Trabajo Práctico 1

Agustin Alejandro Linari, *Padrón Nro. 81.783*
agustinlinari@gmail.com

Juan Ignacio López Pecora, *Padrón Nro. 84.700*
jlopezpecora@gmail.com

Pablo Daniel Sívori, *Padrón: 84.026*
sivoridaniel@gmail.com

2° Cuatrimestre de 2016
66.20 Organizacion de Computadoras
Facultad de Ingenieria, Universidad de Buenos Aires

25 de octubre de 2016

Resumen

En el presente trabajo utilizamos el conjunto de instrucciones MIPS y el concepto de ABI para resolver parte de la lógica del programa realizado en el trabajo práctico 0.

Índice

Parte I

Desarrollo

1. Introduccion

El objetivo del presente trabajo práctico es familiarizarse con el código de instrucciones MIPS 32. Para ello implementaremos la lógica de cómputo del fractal con dicho código de instrucciones. Finalmente compilaremos el programa en el emulador GXemul para poder obtener el código de instrucciones Mips32.

2. Build

El correspondiente informe se puede construir utilizando el make con la etiqueta doc la cual borra y genera el informe en formato pdf.

3. Diseño e Implementación del Programa

El código fuente del programa se puede encontrar en el anexo A.

3.1. Stack Frame

A continuación mostramos los diagrama de stack frame de las funciones implementadas en MIPS32.

int buff_write(int fd, char* buf, int size)		
ABA (caller)		
	64	size
	60	str
	56	fd
SRA	52	ra
	48	gp
	44	fp
	40	s6
	36	s5
	32	s4
	28	s3
	24	s2
	20	s1
	16	s0
ABA	12	
	8	
	4	
	0	a0 (fd)

Figura 1: Stack1

int buff_flush(int fd)		
ABA (caller)		
	16	fd
SRA	12	gp
	8	fp
LTA	4	
	0	result

Figura 2: Stack2

int print_int(int n, int fd)		
ABA (caller)	44	fd
	40	n
SRA	36	
	32	ra
	28	fp
	24	sp
LTA	20	
	16	r
ABA	12	a3
	8	a2
	4	a1
	0	a0

Figura 3: Stack3

int mips32_plot(param_t* params)		
ABA (caller)	88	params (param_t*)
SRA	84	
	80	ra
	76	gp
	72	fp
LTA	68	
	64	cpi
	60	cpr
	56	res
	52	c
	48	y
	44	x
	40	absz
	36	si
	32	sr
	28	zi
	24	zr
	20	ci
	16	cr
ABA	12	a3
	8	a2
	4	a1
	0	a0

Figura 4: Stack4

4. Corridas de Programa

Se corre el programa obteniendose los siguientes tiempos de ejecución que se detallan en los siguientes cuadros.

Código C	real	usr	sys
1	1m19.363s	1m19.143s	0m0.141s
2	1m23.195s	1m23.014s	0m0.129s
3	1m21.480s	1m19.335s	0m0.133s
promedio	1m 21.346s	1m 20.497s	0.134s

Cuadro 1: Tiempos promedios de ejecución en código C.

Código C	real	usr	sys
1	1m0.449s	0m52.722s	0m7.680s
2	1m0.879s	0m53.042s	0m7.808s
3	1m1.246s	0m53.261s	0m7.937s
promedio	1m0.858s	53.008s	7.808s

Cuadro 2: Tiempos promedios de ejecución en código Mips buffer size 64 bytes.

Código C	real	usr	sys
1	1m4.262s	1m3.237s	0m0.984s
2	1m1.246s	1m0.319s	0m0.883s
3	0m58.547s	0m57.573s	0m0.937s
promedio	1m1.352s	1m0.376s	0.935s

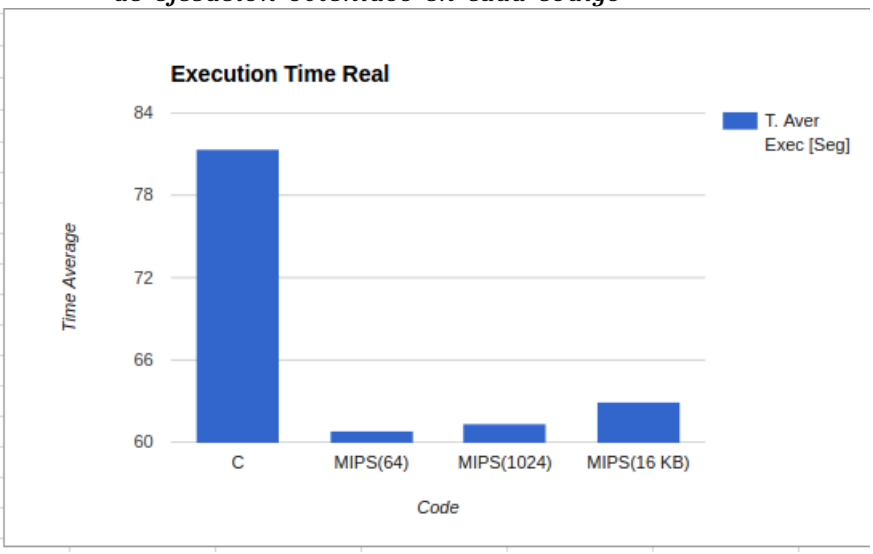
Cuadro 3: Tiempos promedios de ejecución en código Mips buffer size 1 KB.

