MÉTODOS NUMÉRICOS PARA LA COMPUTACIÓN

Tema 4: Librería BLAS

2020/21

---

25 de noviembre de 2020

Grupo 03: José María Amusquívar Poppe y Prashant Jeswani Tejwani

Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería en Informática

Índice

Actividad práctica 2

Se realizan los siguientes ejercicios usando la librería CBLAS nivel 1. Se emplea Matlab para verificar que el resultado es correcto.

Ejercicio 1

Se ha definido dos vectores 3D (Figura 1 y 2) y comprobado que su producto escalar es nulo:

```
v1 = [3.0, -1.0, -3.0];
v2 = [4.0, 3.0, 3.0];
```

Figura 1. Vectores a operar (MATLAB).

```
double v1[3] = { 3.0, -1.0, -3.0 };
double v2[3] = { 4.0, 3.0, 3.0 };
```

Figura 2. Vectores a operar (C++).

```
double res = cblas_ddot(n, v1, inc, v2, inc);
```

Figura 3. Función de CBLAS para el producto escalar.

Si se ejecuta el respectivo comando de CBLAS en C++ (Figura 3), donde el primer parámetro corresponde con el número de elementos del vector; el segundo y el cuarto con los respectivos vectores a operar; y el tercero y quinto con el valor de incremento, qué en este caso será igual a uno puesto que se desea operar los vectores recorriendo secuencialmente sus índices.

Realizando dicha operación se obtiene un resultado igual a cero, lo mismo ocurre si se realiza esta operación en MATLAB usando la función “dot()”. Entonces, se puede concluir que ambos vectores son ortogonales, por lo que tienen un punto de intersección.

Ejercicio 2

Se ha construido dos vectores conteniendo el valor ASCII de los 10 primeros caracteres de un nombre del integrante del grupo y el apellido del otro (Figura 4 y 5), rellenándolo con cero para llegar a los 10. Posteriormente, se obtiene el resultado de sumar al primer vector el triple del segundo. Y, finalmente, el resultado se mapea a caracteres ‘a….z’.

```
name = 'jose maria';
vN = double(name);
surname = 'jeswani000';
vA = double(surname);
```

Figura 4. Vectores ASCII (MATLAB).

```
char nameC[11] = "jose maria";
double* name = convert(nameC);
char surnameC[11] = "jeswani000";
double* surname = convert(surnameC);
```

Figura 5. Vectores ASCII (C++).

```
cblas_daxpy(n, mult, surname, inc, name, inc);
```

*Figura 6. Operación mult*surname+name (C++).*

Una vez definidos ambos vectores, se ejecuta (en C++) la función de CBLAS “daxpy()” (Figura 6), que multiplica por un escalar “mult” el vector “surname”, y a dicho resultado le suma el vector “name”. En MATLAB se obtiene este resultado haciendo explícitamente lo que se pide:

```
res2 = vN + 3*vA;
```

El resultado obtenido usando CBLAS se sobrescribe en el vector “name”, cuyos valores son:

```
Ejercicio 2:
        Valores resultantes de las operaciones (valor=>ASCII):
                {424=>121, 414=>111, 460=>107, 458=>105, 323=>120, 439=>111, 412=>109,
 258=>105, 249=>121, 241=>113}
        Valores ASCII mapeados en 'a'..'z':
                {y, o, k, i, x, o, m, i, y, q}
```

Figura 7. Resultado y mapeado de los valores.

La primera lista de valores mostrados corresponden con los valores obtenidos del cálculo realizado usando “daxpy()” (Figura 6). Esta lista está compuesta por un par de valores, el primero representa el valor obtenido, su pareja corresponde con el valor mapeado en el rango 97-122 (a-z) en código ASCII decimal. Finalmente, la segunda lista corresponde con los caracteres mapeados obtenidos.

Ejercicio 3

Se crea un vector conteniendo todos los dígitos de la fecha de nacimiento (Figura 8 y 9). La nota final de la asignatura de MNC es el resultado de calcular el módulo 11 de la norma2 de ese vector.

```
double date[8] = {2.0, 4.0,
                  0.0, 2.0,
                  1.0, 9.0, 9.0, 9.0};
```

Figura 8. Vector cumpleaños José María (C++).

```
d = [2,4,0,5,1,9,9,9];
```

Figura 9. Vector cumpleaños Prashant (MATLAB).

```
double norm2 = cblas_dnrm2(n, date, inc);
double res = std::fmod(norm2, 11.0);
```

Figura 10. Cálculo de la norma y el módulo 11 (C++).

```
res3 = mod (norm(d, 2), 11);
```

Figura 11. Cálculo de la norma y el módulo 11 (MATLAB).

Para realizar este apartado, primero se comprobó que los resultados coincidían en ambos entornos. Una vez confirmado esto, se ha usado ambas fechas de cumpleaños para realizar los cálculos. El resultado de la Figura 10 y 11 es **5.3707** y **6**, respectivamente. Podría haber ido peor.

Ejercicio 4 (optativo)

A continuación, se estudia el efecto de los parámetros de incremento en C++ del ejercicio anterior.

