

# Práctica 3

## Librería BLAS



**Héctor Garbisu Arocha**

Curso 2015/16

Métodos Numéricos para la Computación

Grado en Ingeniería Informática

Escuela de Ingeniería Informática

Universidad de Las Palmas de Gran Canaria

# Índice

1. Hola MKL .....	pág. 3
2. Nivel 1 .....	pág. 3
3. Nivel 2 .....	pág. 4
4. Nivel 3 .....	pág. 4
5. Comparativa rendimiento .....	pág. 5

# 1. Hola MKL

El primer ejercicio sirve para aprender a condicionar el lugar de trabajo en Visual Studio. Como es un entorno nuevo para algunos alumnos, esta parte, aunque sencilla, lleva algo de tiempo.

En primer lugar creamos un proyecto vacío, por ejemplo pulsando Ctrl+Shift+N. Es el mismo atajo de teclado que en NetBeans.

Al proyecto se le pueden añadir ficheros vacíos desde el menú de proyecto. Agregamos HolaMKL.cpp y se abre automáticamente el editor de código con ese fichero.

Para que el pequeño programa de ejemplo funcione, es necesario establecer en las opciones del proyecto, en el apartado de Intel Performance Libraries, un modo de funcionamiento para MKL: Sequential o Parallel.

Como MKL incluye las librerías BLAS y LAPACK (o sus versiones específicas para C), desde que se “activa” MKL, el editor reconoce la inclusión de <mkl.h>.

El código de ejemplo del guión de la práctica muestra una consola con el mensaje impreso y el tiempo de ejecución en microsegundos. La ventana se cerraría sola si no se hubiera pedido la entrada por teclado (getchar).

## 2. Nivel 1


Las operaciones de Nivel 1 son las que tienen vectores como entrada y salida. Las matrices de más dimensiones se tratan en los niveles 2 y 3.

Vamos a probar tres operaciones diferentes de nivel 1:

La norma 2 (euclídea) de los vectores A y B, producto escalar esos mismos dos vectores y una suma ponderada  $B+2A$ .

Para esas operaciones, BLAS tiene sendas funciones con un nombre más o menos descriptivo. Por ejemplo, la función para la norma es dnorm2: 'd' por doble precisión, 'nrm' para matriz y '2' por el orden.

Si ejecutamos el programa de ejemplo del guión de la práctica se ejecutarán por orden las funciones cblas\_dnorm2, cblas\_ddot y cblas\_daxpy. Mostrando el resultado para unos vectores A y B de 4 elementos, previamente fijados.



```
C:\Users\usuario\Desktop...
Norma de A: 5.477226
Norma de B: 13.190906
Suma ponderada:
7.000000
10.000000
13.000000
16.000000
Húctor Garbisu MCN 2015
```

### 3. Nivel 2

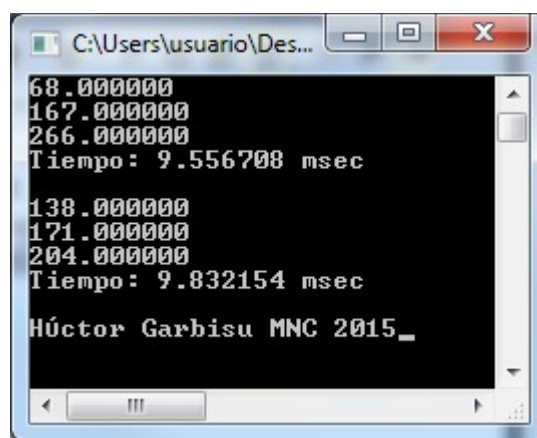
En las operaciones de nivel 2 intervienen tanto matrices como vectores. El producto dos vectores monodimensionales puede resultar en una matriz bidimensional y esa es la operación que se hace con `cblas_dgemv`.

La llamada a la función quedaría de la siguiente forma.

```
cblas_dgemv(CblasRowMajor,CblasNoTrans,3,3,1.0,A,3,B,1,0.0,C,1);
```

Entre todos esos parámetros se indica que se quiere considerar A como una matriz de 3x3, y que queremos sumar C exactamente 0 veces (porque no queremos hacer ese cómputo).

Se ha ejecutado primero una versión en la que no se traspone A, y luego una en la que sí se traspone. El resultado es el siguiente.



```
C:\Users\usuario\Desktop...
68.000000
167.000000
266.000000
Tiempo: 9.556708 msec

138.000000
171.000000
204.000000
Tiempo: 9.832154 msec

Húctor Garbisu MNC 2015_
```

### 4. Nivel 3

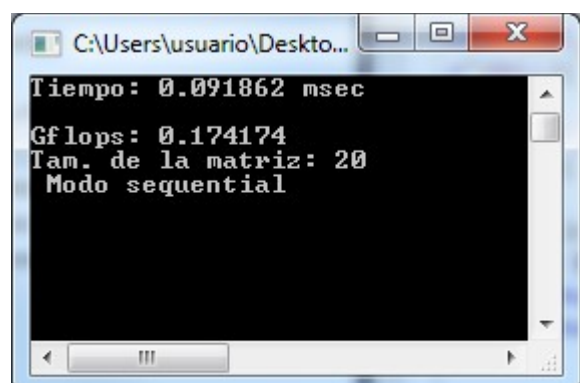
Para terminar de presentar las operaciones de BLAS, haremos operaciones con matrices puras. Concretamente la función `cblas_dgemm`.

Para ello se generan dos matrices aleatorias de NxN elementos y se ejecuta la función que hace la (eficiente) operación:

```
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1, A, N, B, N, 0, C, N);
```

Una vez más hay una enorme lista de parámetros. El escalado de A está a 1 y el de C está a 0 para efectuar  $A*B$  y no otra cosa. El resto sirven para especificar las dimensiones y el tratamiento de las matrices.

