Trabajo Práctico 1: Programación en MIPS

Ronnie Del Pino Cárdenas, *Padrón 93575*
delpinor@gmail.com
@delpinor

Emiliano Vega, *Padrón 76676*
emiliano.vega@mail.com
@emiliano.vega

Romina Casal, *Padrón 86429*
casal.romina@gmail.com
@romina

2do. Cuatrimestre de 2019
86.37 / 66.20 Organización de Computadoras − Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

El presente trabajo práctico nos permitió realizar una comparativa entre la performance de dos implementaciones de un programa en C basado en el algoritmo "La hormiga artista"del trabajo práctico anterior en varios niveles de optimización y reemplazando dos funciones clave a código MIPS. Para el análisis se utilizó el programa /usr/bin/time que nos da los tiempos de ejecución resultante.

1. Introducción

Para la realización de las pruebas se ha creado un Makefile adaptado que permite compilar las versiones tp1_if y tp1_tables en los niveles 0, 1, 2 y 3 de optimización que ofrece gcc, además de las versiones con las funciones new_orientation y move_forward en código assembly.
Se realizará la ejecución de varias opciones de iteraciones con estas versiones para comparar los distintos tiempos de ejecución y observar la tasa de mejora lograda.

2. Proceso de Compilación

Se adaptó el archivo makefile provisto por la cátedra para generar los ejecutables con las distintas opciones de optimización para el código integralmente en C.

make all_tp1_c: Crea los ejecutables tp1_if_opt0, tp1_if_opt1, tp1_if_opt2, tp1_if_opt3 para la versión con jumps en los distintos niveles de optimización, y tp1_tables_opt0, tp1_tables_opt1, tp1_tables_opt2, tp1_tables_opt3 para la versión con tables.

make tp1_tables_asm: Crea el ejecutable tp1_tables_asm para la versión con table que tiene las funciones new_orientation y move_forward en assembly.

make tp1_if_asm: Crea el ejecutable tp1_if_asm para la versión con jumps que tiene las funciones new_orientation y move_forward en assembly.

Creamos un archivo script pruebas.sh que usando **/usr/bin/time** cuenta el tiempo de ejecución de cada prueba y guarda los resultados en un archivo **test_result.txt** Con los valores obtenidos en este archivo se crearon gráficas comparativas de tiempo de ejecución en función de la cantidad de iteraciones.

Por ejemplo, la línea:

```
/usr/bin/time --output=test_result.txt -a -f "%E\treal\t%U\tuser\t%S\tsys" ./tp1_if_opt0 -g 1000x1000 -p RGBW -r LLLL -t $((10000)) > /dev/null
```

Escribirá el siguiente resultado en **test_result.txt**

```
0:11.39 real 11.14 user 0.03 sys
```

De esa medición usaremos el tiempo real de ejecución obtenido para las comparaciones.
NOTA: Es probable que sea necesario instalar en la VM ejecutada con QEMU el programa time que tiene opciones adicionales que el homónimo que trae por defecto no permite usar.