El incremento especifica el espacio guardado entre índice e índice, por tanto, si se especifica un incremento de 1, el array será recorrido secuencialmente empezando por el primer elemento; si se especifica un 2, el array será recorrido dando saltos de 2 en 2, índice 0, índice 2, índice 4... Sin embargo, el incremento también modifica el valor máximo hasta el que se recorre:

(1 + (n -1) *abs (incx))

donde "n" es el número de elementos del vector e "incx" el incremento, por tanto, si éste último es 1, el máximo coincide con el número de elementos del vector, pero si éste es 2 el máximo será el

doble del número de elementos menos 1. Por ello, se debe tener cuidado al especificar este valor pues puede generar cálculos erróneos accediendo a posiciones de memoria incorrectos.

Por ejemplo, si en este ejercicio se especifica un incremento distinto a 1 sin modificar el número de elementos, se obtiene un valor muy grande que es incorrecto (Figura 12). Si se desea iterar sobre los índices pares (2), entonces el número de elementos debe reducirse a la mitad, 4 (Figura 13).

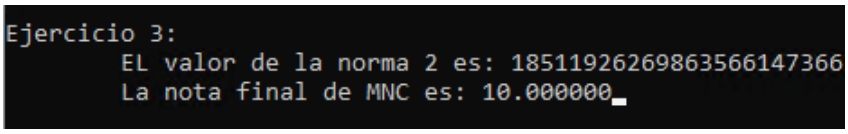

Figura 12. Incremento igual a 2, número de elementos igual a 8. Erróneo.

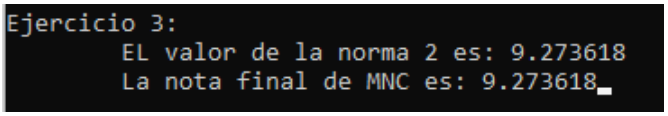

Figura 13. Incremento igual a 2, número de elementos a 4. Correcto.

Si se especifica un incremento de 0, entonces se realiza la norma únicamente del primer elemento del vector (date [0] = 2.0).

Actividad práctica 3

Se realizan los siguientes ejercicios usando la librería CBLAS nivel 2. Se emplea Matlab para verificar que el resultado es correcto.

Ejercicio 1

Se define una matriz (A) y dos vectores (x, y) (Figura 14) para realizar las siguientes operaciones:

```
double A[9] = { 7.0, -3.0, 12.0, 2.0, 15.0, -9.0, -4.0, -19.0, 5.0 };
double x[3] = { -3.0, 9.0, 2.0 };
double y[3] = { 5.0, 17.0, 11.0 };
```

Figura 14. Inicialización de vectores (C++).

Para realizar estas operaciones se emplea la función “dgemv()”, que trabaja a nivel matriz-vector. Esta función acepta parámetros que le especifican si se desea recorrer por filas o columnas, si se traspone la matriz o no, el incremento de los índices, los escalares que multiplicarán los objetos, así como sus dimensiones y los objetos en sí.

```
cblas_dgemv(layout, trans, m, n, alpha, A, lda, x, incx, beta, y, incy);
```

Figura 15. Función utilizada de CBLAS (C++).

a) A*x

La matriz “A” es de 3 por 3, así que el vector “x” será de 3 por 1, y el vector resultado será un vector columna de 3 filas. Empleando la función mostrada en la Figura 15, pasando el parámetro “beta” igual a 0 para que la operación al lado derecha de la suma se elimine.

Comprobando estos resultados MATLAB de la siguiente manera: `aRes = A*X';`
El resultado obtenido en MATLAB de esta operación es:

```
A*X =
{
-24.0
 111.0
 -149.0
 }
```

Figura 16. Resultado obtenido (MATLAB)

b) 3*A*x+4*y

La matriz “A” es de 3 por 3, así que el vector “x” será de 3 por 1, y se multiplicará por el escalar 3; el vector resultado de esta operación será un vector columna de 3 filas. Por lo que el vector “y” tendrá que ser un vector columna (se traspone), multiplicado por el escalar 4. Es resultado de esta operación será un vector columna (al igual que el anterior) de 3 filas.

Comprobando estos resultados MATLAB de la siguiente manera:

```
bRes = 3*A*X' + 4*Y';
```

*Figura 17. Operación 3*A*x + 4*y (MATLAB)*

El resultado obtenido en MATLAB de esta operación es:

```
3*A*X + 4*Y =
{
-52.0
401.0
-403.0
}
```

Figura 18. Resultado obtenido (MATLAB)

Ejercicio 2

Se prueba el efecto de los parámetros de 'layout' y trasposición en C++ del ejercicio anterior.

En el anterior apartado, las operaciones fueron realizadas recorriendo los elementos por filas y sin trasponer la matriz. En este apartado se comprobarán los tres casos restantes.

Recorrer la matriz por columnas y trasponiéndola. Es lógico que los resultados obtenidos en esta prueba sean los mismos que si se recorre la matriz por filas y no se traspone.

```
layout = CblasColMajor;
trans = CblasTrans;
cblas_dgemv(layout, trans, m, n, alpha, A, lda, x, incx, beta, y, incy);
```

Figura 19. Se recorre por columnas y matriz traspuesta (C++).

Recorrer la matriz por columnas y sin trasponer. Esta prueba obtiene unos resultados no vistos hasta ahora, puesto que es un nuevo enfoque.

```
layout = CblasColMajor;
trans = CblasNoTrans;
cblas_dgemv(layout, trans, m, n, alpha, A, lda, x, incx, beta, Y1, incy);
```

Figura 20. Se recorre por columnas y sin matriz traspuesta (C++).

