Informe de Primera Práctica de Arquitectura de Computadores

Máximo García Aroca

October 9, 2023

En este informe, se abordarán varias cuestiones relacionadas con la ejecución de un programa utilizando el sistema de construcción `make`, así como aspectos de programación en C. A continuación, se presentan las respuestas a las preguntas planteadas.

1 Pregunta 1: Prueba de make y make clean

Prueba a ejecutar `make` y comprueba qué archivos se generan. Luego, ejecuta `make clean`. ¿Qué ha ocurrido? ¿Qué hace este comando?

Respuesta: Make es un programa que crea relaciones entre archivos y programas para construir archivos de manera automática.

En este caso, nuestro archivo Makefile tiene varias reglas. Cuando se ejecuta `make`, se registra que está usando la regla `default`, que llama a la regla `all`, la cual ejecuta primero una limpieza de los archivos .o (archivo de objeto general) de código binario para ensamblarse con otros .o y dar como resultado el ejecutable final (estos archivos por separado no son ejecutables, solo tienen información básica de cada .c).

`make clean` ejecutará el comando `rm`, borrando los .o (clase objeto), el ejecutable creado con `make` (benchmark), y los archivos en ensamblador .s, dejando únicamente los archivos originales con los que se compila el ejecutable, que son: `benchmark.c`, `dgemm.c`, y el propio Makefile.

`make` (`default`) ejecutará la regla `clean` y a posteriori creará el ejecutable primero creando los archivos de clase objeto general `benchmark.o` y `dgemm.o`, para finalmente crear el ejecutable `benchmark` con estos .o. Todo esto lo hará con el compilador `gcc`.

2 Pregunta 2: Puntero B en benchmark.c

El fichero `benchmark.c` utiliza la llamada `malloc` para reservar memoria para `buf`, y luego asigna los punteros de las matrices. ¿Por qué `B` apunta a `A+n*n` y no a `A + n*n*sizeof(float)`?

Respuesta: El uso del `sizeof(float)` se hace en el `malloc`, y este sirve para indicar con exactitud cuántos bytes se van a reservar dinámicamente para conseguir un bloque para escribir y leer sin fallos en memoria por `Segmentation fault`. Cuando en el `malloc` se indica `malloc(4 * n * n * sizeof(float))`, lo que se está reservando son 4 matrices de filas y columnas de tamaño `n` (por defecto es 256), y almacenan tipos `float` que ocupan 4 bytes, por lo que el `malloc` está reservando (por defecto) 1048576 bytes.

Ahora, en el "set pointers" se está indicando los rangos de memoria que se le asignan a cada matriz, que son tipos `float`. Al asignar `float *B = A + n*n`, se le está asignando a `B` una posición de memoria que está a `n*n` floats (4 bytes) de distancia a la posición de `A`, que es la del buffer, la primera posición que reserva el `malloc` (lo que podría entenderse como el byte 0 de nuestro bloque). Como por defecto son 256 columnas por 256 filas cada matriz, el puntero `B` apunta a la posición de memoria que le corresponde en el rango devuelto por `malloc` como bloque escribible 256*256 posiciones (direcciones) después de `A`, y en cada una de esas direcciones se almacena un tipo `float` (4 bytes).

3 Pregunta 3: Ejecución del programa

Compila el programa con `make` y ejecútalo después. ¿Qué ocurre si no le pasas argumentos? Prueba a pasarle varios valores (por ejemplo, 64, 128, 256, 1024, etc.) y anota qué salida te da el programa. Verás que para cada valor se devuelve una métrica de rendimiento. Ten en cuenta los siguientes aspectos metodológicos:

Respuesta:

