PRÁCTICA 1

PROYECTO HARDWARE

22/10/2020

Aarón Ibáñez Espés 779088
Ángel Espinosa Gonzalo 775750

1. Resumen ejecutivo

Este documento es una memoria técnica de la primera práctica de proyecto Hardware.

Esta primera práctica se basa en optimizar el rendimiento del juego sudoku, acelerando las funciones computacionalmente más costosas. Para ello se parte de funciones implementadas en C y se traducen a ARM, intentado optimizar su ejecución.

Además de ello, se pretende comparar el rendimiento de las funciones implementadas en ARM con las funciones en C aplicando los diferentes niveles de optimización del compilador. Los valores que se van a comparar son el tiempo de ejecución, el tamaño de la función en bytes y el número de instrucciones que ejecuta cada una de las funciones.

Para todo ello se va a utilizar el entorno de desarrollo Keil µVision simulando el controlador ARM LPC2105.

A lo largo del documento se muestra la implementación de las funciones, su rendimiento y optimizaciones y la comparación de rendimiento con las funciones implementadas en C.

2. Código fuente comentado

En las siguientes páginas se incluyen capturas de las funciones implementadas durante la práctica, debidamente comentadas y con una cabecera donde se explica la tarea que realiza cada una de ellas.

2.2 candidatos_propagar_arm

```
; Funcion:
;   Función propagar_arm, adaptación de la función propagar a ARM.
; Parametros de la funcion:
;   R0: @tablero, R1: fila, R2:columna
; Descripcion:
;   La funcion candidatos_propagar_arm propaga el valor de una determinada celda en C
;   para actualizar las listas de candidatos de las celdas en su su fila, columna y región
;   Recibe como parametro la cuadricula, y la fila y columna de la celda a propagar; no devuelve nada

    PRESERVE8
    AREA DATOS, data
region DCB 0,0,0,3,3,3,6,6,6
    AREA    f_candidatos_propagar_arm, CODE, READONLY
    EXPORT candidatos_propagar_arm
candidatos_propagar_arm
    STMDB R13!,{R4-R10,R11,R12,R14}
    MOV R6,R0                                   ; R6 es @cuadricula
    MOV R7,R1                                   ; R7 es la fila
    MOV R8,R2                                   ; R8 es la columna

    LDR R2,=region                              ; R2 es la @ del inicio del vector region

    ADD R0,R6,R7,LSL #5                         ; valor = celda_leer_valor
    ADD R1,R0,R8,LSL #1                         ; En R1 hemos calculado la @ celda
    LDRH R0,[R1]                                ; En R0 guardamos el valor de la celda
    AND R1,R0,#0x0000000F                       ; valor
    MOV R9,#0x00000000                          ; R9 es la j del primer for (iterador)
INI_FOR_1
    CMP R9,#0x00000009
    BEQ FIN_FOR_1
    ADD R4,R6,R7,LSL #5                         ; @ fila
    ADD R0,R4,R9,LSL #1                         ; En R0 guardamos la @ celda
    MOV R4,#0x00000007
    LDRH R5,[R0]                                ; En R5 guardamos el valor de la celda
    SUB R3,R1,#0x00000001                       ; valor - 1
    ADD R4,R4,R3                                ; 7 + (valor - 1)
    MOV R10,#0x00000001
    ORR R5,R5,R10,LSL R4                        ; celdaptr OR 7 + (valor - 1)
    STRH R5,[R0]
    ;fin celda eliminar candidatos
    ADD R9,R9,#0x00000001
    B INI_FOR_1
FIN_FOR_1
    MOV R9,#0x00000000                          ; R9 es la i del segundo for

INI_FOR_2
    CMP R9,#0x00000009
    BEQ FIN_FOR_2
    ADD R4,R6,R9,LSL #5                     ; @ columna
    ADD R0,R4,R8,LSL #1                     ; En R0 guardamos la @ celda
    ;LDR R1,[R13]
    ;celda eliminar candidatos
    MOV R4,#0x00000007
    LDRH R5,[R0]                            ; En R5 guardamos el valor de la celda
    SUB R3,R1,#0x00000001                   ; valor - 1
    ADD R4,R4,R3                            ; 7 + (valor - 1)
    MOV R10,#0x00000001
    ORR R5,R5,R10,LSL R4                    ; celdaptr OR 7 + (valor - 1)
    STRH R5,[R0]
    ;fin celda eliminar candidatos
    ADD R9,R9,#0x00000001
    B INI_FOR_2
FIN_FOR_2

    ADD R4,R2,R7
    LDRB R3,[R4]                            ; R3 = init_region[fila]
    ADD R4,R2,R8
    LDRB R5,[R4]                            ; R5 = init_region[columna]
    ADD R4,R3,#0x00000003                   ; R4 = end_i
    ADD R9,R5,#0x00000003                   ; R9 = end_j
INI_I
    CMP R3,R4
    BEQ FUERA_I
    MOV R10,R5
INI_J
    CMP R10,R9
    BEQ FUERA_J
    ADD R2,R6,R3,LSL #5
    ADD R0,R2,R10,LSL #1                    ; En R0 guardamos cuadricula[i][j]
    ;celda eliminar candidatos
    MOV R7,#0x00000007
    LDRH R2,[R0]                            ; R2 = celdaptr
    SUB R8,R1,#0x00000001                   ; valor - 1
    ADD R7,R7,R8                            ; 7 + (valor - 1)
    MOV R8,#0x00000001
    ORR R2,R2,R8,LSL R7                     ; celdaptr OR 7 + (valor - 1)
    STRH R2,[R0]
    ;fin celda eliminar candidatos
    ADD R10,R10,#0x00000001                 ; j++
    B INI_J
FUERA_J
    ADD R3,R3,#0x00000001                   ; i++
    B INI_I
FUERA_I
    LDMIA   R13!,{R4-R10,R11,R12,R14}
    BX R14
    END
```

