Trabajo Práctico 1A
Conjunto de instrucciones MIPS

Sebastián Blázquez, *Padrón 99673*
sebastian.blazquez96@gmail.com

Federico Cavazzoli, *Padrón 98533*
fedecavazzoli@gmail.com

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

Repositorio: https://github.com/seblaz/Orga-6620-TP1

Índice

1. Introducción

Este trabajo práctico consiste en estudiar la performance de una implementación de la Hormiga de Langton que denominaremos 'La hormiga artista' en un ambiente MIPS. Para ello se dispone de dos implementaciones:

1. Implementación base con if's.
2. Implementación alternativa usando tablas.

Las mismas se encuentran programadas en assembler MIPS y en C.

2. Implementación

2.1. Desarrollo del Código Fuente

En este trabajo práctico las dos funciones a implementar fueron new_orientation y move_forward. La primera se encarga de devolver una nueva orientación dada la orientación actual y una regla. Por otro lado la segunda avanza a la hormiga por la matriz, teniendo en cuenta los límites de la misma.

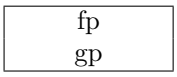
Lo más interesante de la implementación de dichas funciones fue el manejo del Stack con el ABI propuesto por la cátedra, dado fue necesario programar tanto funciones hojas como no-hojas.

2.2. Diagramas de stacks ifs

A continuación mostramos los diagramas de los stacks de las funciones implementadas para la versión con ifs.

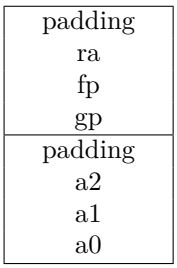
2.2.1. decide()

Función hoja sin variables en el stack, por lo tanto el stack es el mínimo:



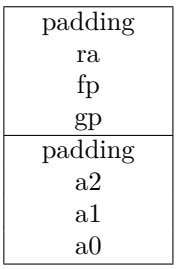
2.2.2. new_orientation()

Función no hoja sin variables en el stack (debido a la optimización realizada) que llama a decide con 3 argumentos, por lo tanto el stack es el siguiente:



2.2.3. move_forward()

Función no hoja sin variables en el stack que llama a adjust con 3 argumentos, por lo tanto el stack es el siguiente:

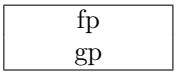


2.3. Diagramas de stacks tables

Luego para la versión con tablas los diagramas de stacks son los siguientes.

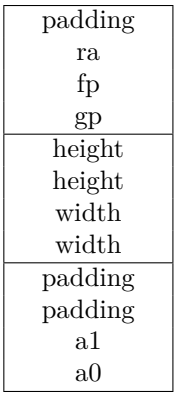
2.3.1. step_west(), step_east(), step_north() y step_south()

Todas estas funciones son similiares (hojas y sin variables en stack) por lo cual llevan un stack mínimo:



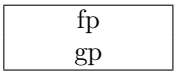
2.3.2. move_forward()

Esta es una función no hoja que guarda variables en el stack, por lo tanto su stack tendrá 16 bytes para guardar los registros correspondientes, 16 para las variables locales y otros 16 para los argumentos:



2.3.3. new_orientation()

Esta es una función hoja que no guarda variables en el stack, por lo tanto su stack es mínimo:



3. Proceso de Compilación

Junto con el código se entregó un archivo Makefile que indica las reglas de compilación ejecutadas por el comando make.

El código fuente se compila ejecutando el comando make con alguna de las posibles reglas.

make all: Crea los ejecutables tp1_if y tp1_tables.

make prof: Crea los ejecutables tp1_if_pf y tp1_tables_pf.

make tp1_if_asm: Crea el ejecutable tp1_if_asm.

make tp1_tables_asm: Crea el ejecutable tp1_tables_asm.

Para eliminar todos los archivos generados por el comando make, se puede ejecutar con la regla clean de la forma:

make clean

Para compilar con los distintos niveles de optimización se editó el archivo Makefile agregando los parámetros -O0, -01, -O2 y -O3.

Por otro lado los ejecutables tp1_if, tp1_tables, tp1_if_asm, tp1_tables_asm se utilizaron para realizar las mediciones con el comando time.

4. Portabilidad

Dado que para programar los ejecutables tp1_if y tp1_tables se utilizó el lenguaje de programación C, sin hacer uso de funciones específicas de sistemas operativos, o de bibliotecas comerciales, los programas pueden ser compilados en varios de ellos. De todas formas, el Makefile presentado fue hecho particularmente para sistemas de tipo UNIX.

Los casos de prueba presentados fueron llevados a cabo en una arquitectura MIPS emulada. El programa de emulación utilizado fue Qemu y el sistema operativo en la máquina emulada fue Debian.

tp1_if y tp1_tables también fueron probados en Ubuntu-Linux corriendo en una arquitectura Intel x86 satisfactoriamente. Sin embargo no se realizó lo mismo para tp1_if_asm y tp1_tables_asm dado que los mismos contienen partes programadas para el assembler de MIPS y no pueden ser ejecutadas en otra arquitectura.

5. Casos de Prueba

En este caso la cátedra proporcionó los casos de prueba que muestran el correcto funcionamiento de los ejecutables tp1_if y tp1_tables, los cuales pueden consultarse en el apéndice A. Para los respectivos ejecutables programados en assembler se probaron los mismos casos de prueba obteniendo los mismos resultados (aunque difieren en tiempo de ejecución).

