la directiva `master` es cierta?
Usuario Profesores

- a) Las hebras se sincronizan al principio y al final de la directiva.
- b) Las hebras no se sincronizan ni al principio ni al final de la directiva.
- c) Las hebras se sincronizan al final de la directiva, pero no al principio.
- d) Las hebras se sincronizan al principio de la directiva, pero no al final.

6

Elección única

¿Cuántas hebras ejecutará una directiva `sections` con 4 secciones (`section`) en una plataforma con 2 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 3?

Usuario Profesores

- a) 2
- b) 4
- c) 1
- d) 3

7

Elección única

¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que `OMP_NUM_THREADS=3`?

```
#pragma omp parallel
{
    #pragma omp critical
    {
        printf("x");
    }
}
```

Usuario Profesores

- a) x
- b) xx
- c) xxx
- d) xxxx

8

Elección única

¿Cuál de las siguientes combinaciones de directivas no es correcta?

Usuario Profesores

- a) `#pragma omp parallel single`
- b) `#pragma omp parallel sections`
- c) `#pragma omp parallel for`
- d) Ninguna otra es correcta

9**La directiva #pragma omp parallel**

Elección única

- Usuario Profesores
- a) Reparte las tareas de un bloque de código entre diferentes hebras
 - b) Permite que un bloque de código pueda ser ejecutado en paralelo por diferentes hebras
 - c) Paraleliza automáticamente un trozo de código en función de la arquitectura de la plataforma de ejecución
 - d) Paraleliza automáticamente un trozo de código, pero solo de forma genérica, sin tener en cuenta la arquitectura de la plataforma de ejecución

10

Elección única

¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva `critical` en una plataforma con 3 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 2?

Usuario Profesores

- a) 2
- b) 3
- c) 1
- d) 4

11

Elección única

Cuando se mide el tiempo de ejecución de un programa mediante la orden del sistema `time`, la suma de los tiempos de usuario y sistema ...

Usuario Profesores

- a) Es siempre menor o igual que el tiempo de ejecución
- b) Es siempre menor o igual que el tiempo de ejecución para programas secuenciales
- c) Es siempre menor o igual que el tiempo de ejecución para programas paralelos
- d) Es siempre igual que el tiempo de ejecución

12

Elección única

¿Se pueden generar las versiones secuencial y paralela de un programa a partir de un mismo fichero fuente?

Usuario Profesores

- a) Sí, generando código condicionado a la existencia del símbolo `_OPENMP`
- b) Sí, generando código condicionado al símbolo `_OPENMP` e incluyendo o no el flag `-fopenmp` a la hora de compilar
- c) Sí, incluyendo o no el flag `-fopenmp` a la hora de compilar
- d) No, es necesario tener dos ficheros fuente, uno para la versión secuencial y otro para la paralela



Examen BP1



Universidad de Granada - Doble Grado en Ingeniería Informática y
Matemáticas
Arquitectura de Computadores



Estudiante: 75578698 de la Hera Luis, Alejandro



Inicio: Hoy, miércoles, 09:40:02

Final: Hoy, miércoles, 09:53:32

Preguntas: 12

Respuestas

válidas:



Puntuación:



Nota:



1

¿Cuándo existe una condición de carrera?

Usuario Profesores

Elección única

- a) Siempre que haya más de una hebra en ejecución
- b) Solamente si se garantiza el acceso en exclusión mutua a un recurso compartido
- c) Cuando el resultado de un programa paralelo depende del orden de ejecución de sus hebras
- d) Siempre que diferentes hebras comparten una variable

2

¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva `critical` en una plataforma con 3 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 2?

Elección única

Usuario Profesores

- a) 1
- b) 3
- c) 2
- d) 4

3

Elección única

¿Cuántas hebras ejecutarán una directiva `single` en una plataforma con 2 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 4?

Usuario Profesores

- a) 4
- b) 2
- c) 3

-  d) 1

4 ¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva `critical` en una plataforma con 4 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 2?

Elección única
Usuario Profesores

-  a) 2
-  b) 1
-  c) 3
-  d) 4

5 ¿Cuál de las siguientes afirmaciones es correcta?

Elección única
Usuario Profesores

-  a) Las directivas `master` y `atomic` tienen una barrera implícita al final
-  b) Ninguna de las otras afirmaciones es correcta
-  c) Con la directiva `master` las hebras se sincronizan al principio pero no al final.
-  d) Con la directiva `atomic` las hebras se sincronizan al principio pero no al final.

6 ¿Para qué sirve la directiva `barrier`?

Elección única
Usuario Profesores

-  a) Para evitar las condiciones de carrera
-  b) Para fijar en el código un punto de sincronización de todas las hebras
-  c) Para proteger el acceso a una variable compartida
-  d) Para que todas las hebras terminen su ejecución en ese punto

7 ¿Cuántas hebras ejecutará una directiva `sections` con 4 secciones (`section`) en una plataforma con 2 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 3?

Elección única
Usuario Profesores

-  a) 4
-  b) 3
-  c) 2
-  d) 1

8 La directiva `#pragma omp parallel`

Elección única
Usuario Profesores

-  a) Paraleliza automáticamente un trozo de código en función de la arquitectura de la plataforma de ejecución
-  b) Paraleliza automáticamente un trozo de código, pero solo de forma genérica, sin tener en cuenta la arquitectura de la plataforma de ejecución
-  c) Permite que un bloque de código pueda ser ejecutado en paralelo por diferentes hebras
-  d) Reparte las tareas de un bloque de código entre diferentes hebras

¿Cuáles de las siguientes directivas de OpenMP incorporan una barrera implícita al final?

Elección única

9

Usuario Profesores

- a) La sincronización se realiza al final de la directiva, pero no al principio
- b) La sincronización se realiza al principio y al final de la directiva
- c) No se realiza sincronización
- d) La sincronización se realiza al principio de la directiva, pero no al final

10

Elección única

De las siguientes directivas de OpenMP, ¿cuáles incorporan una barrera implícita al final?

Usuario Profesores

- a) Ninguna de las otras respuestas es correcta
- b) `parallel` y `critical`
- c) `sections` y `critical`
- d) `parallel` y `master`

11

Elección única

¿Qué directiva usarías para que las hebras ejecuten el siguiente código en exclusión mutua?

```
{b = b-1; c = c + 1;}
```

Usuario Profesores

- a) `atomic`
- b) `master`
- c) `barrier`
- d) `critical`

12

Elección única

¿Qué paradigma de programación paralela implementa OpenMP?

Usuario Profesores

- a) Memoria compartida mediante hebras
- b) SIMD
- c) Paso de mensajes mediante procesos
- d) Memoria compartida mediante procesos

- 1** ¿Qué parámetro hace que gcc genere código ensamblador?
Elección única
 a) -asm
 b) -S
 c) -lrt
 d) -O
- 2** ¿Qué directiva usarías para que las hebras ejecuten el siguiente código en exclusión mutua?
Elección única

```
{b -= 2; }
```

 a) Solo **critical**
 b) **exclusive**
 c) **critical** o **atomic**
 d) Solo **atomic**
- 3** ¿Cuál de las siguientes afirmaciones acerca de la directiva **master** es cierta?
Elección única
 a) Las hebras no se sincronizan ni al principio ni al final de la directiva.
 b) Las hebras se sincronizan al principio y al final de la directiva.
 c) Las hebras se sincronizan al principio de la directiva, pero no al final.
 d) Las hebras se sincronizan al final de la directiva, pero no al principio.
- 4** ¿Para qué sirve la directiva **barrier**?
Elección única
 a) Para evitar las condiciones de carrera
 b) Para fijar un punto en el código que ninguna hebra podrá sobrepasar hasta que lo hayan alcanzado todas las demás
 c) Para que todas las hebras vayan a la misma velocidad
 d) Para proteger el acceso a una variable compartida
- 5** ¿Qué directiva es más adecuada para ejecutar en paralelo un bloque estructurado?
Elección única
 a) **parallel**
 b) **section**
 c) **atomic**
 d) **for**
- 6** ¿Qué función proporciona OpenMP para obtener el tiempo en aplicaciones multihebra?
Elección única
 a) **omp_clock_gettime()**
 b) **omp_get_clocktime()**
 c) **clock_gettime()**
 d) **omp_get_wtime()**
- 7** ¿Cuántas hebras ejecutarán una directiva **single** en una plataforma con 2 cores en la que se ha fijado la variable de entorno **OMP_NUM_THREADS** al valor 4?
Elección única
 a) 1
 b) 3
 c) 2

- c) 2
- d) 4

8

Elección
única

¿Cuándo decimos que existe una condición de carrera?

- a) ~~Siempre que diferentes hebras comparten una variable~~
- b) No existe si varias hebras comparten variables
- c) Solo si se garantiza el acceso en exclusión mutua a una variable compartida
- d) **Cuando el resultado de un programa paralelo depende del orden de ejecución de sus hebras**

9

Elección
única

¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva **critical** en una plataforma con 3 cores en la que se ha fijado la variable de entorno **OMP_NUM_THREADS** al valor 2?

- a) 1
- b) 4
- c) 3
- d) 2

10

Elección
única

¿Qué componentes define la API de programación de OpenMP?

- a) Variables de entorno y funciones de paso de mensajes
- b) Ninguna de las respuestas anteriores es correcta
- c) Directivas del compilador y variables de sincronización entre hebras
- d) Directivas del compilador, funciones y variables de entorno

11

Elección
única

¿Qué paradigma de programación paralela implementa OpenMP?

- a) Memoria compartida mediante procesos
- b) Paso de mensajes mediante procesos
- c) Memoria compartida mediante hebras
- d) SIMD

12

Elección
única

Cuando se mide el tiempo de ejecución de un programa mediante la orden del sistema **time**, la suma de los tiempos de usuario y sistema ...

- a) Es siempre menor o igual que el tiempo de ejecución para programas secuenciales
- b) Es siempre menor o igual que el tiempo de ejecución para programas paralelos
- c) Es siempre menor o igual que el tiempo de ejecución
- d) Es siempre igual que el tiempo de ejecución

He terminado



32 de AC

2 profesores



Juan J...
Escoba...
Julio
Ortega ...

4'58"



22'32"



Sergio
Blas Ríos
Beatriz
Jalón V...
Juan M...
Torres ...
Marcos
Rico G...
Guillermo
López ...
Rafael
Luque ...
Nerea
Fernán...
Jesús
García ...

2'41"



2'43"



2'44"



2'45"



2'46"



2'49"



2'52"



2'55"

...

30 estudiantes



- d) El o

2

¿Qué parámetro hace que gcc genere código ensamblador?

- a) -0
 - b) -lrt
 - c) -S
 - d) -asm

3

¿Cuál de las siguientes no es una directiva de OpenMP?

- a) exclusive
 - b) for
 - c) single
 - d) master

4

¿Cuántas hebras ejecutarán una directiva `sections` con 4 secciones (`section`) en una plataforma con 4 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 2? (considere que no se usan en el código funciones OpenMP)

- a) 3
 - b) 2
 - c) 1
 - d) 4

5

De las siguientes directivas de OpenMP, ¿cuáles incorporan una barrera implícita al final?