Imagen 1. Código comentado de la función candidatos_propagar_arm

2.1 candidatos_actualizar_arm_c

```
; Función actualizar_arm_c, función en arm que llama a la función propagar en c.
; Cabecera: candidatos_actualizar_arm_c(CELDA cuadricula[NUM_FILAS][NUM_COLUMNAS])
; Parametros: CELDA cuadricula[NUM_FILAS][NUM_COLUMNAS] es la celda a tratar
; Descripcion:
; 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 numero de celdas vacias

    PRESERVE8
    AREA    f_candidatos_actualizar_arm_c, CODE, READONLY
    EXPORT candidatos_actualizar_arm_c
    IMPORT candidatos_propagar_c
candidatos_actualizar_arm_c
    STMDB R13!,{R4-R9,R14}
    MOV R6,R0                       ; MOVEMOS A R6 CUADRICULA
    MOV R7,#0x00000000              ; R7 ES CELDAS VACIAS = 0
    MOV R8,#0x00000000              ; R8 ES i
INI_I
    CMP R8,#0x00000009              ; i < NUM_FILAS
    BEQ FUERA_I
    MOV R9,#0x00000000              ; R9 ES j
INI_J
    CMP R9,#0x00000009              ; j < NUM_FILAS
    BEQ FUERA_J
    ADD R1,R6,R8,LSL #5             ; INLINE FUNCION celda_establecer_todos_candidatos LSL #5 es para desplazar $r8 bytes
    ADD R0,R1,R9,LSL #1             ; En R1 previamente hemos calculado la fila y en R0 calculamos la celda
    LDRB R1,[R0]                    ; Cargamos en R1 el contenido de la celda
    AND R1,R1,#0x0000007F           ; Ponemos a 0 los candidatos
    STRH R1,[R0]
    ADD R9,R9,#0x00000001           ; j++
    B INI_J
FUERA_J
    ADD R8,R8,#0x00000001           ; i++
    B INI_I
FUERA_I
    MOV R8,#0x00000000              ; R8 ES i

INI_J_2
    CMP R9,#0x00000009              ; j < NUM_FILAS
    BEQ FUERA_J_2
    ADD R1,R6,R8,LSL #5             ; INLINE CELDA_LEER_VALOR
    ADD R1,R1,R9,LSL #1             ; Calculamos la celda en R1
    LDRH R0,[R1]                    ; Guardamos el contenido de la celda en R0
    AND R1,R0,#0x0000000F
    CMP R1,#0x00000000
    ADDEQ R7,R7,#0x00000001         ; CELDAS VAC?AS ++
    MOVNE R1,R8                     ; PARAMETRO I
    MOVNE R2,R9                     ; PARAMETRO J
    MOVNE R0,R6                     ; PARAMETRO CUADRICULA
    BLNE candidatos_propagar_c
    ADD R9,R9,#0x00000001           ; j++
    B INI_J_2
FUERA_J_2
    ADD R8,R8,#0x00000001           ; i++
    B INI_I_2
FUERA_I_2
    MOV R0,R7                       ; Pasamos Celdad Vacias a R0 para retornarlo
    LDMIA   R13!,{R4-R9,R14}
    BX R14

    END
```