5.1. Funcionamiento de la aplicación

Dado que trabajamos con memoria dinámica, tenemos que verificar que no haya fugas de memoria, ni ningún otro error posible. Para ello utilizamos la herramienta Valgrind en la máquina Host, en este caso es una máquina Linux con Ubuntu.

```
$ uname -a
Linux sebas-ThinkPad-T470 5.0.0-27-generic #28~18.04.1-Ubuntu SMP Thu Sep 26 00:10:46 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux

$ valgrind -v --tool=memcheck --leak-check=yes ./tp0_if -g 10000x10000 -p RGBWN -r RLRLL -t 10 > /dev/null

[...]

==21757== HEAP SUMMARY:
==21757==     in use at exit: 10 bytes in 2 blocks
==21757==   total heap usage: 6 allocs, 4 frees, 400,004,138 bytes allocated
==21757==
==21757== Searching for pointers to 2 not-freed blocks
==21757== Checked 70,456 bytes
==21757==
==21757== 5 bytes in 1 blocks are definitely lost in loss record 1 of 2
==21757==    at 0x4C31B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
```

```
==21757==    by 0x108E0F: xalloc (builders.c:25)
==21757==    by 0x10993E: main (artist_ant.c:139)
==21757==
==21757== 5 bytes in 1 blocks are definitely lost in loss record 2 of 2
==21757==    at 0x4C31B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==21757==    by 0x108E0F: xalloc (builders.c:25)
==21757==    by 0x109985: main (artist_ant.c:146)
==21757==
==21757== LEAK SUMMARY:
==21757==    definitely lost: 10 bytes in 2 blocks
==21757==    indirectly lost: 0 bytes in 0 blocks
==21757==      possibly lost: 0 bytes in 0 blocks
==21757==    still reachable: 0 bytes in 0 blocks
==21757==         suppressed: 0 bytes in 0 blocks
==21757==
==21757== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==21757== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Se ve que el programa pierde memoria (10 bytes), pero es muy poca comparada con la cantidad que se utilizó (400,004,138 bytes) con lo cual podemos ignorar el problema.

5.2. Influencia de los parámetros en el programa

En primer lugar notamos que al ejecutar el programa con los mismos parámetros varias veces obteníamos distintos tiempos de ejecución. Por ello al igual que en el tp0 para disminuir la varianza de las mediciones tomadas se decidió programar un pequeño script en bash que realice varias mediciones y luego calcule el promedio de las mismas (puede consultarse en el apéndice).