Cuando no se le pasan argumentos, el programa utiliza valores por defecto de 256 filas por 256 columnas y su tiempo de ejecución es de aproximadamente 0.09 segundos.

```
int main (int argc, char **argv)
{
  double seconds = 0;
  int n = 256;

  if ( argc > 1 ) n = atoi(argv[1]);

  printf ("Description:\t%s on matrices of size %d x %d\n\n", dgemm_desc, n, n);

  /* allocate memory */
  float* buf = NULL;
  buf = (float*) malloc (4 * n * n * sizeof(float));
  if (buf == NULL) die ("failed to allocate matrices");
```

Para realizar una medición más precisa, se debe ejecutar el programa varias veces y calcular la mediana y la desviación estándar de las métricas de rendimiento, como el tiempo de ejecución. Esto ayuda a filtrar los efectos del sistema operativo y obtener resultados más confiables. También se debe observar la tendencia en las métricas de rendimiento al variar los parámetros de entrada del programa, como el tamaño de las matrices.

Conforme se van cambiando el parámetro de entrada, se modifica el tiempo que tarde en ejecutarse el código. Por ejemplo con `n=64` se crea una matriz con menos filas y menos columnas y se traduce en

un tiempo menor aún a 0.09 segundos.

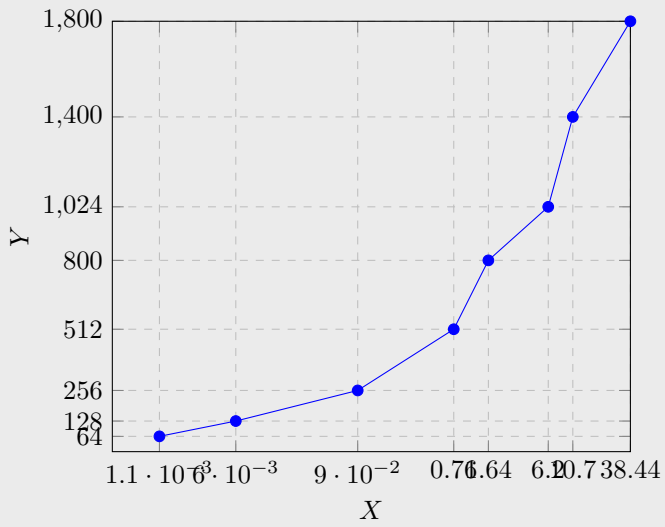
```
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 64
Description:    A naive C version on matrices of size 64 x 64

Time 1.108386e-03 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 64
Description:    A naive C version on matrices of size 64 x 64

Time 1.126732e-03 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 64
Description:    A naive C version on matrices of size 64 x 64

Time 1.127944e-03 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 64
Description:    A naive C version on matrices of size 64 x 64

Time 1.108660e-03 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 64
Description:    A naive C version on matrices of size 64 x 64

Time 1.109845e-03 s
arqcom016@aloe:~/Descargas/MultMat/Naive$
```

Lo que resulta en una media de 0.0011 segundos.

Sin embargo cuando se amplía el valor `n = 1024` por ejemplo da por resultado estos valores de temporización:

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 1024
Description:    A naive C version on matrices of size 1024 x 1024

Time 6.239031e+00 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 1024
Description:    A naive C version on matrices of size 1024 x 1024

Time 6.245994e+00 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 1024
Description:    A naive C version on matrices of size 1024 x 1024

Time 6.230809e+00 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 1024
Description:    A naive C version on matrices of size 1024 x 1024

Time 6.239532e+00 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 1024
Description:    A naive C version on matrices of size 1024 x 1024