Imagen 2. Código comentado de la función candidatos_actualizar_arm_c

2.3 candidatos_actualizar_arm_arm

```
; Funcion:
;   Función candidatos_actualizar_arm_arm, adaptación de las funciones actualizar y propagar en arm con inlining.
; Parametros:
;   CELDA cuadricula[NUM_FILAS][NUM_COLUMNAS] es la celda a tratar lo pasamos por R0
; Descripcion:
;   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 numero de celdas vacias
;   La función incorpora ambos códigos en arm tanto el de actualizar como el de propagar sin llamadas
;   ahorrando así el tiempo que supone la llamada a la función propagar

    PRESERVE8
    AREA DATOS, data
region DCB 0,0,0,3,3,3,6,6,6
    AREA    f_candidatos_actualizar_arm_arm, CODE, READONLY
    EXPORT candidatos_actualizar_arm_arm
candidatos_actualizar_arm_arm
    STMDB R13!,{R4-R9,R11,R12,R14}
    MOV R6,R0                       ; MOVEMOS A R6 CUADRICULA
    MOV R7,#0x00000000              ; R7 ES CELDAS VACIAS = 0
    MOV R8,#0x00000000              ; R8 ES i
INI_I
    CMP R8,#0x00000009              ; i < NUM_FILAS
    BEQ FUERA_I
    MOV R9,#0x00000000              ; R9 ES j
INI_J
    CMP R9,#0x00000009              ; j < NUM_FILAS
    BEQ FUERA_J
    ADD R1,R6,R8,LSL #5             ; INLINE FUNCION celda_establecer_todos_candidatos LSL #5 es para desplazar $r8 bytes
    ADD R0,R1,R9,LSL #1             ; En R0 guardamos la direccion de la celda
    LDRB R1,[R0]                    ; R1 = contenido de la celda
    AND R1,R1,#0x0000007F           ; Ponemos los candidatos a 0
    STRH R1,[R0]
    ADD R9,R9,#0x00000001           ; j++
    B INI_J
FUERA_J
    ADD R8,R8,#0x00000001           ; i++
    B INI_I
FUERA_I
    MOV R8,#0x00000000              ; R8 ES i

INI_I_2
    CMP R8,#0x00000009               ; i < NUM_FILAS
    BEQ FUERA_I_2
    MOV R9,#0x00000000               ; R9 ES j
INI_J_2
    CMP R9,#0x00000009               ; i < NUM_FILAS
    BEQ FUERA_J_2
    ADD R1,R6,R8,LSL #5              ; INLINE CELDA_LEER_VALOR
    ADD R1,R1,R9,LSL #1              ; En R1 guardamos la direccion de la celda
    LDRH R0,[R1]                     ; R1 = contenido de la celda
    AND R1,R0,#0x0000000F            ; Cogemos solo el valor
    CMP R1,#0x00000000               ; Valor != 0
    ADDEQ R7,R7,#0x00000001          ; CELDAS VAC?AS ++
    BEQ FIN_PROPAGAR
    MOVNE R1,R8                      ; PARAMETRO I
    MOVNE R2,R9                      ; PARAMETRO J
    MOVNE R0,R6                      ; PARAMETRO CUADRICULA
    ;-------------------------------------------------------------------------------------BLNE candidatos_propagar_arm
    STMDB R13!,{R4-R10}
    MOV R6,R0                                ;@cuadricula
    MOV R7,R1                                ; fila
    MOV R8,R2                                ; columna

    LDR R2,=region                           ; @ del inicio del vector region

    ADD R0,R6,R7,LSL #5                      ; valor = celda_leer_valor
    ADD R1,R0,R8,LSL #1                      ; R1 = @ celda
    LDRH R0,[R1]                             ; celda
    AND R1,R0,#0x0000000F                    ; R1 = valor
    MOV R9,#0x00000000                       ; R9 es la j del primer for (iterador)
