Laboratorio 3 Organización de Computadores: Caché

Profesores: Felipe Garay, Erika Rosas, Nicolás Hidalgo
Ayudante: Francisco López, Pablo Ulloa
Email: {nombre}.{apellido}@usach.cl

13 de noviembre de 2016

El profesor y los ayudantes son sus clientes. Si algo no queda claro es su deber como ingeniero/a hacer las consultas correspondientes para desarrollar el proyecto de forma correcta.

Enunciado

Una de las formas de hacer los procesadores más rápidos es acercando la memoria principal del computador hacia el procesador mediante una versión más pequeña pero rápida llamada caché.

En este laboratorio analizaremos como distintas configuraciones de cache pueden modificar el desempeño de un programa y además como distintos algoritmos se comportan de distinta manera de acuerdo a su organización de la memoria.

Se debe implementar dos algoritmos de ordenamiento DISTINTOS a los que usted implementó en el laboratorio 1: uno recursivo y uno iterativo. Estos programas deben leer un archivo de texto y entregar uno nuevo ordenado. Los nombres de los archivos de texto se van a pasar por medio de argumentos en Mars de la forma: "java -jar Mars.jar PROGRAMA.asm pa ENTRADA.txt SALIDA.txt". Estos programas deben operar sobre listas enlazadas. Una vez obtenga los resultados elija el peor programa (justifique su criterio) y vuelva a implementarlo utilizando arreglos de memoria contigua. Una vez obtenga estos resultados trate de mejorar este último programa.

Una alternativas a las pruebas unitarias tradicionales, donde se tienen entradas y salidas pre establecidas, es el "fuzzy testing" que consiste en probar propiedades en vez de casos. Por ejemplo se puede tener una función que invierte una lista llamada "reverse" cuya propiedad es que si se aplica dos veces se obtiene la función identidad: reverse(reverse(l)) == l. Esta propiedad debería cumplirse para cualquier lista finita por lo tanto es posible generar automáticamente casos de prueba que verifiquen que se cumple la propiedad.

Así, las funciones de ordenamiento "sort" debería cumplir con la propiedad de la idempotencia, es decir, no importa la cantidad de veces que se aplique la función sort, siempre se obtiene el mismo resultado: sort(l) == sort(sort(l)) == sort(sort(sort(l))) == ... y es posible generar casos de prueba donde cada uno de los e_i son números enteros entre INT_MIN e INT_MAX.

Otra propiedad que se debe cumplir es que si tomo un elemento e_i de la lista y luego un elemento e_{i+1}, entonces la propiedad debería ser: $e_i <= e_{i+1} \forall i \in n, i+1 < n$ para una lista de tamaño n.

Escriba, además de sus dos programas de ordenamiento, un programa que utilice las dos propiedades ya explicadas más una que usted encuentre que pueda resultar útil para validar que las funciones implementadas son correctas correctamente. Puede utilizar cualquier lenguaje de programación que quiera de la lista que aparece en este programa. Este programa de prueba debe entonces generar casos de prueba aleatorios que ejecuten automáticamente sus implementaciones y entreguen como resultado si pasó todas las pruebas o no, en caso de no pasar una prueba, imprima el caso fallido por pantalla. El programa de prueba debe llamar automáticamente a Mars con las pruebas generadas aleatoriamente para comprobar que funciona de forma correcta.

Note que la lista vacía [] también cumple con las propiedades anteriores y por lo tanto es un caso válido.

En total usted debe entregar cinco programas, cuatro en ensamblador MIPS y uno en algún lenguaje de los propuestos.

Procedimiento

1. Escriba su programa para probar las propiedades.
2. Implemente sus programas de ordenamiento.
3. Verifique que sus implementaciones pasan todas las pruebas.
4. Utilice ahora el simulador de cache para observar como se comportan los programas.
5. Elija el peor e implementelo con arreglos de memoria contigua.
6. Analice este nuevo programa y compárelo con el resto.
7. Proponga una hipótesis bajo la cual el programa con el arreglo pueda ser mejorado desde el punto de vista del cache.
8. Mofique el programa (el que usa arreglos) para poner a prueba su hipótesis, ¿Era correcta o no? ¿Por que?.

Pistas e información adicional

Argumentos en MARS

Los argumentos pasados por línea de comando com "java -jar Mars.jar PROGRAMA.asm pa ENTRADA.txt SALIDA.txt" se reciben por el lado de ensamblador como un puntero a strings.