Recorrer la matriz por filas y trasponiéndola. Esta prueba obtiene unos resultados idénticos a los obtenidos en la prueba de recorrido por columnas y no traspuesta.

```
layout = CblasRowMajor;
trans = CblasTrans;
cblas_dgemv(layout, trans, m, n, alpha, A, lda, x, incx, beta, Y2, incy);
```

Figura 21. Se recorre por filas y matriz traspuesta (C++).

Como se puede apreciar, de las 4 configuraciones posibles sólo existen dos variaciones posibles de resultados, ya que las otras dos son simplemente enfoques reflejados de éstas.

Actividad práctica 4

Se realizan los siguientes ejercicios usando la librería CBLAS nivel 3. Se emplea Matlab para verificar que el resultado es correcto.

Ejercicio 1

Se definen tres matrices (A, B y C) de dimensión 3x3 para realizar las siguientes operaciones:

a) A*B

Las matrices creadas son de tipo "Double":

```
double A[9] = {-5.0, 7.0, 11.0, 1.0, -14.0, 3.0, -9.0, 5.0, 16.0};
double B[9] = {8.0, 15.0, 1.0, -5.0, 16.0, -4.0, 4.0, 10.0, -19.0};
double C0[9] = {8.0, 9.0, -7.0, 17.0, -3.0, 8.0, -1.0, 4.0, 1.0};
```

Figura 22. Matrices A, B y C (C++)

```
A = [-5.0 7.0 11.0; 1.0 -14.0 3.0; -9.0 5.0 16.0];
B = [8.0 15.0 1.0; -5.0 16.0 -4.0; 4.0 10.0 -19.0];
C = [8.0 9.0 -7.0; 17.0 -3.0 8.0; -1.0 4.0 1.0];
```

Figura 23. Matrices A, B y C (MATLAB)

A continuación, se pasan los parámetros correspondientes a la función "cblas_dgemm()" para obtener el resultado de la operación, donde alpha = 1, beta = 0.

```
cblas_dgemm(layout, transA, transB, m, n, k, alpha, A, lda, B, ldb, beta, C0, ldc);
```

*Figura 24. Operación A*B (C++)*

```
Calculo de A*B:
        -31.0   147.0   -242.0
        90.0    -179.0   0.0
        -33.0   105.0   -333.0
```

Figura 25. Resultado obtenido

Se comprueban los resultados en MATLAB de la siguiente manera:

```
AB = A*B;
```

*Figura 26. Operación A*B (MATLAB)*

b) $A*B^T$

Para la realización de este apartado simplemente se modifica el parámetro "transB" estableciéndolo a "CblasTrans" y se llama a la misma función que el apartado anterior:

```
double C1[9] = { 8.0, 9.0, -7.0, 17.0, -3.0, 8.0, -1.0, 4.0, 1.0 };
transB = CblasTrans;
```

Figura 27. Modificación de los parámetros (C++)

```
cblas_dgemm(layout, transA, transB, m, n, k, alpha, A, lda, B, ldb, beta, C1, ldc);
```

*Figura 28. Operación A*Bt (C++)*

```
Calculo de A*Bt':
        76.0    93.0    -159.0
        -199.0  -241.0  -193.0
        19.0    61.0    -290.0
```

Figura 29. Resultado obtenido

Se comprueban los resultados en MATLAB de la siguiente manera:

```
ABt = A*transpose(B);
```

*Figura 30. Operación A*Bt (MATLAB)*

c) 2*A*B + 3*C

En este apartado se modifican los parámetros alpha y beta para la realización de la operación:

```
double C2[9] = { 8.0, 9.0, -7.0, 17.0, -3.0, 8.0, -1.0, 4.0, 1.0 };
transB = CblasNoTrans;
alpha = 2;
beta = 3;
```

Figura 31. Cambio de los parámetros (C++)

```
cblas_dgemm(layout, transA, transB, m, n, k, alpha, A, lda, B, ldb, beta, C2, ldc);
```

*Figura 32. Operación 3*A*B + 4*C (C++)*

Se obtiene como resultado la siguiente matriz:

```
Calculo de 2*A*B + 3*C:
        -38.0   321.0   -505.0
        231.0   -367.0  24.0
        -69.0   222.0   -663.0
```

Figura 33. Resultado obtenido

Finalmente, se comprueban los resultados en MATLAB de la siguiente manera:

```
ABC = 2*A*B + 3*C;
```

*Figura 34. Operación A*Bt (MATLAB)*

Ejercicio 2 (optativo)

Se realiza una operación que implique matrices no cuadradas para generar como resultado una matriz de 5x5.

