****

*Práctica 1*

Proyecto Hardware. Curso 2021-22

Juan Plo Andrés 795105 · Ignacio Ortega Lalmolda 610720

EINA - UNIZAR

20 de octubre de 2021

# ÍNDICE DE CONTENIDOS

[1 ÍNDICE DE CONTENIDOS 1](#_Toc85714100)

[2 RESUMEN EJECUTIVO 2](#_Toc85714101)

[2.1 Resumen 2](#_Toc85714102)

[3 INTRODUCCIÓN 3](#_Toc85714103)

[3.1 Mapa de memoria del LPC2105 4](#_Toc85714104)

[4 OBJETIVOS 5](#_Toc85714105)

[4.1 Objetivo del proyecto 5](#_Toc85714106)

[5 METODOLOGÍA 6](#_Toc85714107)

[5.1 Pasos realizados 6](#_Toc85714108)

[5.2 Esquema del proyecto 6](#_Toc85714109)

[5.2.1 Ficheros del proyecto 6](#_Toc85714110)

[5.2.2 Funciones que componen el proyecto 7](#_Toc85714111)

[5.2.3 Código desarrollado en C 8](#_Toc85714112)

[5.2.4 Código desarrollado en ARM y optimizaciones realizadas 10](#_Toc85714113)

[6 RESULTADOS 19](#_Toc85714114)

[6.1 Resultados con la opción de compilación -O0 19](#_Toc85714115)

[6.1.1 Resultados obtenidos 19](#_Toc85714116)

[6.1.2 Análisis de rendimiento 19](#_Toc85714117)

[6.2 Resultados con la opción de compilación -O1 20](#_Toc85714118)

[6.2.1 Resultados obtenidos 20](#_Toc85714119)

[6.2.2 Análisis de rendimiento 20](#_Toc85714120)

[6.3 Resultados con la opción de compilación -O2 20](#_Toc85714121)

[6.3.1 Resultados obtenidos 20](#_Toc85714122)

[6.3.2 Análisis de rendimiento 21](#_Toc85714123)

[6.4 Resultados con la opción de compilación -O3 21](#_Toc85714124)

[6.4.1 Resultados obtenidos 21](#_Toc85714125)

[6.4.2 Análisis de rendimiento 21](#_Toc85714126)

[7 CONCLUSIONES 22](#_Toc85714127)

# RESUMEN EJECUTIVO

## 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?

# 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 μVision 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.

## 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.

|  |
| --- |
|  |
| *Figura 1: Mapa de memoria* |

# OBJETIVOS

## 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.

# METODOLOGÍA

## 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.

## Esquema del proyecto

### 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 |

### 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”: c\_arm, c, arm, arm\_c*, pasándole a cada una su tablero. Devuelve 4 si todos los tableros coinciden con el tablero solución al finalizar. Devuelve -1 en caso contrario. |
| cuadricula\_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. |

### Código desarrollado en C

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

1. 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 cuadricula como primer parametro  \* y devuelve en celdas\_vacias el n�mero de celdas vacias  \*/  static int candidatos\_actualizar\_c(CELDA cuadricula[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(cuadricula[i][j])) {  elminaCandidatos(&cuadricula[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(cuadricula[i][j])) {  celdas\_vacias++;  } else {  candidatos\_propagar\_c(cuadricula, i, j);  }  }  }  //retornar el numero de celdas vacias  return celdas\_vacias;  } |

1. Es vacía

|  |
| --- |
| ES VACÍA |
| /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* devuelve 0 si y sólo sí celda\_leer\_valor(celda) = 0 \*/  \_\_inline static uint8\_t esVacia(CELDA celda)  {  return celda\_leer\_valor(celda) == 0;  } |

1. Elimina candidatos

|  |
| --- |
| ELIMINA CANDIDATOS |
| /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* modifica la celda poniendo a 0 todos sus candidatos (todos son candidatos) \*/  void elminaCandidatos(CELDA \*celdaptr) {  \*celdaptr = \*celdaptr & 0x007F;  } |

### 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 lo 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 |
| ;esVacia  ;---------------------------  ldrh r5,[r0]  and r4,r5,#0x000F  ;---------------------------  cmp r4,#0  bne noEliminar  ;elminaCandidatos si es vacia  ;---------------------------  ldrh r5,[r0]  and r5,r5,#0x007F  strh r5,[r0]  noEliminar (…) | ;esVacia  ;---------------------------  ldrh r5,[r0]  ands r4,r5,#0x000F  ;---------------------------    ;elminaCandidatos si es vacia  ;---------------------------  ldrheq r5,[r0]  andeq r5,r5,#0x007F  strheq r5,[r0] |