Time 6.235065e+00 s
arqcom016@aloe:~/Descargas/MultMat/Naive$
```

Y conforme más se aumenta el parámetro mayor es la diferencia con el anterior, resultando en un programa cuya progresión temporal es exponencial. Cuanto más aumentemos las filas y las columnas más se ensanchará al diferencia de tiempo con el anterior exponencialmente.



En la representación gráfica de la temporización del código para valores cada vez más grandes se aprecia que la el área entre cada uno de los puntos aumenta exponencialmente.

4 Pregunta 4: Análisis del archivo 'Makefile'

En el fichero `Makefile` hay una serie de variables que permiten controlar las opciones de compilación, como `CC, OPT, CFLAGS`, etc. Explica brevemente qué definen y para qué sirven. También hay reglas de compilación, como `all, ensambla o clean.` Explica qué hacen estos comandos. Ejecuta `make ensambla` y comprueba que ocurre. ¿Se han generado algunos ficheros nuevos? ¿qué contienen?

Respuesta:

La variable `CC` almacena el nombre del compilador que se utilizará en las reglas, como "benchmark". El compilador que se utiliza es `gcc`.

La variable `OPT` almacena `-O0`. La bandera `-O` en `gcc` indica las opciones de control de optimización. Al utilizar las banderas de optimización, el compilador intentará mejorar el tamaño del código y cómo funciona, a costa de aumentar el tiempo de compilación y el uso del depurador. `-O0` reduce el tiempo de compilación y permite que el depurador funcione correctamente, tal como se espera de forma predeterminada. Es una bandera que está activada por defecto, por lo que no es necesario aplicarla explícitamente.

La variable `CFLAGS` almacena las banderas:

- `-Wall`: Activa un conjunto de banderas que permiten advertencias adicionales durante la compilación.
- `-std=gnu99`: Determina el estándar de lenguaje. `gnu99`, por ejemplo, permite los comentarios que siguen el estilo de C++ "//".
- `$(OPT)`: Hace referencia a la bandera `-O0` explicada anteriormente.

La variable `LDFLAGS` almacena únicamente la bandera `-Wall`.

La variable `LDLIBS` almacenaría los archivos `.h` (bibliotecas) que contienen funciones y estructuras utilizadas, pero dado que no hay archivos `.h`, esta variable está vacía.

Además, la variable `targets` almacena el nombre de nuestro ejecutable, `objects` almacena los nombres de los objetos que se crearán y utilizarán en la compilación de nuestro ejecutable, y `ass_obj` almacena el nombre de los archivos en ensamblador que se crean con la regla `ensambla`.

A continuación, se presentan las reglas disponibles en el archivo Makefile:

- `default`: Esta regla ejecutará la regla `all`. Se activa si solo se ejecuta `make`.
- `all`: Esta regla ejecuta una limpieza (`clean`) de los objetos (`objects`) y llama a la subregla `benchmark`. La subregla `benchmark` crea los archivos `.o benchmark.o` y `dgemm.o`, para luego compilarlos y generar el ejecutable `benchmark`.
- `%.o`: Esta regla se activa cuando se requieren archivos de clase objeto general. Toma como prerrequisito los archivos `.c` y utiliza el compilador para crear los archivos de clase objeto correspondientes.

- `ensambla`: Esta regla limpia todos los archivos de ensamblador (`.s`) que puedan existir antes de su ejecución. Luego, recrea los archivos de ensamblador en caso de que haya habido cambios en el código.
- `clean`: Esta regla utiliza el comando `rm` para eliminar el ejecutable, los archivos de objeto (`.o`) y los archivos de ensamblador (`.s`).

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ make
rm -f benchmark benchmark.o dgemm.o benchmark.s dgemm.s
gcc  -c -Wall -std=gnu99 -O0 benchmark.c
gcc  -c -Wall -std=gnu99 -O0 dgemm.c
gcc  -o benchmark benchmark.o dgemm.o
arqcom016@aloe:~/Descargas/MultMat/Naive$ make clean
rm -f benchmark benchmark.o dgemm.o benchmark.s dgemm.s
arqcom016@aloe:~/Descargas/MultMat/Naive$ make ensambla
rm -f benchmark benchmark.o dgemm.o benchmark.s dgemm.s
gcc  -S -Wall -std=gnu99 -O0 benchmark.c
gcc  -S -Wall -std=gnu99 -O0 dgemm.c
gcc  -c -Wall -std=gnu99 -O0 benchmark.c
gcc  -c -Wall -std=gnu99 -O0 dgemm.c
gcc  -o benchmark benchmark.o dgemm.o
arqcom016@aloe:~/Descargas/MultMat/Naive$
```

