

Práctica 1

Proyecto Hardware. Curso 2021-22

Juan Plo Andrés 795105 · Ignacio Ortega Lalmolda 610720

EINA - UNIZAR

20 de octubre de 2021

1 ÍNDICE DE CONTENIDOS

1	ÍNDICE DE CONTENIDOS	1
2	RESUMEN EJECUTIVO	2
2.1	Resumen.....	2
3	INTRODUCCIÓN	3
3.1	Mapa de memoria del LPC2105	4
4	OBJETIVOS	5
4.1	Objetivo del proyecto	5
5	METODOLOGÍA	6
5.1	Pasos realizados.....	6
5.2	Esquema del proyecto.....	6
5.2.1	Ficheros del proyecto	6
5.2.2	Funciones que componen el proyecto	7
5.2.3	Código desarrollado en C	8
5.2.4	Código desarrollado en ARM y optimizaciones realizadas	10
6	RESULTADOS	19
6.1	Resultados con la opción de compilación -O0.....	19
6.1.1	Resultados obtenidos.....	19
6.1.2	Análisis de rendimiento	19
6.2	Resultados con la opción de compilación -O1.....	20
6.2.1	Resultados obtenidos.....	20
6.2.2	Análisis de rendimiento	20
6.3	Resultados con la opción de compilación -O2.....	20
6.3.1	Resultados obtenidos.....	20
6.3.2	Análisis de rendimiento	21
6.4	Resultados con la opción de compilación -O3.....	21
6.4.1	Resultados obtenidos.....	21
6.4.2	Análisis de rendimiento	21
7	CONCLUSIONES	22

2 RESUMEN EJECUTIVO

2.1 Resumen

El trabajo desarrollado ha consistido en la realización de un sistema de ayuda al cálculo de los candidatos de las celdas vacías de un sudoku programado de distintas maneras para comprobar después cual de ellas realiza el trabajo de manera más rápida y eficiente. Para ello se han escrito una serie de funciones en C y ensamblador ARM para su posterior comparación.

Una vez obtenidos los resultados se compararán entre sí para comprobar que funciones resultan más eficientes en tiempo y en tamaño. A la hora de comparar los resultados de las funciones escritas en C se compararán según las distintas opciones de compilación para comprobar si merece la pena el esfuerzo de escribir el código en ARM o, si bien, el compilador por si solo genera el código más óptimo en tiempo y espacio.

Finalmente, se verificará como el código en ensamblador resulta más rápido en tiempo de ejecución que el código original en C en todos los escenarios de compilación (-O0, -O1, -O2) salvo en -O3, en donde la versión en C será más rápida en tiempo de ejecución, pero no así en tamaño del código generado. También, se ha de tener en cuenta el coste en tiempo y esfuerzo que ha de realizar el programador para realizar el programa en ensamblador, si el compilador lo hace más rápido ¿Merece la pena el esfuerzo? O, aunque no lo hiciese más rápido ¿Merece la pena por ahorrarse unos milisegundos?

3 INTRODUCCIÓN

El trabajo realizado ha consistido en la implementación y optimización del rendimiento una serie de funciones para ayudar al cálculo de los candidatos de los huecos vacíos en un tablero de sudoku para, posteriormente, analizar los resultados obtenidos. A partir de un código original en C, se ha implementado el código equivalente en ensamblador para el procesador LPC2105, y se ha comprobado el rendimiento de las distintas soluciones. El entorno de trabajo empleado ha sido el emulador Keil μ Vision el cual también simula el procesador con el que hemos trabajado.

Las reglas del sudoku que se emplean en la implementación son sencillas, el objetivo es el de rellenar las celdas vacías en un tablero (9x9) de tal modo que un número del 1-9 sólo pueda aparecer una vez en su fila, su columna y en su región (3x3). Nuestro programa calculará los candidatos posibles de cada celda del tablero(9x9) de acuerdo a las reglas descritas anteriormente para ayudar a los jugadores a tomar la decisión de que valor puede ir en cada celda.

3.1 Mapa de memoria del LPC2105

El espacio de memoria del LPC2105 aparece dividido en las siguientes zonas. En la figura 1 se representa dicho mapa de memoria, así como las posiciones que ocupan las funciones de nuestro programa.

AHB PERIPHERALS	0xFFFF FFFF
VPB PERIPHERALS	0xF000 0000 0xE000 0000
RESERVED ADDRESS SPACE	0xE000 0000 0xDFFF FFFF
BOOT BLOCK (RE-MAPPED FROM ON-CHIP F.MEM.)	0x8000 0000 0x7FFF FFFF
RESERVED ADDRESS SPACE	0x7FFF E000 0x7FFF DFFF
32 KBYTE ON-CHIP STATIC RAM	0x4000 8000 0x4000 7FFF
RESERVED ADDRESS SPACE	0x4000 0000 0x3FFF FFFF
128 KBYTE ON-CHIP FLASH MEMORY	0x0002 0000 0x0001 FFFF
sudoku9x9	0x0000 0927
main	0x0000 0860 0x0000 085F
esVacía	0x0000 0824 0x0000 0823
eliminaCandidatos	0x0000 0800 0x0000 07FF
cuadrícula_candidatos_verificar	0x0000 07f0 0x0000 07EF
celda_eliminar_candidato	0x0000 0768 0x0000 0767
candidatos_propagar_c	0x0000 0750 0x0000 074F
candidatos_actualizar_c_arm	0x0000 064C 0x0000 064B
candidatos_actualizar_c	0x0000 0580 0x0000 057F
128 KBYTE ON-CHIP FLASH MEMORY	0x0000 04b4 0x0000 0000

