PRÁCTICA 1

AOC

Alumna: Elena Pérez González-Tablas

Subgrupo: 2.1

PRÁCTICA 1

INFORME

ACCESO AL SERVIDOR (compiladorintel.inf.um.es)

Para realizar las prácticas accedo al servidor compiladorintel.inf.um.es proporcionado por el centro de cálculo con mi cuenta y la primera vez le cambio la contraseña que había por defecto (`passwd`).

```
epgt: $ ssh alumno167@compiladorintel.inf.um.es
```

Copio el archivo con el material de esta práctica de mi ordenador personal al servidor en su directorio `/home`.

```
epgt: $ scp Downloads/AOC-prac1-materiales.tar alumno167@compiladorintel.inf.um.es:
```

Lo descomprimo dentro de la máquina y se crea un directorio llamado materiales dentro del cual tenemos tres directorios.

```
alumno167: $ tar xzf AOC-prac1-materiales.tar
alumno167: $ ls -l
total 20
-rw-r--r-- 1 alumno167 alumno167 11582 sep 30 08:15 AOC-prac1-materiales.tar
drwxr-xr-x 3 alumno167 alumno167  4096 sep 30 09:58 intel
drwxrwxr-x 5 alumno167 alumno167  4096 sep 30 08:24 materiales
```

CARACTERÍSTICAS DEL COMPILADOR INTEL ICC EN EL SERVIDOR

Para utilizar el compilador `icc` *(C)* y `icpc` *(C++)* debemos de añadir al *path* el directorio del compilador de intel con el comando `source` y la arquitectura que queremos usar. Ejecutamos ese script en nuestro *shell* actual (*`source == .`*)

```
alumno167: $ source /opt/intel/bin/iccvars.sh intel64
```

Obtiene buen rendimiento en este servidor, es la versión 16.0.0 de Intel que es relativamente antigua, pero soporta el procesador que vamos a utilizar y optimiza muy bien. El compilador *GNU GCC* no lo vamos a usar.

CARACTERÍSTICAS DEL SERVIDOR

Para obtener el modelo del procesador y la cantidad de memoria que dispone utilizo el comando `lshw` aunque no puedo ver todos los detalles porque no soy *root* en este servidor.

```
alumno167: $ lshw
…
*-memory
          description: System memory
          physical id: 0
          size: 7864MiB
*-cpu
          product: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz
          vendor: Intel Corp.
          physical id: 1
          bus info: cpu@0
          size: 3297MHz
          capacity: 3297MHz
          width: 64 bits
…
```

Tiene un procesador con microarquitectura *Skylake*. El procesador es Intel®Core™ i5, nos indica que es gama media, en concreto es de 6º generación (i5-6400). Para obtener el resto de los detalles accedo a la página web https://ark.intel.com/content/www/us/en/ark.html.

Se trata de un procesador con 4 cores, con frecuencia de media de 2.70GHz y frecuencia máxima 3.3GHz (cuando se está ejecutando con un solo core y está en modo turbo) pero va variando en función de la carga de procesador, la temperatura y algunos otros factores.

Tiene una litografía de 14 nanómetros, es bastante moderno y tiene cada core dos cachés privadas con 32KiB (de datos y otra de instrucciones de parecido tamaño) de caché L1. Además, cada core también tiene una caché de segundo nivel de 256KiB, y 6MiB de caché L3 compartida, con TDP de 65 W, y con ISA soportado de SSE4.1/4.2, AVX2. Estas instrucciones nos van a servir para vectorizar los programas que vamos a probar de forma que explotemos el paralelismo de datos de forma más eficiente.

En esta práctica es SO es irrelevante porque los programas que voy a ejecutar apenas lo utilizan ya que no realizan operaciones de entrada y salida. El servidor es compartido, multitarea y multiusuario. Es relevante porque cada vez que hagamos una prueba para medir el tiempo de ejecución de un programa debo tener en cuenta que es posible que otro usuario esté ejecutando algo. Los recursos del procesador se repartirán entre todos los usuarios que estén ejecutando y el tiempo de ejecución será mayor del que realmente es capaz de obtener el procesador. Por este motivo, hay que hacer varias pruebas y hacer la media eliminando algunos experimentos cuyo resultado sea claramente incorrecto.

Con el comando `w | wc` iré observando cuantos usuarios están conectados mientras hago mi práctica, en este ejemplo hay 11 usuarios conectados.

```
alumno167: $ w | wc
13     108     905
```

RENDIMIENTO TEÓRICO PICO DEL PROCESADOR

Para saber si el resultado obtenido es bueno calculo el **rendimiento teórico pico** para el procesador. Uso todos los núcleos posibles del procesador y unidades vectoriales.

$$\begin{aligned} Rendimiento_{pico}(GFLOP/s) &= (Flop/operation) \times (\#\ operations/instruction) \times (\#\ Instructions/cycle) \\ &\quad \times Freq.(GHz) \times (\#\ cores/socket) \times (\#\ sockets/node) \end{aligned}$$

$$\begin{aligned} Rendimiento_{pico}(SP) &= 2\ (FMA\ operations) \times 8\ (SP\ SIMD\ Lane) 2\ (\#VectorLanes(Add, Mul)) \times \\ &\quad \times 3{,}1\ (GHz) \times 4\ (\#\ cores) \\ &\sim 396{,}8\ GFLOP/sec \end{aligned}$$

Cuando se utiliza un solo core, la frecuencia es 3,3 GHz, pero cuando se están utilizando todas las unidades funcionales del procesador la frecuencia se reduce aproximadamente a 3,1 GHz.

Tiene un socket, cado socket tiene cuatro cores y cada core va a 3,1 GHz. El procesador, además puede ejecutar varias instrucciones por ciclo, es un procesador superescalar como la mayoría de los procesadores actuales. Las operaciones en coma flotante de simple precisión tienen dos unidades funcionales. Cada instrucción va a ser una instrucción vectorial porque utiliza AVX2 y trabaja ocho (256bits / 32 bits) datos a la vez. Como quiero obtener el máximo rendimiento utilizo la instrucción vectorial FMA.