5 Pregunta 5: Código en ensamblador del `degemm.s`

Examina el fichero `dgemm.s` e intenta determinar qué es lo que hacen las instrucciones que aparecen. Utiliza la herramienta Compiler Explorer para ilustrar cómo se transforma cada sentencia de alto nivel del programa original en su bloque de instrucciones ensamblador, y explica brevemente qué va ocurriendo.

Respuesta:

El código de `dgemm.c` se encarga de computar o calcular, para cada fila y columna de B, el valor correspondiente en la matriz C. Este valor se obtiene multiplicando las posiciones correspondientes de las dos matrices y sumándolo al valor actualmente almacenado en C.

El comienzo del archivo `.s` contiene el primer bucle que corresponde a las filas de la matriz A. Salta a la etiqueta `L2` para la fila en la que estamos y el límite `n`, y luego vuelve a saltar, esta vez a `L7`.

Cuando llega a `L7`, el concepto es el mismo pero con las columnas de la matriz B, saltando a `L3` y comparando finalmente si la variable que en el programa C llamamos "j" ha excedido el límite `n`. Si la comparación es válida, salta a `L6`. Como utiliza una instrucción de salto con enlace (Jump and Link), volverá y sumará 1 para continuar con la iteración.

De `L6`, saltará a `L4`, donde realizará exactamente el mismo procedimiento que anteriormente para terminar de cargar todos los datos necesarios antes de saltar a `L5`. En `L5` es donde se realiza el cálculo de la multiplicación de las matrices y luego vuelve a saltar a `L4`, donde se almacenan los resultados en la matriz C.

```c
void square_dgemm (int n, float* A, float* B, float* C)
{
  /* For each row i of A */
  for (int i = 0; i < n; ++i)
    /* For each column j of B */
    for (int j = 0; j < n; ++j)
    {
      /* Compute C(i,j) */
      float cij = C[i*n+j]; /* cij = C[i][j] */
      for( int k = 0; k < n; k++ )
                cij += A[k+i*n] * B[j+k*n]; /* cij += A[i][k]*B[k][j] */
      C[i*n+j] = cij;   /*C[i][j] = cij*/
    }
}
```

x86-64 gcc 13.2 (Editor #1)

x86-64 gcc 13.2 | Compiler options...

A | Output... | Filter... | Libraries | Overrides | + Add new... | Add tool...

```
        mov     eax, DWORD PTR [rbp-8]
        add     eax, edx
        cdqe
        lea     rdx, [0+rax*4]
        mov     rax, QWORD PTR [rbp-40]
        add     rax, rdx
        movss   xmm0, DWORD PTR [rax]
        mulss   xmm0, xmm1
        movss   xmm1, DWORD PTR [rbp-12]
        addss   xmm0, xmm1
        movss   DWORD PTR [rbp-12], xmm0
        add     DWORD PTR [rbp-16], 1
.L4:
        mov     eax, DWORD PTR [rbp-16]
        cmp     eax, DWORD PTR [rbp-20]
        jl      .L5
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, DWORD PTR [rbp-20]
        mov     edx, eax
        mov     eax, DWORD PTR [rbp-8]
        add     eax, edx
        cdqe
        lea     rdx, [0+rax*4]
        mov     rax, QWORD PTR [rbp-48]
        add     rax, rdx
        movss   xmm0, DWORD PTR [rbp-12]
        movss   DWORD PTR [rax], xmm0
        add     DWORD PTR [rbp-8], 1
.L3:
        mov     eax, DWORD PTR [rbp-8]
        cmp     eax, DWORD PTR [rbp-20]
        jl      .L6
        add     DWORD PTR [rbp-4], 1
.L2:
        mov     eax, DWORD PTR [rbp-4]
        cmp     eax, DWORD PTR [rbp-20]
        jl      .L7
        nop
        nop
        pop     rbp
        ret
```