Figura 1: Mapa de memoria

4 OBJETIVOS

4.1 Objetivo del proyecto

El objetivo principal del proyecto es el de poder comparar los distintos niveles de compilación en C y compararlos con nuestra versión ARM para detectar cual es la función más costosa en tiempo y comprobar si el compilador es más rápido y está mejor optimizado que nuestro código ARM según los diferentes niveles de compilación:

- -O0: Sin ninguna optimización
- -O1: Pequeñas optimizaciones tratando de reducir el tamaño del código generado, pero no que no reduce apenas el tiempo de ejecución. Opción de compilación intermedia entre -O0 y -O2.
- -O2: Más optimizaciones que -O1
- -O3: Todas las optimizaciones posibles. Aumenta el tamaño del código.

Se comparará según los niveles de compilación en tiempo y memoria para determinar si merece la pena el esfuerzo de escribir el código en ARM o, si bien, el trabajo realizado por el compilador es suficiente.

También se comprobará que todas las versiones del código tienen una funcionalidad equivalente y que todas las versiones obtienen el mismo resultado tras su ejecución variando únicamente en tiempo y en espacio.

5 METODOLOGÍA

5.1 Pasos realizados

Lo primero que realizamos fue escribir el código en C de candidatos actualizar y comprobar su correcto funcionamiento.

A continuación, comenzamos a escribir el código en ensamblador de *"candidatos_actualizar_arm_c"* y de *"candidatos_propagar_arm"* y, de nuevo, verificar su correcto funcionamiento. Algunas decisiones tomadas fueron las de tratar de usar el mínimo número de instrucciones y reutilizar valores de los registros, previamente cargados, etc.

Finalmente, una vez que el sistema funcionaba correctamente, comenzamos a realizar las métricas y a comparar los resultados.

5.2 Esquema del proyecto

5.2.1 Ficheros del proyecto

En la siguiente tabla se resumen los ficheros que componen el proyecto:

FICHEROS EN EL PROYECTO		
Fichero	Líneas	Descripción
sudoku_2021.uvproj		Fichero que contiene el proyecto para el emulador Keil uVision
sudoku_2021.c	182	Contiene el programa principal que realiza las llamadas a todas las funciones, así como las funciones escritas en C
sudoku_2021.h	75	Fichero de cabeceras de las funciones de sudoku_2021.c
celda.h	56	Contiene las operaciones que se aplican a una celda
funciones.s	426	Contiene el código de las funciones desarrolladas en ensamblador
startup.s		Fichero de inicio con parámetros para el compilador
tableros.h	75	Contiene una serie de tableros en hexadecimal con la situación inicial de cada uno para cada versión. También incluye un tablero solución para verificar la corrección de cada versión ejecutada.
Carpeta Listings		Generada por el entorno de desarrollo. Contiene el mapa de memoria de las funciones (dirección de memoria de inicio y final, tamaño, etc..)
Carpeta Objects		Generada por el entorno de desarrollo. Contiene los binarios generados al compilar el proyecto

5.2.2 Funciones que componen el proyecto

En la siguiente tabla se incluyen las funciones que componen cada fichero, así como una pequeña descripción de su comportamiento y parámetros.

FUNCIONES EN EL PROYECTO		
Fichero	Funciones	Descripción
sudoku_2021.c	main	realiza la llamada a sudoku9x9 pasándole los tableros de cada versión.
	sudoku9x9	Recibe como parámetros los diferentes tableros. Realiza llamadas a "candidatos_actualizar": <i>c_arm, c, arm, arm_c</i> , pasándole a cada uno su tablero. Devuelve 4 si todos los tableros coinciden con el tablero solución al finalizar. Devuelve -1 en caso contrario.
	cuadrícula_candidatos_verificar	Recibe como parámetro un tablero y el tablero solución. Verifica la corrección de cada celda comparándola la celda con la solución. Devuelve 1 si el resultado es correcto.
	candidatos_actualizar_c	Versión del código en C. Recibe como parámetro una celda. Calcula todas las listas de candidatos (tras borrar o cambiar un valor) y llama a "candidatos_propagar_c". Devuelve el número de celdas vacías.
	candidatos_actualizar_c_arm	Igual que la anterior solo que llama a "candidatos_propagar_arm".
	candidatos_propagar_c	Versión en C. Recibe como parámetro una celda, su fila y su columna. Propaga el valor de una celda para actualizar la lista de candidatos de su fila, columna y región. Recibe la celda a propagar como parámetro.
celda.h	celda_eliminar_candidato	Recibe como parámetro el puntero a una celda y un valor. Elimina el candidato que coincida con valor.
	celda_poner_valor	Recibe como parámetro el puntero a una celda y un valor. Pone como valor el valor recibido.
	celda_leer_valor	Recibe como parámetro una celda. Devuelve el valor almacenado en ella.
	esVacia	Recibe como parámetro una celda. Devuelve 0 si su valor es igual a 0.
	eliminaCandidatos	Recibe como parámetro el puntero a una celda. Pone a 0 todos sus candidatos.
funciones.s	candidatos_actualizar_arm_c	Versión en ARM de "candidatos_actualizar_c" la cual también llama a "candidatos_propagar_c".
	candidatos_propagar_arm	Versión en ARM de "candidatos_propagar_c".
	candidatos_actualizar_arm	Versión en ARM de "candidatos_actualizar_c" incluyendo en su propio código a "candidatos_propagar_arm", sin llamadas a función.