Por el enunciado debíamos utilizar una matriz de 1024x1024 e iteraciones de 10^n, con n entre 4 y 9, y se decidió utilizar los colores RGB y la regla LRL.

6. Análisis de desempeño

Realizamos un análisis de desempeño de las distintas implementaciones con los parámetros descriptos en la sección anterior. Para ello ejecutamos cada uno de los programas en los distintos niveles de optimización (0, 1, 2, y 3) 10 veces y calculamos sus promedios (en segundos):

Cuadro 1: implementación ifs

Cant de iteraciones	-O3	-O2	-O1	-O0	assembler
1000	3.580	3.585	3.585	3.648	3.422
10000	3.485	3.725	4.070	3.840	3.345
100000	3.840	3.835	3.790	3.975	3.615
1000000	5.195	5.535	5.095	6.890	6.545
10000000	20.985	21.795	27.69	35.655	32.465
100000000	186.575	181.975	189.47	333.33	260.64

Cuadro 2: implementación tables

Cant de iteraciones	-O3	-O2	-O1	-O0	assembler
1000	3.419	3.545	4.070	3.497	3.376
10000	3.535	3.550	3.935	3.595	3.280
100000	3.635	3.650	4.290	3.970	3.765
1000000	6.055	5.705	5.720	7.930	6.005
10000000	26.290	27.910	26.340	45.635	29.145
100000000	249.310	251.380	232.170	424.105	270.115

A continuación mostramos un gráfico de los tiempos de ejecución de las distintas implementaciones en función del nivel de optimización:

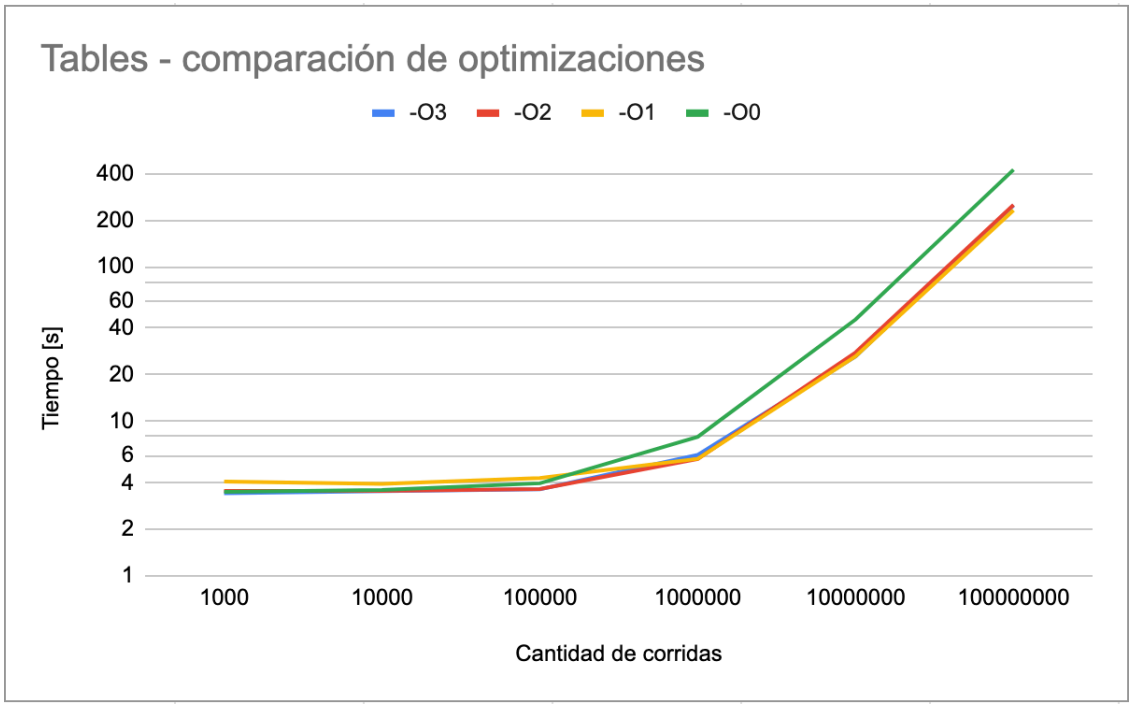


Figura 1: Gráfico del tiempo de ejecución de la implementación con tablas

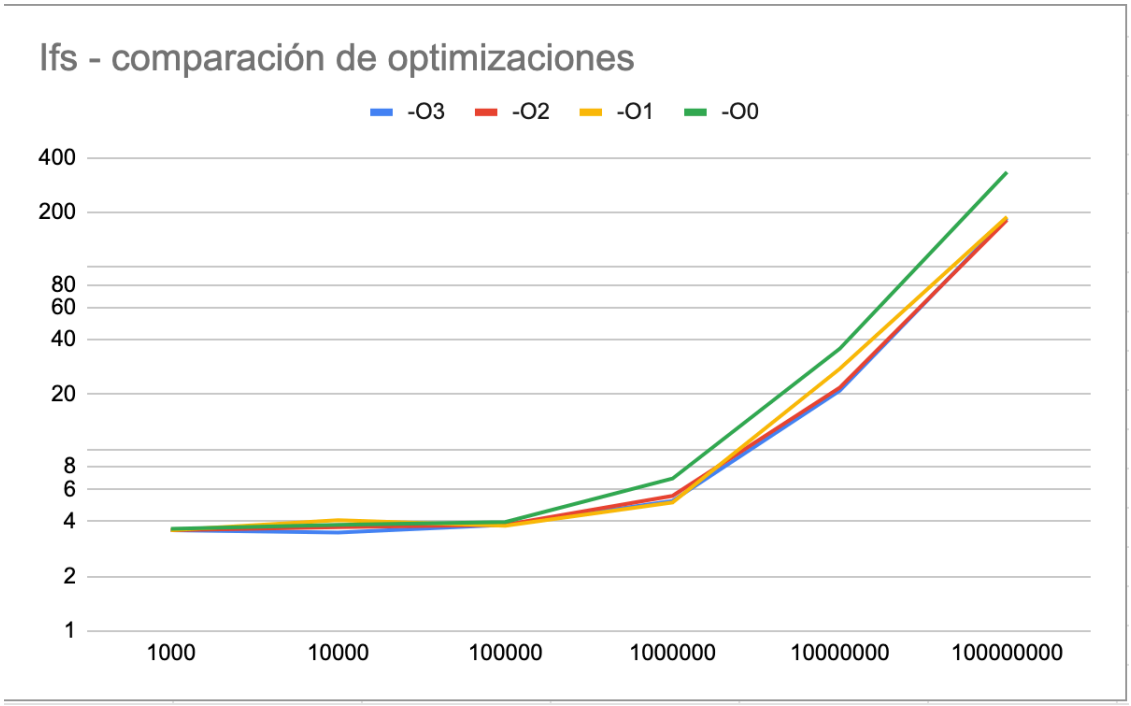


Figura 2: Gráfico del tiempo de ejecución de la implementación con if

En los gráficos anteriores queda en evidencia que sin importar el nivel de optimización resulta menos eficiente la implementación con tables que aquella realizada con ifs. Por ejemplo para la mayor cantidad de iteraciones el máximo tiempo de ejecución con tables pasa los 400 segundos y la ejecución de la implementación con ifs no llega a este valor.

También se puede ver que en ambas implementaciones luego de pasar de la optimización -O0 a la primera optimización -O1 hay una mejora de casi el doble, dado que con -O0 ambas están cercanas a los 400 segundos y con -O1 están cercanas a los 200.