INI_FOR_1_AUX
    CMP R9,#0x00000009
    BEQ FIN_FOR_1_AUX
    ADD R4,R6,R7,LSL #5                      ; @ fila
    ADD R0,R4,R9,LSL #1                      ; R0 es la @ de la celda
    ;LDR R1,[R13]
    ;celda eliminar candidatos
    MOV R4,#0x00000007
    LDRH R5,[R0]                             ; R5 = celdaptr
    SUB R3,R1,#0x00000001                    ; valor - 1
    ADD R4,R4,R3                             ; 7 + (valor - 1)
    MOV R10,#0x00000001
    ORR R5,R5,R10,LSL R4                     ; celdaptr OR 7 + (valor - 1)
    STRH R5,[R0]
    ;fin celda eliminar candidatos
    ADD R9,R9,#0x00000001                    ; i++
    B INI_FOR_1_AUX
FIN_FOR_1_AUX
    MOV R9,#0x00000000                       ; R9 es la i del segundo for
```

```
INI_FOR_2_AUX
    CMP R9,#0x00000009
    BEQ FIN_FOR_2_AUX
    ADD R4,R6,R9,LSL #5                     ; @ columna
    ADD R0,R4,R8,LSL #1                     ; R0 = @ celda
                                            ; celda eliminar candidatos
    MOV R4,#0x00000007
    LDRH R5,[R0]                            ; R5 = celdaptr
    SUB R3,R1,#0x00000001                   ; valor - 1
    ADD R4,R4,R3                            ; 7 + (valor - 1)
    MOV R10,#0x00000001
    ORR R5,R5,R10,LSL R4                    ; celdaptr OR 7 + (valor - 1)
    STRH R5,[R0]
    ;fin celda eliminar candidatos
    ADD R9,R9,#0x00000001                   ; i++
    B INI_FOR_2_AUX
FIN_FOR_2_AUX

    ADD R4,R2,R7
    LDRB R3,[R4]                            ; R3 = init_region[fila]
    ADD R4,R2,R8
    LDRB R5,[R4]                            ; R5 = init_region[columna]
    ADD R4,R3,#0x00000003                   ; R4 = end_i
    ADD R9,R5,#0x00000003                   ; R9 = end_j
INI_I_AUX
    CMP R3,R4                               ; i > end_i
    BEQ FUERA_I_AUX
    MOV R10,R5                              ; j = init_region[columna]
INI_J_AUX
    CMP R10,R9                              ; j < end_j
    BEQ FUERA_J_AUX
    ADD R2,R6,R3,LSL #5
    ADD R0,R2,R10,LSL #1                    ; R0 = cuadricula[i][j]
                                            ; celda eliminar candidatos
    MOV R7,#0x00000007
    LDRH R2,[R0]                            ; R2 = contenido de la celda
    SUB R8,R1,#0x00000001                   ; valor - 1
    ADD R7,R7,R8                            ; 7 + (valor - 1)
    MOV R8,#0x00000001
    ORR R2,R2,R8,LSL R7                     ; celdaptr OR 7 + (valor - 1)
    STRH R2,[R0]
                                            ; fin celda eliminar candidatos
    ADD R10,R10,#0x00000001                 ; j++
    B INI_J_AUX
FUERA_J_AUX
    ADD R3,R3,#0x00000001                   ; i++
    B INI_I_AUX
FUERA_I_AUX
    LDMIA   R13!,{R4-R10}

    ;-------------------------------------------------------------------------------------FIN candidatos_propagar
FIN_PROPAGAR
    ADD R9,R9,#0x00000001                   ; j++
    B INI_J_2
FUERA_J_2
    ADD R8,R8,#0x00000001                   ; i++
    B INI_I_2
FUERA_I_2
    MOV R0,R7                               ; Pasamos Celdad Vacias a R0 para retornarlo
    LDMIA   R13!,{R4-R9,R11,R12,R14}
    BX R14

    END
```