5.2.3 Código desarrollado en C

A continuación, se presenta el código nuevo desarrollado en C que se ha añadido al proyecto.

A. Candidatos actualizar C

CANDIDATOS ACTUALIZAR C

```
/* *****  
 * calcula todas las listas de candidatos (9x9)  
 * necesario tras borrar o cambiar un valor (listas corrompidas)  
 * retorna el numero de celdas vacias */  
  
/* Init del sudoku en codigo C invocando a propagar en C  
 * Recibe la cuadrícula como primer parametro  
 * y devuelve en celdas_vacias el número de celdas vacias  
 */  
static int candidatos_actualizar_c(CELDA cuadrícula[NUM_FILAS][NUM_COLUMNAS])  
{  
    int celdas_vacias = 0;  
    uint8_t i;  
    uint8_t j;  
  
    //borrar todos los candidatos  
    for (i=0; i < NUM_FILAS; i++) {  
        for (j=0; j < NUM_FILAS; j++) {  
            if (esVacia(cuadrícula[i][j])) {  
                eliminaCandidatos(&cuadrícula[i][j]);  
            }  
        }  
    }  
  
    //recalcular candidatos de las celdas vacias calculando cuantas hay vacias  
    for (i=0; i < NUM_FILAS; i++) {  
        for (j=0; j < NUM_FILAS; j++) {  
            if (esVacia(cuadrícula[i][j])) {  
                celdas_vacias++;  
            } else {  
                candidatos_propagar_c(cuadrícula, i, j);  
            }  
        }  
    }  
  
    //retornar el numero de celdas vacias  
    return celdas_vacias;  
}
```

B. Es vacía

ES VACÍA

```
/* *****  
 * devuelve 0 si y sólo si celda_leer_valor(celda) = 0 */  
__inline static uint8_t esVacia(CELDA celda)  
{  
    return celda_leer_valor(celda) == 0;  
}
```

C. Elimina candidatos

ELIMINA CANDIDATOS

```
/* *****  
 * modifica la celda poniendo a 0 todos sus candidatos (todos son candidatos) */  
void eliminaCandidatos(CELDA *celdaptr) {  
    *celdaptr = *celdaptr & 0x007F;  
}
```

5.2.4 Código desarrollado en ARM y optimizaciones realizadas

Respecto a la versión en C una de las optimizaciones con las que cuentan todas las funciones es el “inlining” de todas las funciones de celda.h, es decir, que el cuerpo de las funciones de celda.h está embebido directamente en el código ensamblador, evitando llamadas a subrutinas evitándose así las instrucciones que suponen. Otra de las optimizaciones que se realizan es el uso de los flags de condición de las instrucciones, para evitar saltos “branch” e instrucciones cmp antes de los saltos como se muestra en la siguiente tabla:

OPTIMIZACIÓN CON FLAGS	
Instrucciones originales	Instrucciones optimizadas
<pre>;esVacía ;----- ldrh r5,[r0] and r4,r5,#0x000F ;----- cmp r4,#0 bne noEliminar ;eliminaCandidatos si es vacía ;----- ldrh r5,[r0] and r5,r5,#0x007F strh r5,[r0] noEliminar (...)</pre>	<pre>;esVacía ;----- ldrh r5,[r0] ands r4,r5,#0x000F ;----- ;eliminaCandidatos si es vacía ;----- ldrheq r5,[r0] andeq r5,r5,#0x007F strheq r5,[r0]</pre>

Además, los bucles realizados en vez de ser, por ejemplo, desde 0 hasta NUM_FILAS se realizan desde NUM_FILAS hasta 0, ahorrando un número de instrucciones igual a NUM_FILAS por bucle como se muestra en la tabla a continuación:

OPTIMIZACIÓN DE BUCLES	
Bucle original	Bucle optimizado
<pre>mov r7,#NUM_FILAS for1 ;for(j=0;j>NUM_FILAS;j++) (...) add r5,r5,#1 cmp r5,r7 bne for1</pre>	<pre>mov r7,#NUM_FILAS for1 ;for(j=NUM_FILAS;j>0;j--) (...) subs r7,r7,#1 bne for1</pre>

A continuación, se presenta el código nuevo desarrollado en ARM que se ha añadido al proyecto.