- a) sections y critical
 - b) parallel y sections
 - c) atomic y critical
 - d) parallel y master

6

La directiva #pragma omp parallel

- a) Paraleliza automáticamente un trozo de código, pero solo de forma genérica, sin tener en cuenta la arquitectura de la plataforma de ejecución
 - b) Permite que un bloque de código pueda ser ejecutado en paralelo por diferentes hebras
 - c) Reparte las tareas de un bloque de código entre diferentes hebras
 - d) Paraleliza automáticamente un trozo de código en función de la arquitectura de la plataforma de ejecución

7

¿Cuál de las siguientes afirmaciones acerca de la directiva master es cierta?

- a) Las hebras se sincronizan al final de la directiva, pero no al principio.
 - b) Las hebras se sincronizan al principio y al final de la directiva.
 - c) Las hebras no se sincronizan ni al principio ni al final de la directiva.
 - d) Las hebras se sincronizan al principio de la directiva, pero no al final.

8

¿Cuál de las siguientes no es una directiva de OpenMP?

- a) atomic
 - b) single
 - c) schedule
 - d) master

9Elección
única

¿Cuál de las siguientes directivas no incorpora una barrera implícita al final?

- a) `for`
- b) `sections`
- c) `parallel`
- d) `atomic`

10Elección
única

¿Qué paradigma de programación paralela implementa OpenMP?

- a) SIMD
- b) Memoria compartida mediante procesos
- c) Memoria compartida mediante hebras
- d) Paso de mensajes mediante procesos

11Elección
única

¿Qué componentes define la API de programación de OpenMP?

- a) Directivas del compilador y variables de sincronización entre hebras
- b) Directivas del compilador, funciones y variables de entorno
- c) Ninguna de las respuestas anteriores es correcta
- d) Variables de entorno y funciones de paso de mensajes

12Elección
única

¿Qué función proporciona OpenMP para obtener el tiempo en aplicaciones multihebra?

- a) `omp_get_wtime()`
- b) `omp_get_clocktime()`
- c) `clock_gettime()`
- d) `omp_clock_gettime()`

He terminado

Información Documental UGR

¿Qué es SWAD	Manual breve	Condiciones legales	Twitter
What is SWAD?	Brief manual	[E]Protección de datos	Facebook
Publicaciones	Guía usuario	[I]Twitter	SWAD
Funcionalidad	User guide	[E]Estadísticas	Wikipedia
Difusión	Presentación	Póster	
Prensa	Videotutoriales	Servidor	
	Logos	Encuentro	

Community Software

Source code	SWADroid	Google	iSWAD App Store
Download	SWADroid	Blog	iSWAD Twitter
Install	SWADroid	Twittei	SWAD GitHub
Database	SWADroid	Google	
Translation	SWADroid	Github	
alternativeTo API	SWADroid	Open HUB	
startupRANK	SWADroid	Open HUB	
Capterra	Roadmap		
SourceForge	Authors		
GitHub	Implementation		
Open HUB			

iOS



Universidad de Granada

Consultas y problemas: swad@ugr.es

Acerca de SWAD 20.30 (2021-02-11) Página generada en 36 ms y enviada en 509 µs

Examen BP1



Universidad de Granada - Doble Grado en Ingeniería Informática y
Matemáticas
Arquitectura de Computadores



Estudiante: 75925602
75925602A Luque Framit, Rafael



Inicio: Hoy, viernes, 09:41:30

Final: Hoy, viernes, 09:48:50

Preguntas: 12

Respuestas
válidas:



Puntuación:



Nota:



1

Elección única

¿Cuántas hebras ejecutarán una directiva `single` en una plataforma con 2 cores en la que se ha fijado la variable de entorno `OMP_NUM_THREADS` al valor 4?

Usuario Profesores

- a) 1
- b) 4
- c) 3
- d) 2

2

Elección única

¿Qué directivas OpenMP son las más adecuadas para crear varias hebras y repartirles las iteraciones de un bucle?

Usuario Profesores

- a) `parallel` y `for`
- b) Solo `for`
- c) `parallel` y `sections`
- d) `parallel` y `single`

3

Elección única

¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que `OMP_NUM_THREADS=3`?

```
#pragma omp parallel
{
    #pragma omp single
    {
        printf("x");
    }
}
```

}

Usuario Profesores

- a) xx
- b) xxx
- c) x
- d) Indeterminado, porque existe condición de carrera

4

¿Qué directivas admiten una forma combinada?

Elección única

- Usuario Profesores
- a) `parallel y for`
 - b) Todas las respuestas son correctas
 - c) `parallel y reduction`
 - d) `parallel y schedule`

5

`master ...`

Elección única

- Usuario Profesores
- a) no es una directiva de trabajo compartido y no tiene barrera final implícita
 - b) es una directiva de trabajo compartido sin barrera final implícita
 - c) es una directiva de trabajo compartido con barrera final implícita
 - d) es una directiva con barrera final implícita pero no de trabajo compartido

6

¿Qué directiva usarías para convertir en sección crítica el siguiente código:

Elección única

`a = b;`
`c = d + 1;`

Usuario Profesores

- a) `critical`
- b) `atomic`
- c) `master`
- d) `barrier`

7

¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que `OMP_NUM_THREADS=3`?

Elección única

```
#pragma omp parallel
{
    #pragma omp critical
    {
        printf("x");
    }
}
```

Usuario Profesores

- a) xx
- b) x
- c) xxx
- d) xxxx

8

¿Cuántas hebras ejecutarán una directiva `sections` con 4 secciones (`section`)

- 8** Elección única en una plataforma con 8 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 3? (considere que no se usan en el código funciones OpenMP)
- Usuario Profesores
- a) 2
 - b) 3
 - c) Ninguna de las otras respuestas es correcta
 - d) 1
- 9** Elección única ¿Cuál de las siguientes afirmaciones es correcta?
- Usuario Profesores
- a) Las directivas master y atomic tienen una barrera implícita al final
 - b) Ninguna de las otras afirmaciones es correcta
 - c) Con la directiva master las hebras se sincronizan al principio pero no al final.
 - d) Con la directiva atomic las hebras se sincronizan al principio pero no al final.
- 10** Elección única El número de MIPS de un programa...
- Usuario Profesores
- a) siempre será mayor o igual que su número de MFLOPS
 - b) será mayor que su número de MFLOPS sólo si el programa usa datos en coma flotante
 - c) no guarda ninguna relación con su número de MFLOPS
 - d) puede ser menor que su número de MFLOPS si el programa es de cálculo intensivo
- 11** Elección única ¿Cuántas hebras ejecutarán una directiva sections con 4 secciones (section) en una plataforma con 4 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2? (considere que no se usan en el código funciones OpenMP)
- Usuario Profesores
- a) 2
 - b) 3
 - c) 4
 - d) 1
- 12** Elección única ¿Qué función proporciona OpenMP para obtener el tiempo en aplicaciones multihebra?
- Usuario Profesores
- a) omp_get_clocktime()
 - b) omp_clock_gettime()
 - c) clock_gettime()
 - d) omp_get_wtime()

8. Analiza el siguiente código e indica qué nos dirá el compilador referido a las siguientes líneas de código

```
int i, n = 1;  
#pragma omp parallel for default(none) private(i)  
    for (i = 0; i < 5; ++i)  
        n += i;
```

- a) La variable i no puede ser privada
- b) La variable n no está especificada**
- c) La variable n no está especificada como privada
- d) Ninguna opción de las anteriores

9. Indica qué valor tendrá la variable ret después de ejecutar la siguiente reducción cuando se ejecuta con 4 hebras

```
int i, n = 6, ret= 1;  
#pragma omp parallel reduction(+:ret)  
for (i=omp_get_thread_num(); i<omp_get_max_threads();  
i+=omp_get_num_threads())  
    ret += i;
```

- a) 5
- b) 7**
- c) El valor de ret será indeterminado porque existe una condición de carrera**
- d) 3

10. Asumiendo que v2 de dimensión N y que todos sus elementos están inicializados a cero, ¿cuál de los siguientes códigos calcula de forma correcta el producto de la matriz m (dimensión NxN) por el vector v1 (dimensión N) paralelizando el bucle que recorre las columnas?

- a) #pragma omp parallel private(j) for(i=0;i<N;i++){ #pragma omp for reduction(+:v2[i]) for(j=0;j<N;j++){ v2[i] += m[i][j]*v1[j]; } }
- b) #pragma omp parallel for private(j) for(i=0;i<N;i++){ for(j=0;j<N;j++){ v2[i] += m[i][j]*v1[j]; } }**
- c) #pragma omp parallel private(i,j) for(i=0;i<N;i++){ for(j=0;j<N;j++){ v2[i] += m[i][j]*v1[j]; } }
- d) #pragma omp parallel private(i) for(i=0;i<N;i++){ #pragma omp for reduction(+:v2[i]) for(j=0;j<N;j++){ v2[i] += m[i][j]*v1[j]; } }

11. ¿Cuál de las siguientes directivas no admite ninguna cláusula?

- a) single
- b) parallel
- c) Todas las directivas admiten alguna cláusula
- d) critical**

12. Supongamos una máquina en la que el número de hebras de las que se puede disponer para ejecutar zonas paralelas de código es limitado. En esas condiciones ¿qué valdrá la variable n al terminar de ejecutar el siguiente código?

```
int n = 1;  
#pragma omp parallel for firstprivate(n) lastprivate(n)  
    for (int i = 0; i < 5; ++i)  
        n += 1;
```

- a) n al salir del bucle está indefinido porque es privada
- b) Dependerá del número de hebras que lo ejecuten en cada momento**
- c) 1
- d) 5

12. ¿Cuál de las siguientes directivas no admite ninguna cláusula?

- a) single**
- b) critical**
- c) parallel
- d) Todas las directivas admiten alguna cláusula

13. Supongamos una máquina en la que el número de hebras de las que se puede disponer para ejecutar zonas paralelas de código es limitado. En esas condiciones, ¿qué valdrá la variable n al terminar de ejecutar el siguiente código?

```
int n = 1;  
#pragma omp parallel for firstprivate(n) lastprivate(n)  
    for (int i = 0; i < 5; ++i)  
        n += i;
```

- a) n al salir del bucle está indefinida porque es privada
- b) Dependerá del número de hebras que lo ejecuten en cada momento**
- c) 1
- d) 5

14. ¿Cuál será el valor de n tras ejecutar el siguiente código?

```
int i, n=2;  
#pragma omp parallel shared(n) private(i)  
    for(i=0; i < 4; i++){  
        #pragma omp single  
        {  
            n += i;  
        }  
    }
```

- a) 16
- b) 8**
- c) 2
- d) Indeterminado

15. ¿Cuál de los siguientes fragmentos de código paralelo calcula correctamente la sumatoria de los primeros números impares hasta el N = 1000?

- a) int sum = 0;
 #pragma omp parallel for
 for(long i = 1; i < N; i += 2)
 sum += i;
- b) int sum = 0;
 #pragma omp parallel
 for(long i = 1; i < N; i += 2)
 sum += i;
- c) int sum = 0;
 #pragma omp parallel sections{
 #pragma omp section
 for (long i = 1; i < N; i += 4)
 sum += i;
 #pragma omp section
 for (long i = 3; i < N; i += 4)
 sum += i;
 }
- d) Ninguna otra respuesta es correcta

16. Indica cual será el valor de la variable n al final de la ejecución del siguiente código

```
int n = 0;  
#pragma omp parallel for reduction(*:n)  
    for (int i = n; i < size; ++i)  
        n += i;
```

- a) Ninguna respuesta es correcta
- b) El valor de n será igual a size -1
- c) 0
- d) Dependerá del valor de la variable size

17. ¿Cuánto vale ret al final?