ANCHO DE MEMORIA PICO TEÓRICO

El ancho de memoria pico, es decir, cuantos datos podemos transferir de una posición de memoria a otra por segundo.

$$Bandwidth_{pico} GB/sec = (\#\ Controlador\ Memoria) \times (\#\ Canales/Control.) \times (\#\ Bytes) \times Freq.-Bus-Mem\ (GT/s)$$

$$Bandwidth_{pico} = 1\ (Controlador\ Memoria) \times 1\ (Canales/Control.) \times 8\ (Bytes) \times 2{,}133\ (GT/s) \sim 17{,}1\ GB/sec$$

Tengo que suponer que estoy utilizando todos los accesos a memoria que puede analizar el procesador en paralelo, pidiendole al procesador que ejecute en paralelo todas las instrucciones de memoria que pueda para lo cual el procesador utilizara todos los controladores de memoria que tenga. Cada controlador puede ser capaz de servir varias peticiones a la vez porque tenga varios canales y cada petición de memoria podrá mover un número determinado de bytes que dependerá de la instrucción de memoria que esté utilizando.

Comando para obtener la frecuencia del bus de memoria, pero solo podré ver la información si soy *root*. Para obtener la cantidad de memoria que tiene instalada, encontramos la información en */proc/info*.

```
alumno167: $ lshw -class memory
alumno167: $ more /proc/meminfo
```

Estudiamos el rendimiento pico del procesador y la memoria para saber cuando estemos optimizando si hemos llegado a un punto satisfactorio o no, si lo hemos alcanzado ya no hace falta optimizar más.

MEDICIÓN DEL RENDIMIENTO PICO DEL PROCESADOR Y DE SU ANCHO DE BANDA A MEMORIA

Analizo el código de cada programa, el cual mide el tiempo y divide los gflops / tiempo para obtener el rendimiento pico del procesador donde se esta ejecutando.

1. helloflops1.c

Realiza operaciones en punto flotante de las más eficientes (FMA), el mayor número de operaciones en un bucle para ejercitar el procesador y tratar de medir su rendimiento. Para ello, trabaja con dos arrays que ocupan en memoria cada uno 1024*1024 = 1048576 bytes = 1Mbytes multiplicado por el tamaño de los elementos (float) = 1Mbytes * 4 = 4 Mbytes. Aunque tengan ese tamaño, el programa solo accede a las 64 primeras posiciones. Esto se debe a que no queremos fallos en caché, 256 bytes de cada array, y en total 512 bytes, como el tamaño de la línea de caché es de 64 (casi todos los procesadores son así) por tanto, son solamente 8 líneas de caché.

Le ha indicado al compilador que la dirección del primer elemento sea múltiplo de 64 = aligned 64, para que en cada línea de caché nos quepan 16 elementos, si no estuviera alineada cuando accediera a uno de los elementos del array podría ser que algún elemento estuviera en dos cachés distintas y sería más lento. Todos los accesos a memoria están alineados.

Mide el tiempo que tarda en ejecutarse el bucle anidado donde en cada iteración del bucle interno se realiza una operación de FMA, dos lecturas a memoria y una escritura a memoria. Para medir el rendimiento contamos cuantas operaciones hemos realizado (64 * 1000000000) y cada vez que se ha ejecutado esa línea se han realizado dos operaciones aritméticas en coma flotante (suma y multiplicación). Por último, divide los *gflops* entre el tiempo consumido.

```
alumno167: $ icc -Wall -O2 -xCORE-AVX2 -o helloflops1 helloflops1.c
```

Optimizamos con -02, -x se trata de un procesador que soporta la arquitectura CORE-AVX2 y que genere el ejecutable.

```
alumno167: $ ./helloflops1
GFlops =    128.000, Secs =      2.293, GFlops per sec =     55.822

alumno167: $ ./helloflops1
GFlops =    128.000, Secs =      2.251, GFlops per sec =     56.860

alumno167: $ ./helloflops1
GFlops =    128.000, Secs =      2.241, GFlops per sec =     57.110

alumno167: $ ./helloflops1
GFlops =    128.000, Secs =      2.557, GFlops per sec =     50.055

alumno167: $ ./helloflops1
GFlops =    128.000, Secs =      2.306, GFlops per sec =     55.512
```

$\text{Rendimiento}_{\text{helloflops1.c}}$ = (55.822+56.860+57.110+50.055+55.512)/5 = **55.0718 GFLOP/sec**

No obtenemos el rendimiento máximo porque solo hay un solo hilo, por lo cual solo estamos ejecutando un core, sería la cuarta parte, aun así, no usamos el 100. El compilador de Intel no es capaz de optimizar este programa de la forma óptima.

2. helloflops2.c

Diferencia entre este programa y el anterior es que usa los cuatro cores creando hilos con los *pragmas omp*. Automáticamente paraleliza los bucles y cada hilo ejecuta una parte de la caché, 64 cada uno para que no tengan en su caché información de otros hilos.

```
alumno167: $ icc -Wall -O2 -xCORE-AVX2 -qopenmp -o helloflops2 helloflops2.c
```

Para que reconozca los *pragmas* se añade *-qopenmp.*

```
alumno167: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      1.406, GFlops per sec =    364.100

alumno167: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      1.396, GFlops per sec =    366.659

alumno167: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      1.398, GFlops per sec =    366.358

alumno167: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      1.399, GFlops per sec =    365.994


alumno167: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      1.366, GFlops per sec =    374.754
```

Ahora el rendimiento si que es el esperado:

Rendimiento$_{helloflops2.c}$ = (364.100+366.659+366.358+ 365.994+ 374.754)/5 = **292.6222 GFLOP/sec**

Nos permite cambiar el número de hilos y ver como escala. Pero si ponemos más de 4 hilos el rendimiento no aumenta porque ya con ese número de hilos ha alcanzado el rendimiento pico.

```
alumno167: $ ./helloflops2 1
Initializing
Starting Compute on 1 threads
GFlops =    128.000, Secs =      1.295, GFlops per sec =     98.833

alumno167: $ ./helloflops2 2
Initializing
Starting Compute on 2 threads
GFlops =    256.000, Secs =      1.241, GFlops per sec =    206.227

alumno167: $ ./helloflops2 3
Initializing
Starting Compute on 3 threads
GFlops =    384.000, Secs =      1.253, GFlops per sec =    306.564

alumno167: $ ./helloflops2 4
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      1.304, GFlops per sec =    392.766

alumno167: $ ./helloflops2 8
Initializing
Starting Compute on 8 threads
GFlops =   1024.000, Secs =      2.688, GFlops per sec =    380.886
```

3. hellomem.c

En este programa medimos el rendimiento del sistema de memoria. Está hecho a partir de los programas anteriores y mide cuanto se tarda en copiar datos de un array a otro para ello se definen dos arrays de (1000*1000*64) de *real* (8 bytes) = 1Gbytes y utiliza todos los cores. Se copia varias veces para tener mayor precisión.