D. Candidatos actualizar ARM

CANDIDATOS ACTUALIZAR ARM C

Parámetros recibidos

Registro	Descripción
r0	@ini_cuadrícula

Código en ARM

```

candidatos_actualizar_arm_c
    PUSH {r4-r10,lr}           ;regs a usar y lr
    mov r9,#0                  ;r9 = celdas_vacias
    mov r7,#NUM_FILAS          ;r7 = i (AMBAS, numero de iteraciones del
                                primer for y para direccionar la fila de la
                                celda)

    subs r7,r7,#1              ;como el valor de NUM_FILAS es 9,
                                le resto 1 para poder direccionar bien las
                                celdas

    mov r6,r0                  ;como la dirección de la celda necesita estar
                                en r0 para pasarlo como parámetro a esVacia y
                                eliminaCandidatos

b11    mov r8,#NUM_FILAS        ;r8 = j (AMBAS, numero de iteraciones del
                                segundo for y para direccionar la columna de
                                la celda)

    subs r8,r8,#1              ;como el valor de NUM_FILAS es 9, le resto 1
                                para poder direccionar bien las celdas

b21    mov r0,r6                ;recupero el valor de la primera celda para
                                calcular la que toca

    add r0,r0,r8,LSL#1
    add r0,r0,r7,LSL#5          ;calculo la dirección de la celda

    ;esVacia
    ;-----
    ldrh r5,[r0]                ;r5 = 31-16(DONT CARE) 15-0(INFORMACION DE LA
                                CELDA)

    ands r4,r5,#0x000F          ;r4 = valor de la celda
    ;-----

    ;eliminaCandidatos si es vacia
    ;-----
    andeq r5,r5,#0x000F
    strheq r5,[r0]
    ;-----

    subs r8,r8,#1              ;una iteración del segundo for
    bpl b21                    ;acaba cuando j < 0 para calcular la columna 0

    subs r7,r7,#1              ;una iteración del primer for
    bpl b11                    ;acaba cuando i < 0 para calcular la fila 0

    ;SEGUNDO PAR DE BUCLES ANIDADOS

```

	<pre> ;----- mov r7,#NUM_FILAS subs r7,r7,#1 mov r6,r0 </pre>	<pre> ;r7 = i (AMBAS, numero de iteraciones del primer for y para direccionar la fila de la celda) ;como el valor de NUM_FILAS es 9, Le resto 1 para poder direccionar bien las celdas ;como la dirección de la celda necesita estar en r0 para pasarlo como parámetro a esVacía y eliminaCandidatos </pre>
b12	<pre> mov r8,#NUM_FILAS subs r8,r8,#1 </pre>	<pre> ;r8 = j (AMBAS, numero de iteraciones del segundo for y para direccionar la columna de la celda) ;como el valor de NUM_FILAS es 9, Le resto 1 para poder direccionar bien las celdas </pre>
b22	<pre> mov r0,r6 add r0,r0,r8,LSL#1 add r0,r0,r7,LSL#5 ;esVacía ;----- ldrh r5,[r0] ands r4,r5,#0x000F ;----- addeq r9,r9,#1 movne r0,r6 movne r1,r7 movne r2,r8 blne candidatos_propagar_c subs r8,r8,#1 bpl b22 subs r7,r7,#1 bpl b12 mov r0,r9 POP {r4-r10,pc} </pre>	<pre> ;recupero el valor de la primera celda para calcular la que toca ;calculo la dirección de la celda ;r5 = 31-16(DONT CARE) 15-0(INFORMACION DE LA CELDA) ;r4 = valor de la celda ;si estaba vacía sumo 1 a celdas vacías ;si la celda estaba vacía, propago sus candidatos(r0 = dirección celda, r1 = i, r8 = j) ;una iteración del segundo for ;acaba cuando j < 0 para calcular la columna 0 ;una iteración del primer for ;acaba cuando i < 0 para calcular la fila 0 ;se pasan las celdas vacías a r0 para devolverlas ;restaurar los registros usados y vuelta del pc a sudoku_2021.c </pre>

E. Candidatos propagar ARM

CANDIDATOS PROPAGAR ARM	
Parámetros recibidos	
Registro	Descripción
r0	@ini_cuadrícula
r1	fila
r2	columna
Código en ARM	
<pre> candidatos_propagar_arm PUSH {r4-r12,lr} ; registros a usar ; obtener la cuadrícula[filas][columnas] add r6,r0,r1,LSL#5 ; r6 = @ini_fila add r5,r6,r2,LSL#1 ; r5 = cuadrícula[filas][columnas] ldrh r12,[r5] ; r12 = cuadrícula[filas][columnas] = mem[@ini_fila + col*2] (r6+r5 = @cuadrícula[filas][columnas]) ; valor = celda_leer_valor and r12,#0x00F ; r12 = valor ; ----- for1: mov r7,#NUM_FILAS ; r7 = j = 9 ; for(j=NUM_FILAS;j>0;j--) ; celda_eliminar_candidato mov r11,#0x40 ; 0000 0000 0100 0000 mov r11,r11,LSL r12 ; muevo el 1 tantas posiciones como valor y así obtengo la máscara para eliminar el candidato ldrh r4,[r6] ; r4 = mem[r6] orr r4,r4,r11 ; r4 = or bit a bit para poner a 1 el bit de r4 que está a 1 en r11 y no tocar el resto strh r4,[r6] ; mem[r6] = r4 ; ----- add r6,r6,#2 ; avanza a la columna siguiente subs r7,r7,#1 ; r7 = j-- (condición de fin de bucle j = 0) bne for1 ; ----- mov r7,#NUM_FILAS ; r7 = i = 9 sub r5,r5,r1,LSL#5 ; r5 = @ini_col for2: ; for(i=NUM_FILAS;i>0;i--) ; celda_eliminar_candidato mov r11,#0x40 ; 0000 0000 0100 0000 mov r11,r11,LSL r12 ; muevo el 1 tantas posiciones como valor y así obtengo la máscara para eliminar el candidato ldrh r4,[r5] ; r4 = mem[r5] orr r4,r4,r11 ; r4 = or bit a bit para poner a 1 el bit de r4 que está a 1 en r11 y no tocar el resto </pre>	