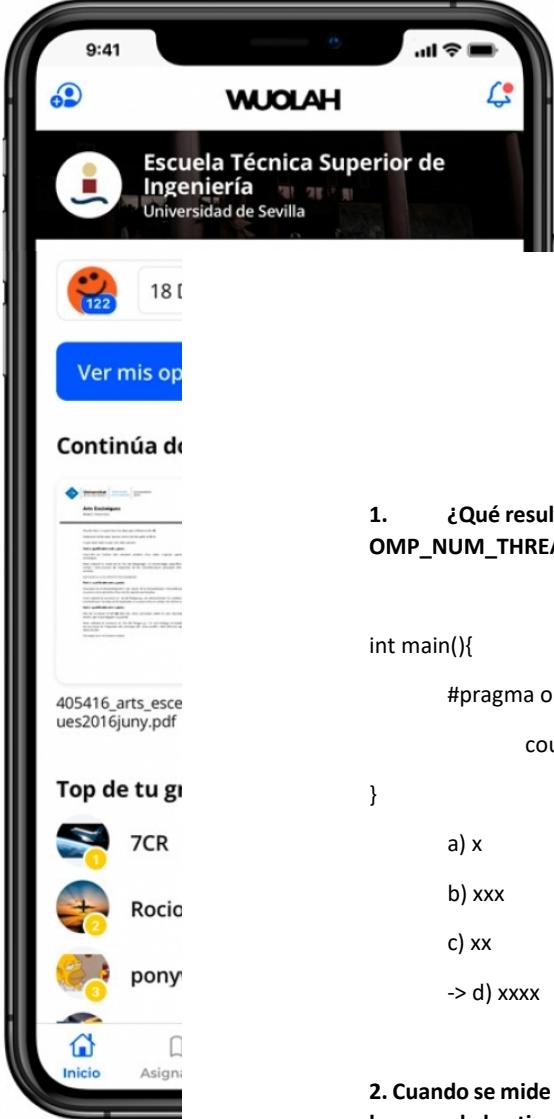
```
int i, n = 6; ret = 1;  
#pragma omp parallel reduction(+:ret) private(i)  
for (i=omp_get_thread_num(); i<n; i+=omp_get_num_threads())  
    ret += i;  
return ret;
```

- a) 15
- b) 1
- c) 16
- d) Ninguna de las otras respuestas es correcta

18. ¿Cuál de las siguientes afirmaciones es correcta?

- a) Las cláusulas ajustan el comportamiento de las directivas

- b) Ninguna otra respuesta es correcta
- c) Directivas y cláusulas son iguales
- d) Las directivas ajustan el comportamiento de las cláusulas



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

1. ¿Qué resultado muestra por pantalla la ejecución del siguiente código suponiendo que **OMP_NUM_THREADS=4?**

```
int main(){  
    #pragma omp parallel  
    cout << "x";  
}  
  
a) x  
b) xxx  
c) xx  
-> d) xxxx
```

2. Cuando se mide el tiempo de ejecución de un programa mediante la orden del sistema **time**, la suma de los tiempos de usuario y sistema.

- a) Para programas paralelos, es siempre menor o igual que el tiempo de ejecución.
- > b) Para programas secuenciales, es siempre menor o igual que el tiempo de ejecución.
- c) Es siempre menor o igual que el tiempo de ejecución.
- d) Es siempre igual que el tiempo de ejecución.

3. De las siguientes directivas de OpenMP, ¿cuáles incorporan una barrera implícita al final?

- a) parallel y master
- b) atomic y critical
- c) sections y critical
- > d) parallel y sections

4. ¿Cuántas hebras ejecutarán una directiva **section** con 4 secciones (**section**) en una plataforma con 8 cores en la que se ha fijado la variable de entorno **OMP_NUM_THREADS** al valor 3?

- > a) 3
- b) 2
- c) 1
- d) Ninguna de las otras respuestas es correcta

5. ¿Qué función proporciona OpenMP para obtener el tiempo en aplicaciones multi-hebra?

- a) omp_get_clocktime()
- b) omp_clock_gettime()
- > c) omp_get_wtime()
- d) clock_gettime()

6. ¿Cuál de las siguientes afirmaciones acerca del sections es cierta?

- a) La sincronización se realiza al principio de la directiva, pero no al final.
- > b) La sincronización se realiza al final de la directiva, pero no al principio.
- c) No se realiza sincronización.
- d) La sincronización se realiza al principio y al final de la directiva.

7. ¿Qué directivas OpenMP se deben usar para diseñar un programa que ejecute en paralelo dos funciones independientes?

- a) single y sections
- > b) parallel y sections
- c) parallel y for
- d) for y sections

8. ¿Cuándo decimos que existe una condición de carrera?

- a) No existe si varias hebras comparten variables.
- > b) Cuando el resultado de un programa paralelo depende del orden de ejecución de sus hebras.
- c) Siempre que diferentes hebras comparten una variable.
- d) Solo si se garantiza el acceso en exclusión mutua a una variable compartida.

9. ¿Se pueden generar las versiones secuencial y paralela de un programa a partir de un mismo fichero fuente?

- > a) Sí, generando código condicionado al símbolo _OPENMP e incluyendo o no el flag -fopenmp a la hora de compilar.
- b) Sí, generando código condicionado a la existencia del símbolo _OPENMP

- c) Sí, incluyendo o no el flag -fopenmp a la hora de compilar.
- d) No, es necesario tener dos ficheros fuente, uno para la versión secuencial y otro para la paralela.

10. ¿Cuántas hebras ejecutará una directiva sections con 4 secciones (section) en una plataforma con 2 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 3?

- a) 1
- b) 4
- > c) 2
- d) 3

11. ¿Cuál de las siguientes directivas no incorpora una barrera implícita al final?

- > a) atomic
- b) sections
- c) for
- d) parallel

12. ¿Cuál de las siguientes no es una directiva de OpenMP?

- a) for
- b) single
- c) master
- > d) exclusive

13. Paradigma de OpenMP

- ? -> a) Memoria compartida mediante hebras
- b) Paso de mensajes mediante procesos
- c) Memoria compartida mediante procesos
- d) SIMD

14. ¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva en una plataforma con 4 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2?

- a) 1
- b) 4
- > c) 2
- d) 3

15. ¿Para qué sirve la directiva barrier?

- > a) Para fijar en el código un punto de sincronización de todas las hebras.
- b) Para evitar las condiciones de carrera.
- c) Para proteger el acceso a una variable compartida.
- d) Para que todas las hebras terminen su ejecución en ese punto.

16. ¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva en una plataforma con 3 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2?

- a) 3
- b) 1
- > c) 2
- d) 4

17. Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
{
    sumalocal=0;
    #pragma omp for
    for(i=0; i<n; i++)
        sumlocal += a[i];
    #pragma omp barrier
    #pragma omp critical
        suma = suma + sumalocal;
    #pragma omp barrier
    #pragma omp single
```

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



```
    printf("La suma es = %d/n", suma);  
}
```

a) El valor de la suma que se imprime sería correcto si cambiáramos private(sumalocal) por private(sumalocal, suma).

- > b) Uno de los #pragma omp barrier es innecesario.
- c) Todas las demás respuestas son incorrectas.
- d) El valor de la suma que se imprime no es siempre correcto.

18. ¿Qué resultado muestra por pantalla la ejecución del siguiente código suponiendo que OMP_NUM_THREADS=3?

```
#pragma omp parallel  
#pragma omp single  
    cout << "x";
```

- > a) x
- b) xxx
- c) xx
- d) xxxx

19. ¿Cuáles de las siguientes afirmaciones es correcta?

- a) Con la directiva master las hebras se sincronizan al principio pero no al final.
- b) Con la directiva atomic las hebras se sincronizan al principio pero no al final.
- c) Las directivas master y atomic tienen una barrera implícita al final.
- > d) Ninguna de las otras afirmaciones es correcta.

20. La API de programación de OpenMP está formada por ...

- a) Variables de entorno definidas en consola y funciones de paso de mensajes.
- b) Variables del compilador y funciones para la medición de tiempo en programas con varias hebras.
- > c) Directivas del compilador, funciones de biblioteca, y variables de entorno.



WUOLAH

- d) Ninguna de las otras respuestas es correcta.

21. ¿Qué directivas admiten una forma combinada?

- a) parallel y schedule
- b) Todas las respuestas son correctas
- > c) parallel y for
- d) parallel y reduction

22. ¿Qué componentes define la API de programación de OpenMP?

- a) Directivas de compilador y variables de sincronización entre hebras.
- b) Ninguna de las respuestas anteriores es correcta.
- c) Variables de entorno y funciones de paso de mensajes.
- > d) Directivas de compilador, funciones y variables de entorno.

23. ¿Cuál de las siguientes afirmaciones acerca de la directiva master es cierta?

- a) Las hebras se sincronizan al principio y al final de la directiva.
- > b) Las hebras no se sincronizan ni principio ni al final de la directiva.
- c) Las hebras se sincronizan al final de la directiva, pero no al principio.
- d) Las hebras se sincronizan al principio de la directiva, pero no al final.

24. ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=4?

```
#pragma omp parallel
```

```
    printf("x");
```

- >a) xxxx
- b) Ninguna otra respuesta es correcta
- c) xxx
- d) x

25. ¿Cuántas hebras ejecutarán una directiva section con 4 secciones (section) en una plataforma con 8 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 3? (considere que no se usan en el código funciones OpenMP)

- > a) 3
- b) Ninguna de las otras respuestas es correcta
- c) 2
- d) 1

26. El número de MIPS de un programa...

- > a) siempre será mayor o igual que su número de MFLOPS.
- b) no guarda ninguna relación con su número de MFLOPS.
- c) puede ser menor que su número de MFLOPS si el programa es de cálculo intensivo.
- d) será mayor que su número de MFLOPS sólo si el programa usa datos en coma flotante.

27. ¿Qué directiva usarías para que las hebras ejecuten el siguiente código en exclusión mutua?

{b -= 2;}

- a) Solo critical
- b) exclusive
- c) Solo atomic
- > d) critical o atomic

28. ¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva critical en una plataforma con 3 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2?

- a) 4
- b) 2
- > c) 1
- d) 3

29. ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=3?

```
#pragma omp parallel
```

```
#pragma omp single
```

```
    cout << "x";
```

- a) xxx
- b) xx
- > c) x
- d) Indeterminado, porque existe condición de carrera.

30. ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=4?