Finalmente en ambos gráficos se puede observar que no se aprecian diferencias significativas entre -O1, -O2 y -O3.

A continuación se muestran los gráficos que representan la comparación de los métodos de tablas e if's para cada nivel de optimización.

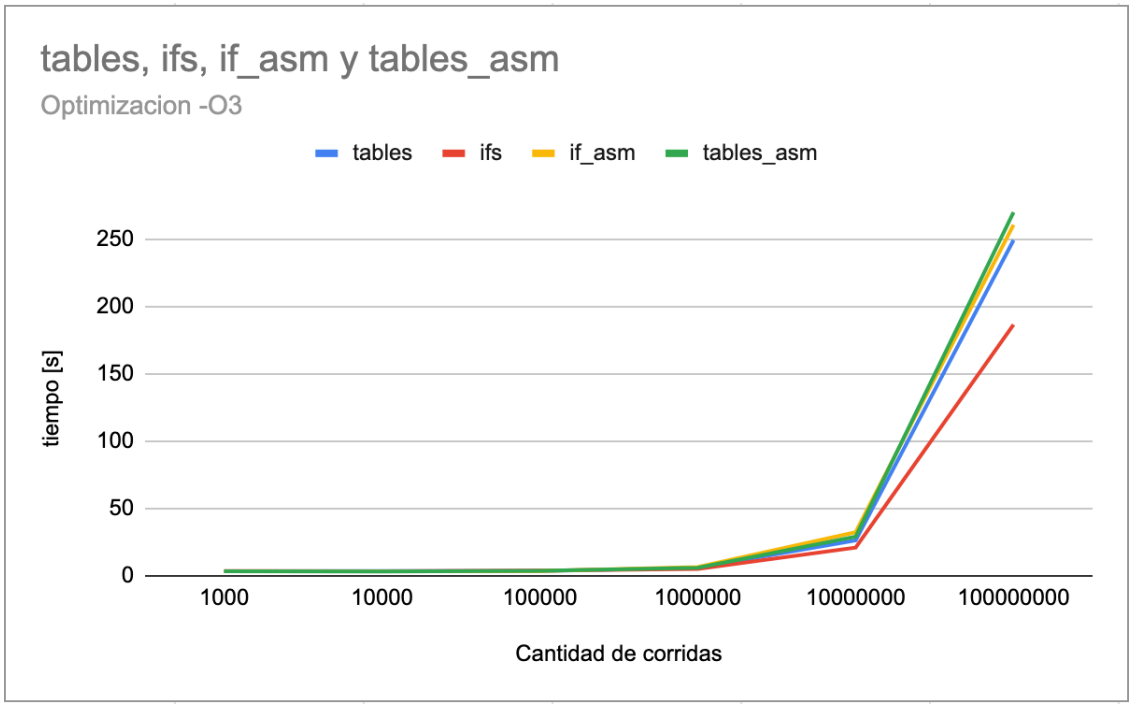


Figura 3: Optimización -O3 con código asembler

En el gráfico anterior se puede observar como en un principio, con pocas iteraciones, el desempeño de todas las implementaciones resulta similar. Pero cuando se pasan las 10 millones de iteraciones las diferencias comienzan a resultar significativas. Se puede notar que la implementación con ifs es siempre mejor que aquella realizada con tables y también queda en evidencia que con un nivel de optimización -O3 el código hecho en asembler por nosotros no resulto mas performante, si no que resulto mucho mas lento (casi 200 segundos de diferencia).

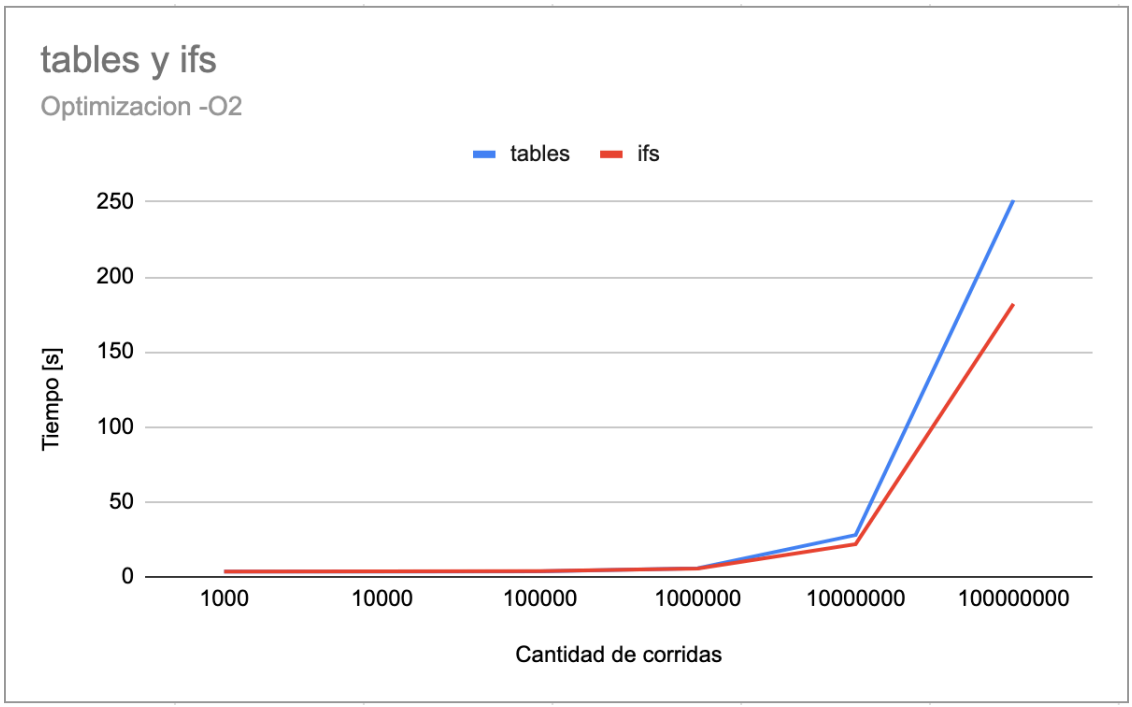


Figura 4: Gráfico del tiempo de ejecución con optimización -O2

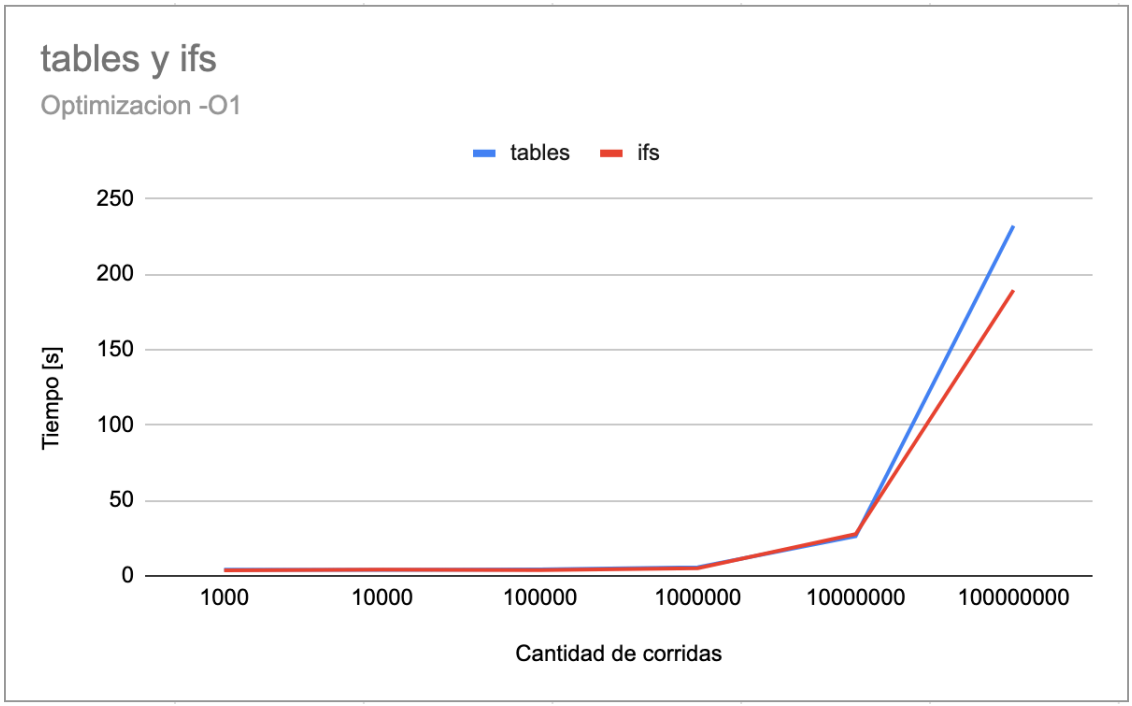


Figura 5: Gráfico del tiempo de ejecución con optimización -O1

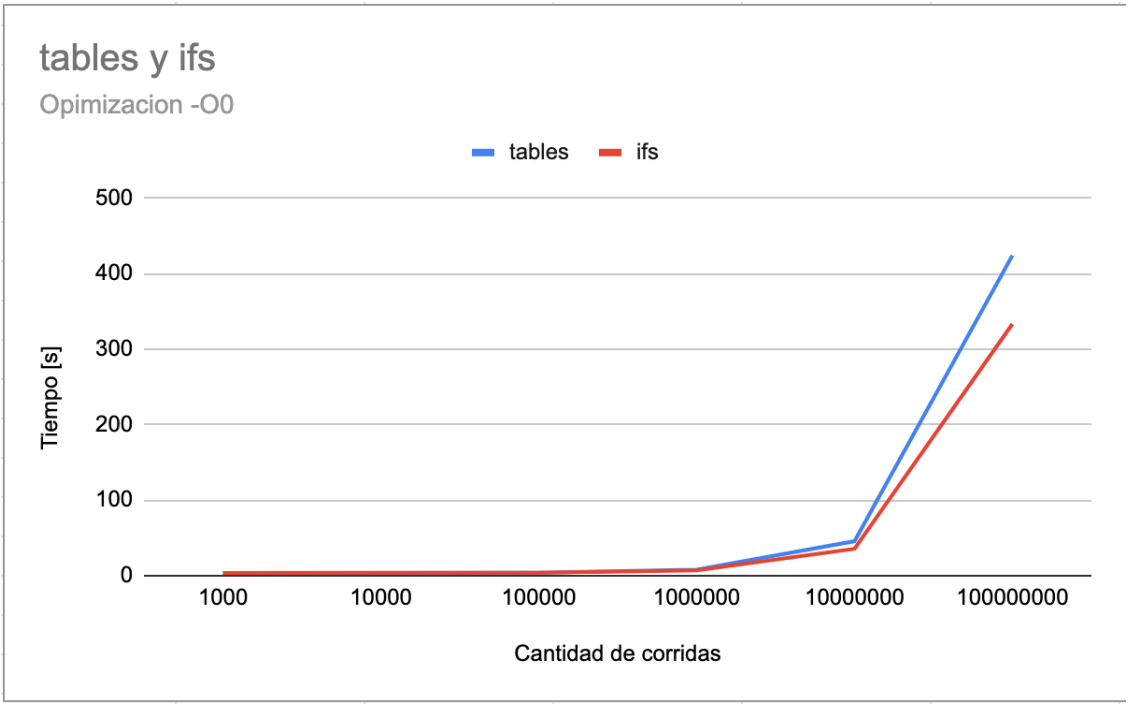


Figura 6: Gráfico del tiempo de ejecución con optimización -O0

En todos los gráficos anteriores se puede ver como a menos de 1 millón de iteraciones no es tan notable la diferencia de tiempos de ejecución de las distintas implementaciones, pero pasada esta cantidad la diferencia resulta notable y siempre el programa que utiliza tables es mas lento que aquel que usa ifs.

7. Conclusión

Al igual que en el tp0 a partir de los resultados obtenidos podemos concluir que las supuestas mejoras realizadas (de la implementación con tablas) en general no produjeron ninguna disminución en el tiempo de ejecución. Esto es debido a que en dichas mejoras se realiza la declaración e inicialización de una variable en memoria, la cual requiere mayor tiempo para ser accedida que la ejecuciónn de las instrucciones de salto.

