Trabajo Práctico N°0

"Infraestructura básica"

Belén Beltran, Padrón Nro. 91.718
belubeltran@gmail.com

Pablo Ariel Rodriguez, Padrón Nro. 93.970
prodriguez@fi.uba.ar

Federico Martín Rossi, Padrón Nro. 92.086
federicomrossi@gmail.com

2do. Cuatrimestre 2013
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Índice

1. Introducción

En el presente trabajo se tiene como objetivo la familiarización con las herramientas de software que se utilizarán a lo largo de los siguientes trabajos prácticos, llevando a cabo la implementación de un programa que resuelve cierta problemática piloto detallada en los próximos apartados.

La implementación se realizará en el lenguaje de programación C. Luego se ejecutará la aplicación sobre una plataforma *NetBSD/MIPS-32* mediante el emulador *GXEmul* [1]. Finalmente generaremos, mediante el compilador *GCC* [2], el equivalente en assembler MIPS del código fuente en C.

Todos los archivos y códigos fuente aquí mencionados, así como también el presente informe, pueden ser descargados como un archivo comprimido ZIP del repositorio del grupo[1].

2. Compilación

La herramienta para compilar el código en lenguaje C será el *GCC*.

Para automatizar las tareas de compilación se hace uso de la herramienta *GNU Make*. Los Makefiles utilizados para la compilación se incluyen junto al resto de los archivos fuentes del presente trabajo.

3. Utilización

En los siguientes apartados se especifica la forma en la que debe ser ejecutado el programa implementado en lenguaje C. El resultado de la compilación con "make" será un programa ejecutable, de nombre *tp0*, que podrá ser invocado con los siguientes parámetros:

- *-h*: Imprime ayuda para la utilización del programa;
- *-V*: Imprimer la versión actual del programa;
- *-r [Width]x[Height]*: Setea la resolución en píxeles del bitmap;
- *-o [Path]*: Especifica la ruta del archivo de salida sobre el cual se guarda el tablero generado por la aplicación.

4. Implementaciones

En lo que sigue de la sección, se presentarán los códigos fuente de las implementaciones de los algoritmos. Aquellos lectores interesados en la implementación completa de los dos programas, pueden dirigirse a los apéndices ubicados al final del presente informe. Recordamos que se han separado las implementaciones en dos de manera de poder mantener un orden entre ambos.

4.1. Implementación en C

La implementación del programa fue divida en los siguientes módulos:

- **tp0**: Programa principal responsable de interpretar los parámetros especificados a través de la terminal de modo de que realice las tareas solicitadas por el usuario. Su función es, de acuerdo al parámetro o los parámetros especificados, llevar a cabo la ejecución solicitada haciendo uso de módulos externos;
- **pgm**: Módulo encargado de generar una imagen de un tablero de ajedrez (cuadrado, de ocho casilleros de lado) conforme a los parámetros ingresados. Se utiliza para ello el formato gráfico PGM o *portable gray map*. De especificarse un archivo de salida, la imagen en formato PGM se almacenará en este medio.

4.1.1. Algoritmo *Bubblesort*

En el *Código* **??** se muestra el header de la librería, donde se declara la función Bubblesort, mientras que en el *Código* **??** se muestra la definición de la librería.

[1] URI del Repositorio: `https://github.com/federicomrossi/6620-trabajos-practicos-2C2013/tree/master/tp0`

4.1.2. Algoritmo *Shellsort*

En el *Código* **??** se muestra el header de la librería, donde se declara la función Shellsort, mientras que en el *Código* **??** se muestra la definición de la librería.

4.2. Implementación en Assembly

La implementación del programa fue divida en los siguientes módulos:

- **tp0**: Solamente recibe un texto como argumento por linea de comandos y lo imprime ordenandolo mediante shellsort. Esta implementado en C;
- **fileloader**: Se utiliza el mismo de la otra implementación;
- **swap**: Función implementada en assembler para intercambiar el valor de dos registros recibidos como parametros;
- **compare**: Función similar a strcmp de C. Recibe dos argumentos de tipo texto y decide cual es el que precede alfabeticamente.
- **shellsort**: Implementación en assembler del algoritmo de ordenamiento.