Para ello se inicializan dos matrices (M1, M2), donde la dimensión de M1 es 5x2 y la de M2 es 2x5, dando como resultado una matriz (M3) de 5x5.

```
double M1[10] = { 2.0, -5.0, -4.0, 3.0, 3.0, 3.0, -4.0, -8.0, 7.0, 9.0 };
double M2[10] = { 5.0, 6.0, -9.0, 6.0, 4.0, -3.0, -6.0, -5.0, 7.0, 2.0 };
double M3[25] = { -2.0, 3.0, -5.0, -5.0, 3.0, -2.0, 7.0, 6.0, -5.0, 0.0, -2.0,
   5.0, -5.0, 8.0, 7.0, -1.0, 9.0, -5.0, -6.0, 2.0, -1.0, 0.0, 3.0, 5.0, -3.0 };
```

Figura 35. Matrices M1, M2 y M3 (C++)

```
M1 = [2.0 -5.0; -4.0 3.0; 3.0 3.0; -4.0 -8.0; 7.0 9.0];
M2 = [5.0 6.0 -9.0 6.0 4.0; -3.0 -6.0 -5.0 7.0 2.0];
M3 = [-2.0 3.0 -5.0 -5.0 3.0; -2.0 7.0 6.0 -5.0 0.0; -2.0 5.0 -5.0 8.0 7.0;
      -1.0 9.0 -5.0 -6.0 2.0; -1.0 0.0 3.0 5.0 -3.0];
```

Figura 36. Matrices M1, M2 y M3 (MATLAB)

La operación que se realiza es la siguiente: **6 * M1**(5x2) *** M2**(2x5) **- 6 * M3**(5x5). Donde alpha = 6 y beta = -6:

```
cblas_dgemm(layout, transA, transB, m, n, k, alpha, M1, lda, M2, ldb, beta, M3, ldc);
```

*Figura 37. Operación 6*M1*M2 - 6*M3 (C++)*

Se obtiene como resultado una matriz de 5x5:

```
Calculo de 6*M1*M2 -6*M3:
        162.0   234.0   72.0    -108.0  -30.0
        -162.0  -294.0  90.0    12.0    -60.0
        48.0    -30.0   -222.0  186.0   66.0
        30.0    90.0    486.0   -444.0  -204.0
        54.0    -72.0   -666.0  600.0   294.0
```

Figura 38. Resultado obtenido

Se comprueban los resultados en MATLAB de la siguiente manera:

```
opt = 2*M1*3*M2 - 6*M3;
```

*Figura 39. Operación 6*M1*M2 - 6*M3 (MATLAB)*

Actividad práctica 5

Para realizar los siguientes apartados se emplea una función que genera una matriz de números aleatorios (“Double” o “Float”), según se requiera. Para llevar este cometido se debe reservar el espacio necesario en memoria acorde al tipo de datos que se almacenará.

```
double* generateMatrixDouble(int m, int n) {
    double *X = (double*)mkl_malloc((double)m*(double)n * sizeof(double), 64);

    if (X == NULL) {
        perror("Error malloc");
        exit(1);
    }

    for (int i = 0; i < m*n; i++) {
        X[i] = (double)rand() / (double)RAND_MAX;
    }

    return X;
}
```

Figura 40. Función que genera una matriz de tipo Double (C++).

Las funciones usadas para el manejo de memoria son “mkl_malloc()” y “mkl_free()”, la primera reserva espacio en memoria, la segunda libera dicho espacio.

Ejercicio 1

Se crean tres matrices (A, B y C) de dimensión NxN y se rellenan con valores aleatorios tipo ‘double’.

Para realizar este apartado, simplemente se llama al método mencionado en la Figura 39, pasándole como parámetros el número de filas y el número de columnas, qué en este caso, ambos valores coinciden al tratarse de una matriz cuadrada.

```
double *A = generateMatrixDouble(3, 3);
double *B = generateMatrixDouble(3, 3);
double *C = generateMatrixDouble(3, 3);

mkl_free(A);
mkl_free(B);
mkl_free(C);
```

Figura 41. Generación de 3 matrices, y liberación de estas (C++).

Recalcar que no se tratan de matrices en sí, sino de vectores. Para tratarlos como matrices simplemente se especifica el número de filas y columnas en la función determinada de “CBLAS”.

Ejercicio 2

Se calcula el número de GFLOPS para los distintos valores de N, realizando el promedio de 100 ejecuciones de la operación A*B.

Para llevar a cabo este apartado, se utilizará la función de "CBLAS" que opera matrices con matrices, y que viene definido como:

```
cblas_dgemm(layout, transA, transB, m, n, k, alpha, A, lda, B, ldb, beta, C, ldc);
```

*Figura 42. Operación A*B (C++)*

Tal y como se explicó en apartados anteriores, esta función también recibe varios parámetros, de los que cabe destacar que "beta" es igual a cero puesto que se desea eliminar el operando a la derecha de la suma. De este modo sólo se opera la multiplicación de matrices.

Para el cálculo de este apartado se ha desarrollado un bucle que recorre desde 100 hasta 1000, dando saltos de 20 en 20. En cada iteración de este bucle se ejecuta un bucle interno de 100 iteraciones que contiene la instrucción mencionada arriba. Así pues, en cada iteración del bucle superior genera 3 nuevas matrices cuadradas cuya dimensión corresponde con el número de iteración. Estas tres matrices serán las que se operarán en el bucle interno usando la función correspondiente de "CBLAS" (la matriz "C" es irrelevante, ya que no forma parte de la ecuación solicitada).

```
start = dsecnd();
for (int j = 0; j < 100; j++) {
    cblas_dgemm(layout, transA, transB, m, n, k, alpha, A, lda, B, ldb, beta, C, ldc);
}
fin = (dsecnd() - start);
```

Figura 43. Cálculo del tiempo total de 100 iteraciones (C++).

