Universidad Zaragoza

Proyecto Hardware
Practica 1

Desarrollo combinado y optimización de programas en multiples lenguajes de alto y bajo nivel

Guillermo Robles González - NIP: 604409
Sergio Martín Segura - NIP: 622612

Supervisado por:
Javier Resano Ezcaray (coordinador)
María Villarroya Gaudó
Enrique Torres Moreno
Jesús Alastruey Benedé
Darío Suárez Gracia

13 de diciembre de 2015

Índice

1. Resumen **2**

2. Introducción **2**

3. Objetivos **2**

4. Metodología **3**

4.1. Diseño

Dedicación: aprox 2 horas 3

4.2. Implementación

Dedicación: aprox 3 horas 3

4.3. Control de Resultados

Dedicación: aprox 3 horas 3

4.4. Obtención de métricas

Dedicación: aprox 2 horas 3

5. Resultados **4**

6. Conclusiones **5**

6.1. Margen de mejora 6

7. Bibliografía **7**

1. Resumen

El trabajo que se realiza es la programación de una serie de funciones y rutinas, en varios lenguajes de bajo nivel y uno de alto nivel, para la ayuda a la realización del conocido puzle sudoku. Este trabajo se enmarca dentro de la asignatura de Proyecto Hardware de 3 del grado superior en Ingeniería informática por la Universidad de Zaragoza. Además, se comparará la eficiencia relativa de los distintos lenguajes, y se verá la capacidad de los compiladores modernos de realizar compilación cruzada, permitiéndonos combinar código escrito en múltiples lenguajes.

2. Introducción

Sudoku es un puzle desarrollado en 1970, muy popular en Japón desde 1986 y conocido mundialmente desde 2005. El objetivo del sudoku es rellenar una cuadrícula de 9 x 9 celdas (81 casillas) dividida en subcuadrículas de 3 x 3 (también llamadas cajas o regiones)con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas de las celdas. Aunque se podrían usar colores, letras, figuras, se conviene en usar números para mayor claridad, lo que importa, es que sean nueve elementos diferenciados, que no se deben repetir en una misma fila, columna o subcuadrícula. Por el propio diseño del puzle es parte primordial de su resolución conocer qué números pueden ser candidatos de ocupar una casilla concreta y, por extensión, todas las demás casillas. Por eso nuestro proyecto es desarrollar un asistente electrónico que de forma automática sugiera los candidatos para cada celda dado un puzle concreto. En esta primera parte, el trabajo consiste en la creación de una serie de rutinas que, dada una cuadrícula decidirá cuales son los posibles candidatos para cada casilla vacía.

3. Objetivos

Aunque el objetivo principal del proyecto sea la creación de un sencillo programa de ayuda a la resolución del popular puzzle. Desde un punto de vista pedagógico existen otros objetivos no estrictamente relacionados con el objetivo principal:

- Aprender a desarrollar codigo para microcontroladores en múltiples lenguajes (tanto de alto como de bajo nivel), y conocer las tecnicas que existen para el entrelazado y compilado conjunto de un programa escrito en múltiples lenguajes.
- Comparar a nivel puramente técnico múltiples lenguajes y combinaciones de los mismos, principalmente mediante la métrica "nº de instrucciones de procesador para terminar una tarea"
- Realización justificada de toma de decisiones sobre el proyecto, tanto en aspectos de diseño como de implementación
- Documentar correctamente el desarrollo y características del producto en cuestión

4. Metodología

Se ha usado una metodología de prototipado rápido, basada en la creación rápida de prototipos funcionales, que se iran refinando mediante revisiones sucesivas, hasta llegar a una versión que abarque toda la funcionalidad deseada y cumpla los objetivos de velocidad, eficiencia o calidad deseados.
Esta metodología permite obtener rápidamente versiones funcionales, ya sea para testeo o discusión con el cliente, sin embargo sufre el defecto de que una vez establecido un algoritmo, tiende a ser difícil cambiarlo. Se ha evitado este defecto revisando los algoritmos usados y la forma de implementarlos con cuidado, y realizando a veces reimplementaciones completas de ciertos trozos del programa.

