

Multiprocesadores 3º Informática

Apellidos y nombre:

1ª convocatoria, 15 junio 2020

--	--	--	--	--	--	--	--

Problema 1

Rendimiento de un código vectorizable en ZV

20 puntos

Estudiad el rendimiento de este código en un procesador vectorial ZV con 16 bancos de memoria. Cada banco contiene palabras de 8 bytes. Restricción: las lecturas con *stride* que tardan más de un ciclo en suministrar cada elemento NO encadenan.

R _A = 0x1000, R _B = 0x8000, R _C = 0xF000, R _S = 64, F _α = 3.1416			
1\$:	LV	V1, R _A	sección residual al final
	LVWS	V2, R _B , R _S	
	MULSV	V3, V2, F _α	
	ADDV	V4, V3, V1	
	SV	R _C , V4	

código de seccionado	; 5 instrucciones escalares independientes ; salto no retardado, @dst se calcula en D1
-------------------------	--

a) Calcula los ciclos por elemento de las instrucciones LV y LVWS.

LV	V1, R _A	BDDLX	4	X
		12 1		2
LVWS	V2, R _B , R _S			
MULSV	V3, V2, F _α			
ADDV	V4, V3, V1			
SV	R _C , V4			

b) Diagrama de tiempos para la primera sección y media, señalando el principio de los encadenamientos y las detenciones por cualquier causa. Anota el ciclo de finalización de cada instrucción vectorial y de los puntos de interés.

c) Calcula C_n para n ≤ 64.

C_n =

d) Calcula C_n para n >> 64. Marca el camino crítico en el diagrama.

C_n =

e) Calcula R_∞

R_∞ =

f) Calcula N_{1/2}

Problema 2

Dependencias, vectorizar y paralelizar

16 puntos

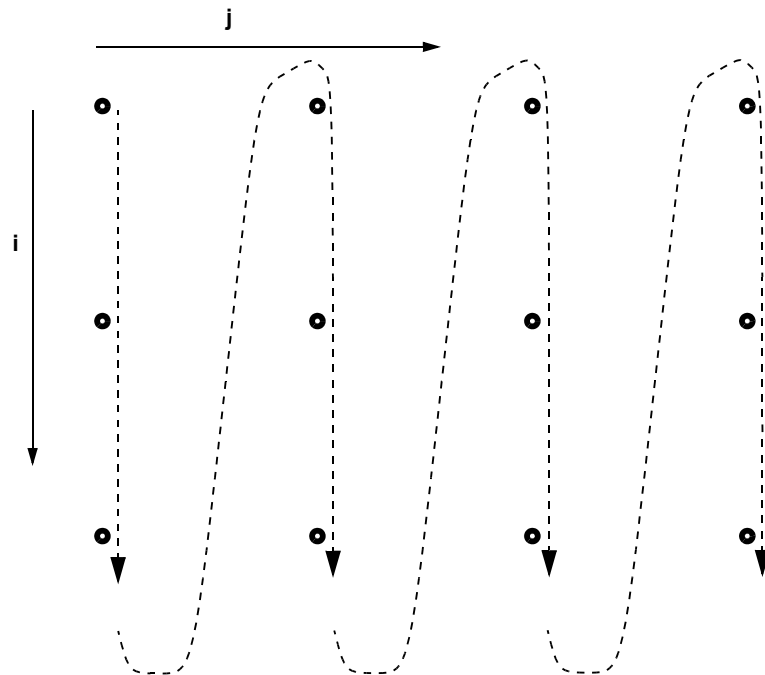
Dado el siguiente código:

```

real*8 A( , ), B( , )
...
do j= 1,N-1
  do i= 1,M-1
S1:    A(i+1,j+1)= A(i,j+1) + B(i,j+1)
S2:    B(i,j)= C(i,j) + 2.0
  enddo
enddo

```

a) Dibuja las dependencias sobre el diagrama de recorrido del espacio de iteraciones:



Queremos ejecutar este código en un multiprocesador de memoria compartida, en el que los procesadores tienen capacidad de ejecución vectorial.

b) Diagrama de dependencias..

dist en j dist en i

--	--

S1

S2

c) Indica si son posibles los modos de ejecución siguientes, añadiendo una explicación muy corta. SEC, VEC y PAR significan ejecución secuencial, vectorial y paralela.