Se realizan dos operaciones de acceso a memoria, una escritura y una lectura y cada una de esas operaciones a memoria ha movido 8 bytes (*real*) en cada iteración el número de veces repetido. El ancho de banda se obtiene dividiendo el resultado anterior entre el tiempo transcurrido. Tarda bastante en ejecutarse.

```
alumno167: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     80.676, GBytes per sec =     12.693

alumno167: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     80.634, GBytes per sec =     12.699

alumno167: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     80.673, GBytes per sec =     12.693

alumno167: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     80.709, GBytes per sec =     12.688

alumno167: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     80.651, GBytes per sec =     12.697
```

$\text{Bandwidth}_{\text{hellomem.c}}$ = (12.693+12.699+12.693+ 12.688+ 12.697)/5 = **12.694 GB/sec**

4. copy.cc

Este programa es parecido al anterior, pero utiliza la librería *mkl* para medir el tiempo y está optimizado de forma que realiza el mínimo número de iteraciones y obtiene exactamente el mismo resultado, pero en menor tiempo. También, calcula la media y la desviación típica. Además, se compila con el comando *icpc* porque es un programa escrito en c++.

```
alumno167: $ ./copy
Benchmarking array copy.
Size of each array: 1.280 GB
    Platform: CPU
     Threads: 4
    Affinity: (null)
Trial    Time, s      Perf, GB/s
    1  4.241e-01           12.07 *
    2  4.283e-01           11.95 *
    3  4.212e-01           12.15
    4  4.306e-01           11.89
    5  4.251e-01           12.05
    6  4.247e-01           12.05
    7  4.343e-01           11.79
    8  4.294e-01           11.92
    9  4.266e-01           12.00
   10  4.239e-01           12.08
-----------------------------------------------------
Average performance:           11.99 +- 0.11 GB/s
-----------------------------------------------------
* - warm-up, not included in average
```

OPCIONES DE COMPILACIÓN

Soportan 4 niveles de optimización. Podemos ejecutarlo con distintas opciones cuando ejecutamos:

```
alumno167: $ ./helloflops1-00 // no optimiza
GFlops =    128.000, Secs =      2.262, GFlops per sec =    56.600

alumno167: $ ./helloflops1-02-noavx2 // optimiza pero no vectoriza con la misma eficiencia
GFlops =    128.000, Secs =      9.200, GFlops per sec =    13.913

alumno167: $ ./helloflops1-02-novec // no vectoriza, el rendimiento es menor
GFlops =    128.000, Secs =      29.208, GFlops per sec =    4.382
```

EVITAR LOS RIESGOS DE DATOS

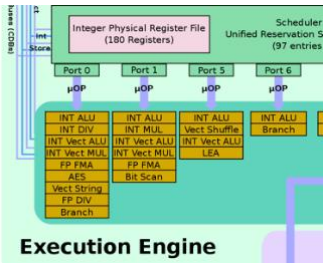
1. sumaflops1.c

Tiene un variable suma1 que está inicializada a 0 y se le va sumando uno, por defecto 64 veces, pero se puede pasar como segundo argumento del programa el número de repeticiones. El bucle interno además se ejecuta por defecto 10000000 o es el primer argumento del programa. Por defecto se ejecutará esa instrucción 640000000 veces, pero el resultado de suma1 que es una variable de tipo *float* que equivalen a 4 bytes tendrá el valor de 16777216.000. Esto problema no es de rango si no de precisión, cuando se usa aritmética en coma flotante si sumamos a un número muy grande en valor absoluto le sumamos un número muy pequeño en valor absoluto, el número pequeño se pierde. Se podría resolver cambiando el tipo de la variable a *double*. El rendimiento tampoco es el esperado porque hay dependencia entre instrucciones, aunque tengamos recursos disponibles en el procesador.

```
alumno167: $ ./sumaflops1
Starting Compute
GFlop =    0.640, Secs =    0.885, GFlop per sec =   0.723 (GFLOPs), suma1 = 16777216.000

alumno167: $ ./sumaflops1
Starting Compute
GFlop =      0.640, Secs =      0.897, GFlop per sec =      0.714 (GFLOPs), suma1 = 16777216.000

alumno167: $ ./sumaflops1
Starting Compute
GFlop =      0.640, Secs =      0.896, GFlop per sec =      0.714 (GFLOPs), suma1 = 16777216.000

alumno167: $ ./sumaflops1
Starting Compute
GFlop =      0.640, Secs =      0.870, GFlop per sec =      0.735 (GFLOPs), suma1 = 16777216.000

alumno167: $ ./sumaflops1
Starting Compute
GFlop =      0.640, Secs =      0.898, GFlop per sec =      0.713 (GFLOPs), suma1 = 16777216.000
```

$\text{Rendimiento}_{\text{sumaflops1.c}} = (0.723 + 0.714 + 0.714 + 0.735 + 0.713)/5 =$ **0.7198 GFLOP/sec**

Tiene dos unidades vectoriales en cada núcleo, para comprobar ese valor nos tenemos que fijar en el modelo del procesador en esta página web https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client)#Architecture se puede ver que hay dos unidades funcionales llamadas *FP FMA* de coma flotante que son capaces de hace multiplicaciones y sumas combinadas.