Por otro lado una de las primeras observaciones que podemos realizar es que el nivel de optimización 0 es el que mayor tiempo de ejecución tiene. Este tiempo se va reduciendo conforme va aumentando el nivel de optimización, lo que resulta lógico porque en general cuanto mas optimizado esta el código de asembler mejor uso de los recursos del procesador se hará.

Luego vemos que con la implementación de tp_if_asm se ve que con optimización 0 es mejor que su par programado en C. Sin embargo esto cambia para el nivel 1, pero luego ambas implementaciones resultan muy similares para los niveles 2 y 3. Esto puede explicarse dado que en principio con pocas optimizaciones nuestro código es mejor que el compilado desde C, pero luego se emparejan a medida que se aplican los mismos a mayor nivel. Algo similar ocurre con la versión de tables, aunque siempre se mantiene la tendencia de que es peor que su versión de ifs.

Consecuentemente podemos concluir que con un assembler programado por el usuario no significa que se tendrán mejores resultados de performance. Resulta preferible compilar el código con mejores niveles de optimización y programar en C, que implementa las convenciones de stack necesarias para que el código sea compatible y provee un mayor grado de abstracción y facilidad.

Finalmente resulta pertinente justificar porque es que la implementación tables resulto siempre mas costosa (en tiempo) que la implementación con ifs y esto se debe a la forma de acceso a datos del programa. Cuando se utiliza el sistema tables se hacen muchos mas accesos a memoria que cuando se ejecuta el programa implementado con ifs. Como es tema sabido de la materia el acceso a memoria es muy costoso en tiempo.

Referencias

[1] Qemu. https://www.qemu.org/.

[2] Debian. https://www.debian.org/index.es.html.

[3] Vim the Editor. http://www.vim.org/.

[4] B. Kernighan, D.Ritchie, *The C Programming Languaje*. Person Prentice Hall

[5] B. Kernighan, R. Pike, *The Unix Programming Environment*. Prentice Hall

[6] Valgrind http://valgrind.org/

[7] See MIPS Run Linux, *Dominic Sweetman*, segunda edición.

[8] System V Application Binary Interface, *MIPS RISC Processor*, tercera edición.

8. Enunciado

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`

9. Código fuente

9.1. Script en bash para medir el tiempo promedio