```
#pragma omp parallel
```

```
    cout << "x";
```

- a) xxx
- b) Ninguna otra respuesta es correcta.
- > c) xxxx
- d) x

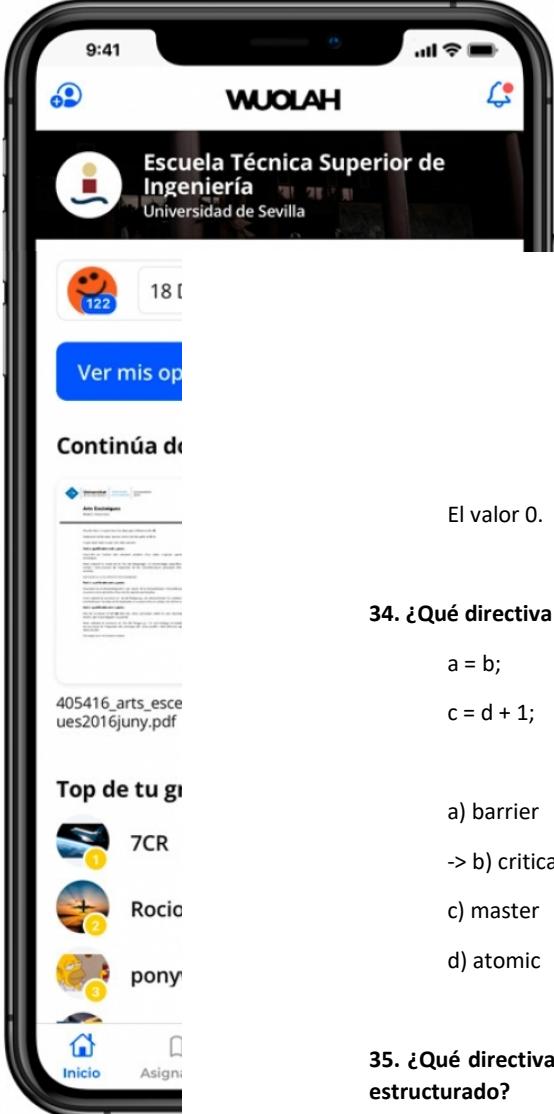
31. De las siguientes directivas de OpenMP, ¿cuáles incorporan una barrera implícita al final?

- a) parallel y master
- > b) Ninguna de las otras respuestas es correcta
- c) sections y critical
- d) parallel y critical

32. ¿Cuál de las siguientes combinaciones de directivas no es correcta?

- a) #pragma omp parallel sections
- b) #pragma omp parallel for
- > c) #pragma omp parallel single
- d) Ninguna otra es correcta

33. ¿Qué identificador tiene la hebra máster en la ejecución de un programa paralelo?



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

- Continúa d
- El valor 0.
- 34. ¿Qué directiva usarías para convertir en sección crítica el siguiente código?**
- a = b;
c = d + 1;

- a) barrier
-> b) critical
c) master
d) atomic

- 35. ¿Qué directiva OpenMP se encarga de crear hebras para ejecutar en paralelo su bloque estructurado?**

- a) atomic
-> b) parallel
c) section
d) for

- 36. ¿Qué directivas OpenMP son las más adecuadas para crear varias hebras y repartirles las iteraciones de un bucle?**

- a) parallel y single
b) parallel y sections
-> c) parallel y for
d) Solo for

- 37. ¿Cuál de las siguientes no es una directiva OpenMP?**

- > a) schedule
b) master
c) atomic
d) single

- 38. La directiva #pragma omp parallel**

a) Paraleliza automáticamente un trozo de código...

-> b) Permite que un bloque de código pueda ser ejecutado en paralelo por diferentes hebras.

c) Reparte las tareas de un bloque de código entre diferentes hebras.

d) Paraleliza automáticamente un trozo de código en función de la arquitectura...

39. ¿Cuántas hebras pueden ejecutar en paralelo el bloque estructurado de una directiva critical en una plataforma con 4 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2?

a) 4

-> b) 2

c) 1

d) 3

40. ¿Qué resultado muestra por pantalla la ejecución del siguiente código que no usa funciones OpenMP suponiendo que OMP_NUM_THREADS=3?

```
#pragma omp parallel
{
    #pragma omp critical
    {
        cout << "x";
    }
}
```

a) xxxx

-> b) xxx

c) x

d) xx

41. ¿Cuántas hebras ejecutarán una directiva section con 4 secciones (section) en una plataforma con 4 cores en la que se ha fijado la variable de entorno OMP_NUM_THREADS al valor 2? (considere que no se usan en el código funciones OpenMP)

-> a) 2

- b) 4
- c) 3
- d) 1

42. ¿Qué directivas no admiten una forma combinada?

- a) parallel y for
- b) Ninguna otra respuesta es correcta
- > c) parallel y single
- d) parallel y sections

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



<https://swad.ugr.es/es>

?

Tema 2 Prueba Evaluación Continua

eniería Informática
dores



Preguntas: 10

Respuestas
válidas: 0

Puntuación: 0

Nota: 0

- 1** En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) prefijo paralelo, el procesador P2 recibe información solo del procesador P3 y del propio P2
V/F Usuario Profesores

F 0

- 2** En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) prefijo paralelo, el procesador P2 recibe información de los procesadores P0, P1, y del propio P2 (aparte de otras posibles comunicaciones)
V/F Usuario Profesores

V 0

- 3** La expresión para la ley de Gustafson es $S=f+p*(1-f)$, donde f es la fracción no paralelizable del tiempo de ejecución paralelo y p es el número de procesadores que intervienen.
V/F Usuario Profesores

V 0

- 4** Un programa paralelo tarda 200 ns. Durante 50 ns solo puede ser ejecutado por un procesador y durante los otros 150 ns intervienen 4 procesadores (todos ellos igual de cargados). La sobrecarga se considera despreciable. El valor de la ganancia de velocidad es menor que 3.
V/F Usuario Profesores

F 0

- 5** Un programa paralelo tarda 20 ns. Durante 10 ns solo puede ser ejecutado por d d t l t 10 i t i 5 d (t d



WUOLAH

5 V/F un procesador y durante los otros 10 ns intervienen 5 procesadores (todos ellos igual de cargados). El valor de la f de la ley de Gustafson es 0.5

Usuari@Profesores

V 

6 V/F El tiempo de sobrecarga u overhead de un programa paralelo se debe únicamente al tiempo de comunicación entre los procesadores

Usuari@Profesores

F 

7 V/F La ganancia de velocidad que consiguen p procesadores en un código secuencial que tarda un tiempo T_s en ejecutarse en un procesador, con una fracción no paralela de T_s igual a 0, un grado de paralelismo igual a n y un tiempo de overhead igual a 0 es igual a p para $p < n$

Usuari@Profesores

V 

8 V/F En un computador MIMD no se puede utilizar el modo de programación SPMD (Single Program Multiple Data)

Usuari@Profesores

F 

9 V/F En la asignación estática de tareas a procesos/hebras, distintas ejecuciones pueden asignar distintas tareas a un procesador o núcleo

Usuari@Profesores

F 

10 Dado el bucle

V/F