2. sumaflops2.c

Este programa es parecido al anterior, dentro del bucle en cada iteración se producen dos operaciones de coma flotante. Por tanto, realiza el doble de instrucciones, lo normal sería que tardara el doble de tiempo en ejecutarse, pero es el mismo porque se realiza en paralelo ya que tenemos dos unidades funcionales y no hay dependencia de datos.

```
alumno167: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      0.902, GFlop per sec =      1.419 (GFLOPs), suma1 = 33554432.000
alumno167: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      0.893, GFlop per sec =      1.434 (GFLOPs), suma1 = 33554432.000
alumno167: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      0.865, GFlop per sec =      1.479 (GFLOPs), suma1 = 33554432.000
alumno167: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      0.830, GFlop per sec =      1.542 (GFLOPs), suma1 = 33554432.000
alumno167: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      0.879, GFlop per sec =      1.457 (GFLOPs), suma1 = 33554432.000
```

Rendimiento$_{sumaflops2.c}$ = (1.419 + 1.434 + 1.479 + 1.542 + 1.457)/5 = **1.4662 GFLOP/sec**

3. sumaflops3.c

```
alumno167: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      0.866, GFlop per sec =      2.217 (GFLOPs), suma1 = 50331648.000

alumno167: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      0.906, GFlop per sec =      2.120 (GFLOPs), suma1 = 50331648.000

alumno167: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      0.895, GFlop per sec =      2.145 (GFLOPs), suma1 = 50331648.000

alumno167: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      0.845, GFlop per sec =      2.273 (GFLOPs), suma1 = 50331648.000

alumno167: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      0.865, GFlop per sec =      2.220 (GFLOPs), suma1 = 50331648.000
```

Rendimiento$_{sumaflops3.c}$ = (2.217 + 2.120 + 2.145 + 2.273 + 2.220)/5 = **2.195 GFLOP/sec**

4. sumaflops4.c

```
alumno167: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      0.786, GFlop per sec =      3.255 (GFLOPs), suma1 = 67108864.000

alumno167: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      0.843, GFlop per sec =      3.035 (GFLOPs), suma1 = 67108864.000

alumno167: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      0.887, GFlop per sec =      2.885 (GFLOPs), suma1 = 67108864.000

alumno167: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      0.796, GFlop per sec =      3.214 (GFLOPs), suma1 = 67108864.000

alumno167: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      0.897, GFlop per sec =      2.855 (GFLOPs), suma1 = 67108864.000
```

Rendimiento$_{sumaflops4.c}$ = (3.255 + 3.035 + 2.885 + 3.214 + 2.855)/5 = **3.0488 GFLOP/sec**

5. sumaflops5.c

```
alumno167: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      0.906, GFlop per sec =      3.532 (GFLOPs), suma1 = 83886080.000

alumno167: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      0.807, GFlop per sec =      3.965 (GFLOPs), suma1 = 83886080.000

alumno167: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      0.901, GFlop per sec =      3.550 (GFLOPs), suma1 = 83886080.000

alumno167: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      0.784, GFlop per sec =      4.080 (GFLOPs), suma1 = 83886080.000

alumno167: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      0.863, GFlop per sec =      3.709 (GFLOPs), suma1 = 83886080.000
```

Rendimiento$_{sumaflops5.c}$ = (3.532 + 3.965 + 3.550 + 4.080 + 3.709)/5 = **3.7672 GFLOP/sec**

6. sumaflops6.c

```
alumno167: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      0.837, GFlop per sec =      4.589 (GFLOPs), suma1 = 100663296.000

alumno167: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      0.896, GFlop per sec =      4.284 (GFLOPs), suma1 = 100663296.000

alumno167: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      0.856, GFlop per sec =      4.484 (GFLOPs), suma1 = 100663296.000

alumno167: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      0.896, GFlop per sec =      4.286 (GFLOPs), suma1 = 100663296.000

alumno167: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      0.888, GFlop per sec =      4.326 (GFLOPs), suma1 = 100663296.000
```

Rendimiento$_{sumaflops6.c}$ = (4.589 + 4.284 + 4.484 + 4.286 + 4.326)/5 = **4.614 GFLOP/sec**

7. sumaflops7.c

```
alumno167: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      0.895, GFlop per sec =      5.007 (GFLOPs), suma1 = 117440512.000

alumno167: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      0.900, GFlop per sec =      4.977 (GFLOPs), suma1 = 117440512.000

alumno167: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      0.873, GFlop per sec =      5.129 (GFLOPs), suma1 = 117440512.000

alumno167: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      0.852, GFlop per sec =      5.260 (GFLOPs), suma1 = 117440512.000

alumno167: $ ./sumaflops7
Starting Compute
```

```
GFlop =      4.480, Secs =      0.897, GFlop per sec =      4.992 (GFLOPs), suma1 = 117440512.000
```

Rendimiento$_{sumaflops7.c}$ = (5.007 + 4.977 + 5.129 + 5.260 + 4.992)/5 = **5.073 GFLOP/sec**

8. sumaflops8.c

```
alumno167: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      0.886, GFlop per sec =      5.780 (GFLOPs), suma1 = 134217728.000

alumno167: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      0.777, GFlop per sec =      6.591 (GFLOPs), suma1 = 134217728.000

alumno167: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      0.842, GFlop per sec =      6.079 (GFLOPs), suma1 = 134217728.000

alumno167: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      0.865, GFlop per sec =      5.922 (GFLOPs), suma1 = 134217728.000

alumno167: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      0.878, GFlop per sec =      5.829 (GFLOPs), suma1 = 134217728.000
```

Rendimiento$_{sumaflops8.c}$ = (5.780 + 6.591 + 6.079 + 5.922 + 5.829)/5 = **6.0402 GFLOP/sec**

9. sumaflops9.c

```
alumno167: $ ./sumaflops9
Starting Compute
GFlop = 5.760, Secs = 0.968, GFlop per sec = 5.953 (GFLOPs), suma1 = 134217728.000, suma2 =
16777216.000

alumno167: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      0.995, GFlop per sec =      5.791 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000

alumno167: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      0.995, GFlop per sec =      5.790 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000

alumno167: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      0.890, GFlop per sec =      6.470 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000

alumno167: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      0.877, GFlop per sec =      6.569 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000
```