$\left[\begin{array}{l} j \text{ PAR} \\ i \text{ SEC} \end{array} \right]$
 sí/no ☐

$\left[\begin{array}{l} j \text{ SEC} \\ i \text{ VEC} \end{array} \right]$
 sí/no ☐

$\left[\begin{array}{l} j \text{ PAR} \\ i \text{ VEC} \end{array} \right]$
 sí/no ☐

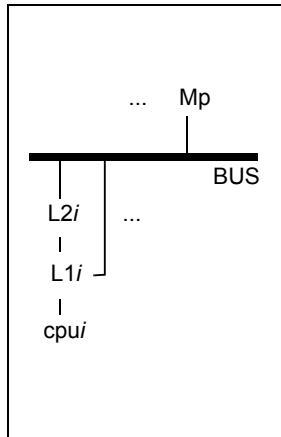
d) ¿Es correcto intercambiar los bucles? Explicación muy corta.

e) Propón alguna transformación que disminuya el tiempo de ejecución del código original en un multiprocesador vectorial.

Problema 3

Coherencia de dos niveles exclusivos de cache en bus compartido

35 puntos



Consideramos un multiprocesador de memoria compartida basado en bus. Cada procesador tiene dos niveles privados de cache cuyos contenidos están en exclusión: $L2 \cap L1 = \emptyset$.

L2, la memoria cache de segundo nivel:

- mantiene coherencia mediante observación e invalidación
- protocolo MSI, con la definición de más abajo
- recibe *todos* los bloques expulsados de L1

L1, la memoria cache de primer nivel:

- es de escritura retardada y carga bloque en caso de fallo
- Protocolo MSI convencional, sin observación de bus
- Los comandos que envía a L2, que actúa como intermediaria son: **rB**, **rBinv**, **inv**, **WBclean** y **WBdirty**.

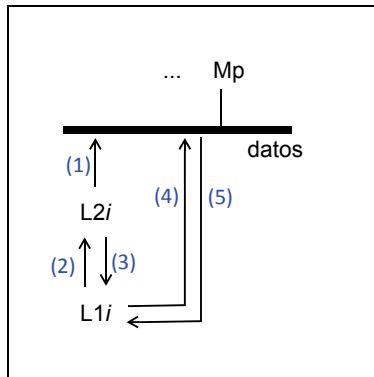
L2 filtra parte del tráfico de coherencia, de forma que L1 únicamente es molestada cuando no hay más remedio.

Vamos a centrarnos en el diseño del protocolo de coherencia del segundo nivel.

La definición de los estados **MSI** de la L2i es la siguiente:

- **I** Como siempre: el bloque x no está en el conjunto que toca.
- **M** ($1L2 \neq Mp$). Existe una sola copia sucia en todo el sistema, y es ésta.
- **S** ($L2i = Mp = n(L2j \oplus L1j)$, $n \geq 0$, $i \neq j$).
O sea, L2i tiene una copia limpia, que puede estar replicada, o no, en las caches de otros procesadores.
Por supuesto, si está en L2j no está en L1j, y viceversa.

a) Las líneas representan caminos de datos, por donde circulan bloques, las flechas el sentido. Escribe en cada número la condición necesaria para que ocurra ese movimiento de bloque.



- (1)

- (2)

- (3)

- (4)

- (5)

b) **Protocolo de L2**: transiciones y acciones resultantes de la interacción L1-L2. Especifica las cargas de bloques desde L1 mediante: **L2 + x**.

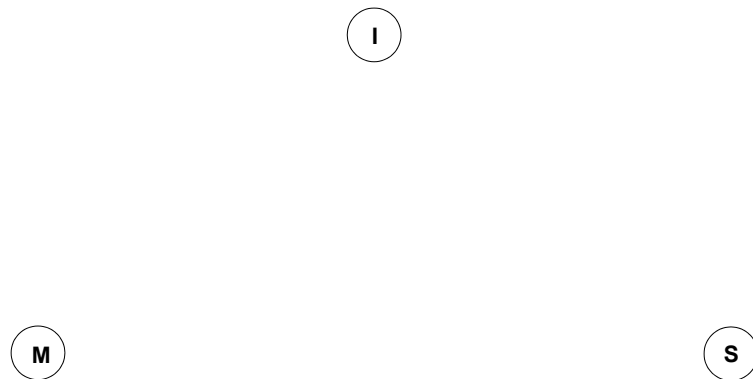


c) **Protocolo de L2**: transiciones y acciones resultantes de la observación del bus.