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 |
| mov r7,#NUM\_FILAS  ;for(j=0;j>NUM\_FILAS;j++)  for1 (…)  add r5,r5,#1  cmp r5,r7  bne for1 | mov r7,#NUM\_FILAS  ;for(j=NUM\_FILAS;j>0;j--)  for1 (…)  subs r7,r7,#1  bne for1 |

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

1. Candidatos actualizar ARM

|  |  |
| --- | --- |
| CANDIDATOS ACTUALIZAR ARM C | |
| Parámetros recibidos | |
| Registro | Descripción |
| r0 | @ini\_cuadricula |
| 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  ;---------------------------  ;elminaCandidatos si es vacia  ;---------------------------  andeq r5,r5,#0x007F  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  ;---------------------------  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  b12 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  b22 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  ;---------------------------  addeq r9,r9,#1 ;si estaba vacía sumo 1 a celdas vacías  movne r0,r6  movne r1,r7  movne r2,r8    blne candidatos\_propagar\_c ;si la celda estaba vacía, propago sus  candidatos(r0 = dirección celda, r1 = i, r8 =  j)    subs r8,r8,#1 ;una iteración del segundo for  bpl b22 ;acaba cuando j < 0 para calcular la columna 0    subs r7,r7,#1 ;una iteración del primer for  bpl b12 ;acaba cuando i < 0 para calcular la fila 0  mov r0,r9 ;se pasan las celdas vacías a r0 para  devolverlas  POP {r4-r10,pc} ;restaurar los registros usados y vuelta del pc  a sudoku\_2021.c | |

1. Candidatos propagar ARM

|  |  |
| --- | --- |
| CANDIDATOS PROPAGAR ARM | |
| Parámetros recibidos | |
| Registro | Descripción |
| r0 | @ini\_cuadricula |
| r1 | fila |
| r2 | columna |
| Código en ARM | |
| candidatos\_propagar\_arm    PUSH {r4-r12,lr} ; registros a usar    ;obtener la cuadricula[fila][columna]  add r6,r0,r1,LSL#5 ;r6 = @ini\_fila  add r5,r6,r2,LSL#1 ;r5 = cuadricula[fila][columna]    ldrh r12,[r5] ;r12 = cuadricula[fila][columna] =  mem[@ini\_fila + col\*2] (r6+r5 =  @cuadricula[fila][columna])  ;valor = celda\_leer\_valor  and r12,#0x00F ;r12 = valor  ;---------------------------  mov r7,#NUM\_FILAS ;r7 = j = 9  for1 ;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  strh r4,[r5] ;mem[r5] = r4  ;---------------------------  add r5,r5,#32 ;r5 = cuadricula[fila + 1][columna]  subs r7,r7,#1 ;r7 = i-- (condición de fin de bucle i = 0)  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] ;r7 = init\_i (fila)  ldrb r8,[r8] ;r8 = init\_j (col)  add r9,r7,#3 ;r9 = end\_i  add r10,r8,#3 ;r10 = end\_j  for3 ;for(i=init\_i; i<end\_i; i++) ;avanza fila a fila  mov r6,r8 ;se restaura en r6 el valor de la columna para  repetir for4  ;Calcula @cuadricula[ini\_i][ini\_j]  add r5,r0,r7,LSL#5  add r5,r5,r8,LSL#1  for4 ;for(j=init\_j; j<end\_j; j++) ;avanza columna a columna  ;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] = valor cuadricula  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,[r5] ;mem[r5] = r4  ;---------------------------  add r5,r5,#2 ;avanza r6 en la región  add r6,r6,#1 ;avanza en la columna  cmp r6,r10 ;se compara r6 (init\_j que va avanzando por  las columnas) con r10 end\_j  bne for4  ;---------------------------  add r7,r7,#1 ;avanzar la fila en para for3  cmp r7,r9 ;se compara r7 (init\_i que va avanzando por  las filas) con end\_i  bne for3  ;---------------------------  POP {r4-r12,pc} ;restaurar los registros usados y vuelta del  pc a sudoku\_2021.c | |