Rendimiento$_{sumaflops9.c}$ = (5.953 + 5.791 + 5.790 + 6.470 + 6.569)/5 = **6.1146 GFLOP/sec**

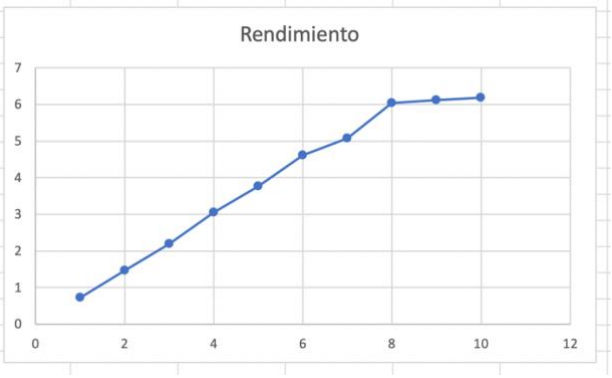
10. sumaflops10.c

```
alumno167: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      1.053, GFlop per sec =      6.076 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000
alumno167: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      1.045, GFlop per sec =      6.125 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000
```

```
alumno167: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      0.976, GFlop per sec =      6.555 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000

alumno167: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      1.070, GFlop per sec =      5.980 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000

alumno167: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      1.036, GFlop per sec =      6.180 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000
```

Rendimiento$_{sumaflops2.c}$ = (6.076 + 6.125 + 6.555 + 5.980 + 6.180)/5 = **6.1832 GFLOP/sec**

Sumaflops	Rendimiento
1	0,7198
2	1,4662
3	2,1950
4	3,0488
5	3,7672
6	4,6140
7	5,0730
8	6,0402
9	6,1146
10	6,1832



Puedo concluir con que hay dos unidades funcionales que tienen 4 de latencia porque a partir del programa sumaflops8.c el rendimiento empieza a ser constante.

BIBLIOTECAS HPC

Es un programa que permite generar ficheros con una matriz.

```
alumno167: $ icpc -Wall -02 -xCORE-aVX2 -o gen.c gen
alumno167: $ ./gen Mb 2048 2048
```

1. MMultSeq2D.c

Hay dependencia de datos y hay que leer de memoria. El cambio de orden de los bucles *for* influye porque en uno se accede a la matriz por columnas y el otro por filas.
En el primero programa se obtiene el rendimiento desde dos bucles distintos, en la teoría el primero debería tener mejor rendimiento por la variable temporal, sin embargo, en la práctica el compilador genera un código parecido. Solo se usa un core por tanto el 25% del procesador. Cada acceso a un elemento de la matriz implica dos accesos a memoria.

2. MMult2Seq2D.c

Es igual que el programa anterior, pero resuelve el anterior problema de dos accesos a memoria por cada elemento y por tanto es un poco mejor el rendimiento. Para mejorar este programa se debería de paralelizar añadiendo un *pragma* y obtenemos el doble del rendimiento, pero todavía no es el rendimiento pico.

3. MMultmkl2D.c

Reserva las matrices de la misma manera que el anterior programa y para realizar la multiplicación llama directamente al procedimiento de la biblioteca *MKL*. El rendimiento de esta versión llega a 207.99 GFLOPS/s

CUESTIONES

1. Detalla las características relevantes del ordenador elegido (microarquitectura de CPU y memoria).

Para obtener el modelo del procesador y la cantidad de memoria que dispone utilizo el comando lshw.

```
epgt: $ sudo lshw
…
*-memory
         description: System memory
         physical id: 0
         size: 16GiB
*-cache:0
         description: L1 cache
         physical id: 1b
         slot: L1 Cache
         size: 128KiB
         capacity: 128KiB
         capabilities: synchronous internal write-back unified
         configuration: level=1
    *-cache:1
         description: L2 cache
         physical id: 1c
         slot: L2 Cache
         size: 512KiB
         capacity: 512KiB
         capabilities: synchronous internal write-back unified
         configuration: level=2
    *-cache:2
         description: L3 cache
         physical id: 1d
         slot: L3 Cache
         size: 4MiB
         capacity: 4MiB
         capabilities: synchronous internal write-back unified
         configuration: level=3
   description: CPU
         product: Intel(R) Core(TM) i7-7660U CPU @ 2.50GHz
         vendor: Intel Corp.
         physical id: 1e
         bus info: cpu@0
         version: Intel(R) Core(TM) i7-7660U CPU @ 2.50GHz
         serial: To Be Filled By O.E.M.
         slot: U3E1
         size: 1816MHz
         capacity: 4GHz
         width: 64 bits
         clock: 25MHz
         capabilities: lm fpu fpu_exception wp vme de pse tsc msr pae
mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxs
r sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp x86-64 constant_tsc
art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid ap
erfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdb
g fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadli
ne_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fau
lt epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpri
ority ept vpid ept_ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms
invpcid rtm mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec x
getbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hw
p_epp md_clear flush_l1d cpufreq
         configuration: cores=2 enabledcores=2 threads=4
…
```

Tiene un procesador con microarquitectura Products formerly Kaby Lake. El procesador es Intel®Core™ i7, nos indica que es gama alta, en concreto es de 7º generación (i7-7660U). Para obtener el resto de los detalles accedo a la página web https://ark.intel.com/content/www/us/en/ark/products/97537/intel-core-i7-7660u-processor-4m-cache-up-to-4-00-ghz.html?wapkw=i7-7660U.

Se trata de un procesador con 2 cores, con frecuencia de media de 2.50GHz y frecuencia máxima 4GHz (cuando se está ejecutando con un solo core y está en modo turbo) pero va variando en función de la carga de procesador, la temperatura y algunos otros factores.

Tiene una litografía de 14 nanómetros, tiene una caché L1 de 128KiB, L2 de 512KiB y la caché L3 compartida de 4MiB, con TDP de 15W, y con ISA soportado de SSE4.1/4.2, AVX2. Estas instrucciones nos van a servir para vectorizar los programas que vamos a probar de forma que explotemos el paralelismo de datos de forma más eficiente. El tamaño de la memoria es de 16GiB.