De este modo, se puede calcular el tiempo que tarda en ejecutar 100 operaciones de "A*B", guardando el tiempo antes de entrar a este bucle interno, y restándolo al obtenido después de salir del bucle interno, tal como se visualiza en la Figura 43.

Una vez obtenido el tiempo total de ejecución, se procede a calcular el número de GFLOPS, cuyo valor será almacenado en un vector de tiempos.

```
totalTime[p] = (100 / fin) / 1e9;
```

Figura 44. Cálculo de GFLOPS

Estos valores serán de utilidad para la comparativa de tiempos con el siguiente apartado.

Ejercicio 3

Se repite las pruebas utilizando el modo 'Parallel' para comparar los resultados.

Este apartado es idéntico al apartado anterior, pues el código se reutiliza, simplemente se cambia el tipo de ejecución, cambiándolo de modo Secuencial a modo Paralelo. Hecho esto, se procede a calcular unos nuevos tiempos de ejecución para poder comparar con el anterior.

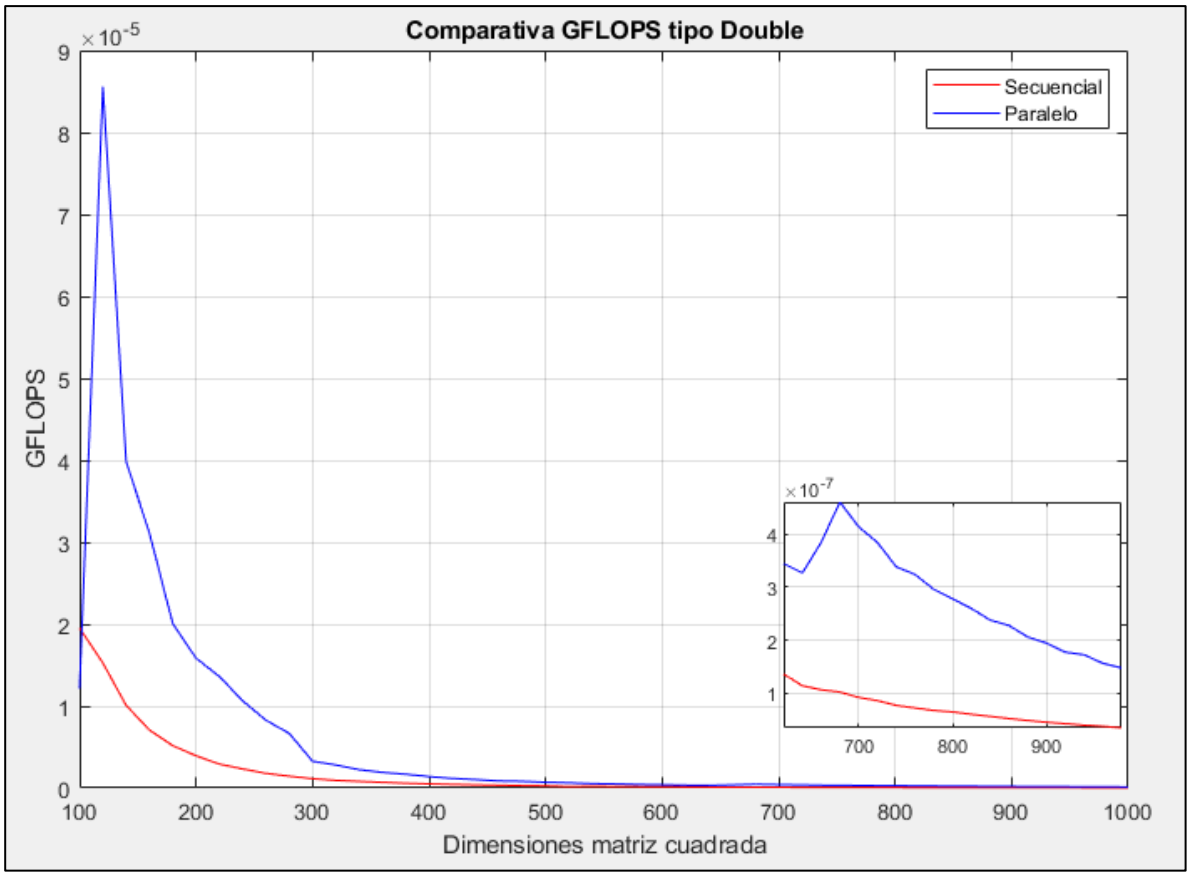

Figura 45. Comparativa de GFLOPS entre modo secuencial y paralelo en C++.

Tal como se aprecia en la gráfica anterior, el modo paralelo es capaz de realizar más operaciones por segundo. Esto se debe, claramente, a que el modo paralelo puede utilizar varios núcleos del procesador según sean necesarios, y también emplea hilos de ejecución, todo ello con el objetivo de realizar varias tareas a la vez, pues tal como dice su nombre, paraleliza procesos.

Ejercicio 4 (optativo)

Se realizan pruebas con aritmética de precisión simple para comparar los resultados.

Al igual que los apartados anteriores, éste simplemente difiere en el tipo de matriz generada, pues ahora se ha de usar una matriz de tipo “Float” (4 bytes), la mitad que una de tipo “Double” (8 bytes).

Se ha calculado el número de GFLOPS tanto en modo secuencial como en modo paralelo, para poder realizar una adecuada comparativa.

