2º curso / 2º cuatr.

Grado Ingeniería Informática

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 0. Entorno de programación

Estudiante (nombre y apellidos):
Grupo de prácticas y profesor de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Parte I. Ejercicios basados en los ejemplos del seminario práctico

Crear el directorio con nombre `bp0` en atcgrid y en el PC local.

NOTA: En las prácticas se usa slurm como gestor de colas. Consideraciones a tener en cuenta:

- Slurm está configurado para asignar recursos a los procesos (llamados *tasks* en slurm) a nivel de core físico. Esto significa que por defecto slurm asigna un core a un proceso, para asignar más de uno se debe usar con `sbatch/srun` la opción `--cpus-per-task`.
- En slurm, por defecto, `cpu` se refiere a cores lógicos (ej. en la opción `--cpus-per-task`), si no se quieren usar cores lógicos hay que añadir la opción `--hint=nomultithread` a `sbatch/srun`.
- Para asegurar que solo se crea un proceso hay que incluir `-n1` en `sbatch/srun`.
- Para que no se ejecute más de un proceso en un nodo de atcgrid hay que usar `--exclusive` con `sbatch/srun` (se recomienda no utilizarlo en los `srun` dentro de un script).
- Los `srun` dentro de un script heredan las opciones fijadas en el `sbatch` que se usa para enviar el script a la cola slurm.

1. Ejecutar `lscpu` en el PC y en un nodo de cómputo de atcgrid. (Crear directorio `ejer1`)

(a) Mostrar con capturas de pantalla el resultado de estas ejecuciones.

RESPUESTA:

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC] 2020-02-25 martes
$lscpu
Arquitectura:                    x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:              Little Endian
CPU(s):                          8
Lista de la(s) CPU(s) en línea:      0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:          4
«Socket(s)»                      1
Modo(s) NUMA:                    1
ID de fabricante:                GenuineIntel
Familia de CPU:                  6
Modelo:                          158
Nombre del modelo:               Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Revisión:                        9
CPU MHz:                         3146.219
CPU MHz máx.:                    3800,0000
CPU MHz mín.:                    800,0000
BogoMIPS:                        5616.00
Virtualización:                  VT-x
Caché L1d:                       32K
Caché L1i:                       32K
Caché L2:                        256K
Caché L3:                        6144K
CPU(s) del nodo NUMA 0:          0-7
```

```
[RaulCastroMoreno b2estudiante4@atcgrid:~] 2020-02-20 jueves
$srun -p ac lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Model name:            Intel(R) Xeon(R) CPU           E5645  @ 2.40GHz
Stepping:              2
CPU MHz:               1600.000
CPU max MHz:           2401,0000
CPU min MHz:           1600,0000
BogoMIPS:              4800.38
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              12288K
NUMA node0 CPU(s):     0-5,12-17
NUMA node1 CPU(s):     6-11,18-23
```

(b) ¿Cuántos cores físicos y cuántos cores lógicos tienen los nodos de cómputo de atcgrid y el PC? Razonar las respuestas

RESPUESTA:

Atcgrid: Tiene 2 sockets * 6 cores por socket = 12 cores fisicos

12 cores fisicos * 2 Thread por core = 24 cores lógicos

PC: Tiene 1 socket * 4 cores por socket = 4 cores fisicos

4 cores fisicos * 2 Thread por core = 8 cores lógicos

2. Compilar y ejecutar en el PC el código `HelloOMP.c` del seminario (recordar que se debe usar un directorio independiente para cada ejercicio dentro de `bp0` que contenga todo lo utilizado, implementado o generado durante el desarrollo del mismo, para el presente ejercicio el directorio sería `ejer2`, como se indica en las normas de prácticas).

(a) Adjuntar capturas de pantalla que muestren la compilación y ejecución en el PC.

RESPUESTA:**(Le he añadido un salto de linea a HelloOMP.c para que se vea mas claro)**

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC] 2020-02-25 martes
$gcc -O2 -fopenmp -o HelloOMP HelloOMP.c
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC] 2020-02-25 martes
$./HelloOMP
(6:!!!Hello world!!!)
(7:!!!Hello world!!!)
(4:!!!Hello world!!!)
(1:!!!Hello world!!!)
(3:!!!Hello world!!!)
(5:!!!Hello world!!!)
(2:!!!Hello world!!!)
(0:!!!Hello world!!!)
```

(b) Justificar el número de "Hello world" que se imprimen en pantalla teniendo en cuenta la salida que devuelve `lscpu`.