```
#!/bin/bash

avg_time() {
    #
    # usage: avg_time n command ...
    #
    n=$1; shift
    (($# > 0)) || return                    # bail if no command given
    for ((i = 0; i < n; i++)); do
        { time -p "$@" > /dev/null; } 2>&1 # ignore the output of the command
                                            # but collect time's output in stdout
    done | awk '
        /real/ { real = real + $2; nr++ }
        /user/ { user = user + $2; nu++ }
        /sys/  { sys  = sys  + $2; ns++}
        END    {
                  if (nr>0) printf("real %.4f\n", real/nr);
                  if (nu>0) printf("user %.4f\n", user/nu);
                  if (ns>0) printf("sys %.4f\n",  sys/ns)
                }'
}

echo "con tables:"
avg_time $1 ./tp1_tables "${@:2}"
echo ""

echo "con ifs:"
avg_time $1 ./tp1_if "${@:2}"
echo ""

echo "con ifs hecho en asm:"
avg_time $1 ./tp1_if_asm "${@:2}"
echo ""

echo "con tables hecho en asm:"
avg_time $1 ./tp1_tables_asm "${@:2}"
echo ""
```

9.2. Assembler con implmentación de ifs

```
#include <regdef.h>

.data
error_str: .asciiz "Orientacion invalida"

.text
.align 2
.ent decide
decide:
    .frame fp, 8, ra
    .set noreorder
    addiu sp, sp, -8
    sw fp, 4(sp)
    sw gp, 0(sp)
    move fp, sp
```

```
    sw a0, 8(fp) # guardo a0 en el stack anterior
    sw a1, 12(fp) # guardo a1 en el stack anterior
    sw a2, 16(fp) # guardo a2 en el stack anterior

    li  t0, 1 # t0 = 1 = RIGHT
    beq a0, t0, go_right # d == RIGHT?
    nop
go_left:
    move v0, a1 # devuelvo a1 = go_left
    b return_decide
    nop
go_right:
    move v0, a2 # devuelvo a2 = go_right

return_decide:
    lw fp, 4(sp) # cargo el fp anterior
    lw gp, 0(sp) # cargo el gp anterior
    addiu sp, sp, 8 # dejo el stack como estaba
    jr ra # vuelvo a la rutina anterior
    nop
.set reorder
.end decide