1. Candidatos actualizar ARM

|  |  |
| --- | --- |
| CANDIDATOS ACTUALIZAR ARM | |
| Parámetros recibidos | |
| Registro | Descripción |
| r0 | @ini\_cuadricula |
| Código en ARM | |
| candidatos\_actualizar\_arm\_c  PUSH {r4-r12,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 direccion de la celda necesita estar  en r0 para pasarlo como parametro a esVacia y  eliminaCandidatos    arm\_arm\_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    arm\_arm\_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 direccion 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  ;---------------------------  ;elminaCandidatos si es vacia  andeq r5,r5,#0x007F  strheq r5,[r0]  ;---------------------------  subs r8,r8,#1 ;una iteracion del segundo for  bpl arm\_arm\_b21 ;acaba cuando j < 0 para calcular la columna 0    subs r7,r7,#1 ;una iteracion del primer for  bpl arm\_arm\_b11 ;acaba cuando i < 0 para calcular la fila 0    ;SEGUNDO PAR DE BUCLES ANIDADOS  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 direccion de la celda necesita estar  en r0 para pasarlo como parametro a esVacia y  eliminaCandidatos  arm\_arm\_b12 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  arm\_arm\_b22 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 direccion 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  ;---------------------------  addeq r9,r9,#1 ;si estaba vacia sumo 1 a celdas vacias  movne r0,r6  beq noPropagar  mov r1,r7  mov r2,r8    push {r6-r9}    ;candidatos\_propagar\_arm  ;=====================================================================================    ;obtener la cuadricula[fila][columna]  add r6,r0,r1,LSL#5 ;r6 = @ini\_fila  add r5,r6,r2,LSL#1 ;r5 = cuadricula[fila][columna]    ldrh r12,[r5] ;r12 = cuadricula[fila][columna] =  mem[@ini\_fila + col\*2] (r6+r5 =  @cuadricula[fila][columna])    ;valor = celda\_leer\_valor  and r12,#0x00F ;r12 = valor  ;---------------------------    mov r7,#NUM\_FILAS ;r7 = j = 9  arm\_arm\_for1 ;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-- (concición de fin de bucle j = 0)  bne arm\_arm\_for1  ;---------------------------  mov r7,#NUM\_FILAS ;r7 = i = 9  sub r5,r5,r1,LSL#5 ;r5 = @ini\_col  arm\_arm\_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  strh r4,[r5] ;mem[r5] = r4  ;---------------------------  add r5,r5,#32 ;r5 = cuadricula[fila + 1][columna]  subs r7,r7,#1 ;r7 = i-- (concición de fin de bucle i = 0)  bne arm\_arm\_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] ;r7 = init\_i (fila)  ldrb r8,[r8] ;r8 = init\_j (col)  add r9,r7,#3 ;r9 = end\_i  add r10,r8,#3 ;r10 = end\_j  arm\_arm\_for3 ;for(i=init\_i; i<end\_i; i++) ;avanza fila a fila  mov r6,r8 ;se restaura en r6 el valor de la columna para  repetir for4  ;Calcula @cuadricula[init\_i][init\_j]  add r5,r0,r7,LSL#5  add r5,r5,r8,LSL#1    arm\_arm\_for4 ;for(j=init\_j; j<end\_j; j++) ;avanza columna a columna  ;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] = valor cuadricula  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,[r5] ;mem[r5] = r4  ;---------------------------  add r5,r5,#2 ;avanza r6 en la región  add r6,r6,#1 ;avanza en la columna  cmp r6,r10 ;se compara r6 (init\_j que va avanzando por  las columnas) con r10 end\_j  bne arm\_arm\_for4  ;---------------------------  add r7,r7,#1 ;avanzar la fila en para for3  cmp r7,r9 ;se compara r7 (init\_i que va avanzando por  las filas) con end\_i  bne arm\_arm\_for3  ;---------------------------  pop{r6-r9}  ;==========================================================================    noPropagar subs r8,r8,#1 ;una iteracion del segundo for  bpl arm\_arm\_b22 ;acaba cuando j < 0 para calcular la columna 0    subs r7,r7,#1 ;una iteracion del primer for  bpl arm\_arm\_b12 ;acaba cuando i < 0 para calcular la fila 0  mov r0,r9 ;se pasan las celdas vacías a r0 para  devolverlas  POP {r4-r12,pc} ;restaurar los registros usados y vuelta del pc  a sudoku\_2021.c | |

# 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.

## 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.

### Resultados obtenidos

|  |  |  |
| --- | --- | --- |
| RESULTADOS OBTENIDOS -O0 | | |
| Función | Coste en tiempo | Coste en memoria |
| candidatos\_actualizar\_c | 2,61775 ms | 200 B |
| candidatos\_actualizar\_arm | 1,28966 ms | 336 B |
| candidatos\_actualizar\_c\_arm | 1,60025 ms | 200 B |
| candidatos\_actualizar\_arm\_c | 2,40842 ms | 156 B |

### 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 critica 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”.*

## 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.

### 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 |

### 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.**

## 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.

### 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 |

### Análisis de rendimiento

Sobre el nivel de optimización O2, no difiere mucho del O1, pero como si intenta hacer que mejore el rendimiento del codigo se nota una mejoria en los tiempos de ejecucion y en el tamaño del codigo 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 mejoría respecto al nivel O1** (42,88%).

## 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.

### 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 |

### 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 mas 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.**

# 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.