1. Práctica de Optimización

1.1 Producto de Matrices

Código C que vamos a utilizar:

```
void basic_gemm(int n, float *A, float *B, float *C){
  for (int i=0; i < n; ++i) // índice de la fila
    for (int j=0; j < n; ++j) // índice de la columna
      for (int k=0; k < n; k++) // índice para el producto escalar
        C[i*n+j] = C[i*n+j] + A[k+i*n]*B[j+k*n];
}
```

Este código se puede mejorar utilizando la versión Naïve de este:

```
void naive_gemm(int n, float *A, float *B, float *C){
  for (int i=0; i < n; ++i) // índice de la fila
    for (int j=0; j < n; ++j){ // índice de la columna
      float cij = C[i*n+j];
      for (int k=0; k < n; k++) // índice para el producto escalar
        cij += cij + A[k+i*n]*B[j+k*n];
        C[i*n+j] = cij;
    }
}
```

Se proporcionan tres ficheros con los que vamos a realizar las pruebas y, en base a estas, vamos a responder las cuestiones:

1.1.1 Prueba a ejecutar `make` y comprueba los ficheros que se generan. Después ejecuta `make clean`, ¿qué ha ocurrido?

- Se genera un fichero `benchmark` sin extensión y otro con extensión `.o` y, de `dgemm.c`, se genera un fichero con la extensión `.o`.
- Después, ejecutamos `make clean` y se eliminan dichos ficheros.

1.1.2 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 $A + n * n * sizeof(float)$?

- Porque el puntero *A* es de tipo *float*, por lo que es innecesario utilizar el tamaño de dicho tipo ya que está implícito en el puntero.

1.1.3 Compila el programa con `make` y ejecútalo después. ¿Qué ocurre si no le pasas argumentos? Prueba a dar varios valores (por ejemplo 64, 128, 256, 1024, 4096, 8192) y anota que te da la salida del programa.

Para la realización de este apartado se realizará un *script* en *bash* para ejecutar el programa 5 veces y calcular la media.

ARGUMENTOS	64	128	256	512	1024
TIEMPO	9,36E-04	5,55E-03	4,27E-02	3,39E-01	3,06

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

- Se han generado un `.o` y un `.s` de `dgemm` y de `benchmark`. Además, se ha creado un archivo sin extensión de `benchmark`.
- El archivo `benchmark` es el fichero ejecutable, los `.s` son los programas compilados a lenguaje ensamblador y los `.o` son los binarios de esos programas. A partir de estos dos se crea el ejecutable.

1.1.5 Examina el fichero `dgemm.s` e intenta determinar qué es lo que hacen las instrucciones que aparecen. Puedes usar la herramienta *Compiler Explorer* para ayudarte.

- En primer lugar se define donde se van a guardar los valores de los parámetros, en este caso, se usa un *frame pointer* para guardar las variables, concretamente las posiciones: *[rbp - 20]* guarda el parámetro *n*, *[rbp - 32]* guarda *A, [rbp - 40]* guarda *B y [rbp - 48]* guarda *C.*
- El bucle que recorre las filas está controlado por *L7*, donde se crea la variable *i* la cual se guarda en *[rbp-4]*. Se compara *i* con *n* mientras que sea menor que *n,* en otro caso, el *frame pointer* se salva porque habría terminado la rutina.
- Ahora entraríamos en *L6*, que se encarga de recorrer cada columna de la fila en la que se encuentra mediante la variable *j* que se encuentra en *[rbp - 8]* y de guardar en *[rbp - 12]* el resultado de *C[i*n + j]*.
- Por último, se entra en *L5*, que se encarga de recorrer el último bucle de la misma forma que en los anteriores y de realizar las últimas dos operaciones.

1.1.6 Cambia el valor de -O0 a -O2 y vuelve a generar el fichero `dgemm.s` ¿Puedes identificar cambios en el código?

- El primer cambio que podemos apreciar es a la hora de guardar las variables, ya que no se guardan en direcciones de memoria, si no que se guardan en registros.

- Otro cambio lo podemos encontrar en los bucles, los bucles secundarios se limitan a comparar sus variables para saber si se ha terminado, mientras que el primer bucle es el que se encarga de modificar las variables de los bucles secundarios, es decir, es el que controla los bucles.