Nombre: Liang Liang Chen Xu Grupo: 12

Nombre:_____

Hoja de respuesta al Estudio Previo

1. Hacer "inlining"de una función significa:

   Es una optimización que consiste en reemplaza la llamada a una función por el cuerpo de ella misma eliminando la necesidad de invocar a la función en tiempo de ejecución, mejorando el rendimiento ya que evita los cambios de contexto, saltar a la ubicación de la función…

2. La opción específica de compilación de *gcc* que permite al compilador hacer "inlining"de todas las funciones simples es (especifica si se activa o no al activar la opción -O2). ¿Para qué sirve la opción -finline-limit?:

   La opción específica es –finline-functions y no se activa con el –O2. La opción –finline-limit establece un límite en el tamaño de las funciones que se consideran para el inlining.

3. Explica una forma práctica de saber si en un programa ensamblador existe la función "Pepito" y cómo averiguar si, además de existir, esa función es invocada o no:

   Hacer un grep –n "Pedrito" programa.s y ver si "Pedrito" y ver si aparece como tag de función(Pedrito:). Para ver si está invocada grep –n "call Pedrito" programa.s

4. El primer código ensamblador tiene:

   Instr. estáticas: 5 Instr. dinámicas: 5.143.533

   Si la ejecución tarda 12 ms y 15000000 de ciclos:

   MIPS: 428.628 IPC: 0.34 CPI: 2.92

   Frecuencia: 1.25 GHz

5. El segundo código (compilado con -O) tiene:

   Instr. estáticas: 4 Instr. dinámicas: 5.143.528

   Si la ejecución tarda 4 ms y 5000000 de ciclos:

   MIPS: 1285.89 **CPI:** 0.97 **Frecuencia:** 1.25 GHz

   Speedup: 3

   Las igualdades y diferencias observadas respecto al apartado anterior se deben a:

   En esta segunda versión observamos que se han reducido las instrucciones estáticas y dinámicas ligeramente. El tiempo de ejecución y los ciclos se han reducido en un 77%, de esta manera aumentando los MIPS. Al haber reducido los ciclos y el número de instrucciones hace que este disminuya. La frecuencia se mantiene y llegamos a la conclusión que el programa optimizado es 3x mas rápido que el original.

6. El programa total puede obtener un Speedup de:

   Si el código es instantáneo: 1.06 Si se compila con -O: 1.02

7. Una forma práctica para medir el rendimiento (MIPS e IPC) del programa en C que acabamos de ver es:

   En primer lugar podemos obtener el número de instrucciones a partir del comando $valgrind –tool=lackey ./miprograma, seguidamente con el programa tiempo.c podemos utilizar la función GetTime() para obtener el tiempo total en ms y con la librería cycle.h obtenemos el número de ciclos para poder calcular el IPC. De esta manera aplicamos la fórmula del Mips ya que disponemos de todas las variables y calculamos el IPC como num_instr/num_ciclos.

8. Dadas 5 ejecuciones de 10 ms, 8ms, 13 ms, 8ms y 2ms. Su media:

   Geométrica: 6.98 Aritmética: 8.2

   Descartando los valores extremos su media es:

   Geométrica: 8.61 Aritmética: 8.6

   Se observa que:

   Al descartar los valores extremos la media geométrica y aritmética no presentan casi diferencia y por lo tanto es bastante preciso.

Nombre:_____ Grupo:_____

Nombre:_____

Hoja de respuestas de la práctica

1. Instrucciones dinámicas del código de ejemplo `Simple.c`:

   Sin optimizar: [ ] Optimizado: [ ]

   Explicación de los resultados:

   _____

   _____

   _____

2. Rellena la siguiente tabla sobre el programa `Poker.c`:

	Tiempo	Ciclos	Instrucciones	MIPS	CPI	Frec.	Speedup
-O0							
-O2							
-O2 + “inlining”							

3. Rellena la siguiente tabla sobre la rutina `PierdeTiempo` (tened cuidado de no incluir el `printf` en los ciclos de la rutina) del programa `Poker.c`:

	Ciclos Programa	Ciclos rutina	% de Ciclos	Speedup teórico máximo
-O0				
-O2				

   Las variaciones entre las dos filas del apartado anterior se deben a:

   _____

   _____

   _____