Imagen 3. Código comentado de la función candidatos_actualizar_arm_arm

3. Mapa de memoria

En las siguientes imagenes se muestran el mapa de memoria del programa basado en las direcciones de memoria generadas al compilar el programa con el flag -O0 y los bloques de activación de las funciones candidatos_actualizar_arm_c, candidatos_propagar_arm y candidatos_actualizar_arm_arm programadas durante la practica.

Cabe destacar que si se compila con otro flag diferente al utilizado las direcciones de memoria pueden variar, sin embargo, la estructura del código en memoria será igual.

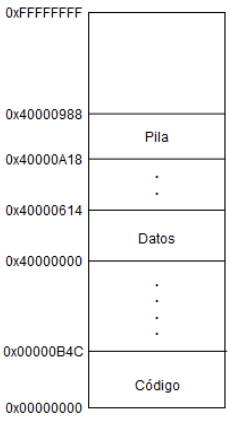

Imagen 3. Mapa de memoria del programa

En los siguientes esquemas se representan los marcos de pila que se utilizan para las distintas funciones implementadas en ARM.

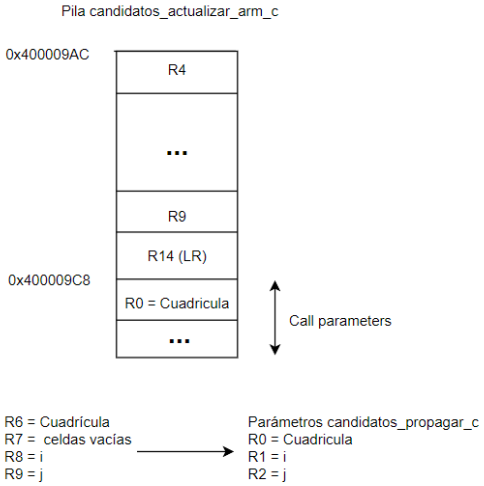

Imagen 4. Marco de pila candidatos_propagar

Como mencionamos posteriormente solo se apila r14 (Link Register) y los registros r4-r10 que van a ser utilizados por la función en cuestión. En la imagen superior se definen las variables más relevantes asignadas a cada registro y en el caso de la función actualizar la

distribución de los registros que utilizamos para asignar los parámetros de llamada de la función candidatos_propagar.

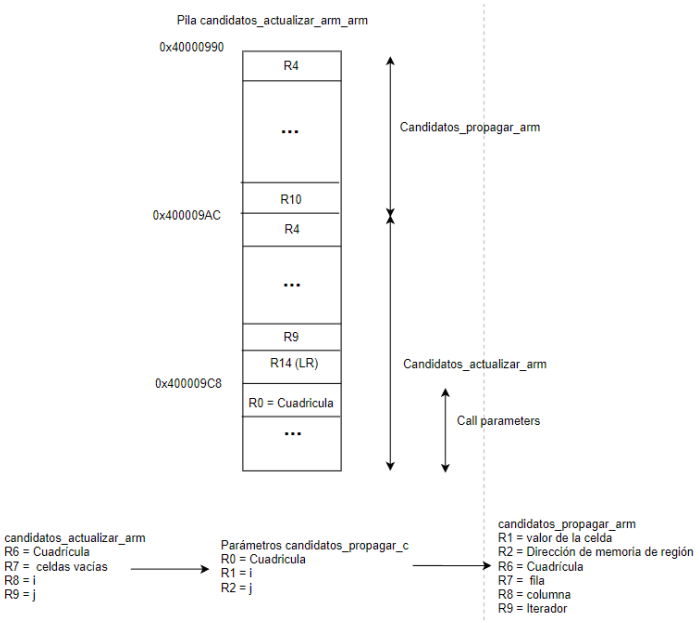

Imagen 5. Marco de pila candidatos_actualizar

El segundo marco de pila presentado es el de la función que implementa candidatos_actualizar y candidatos_propagar en arm. Se llama a candidatos_actualizar con la cuadrícula como parámetro y se apilan los registros que va a utilizar. Posteriormente se alteran r0, r1 y r2 para guardar los parámetros de propagar y se pushean de nuevo los registros r4-r10 para utilizarlos en la función propagar.
El código de propagar se ejecuta sin llamadas, incrustando el código en la función candidatos_actualizar, de esta forma se evitan los costes que supone la llamada a la función. La variable celdas_vacias que almacena el número inicial de celdas vacías que tiene el sudoku y que debe retornar la función candidatos_actualizar se devuelve en r0 a la función “padre” que llama a candidatos_actualizar.

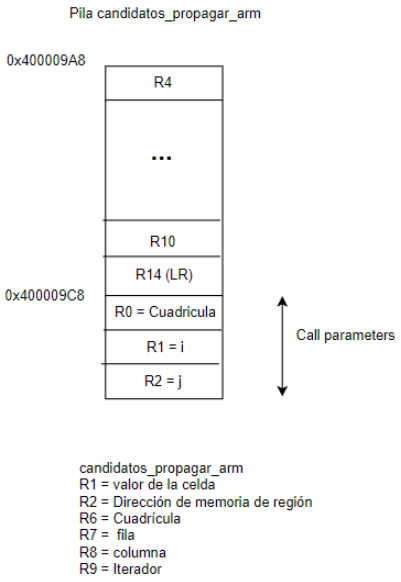

Imagen 6. Marco de pila candidatos_propagar

Finalmente se muestra la pila de la función candidatos_propagar en arm. Como ya se ha mencionado anteriormente se utiliza r0-r2 para los parámetros con los que se llama a dicha función. En la parte inferior de la imagen se muestra una leyenda con las variables que son almacenadas en los registros a lo largo de la ejecución de la función.