```
for (i=0;i<lter;i++) {  
    código para i  
}
```

Mediante

```
for (i=idT*(lter/nT); i<((idT+1)*(lter/nT);i++) {  
    código para i  
}
```

Se consigue la distribución estática round-robin de las lter iteraciones del bucle entre nT hebras, cuyo identificador es idT ($idT=0,1,\dots,nT-1$) (Nota: lter es múltiplo de nT)

Usuari@Profesores

F 

PRESUME DE SONRISA

Escanea este código y estrena tu ortodoncia invisible

AL TERMINAR TU TRATAMIENTO
BLANQUEAMIENTO* DENTAL GRATIS

*Blanqueamiento bajo prescripción médica. Promoción no acumulable a otros descuentos y/o promociones. CS10715



VITALDENT

Queremos verte sonreír

VITALDENT

Queremos verte sonreír

1. OpenMP es una biblioteca que permite hacer programas paralelos con paso de mensajes.

F -> eso es MPI, OpenMP es una interfaz de programación para la programación de multiproceso y memoria compartida.

2. MPI es una biblioteca de paso de mensajes

V

3. El tiempo de comunicación entre procesos forma parte del overhead de un programa paralelo.

V -> es la suma de los tiempos de comunicación y sincronización de los procesos, el tiempo de creación y terminación de hebras y las funciones que se encuentren de forma no secuencial.

4. El tiempo de sobrecarga u overhead es un componente del tiempo de procesamiento paralelo junto con el tiempo de comunicación.

F

5. La asignación de carga dinámica afecta al tiempo de overhead del programa paralelo.

V

6. La asignación de carga dinámica se realiza antes de la ejecución del programa paralelo.

F -> se hace en tiempo de ejecución

7. En la asignación de carga estática se asigna el trabajo que va a realizar cada procesador antes de la ejecución.

V

8. Para equilibrar la carga asignada a los procesadores interesa asignar más carga a los procesadores más rápidos

V, porque el equilibrio de carga consiste en que lleguen en momentos diferentes y tengan que estar esperando.

9. La falta de equilibrado de la carga es una de las causas de que haya tiempo de sobrecarga u overhead en los programas paralelos.

V, porque aumenta el tiempo en el que hebras tengan que esperar

10. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, cosa que no ocurre en la comunicación gossiping.

F, en gossiping sí que reciben información de todos los procesadores.

11. En la comunicación colectiva all-scatter todos los procesadores reciben información de todos, cosa que también ocurre en la comunicación gossiping.

V

12. En la comunicación colectiva de tipo gossiping todos los procesadores envían información, pero no todos los procesadores reciben.

F, todos los procesadores envían y reciben

13. La acumulación implica comunicación colectiva todos con todos.

F, es todos a uno

14. La acumulación es un modo de comunicación colectiva en el que todos los procesadores envían información a uno de ellos.

V, son todos a uno.

15. La difusión implica comunicación colectiva todos con todos.

F, puede ser también uno a uno.

16. La dispersión colectiva todos a uno.

F, puede ser también todos a todos.

17. Tanto la difusión como la dispersión implican comunicación de un procesador a todos los demás.

V -> uno a todos y todos a todos.

18. En un multicomputador con 4 procesadores (P0 a P3), mediante la comunicación de recorrido (scan) prefijo paralelo, el procesador P2 recibe información de los procesos P0, P1 y P2.

V, en el prefijo los procesadores mandan información a todos los procesos con índice igual o superior.

19. En un multicomputador con 4 procesadores (P0 a P3), mediante la permutación de rotación, el procesador P0 envía información al procesador P1 y recibe de P2.

F, P0 envía a P1 y recibe de P3.

20. La expresión para la ley de Gustafson es $S = f + p * (1-f)$, donde f es la fracción de no paralelizable del tiempo de ejecución presencial y p es el número de procesadores que intervienen.

V, f es la parte secuencial, p los procesadores y 1-f la parte no secuencial.

21. Un programa paralelo tarda 200 ns. Durante 50 ns solo puede ser ejecutado por un procesador y durante los 150 ns intervienen 4 procesadores. La sobrecarga es despreciable. El valor de la ganancia de velocidad es mayor que 3.

Verdadero

$$T_s = (1 * 50 + 4 * 150) = 650$$
$$S = T_s / T_p = 650 / 200 = 3.25$$

22. Un programa paralelo tarda 20 ns . Durante 10 ns sólo puede ser ejecutado por un procesador y durante los otros 10 ns intervienen 5 procesadores. El valor de la f es 0.5.

Falso

$$T_s = 10 \text{ (parte no paralelizable)} * 1 + 10 * 5 \text{ (parte paralelizable)} = 60$$

$$f = 10 / 60 = 0.166$$

23. El tiempo de sobrecarga u overhead es un componente del tiempo de procesamiento paralelo junto con el tiempo de comunicación.

F

24. En un computador MIMD no se puede utilizar el modo de programación SPMD

F

25. En la asignación dinámica de tareas a procesos, distintas ejecuciones pueden asignar distintas tareas a un procesador.

V



**FORMACIÓN ONLINE Y
PRESENCIAL EN GRANADA**

**Clases de Inglés B1, B2, C1
DELF B1 y DELF B2 de Francés**

Centro Preparador
PREMIUM
Sede de Matriculaciones
Sede de Exámenes Oficiales

EXAMS
ANDALUCIA
English Qualification®


**PUERTA
REAL**
Academia de Enseñanza
academia-granada.es

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

**MEJORA TU
INGLÉS**
PLAZAS DISPONIBLES

**ASIGNATURAS DE UNIVERSIDAD:
HACEMOS GRUPOS
PARA CLASES DE APOYO**

PREGUNTAS VF T2 AC PROGRAMACION PARALELA

RESPUESTAS

TODOS LOS APARTADOS

WUOLAH

PREGUNTAS VF T2 AC PROGRAMACION PARALELA

1. Herramientas, estilos, estructuras en programación paralela Respuestas (ver antes el pdf de preguntas)

1. En programación paralela, el programador es siempre el encargado de dividir el programa en tareas, agruparlas en threads, asignarlos a un núcleo físico y sincronizarlos.

f

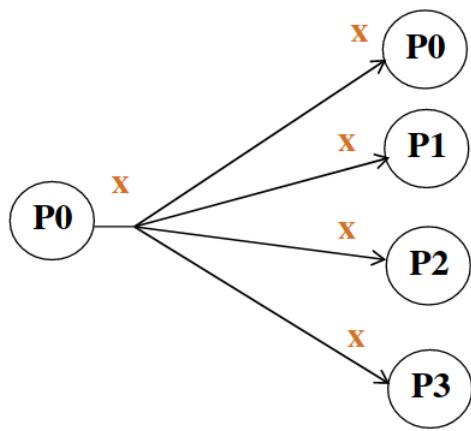
- División en unidades de cómputo independientes (tareas).
- Agrupación/asignación de tareas o carga de trabajo (código, datos) en procesos/threads.
- Asignación a procesadores/núcleos.
- Sincronización y comunicación.

Los debe abordar la herramienta de programación o el programador o ambos

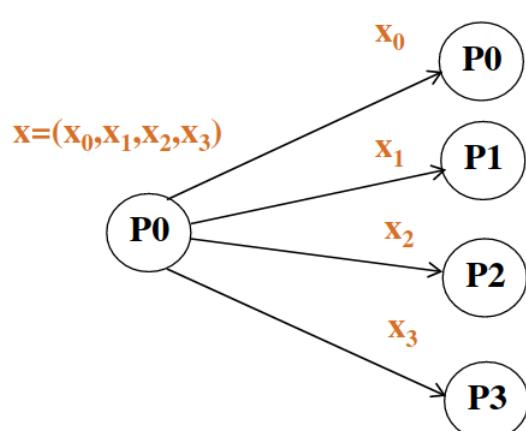
2. En una difusión, a cada variable se le asigna su valor correspondiente mientras que en una dispersion, todas las variables adquieren el mismo valor