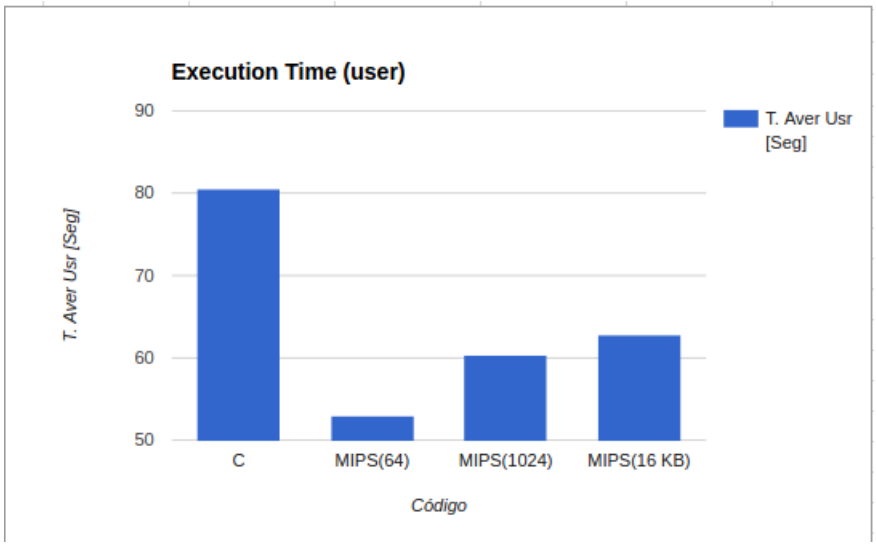
Código C	real	usr	sys
1	1m3.797s	1m3.589s	0m0.156s
2	1m0.617s	1m0.480s	0m0.133s
3	1m4.500s	1m4.249s	0m0.215s
promedio	1m2.971s	1m2.772s	0.168s

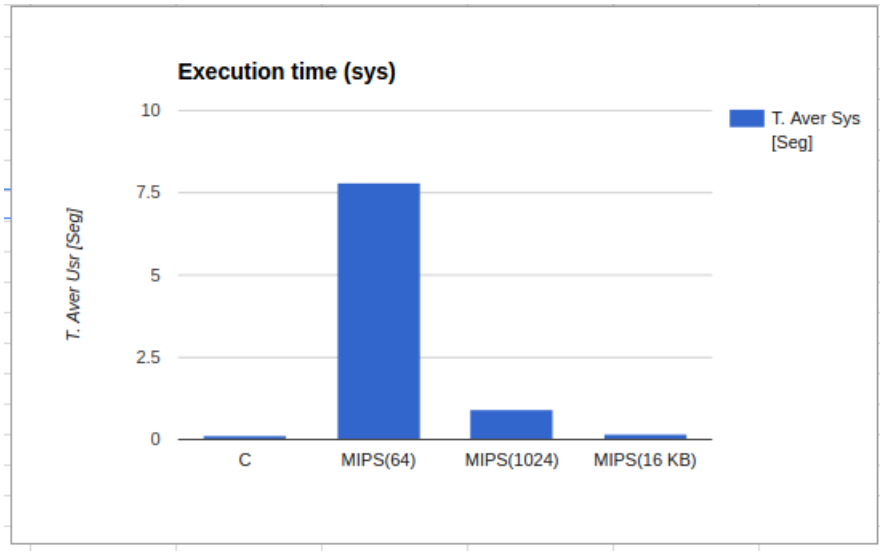
Cuadro 4: Tiempos promedios de ejecución en código Mips buffer size 16 KB.

- Todos los programas fueron compilados con el parámetro -O0 (sin optimizaciones).
- Todas las mediciones corresponden a la ejecución de los programas utilizando el comando time.

Representamos gráficamente los valores obtenidos en los cuadros anteriores, y hacemos una comparación entre los tiempos promedios de ejecución obtenidos en cada código




Execution Time (user)
90
80
70
60
50
T. Aver Usr [Seg]
C
MIPS(64)
MIPS(1024)
MIPS(16 KB)
Código
T. Aver Usr [Seg]


Execution time (sys)
10
7.5
5
2.5
0
T. Aver Usr [Seg]
C
MIPS(64)
MIPS(1024)
MIPS(16 KB)
T. Aver Sys [Seg]

Speed Up	Value
tc / tmips 64B	1.34
tc / tmips 1KB	1.33
tc / tmips 16KB	1.29

Cuadro 5: Speed UP

4.1. Observaciones

- La implementación de mips en cualquiera de sus variantes se ejecuta en menos tiempo que la implementación C pura, con un speed up aproximado de 1/3
- A medida que el buffer se agranda, el tiempo de sistema se reduce. Esto se debe a que se producen menos syscalls a write.
- Podemos plantear la hipótesis razonable de que printf está implementada con un buffer (debido al bajo sys time). Para nuestra implementación MIPS, el tamaño del buffer que obtuvo un tiempo sys del mismo orden que la implementación C fue de 16 KB.
- Contrario a lo que nuestra intuición indicaba, aumentar el buffer para valores mayores a 64 bytes no necesariamente significó (en promedio) en un aumento de performance. Esto puede estar relacionado con la arquitectura del cache emulado, el tamaño de bloque y su política de reemplazo.