.align 2
.globl new_orientation
.ent new_orientation
new_orientation:
    .frame fp, 32, ra  # stack de 32 bytes (16 bytes para SRA y 16 para ABA)
    .set noreorder
    .cpload t9  # gp en t9 con valor correcto
    addiu sp, sp, -32  # creo el stack de 32 bytes
    sw ra, 24(sp) # guardo ra (return address)
    sw fp, 20(sp) # guardo fp (frame pointer)
    move fp, sp # actualizo al nuevo fp
    .cprestore 16 # restablece el gp

    sw a0, 32(fp) # guardo a0 en el stack anterior
    sw a1, 36(fp) # guardo a1 en el stack anterior

    move t1, a1 # guardo la regla en otro registro

    beq a0, $0, norte # orientation = NORTH
    nop

    li t0, 1 # t0 = 1 (SOUTH)
    beq a0, t0, sur
    nop

    li t0, 2 # t0 = 2 (EAST)
    beq a0, t0, este
    nop

    li t0, 3 # t0 = 3 (WEST)
    beq a0, t0, oeste
    nop

    la a0, error_str # se frena la ejecucion
    jal doPanic

oeste:
```

```
    li a1, 1
    li a2, 0
    b call_decide
    nop
este:
    li a1, 0
    li a2, 1
    b call_decide
    nop
norte:
    li a1, 3
    li a2, 2
    b call_decide
    nop
sur:
    li a1, 2
    li a2, 3
call_decide:
    move a0, t1 # guardo rule en a0
    jal decide
    nop
    move sp, fp  # vuelvo a dejar el stack pointer como antes de llamar decide
    lw ra, 24(sp) # cargo el ra anterior.
    lw fp, 20(sp) # cargo el fp anterior.
    lw gp, 16(sp) # cargo el gp anterior.
    addiu sp, sp, 32 # deshago el stack.
    jr ra # vuelvo a la rutina que me llamo.
    nop
.set reorder
.end new_orientation

.text
.align 2
.globl move_forward
.ent move_forward
move_forward:
    .frame fp, 8, ra # stack minimo de 8 bytes
    .set noreorder
    .cpload t9
    addiu sp,sp,-8 # aumento el stack
    sw fp, 4(sp) # guardo el fp
    move  fp, sp # actualizo el fp
    .cprestore 0 # guardo el gp

    sw a0, 8(fp) # guardo a0 en el stack anterior
    sw a1, 12(fp) # guardo a1 en el stack anterior
    sw a2, 16(fp) # guardo a2 en el stack anterior

    lw  t0, 8(a0) # t0 = ant->o
    beq t0, $0, north # $0 = 0 = norte.
    nop

    li  t1, 1 # t1 = 1 = SOUTH
    beq t0, t1, south
    nop

    li t1, 2 # t1 = 2 = EAST
    beq t0, t1, east
    nop

    li t1, 3 # t1 = 3 = WEST
```

```
    beq t0, t1, west
    nop

    la a0, error_str  # se frena la ejecucion
    jal doPanic

north:
    lw t1, 4(a0) # t1 = ant->y
    addiu t1, t1, -1 # t1 = ant->y - 1
    divu $0, t1, a2 # ant->y - 1 / height
    mfhi t1 # t1 = modulo de la division anterior
    sw t1, 4(a0) # guardo el modulo en ant->y
    b return
    nop
south:
    lw t1, 4(a0) # t1 = ant->y
    addiu t1, t1, 1 # t1 = ant->y + 1
    divu $0, t1, a2 # ant->y - 1 / height
    mfhi t1 # t1 = modulo de la division anterior
    sw t1, 4(a0) # guardo el modulo en ant->y
    b return
    nop
east:
    lw t1, 0(a0)
    addiu t1, t1, 1
    divu $0, t1, a2
    mfhi t1
    sw t1, 0(a0)
    b return
    nop
west:
    lw t1, 0(a0)
    addiu t1, t1, -1
    divu $0, t1, a2
    mfhi t1
    sw t1, 0(a0)

return:
    move v0, a0 # devuelvo a0
    move sp, fp # dejo el sp como estaba antes (innecesario en este caso)
    lw fp, 4(sp)
    lw gp, 0(sp)
    addiu sp, sp, 8
    jr ra
    nop

.set reorder
.end move_forward
```

9.3. Assembler con implmentación de tables

```
#include <regdef.h>

.data
.align 2
error_str: .asciiz "Orientacion invalida"
allowed_forward:
    .word step_south
    .word step_east
    .word step_north
```

```
    .word step_west

rotation_rules:
    .word 3 # WEST
    .word 2 # EAST
    .word 2 # EAST
    .word 3 # WEST
    .word 0 # NORTH
    .word 1 # SOUTH
    .word 1 # SOUTH
    .word 0 # NORTH

.text
.align 2
.ent step_north
step_north:
    .frame fp, 8, ra # stack minimo
    .set noreorder
    addiu sp, sp, -8
    sw fp, 4(sp)
    sw gp, 0(sp)
    move fp, sp # fin setup stack

    sw a0, 8(fp) # guardo a0 en el stack anterior
    sw a1, 12(fp) # guardo a1 en el stack anterior

    lw t1, 4(a0) # t1 = ant->y
    addiu t1, t1, -1 # t1 = ant->y - 1
    divu $0, t1, a1 # ant->y - 1 / height
    mfhi t1 # t1 = modulo de la division anterior
    sw t1, 4(a0)

    lw fp, 4(sp) # cargo el fp anterior
    lw gp, 0(sp) # cargo el gp anterior
    addiu sp, sp, 8 # dejo el stack como estaba
    jr ra # vuelvo a la rutina anterior
    nop
.set reorder
.end step_north