f

Difusión (*broadcast*)



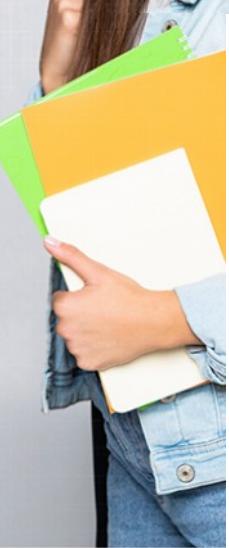
Dispersión (*scatter*)



MEJORA TU INGLÉS

PLAZAS DISPONIBLES

ASIGNATURAS DE UNIVERSIDAD:
HACEMOS GRUPOS
PARA CLASES DE APOYO



FORMACIÓN ONLINE Y
PRESENCIAL EN GRANADA

Clases de Inglés B1, B2, C1
DELF B1 y DELF B2 de Francés

Centro Preparador
PREMIUM

Sede de Matriculaciones
Sede de Exámenes Oficiales

EXAMS
ANDALUCIA
English Qualification



PUERTA
REAL

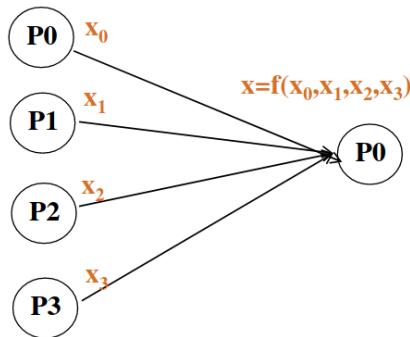
Academia de Enseñanza

academia-granada.es

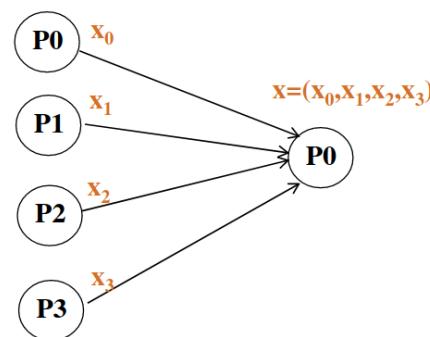
3. Una acumulación es una reducción donde el resultado es un vector. Para las reducciones, todos los resultados se combinan.

t

Reducción



Acumulación (*gather*)



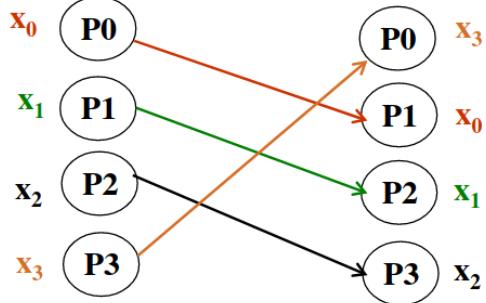
4. Los tipos de comunicación múltiple (uno a uno) son rotación y permutación

f

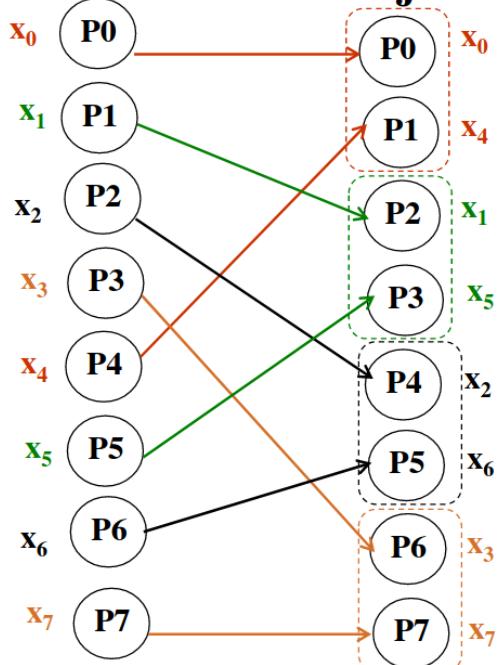
Comunicación múltiple uno-a-uno



Permutación rotación:

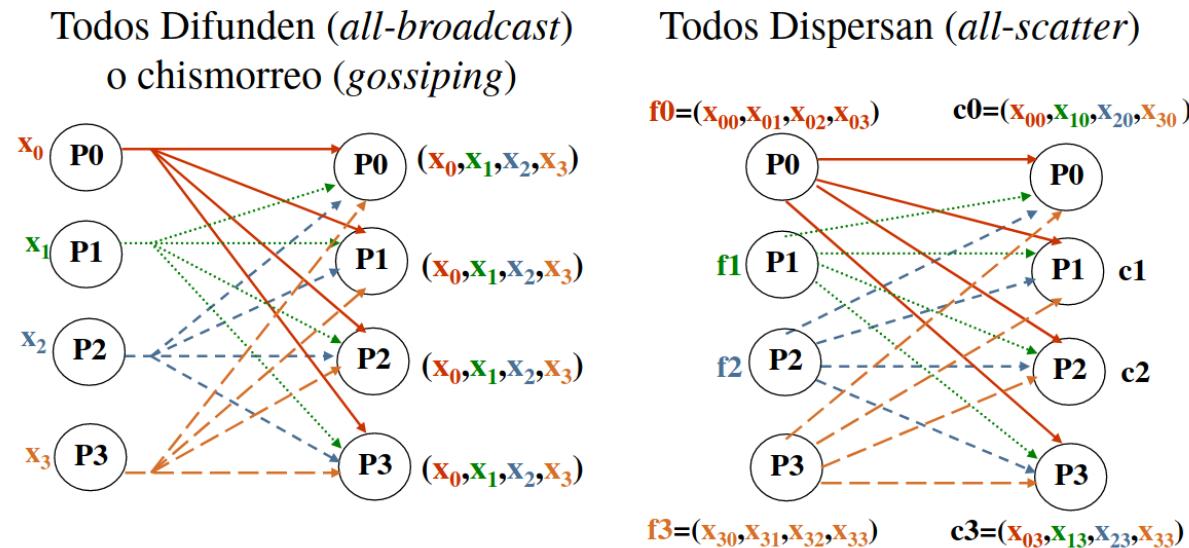


Permutación baraje-2:



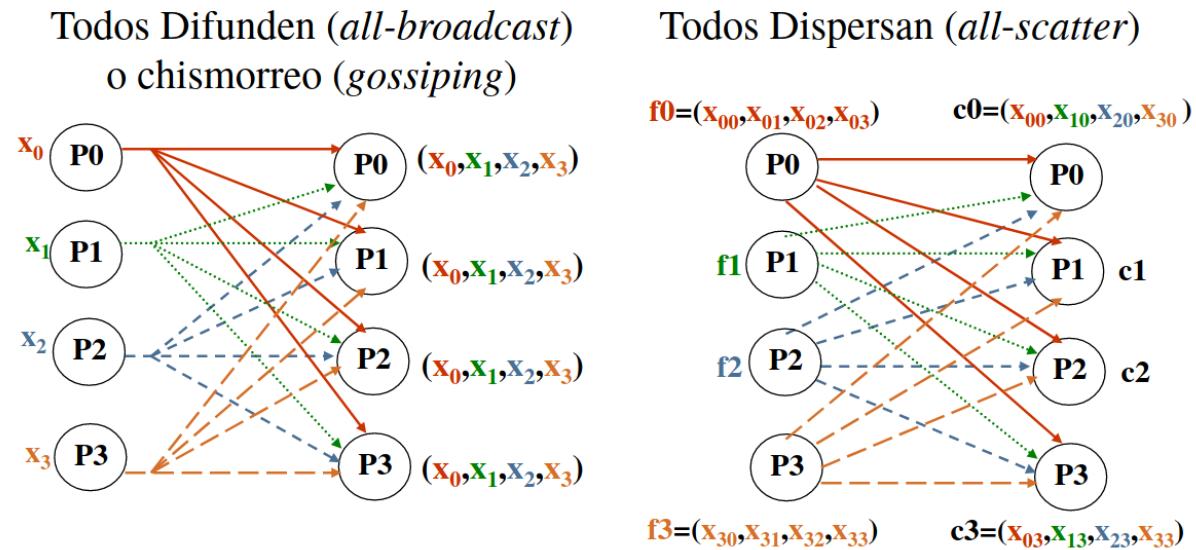
5. En un chismorreo (todos difunden) se pasa de varias variables que pueden ser diferentes a varios vectores que son todos iguales

t



6. En un all-scatter (todos dispersan) se pasa de varios vectores que son diferentes a varios vectores que son diferentes, pero que son una combinación de los anteriores

f

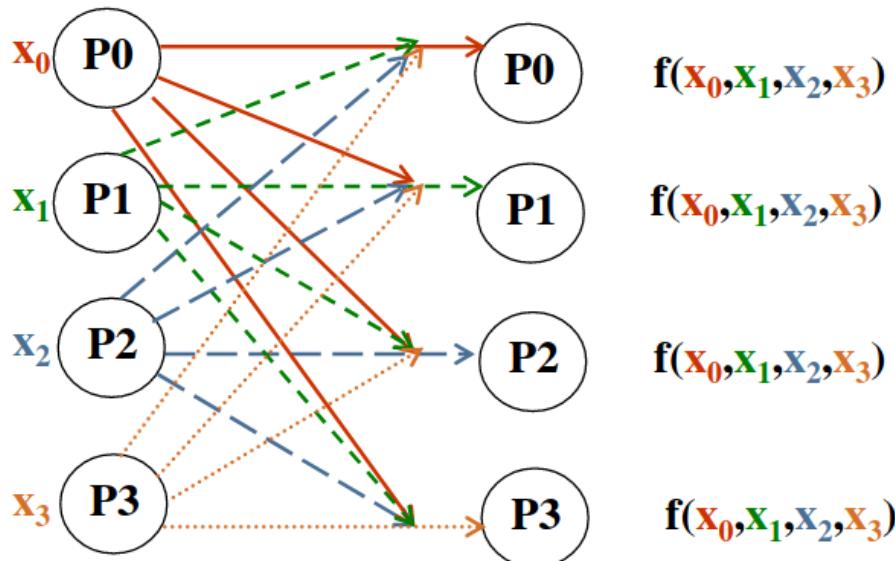




7. En todos combinan, a partir de un montón de variables se consigue un vector

f

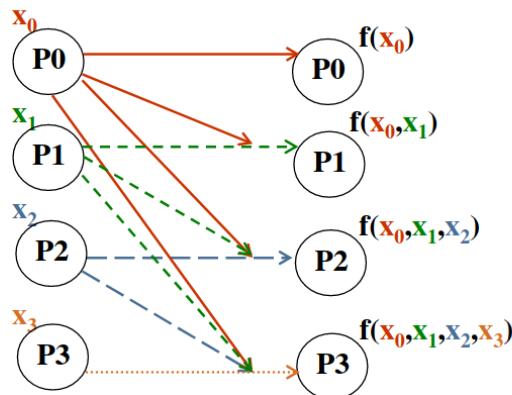
Todos combinan



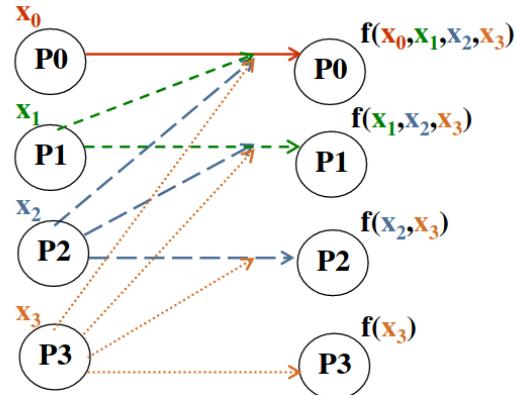
8. En los recorridos paralelos a partir de varios vectores de igual tamaño, se consiguen varios vectores de tamaños crecientes

f

Recorrido (scan) prefijo paralelo



Recorrido sufijo paralelo



9. En OpenMP podemos hacer uso de la comunicación colectiva para difusión con la directiva reduction

f

Es para todos a uno

Uno-a-todos	Difusión (Seminario pract. 2)	✓ Cláusula firstprivate (desde thread 0) ✓ Directiva single con cláusula copyprivate ✓ Directiva threadprivate y uso de cláusula copyin en directiva parallel (desde thread 0)
Todos-a-uno	Reducción (Seminario pract. 2)	✓ Cláusula reduction
Servicios compuestos	Barreras (Seminario pract. 1)	✓ Directiva barrier

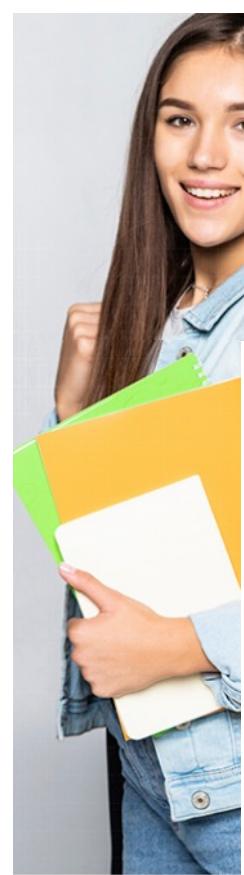
10. En OpenMP podemos obtener comunicación colectiva para servicios compuestos haciendo uso de la directiva barrier para crear los patrones de sincronización deseados

t

Ejemplo: comunicación colectiva en OpenMP

AC  ATC

Uno-a-todos	Difusión (Seminario pract. 2)	✓ Cláusula firstprivate (desde thread 0) ✓ Directiva single con cláusula copyprivate ✓ Directiva threadprivate y uso de cláusula copyin en directiva parallel (desde thread 0)
Todos-a-uno	Reducción (Seminario pract. 2)	✓ Cláusula reduction
Servicios compuestos	Barreras (Seminario pract. 1)	✓ Directiva barrier



FORMACIÓN ONLINE Y PRESENCIAL EN GRANADA

Clases de Inglés B1, B2, C1
DELF B1 y DELF B2 de Francés

Centro Preparador
PREMIUM
Sede de Matriculaciones
Sede de Exámenes Oficiales

EXAMS
ANDALUCIA
English Qualification®

**PUERTA
REAL**
Academia de Enseñanza
academia-granada.es

11. Con OpenMP no tenemos directivas para comunicación colectiva uno a uno y todos a todos

v

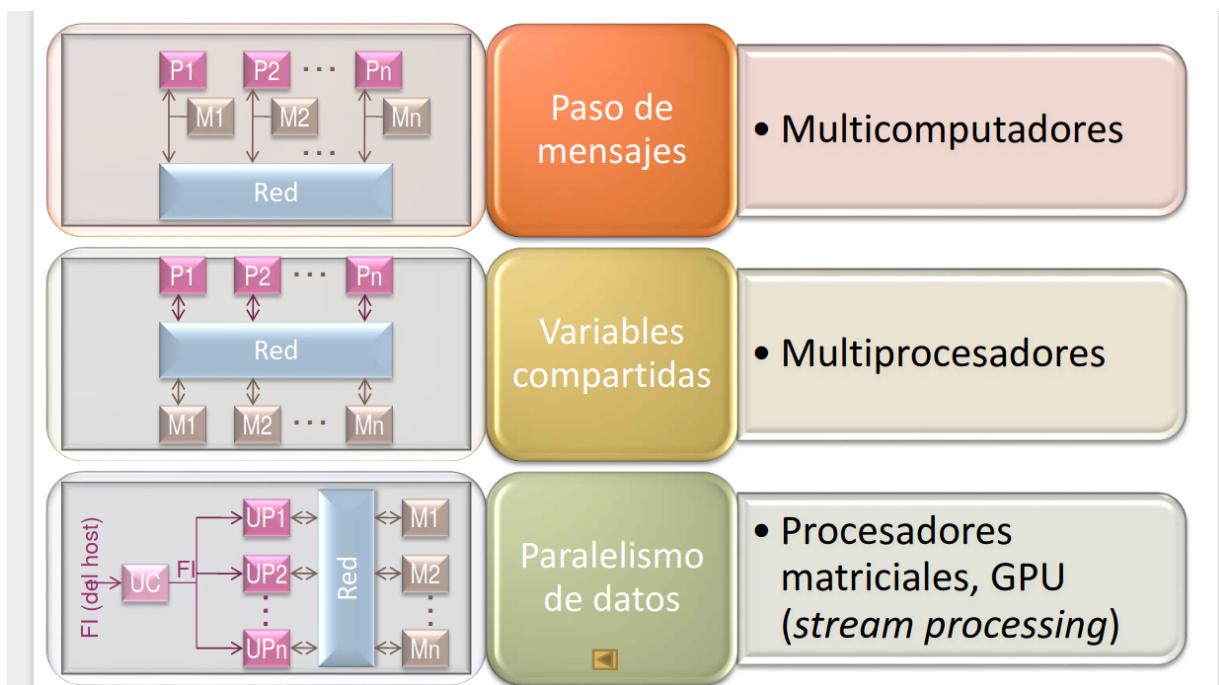
OpenMP

CETTCI

Uno-a-todos	Difusión (Seminario pract. 2)	✓ Cláusula <code>firstprivate</code> (desde thread 0) ✓ Directiva <code>single</code> con cláusula <code>copyprivate</code> ✓ Directiva <code>threadprivate</code> y uso de cláusula <code>copyin</code> en directiva <code>parallel</code> (desde thread 0)
Todos-a-uno	Reducción (Seminario pract. 2)	✓ Cláusula <code>reduction</code>
Servicios compuestos	Barreras (Seminario pract. 1)	✓ Directiva <code>barrier</code>

12. Los estilos de programación y sus arquitecturas paralelas correspondientes son paso de mensajes (multiprocesadores), variables compartidas (multicomputadores) y paralelismo de datos (procesadores matriciales/GPU)

f



13. La diferencia entre MPI (visto en la asignatura SCD) y OMP (visto en la asignatura AC) es que MPI es una biblioteca de funciones para paso de mensajes y OMP son directivas de compilador para programación paralela por variables compartidas.

V

- Paso de mensajes (*message passing*)
 - Lenguajes de programación: Ada, Occam
 - API (Bibliotecas de funciones): MPI, PVM
- Variables compartidas (*shared memory, shared variables*)
 - Lenguajes de programación: Ada, Java
 - API (directivas del compilador + funciones): OpenMP
 - API (Bibliotecas de funciones): POSIX Threads, shmem, Intel TBB (*Threading Building Blocks*)
- Paralelismo de datos (*data parallelism*)
 - Lenguajes de programación: HPF (*High Performance Fortran*), Fortran 95 (forall, operaciones con matrices/vectores)
 - API (funciones - *stream processing*): Nvidia CUDA

14. Algunas de las estructuras típicas de código paralelo son el modelo cliente servidor, divide y vencerás, máster-slave y segmentación.

v

Estructuras típicas de procesos/threads/tareas en código paralelo

Descomposición de dominio o descomposición de datos

cliente/servidor

Divide y vencerás o descomposición recursiva

Segmentación o flujo de datos

Master-Slave, o granja de tareas

PREGUNTAS VF T2 AC

PROGRAMACION PARALELA

2. Proceso de paralelización

Respuestas

(ver antes el pdf de preguntas)

1. La asignación dinámica de tareas a procesos se basa en que el programador pueda asignar distintas tareas a un procesador o core

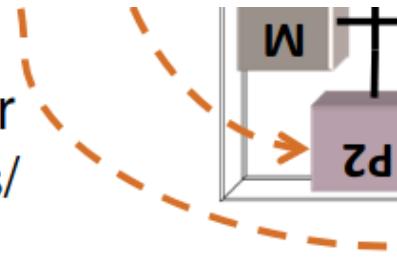
f

- Tipos de asignación:
 - Estática
 - Está determinado qué tarea va a realizar cada procesador o core
 - Explícita: programador
 - Implícita: herramienta de programación al generar el código ejecutable
 - Dinámica (en tiempo de ejecución)
 - Distintas ejecuciones pueden asignar distintas tareas a un procesador o core
 - Explícita: el programador
 - Implícita: herramienta de programación al generar el código ejecutable

2. El mapeo es asignar threads a cores/procesadores

v

Mapeo: asignar threads a cores/
procesadores



3. Para indicar asignación estática del parallelismo con OpenMP se utiliza la cláusula schedule(static)

t