$a0 corresponde a argc en C y $a1 corresponde a argv, por lo tanto para acceder al primer argumento recibido se debe hacer:

```
lw $t0, 0($a1)
```

Donde $t0 va a ser un puntero a un string.

Lenguajes permitidos para hacer las pruebas

- C/C++ (gcc, debe correr en GNU/Linux).
- Python.
- Ruby.
- Bash.
- Node/Javascript.
- Java/Scala.
- OCaml.
- Haskell.

Si quiere utilizar algún otro lenguaje consulte primero.

Limitaciones de entrada

La entrada puede ser cualquier entero con signo entre INT_MIN e INT_MAX de tamaño 4 bytes. El largo de los archivos es arbitrario y tiene terminaciones de líneas UNIX, es decir \n y no \r \n.

Arreglos de memoria contigua

Dado que los archivos de entrada pueden ser de tamaño arbitrario: ¿Cómo podría hacerlo dentro de un arreglo sin saber previamente el tamaño que va a tener? ¿Cómo funciona la estructura vector de C++?

Una forma de obtener memoria contigua es con la syscall 9. Recibe como argumento la cantidad de bytes que se quieren para el nuevo espacio de memoria y retorna un puntero a este espacio de memoria, por lo que puede ser un punto de partida para esta etapa del laboratorio.

Análisis: iterativo vs recursivo

Tenga en cuenta que el simulador de cache utiliza todos los accesos a memoria, es decir, en el caso de la función recursiva debe tomar en cuenta además de los accesos al arreglo los accesos a la memoria en el stack de MIPS.

Estructuras de datos en MIPS

Una estructura de datos no es más que una abstracción de la organización de la memoria proporcionada por el compilador. Al final solamente son bytes que se organizan en la memoria de acuerdo a las reglas que nosotros inventemos.

En C, una lista enlazada se puede crear utilizando struct:

```
typedef struct Nodo_ {
   int elemento;
   Nodo *siguiente;
} Nodo;
```

¿Qué significa esto?. No es más que decirle al compilador que todo lo que nosotros llamemos Nodo se va a componer de dos elementos realmente: un entero y un puntero al siguiente nodo. Dado que estamos trabajando con un simulador de MIPS de 32 bits, el puntero consiste en 32 bits. Si sumamos además el tamaño del entero, 32 bits, esto nos da una estructura que tiene un tamaño total de 64 bits u 8 bytes. Si vamos al compilador de C y le pedimos que nos diga cual es el tamaño de esta estructura, nos va a decir que son 16 bytes, esto es porque se trata de agrupar los bytes en potencias de 8 y dado que estamos usando un sistema de 64 bits donde los punteros son de 8 bytes, tenemos en total $8 + 4 = 12$, redondeado a la siguiente potencia de 8, 16 bytes.

Nosotros simplemente vamos a tomar como tamaño de la estructura la suma de los datos que se encuentran dentro de ella, en este caso sizeof(Nodo) == 8.

En C, para obtener memoria para este nodos obtendríamos un puntero mediante malloc, en mips vamos a usar la syscall 9. Para acceder a los elementos dentro de esta estructura necesitamos una convención, asignar una posición dentro de este espacio para cada uno de los elementos. Digamos que van a estar en orden: priemro elemento y luego siguiente. Para acceder a los elementos basta con utilizar las instrucciones de acceso a memoria:

```
# Asumimos que la memoria reservada se encuentra en $t0

addi $t1, $zero, 2
sw $t1, 0($t0)    # guardamos el elemento al principio de nuestra estructura
sw $zero, 4($t0)  # El siguiente elemento aún no existe, así que es NULL
```

Para un siguiente elemento simplemente asignaríamos al offset 4 el puntero a el.

Diferencias entre las estructuras

En clases vieron que existen dos princios que afectan el desempeño del cache: localidad espacial y localidad temporal.

Viendo las diferentes estructuras de datos, como los programas acceden a los datos (orden de las instrucciones de acceso a memoria), etc, ¿Cómo se podrían analizar los efectos que tienen en las tasas de hit y miss?