4. Descripción de las optimizaciones

candidatos_actualizar_arm_c:
La función candidatos_actualizar_arm_c consiste en una traducción del codigo de candidatos_actualizar_c a ensamblador, llamando a candidatos_propagar_c.

En esta función los registros R0-R3 se usan para pasar los parámetros y los registros R4-R9 se apilan junto con el LR, para salvar el contexto de la rutina anterior. En este caso se apilan solamente los registros R4-R9 debido a que no son necesarios más registros en la función. Además, no se apilan R11 ni R12 ya que no se accede a memoria para cargar o guardar variables locales. El resultado de la función se devuelve por el registro R0.

Para la llamada a la función candidatos_actualizar_c se almacenan en los registros R0-R3 los parámetros para su correcta ejecución.

Finalmente, las optimizaciones aplicadas sobre el código han sido el uso de instrucciones predicadas en estructuras condicionales para evitar saltos innecesarios en el codigo, ademas de ello, también se ha evitado el uso de operaciones de multiplicación sustituyendolas por desplazamiento a la izquierda para multiplicar por potencias de 2.
Se ha intentado proponer un diseño de los bucles que realice el menor número de instrucciones mediante un planteamiento previo y las optimizaciones mencionadas.

candidatos_propagar_arm:
La función candidatos_propagar_arm consiste en una traducción del código de candidatos_propagar_c a ensamblador.

Para recibir los parámetros se han utilizado los registros R0-R3 y se han apilado los registros R4-R10 para que sean utilizados por la función. Además, no se apilan R11 ni R12 ya que no se accede a memoria para cargar o guardar variables locales, la situación es la misma que para la función candidatos_actualizar.

Con respecto a las optimizaciones aplicadas sobre el código, al igual que en el apartado anterior han sido el uso de instrucciones predicadas en estructuras condicionales para evitar saltos innecesarios en el codigo, ademas de ello, también se ha evitado el uso de operaciones de multiplicación sustituyendolas por desplazamiento a la izquierda para multiplicar por potencias de 2.

candidatos_actualizar_propagar_arm:
La función camdidatos_actualizar_propagar_arm incorpora las dos funciones anteriores incrustando el código de candidatos_propagar en arm en el de candidatos_actualizar en arm. Las optimizaciones son las mismas que las aplicadas en las funciones por separado pero evitando los costes que supone la llamada a la función candidatos_propagar.

5. Comparación entre funciones ARM y C

En este apartado se comparan las funciones en C con las implementadas en ARM. Dicha comparación se hace entre *propagar_C(), actualizar_C(), propagar_arm()* y *actualizar_arm()* y los aspectos comparados son el tiempo de ejecución, el tamaño en bytes y el número de instrucciones ejecutadas en cada una de las funciones y para cada uno de los diferentes niveles de optimización.