```
Func1 () { ... }
Func2 () { ...
    #pragma omp parallel for \
                schedule(static)
    for (i=0;i<N;i++) {
        código para i
    }
...
}
Func3 () { ... }
Main () {
```





FORMACIÓN ONLINE Y PRESENCIAL EN GRANADA

Clases de Inglés B1, B2, C1
DELF B1 y DELF B2 de Francés

Centro Preparador
PREMIUM
Sede de Matriculaciones
Sede de Exámenes Oficiales

EXAMS
ANDALUCIA
English Qualification™



**PUERTA
REAL**
Academia de Enseñanza
academia-granada.es

4. Cuando la asignación de threads es dinámica, no se puede hacer explícita

f

AC ATC

➤ Asignación **dinámica** y explícita de las iteraciones de un bucle:

Bucle

```
for (i=0;i<Iter;i++) {  
    código para i  
}
```

```
lock(k);  
    n=i; i=i+1;  
unlock(k);  
while (n<Iter) {  
  
    código para n  
  
    lock(k);  
        n=i; i=i+1;  
    unlock(k);  
}
```



5. El SO operativo se encarga del mapeo. El programador no puede influir.

f

AC NV MYC

- Normalmente se deja al SO el mapeo de threads (*light process*)
- Lo puede hacer el entorno o sistema en tiempo de ejecución (*runtime system*) de la herramienta de programación
- El programador puede influir

PREGUNTAS VF T2 AC PROGRAMACION PARALELA

3. Evaluación de prestaciones en procesamiento paralelo Respuestas (ver antes el pdf de preguntas)

1. La ley de Amdahl es la siguiente

$$= \frac{p}{1 + f(p - 1)}$$

donde:

- p es la fracción de tiempo donde no se aplica la mejora
- f es el incremento si se aplica la mejora todo el tiempo

f

$$S(p) = \frac{T_s}{T_p(p)} \leq \frac{T_s}{f \cdot T_s + \frac{(1-f) \cdot T_s}{p}} = \frac{p}{1 + f(p - 1)} \rightarrow \frac{1}{f} (p \rightarrow \infty)$$

- S : Incremento en velocidad que se consigue al aplicar una mejora. (paralelismo)
- p : Incremento en velocidad máximo que se puede conseguir si se aplica la mejora todo el tiempo. (número de procesadores)
- f : fracción de tiempo en el que no se puede aplicar la mejora. (fracción de t. no paralelizable)

2. Con un estudio de escalabilidad se pretende evaluar en qué medida un programa podría llegar a comunicarse con otros procesos.

f

Con un estudio de escalabilidad se pretende evaluar en qué medida aumentan las prestaciones de un código paralelo conforme se incrementa el número de procesadores usados en la ejecución. El incremento o **ganancia en prestaciones**, también denominado ganancia en velocidad (*speed-up*), que se obtiene utilizando p procesadores se calcula dividiendo el tiempo de ejecución secuencial T_s por el tiempo de ejecución en paralelo usando los p procesadores $T_p(p)$:

$$S(p) = \frac{\text{Prestaciones}(p)}{\text{Prestaciones}(1)} = \frac{T_s}{T_p(p)} \quad (1)$$

3. La ley de Amdahl solo se puede aplicar cuando hablamos de programación paralela

f

(Capítulo 1), se puede usar en cualquier sistema al que se aplica una mejora: la ganancia en prestaciones que se puede conseguir aplicando una mejora a un sistema está limitada por la parte (fracción) del sistema que no usa la mejora.

4. La ley de Gustafson se usa cuando se emplea paralelización para mejorar la calidad del resultado, no disminuir el tiempo de ejecución.

t

Esta expresión se denomina **ganancia escalable** en J. L. Gustafson, 1988. Mientras que Gene M. Amdahl asume que el tiempo de ejecución secuencial (o tamaño del problema) se mantiene constante conforme se incrementa el número de procesadores, concluyendo que la ganancia en prestaciones está limitada debido a la parte del código no paralelizable, John L. Gustafson mantiene constante el tiempo de ejecución paralelo y muestra que la ganancia puede crecer con pendiente constante conforme se incrementa el número de procesadores.

5. Cuantos más procesadores se utilicen, mayor será el incremento en la velocidad

f

La escalabilidad para este caso ideal es una línea recta ((3) en Figura 5 y Figura 6)). En la práctica la curva de escalabilidad no será una línea recta, aunque podrá parecerlo para un rango de procesadores (desde 0 a un valor). Por lo general, la ganancia estará por debajo de la ideal, porque puede haber (ver Figura 6):

1. una fracción f de código no paralelizable,
2. un reparto no equilibrado de trabajo, y/o
3. una sobrecarga no despreciable.



Examen BP2

Universidad de Granada - Doble Grado en Ingeniería Informática y
Matemáticas
Arquitectura de Computadores



Estudiante: 75925602
75925602A Luque Framit, Rafael



Inicio: Hoy, viernes, 09:40:03

Final: Hoy, viernes, 09:54:11

Preguntas: 10

Respuestas válidas:

Puntuación:

Nota:

1 Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
```

Elección única

```

sumalocal = 0;
#pragma omp for
for (i=0; i<n; i++)
    sumalocal += a[i];
#pragma omp barrier
#pragma omp critical
    suma = suma + sumalocal;
#pragma omp barrier
#pragma omp single
    printf("La suma es =%d\n", suma)

```

5 Usuário Professores

- a) El valor de suma que se imprime sería correcto si cambiamos `private(sumalocal)` por `private(sumalocal, suma)`
 - b) Todas las demás respuestas son incorrectas
 - c) El valor de suma que se imprime no es siempre correcto
 - d) Uno de los `#pragma omp barrier`s es innecesario

2 ¿Existe algún problema que impida compilar el siguiente código?

Eleción única int n = 1;

```
#pragma omp parallel for default(none)
for (int i = 0; i < 5; ++i)
    n += i;
```

Usuario Profesores

- a) Ninguna otra respuesta es correcta
- b) Existe una condición de carrera
- c) El ámbito de la variable *i* no está definido
- d) El ámbito de la variable *n* no está definido.

3

Elección única

¿Cuál de los siguientes fragmentos de código tardará menos en calcular la sumatoria de los primeros $N = 2^{25}$ números impares en una máquina con 4 procesadores?

Usuario Profesores

- a) long sum = 0;

```
#pragma omp parallel sections
{
    #pragma omp section
    for (long i = 1; i < N; i += 4)
        #pragma omp atomic
        sum += i;
    #pragma omp section
    for (long i = 3; i < N; i += 4)
        #pragma omp atomic
        sum += i;
}
```
- b) long sum = 0;

```
#pragma omp parallel for
for (long i = 1; i < N; i += 2)
    #pragma omp atomic
    sum += i;
```
- c) long sum = 0;

```
#pragma omp parallel sections
{
    #pragma omp section
    for (long i = 1; i < N; i += 2)
        #pragma omp atomic
        sum += i;
}
```
- d) long sum = 0;

```
#pragma omp parallel
{
    long p = 0;
    #pragma omp for
    for (long i = 1; i < N; i += 2)
        p += i;
    #pragma omp atomic
    sum += p;
}
```

4

¿Qué valdrá la variable *n* después de la ejecución del siguiente código?

Elección única

```
int n = -1;
```

```
"-----"
```

```

#pragma omp parallel for lastprivate(n)
for (int i = 0; i < 4; ++i)
    #pragma omp atomic
        n += i;

```

Usuario Profesores

- a) 5
- b) 3
- c) Dependerá del número de hebras
- d) 2

5 ¿Cuál será el valor de n tras ejecutar el siguiente código?

Elección única

```

int i, n=2;
#pragma omp parallel shared(n) private(i)
for(i=0; i < 4; i++){
    #pragma omp single
    {
        n += i;
    }
}

```

Usuario Profesores

- a) 8
- b) 16
- c) 2
- d) Indeterminado

6 ¿Cuánto vale n al final?

Elección única

```

int n = 1;
#pragma omp parallel for reduction(*:n)
for (int i = n; i < 5; ++i)
    n *= i;
return n;

```

Usuario Profesores

- a) 1
- b) 6
- c) 0
- d) 24

7 ¿Cuál de las siguientes directivas no admite ninguna cláusula?

Elección única

Usuario Profesores

- a) parallel
- b) single
- c) Todas las directivas admiten alguna cláusula
- d) critical

8 ¿Cuál será el valor de n tras ejecutar el siguiente código?

Elección única

```

int n = 1;
#pragma omp parallel for lastprivate(n)
for (int i = 0; i < 5; ++i)

```

```
n = i;  
return n;
```

Usuario Profesores

- a) 4
- b) Indeterminado
- c) 5
- d) 1

9

¿Cuánto vale n tras ejecutar el siguiente código?

Elección única

```
int n = -1;  
#pragma omp parallel for shared(n)  
for (int i = 0; i < 4; ++i)  
    #pragma omp atomic  
    n += i;  
return n;
```

Usuario Profesores

- a) 6
- b) No se puede calcular con esta información
- c) -1
- d) 5

10

Asumiendo que $v2$ es de dimensión N y que todos sus elementos están inicializados a cero, ¿cuál de los siguientes códigos calcula de forma correcta el producto de la matriz m (dimensión $N \times N$) por el vector $v1$ (dimensión N) paralelizando el bucle que recorre las columnas?

Elección única

Usuario Profesores

- a)

```
#pragma omp parallel for private(j)  
for(i=0;i<N;i++){  
    for(j=0;j<N;j++){  
        v2[i] += m[i][j]*v1[j];  
    }  
}
```
- b)

```
#pragma omp parallel private(i)  
for(i=0;i<N;i++){  
    #pragma omp for reduction(+:v2[i])  
    for(j=0;j<N;j++){  
        v2[i] += m[i][j]*v1[j];  
    }  
}
```
- c)

```
#pragma omp parallel private(i,j)  
for(i=0;i<N;i++){  
    for(j=0;j<N;j++){  
        v2[i] += m[i][j]*v1[j];  
    }  
}
```
- d)

```
#pragma omp parallel private(j)  
for(i=0;i<N;i++){  
    #pragma omp for reduction(+:v2[i])  
    for(j=0;j<N;j++){  
        v2[i] += m[i][j]*v1[j];  
    }  
}
```

}



Examen BP2



Universidad de Granada - Doble Grado en Ingeniería Informática y
Matemáticas
Arquitectura de Computadores



Estudiante: 77034961 Córdoba Martínez, Rafael



Inicio: Hoy, viernes, 09:40:08

Final: Hoy, viernes, 09:54:03

Preguntas: 10

Respuestas

válidas:

Puntuación:

Nota:

1 ¿Cuánto vale n al final?

Elección única

```
int n = 1;
#pragma omp parallel
{
    int p = 0;
    #pragma omp single copyprivate(p)
    p = 2;
    #pragma omp for reduction(+:n)
    for (int i = 0; i < 5; ++i)
        n += p;
}
return n;
```

Usuario Profesores

- a) 5
- b) 10
- c) 1
- d) 11

2 Si se quiere difundir el valor de una variable inicializada por una hebra en una directiva single a las variables del mismo nombre del resto de hebras. ¿Cuál sería la cláusula que se tendría que utilizar?

Elección única

Usuario Profesores

- a) copyprivate
- b) Ninguna de las anteriores
- c) firstprivate
- d) lastprivate

3 ¿Cuál es la única directiva con la que se puede usar la cláusula `copyprivate`?