4.1. Diseño

Dedicación: aprox 2 horas

El primer paso para afrontar el proyecto ha sido la estructuración y división del problema para solucionar sus partes de modo ordenado y coherente.
Con técnicas como la división en ficheros de las rutinas y la definición (pre-condición y post-condición) previa a la implementación hemos logrado construir un código bien estructurado de una forma ágil y sencilla sin preocuparnos en el momento del diseño de los problemas de implementación que irían surgiendo.

4.2. Implementación

Dedicación: aprox 3 horas

La implementación se ha realizado conjuntamente, no realizando reparto de tareas, ya que se ha considerado que dado el tamaño del problema en cuestión no merecía la pena.

4.3. Control de Resultados

Dedicación: aprox 3 horas

Para el control de resultados se han utilizado 3 tablas de sudoku, una de ellas aportada con el trabajo, y las otras obtenidas de revistas de pasatiempos.
Dada la metodología usada, se realizaban pruebas constantemente, midiéndose tanto la correctitud de los algoritmos, como la eficiencia de los mismos.
El control de resultados se realizó de manera manual, con algo de ayuda de utilidades de tratamiento de texto.

4.4. Obtención de métricas

Dedicación: aprox 2 horas

Para las mediciones de correctitud, primero se realizó y comprobó el código en C, siendo esta nuestra implementación base, y a partir de esa implementación se resolvieron las cuadriculas a utilizar, las cuales fueron comprobadas manualmente. A continuación, mediante la función de exportado de zonas de memoria de Eclipse y la utilidad diff (que permite comparar archivos de texto o binarios y resaltar las diferencias) se comprobaba si las implementaciones

en otros lenguajes eran correctas, comparando su salida con la salida de ejemplo.
Para la medición del tiempo de ejecución (medida poco útil en un entorno real, dado que depende del sistema de prueba, pero adecuado para comparar las distintas implementaciones) se repitio la ejecución de las rutinas en cuestion varias veces, dado que tardaban demasiado poco para ser medidas normalmente. Para ello se utilizó un cronómetro externo al sistema, que se iniciaba y detenía manualmente, por ello, se ralizan varias medidas, para evitar sesgo humano.
Para las mediciones de tamaño de código (tanto estático como dinámico) se ha usado la ayuda del debugger gdb, mediante la interfaz gráfica ofrecida por el IDE Eclipse.

5. Resultados

En la escritura de las pruebas se utiliza la notación c_a para indicar que la función llamadora (en nuestro caso la que recorre la cuadrícula) esta realizada en C, y la función hoja (en nuestro caso, la que analiza una casilla particular y marca sus candidatos) esta en ARM; de la misma forma se usa la sigla t para funciones Thumb.
Todas las medidas están en segundos

Tiempos de ejecución:

- Tiempo de ejecución

  1000 pruebas:

Función	$t_1(s)$	$t_2(s)$	$t_3(s)$	$t_4(s)$	$t_5(s)$	Media (s)	TPE[1] (s)
c_c	16.02	16.23	16.27	16.19	16.11	16.16	0.001616
c_a	3.22	2.8	2.76	2.73	2.73	2.848	0.000285
c_t	3.57	3.39	3.51	3.51	3.51	3.498	0.000349
a_c	16.43	17.12	15.92	15.73	15.85	16.21	0.001621
a_a	2.5	2.63	2.48	2.54	2.57	2.544	0.000254
a_t	3.22	3.2	3.26	3.16	3.32	3.232	0.000323

[1]Tiempo Por Ejecución de la subrutina en cuestión

10000 pruebas:

