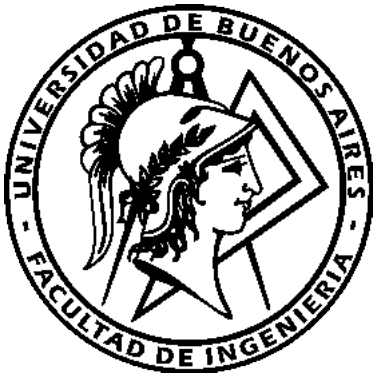
(66.20) Oganizacion de Computadoras: TP 1

Christian Angelone (93971) christiangelone@gmail.com

Agustin Gaillard (94849) agufiuba@gmail.com

1er cuatrimestre 2019



1 Introduccion

Este trabajo practico, trata de reflejar el impacto de performance que conlleva escribir ciertas partes de una aplicacion en assembly (en particular, Assembly MIPS). Para ello, se desarrollo una aplicacion que implementa el juego 'artist ant', tomando metricas de performance (tiempo de ejecucion, speedup) sobre una parte de su codigo, en C y luego en Assembly MIPS.

2 Disenio y Implementacion

Decidimos Diseniar la aplicacion de manera modular, aislando la implementacion de 'paint' para que esta sea facilmente intercambiable por cualquier variante. En cuanto a los generadores, decidimos dejarlas especificadas en un punto h e implementarlas en el archivo principal para ahorrar tiempo.

3 Instrucciones de Compilacion

Ejecutar:

```
$ gcc artist_ant.c paint.s -o artist_ant
```

4 Instrucciones de Ejecucion

Ejecutar:

```
root@debmips:~/tp1$ ./artist_ant -h
    ./artist_ant -g <dimensions> -p <colors> -r <rules> -n <n>
    -g --grid: wxh
    -p --palette: Combination of R|G|B|Y|N|W
    -r --rules: Combination of L|R
    -n --times: Iterations
    -h --help: Print this message and exit
    -v --verbose: Version number
```

5 Casos de prueba

```
root@debmips:~/tp1$ ./artist_ant -g '2x2' -p 'N|W' -r 'R|L' -n 1
P3
2 2
255
0   0   0   0   0   0
0   0   0   255 255 255
-----------------------------------
c: 10 ms assembly: 9 ms

root@debmips:~/tp1$ ./artist_ant -g '3x3' -p 'R|G|B' -r 'L|R|L' -n 3
P3
3 3
255
255 0   0   255 0   0   255 0   0
0   0   255 0   255 0   255 0   0
255 0   0   255 0   0   255 0   0
-----------------------------------
c: 11 ms assembly: 10 ms

root@debmips:~/tp1$ ./artist_ant -g '3x3' -p 'Y|N|B' -r 'L|R|L' -n 2
P3
3 3
255
255 255 0   255 255 0   255 255 0
0   0   255 0   0   0   255 255 0
255 255 0   255 255 0   255 255 0
-----------------------------------
```

```
c: 11 ms assembly: 10 ms
```

6 Resultados

En promedio, comparando tiempos de ejecucion entre implementar la funcion paint en C sin optimizaciones por un lado y en assembly por otro, solo notamos un speedup 1.1, es decir que solo tuvimos una mejora del 10

7 Conclusiones

- Al tener mayor granularidad al acceso a memory en assembly, es posible ver/hacer con mayor facilidad que el programa acceda a direcciones de memoria ya visitadas y asi pegarle a la cache en vez de a la RAM
- Entre menos llamados a funciones mas rapido es la ejecucion, dado que no se pierde tiempo en armado de stacks y guardado de variables.
- Nunca pases structs grandes por parametro a funcion/es, reventas el stack, usar siempre puntero al struct en cuestion.
- Programar netamente en assembly carece de sentido en cuestiones generales, dado que la mejora no seria perceptible en comparacion al resultado que podria generar el compilador. El unico motivo que encontramos para el uso de assembly puro, es un handleo de dirrecionamiento a memoria "raro" que desde c seria dificil realizar o usar instrucciones especificas a nivel assembly que el compilador no tiene en cuenta.