3. Desarrollo

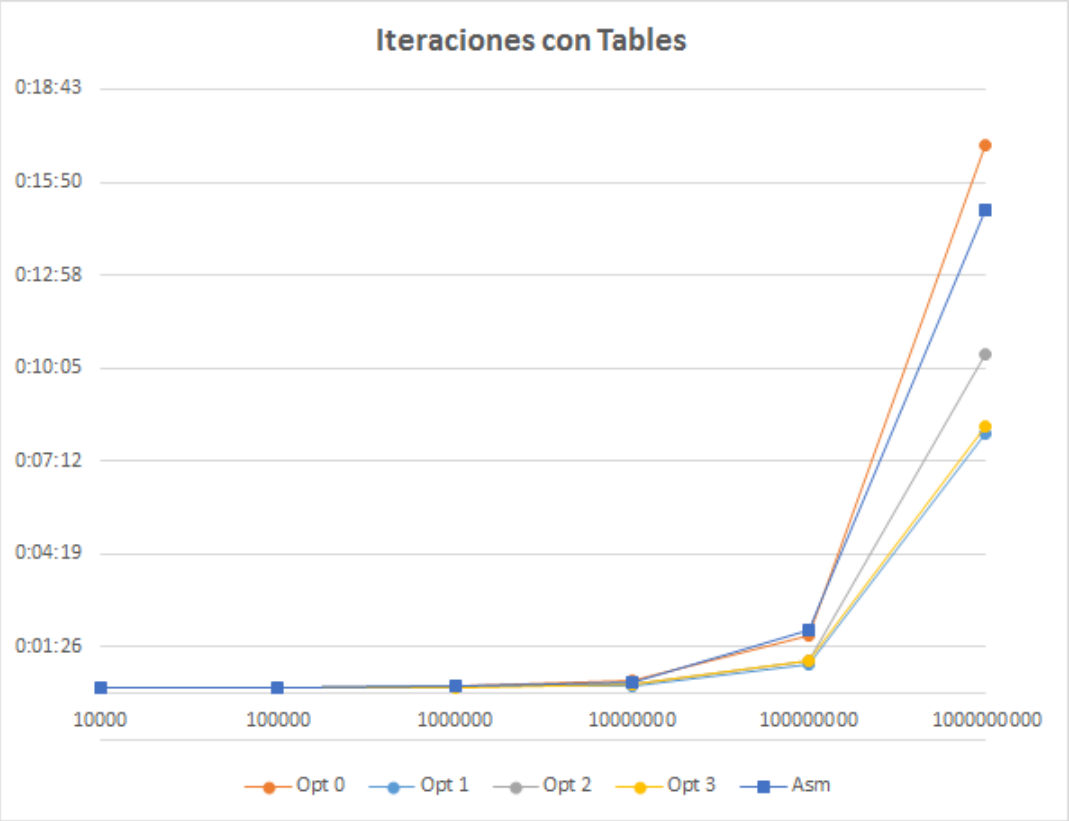
Para las adaptaciones a assembly en general, usamos la convención ABI dada en clase, conservando los parámetros recibidos en la LTA del stack y reservando espacio para los registros $gp, $fp y $ra en la SRA.
Para el caso de la versión con jumps, decidimos implementar las funciones **decide()** y **adjust()** en forma inline en el código de **new_orientation** y **move_forward**, ya que la primera era una macro, y en el caso de la segunda para reducir la creación de llamadas con sus respectivos stacks.
En el caso de la versión con tables, reservamos en el segmento de datos los arrays **rotation_rules** y **allowed_forward** y para tratarlos como variables estáticas en lugar de reservar continuamente el espacio y asignar los valores dinámicamente.