Output (/0) x86-64 gcc 13.2 - 497ms (6356B) ~324 lines filtered Compiler License

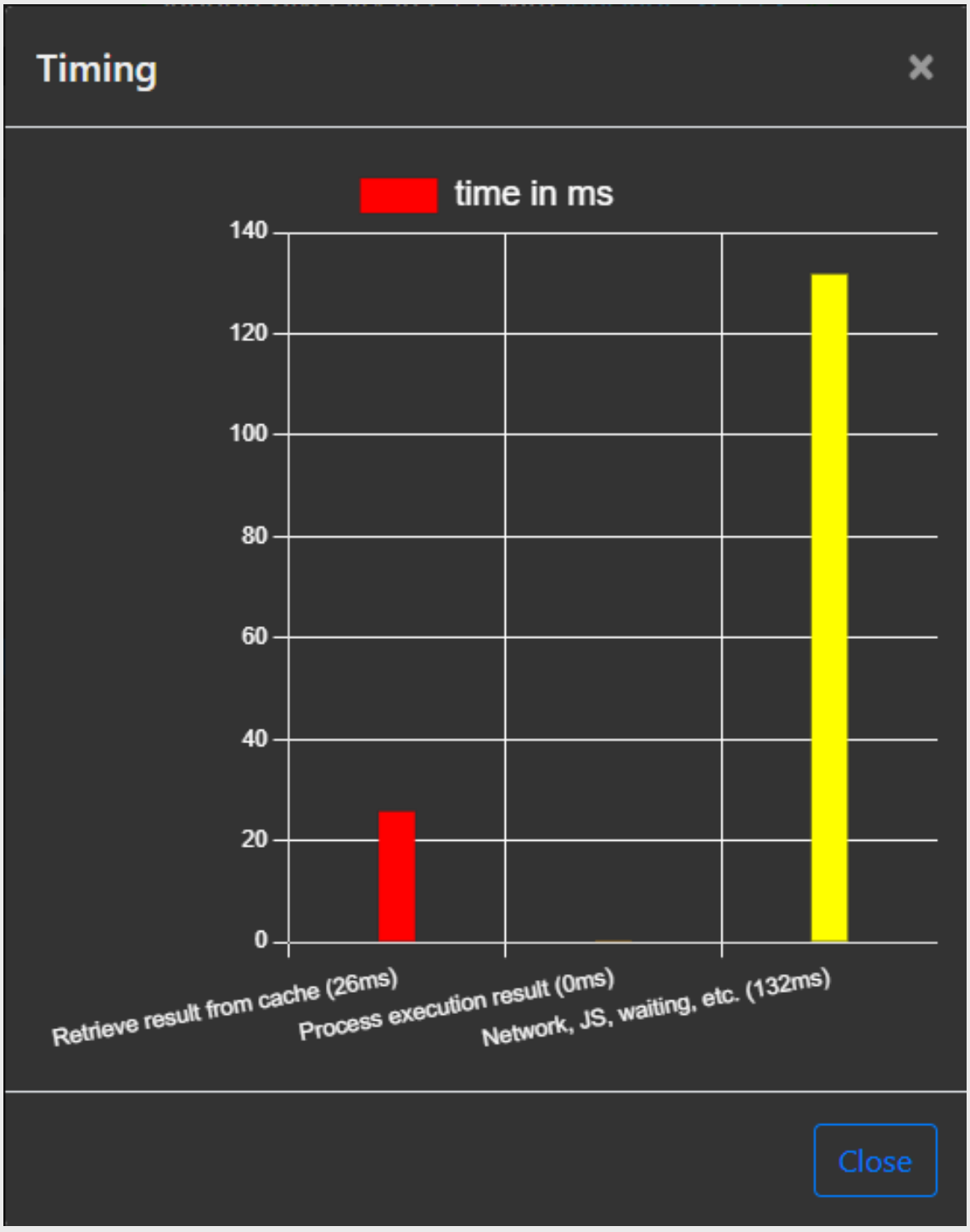
6 Pregunta 6: Análisis de optimización en la generación de `dgemm.s`