	<pre> strh r4,[r5] ;----- add r5,r5,#32 subs r7,r7,#1 bne for2 ;----- ;Encontrar el inicio de La región ;init_i = init_region[fila]; ;init_j = init_region[columna]; ;end_i = init_i + 3; ;end_j = init_j + 3; LDR r6,=init_region add r7,r6,r1 add r8,r6,r2 ldrb r7,[r7] ldrb r8,[r8] add r9,r7,#3 add r10,r8,#3 </pre>	<pre> ;mem[r5] = r4 ;r5 = cuadrícula[filas + 1][columna] ;r7 = i-- (condición de fin de bucle i = 0) ;r7 = init_i (fila) ;r8 = init_j (col) ;r9 = end_i ;r10 = end_j </pre>
for3	<pre> ;for(i=init_i; i<end_i; i++) mov r6,r8 ;Calcula @cuadrícula[ini_i][ini_j] add r5,r0,r7,LSL#5 add r5,r5,r8,LSL#1 </pre>	<pre> ;avanza fila a fila ;se restaura en r6 el valor de la columna para repetir for4 </pre>
for4	<pre> ;for(j=init_j; j<end_j; j++) ;celda_eliminar_candidato mov r11,#0x40 mov r11,r11,LSL r12 ldrh r4,[r5] orr r4,r4,r11 strh r4,[r5] ;----- add r5,r5,#2 add r6,r6,#1 cmp r6,r10 bne for4 ;----- add r7,r7,#1 cmp r7,r9 bne for3 ;----- POP {r4-r12,pc} </pre>	<pre> ;avanza columna a columna ;0000 0000 0100 0000 ;muevo el 1 tantas posiciones como valor y así obtengo la máscara para eliminar el candidato ;r4 = mem[r5] = valor cuadrícula ;r4 = or bit a bit para poner a 1 el bit de r4 que está a 1 en r11 y no tocar el resto ;mem[r5] = r4 ;avanza r6 en la región ;avanza en la columna ;se compara r6 (init_j que va avanzando por las columnas) con r10 end_j ;avanzar la fila en para for3 ;se compara r7 (init_i que va avanzando por las filas) con end_i ;restaurar los registros usados y vuelta del pc a sudoku_2021.c </pre>

F. Candidatos actualizar ARM

CANDIDATOS ACTUALIZAR ARM	
Parámetros recibidos	
Registro	Descripción
r0	@ini_cuadrícula
Código en ARM	
<pre> candidatos_actualizar_arm_c PUSH {r4-r12,lr} mov r9,#0 mov r7,#NUM_FILAS subs r7,r7,#1 mov r6,r0 arm_arm_b11 mov r8,#NUM_FILAS subs r8,r8,#1 arm_arm_b21 mov r0,r6 add r0,r0,r8,LSL#1 add r0,r0,r7,LSL#5 ;esVacia ldrh r5,[r0] ands r4,r5,#0x000F ;----- ;eliminaCandidatos si es vacia andeq r5,r5,#0x007F strheq r5,[r0] ;----- subs r8,r8,#1 bpl arm_arm_b21 subs r7,r7,#1 bpl arm_arm_b11 ;SEGUNDO PAR DE BUCLES ANIDADOS mov r7,#NUM_FILAS </pre>	
<pre> ;regs a usar y lr ;r9 = celdas_vacias ;r7 = i (AMBAS, numero de iteraciones del primer for y para direccionar la fila de la celda) ;como el valor de NUM_FILAS es 9, le resto 1 para poder direccionar bien las celdas ;como la direccion de la celda necesita estar en r0 para pasarlo como parametro a esVacia y eliminaCandidatos ;r8 = j (AMBAS, numero de iteraciones del segundo for y para direccionar la columna de la celda) ;como el valor de NUM_FILAS es 9, le resto 1 para poder direccionar bien las celdas ;recupero el valor de la primera celda para calcular la que toca ;calculo la direccion de la celda ;r5 = 31-16(DONT CARE) 15-0(INFORMACION DE LA CELDA) ;r4 = valor de la celda ;una iteracion del segundo for ;acaba cuando j < 0 para calcular la columna 0 ;una iteracion del primer for ;acaba cuando i < 0 para calcular la fila 0 ;r7 = i (AMBAS, numero de iteraciones del primer for y para direccionar la fila de la </pre>	