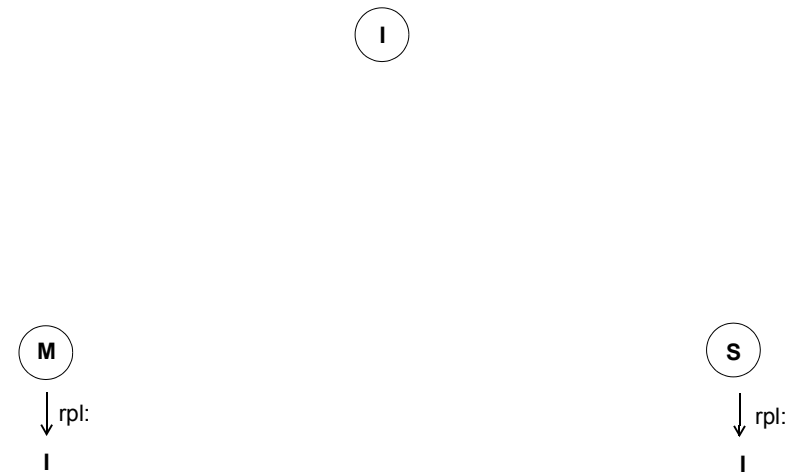


- d) L2 puede filtrar los comandos de coherencia que observa en el bus, evitando actividad en su L1, pero no siempre es posible ...
¿Cuándo L2 puede filtrar y cuándo debe molestar a su L1?

Protocolo de L1. Dibuja las transiciones y acciones en L1 debidas a comandos recibidos desde L2.
Una respuesta negativa desde L1 hacia L2 se representa como: **L2(nack)**.



- e) **Protocolo de L1.** Dibuja las transiciones y acciones en L1 debidas a comandos recibidos desde el procesador.



Problema 4

Sincronización

5 puntos

Escribe en ensamblador (RISC V) el código correspondiente a un semáforo multimarca, es decir, un semáforo que gestiona el acceso concurrente a una región crítica de hasta N hilos de ejecución. Implementa las funciones de `lock()` y `unlock()`. Asume unas primitivas de sincronización tipo load-link y store-conditional.

Problema 5

Prácticas Vectorización

14 puntos

Tenemos el siguiente código C:

```
[...]
for (i = 0; i < LEN; i++) {
    x[i] = alpha*x[i] + beta;
}
[...]
```

Al compilarlo con gcc 7.3, indicando soporte AVX512 con el flag `-mavx512vl`, obtenemos este código ensamblador:

```
.bucle:
    vmulpd  (%rax),%zmm3,%zmm0
    add     $0x40,%rax
    vaddpd  %zmm2,%zmm0,%zmm0
    vmovapd %zmm0,-0x40(%rax)
    cmp     %rax,%rbx
    jne     bucle
```

a) Responde las siguientes preguntas **con breve justificación**:

a1) ¿Cuántos bytes del vector `x[]` se procesan en una iteración del bucle ensamblador?

a2) ¿Con qué tipo de dato se está operando (int, float, double ...)?

a3) ¿Cuántos elementos del vector `x[]` se procesan en una iteración del bucle ensamblador?

- a4) Sabiendo que al inicio del bucle se realiza la siguiente inicialización:
%rbx = %rax + 0x4000
¿cuál es el valor de LEN?

- b) Medimos los siguientes tiempos para una ejecución completa de las versiones escalar y vectorial de este bucle en un Intel Xeon Gold 5120:

- Escalar: 814.2 ns - Vectorial: 188.9 ns

Calcula las siguientes métricas:

- b1) Aceleración (*speedup*) de la versión vectorial sobre la escalar.

- b2) Velocidad de la versión vectorial (GFLOPS).

Problema 6

Prácticas Paralelización

10 puntos

Tenemos el siguiente código C:

```
void
gauss_filter(matrix_t * restrict matrix_in, vector_t * restrict vector_in,
vector_t * restrict vector_out){
{
    [...]

    for (int i=0; i < width; i++) {

        for (int j=0; j < height; j++) {
            vector_out[i] += (matrix_in[i][j] * vector_in[j]) ;
        }
    }

    [...]
}
```

- a) Indica las opciones de compilación **estrictamente necesarias** en cc Sun que paraleliza de forma automática y que muestran en la salida estándar los bucles paralelizados.
- b) Añade las directivas OpenMP **estrictamente necesarias** para describir el paralelismo del bucle (sobre el propio código de arriba).
- c) Indica las opciones de compilación **estrictamente necesarias** para compilar el código resultante en el apartado anterior.