4.2.1. Algoritmo *Shellsort*

En el *Código* **??** se muestra la implementación en assembly del algoritmo Shellsort.

5. Debugging

Para analizar el correcto funcionamiento de los programas se crearon programas adhoc que fueran corroborando el correcto funcionamiento de cada uno de los modulos de manera individual. La idea era simular el concepto de test unitario de lenguajes de alto nivel.

Una vez que todos los módulos funcionaban de la manera esperada, se utilizó el programa principal como test de integración. Finalmente se utilizó la herramienta *Valgrind* para realizar un análisis minucioso del uso de memoria, corrigiendo así las pérdidas de memoria que se presentaban en cada módulo.

6. Pruebas

Para realizar las pruebas de cada algoritmo se procesaron cuatro textos provistos por la cátedra. Estos son "*Alicia en el País de las Maravillas*" (173kB), "*Beowulf*" (220kB), una "*Enciclopedia*" (643kB) y "*Don Quijote*" (2147kB).

A cada ejecución se le midió el tiempo de procesamiento mediante el comando *GNU "time"* [**?**]. A continuación un ejemplo de su utilización:

```
$ time ./tp0 -b alice.txt
```

Esto imprimirá por pantalla el tiempo de ejecución que fue tabulado para cada caso. En el *Cuadro 1* se muestran los valores de tiempo obtenidos.

Archivo de texto	bubblesort.c [s]	shellsort.c [s]	shellsort.S [s]
alice.txt	571,715	2,207	1,035
beowulf.txt	958,527	3,418	1,500
cyclopedia.txt	>3600 *(timeout)*	12,184	6,914
elquijote.txt	>3600 *(timeout)*	48,957	33,098

Cuadro 1: *Tiempos en segundos obtenidos en la ejecución de los distintos algoritmos.*

En los graficos que siguen se muestran distintas comparaciones hechas con estos datos obtenidos, los cuales nos llevarán a la conclusión final.

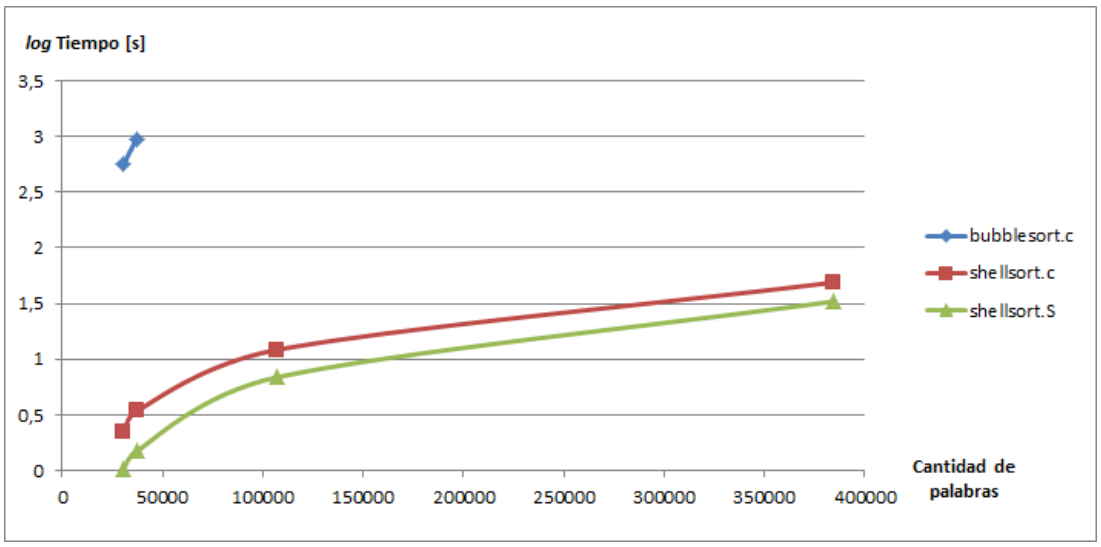

Figura 1: *Gráfico de tiempo insumido (en segundos) contra el tamaño de la muestra (cantidad de palabras)*