4. Resultados obtenidos

Ejecutamos el script **pruebas.sh** y arroja los siguientes resultados que exporta al archivo **test_result.txt**:

```
Pruebas TP1
Grafico de iteraciones
tp1_if - optimizacion 0
0:11.39 real 11.14 user 0.03 sys
0:12.41 real 12.35 user 0.04 sys
0:12.19 real 12.16 user 0.02 sys
0:18.53 real 18.50 user 0.01 sys
1:19.81 real 79.50 user 0.04 sys
11:41.06 real 700.63 user 0.03 sys
tp1_if - optimizacion 1
0:10.25 real 10.20 user 0.03 sys
0:10.66 real 10.62 user 0.02 sys
0:10.18 real 10.14 user 0.03 sys
0:15.36 real 15.30 user 0.04 sys
0:50.10 real 50.03 user 0.02 sys
6:47.14 real 406.88 user 0.02 sys
tp1_if - optimizacion 2
0:11.16 real 11.12 user 0.03 sys
0:11.11 real 11.08 user 0.02 sys
0:12.02 real 11.96 user 0.04 sys
0:18.66 real 18.62 user 0.02 sys
0:47.17 real 47.12 user 0.02 sys
6:26.73 real 386.48 user 0.03 sys
tp1_if - optimizacion 3
0:09.87 real 9.83 user 0.02 sys
0:09.99 real 9.95 user 0.02 sys
0:11.56 real 11.52 user 0.02 sys
0:14.15 real 14.12 user 0.01 sys
0:45.25 real 45.21 user 0.00 sys
6:00.04 real 359.80 user 0.03 sys
tp1_tables - optimizacion 0
0:09.61 real 9.57 user 0.01 sys
0:11.39 real 11.37 user 0.00 sys
0:12.67 real 12.63 user 0.02 sys
0:22.76 real 22.71 user 0.03 sys
1:47.64 real 107.54 user 0.03 sys
17:00.54 real 1019.67 user 0.05 sys
tp1_tables - optimizacion 1
0:08.93 real 8.89 user 0.02 sys
0:10.60 real 10.56 user 0.02 sys
0:12.15 real 12.11 user 0.02 sys
0:14.79 real 14.74 user 0.03 sys
0:55.26 real 55.21 user 0.01 sys
8:04.03 real 483.73 user 0.02 sys
tp1_tables - optimizacion 2
0:11.35 real 11.31 user 0.02 sys
0:10.24 real 10.19 user 0.04 sys
0:12.67 real 12.61 user 0.03 sys
0:16.59 real 16.55 user 0.02 sys
0:59.45 real 59.38 user 0.03 sys
10:28.80 real 628.41 user 0.04 sys
tp1_tables - optimizacion 3
0:09.76 real 9.71 user 0.03 sys
0:11.14 real 11.10 user 0.02 sys
0:10.42 real 10.37 user 0.03 sys
0:17.49 real 17.45 user 0.02 sys
0:59.53 real 59.45 user 0.03 sys
8:16.94 real 496.64 user 0.02 sys
tp1_if - assembly
0:10.65 real 10.62 user 0.01 sys
0:11.28 real 11.24 user 0.02 sys
0:11.25 real 11.22 user 0.01 sys
0:24.52 real 24.47 user 0.02 sys
1:31.24 real 91.13 user 0.03 sys
14:27.08 real 866.43 user 0.04 sys
tp1_tables - assembly
0:11.83 real 11.80 user 0.01 sys
0:10.47 real 10.42 user 0.03 sys
0:13.32 real 13.26 user 0.03 sys
0:19.97 real 19.92 user 0.03 sys
1:56.73 real 116.64 user 0.01 sys
14:57.11 real 896.46 user 0.05 sys
```

Con los datos obtenidos en las pruebas realizamos dos gráficos comparativas entre las distintas optimizaciones y la versión con funciones en assembly para cada implementación que reflejan el tiempo de ejecución en función de la cantidad de iteraciones:



En ellos observamos dos tendencias:
En la implementación con jumps, las mediciones nos muestran que la implementación con las funciones **new_orientation** y **move_forward** en assembly tuvieron una performance inferior que las contrapartidas en C, incluso la compilada con la opción -O0. Otra cosa que podemos observar es que de la optimización con opción -O0 a -O1 ya una mejora más significativa que la obtenida entre las compilaciones hechas con opciones -O1, -O2 y -O3.
En la implementación con tables, se observa que la implementación con las funciones en assembly mostró un mejor resultado que la opción -O0, pero aún menos performante que con la opción -O1. Además, entre las opciones -O1 y -O2 hay una diferencia de performance mayor que entre las análogas de la implementación con jumps. Luego, entre las compilacionese con -O2 y -O3 sí coincide la tendencia a mostrar una mejora poco significativa.

5. Conclusión

De acuerdo a los resultados obtenidos al implementar las funciones solicitadas en assembly, concluimos por un lado que la implementación no mejora la ejecución en el caso de Jumps, ni siquiera comparado con la opción de optimización 0, y en el caso de Tables, sólo mejora con respecto a esta optimización, lejos de las demás opciones. Suponemos que se están dando dos situaciones. Por un lado, que el código assembly generado por el compilador en todos los niveles de optimización es casi siempre más performante que el que hemos realizado. Y por otro lado, que aún siéndolo, las funciones implementadas no son mejora suficiente para influir en un cambio de performance del programa en general.

Referencias

[1] Profiling and Timing Code .
https://jakevdp.github.io/PythonDataScienceHandbook/01.07-timing-and-profiling.html.

[2] C to MIPS compiler .
http://reliant.colab.duke.edu/c2mips/.