Informe, análisis y bibiliografías

El informe contiene una introducción que consiste en los siguientes elementos:

- Motivación: ¿Por que es importante el problema?
- Objetivo general: Es uno solo. Su conclusión se debe hacer a partir de este objetivo.
- Objetivos específicos: Cada objetivo específico es lo que se debe hacer en el trabajo para poder cumplir con el objetivo general. Son numerados.
- Problema: ¿Qué es lo que se intenta resolver?. Se plantea generalmente como una pregunta.
- Herramientas: ¿Qué programas o, en general, herramientas utilizó para el trabajo?. Indique las versiones.
- Organización del documento: ¿Cuáles son los capítulos de su informe y que contienen?

Un análisis es explicar cada uno de los fenómenos que se observan en sus experimentos. Si usted obtiene por ejemplo en una ejecución E_1 0.2 segundos y en una ejecución E_2 0.3, indicar estos resultados sin justificar no es un análisis. Un análisi podría ser: "Dado que E_1 utiliza las instrucciones X en vez de las Y como en el caso de E_2, se obtienen mejores tiempos ya que el CPI de las instrucciones X es Z". Claramente este es solamente un ejemplo y se pueden hacer muchos análisis más.

Otro punto a tomar en cuenta es que puede que obtenga resultados erróneos a simple vista, trate de investigar fuera de lo visto en clases para tratar de encontrar una explicación.

Parámetros del programa de prueba

El programa que va a generar las pruebas aleatoreas debe poder configurarse de acuerdo a la cantidad de pruebas aleatoreas a generar, mínimo y máximo de los elementos de la lista (entre INT_MIN e INT_MAX) y tamaño de la lista (0 hasta INT_MAX).

Comentarios

Una buena forma de documentar un programa es comentando las entradas y salidas de las funciones. Por ejemplo en C:

```c
/**
* Suma dos números.
* @param x El primer número a sumar.
* @param y El segundo número a sumar.
* @return La suma de x e y.
*/
int sumar(int x, int y){
  return x + y;
}
```

O en MIPS:

```
# Suma de dos números
# Argumentos:
# $a0: primer número
# $a1: segundo número
# Retorna la suma de $a0 con $a1
add $v0, $a0, $a1
jr $ra
```

Comentar todas las líneas puede resultar excesivo, incluso en ensamblador, pero queda a libertar suya si quiere hacerlo o no.

Referencias del informe

Existen muchos estilos de referencias que se pueden utilizar en un informe pero las más utilizadas en los papers y en el departamento de informática son: APA e IEEE. Si está usando LaTeXpuede utilizar bibtex para autogenerar estas referencias a partir de un archivo .bib.

Si usted está diciendo por ejemplo: "MIPS es una arquitectura de procesadores RISC que se ha utilizado para...", eso fue sacado de alguna parte, entonces en el texto debe incluir de donde lo sacó. "MIPS es una arquitectura de procesadores RISC que se ha utilizado para... [1]"(en caso de utilizar IEEE). De esta forma el lector puede ir al final de su informe y revisar la referencia número 1 donde puede obtener más información o saber que lo que usted puso está justificado por algún experto en el área.

Imágenes y tablas sacadas de libros o Internet también deben ser referenciadas en el subtítulo de cada una de ellas. Por ejemplo: "Figura 1: Tipos de instrucciones en la arquitectura MIPS [2]."

MoSCoW del laboratorio

MoSCoW es un método para priorizar tareas en un proyecto. Consiste en asignar un nivel de necesidad de entre los cuatro disponibles. Nosotros vamos a utilizar solamente dos: "must have" y "should have".

Todo lo que se indique como "must have" debe ser incluido para recién comenzar a evaluar el laboratorio mientras que todo lo que quede fuera de esta categoría se considera como "should have" y afecta solamente la nota. En otras palabras, si falta un "must have" se va a calificar con la nota mínima tanto programa como informe.

Los siguientes puntos son los "must have":

- Cada análisis debe tener su correspondiente experimento.
- Cada experimento debe tener su correspondiente programa.
- Cada programa debe entregar la respuesta correcta (archivo con los números ordenados de menor a mayor).
- Sus programas de ordenamiento debe recibir los archivos de entradas por medio de la línea de comandos.
- Debe hacer pruebas suficientemente diferentes (configuraciones de caché) para poder descubrir el comportamiento de las estructuras de datos.
- Tanto programa como informe deben ser entregados primero por usachvirtual y luego el informe de forma impresa.
- El análisis del informe debe poder jusficiar los resultados obtenidos.
- Un Makefile debe ser incluido para construir sus programas en caso de ser necesario.
- Sus programas deben funcionar en GNU/Linux incluyendo la lectura de archivos con el separador de líneas de UNIX.
- Referencias al final del documento deben ser utilizadas en el texto.