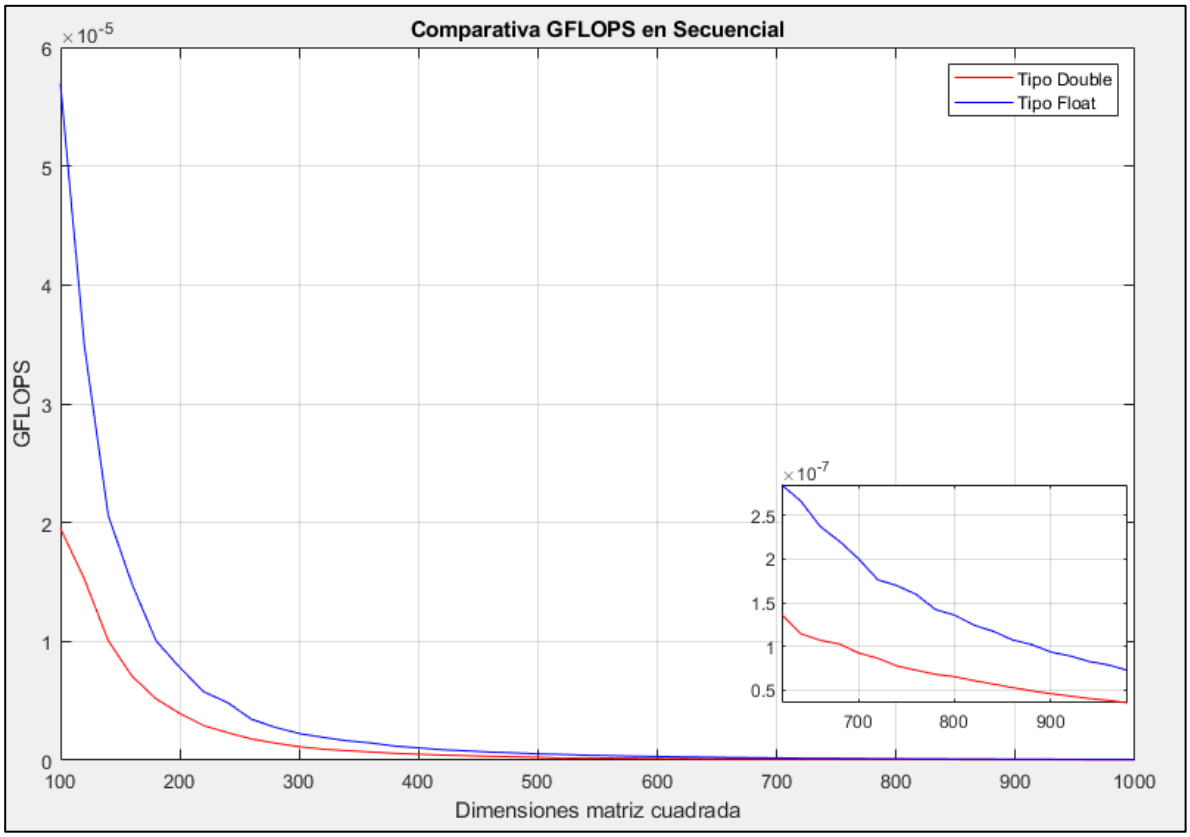

Figura 46. Gráfica comparativa en modo Secuencial en C++.

Tal como se puede apreciar en la Figura 46, al comparar los resultados en modo secuencial de tipo de datos “Double” y “Float”, con tipo “Float” se puede alcanzar un mayor número de GFLOPS que si se emplea tipo “Double”. Esto se debe al tamaño de bytes que se emplea para almacenar estos datos, pues el tipo “Float” emplea, únicamente, 4 bytes, al contrario que el tipo “Double” que emplea el doble, 8 bytes. Entonces, analizando la gráfica y estos datos, se puede concluir que usando datos de tipo “Float” es posible alcanzar el doble o más en cuanto GFLOPS que empleando “Double”.