Parte I

Apéndice

A. Código fuente

Repositorio: github.com/romicasal/orga6620/tree/tp1_delpinor

A.1. ant_engine_jumps.S

```
#include <regdef.h>
#define ON 0
#define OS 1
#define OE 2
#define OW 3


.data
ime: .asciiz "TP1 2Q2019: Implement me!"

.text
.align 2
.globl new_orientation
.ent new_orientation

new_orientation:
#    la a0, ime
#    jal doPanic
#armo stack
   subu  sp, sp, 24
   sw ra, 16(sp)
   sw fp, 12(sp)
   sw gp, 8(sp)
   move  fp, sp

   # guardo parametros
   sw a0, 0(sp)   # orientation
   sw a1, 4(sp)   # rule
   # cargo parametros en registros
   move  t0, a0
   move  t1, a1

   # orientacion?

norte:   bne   t0, 0, sur
   bne   t1, zero, norte_derecha # RL = 0
   addi  t2, zero, 3       # OW = 3
   j  return_value
norte_derecha:
   addi  t2, zero, 2       # OE = 2
   j  return_value
sur:  bne   t0, 1, este
   bne   t1, zero, sur_derecha
   addi  t2, zero, 2
   j  return_value
sur_derecha:
   addi  t2, zero, 3
   j  return_value
este: bne   t0, 2, oeste
   bne   t1, zero, este_derecha
   addi  t2, zero, 0       # ON = 0
   j  return_value
```

```
este_derecha:
   addi  t2, zero, 1       # OS = 1
   j  return_value

oeste:   bne   t1, zero, oeste_derecha
   addi  t2, zero, 1
   j  return_value
oeste_derecha:
   addi  t2, zero, 0

return_value:
   # return new orientation
   move  v0, t2

   #borro stack
   lw gp, 8(sp)
   lw fp, 12(sp)
   lw ra, 16(sp)
   addu  sp, sp, 24
   jr ra


.end new_orientation

.text
.align 2
.globl move_forward
.ent move_forward
move_forward:
   #    la a0, ime
   #   jal doPanic
   # allocate stack frame(SRA=8 LTA=0 ABA=0)
   addi  sp, sp, -8
   sw    fp, 4(sp)
   sw    gp, 0(sp)
   move  fp, sp

   # load struct ant_t
   lw    t0, 0(a0)         # x
   lw    t1, 4(a0)         # y
   lw    t2, 8(a0)         # o
                        # switch statement
   bne      t2, ON, case_south
                        # adjust(&ant->y, ant->y - 1, height);
   addi  t1, t1, -1        # ant->y - 1
   div      t1, a2            # ant->y / heigth
   mfhi  t5              # remainder moved into t5
   sw    t5, 4(a0)         # &ant->y
   j     done
case_south:
   bne      t2, OS, case_east
   addi  t1, t1, 1         # ant->y + 1
   div      t1, a2            # ant->y / height
   mfhi  t5              # remainder moved into t5
   sw    t5, 4(a0)         # &ant->y
   j     done
case_east:
   bne      t2, OE, case_west
   addi  t0, t0, 1         # ant->x + 1
   div      t0, a1            # ant->x / width
   mfhi  t5              # remainder moved into t5
   sw    t5, 0(a0)         # &ant->x
   j     done
case_west:
   #bne     t2, OW, case_default
   addi  t0, t0, -1        # ant->x + 1
```

```
    div     t0, a1              # ant->x / width
    mfhi    t5              # remainder moved into t5
    sw      t5, 0(a0)           # &ant->x
    j       done
done:
    lw      fp, 4(sp)
    lw      gp, 0(sp)
    addiu   sp, sp, 8
    move    v0, a0              # Save the return value in v0
    jr      ra
.end move_forward
```