	<i>celda)</i>
<code>subs r7,r7,#1</code>	<i>;como el valor de NUM_FILAS es 9, Le resto 1 para poder direccionar bien las celdas</i>
<code>mov r6,r0</code>	<i>;como la direccion de la celda necesita estar en r0 para pasarlo como parametro a esVacia y eliminaCandidatos</i>
<code>arm_arm_b12 mov r8,#NUM_FILAS</code>	<i>;r8 = j (AMBAS, numero de iteraciones del segundo for y para direccionar la columna de la celda)</i>
<code>subs r8,r8,#1</code>	<i>;como el valor de NUM_FILAS es 9, Le resto 1 para poder direccionar bien las celdas</i>
<code>arm_arm_b22 mov r0,r6</code>	<i>;recupero el valor de la primera celda para calcular la que toca</i>
<code>add r0,r0,r8,LSL#1</code>	
<code>add r0,r0,r7,LSL#5</code>	<i>;calculo la direccion de la celda</i>
<code>;esVacia</code>	
<code>ldrh r5,[r0]</code>	<i>;r5 = 31-16(DONT CARE) 15-0(INFORMACION DE LA CELDA)</i>
<code>ands r4,r5,#0x000F</code>	<i>;r4 = valor de la celda</i>
<code>;-----</code>	
<code>addeq r9,r9,#1</code>	<i>;si estaba vacia sumo 1 a celdas vacias</i>
<code>movne r0,r6</code>	
<code>beq noPropagar</code>	
<code>mov r1,r7</code>	
<code>mov r2,r8</code>	
<code>push {r6-r9}</code>	
<code>;candidatos_propagar_arm</code>	
<code>;=====</code>	
<code>;obtener la cuadrícula[filas][columnas]</code>	
<code>add r6,r0,r1,LSL#5</code>	<i>;r6 = @ini_filas</i>
<code>add r5,r6,r2,LSL#1</code>	<i>;r5 = cuadrícula[filas][columnas]</i>
<code>ldrh r12,[r5]</code>	<i>;r12 = cuadrícula[filas][columnas] = mem[@ini_filas + col*2] (r6+r5 = @cuadrícula[filas][columnas])</i>
<code>;valor = celda_leer_valor</code>	
<code>and r12,#0x000F</code>	<i>;r12 = valor</i>
<code>;-----</code>	
<code>mov r7,#NUM_FILAS</code>	<i>;r7 = j = 9</i>
<code>arm_arm_for1 ;for(j=NUM_FILAS;j>0;j--)</code>	
<code>;celda_eliminar_candidato</code>	
<code>mov r11,#0x40</code>	<i>;0000 0000 0100 0000</i>
<code>mov r11,r11,LSL r12</code>	<i>;muevo el 1 tantas posiciones como valor y así obtengo la máscara para eliminar el candidato</i>
<code>ldrh r4,[r6]</code>	<i>;r4 = mem[r6]</i>
<code>orr r4,r4,r11</code>	<i>;r4 = or bit a bit para poner a 1 el bit de r4</i>

	<i>que está a 1 en r11 y no tocar el resto</i>
<code>strh r4,[r6]</code>	<code>;mem[r6] = r4</code>
<code>;-----</code>	
<code>add r6,r6,#2</code>	<i>;avanza a la columna siguiente</i>
<code>subs r7,r7,#1</code>	<i>;r7 = j-- (concición de fin de bucle j = 0)</i>
<code>bne arm_arm_for1</code>	
<code>;-----</code>	
<code>mov r7,#NUM_FILAS</code>	<i>;r7 = i = 9</i>
<code>sub r5,r5,r1,LSL#5</code>	<i>;r5 = @ini_col</i>
<code>arm_arm_for2 ;for(i=NUM_FILAS;i>0;i--)</code>	
<code>;celda_eliminar_candidato</code>	
<code>mov r11,#0x40</code>	<i>;0000 0000 0100 0000</i>
<code>mov r11,r11,LSL r12</code>	<i>;muevo el 1 tantas posiciones como valor y así obtengo la máscara para eliminar el candidato</i>
<code>ldrh r4,[r5]</code>	<i>;r4 = mem[r5]</i>
<code>orr r4,r4,r11</code>	<i>;r4 = or bit a bit para poner a 1 el bit de r4 que está a 1 en r11 y no tocar el resto</i>
<code>strh r4,[r5]</code>	<i>;mem[r5] = r4</i>
<code>;-----</code>	
<code>add r5,r5,#32</code>	<i>;r5 = cuadrícula[fila + 1][columna]</i>
<code>subs r7,r7,#1</code>	<i>;r7 = i-- (concición de fin de bucle i = 0)</i>
<code>bne arm_arm_for2</code>	
<code>;-----</code>	
<code>;Encontrar el inicio de la región</code>	
<code>;init_i = init_region[fila];</code>	
<code>;init_j = init_region[columna];</code>	
<code>;end_i = init_i + 3;</code>	
<code>;end_j = init_j + 3;</code>	
<code>LDR r6,=init_region</code>	
<code>add r7,r6,r1</code>	
<code>add r8,r6,r2</code>	
<code>ldrb r7,[r7]</code>	<i>;r7 = init_i (fila)</i>
<code>ldrb r8,[r8]</code>	<i>;r8 = init_j (col)</i>
<code>add r9,r7,#3</code>	<i>;r9 = end_i</i>
<code>add r10,r8,#3</code>	<i>;r10 = end_j</i>
<code>arm_arm_for3 ;for(i=init_i; i<end_i; i++)</code>	<i>;avanza fila a fila</i>
<code>mov r6,r8</code>	<i>;se restaura en r6 el valor de la columna para repetir for4</i>
<code>;Calcula @cuadrícula[init_i][init_j]</code>	
<code>add r5,r0,r7,LSL#5</code>	
<code>add r5,r5,r8,LSL#1</code>	
<code>arm_arm_for4 ;for(j=init_j; j<end_j; j++)</code>	<i>;avanza columna a columna</i>
<code>;celda_eliminar_candidato</code>	
<code>mov r11,#0x40</code>	<i>;0000 0000 0100 0000</i>
<code>mov r11,r11,LSL r12</code>	<i>;muevo el 1 tantas posiciones como valor y así obtengo la máscara para eliminar el candidato</i>