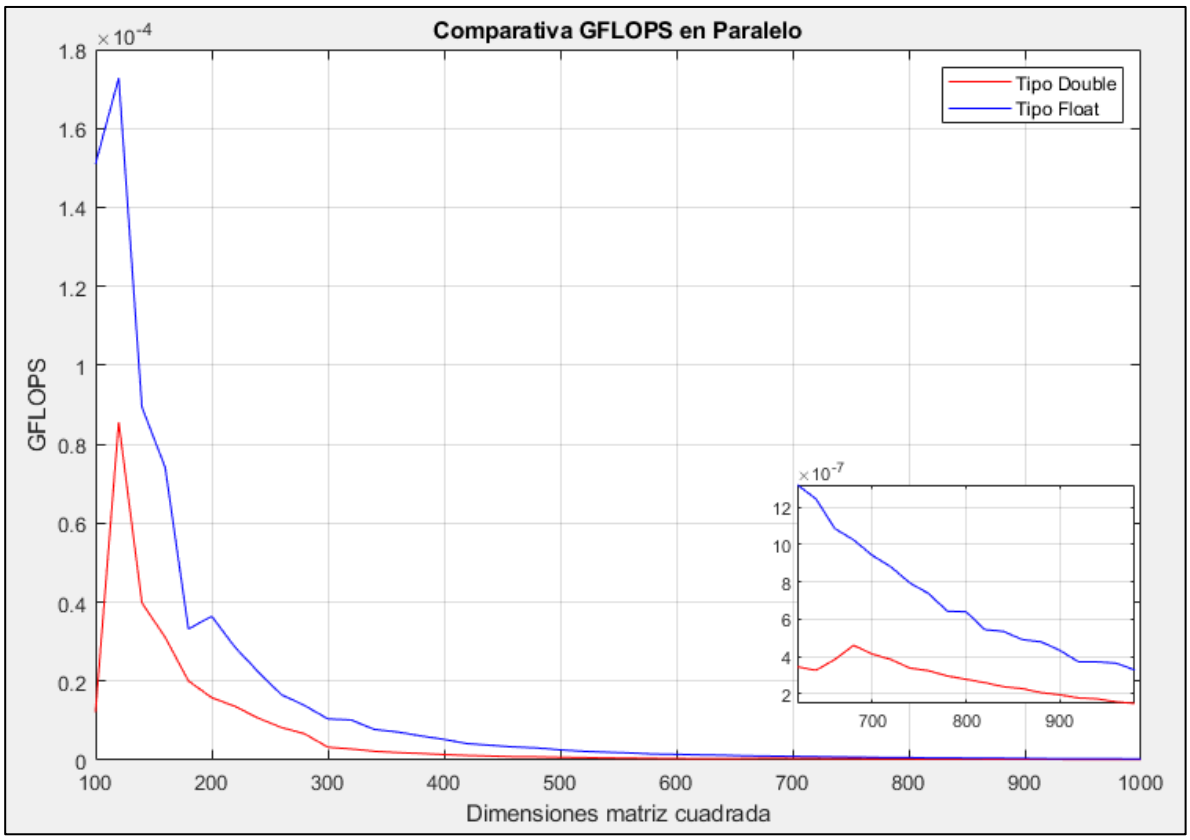

Figura 47. Gráfica comparativa en modo paralelo.

Analizando la gráfica de la Figura 47, sabiendo que usar el modo paralelo otorga mayor eficacia en cuanto a tiempos de ejecución que el modo secuencial, se puede concluir de la misma manera que el apartado anterior, ya que los datos de tipo “Float” alcanzan un mayor número de GFLOPS que los de tipo “Double”.

Ejercicio 5 (optativo)

Se realiza una comparativa de tiempos de ejecución con otros lenguajes como Python y Java.

Para este apartado se calculará el tiempo promedio para cada dimensión de matriz, empleando datos de tipo "Double", con dimensiones que va desde 100 hasta 1000 dando saltos de 20 en 20. Para hacer más cómoda la realización de esto, para cada lenguaje de programación se ha generado un fichero ".csv", los cuales serán cargados y procesados por Matlab, con el fin de generar una única gráfica que enseñe una correcta comparativa.