.align 2
.ent step_south
step_south:
    .frame fp, 8, ra # stack minimo
    .set noreorder
    addiu sp, sp, -8
    sw fp, 4(sp)
    sw gp, 0(sp)
    move fp, sp # fin setup stack

    sw a0, 8(fp) # guardo a0 en el stack anterior
    sw a1, 12(fp) # guardo a1 en el stack anterior

    lw t1, 4(a0) # t1 = ant->y
    addiu t1, t1, 1 # t1 = ant->y + 1
    divu $0, t1, a1 # ant->y + 1 / height
    mfhi t1 # t1 = modulo de la division anterior
    sw t1, 4(a0)

    lw fp, 4(sp) # cargo el fp anterior
    lw gp, 0(sp) # cagor el gp anterior
```

```
    addiu sp, sp, 8 # dejo el stack como estaba
    jr ra # vuelvo a la rutina anterior
    nop
.set reorder
.end step_south

.align 2
.ent step_west
step_west:
    .frame fp, 8, ra # stack minimo
    .set noreorder
    addiu sp, sp, -8
    sw fp, 4(sp)
    sw gp, 0(sp)
    move fp, sp # fin setup stack

    sw a0, 8(fp) # guardo a0 en el stack anterior
    sw a1, 12(fp) # guardo a1 en el stack anterior

    lw t1, 0(a0) # t1 = ant->x
    addiu t1, t1, -1 # t1 = ant->x - 1
    divu $0, t1, a1 # ant->x - 1 / width
    mfhi t1 # t1 = modulo de la division anterior
    sw t1, 0(a0)

    lw fp, 4(sp) # cargo el fp anterior
    lw gp, 0(sp) # cargo el gp anterior
    addiu sp, sp, 8 # dejo el stack como estaba
    jr ra # vuelvo a la rutina anterior
    nop
.set reorder
.end step_west

.align 2
.ent step_east
step_east:
    .frame fp, 8, ra # stack minimo
    .set noreorder
    addiu sp, sp, -8
    sw fp, 4(sp)
    sw gp, 0(sp)
    move fp, sp # fin setup stack

    sw a0, 8(fp) # guardo a0 en el stack anterior
    sw a1, 12(fp) # guardo a1 en el stack anterior

    lw t1, 0(a0) # t1 = ant->x
    addiu t1, t1, 1 # t1 = ant->x + 1
    divu $0, t1, a1 # ant->x + 1 / width
    mfhi t1 # t1 = modulo de la division anterior
    sw t1, 0(a0)

    lw fp, 4(sp) # cargo el fp anterior
    lw  gp, 0(sp) # cargo el gp anterior
    addiu sp, sp, 8 # dejo el stack como estaba
    jr ra # vuelvo a la rutina anterior
    nop
.set reorder
.end step_east

.align 2
```

```
.globl new_orientation
.ent new_orientation
new_orientation:
    .frame fp, 8, ra  # stack minimo de 8 bytes (8 bytes para SRA)
    .set noreorder
    .cpload t9  # gp en t9 con valor correcto
    addiu sp, sp, -8  # creo el stack de 8 bytes
    sw fp, 4(sp) # guardo fp (frame pointer)
    move fp, sp # actualizo al nuevo fp
    .cprestore 0 # guardo el gp

    sw a0, 8(fp) # guardo a0 en el stack anterior
    sw a1, 12(fp) # guardo a1 en el stack anterior

    sll t1, a0, 3 # t1 = orientation * 8
    sll t3, a1, 2 # t3 = rule * 4
    add t1, t1, t3 # t1 = t1 + rule * 4
    la t2, rotation_rules # t2 = rotation_rules
    addu t2, t2, t1 # t2 = t2 + t1
    lw v0, 0(t2) # v0 = valor en la direccion de t2

    lw gp, 0(fp) # cargo gp guardado en stack
    move sp, fp  # vuelvo a dejar el stack pointer como antes de llamar a new_orientation
    lw fp, 4(sp) # cargo el fp anterior.
    addiu sp, sp, 8 # deshago el stack.
    jr ra # vuelvo a la rutina que me llamo.
    nop
.set reorder
.end new_orientation

.text
.align 2
.globl move_forward
.ent move_forward
move_forward:
.frame fp, 48, ra # stack de 40 bytes (16 para SRA, 16 para LTA, 16 para ABA)
    .set noreorder
    .cpload t9
    addiu sp, sp, -48 # aumento el stack
    sw ra, 40(sp) # guardo ra
    sw fp, 36(sp) # guardo el fp
    move  fp, sp # actualizo el fp
    .cprestore 32 # guardo el gp

    sw a0, 48(fp) # guardo a0 en el stack anterior
    sw a1, 52(fp) # guardo a1 en el stack anterior
    sw a2, 56(fp) # guardo a2 en el stack anterior

    sw a2, 28(fp) # height
    sw a2, 24(fp) # height
    sw a1, 20(fp) # width
    sw a1, 16(fp) # width
    move t8, a0
    lw t0, 8(a0) # t0 = ant->o
    sll t0, t0, 2 # t0 = ant->o * 4
    li t1, 28 # t1 = 28
    subu t1, t1, t0 # t1 = 28 - ant->0 * 4
    addu t2, fp, t1 # t2 = t1(fp)
    lw t2, 0(t2) # t2 = value of t2 = bound
    la t3, allowed_forward # t3 = address of allowed_forward
    addu t3, t3, t0 # t3 = allowed_forward + ant->o * 4
```

```
    lw t3, 0(t3) # t3 = value of t3 = go_forward (address of step func)

    move a1, t2 # a1 = bound
    jal t3 # call go_forward

    move v0, t8 # devuelvo a0
    move sp, fp # dejo el sp como estaba antes (innecesario en este caso)
    lw gp, 32(sp)
    lw fp, 36(sp)
    lw ra, 40(sp)
    addiu sp, sp, 48
    jr ra
    nop

.set reorder
.end move_forward
```