A.2. ant_engine_tables.S

```
#include <regdef.h>
#define ON 0
#define OS 1
#define OE 2
#define OW 3
.data
ime: .asciiz "TP1 2Q2019: Implement me!"
rotation_rules:    .word 3, 2, 2, 3, 0, 1, 1, 0
allowed_forward: .word 0, 1, 2, 3       # steps
.text
.align 2
.globl new_orientation
.ent new_orientation

new_orientation:
#     la a0, ime
#     jal doPanic
    #armo stack
    subu  sp, sp, 24
    sw ra, 16(sp)
    sw fp, 12(sp)
    sw gp, 8(sp)
    move  fp, sp

    # guardo parametros
    sw a0, 0(sp)    # orientation
    sw a1, 4(sp)    # rule

    # cargo parametros en registros
    # y calculo indice en el array
    mul   t0, a0, 8   # indice orientation x 4bytes = word
    mul   t1, a1, 4
    add   t0, t0, t1
    la t2, rotation_rules(t0)

    # return new orientation
    lw v0, (t2)

    #borro stack
    lw gp, 8(sp)
    lw fp, 12(sp)
    lw ra, 16(sp)
    addu  sp, sp, 24
    jr ra

.end new_orientation

.text
.align 2
.globl move_forward
.ent move_forward
move_forward:
```

```
    #la a0, ime
    #jal doPanic
    addi sp, sp, -32       #allocate stack frame(SRA=16 LTA=16 ABA=0)
    sw      ra, 24(sp)
    sw      fp, 20(sp)
    sw      gp, 16(sp)
    sw      a2, 12(sp)        # relevants_bouds[0] = height
    sw      a2, 8(sp)            # relevants_bouds[1] = height
    sw      a1, 4(sp)            # relevants_bouds[2] = width
    sw      a1, 0(sp)            # relevants_bouds[3] = width
    lw      t3, 8(a0)            # load ant_t->o = index
    sll     t3, t3, 2            # index*4
    add     t4, t3, sp        # base + index*4
    lw      t4, 0(t4)            # = bound
    la      t5, allowed_forward  # base array address
    add     t5, t5, t3        # base + index*4
   lw    t5, 0(t5)            # = step_fun
   bne      t5, ON, step_south      # step_north
   lw    t6, 4(a0)            # ant->y
   addi  t6, t6, -1        # ant->y - 1
   div      t6, t4            # ant->y / height
   mfhi  t6                # remainder moved into t6
   sw    t6, 4(a0)            # &ant->y
   j     done
step_south:
   bne      t5, OS, step_east
   lw    t6, 4(a0)            # ant->y
   addi  t6, t6, 1            # ant->y + 1
   div      t6, t4            # ant->y / height
   mfhi  t6                # remainder moved into t6
   sw    t6, 4(a0)            # &ant->y
   j     done
step_east:
   bne      t5, OE, step_west
   lw    t6, 0(a0)            # ant->x
   addi  t6, t6, 1            # ant->x + 1
   div      t6, t4            # ant->x / width
   mfhi  t6                # remainder moved into t6
   sw    t6, 0(a0)            # &ant->x
   j     done
step_west:
   lw    t6, 0(a0)            # ant->x
   addi  t6, t6, -1        # ant->x - 1
   div      t6, t4            # ant->x / width
   mfhi  t6                # remainder moved into t6
   sw    t6, 0(a0)            # &ant->x
   j     done
done:
   lw    ra, 24(sp)
    lw      fp, 20(sp)
    lw      gp, 16(sp)
    addi sp, sp, 32
    move v0, a0
    jr      ra
.end move_forward
#fin
```

B. Diagramas de stacks

B.1. ant_engine_jumps.S

new_orientation

Sección	Variable	Posición
SRA	-	20
	ra	16
	fp	12
	gp	8
LTA	orientation	4
	rule	0

move_forward

Sección	Variable	Posición
SRA	fp	4
	gp	0

B.2. ant_engine_tables.S

new_orientation

Sección	Variable	Posición
SRA	-	20
	ra	16
	fp	12
	gp	8
LTA	orientation	4
	rule	0

move_forward

Sección	Variable	Posición
SRA	-	28
	ra	24
	fp	20
	gp	16
LTA	a	12
	b	8
	c	4
	d	0

C. Enunciado original

Trabajo práctico 1A: Conjunto de instrucciones MIPS

1. Objetivo

El objetivo de este trabajo es proveer una versión del programa en la que las funciones `new_orientation` y `move_forward` estén codificadas en assembly MIPS32, y comparar su desempeño con el de esas mismas funciones compiladas con diversos grados de optimización.