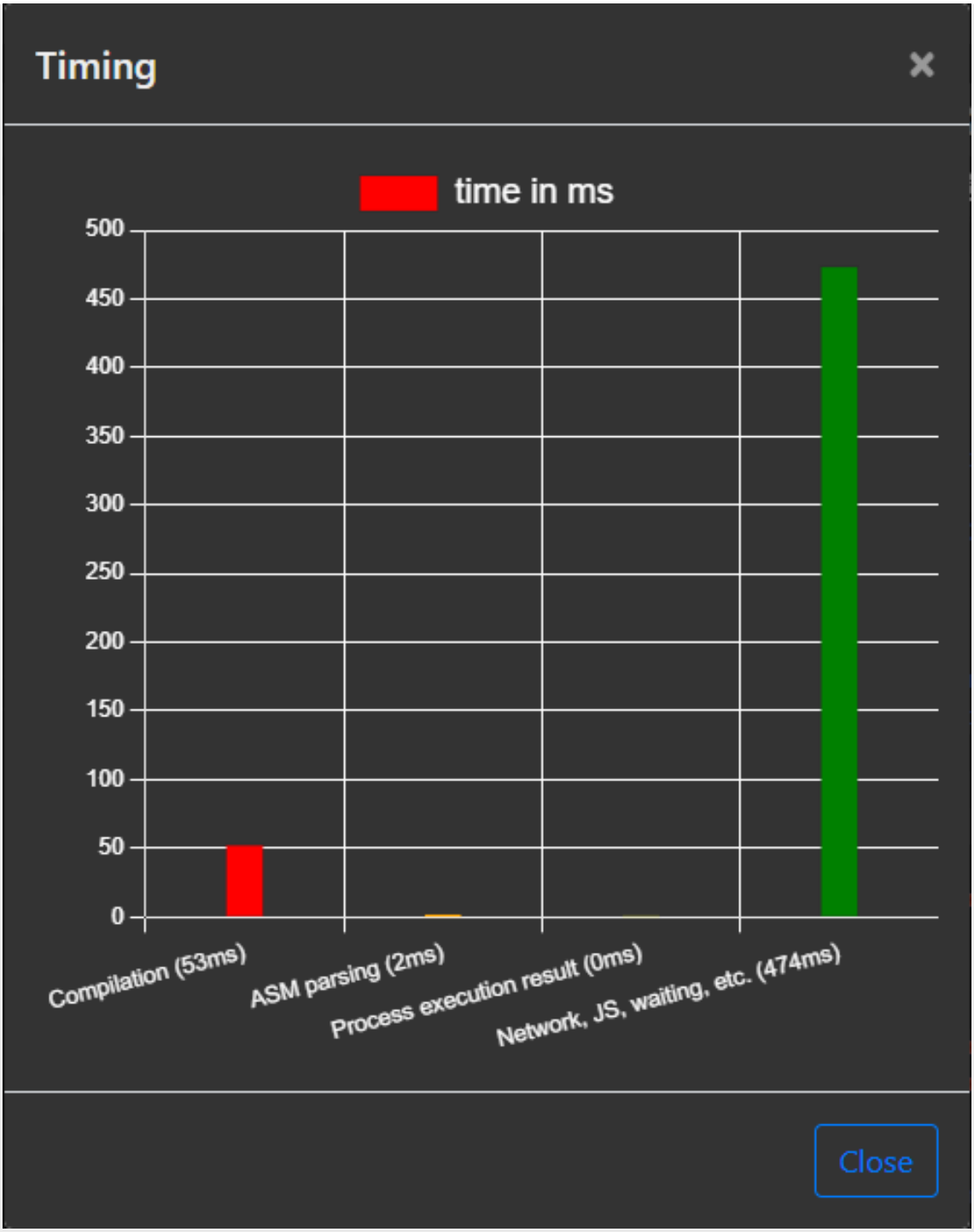
Cambia el valor de –O0 en OPT a –O2 y vuelve a generar el fichero dgemm.s. Explica qué hace cada opción de compilación que se prueba (- O0,-O2) y analiza los cambios que aparecen en el código (ilustrándolos con los bloques de instrucciones que se han modificado) y el impacto que estos cambios tiene en los tiempos de