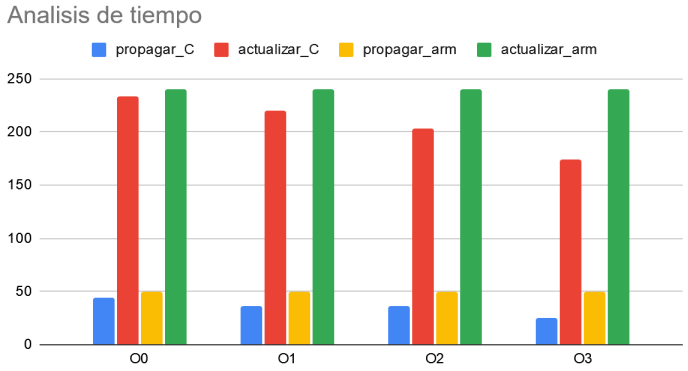

Gráfico 1. Comparación de tiempo

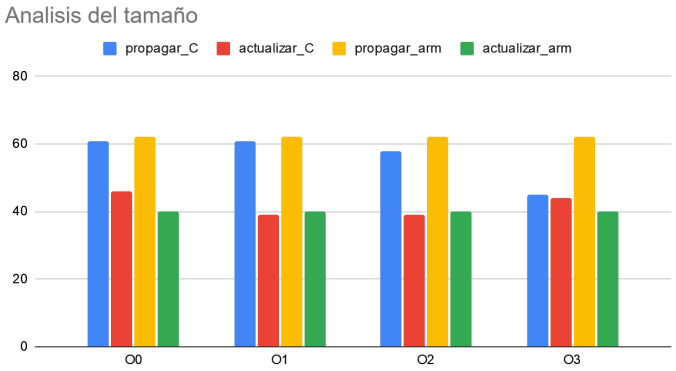

Gráfico 2. Comparación de tamaño

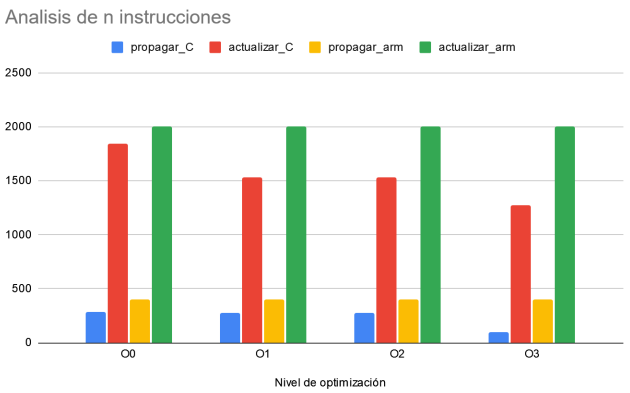

Gráfico 3. Comparación de instrucciones ejecutadas

En el Anexo 1, se proporcionan los datos numéricos a partir de los cuales se han obtenido las gráficas y en el apartado 6 se comenta el análisis de rendimiento apreciable en las mismas.

6. Análisis de rendimiento

Con respecto al rendimiento de las funciones, para las funciones en c se puede observar como va mejorando tanto tiempo de ejecución como el tamaño de la función como el número de instrucciones ejecutadas conforme se aumenta el nivel de optimización. Para -O1 el compilador trata de hacer más “inlining”, en -O2 se comienza a ver que desaparecen las llamadas a las funciones en c, tanto prólogo como el epílogo, y la variable correcto, dado que finalmente la variable no se usa, deja de estar en el “scope” por lo que solo se puede ver su valor en el registro en el que ha sido almacenada puesto que no aparece en el call stack del modo debug. Finalmente, con -O3 el compilador intenta evitar saltos para reducir costes. Se observa un cambio bastante significativo en las tres medidas de rendimiento tomadas para la ejecución con el nivel de optimización -O3 con respecto a -O2.
Cabe destacar que para la función actualizar en c aumenta levemente su tamaño en Bytes en -O3 frente a -O2 pero ofrece resultados notablemente mejores frente al resto de niveles de optimización en cuanto a tiempo e instrucciones ejecutadas.

En cuanto a las funciones implementadas en ARM no son optimizadas por el compilador, por lo que las medidas serán las mismas para los cuatro niveles de optimización. Esto se debe a que no se hacen llamadas a las funciones de celda en el código ensamblador, sino que, se ha implementado el código de las mismas dentro de las funciones actualizar y propagar. Esto provoca también que los resultados sean un poco peores comparados con los de la funciones en c para niveles de optimización bajos ya que habría que sumarle a los costes de las funciones en c los de las funciones de celda.h.

De esta forma para niveles bajos de optimización se observan mejores resultados para la función Actualizar_propagar sobretodo pero también para el resto de funciones en ARM puesto que evitan llamadas innecesarias y el código está implementado de manera efectiva. Con el avance de los niveles de optimización observamos que el compilador ejecuta de forma mucho más eficiente las funciones.

En las gráficas mostradas arriba no se aprecia que Actualizar_propagar sea la menos costosa al principio puesto que al incluir el código de propagar este se ejecuta 30 veces durante la ejecución por lo que el coste será la suma de esas 30 ejecuciones de propagar más la correspondiente de actualizar.

En las siguientes gráficas se muestra una comparación del tiempo de ejecución, tamaño en bytes y número de instrucciones de las funciones implementadas en c, ya que son las que el compilador optimiza en mayor medida.