Elección única



Usuario Profesores

- a) master
- b) Se puede usar con más de una directiva
- c) atomic
- d) single

4 ¿Cuál de los siguientes fragmentos de código tardará menos en calcular la sumatoria de los primeros $N = 2^{25}$ números impares en una máquina con 4 procesadores?

Elección única

Usuario Profesores

- a) long sum = 0;
`#pragma omp parallel`
`{`
 `long p = 0;`
`#pragma omp for`
 `for (long i = 1; i < N; i += 2)`
 `p += i;`
`#pragma omp atomic`
 `sum += p;`
`}`
- b) long sum = 0;
`#pragma omp parallel sections`
`{`
 `#pragma omp section`
 `for (long i = 1; i < N; i += 4)`
 `#pragma omp atomic`
 `sum += i;`
 `#pragma omp section`
 `for (long i = 3; i < N; i += 4)`
 `#pragma omp atomic`
 `sum += i;`
`}`
- c) long sum = 0;
`#pragma omp parallel for`
`for (long i = 1; i < N; i += 2)`
 `#pragma omp atomic`
 `sum += i;`
- d) long sum = 0;
`#pragma omp parallel sections`
`{`
 `#pragma omp section`
 `for (long i = 1; i < N; i += 2)`
 `#pragma omp atomic`
 `sum += i;`
`}`

5 ¿Cuánto vale `ret` al final?

Elección única

```
int i, n = 6, ret = 1;
#pragma omp parallel reduction(+:ret) private(i)
for (i=omp_get_thread_num(); i<n; i+=omp_get_num_threads())
    ret += i;
return ret;
```

Usuario Profesores

- a) 1
- b) 16
- c) Ninguna de las otras respuestas es correcta
- d) 15

6

Elección única

Asumiendo que $v2$ es de dimensión N y que todos sus elementos están inicializados a cero, ¿cuál de los siguientes códigos calcula de forma correcta el producto de la matriz m (dimensión NxN) por el vector $v1$ (dimensión N) paralelizando el bucle que recorre las columnas?

Usuario Profesores

- a)

```
#pragma omp parallel private(j)
for(i=0;i<N;i++){
    #pragma omp for reduction(+:v2[i])
    for(j=0;j<N;j++){
        v2[i] += m[i][j]*v1[j];
    }
}
```
- b)

```
#pragma omp parallel for private(j)
for(i=0;i<N;i++){
    for(j=0;j<N;j++){
        v2[i] += m[i][j]*v1[j];
    }
}
```
- c)

```
#pragma omp parallel private(i,j)
for(i=0;i<N;i++){
    for(j=0;j<N;j++){
        v2[i] += m[i][j]*v1[j];
    }
}
```
- d)

```
#pragma omp parallel private(i)
for(i=0;i<N;i++){
    #pragma omp for reduction(+:v2[i])
    for(j=0;j<N;j++){
        v2[i] += m[i][j]*v1[j];
    }
}
```

7

Elección única

Indica cual será el valor de la variable n al final de la ejecución del siguiente código:

```
int n = 0;
#pragma omp parallel for reduction(*:n)
for (int i = n; i < size; ++i)
    n *= i;
```

Usuario Profesores

- a) 0
- b) El valor de n será igual a $size - 1$.
- c) Ninguna respuesta es correcta.
- d) Dependerá del valor de la variable $size$.

8

¿Cuál será el valor de n tras ejecutar el siguiente código?

Elección única

```
int i, n=2;
#pragma omp parallel shared(n) private(i)
for(i=0; i < 4; i++){
    #pragma omp single
    {
        n += i;
    }
}
```

Usuario Profesores

- a) 16
- b) 2
- c) 8
- d) Indeterminado

9

Elección única

Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
{
    sumalocal = 0;
    #pragma omp for
    for (i=0; i<n; i++)
        sumalocal += a[i];
    #pragma omp barrier
    #pragma omp critical
        suma = suma + sumalocal;
    #pragma omp barrier
    #pragma omp single
        printf("La suma es =%d\n", suma);
}
```

Usuario Profesores

- a) Tendríamos el mismo comportamiento si cambiamos `critical` por `atomic`
- b) Tendríamos el mismo comportamiento si eliminamos los dos `#pragma omp barrier`
- c) El valor de `suma` que se imprime es correcto
- d) Todas las demás respuestas son incorrectas

10

Elección única

Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
{
    sumalocal = 0;
    #pragma omp for
    for (i=0; i<n; i++)
        sumalocal += a[i];
    #pragma omp barrier
    #pragma omp critical
        suma = suma + sumalocal;
    #pragma omp barrier
    #pragma omp single
        printf("La suma es =%d\n", suma);
}
```

Usuario Profesores

- a) Uno de los `#pragma omp barrier` es innecesario
- b) El valor de `suma` que se imprime no es siempre correcto

-  c) Todas las demás respuestas son incorrectas
-  d) El valor de suma que se imprime sería correcto si cambiamos `private(sumalocal)` por `private(sumalocal, suma)`



Examen BP2

Universidad de Granada - Doble Grado en Ingeniería Informática y Matemáticas
Arquitectura de Computadores



Estudiante: 44669141 Martinez Diaz, David



Inicio: Hoy, viernes, 09:40:09

Final: Hoy, viernes, 09:52:03

Preguntas: 10

Respuestas

válidas:

Puntuación:

Nota:

1

Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

```
#pragma omp parallel private(sumalocal)
{
```

```
    sumalocal = 0;
    #pragma omp for
    for (i=0; i<n; i++)
        sumalocal += a[i];
    #pragma omp barrier
    #pragma omp critical
        suma = suma + sumalocal;
    #pragma omp barrier
    #pragma omp single
        printf("La suma es=%d\n", suma);
}
```

Elección única Usuario Profesores

- a) Uno de los `#pragma omp barrier` es innecesario
- b) El valor de suma que se imprime no es siempre correcto
- c) Todas las demás respuestas son incorrectas
- d) El valor de suma que se imprime sería correcto si cambiamos `private(sumalocal)` por `private(sumalocal, suma)`

2

Asumiendo que `v2` es de dimensión `N` y que todos sus elementos están inicializados a cero, ¿cuál de los siguientes códigos calcula de forma correcta el producto de la matriz `m` (dimensión `NxN`) por el vector `v1` (dimensión `N`) paralelizando el bucle que recorre las columnas?

Elección única Usuario Profesores

- a) `#pragma omp parallel private(j)`
`for(i=0;i<N;i++){`
 `#pragma omp for reduction(+:v2[i])`
 `for(j=0;j<N;j++) j`

```

        v2[i] += m[i][j]*v1[j];
    }
}

⊗ b) #pragma omp parallel private(i)
    for(i=0;i<N;i++){
        #pragma omp for reduction(+:v2[i])
        for(j=0;j<N;j++){
            v2[i] += m[i][j]*v1[j];
        }
    }

⊗ c) #pragma omp parallel private(i,j)
    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
            v2[i] += m[i][j]*v1[j];
        }
    }

⊗ d) #pragma omp parallel for private(j)
    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
            v2[i] += m[i][j]*v1[j];
        }
    }
}

```

- 3** Analiza el siguiente código e indica qué nos dirá el compilador referido a las siguientes líneas de código.

Elección única

```

int i, n = 1;
#pragma omp parallel for default(none) private(i)
for (i = 0; i < 5; ++i)
    n += i;

```

Usuario Profesores

- ⊗** a) La variable *n* no está especificada como privada
- ⊗** b) La variable *i* no puede ser privada
- **⊗** c) La variable *n* no está especificada
- ⊗** d) Ninguna opción de las anteriores.

- 4** Indica si el valor que tomará la variable *v* al salir de la zona paralela será siempre el mismo para cualquier entorno de ejecución.

Elección única

```

int main() {
    int i, n = 7, v = 1;
    int a[n];
    for (i=0; i<n; i++)    a[i] = i;
    #pragma omp parallel for firstprivate(v) lastprivate(v)
    for (i=0; i<n; i++){
        v += a[i];
        printf("\nthread %d v=%d / ", omp_get_thread_num(), v);
    }
    printf("\nFuera de la construcción parallel for v=%d\n", v);
    return 0;
}

```

Usuario Profesores

- ⊗** a) No, porque al salir de la zona paralela la variable volverá a ser 1.
- **⊗** b) No, será diferente dependiendo del número de hebras que lo ejecuten.

- c) Si, será siempre el mismo, /
- d) Sí, será siempre el mismo, aunque cambie el número de hebras

5 ¿Cuánto vale n al final?

Elección única

```
int n = 1;
#pragma omp parallel
{
    int p = 0;
    #pragma omp single copyprivate(p)
        p = 2;
    #pragma omp for reduction(+:n)
    for (int i = 0; i < 5; ++i)
        n += p;
}
return n;
```

Usuario Profesores

- a) 1
- b) 11
- c) 10
- d) 5

6 ¿Existe algún problema que impida compilar el siguiente código?

Elección única

```
int n = 1;
#pragma omp parallel for default(none)
for (int i = 0; i < 5; ++i)
    n += i;
```

Usuario Profesores

- a) Existe una condición de carrera
- b) El ámbito de la variable i no está definido
- c) El ámbito de la variable n no está definido.
- d) Ninguna otra respuesta es correcta

7 Sobre el código que aparece a continuación, ¿qué afirmación es correcta?

Elección única

```
#pragma omp parallel private(sumalocal)
{
    sumalocal = 0;
    #pragma omp for
    for (i=0; i<n; i++)
        sumalocal += a[i];
    #pragma omp atomic
        suma += sumalocal;
    #pragma omp single
        printf("La suma es =%d\n", suma);
}
```

Usuario Profesores

- a) Todas las demás respuestas son incorrectas
- b) Tendríamos el mismo comportamiento si cambiamos atomic por critical
- c) La hebra master imprime el valor de suma
- d) El valor de suma que se imprime es siempre el correcto

8 ¿Cuál de los siguientes fragmentos de código funciona de manera incorrecta?

Elección única

Usuario Profesores

- a) #pragma omp parallel for

-  a) `#pragma omp parallel for`
`for (long k = 0; k < 100; k++){`
 `x = array[k];`
 `array[k] = x+5;`
`}`
-  b) Todos los códigos funcionan de manera correcta
-  c) `#pragma omp parallel for private(x)`
`for (long k = 0; k < 100; k++){`
 `x = array[k];`
 `array[k] = x+5;`
`}`
-  d) `#pragma omp parallel for`
`for (long k = 0; k < 100; k++){`
 `int x;`
 `x = array[k];`
 `array[k] = x+5;`
`}`

9

¿Cuánto vale n al final?

Elección única

```
int n = 1;
#pragma omp parallel for reduction(*:n)
for (int i = n; i < 5; ++i)
    n *= i;
return n;
```

Usuario Profesores

-  a) 6
-  b) 1
-  c) 0
-  d) 24

10

Indica qué valor tendrá la variable `ret` después de ejecutar la siguiente reducción cuando se ejecuta con 4 hebras:

Elección única

```
int i, n = 6, ret = 1;
#pragma omp parallel reduction(+:ret)
for (i=omp_get_thread_num(); i<omp_get_max_threads(); i+=omp_get_num_threads())
    ret += i;
```

Usuario Profesores

-  a) 7
-  b) 3
-  c) 5
-  d) El valor de `ret` será indeterminado porque existe una condición de carrera