En esta práctica es SO es irrelevante porque los programas que voy a ejecutar apenas lo utilizan ya que no realizan operaciones de entrada y salida.

```
epgt: $ hostnamectl
Static hostname: epgt
        Icon name: computer-laptop
          Chassis: laptop
       Machine ID: b1c71aac01be4e058b7e5aedffe79298
          Boot ID: 5b4164b37ef3471f9f3ec5751a0a3c1b
 Operating System: Ubuntu 19.10
           Kernel: Linux 5.3.0-64-generic
     Architecture: x86-64
```

2. ¿Cuál es el rendimiento pico que puede alcanzar teóricamente usando coma flotante de simple precisión (en GFLOPS)?

Para saber si el resultado obtenido es bueno calculo el rendimiento teórico pico para el procesador. Uso todos los núcleos posibles del procesador y unidades vectoriales.

$$Rendimiento_{pico}(GFLOP/s) = (Flop/operation) \times (\#\ operations/instruction) \times (\#\ Instructions/cycle) \times Freq.(GHz) \times (\#\ cores/socket) \times (\#\ sockets/node)$$

Cuando se utiliza un solo core, la frecuencia es 4 GHz, pero cuando se están utilizando todas las unidades funcionales del procesador la frecuencia se reduce aproximadamente a 3.75 GHz.

Tiene un socket, cada socket tiene dos cores y cada core va a 3,75 GHz. El procesador, además puede ejecutar varias instrucciones por ciclo, es un procesador superescalar como la mayoría de los procesadores actuales. Las operaciones en coma flotante de simple precisión tienen dos unidades funcionales (https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake#Sockets.2FPlatform). Cada instrucción va a ser una instrucción vectorial porque utiliza AVX2 y trabaja ocho (256bits / 32 bits) datos a la vez. Como quiero obtener el máximo rendimiento utilizo la instrucción vectorial FMA.

Rendimiento$_{pico}$(SP) = 2 (FMA operations) x 8 (SP SIMD lane)x 2(VectorLanes(Add,Mul))x 3,75 (GHz) x 2 (cores) = **240 GFLOP/sec**

3. ¿Cuál es el ancho de memoria pico que puede alcanzar teóricamente?

El ancho de memoria pico, es decir, cuantos datos podemos transferir de una posición de memoria a otra por segundo.

$$Bandwidth_{pico} GB/sec = (\#\ Controlador\ Memoria) \times (\#\ Canales/Control.) \times (\#\ Bytes) \times Freq.-Bus-Mem\ (GT/s)$$

Bandwidth$_{pico}$ = 1 (Controlador Memoria) x 1 (Canales/Control) x 16 (Bytes) x 2,133 (GT/s) = **34,13 GB/sec**

Tiene una memoria RAM de 16 GiB, tiene un único controlador de memoria, que maneja un canal de acceso a memoria con capacidad para poner 2 bancos de memoria DIMM, con un ancho de banda de 64 bits (8 bytes), siendo la memoria instalada un único DIMM de 16 GiB de 2133MHz.

Comando para obtener la frecuencia del bus de memoria, pero solo podré ver la información si soy *root*. Para obtener la cantidad de memoria que tiene instalada, encontramos la información en */proc/info.*

```
epgt: $ lshw -class memory
*-memory
      description: System Memory
      physical id: 0
      slot: System board or motherboard
      size: 16GiB
    *-bank:0
         description: SODIMM LPDDR3 Synchronous 2133 MHz (0,5 ns)
         product: MT52L1G32D4PG-093
         vendor: Micron Technology
         physical id: 0
         serial: 0x00000000
         slot: DIMM0
         size: 8GiB
         width: 64 bits
         clock: 2133MHz (0.5ns)
    *-bank:1
         description: SODIMM LPDDR3 Synchronous 2133 MHz (0,5 ns)
         product: MT52L1G32D4PG-093
         vendor: Micron Technology
         physical id: 1
         serial: 0x00000000
         slot: DIMM0
         size: 8GiB
         width: 64 bits
         clock: 2133MHz (0.5ns)
*-memory UNCLAIMED
      description: Memory controller
      product: Sunrise Point-LP PMC
      vendor: Intel Corporation
      physical id: 1f.2
      bus info: pci@0000:00:1f.2
      version: 21
      width: 32 bits
      clock: 33MHz (30.3ns)
      configuration: latency=0
      resources: memory:92824000-92827fff
epgt: $ more /proc/meminfo
```

4. Mide el rendimiento pico en coma flotante de simple precisión del ordenador elegido utilizando los benchmarks helloflops1 y helloflops2. Explica los resultados obtenidos.

```
epgt: $ gcc -Wall -O3 -march=native helloflops1.c -o helloflops1

epgt: $ gcc -Wall -O3 -march=native -fopenmp helloflops2.c -o helloflops2
```

Ejecutando el programa **helloflops1.c** explicado anteriormente en el informe, no obtengo el rendimiento máximo porque solo hay un hilo, por lo tanto, solo esta ejecutándose un core de los dos que tiene.

Rendimiento$_{\text{helloflops1.c}}$ = (126.489 + 125.948 + 126.442 + 126.461 + 126.316)/5 = **126.3312 GFLOP/sec**

```
epgt: $ ./helloflops1
Initializing
Starting Compute
GFlops =    128.000, Secs =      1.012, GFlops per sec =    126.489

epgt: $ ./helloflops1
Initializing
Starting Compute
GFlops =    128.000, Secs =      1.016, GFlops per sec =    125.948

epgt: $ ./helloflops1
Initializing
Starting Compute
GFlops =    128.000, Secs =      1.012, GFlops per sec =    126.442
```

```
epgt: $ ./helloflops1
Initializing
Starting Compute
GFlops =    128.000, Secs =      1.012, GFlops per sec =    126.461

epgt: $ ./helloflops1
Initializing
Starting Compute
GFlops =    128.000, Secs =      1.013, GFlops per sec =    126.316
```

Ejecutando el programa **helloflops2.c** explicado anteriormente en el informe, al tener más hilos (utilizando 4 aunque tiene 2 cores), automáticamente paraleliza el código y obtiene un rendimiento mucho más optimo (alrededor del 90% del rendimiento máximo), pero no llega a ser el pico.