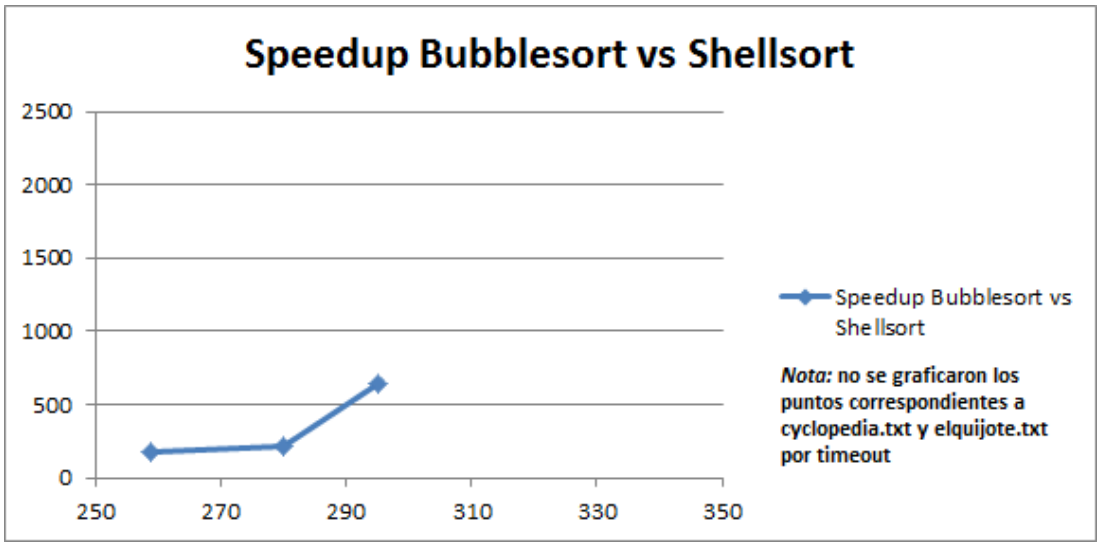

Figura 2: *Gráfico del speedup del Bubblesort contra Shellsort para los diversos tamaños de archivo.*

7. Conclusiones

Al realizar la comparación entre los algoritmos bubblesort y shellsort en C, vemos que la diferencia entre ambos es notable. Estamos hablando de varios ordenes de maginitud.

Esto se debe a que por su codigo bubblesort tiene un coste de $O(n^2)$ mientras que si bien no podemos determinarlo fácilmente para shellsort, ya que depende mucho del gap elegido, estamos hablando de algo cercano a $O(nlog(n))$. Esto hace que para textos grandes no sea una opción viable la utilización de bubblesort, mientras que shellsort prueba ser muy eficiente y el esfuerzo de codificación no es demsiado grande.

En cuanto a la comparación de assembler y C, notamos una ligera ventaja del primero. Creemos que esto se debe a que muchas de las cosas que el algoritmo en C utiliza como variables en memoria principal, assembler utiliza registros los cuales presentan un tiempo de lecto/escritura notablemente menor.

Referencias

[1] The NetBSD project, `http://www.netbsd.org/`

[2] GCC, the GNU Compiler Collection, `http://gcc.gnu.org/`

[3] PGM format speciffication, `http://netpbm.sourceforge.net/doc/pgm.html`

[4] J. L. Hennessy and D. A. Patterson, "Computer Architecture. A Quantitative Approach," 4th Edition, Morgan Kaufmann Publishers, 2000.

Apéndices

A. Implementación completa en lenguaje C

A.1. ***tp0.c*. Implementación del main del programa**

A.2. ***bubblesort.h*. Declaración de la librería Bubblesort**

A.3. ***bubblesort.c*. Definición de la librería Bubblesort**

A.4. ***shellsort.h*. Declaración de la librería Shellsort**

A.5. ***shellsort.c*. Definición de la librería Shellsort**

A.6. ***fileloader.h*. Declaración de la librería File Loader**

A.7. ***fileloader.c*. Definición de la librería File Loader**

B. Implementación completa de Shellsort en assembly MIPS

B.1. ***tp0.c*. Implementación del main del programa**

B.2. ***shellsort.S*. Implementación del algoritmo Shellsort**

B.3. ***compare.S*. Implementación del comparador de palabras**

B.4. ***swap.S*. Implementación del comparador de palabras**