Consideraciones

Generales

- Las copias entre compañeros e Internet serán calificadas con la nota mínima.
- El laboratorio es individual.
- En caso de no cumplir con un "must have" se va a calificar con la nota mínima.
- Si el laboratorio es entregado con 4 o más días de atraso se considerará que no se ha presentado ningún trabajo.
- Si no entrega uno de los 3 laboratorios del semestre entonces se reprueba todo el laboratorio.
- Los programas corresponden a un 40 % de la nota y el informe un 60 % de la nota de entrega del laboratorio. Note que no puede hacer el informe sin hacer el o los programas ya que debe basarse en ellos.
- Por cada día de atraso se descuenta 1 punto a la nota general del laboratorio (esto incluye la entrega atrasada de informes). Ej: si el laboratorio se debe entregar a las 23:50 y se entrega a las 23:55 hay un punto de descuento. Si se entrega a las 23:55 del siguiente día hay dos puntos de descuento. Si tiene un 7 como nota de laboratorio (promedio entre informe y programa) entonces tendría un 6.
- Debe entregar en el espacio habilitado en usachvirtual una carpeta comprimida (.zip o .tar.gz o .tar.bz2) con el código fuente del programa en una carpeta llamada "src" y el informe y manual de usuario en pdf. Este archivo debe llamarse: Apellido1_Apellido2.(zip—tar.bz2)
- La fecha de entrega es el jueves 8 de diciembre a las 23:55 por usachvirtual. No se corregirán laboratorios entregados por otros medios.
- El laboratorio consiste en informe más programa. Si no entrega uno de estos elementos se calificará con la nota mínima.

Programa

- Recuerde poner comentarios en su programa e indicar los argumentos que reciben las funciones y que es lo que retornan (también indique aquellas funciones que no tienen un valor de retorno).
- El programa debe funcionar en sistemas GNU/Linux. Para que el programa se califique de forma correcta debe poder ejecutarse sin problemas en este sistema operativo, de lo contrario no se va a poder evaluar el programa.
- Los programas consisten en las siguientes ponderaciones:
  1. Algoritmo iterativo (Linked List) (15 %).
  2. Algoritmo recursivo (Linked List) (15 %).
  3. Mejora peor algoritmo (Array) (25 %)
  4. Mejora propuesta (hipótesis) al algoritmo (implementación) (25 %)
  5. Programa de pruebas (20 %).
- Note que no puede hacer las mejoras sin implementar los algoritmos con listas enlazadas

Informe

- El informe debe ser entregado impreso en secretaría con el nombre del profesor, ayudantes y asignatura a más tardar 12 hrs desde la fecha de entrega por usachvirtual. Además debe subir el informe junto con el programa a usachvirtual. Ej: Si el laboratorio se debe entregar a las 23:50 por usachvirtual, entonces el informe debe estar a las 12:00 en secretaría. Se aplicará el mismo criterio que el que aparece en las consideraciones generales para los atrasos.
- El informe debe contener las siguientes secciones (consulte el formato de memoria para más información sobre lo que deben llevar estas secciones):
  1. Introducción: Objetivo generales, objetivos específicos, organización del documento, motivación, problema, herramientas. (10 %)
  2. Marco teórico: Lo que un lector debería saber para entender su trabajo (**NO ES UN GLOSARIO**). (15 %)
  3. Desarrollo: Explicar como se construyó el programa y presentar resultados tanto de programas como de las pruebas de la hipótesis.(25 %)
  4. Análisis: Explicar los resultados utilizando conceptos vistos en clases, hipótesis y análisis la hipótesis (30 %)
  5. Conclusión: Sobre los objetivos planteados en la introducción. (15 %)
  6. Referencias (5 %)
- Debe utilizar el formato de presentación de memoria disponible en el archivo "Propuesta de normas para presentación del trabajo de titulación de pregrado, Departamento de Ingeniería Informática" que puede encontrar en la página del curso. Se descontará una décima por cada error en el formato.
- Cuide la ortografía. Se descontará una décima por cada falta.
- Incluya referencias en la bibliografía indicando claramente el texto citado. Utilice el formato APA (parte del formato del informe) o IEEE. En caso de encontrar textos que no hayan sido citados se considerará como copia.