4.2. Conclusión

Con la realización de este trabajo hemos podido apreciar la diferencia de performance entre dos implementaciones de distinta naturaleza de un mismo algoritmo, implementado en C y en assembly MIPS32.

A la hora de programar, es común que se codifique utilizando lenguajes de alto nivel. El lenguaje de programación C es un lenguaje de propósito general clásico cuyo diseño provee construcciones que mapean de manera eficiente instrucciones de máquina típicas. Las ventajas de utilizar un lenguaje de alto nivel como C son portabilidad (a nivel código fuente) entre diferentes arquitecturas donde se haya implementado el compilador, aumento de productividad - dado que se abstrae de cuestiones de bajo nivel íntimamente ligadas con la arquitectura de la máquina - y reducción en el costo de mantenimiento. Sin embargo, estas ventajas traen aparejado un costo en la performance del programa.

En algunos casos, los requerimientos funcionales de un programa requieren de una performance que puede ser difícil de alcanzar para una implementación en un lenguaje de alto nivel. Mediante un análisis cuantitativo, se determina qué segmentos de código consumen la mayor cantidad de recursos de una computadora -ciclos de CPU, memoria, etc-. Para el caso particular de este trabajo, la función de cómputo del fractal es central en el desempeño de la aplicación.

4.3. Pruebas

Parte II

Apendice

A. Codigo fuente

B. Enunciado original

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS
2^o cuatrimestre de 2016

`$Date: 2016/10/02 21:23:43 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto en la sección 4.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

Se trata de un modificar un programa que dibuje el conjunto de Julia y sus vecindades introducido en el *TP0* [1], en el cual la lógica de cómputo del fractal deberá tener soporte nativo para MIPS32 sobre NetBSD/pmax.

El código fuente con la versión inicial del programa, se encuentra disponible en [2]. El mismo deberá ser considerado como punto de partida de todas las implementaciones.

4.1. Soporte para MIPS

El entregable producido en este trabajo deberá implementar la lógica de cómputo del fractal en assembly MIPS32, con soporte nativo para NetBSD/pmax.

Para ello, cada grupo deberá tomar el código fuente de base para este TP, [2], y reescribir la función `mips32_plot()` sin cambiar su API. Esta función está ubicada en el archivo `mips32_plot.c`.

4.2. Casos de prueba

El informe trabajo práctico deberá incluir una sección dedicada a verificar el funcionamiento del código implementado. Para ello, será necesario escribir pruebas orientadas a probar el programa completo, ejercitando los casos más comunes de funcionamiento, los casos de borde, y también casos de error.

4.3. Compilación

El código fuente provisto por la cátedra provee los makefiles necesarios para compilar el ejecutable a partir de la versión en C con el archivo `mips32_plot.c`. Para poder compilar el código desarrollado deberán cambiar la definición en el archivo `Makefile.in` la línea número 6:

```
SRCS = mips32_plot.c main.c mygetopt_long.c
```

por

```
SRCS = mips32_plot.S main.c mygetopt_long.c
```

Luego deberán invocar la siguiente secuencia de comandos para limpiar los archivos temporales y generar los nuevos Makefiles:

```
$ make clean
$ make makefiles
$ make
```

4.4. Detalles de la implementación

Para optimizar los accesos a las llamadas a servicio del sistema (`syscalls`), deben utilizar un buffer de n bytes para escribir los datos de salida para luego ser enviados al archivo de salida. El tamaño n debe ser parametrizable mediante un `#define`.

5. Informe

El informe, a entregar en formarto impreso y digital[1] deberá incluir:

- Documentación relevante al diseño e implementación del código esarrollado para adaptar el programa. Incluír el diagrama de *stack frame* de las funciones implementadas en MIPS32.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente. Especificar modificaciones realizadas a los archivos provistos por la cátedra si es que los hubo.
- Las corridas de prueba, con los comentarios pertinentes.[2]
- El código fuente, en lenguaje C (y MIPS32 donde corresponda)
- Este enunciado.

[1] En CD, DVD o memoria flash.

[2] Las pruebas provistas deben ejecutarse correctamente en NetBSD sobre MIPS32 sin modificación alguna.

6. Fecha de entrega

La fecha de vencimiento será el Martes 01/11.

Referencias

[1] Trabajo Práctico 0, 2do cuatrimestre de 2016.
`https://groups.yahoo.com/neo/groups/orga-comp/files/TPs/tp0-2016-2q.pdf`.

[2] Código fuente con el esqueleto del trabajo práctico.
`https://drive.google.com/open?id=0B93s6e6NY_j1TFV2TFBqbUNKZ3M`.