	<code>Ldrh r4,[r5]</code>	<code>;r4 = mem[r5] = valor cuadrícula</code>
	<code>orr r4,r4,r11</code>	<code>;r4 = or bit a bit para poner a 1 el bit de r4</code>
	<code>strh r4,[r5]</code>	<code>que está a 1 en r11 y no tocar el resto</code>
	<code>;-----</code>	<code>;mem[r5] = r4</code>
	<code>add r5,r5,#2</code>	<code>;avanza r6 en la región</code>
	<code>add r6,r6,#1</code>	<code>;avanza en la columna</code>
	<code>cmp r6,r10</code>	<code>;se compara r6 (init_j que va avanzando por</code>
	<code>bne arm_arm_for4</code>	<code>las columnas) con r10 end_j</code>
	<code>;-----</code>	
	<code>add r7,r7,#1</code>	<code>;avanzar la fila en para for3</code>
	<code>cmp r7,r9</code>	<code>;se compara r7 (init_i que va avanzando por</code>
	<code>bne arm_arm_for3</code>	<code>las filas) con end_i</code>
	<code>;-----</code>	
	<code>pop{r6-r9}</code>	
	<code>;=====</code>	
noPropagar	<code>subs r8,r8,#1</code>	<code>;una iteración del segundo for</code>
	<code>bpl arm_arm_b22</code>	<code>;acaba cuando j < 0 para calcular la columna 0</code>
	<code>subs r7,r7,#1</code>	<code>;una iteración del primer for</code>
	<code>bpl arm_arm_b12</code>	<code>;acaba cuando i < 0 para calcular la fila 0</code>
	<code>mov r0,r9</code>	<code>;se pasan las celdas vacías a r0 para</code>
		<code>devolverlas</code>
	<code>POP {r4-r12,pc}</code>	<code>;restaurar los registros usados y vuelta del pc</code>
		<code>a sudoku_2021.c</code>

6 RESULTADOS

A continuación, se muestran y analizan los resultados obtenidos para los distintos niveles de compilación. Notar que la versión "*candidatos_actualizar_arm*" va a arrojar en todos los casos los mismos resultados debido a que está escrita directamente en ensamblador por lo que el nivel de compilación no le afecta.

6.1 Resultados con la opción de compilación -O0

Opción de compilación por defecto. En esta opción no se aplica ninguna optimización, el código en ensamblador hace una traducción casi literal del código en C. En los resultados obtenidos la función "*candidatos_actualizar_arm*" no va a variar según ninguna opción de compilación debido a que toda ella está escrita en ensamblador.

6.1.1 Resultados obtenidos

RESULTADOS OBTENIDOS -O0		
Función	Coste en tiempo	Coste en memoria
<i>candidatos_actualizar_c</i>	2,61775 ms	200 B
<i>candidatos_actualizar_arm</i>	1,28966 ms	336 B
<i>candidatos_actualizar_c_arm</i>	1,60025 ms	200 B
<i>candidatos_actualizar_arm_c</i>	2,40842 ms	156 B

6.1.2 Análisis de rendimiento

En este caso tenemos el mayor coste en tiempo de todos los niveles de compilación. Al tratarse de una traducción casi literal del código en C, no está activo el *inlining*, conservando todas las llamadas a las funciones de C del código original, como todas las llamadas a las funciones de "*celda.h*", penalizando en tiempo y en número de instrucciones ejecutadas. Así se observa como en la versión "*candidatos_actualizar_arm*", que ha sido escrita sin esas llamadas a función o el uso de *flags* de condición en instrucciones (para reducir el número de instrucciones), se obtiene un tiempo de ejecución de 1,2 ms frente a 2,6 ms de la versión en C, ósea, un *SpeedUp* del **50,73% más rápida nuestra versión en ensamblador frente a la versión en C**.

Asimismo, entre las versiones "*candidatos_actualizar_c_arm*" y "*candidatos_actualizar_arm_c*" que sólo se diferencian en que una tiene en C la parte de "*candidaros_propagar*" y la parte de "*candidatos_actualizar*" en ensamblador y viceversa, se observa como entre las dos funciones, "*candidaros_propagar*" es la que tiene mayor peso (en instrucciones y tiempo) por lo que es una función crítica que si se optimiza se ganará sustancialmente en tiempo. En este caso el *speedUp* entre las versiones es de un **33,5% más rápida la versión de "candidaros_propagar" en ensamblador**, en gran medida gracias al *inlining* que hace la versión en ensamblador que se ahorra todas las llamadas a funciones de "*celda.h*".