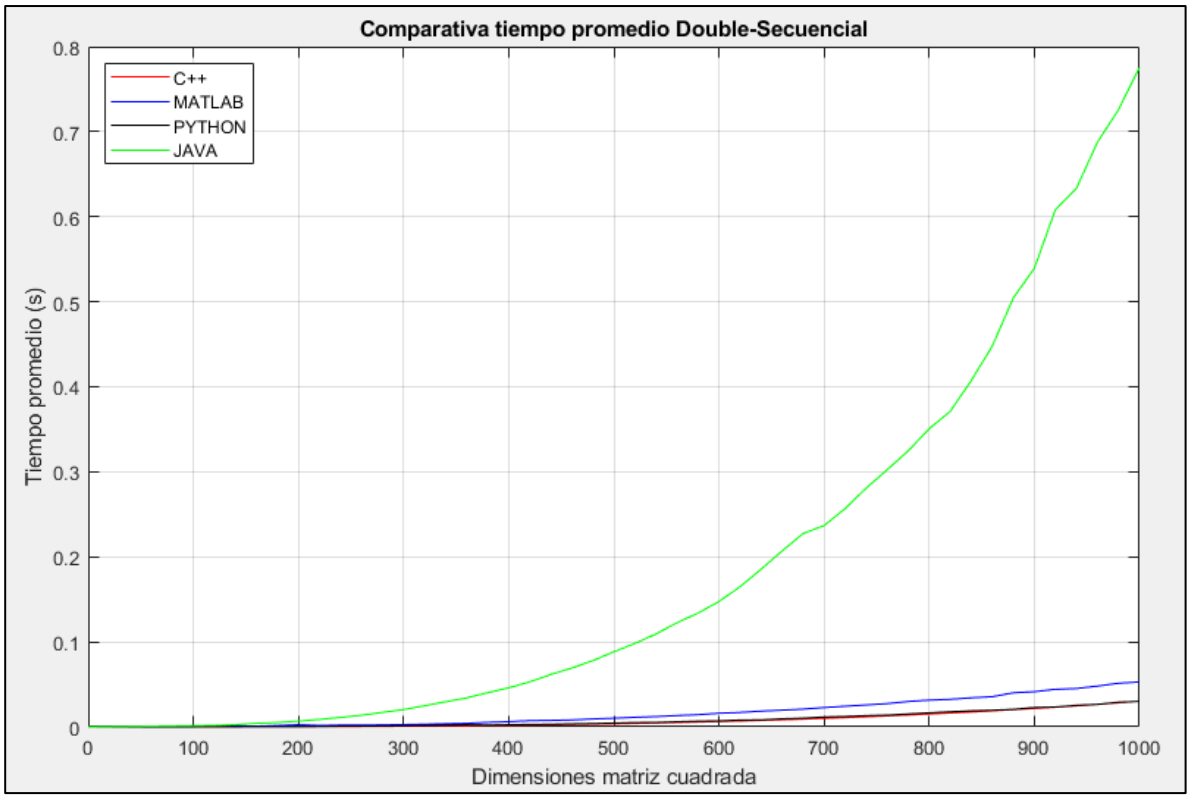

Figura 48. Comparación del tiempo promedio de ejecución de todos los lenguajes elegidos.

Se puede apreciar que los tiempos obtenidos en Java son elevados ya que emplea una distinta metodología de cálculo que C++ o Python, en específico, para realizar el cálculo de matrices en Java se empleó la librería "ejml" (Efficient Java Matrix Library).

Puesto que los tiempos de JAVA están muy distantes del resto, se desarrolla una gráfica final que contenga simplemente los tiempos de Matlab, Python y C++.

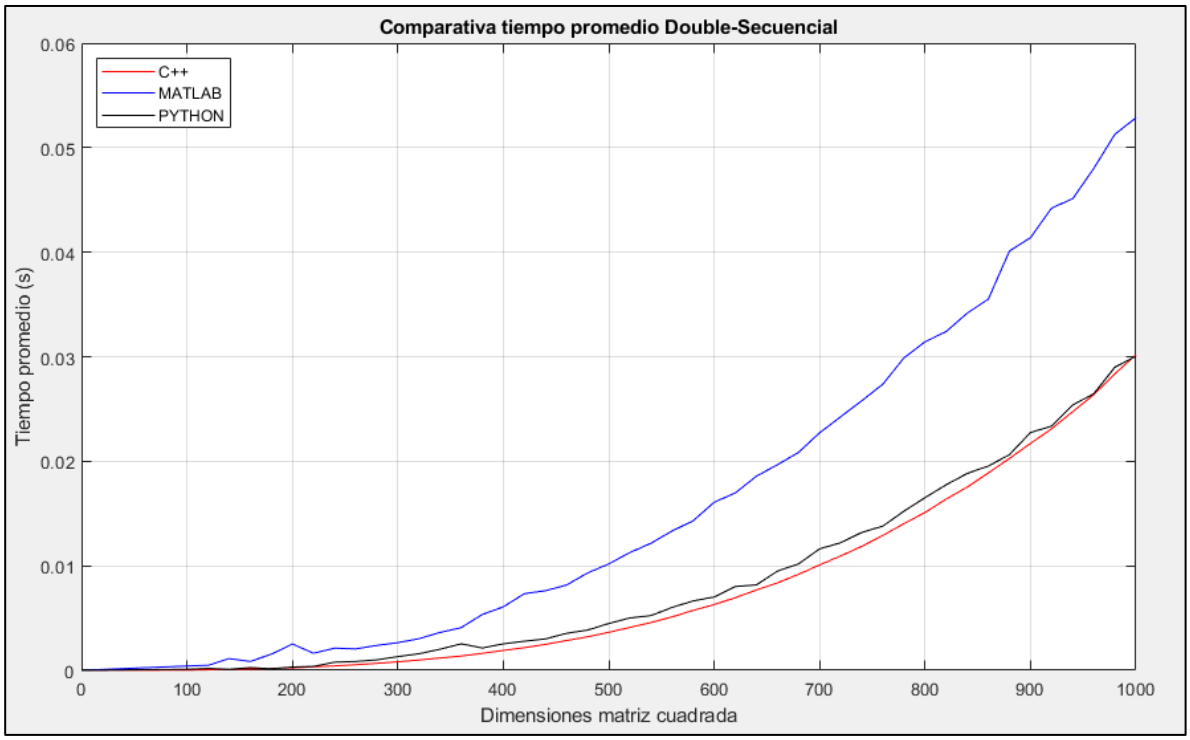

Figura 49. Comparación del tiempo promedio de ejecución de los lenguajes con tiempos equiparables.

Los tiempos obtenidos en Python sorprenden y llegan a ser muy similares a C++ (aunque superiores). Esto se debe a que se ha hecho uso de la librería "NumPy" la cual está vinculada a bibliotecas de álgebra lineal optimizadas como BLAS y LAPACK que tienen efectos importantes en el rendimiento.

MATLAB también utiliza la implementación altamente optimizada de LAPACK de Intel (MKL), específicamente la función "dgemm()". Como no documentan qué algoritmo específico utilizan, si se emplea MKL desde C++, en principio, debería verse un rendimiento similar.

Pues como se puede apreciar en la gráfica de la Figura 49, C++ es más rápido. Esto puede ser debido a la creación de las matrices aleatorias de tipo "Double", que C++ llama directamente a las rutinas de la librería o que es un lenguaje que trabaja a bajo nivel.

Referencias

Daniel, J. (s.f.). *ULPGC*. Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/411998/mod_resource/content/9/4%20BLAS.pdf

GitHub. (s.f.). Obtenido de https://github.com/Prashant-JT/MNC_BLAS