ejecución.

Repuesta:

Estos son los tiempos de compilación para la `flag -O0` y la `flag -O2` respectivamente:





Como se puede observar el tiempo de compilación cuando se pide una optimización del código ensamblador de nivel 2 (`-O2`) es significantemente mayor aunque no supone mucho problema gracias a lo corto que es el programa. Y significantemente se ha notado en el código ensamblador que se ha generado puesto que cuando no se ejecutan herramientas de optimización el código queda en 81 líneas, mientras que cuando sí se hace el código se queda en 46, además de que simplifica también en etiquetas. Con estas mejoras de optimización el código generado tiene un mejor rendimiento que el primero.

Modificando el parámetro o `flag -O2` para optimizar el código, en el mismo `Makefile` ya se representa bien el cambio en el rendimiento del código reduciendose los tiempos en comparación a las medidas de temporización que se hicieron en el apartado de la *pregunta 3*.

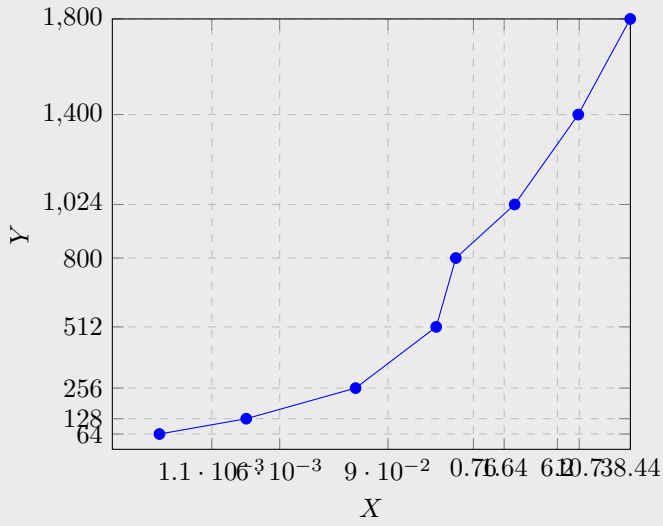
Como se puede observar para los siguientes valores, 64, 256, 1024, se reducen los tiempos pero no cambia la naturaleza exponencial del código que aunque ahora representa una función con valores más próximos al 0 desde un principio, sigue creciendo exponencialmente.

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 64
Description:    A naive C version on matrices of size 64 x 64

Time 2.949582e-04 s
```

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 3.399405e-02 s
```

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark 1024
Description:    A naive C version on matrices of size 1024 x 1024

Time 2.133210e+00 s
```

Los valores aproximados de tomar varias mediciones se representan en la siguiente gráfica.



Una apreciación de los resultados obtenidos es que a partir de la 7ª medida (`n = 1400`) los resultados no tienen una diferencia notable a los conseguidos sin optimizar el código, sólo se diferencian en unas pocas centésimas.

7 Bibliografía y enlaces

- https://godbolt.org/
- https://es.stackoverflow.com/
- https://es.overleaf.com/learn/latex/Tutorials