6.2 Resultados con la opción de compilación -O1

En este caso realiza un conjunto mínimo de optimizaciones en el código generado respecto al nivel-O0. Esta opción de compilación a diferencia de las demás (-O2 y -O3) trata de reducir el tamaño del código generado, pero no reduce apenas el tiempo de ejecución. Se trata de una opción de compilación intermedia entre -O0 y -O2.

6.2.1 Resultados obtenidos

RESULTADOS OBTENIDOS -O1		
Función	Coste en tiempo	Coste en memoria
candidatos_actualizar_c	2,25767 ms	208 B
candidatos_actualizar_arm	1,28967 ms	336 B
candidatos_actualizar_c_arm	1,53766 ms	212 B
candidatos_actualizar_arm_c	2,11192 ms	156 B

6.2.2 Análisis de rendimiento

En los resultados obtenidos se observa una pequeña reducción en tiempo con respecto a -O0, manteniendo el tamaño del código más o menos igual respecto al nivel -O0. En este caso la reducción en tiempo del código al código en C de "candidatos_actualizar_c" con respecto a la versión en ensamblador de "candidatos_actualizar_arm" ofrece un speedUp de un **42,88% más rápida la versión en ensamblador**.

6.3 Resultados con la opción de compilación -O2

Realiza todas las opciones de compilación posibles sin que supongan un gran impacto en el tamaño. Como resultado el tiempo de compilación es mayor y la velocidad de ejecución es también mayor si lo comparamos con la opción -O0 Incluye todos los flags de optimización de -O1 y algunos más.

6.3.1 Resultados obtenidos

RESULTADOS OBTENIDOS -O2		
Función	Coste en tiempo	Coste en memoria (idem -O1)
candidatos_actualizar_c	2,25517 ms	208 B
candidatos_actualizar_arm	1,28966 ms	336 B
candidatos_actualizar_c_arm	1,46775 ms	212 B
candidatos_actualizar_arm_c	2,17942 ms	156 B

6.3.2 Análisis de rendimiento

Sobre el nivel de optimización O2, no difiere mucho del O1, pero como si intenta hacer que mejore el rendimiento del código se nota una mejora en los tiempos de ejecución y en el tamaño del código en Bytes. Así el SpeedUp con esta opción de compilación, de **"candidatos_actualizar_arm"** sobre **"candidatos_actualizar_c"** es un **42,82% más rápida** que **no representa una gran mejora respecto al nivel O1 (42,88%)**.

6.4 Resultados con la opción de compilación -O3

Esta opción da la mayor velocidad posible al código en C compilado. Realiza todas las optimizaciones posibles, pero, a cambio, el tamaño del código aumentará, así como el tiempo de compilación.

6.4.1 Resultados obtenidos

RESULTADOS OBTENIDOS -O3		
Función	Coste en tiempo	Coste en memoria
candidatos_actualizar_c	0,9315 ms	428 B
candidatos_actualizar_arm	1,28950 ms	336 B
candidatos_actualizar_c_arm	1,32242 ms	160 B
candidatos_actualizar_arm_c	1,29558 ms	156 B

6.4.2 Análisis de rendimiento

Sobre el nivel de optimización O3, aquí hay un gran cambio en el rendimiento visible en los tiempos de ejecución los cuales se reducen drásticamente, superando por aproximadamente 2 ms nuestra solución, pero a cambio usa mucho más código.

En este caso la versión en C **"candidatos_actualizar_c"** tiene un **SpeedUp del 27,76% por lo que es rápida con respecto a "candidatos_actualizar_arm"**, nuestra versión en ensamblador. **Este es el único caso en que el compilador ofrece una mayor velocidad** de ejecución con respecto a la versión en ensamblador. Sin embargo, a pesar de que en este caso el compilador en tiempo mejora la versión en ensamblador, el coste en memoria se dispara, **siendo la única versión en la que la función en c "candidatos_actualizar_c" ocupa más que "candidatos_actualizar_arm"**, en concreto , un **127,38% más**.

7 CONCLUSIONES

Para terminar, nuestro trabajo ha servido para demostrar y entender que, si queremos programar y que sea óptimo en tiempo, la mejor opción es confiar en el compilador. En este caso no merece la pena el esfuerzo en tiempo (ni es económicamente rentable), por parte del programador de ponerle a programar este programa en ensamblador si tiene a su disposición un compilador de C.

Para los casos en los que el código en ensamblador resultaba más óptimo en tiempo (-O0, -O1 y -O2), la diferencia es de 1ms, lo que para este caso no merece la pena. Si pensamos en como se jugará el sudoku el tablero estará casi siempre esperando la entrada de un usuario, que decida que valor poner en la celda, por lo que ese 1ms que ganamos en actualizar los candidatos tras recibir la entrada del usuario resulta despreciable y no creemos que merezca la pena el esfuerzo de programar en ensamblador para ganar tan poco.