RESPUESTA:

Esto es debido a que tengo 8 cpu's lógicos y cada uno de ellos ejecuta el proceso, por lo que se me imprimen 8 veces "Hello World" (0-7).

3. Copiar el ejecutable de `HelloOMP.c` que ha generado anteriormente y que se encuentra en el directorio `ejer2` del PC al directorio `ejer2` de su home en el *front-end* de atcgrid. Ejecutar este código en un nodo de cómputo de atcgrid a través de cola ac del gestor de colas (no use ningún *script*) utilizando directamente en línea de comandos:

(a) `srun -p ac –n1 --cpus-per-task=12 --hint=nomultithread HelloOMP`

Adjuntar capturas de pantalla que muestren el envío a la cola de la ejecución y el resultado de esta ejecución tal y como la devuelve el gestor de colas.

RESPUESTA:

```
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp0/ejer3] 2020-02-25 martes
$srun -p ac -n1 --cpus-per-task=12 --hint=nomultithread HelloOMP
(2:!!!Hello world!!!)
(3:!!!Hello world!!!)
(11:!!!Hello world!!!)
(7:!!!Hello world!!!)
(4:!!!Hello world!!!)
(10:!!!Hello world!!!)
(8:!!!Hello world!!!)
(1:!!!Hello world!!!)
(0:!!!Hello world!!!)
(6:!!!Hello world!!!)
(5:!!!Hello world!!!)
(9:!!!Hello world!!!)
```

(b) `srun -p ac -n1 --cpus-per-task=24 HelloOMP`

Adjuntar capturas de pantalla que muestren el envío a la cola de la ejecución y el resultado de esta ejecución tal y como la devuelve el gestor de colas.

```
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp0/ejer3] 2020-02-25 martes
srun -p ac -n1 --cpus-per-task=24 HelloOMP
(0:!!!Hello world!!!)
(5:!!!Hello world!!!)
(7:!!!Hello world!!!)
(18:!!!Hello world!!!)
(8:!!!Hello world!!!)
(13:!!!Hello world!!!)
(20:!!!Hello world!!!)
(10:!!!Hello world!!!)
(16:!!!Hello world!!!)
(6:!!!Hello world!!!)
(1:!!!Hello world!!!)
(9:!!!Hello world!!!)
(2:!!!Hello world!!!)
(15:!!!Hello world!!!)
(3:!!!Hello world!!!)
(22:!!!Hello world!!!)
(4:!!!Hello world!!!)
(21:!!!Hello world!!!)
(12:!!!Hello world!!!)
(14:!!!Hello world!!!)
(19:!!!Hello world!!!)
(17:!!!Hello world!!!)
(11:!!!Hello world!!!)
(23:!!!Hello world!!!)
```

RESPUESTA:

(c) `srun -p ac –n1 HelloOMP`

Adjuntar capturas de pantalla que muestren el envío a la cola de la ejecución y el resultado de esta ejecución tal y como la devuelve el gestor de colas.

RESPUESTA:

```
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp0/ejer3] 2020-02-25 martes
$srun -p ac -n1 HelloOMP
(0:!!!Hello world!!!)
(1:!!!Hello world!!!)
```

(d) ¿Qué orden `srun` usaría para que HelloOMP utilice los 12 cores físicos de un nodo de cómputo de atcgrid (se debe imprimir un único mensaje desde cada uno de ellos, en total, 12)?

RESPUESTA:

`srun -p ac -n1 --cpus-per-task=12 --hint=nomultithread HelloOMP`

Es básicamente la orden del 3.(a , donde con cpu’s per task decimos que queremos 12, y con --hint=nomultithread decimos que solo use cores físicos y no lógicos.

4. Modificar en su PC `HelloOMP.c` para que se imprima “world” en un `printf` distinto al usado para “Hello”, en ambos `printf` se debe imprimir el identificador del thread que escribe en pantalla. Nombrar al código resultante `HelloOMP2.c`. Compilar este nuevo código en el PC y ejecutarlo. Copiar el fichero ejecutable resultante al front-end de atcgrid (directorio `ejer4`). Ejecutar el código en un nodo de cómputo de atcgrid usando el script `script_helloomp.sh` del seminario (el nombre del ejecutable en el script debe ser `HelloOMP2`).