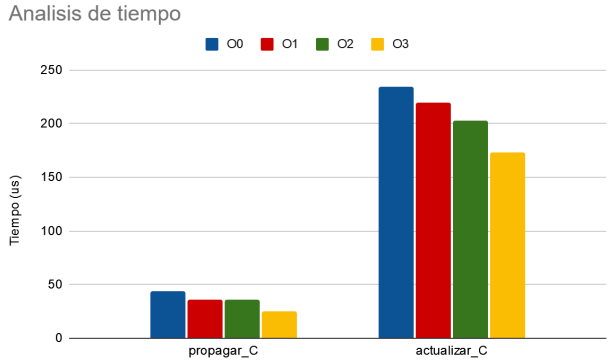

Gráfico 4. Análisis de tiempo

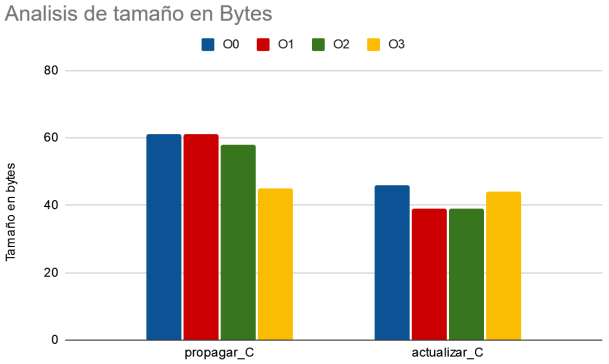

Gráfico 5. Análisis de espacio

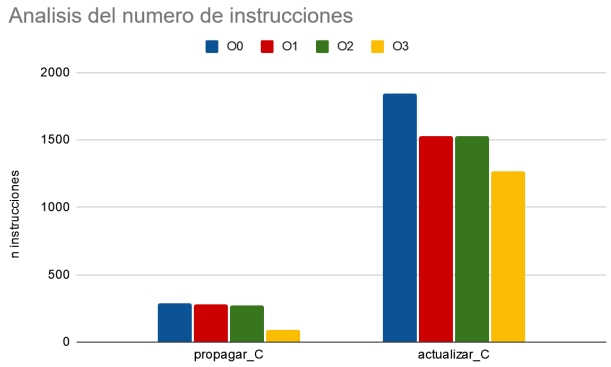

Gráfico 6. Análisis de instrucciones

7. Problemas y sus soluciones encontrados

Se han encontrado diferentes problemas a la hora de la realización de la práctica. Inicialmente con el modo debug de keil aparecieron algunos problemas para encontrar la disposición adecuada de los datos. Aparecieron también algunos problemas con las diferentes llamadas de arm a código en c y viceversa que debían desarrollarse pero fueron solucionados consultando manuales de keil y ARM.

A la hora de la toma de medidas para evitar el inlining que hacen los niveles de optimización sobre el código se observaron también algunas complicaciones que fueron solventadas eliminando el inline de estas funciones.

El proceso de debug sobre las funciones implementadas en ARM que mostraban datos incorrectos fue muy costoso, debido a problemas como el acceso a direcciones de memoria no permitidas, este error se dio debido a que se intentaba acceder a direcciones de memoria que no estaban alineadas o que se utilizaban instrucciones de lectura como LDR cuando solamente se quería leer un byte, por lo que el valor leído era incorrecto, esto se solucionó corrigiendo las instrucciones para que se adecuaran a las necesidades requeridas.

Para garantizar la consistencia de las nuevas funciones implementadas se comprobó poco a poco que las implementaciones en ARM ejecutaban código semejante al de las iniciales en c. Finalmente destacar que también se ha garantizado que las funciones se ejecutan correctamente para todos los niveles de optimización sin alterar los resultados.

8. Conclusiones

Con respecto a las conclusiones observamos que un planteamiento del código por parte del programador es eficaz para niveles bajos de optimización pero que conforme vamos aumentando estos niveles el compilador hace bien su trabajo y mejora la eficiencia en la ejecución. Aun así el compilador no es perfecto y hay aspectos que no sabe optimizar y requiere ayuda de directivas dispuestas por el programador para garantizar una ejecución óptima. Es decir, el compilador es una herramienta bastante potente con respecto a la optimización pero siempre va a haber cosas “a las que no llegue” y en las que el programador marque la diferencia.

9. Anexo 1

Tº execucion				
Nivel de optimización	O0	O1	O2	O3
propagar_C	44,1	36,3	35,9	25
actualizar_C	234	220	203	174
propagar_arm	50	50	50	50
actualizar_arm	240	240	240	240
propagar_actualizar_arm	1728	1728	1728	1728
Tamaño en Bytes				
Nivel de optimización	O0	O1	O2	O3
propagar_C	61	61	58	45
actualizar_C	46	39	39	44
propagar_arm	62	62	62	62
actualizar_arm	40	40	40	40
propagar_actualizar_arm	102	102	102	102
Nº de instrucciones ejecutadas				
Nivel de optimización	O0	O1	O2	O3
propagar_C	289	278	275	94
actualizar_C	1847	1532	1532	1271
propagar_arm	397	397	397	397
actualizar_arm	2001	2001	2001	2001
propagar_actualizar_arm	13728	13728	13728	13728

Tabla 1. Datos de comparaciones