2. Introducción

Este trabajo práctico toma como precendente el Trabajo Práctico 0, que evaluó el desempeño de dos implementaciones posibles de un problema denominado "La hormiga artista". Además se ejercitó la instalación de un ambiente capaz de ejecutar programas para distintas variantes de la arquitectura MIPS.

En la clase del 12/9 se presentó la convención de llamadas a funciones a ser utilizada por la práctica. Partiendo desde este punto, se busca extender el trabajo práctico implementando una parte acotada de la funcionalidad en assembly MIPS32.

Al finalizar el trabajo práctico se deben presentar conclusiones relevantes sobre la implementación realizada incluyendo ventajas y desventajas de la implementación en este lenguaje.

3. Implementación

En el trabajo práctico anterior, existian dos versiones de las siguientes funciones, implementadas en C. En este trabajo las mismas deben estar codificadas en assembly MIPS32, respetando el ABI de la cátedra.

```
orientation_t
new_orientation(orientation_t orientation, rotation_t rule);

ant_t*
move_forward(ant_t* ant, uint32_t width, uint32_t height);
```

3.1. Ejemplos

Listamos las opciones utilizando el comando `--help`

```
./tp1_if --help
  ./tp1_if -g <grid_spec> -p <colour_spec> -r <rule_spec> -t <n>
  -g --grid: wxh
  -p --palette: Combination of RGBYNW
  -r --rules: Combination of LR
  -t --times: Iterations. If negative, its complement will be used.
  -o --outfile: output file. Defaults to stdout.
  -h --help: Print this message and exit
  -v --verbose: Version number

Compile with -DSANITY_CHECK to enable runtime checks
Compile with -DUSE_TABLES to execute ant operations in separate functions
Compile with -DUSE_COL_MAJOR to traverse the grid in column-major order
```

Medimos el tiempo en ejecutar diez mil operaciones en la menor grilla posible, y repetimos escalando la cantidad de operaciones

```
time -p ./tp1_if -g 1x1 -p RGBW -r LLLL -t ((10 * 1000))  > /dev/null

time -p ./tp1_if -g 1x1 -p RGBW -r LLLL -t $((100 * 1000))  > /dev/null

time -p ./tp1_if -g 1x1 -p RGBW -r LLLL -t $((1000 * 1000))  > /dev/null
```

Repetimos, con una grilla significativamente mas grande

```
time ./tp1_if -g 1024x1024 -p RGBW -r LLLL -t $((10 * 1000))  > /dev/null

time ./tp1_if -g 1024x1024 -p RGBW -r LLLL -t $((100 * 1000))  > /dev/null

time ./tp1_if -g 1024x1024 -p RGBW -r LLLL -t $((1000 * 1000))  > /dev/null
```

4. Análisis de desempeño

Compilar la versión en C del programa con los grados de optimización `-O0`,`-O1`,`-O2` y `-O3`, conservando los distintos ejecutables. Graficar el tiempo utilizado por la versión que contiene código assembly y la versión en C compilada con las distintas optimizaciones, para una matriz de $1024 * 1024$ y con 10^n iteraciones, para n entre 4 y 9.

5. Condiciones de entrega

El trabajo práctico vence el 03/10/2019. Debe contener:

- Carátula especificando los datos y contacto de los integrantes del grupo (dirección de correo electrónico, *handle* de slack, ubicación del repositorio de código)
- Decisiones relevantes sobre el diseño, implementación y resolución
- Casos de prueba relevantes documentados
- Código fuente
- Diagramas ilustrando la estructura del `stack` de cada función
- Conclusiones con fundamentos reales
- Este enunciado

6. Recursos

- Hormiga de Langton: `https://es.wikipedia.org/wiki/Hormiga_de_Langton`
- Formato PPM: `http://netpbm.sourceforge.net/doc/ppm.html`
- Imagemagick `https://imagemagick.org/index.php`
- MIPS32 Instruction Reference `http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html`
- Convención de llamadas a funciones, disponibles en el grupo Yahoo de la cátedra (`Files >Material >Assembly MIPS >func_call_conv.pdf`)
- GDB Debugger `https://www.gnu.org/software/gdb/`
- Objdump `https://linux.die.net/man/1/objdump`