Este es el resultado de una ejecución para  $N=20$



```
C:\Users\usuario\Desktop...
Tiempo: 0.091862 msec
Gflops: 0.174174
Tam. de la matriz: 20
Modo sequential
```

## 5. Comparativa de rendimiento

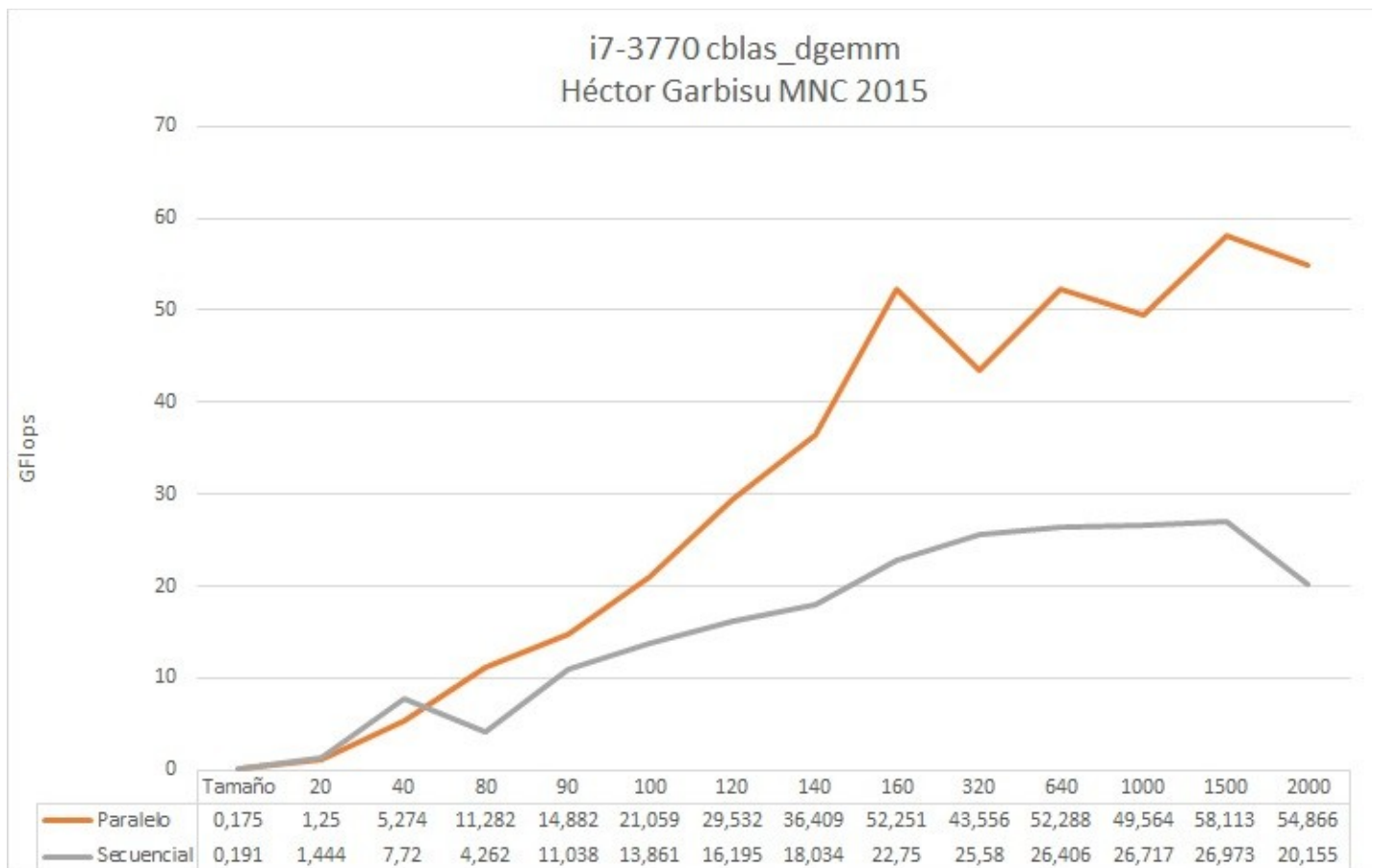
Utilizando el programa del ejercicio anterior, podemos ver de qué forma varía la eficacia de las librerías de alto rendimiento en función del tamaño de las variables. Ya que las funciones están optimizadas para operaciones muy grandes, esperamos que la potencia en Flops que se calcule con ellas sea mayor para ejemplares cada vez mayores hasta que se estabilicen en la que sería aproximadamente la potencia “real”.

Para ello se repite la ejecución para varios tamaños escogidos de forma más o menos representativa, primero con MKL en modo paralelo y luego en secuencial.

Los valores probados varían entre 20 y 2500. Después de una primera ronda de pruebas, comprobé que en el rango entre 80 y 160 se producía una gran subida de potencia calculada; por eso al final esa zona tiene mayor resolución, con nuevas pruebas realizadas en valores de tamaño de 90, 100, 120 y 140.

En la siguiente gráfica no escalada se aprecia la subida de Flops a medida que aumenta el tamaño. Es a partir de 160-200 cuando la eficiencia de la librería alcanza el máximo y la potencia calculada difícilmente sube gracias a incrementar el tamaño.

También se puede ver cómo la mejora de la paralelización sólo empieza a ser notable a partir de ciertos tamaños, llegando incluso a ser más eficiente el modo secuencial si la muestra es muy pequeña.



Este otro gráfico muestra los mismos datos pero a una escala uniforme.