Función	$t_1(s)$	$t_2(s)$	$t_3(s)$	$t_4(s)$	$t_5(s)$	Media (s)	TPE[1] (s)
c_c	155.12	152.56	157.72	158.64	158.64	156.54	0.001565
c_a	23.26	23.13	23.20	23.19	23.19	23.194	0.000232
c_t	31.53	31.41	31.98	31.36	31.36	31.528	0.000315
a_c	155.26	155.26	156.32	156.69	156.69	156.69	0.001567
a_a	21.86	22.32	22.07	22.05	22.05	22.07	0.000221
a_t	27.41	28.08	29.04	28.95	28.95	28.486	0.000285

[1]Tiempo Por Ejecución de la subrutina en cuestión

- Tamaño de código

  En la comparación se ha medido la función candidatos, que a partir de una casilla obtiene los números que pueden ir allí. Se ha usado como base la casilla (0,2) del sudoku dado como ejemplo, dado que es la primera casilla vacía.

	Tamaño	
Función	estático (bytes)	dinámico (instrucciones)
sudoku_candidatos_c	676	2634
sudoku_candidatos_arm	288	349
sudoku_candidatos_thumb (Con prólogo)	254	480
sudoku_candidatos_thumb (Sin prólogo)	214	489

En este caso se han medido los tamaños de código de las funciones que calculan la tabla de sudoku completa; ha sido imposible la medición del tamaño dinámico por razones de coste computacional

Función	Tamaño estático (bytes)
sudoku_recalcular_c_c	172
sudoku_recalcular_c_a	172
sudoku_recalcular_c_t [1]	172
sudoku_recalcular_a_c	80
sudoku_recalcular_a_a	80
sudoku_recalcular_a_t	100

[1]Es de notar que esta función no llama a sudoku_candidatos_thumb directamente, dada la imposibilidad de llamar a funciones en thumb desde C; sino a una función especial sudoku_candidatos_thumb_prologo, que está escrita en ARM y ejecuta la función thumb deseada

6. Conclusiones

La primera observación que se hace es la inmensa diferencia en eficiencia entre C y ensamblador, incluso en algoritmos sencillos como este, sin embargo la diferencia de dificultad al realizar los programas es considerable. Es de notar que la principal diferencia es en la función candidatos, que es la que más carga aporta al programa; también puede verse como el llamador es poco importante, obteniéndose diferencias de apenas unas pocas cienmilésimas de segundo entre las llamadas a c_a y a_a.

La elección de C como función hoja es impensable, dado que se obtienen tiempos del orden de 6-7 veces mayores, una posibilidad sería utilizar una compilación mas agresiva en la eficiencia, que aunque dificulta la comprensión del código generado, generalmente es más eficiente. En la situación en la que estamos (Interacción con un humano) la diferencia de eficiencia entre ASM y Thumb es despreciable, por lo tanto el factor que determinará la elección será el tamaño de la memoria disponible, escogiéndose Thumb en caso de que esta sea baja, y ARM en caso contrario.

En conclusión, se observa que la forma que balancea mejor comodidad de desarrollo y eficiencia es la programación del cuerpo principal de la aplicación en un lenguaje de alto nivel (C

en nuestro caso), y la programación de las subrutinas especialmente costosas en un lenguaje de bajo nivel (ensamblador).

6.1. Margen de mejora

Para la tarea para la cual se va a utilizar (un sistema que incluye interacción con un ser humano) la velocidad de las subrutinas es suficientemente alta, sin embargo, si comparamos nuestra implementación con implementaciones de algunos compañeros, vemos que aún queda muchas posibilidades de mejorar en ese aspecto.

7. Bibliografía

- Manuales de consulta ofrecidos por el profesorado
- `https://es.wikipedia.org/wiki/Sudoku`
- `http://infocenter.arm.com/help/index.jsp` (Especialmente el manual de referencia de ARM7)
- `http://www.arm.com/products/processors/classic/arm7/index.php`
- `http://www.sudoku-solutions.com/`