Arquitectura y Tecnología de Computadores. Curso 08/09

Arquitecturas Paralelas. **Practica VI**

RIESGOS EN PROCESADORES SEGMENTADOS (III)

Análisis de problemas en programas reales

1. Objetivo:

Análisis de la incidencia de los **riesgos estructurales**, los **riesgos por dependencias de datos** y los **riesgos de control** en pequeños programas reales, evaluando para ello el rendimiento del procesador segmentado MIPS R3000 mediante la ejecución tanto simulada como real de dichos programas.

2. Análisis a realizar:

Los análisis a realizar con objeto de poner de manifiesto la influencia de los riesgos aludidos sobre el rendimiento del procesador segmentado son los siguientes:

1. Analizar la **influencia de los problemas estructurales** sobre el ritmo de ejecución en el cauce, tanto los debidos a la escasez de recursos (vías de acceso a memoria), como los debidos a la falta de segmentación en alguna de sus unidades funcionales (unidades de punto flotante).

2. Analizar la **influencia de las dependencias de datos** sobre el ritmo de ejecución en el cauce (una de las instrucciones establece un nuevo valor del contenido de un registro y alguna de las posteriores utiliza el mismo como uno de sus operandos).

3. Analizar la **influencia de las instrucciones de control** sobre el ritmo de ejecución en el cauce (recordar que la instrucción de salto genera un hueco de retardo de salto que se tratará de rellenar de acuerdo con una determinada estrategia de planificación).

En las **dos practicas anteriores**, hemos realizado estos análisis utilizando como fuente pequeños **programas sintéticos** realizados para el estudio de cada tipo de riesgo concreto. En **esta práctica** realizaremos un estudio similar pero utilizando **tres pequeños programas reales**.

3. Pasos a seguir en la práctica:

La primera parte de la práctica se va llevar a cabo mediante el empleo de los dos simuladores de la arquitectura DLX utilizados en la practica 4**: *Windlx* y *DlxView***. Los ficheros fuente necesarios para la simulación pueden descargarse del sitio web de la asignatura.

- Edita el programa **suma1.s** (ver anexo I). Reflexiona primero sobre lo que hace el programa (teniendo en cuenta que lo que hace se repite tantas veces como indica el bucle) y después simula su comportamiento en *Windlx* con el ***forwarding deshabilitado***. ¿qué problemas aparecen?

- Habilita el *forwarding* y analiza el cambio de situación.

- Construye un nuevo programa **suma2.s**, a base de reordenar el código del programa anterior, de tal forma que desaparezcan los ciclos perdidos (manteniendo habilitado el *forwarding*). **IMPORTANTE:** Hay que tener mucho cuidado a la hora de reordenar el código, ya que una mala reordenación puede dar lugar a resultados incorrectos. Para comprobar que el código reordenado funciona igual que el código original, se puede consultar el valor de las variables en memoria después de la simulación, para comprobar que ambos programas escriben los mismos resultados (esto lo puedes hacer pinchando en el menú *Memory-Display* o pulsando la tecla F6).

- Repite los tres puntos anteriores con el programa **sum-res1.s** (ver anexo I).

- Repite lo mismo con el programa **daxpy1.s** (ver anexo I), configurando previamente la unidad de multiplicación flotante con latencia de 5 ciclos, hasta reducir la perdida de ciclos a 3 por iteración.

- Copia a tu cuenta en *centauro* o *lisa* los programas vía ftp (modo ascii). Repite las mismas simulaciones con *Dlxview* observando además como evoluciona el hardware del camino de datos enteros del procesador ¿qué diferencia de comportamiento observas en **daxpy1.s**? ¿a que crees que se debe?

- Para cada uno de los 3 programas anteriores, y teniendo en cuenta los ciclos consumidos, calcula:

  - Ganancia de la versión 1 con *forwarding* respecto a dicha versión sin *forwarding*
  - Ganancia de la versión 2 por ti construida respecto a la primera (ambas con *forwarding*)

- Valora en que medida has conseguido mejorar el rendimiento de los programas

La segunda parte de la práctica se va a realizar sobre la máquina ***oboe.epsig.uniovi.es***. Cada alumno utilizará la misma cuenta que haya utilizado en la practica anterior.

- Copia a tu carpeta los ficheros fuente **suma1.s**, **sum-res1.s** y **daxpy1.s** desde /usr1/users/suarez/prac6/, ya preparados para compilar en OBOE. Compila los programas sin optimización y mide sus tiempos de ejecución.

- Vuelve a compilar los programas, pero con optimización en el compilador, y mide sus tiempos de ejecución.

- Adapta también las nuevas versiones de los programas construidas por ti (**suma2**, **sum-res2** y **daxpy2**), compílalas incluyendo la directiva "**.set noreorder**", y mide sus tiempos de ejecución. Compilando estos fuentes con y sin optimización, los tiempos resultantes deberían ser iguales ¿por qué?