Rendimiento$_{\text{helloflops2.c}}$ = (171.211 + 170.753 + 170.366 + 172.224 + 173.337) /5 = **171.5782 GFLOP/sec**

```
epgt: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      2.990, GFlops per sec =    171.211

epgt: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      2.998, GFlops per sec =    170.753

epgt: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      3.004, GFlops per sec =    170.459

epgt: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      3.005, GFlops per sec =    170.366

epgt: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      2.973, GFlops per sec =    172.224

epgt: $ ./helloflops2
Initializing
Starting Compute on 4 threads
GFlops =    512.000, Secs =      2.954, GFlops per sec =    173.337
```

5. Mide el ancho de banda pico de memoria del ordenador elegido utilizando el benchmark hellomem. Explica el resultado obtenido.

Ejecutando el programa **hellomem.c** explicado anteriormente en el informe, no obtengo el ancho de banda pico de memoria máximo porque se aproxima. Se encuentra en el entorno del 60-70 % del valor teórico que había calculado anteriormente.

Bandwidth$_{\text{hellomem.c}}$ = (25.135 +25.765 +23.370 + 23.998 + 23.443)/5 = **24.3422 GB/sec**

```
epgt: $ gcc -Wall -O3 -march=native -fopenmp hellomem.c -o hellomem


epgt: $./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     40.739, GBytes per sec =     25.135

epgt: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     39.744, GBytes per sec =     25.765
```

```
epgt: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     43.817, GBytes per sec =     23.370

epgt: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     42.671, GBytes per sec =     23.998

epgt: $ ./hellomem
Initializing
Starting BW Test on 4 threads
Gbytes =   1024.000, Secs =     43.681, GBytes per sec =     23.443
```

6. Usando los benchmarks sumaflopsX, calcula la latencia de las sumas en simple precisión del ordenador elegido y trata de determinar de cuántas unidades funcionales dispone cada core para la suma en simple precisión.

Rendimiento$_{\texttt{sumaflops1.c}}$ = (0.437 + 0.440 + 0.436 + 0.441 + 0.441) /5 = **0.439 GFLOP/sec**

Rendimiento$_{\texttt{sumaflops2.c}}$ = (0.880 + 0.881 + 0.880 + 0.880 + 0.881) /5 = **0.8804 GFLOP/sec**

Rendimiento$_{\texttt{sumaflops3.c}}$ = (1.318 + 1.322 + 1.322 + 1.321 + 1.318) /5 = **1.3202 GFLOP/sec**

Rendimiento$_{\texttt{sumaflops4.c}}$ = (1.757 + 1.760 + 1.762 + 1.759 + 1.761) /5 = **1.7598 GFLOP/sec**

Rendimiento$_{\texttt{sumaflops5.c}}$ = (2.198 + 2.198 + 2.203 + 2.202 + 2.199) /5 = **2.2 GFLOP/sec**

Rendimiento$_{\texttt{sumaflops6.c}}$ = (2.634 + 2.601 + 2.642 + 2.633 + 2.631) /5 = **2.6282 GFLOP/sec**

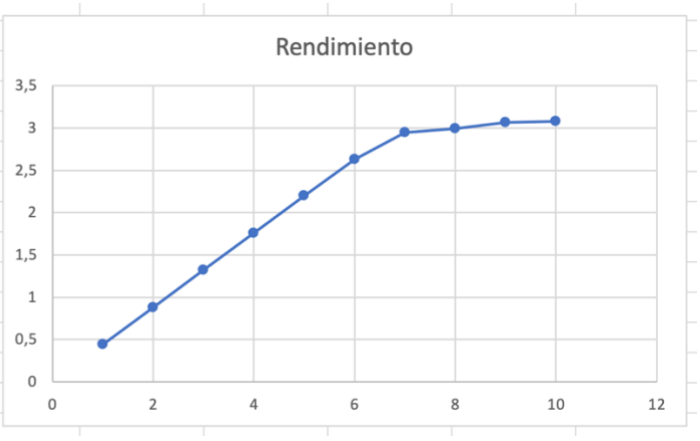
Rendimiento$_{\texttt{sumaflops7.c}}$ = (2.954+ 2.912 + 2.960 + 2.957 + 2.951) /5 = **2.9468 GFLOP/sec**

Rendimiento$_{\texttt{sumaflops8.c}}$ = (3.078 + 2.985 + 2.878 + 2.946 + 3.081) /5 = **2.9936 GFLOP/sec**

Rendimiento$_{\texttt{sumaflops9.c}}$ = (3.069 + 3.059+ 3.070 + 3.058 + 3.065) /5 = **3.0642 GFLOP/sec**

Rendimiento$_{\texttt{sumaflops10.c}}$ = (3.099+ 3.089 + 3.092+ 3.047) /4 = **3.08175 GFLOP/sec**

Sumaflops	Rendimiento
1	0,439
2	0,8804
3	1,3202
4	1,7598
5	2,2000
6	2,6282
7	2,9468
8	2,9936
9	3,0642
10	3,0818