- Calcula las ganancias conseguidas en cada programa con la optimización por una parte y con tus versiones (sin optimización) por otra, respecto a las versiones originales sin optimizar. Valora tu eficiencia frente a la del compilador.

4. Trabajo voluntario a entregar:

- Resumen de los resultados relativos a tiempos consumidos por los programas y ganancias entre versiones, tanto los obtenidos por simulación como a partir de la ejecución sobre el procesador real. Utilizar para ello el modelo de tablas *Excel* del anexo II.

Anexo I: Programas reales de prueba

suma1.s	suma2.s

`.data 0`
`A: .word 0`
`B: .word 1`
`C: .word 2`
 	
`.text`
`main:`
`addi r30, r0, 10`
`nop`
`nop`
 	
`loop: subui r30, r30, 1`
`lw r1, B`
`lw r2, C`
`add r3, r1, r2`
`sw A, r3`
`bnez r30, loop`
`nop`
`trap #0`
	???

Suma de dos variables de memoria

sum-res1.s	sum-res2.s

`.data 0`
`A: .word 0`
`B: .word 1`
`C: .word 2`
`D: .word 0`
`E: .word 3`
`F: .word 4`
 	
`.text`
`main:`
`addi r30, r0, 10`
`nop`
`nop`
 	
`loop: subui r30, r30, 1`
`lw r1, B`
`lw r2, C`
`add r3, r1, r2`
`sw A, r3`
`lw r1, E`
`lw r2, F`
`sub r3, r1, r2`
`sw D, r3`
`bnez r30, loop`
`nop`
`trap #0`
	???

Suma y resta de variables de memoria

daxpy1.s	daxpy2.s

; DAXPY loop of length 100
 	
.data 0x1000
 	
.double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
.double 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19
.double 1.20, 1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29
.double 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.38, 1.39
.double 1.40, 1.41, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49
.double 1.50, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.57, 1.58, 1.59
.double 1.60, 1.61, 1.62, 1.63, 1.64, 1.65, 1.66, 1.67, 1.68, 1.69
.double 1.70, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79
.double 1.80, 1.81, 1.82, 1.83, 1.84, 1.85, 1.86, 1.87, 1.88, 1.89
.double 1.90, 1.91, 1.92, 1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99
 	
.global done
done: .double 0
 	
.data 0x2000
.double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
.double 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19
.double 2.20, 2.21, 2.22, 2.23, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29
.double 2.30, 2.31, 2.32, 2.33, 2.34, 2.35, 2.36, 2.37, 2.38, 2.39
.double 2.40, 2.41, 2.42, 2.43, 2.44, 2.45, 2.46, 2.47, 2.48, 2.49
.double 2.50, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.57, 2.58, 2.59
.double 2.60, 2.61, 2.62, 2.63, 2.64, 2.65, 2.66, 2.67, 2.68, 2.69
.double 2.70, 2.71, 2.72, 2.73, 2.74, 2.75, 2.76, 2.77, 2.78, 2.79
.double 2.80, 2.81, 2.82, 2.83, 2.84, 2.85, 2.86, 2.87, 2.88, 2.89
.double 2.90, 2.91, 2.92, 2.93, 2.94, 2.95, 2.96, 2.97, 2.98, 2.99
 	
dato: .double 3.14
 	
.text
main:
; put 3.14 into f0
ld f0, dato
 	
; put 0x1000 into r1
addi r1, r0, 0x1000
 	
; put 0x2000 into r2
addi r2, r0, 0x2000
 	
loop: ld f2, 0(r1)
multd f4, f2, f0
ld f6, 0(r2)
addd f6, f4, f6
sd 0(r2), f6
addi r1, r1, 8
addi r2, r2, 8
sge r3, r1, done
beqz r3, loop
nop
trap #0
	???

Programa **DAXPY** (Y=a*X+Y en doble precisión)

Anexo II: Resultados

Programa	Tiempos (seg./ciclos)			Comentarios (2 líneas máximo)		
	Oboe	WinDLX	DLXView	**Oboe**	**WinDLX**	**DLXView**
suma1						
suma1 (fw)	-		-			
suma1op		-	-			
suma2 (fw)						
sum-res1						
sum-res1 (fw)	-		-			
sum-res1op		-	-			
sum-res2 (fw)						
daxpy1						
daxpy1(fw)	-		-			
daxpy1op		-	-			
daxpy2 (fw)						

		suma	**sum-res**	**daxpy**
WinDLX	**G** (1fw/1)			
	G (2fw/1fw)			
DLXView	**G** (1fw/1)			
	G (2fw/1fw)			
Oboe	**G** (1op/1)			
	G (2/1)			