Puedo concluir con que hay dos unidades funcionales que tienen 4 de latencia porque a partir del programa sumaflops8.c el rendimiento empieza a ser constante. Son dos unidades funcionales porque lo he consultado antes en la página web https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake#Sockets.2FPlatform.

```
epgt: $ ./sumaflops1
Starting Compute
GFlop = 0.640, Secs = 1.466, GFlop per sec = 0.437 (GFLOPs), suma1 = 16777216.000

epgt: $ ./sumaflops1
Starting Compute
GFlop =      0.640, Secs =      1.456, GFlop per sec =      0.440 (GFLOPs), suma1 = 16777216.000

epgt: $ ./sumaflops1
Starting Compute
GFlop =      0.640, Secs =      1.467, GFlop per sec =      0.436 (GFLOPs), suma1 = 16777216.000

epgt: $ ./sumaflops1
Starting Compute
GFlop =      0.640, Secs =      1.451, GFlop per sec =      0.441 (GFLOPs), suma1 = 16777216.000

epgt: $ ./sumaflops1
Starting Compute
GFlop =      0.640, Secs =      1.453, GFlop per sec =      0.441 (GFLOPs), suma1 = 16777216.000

epgt: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      1.454, GFlop per sec =      0.880 (GFLOPs), suma1 = 33554432.000

epgt: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      1.452, GFlop per sec =      0.881 (GFLOPs), suma1 = 33554432.000

epgt: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      1.455, GFlop per sec =      0.880 (GFLOPs), suma1 = 33554432.000

epgt: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      1.454, GFlop per sec =      0.880 (GFLOPs), suma1 = 33554432.000

epgt: $ ./sumaflops2
Starting Compute
GFlop =      1.280, Secs =      1.453, GFlop per sec =      0.881 (GFLOPs), suma1 = 33554432.000

epgt: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      1.457, GFlop per sec =      1.318 (GFLOPs), suma1 = 50331648.000

epgt: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      1.453, GFlop per sec =      1.322 (GFLOPs), suma1 = 50331648.000

epgt: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      1.452, GFlop per sec =      1.322 (GFLOPs), suma1 = 50331648.000

epgt: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      1.453, GFlop per sec =      1.321 (GFLOPs), suma1 = 50331648.000

epgt: $ ./sumaflops3
Starting Compute
GFlop =      1.920, Secs =      1.457, GFlop per sec =      1.318 (GFLOPs), suma1 = 50331648.000

epgt: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      1.457, GFlop per sec =      1.757 (GFLOPs), suma1 = 67108864.000

epgt: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      1.454, GFlop per sec =      1.760 (GFLOPs), suma1 = 67108864.000

epgt: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      1.453, GFlop per sec =      1.762 (GFLOPs), suma1 = 67108864.000

epgt: $ ./sumaflops4
```

```
Starting Compute
GFlop =      2.560, Secs =      1.455, GFlop per sec =      1.759 (GFLOPs), suma1 = 67108864.000

epgt: $ ./sumaflops4
Starting Compute
GFlop =      2.560, Secs =      1.454, GFlop per sec =      1.761 (GFLOPs), suma1 = 67108864.000

epgt: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      1.456, GFlop per sec =      2.198 (GFLOPs), suma1 = 83886080.000

epgt: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      1.456, GFlop per sec =      2.198 (GFLOPs), suma1 = 83886080.000
epgt: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      1.452, GFlop per sec =      2.203 (GFLOPs), suma1 = 83886080.000

epgt: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      1.453, GFlop per sec =      2.202 (GFLOPs), suma1 = 83886080.000

epgt: $ ./sumaflops5
Starting Compute
GFlop =      3.200, Secs =      1.455, GFlop per sec =      2.199 (GFLOPs), suma1 = 83886080.000

epgt: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      1.458, GFlop per sec =      2.634 (GFLOPs), suma1 = 100663296.000

epgt: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      1.476, GFlop per sec =      2.601 (GFLOPs), suma1 = 100663296.000

epgt: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      1.454, GFlop per sec =      2.642 (GFLOPs), suma1 = 100663296.000

epgt: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      1.458, GFlop per sec =      2.633 (GFLOPs), suma1 = 100663296.000

epgt: $ ./sumaflops6
Starting Compute
GFlop =      3.840, Secs =      1.460, GFlop per sec =      2.631 (GFLOPs), suma1 = 100663296.000

epgt: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      1.517, GFlop per sec =      2.954 (GFLOPs), suma1 = 117440512.000

epgt: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      1.539, GFlop per sec =      2.912 (GFLOPs), suma1 = 117440512.000

epgt: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      1.513, GFlop per sec =      2.960 (GFLOPs), suma1 = 117440512.000

epgt: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      1.515, GFlop per sec =      2.957 (GFLOPs), suma1 = 117440512.000

epgt: $ ./sumaflops7
Starting Compute
GFlop =      4.480, Secs =      1.518, GFlop per sec =      2.951 (GFLOPs), suma1 = 117440512.000

epgt: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      1.664, GFlop per sec =      3.078 (GFLOPs), suma1 = 134217728.000

epgt: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      1.716, GFlop per sec =      2.985 (GFLOPs), suma1 = 134217728.000
```

```
epgt: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      1.779, GFlop per sec =      2.878 (GFLOPs), suma1 = 134217728.000

epgt: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      1.738, GFlop per sec =      2.946 (GFLOPs), suma1 = 134217728.000

epgt: $ ./sumaflops8
Starting Compute
GFlop =      5.120, Secs =      1.662, GFlop per sec =      3.081 (GFLOPs), suma1 = 134217728.000

epgt: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      1.879, GFlop per sec =      3.065 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000

epgt: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      1.877, GFlop per sec =      3.069 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000

epgt: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      1.883, GFlop per sec =      3.059 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000
epgt: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      1.876, GFlop per sec =      3.070 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000

epgt: $ ./sumaflops9
Starting Compute
GFlop =      5.760, Secs =      1.884, GFlop per sec =      3.058 (GFLOPs), suma1 = 134217728.000,
suma2 = 16777216.000

epgt: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      2.065, GFlop per sec =      3.099 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000

epgt: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      2.072, GFlop per sec =      3.089 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000

epgt: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      2.078, GFlop per sec =      3.080 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000

epgt: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =     13.183, GFlop per sec =      0.485 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000

epgt: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      2.070, GFlop per sec =      3.092 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000

epgt: $ ./sumaflops10
Starting Compute
GFlop =      6.400, Secs =      2.100, GFlop per sec =      3.047 (GFLOPs), suma1 = 134217728.000,
suma2 = 33554432.000
```

OPINIÓN

Es un trabajo que requiere tiempo. El mayor aspecto positivo es que la práctica está muy guiada por los profesores y documentada por el boletín. Ha sido muy útil tener una máquina igual para todos porque así tenemos un modelo para después poder hacer los mismos ejercicios asentando los conocimientos con nuestro ordenador personal.