(a) Utilizar: `sbatch -p ac –n1 --cpus-per-task=12 --hint=nomultithread script_helloomp.sh`. Adjuntar capturas de pantalla que muestren el nuevo código, la compilación, el envío a la cola de la ejecución y el resultado de esta ejecución tal y como la devuelve el gestor de colas.

RESPUESTA:

Captura de Nuevo Código de HelloOMP2:

```
/* Compilar con:
gcc -O2 -fopenmp -o HelloOMP HelloOMP.c
*/

#include <stdio.h>
#include <omp.h>

int main(void) {

#pragma omp parallel
        printf("(%d:!!!Hello)\n",
                omp_get_thread_num());
#pragma omp parallel
        printf("(%d:world!!!)\n",
                omp_get_thread_num());
return(0);
}
```

Captura de la compilación:

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC] 2020-02-25 martes
$gcc -O2 -fopenmp -o HelloOMP2 HelloOMP2.c
```

Captura del envío a la cola de ejecución:

```
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp0/ejer4] 2020-02-25 martes
$sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread script_helloomp.sh
Submitted batch job 9934
```

Capturas del resultado de la ejecucion como la devuelve el gestor de colas:

```
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp0/ejer4] 2020-02-25 martes
$cat slurm-9934.out
Id. usuario del trabajo: b2estudiante4
Id. del trabajo: 9934
Nombre del trabajo especificado por usuario: helloOMP
Directorio de trabajo (en el que se ejecuta el script): /home/b2estudiante4/bp0/ejer4
Cola: ac
Nodo que ejecuta este trabajo:atcgrid
Nº de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
CPUs por nodo: 24

 1. Ejecución helloOMP una vez sin cambiar nº de threads (valor por defecto):

(0:!!!Hello)
(8:!!!Hello)
(2:!!!Hello)
(9:!!!Hello)
(7:!!!Hello)
(3:!!!Hello)
(1:!!!Hello)
(5:!!!Hello)
(11:!!!Hello)
(4:!!!Hello)
(6:!!!Hello)
(10:!!!Hello)
(2:world!!!)
(8:world!!!)
(9:world!!!)
(7:world!!!)
(10:world!!!)
(4:world!!!)
(1:world!!!)
(3:world!!!)
(6:world!!!)
(11:world!!!)
(5:world!!!)
(0:world!!!)

 2. Ejecución helloOMP varias veces con distinto nº de threads:


  - Para 12 threads:
```

```
2. Ejecución helloOMP varias veces con distinto nº de threads:


  - Para 12 threads:
(3:!!!Hello)
(9:!!!Hello)
(11:!!!Hello)
(4:!!!Hello)
(1:!!!Hello)
(0:!!!Hello)
(8:!!!Hello)
(2:!!!Hello)
(10:!!!Hello)
(6:!!!Hello)
(7:!!!Hello)
(5:!!!Hello)
(3:world!!!)
(4:world!!!)
(6:world!!!)
(2:world!!!)
(9:world!!!)
(10:world!!!)
(0:world!!!)
(1:world!!!)
(11:world!!!)
(7:world!!!)
(5:world!!!)
(8:world!!!)

  - Para 6 threads:
(4:!!!Hello)
(0:!!!Hello)
(3:!!!Hello)
(5:!!!Hello)
(1:!!!Hello)
(2:!!!Hello)
(4:world!!!)
(3:world!!!)
(1:world!!!)
(0:world!!!)
(5:world!!!)
(2:world!!!)

  - Para 3 threads:
(1:!!!Hello)
(0:!!!Hello)
(2:!!!Hello)
(1:world!!!)
(0:world!!!)
(2:world!!!)

  - Para 1 threads:
(0:!!!Hello)
(0:world!!!)
```

(b) ¿Qué nodo de cómputo de atcgrid ha ejecutado el script? Explicar cómo ha obtenido esta información.

RESPUESTA:

Ha usado el nodo atcgrid1. Esta información la he obtenido de la salida que me da el script helloomp.sh, donde en el apartado : "Nodos asignados al trabajo" especifica que nodos de cómputo ha ejecutado el script.

NOTA: Utilizar siempre con `sbatch` las opciones `–n1` y `--cpus-per-task`, `--exclusive` y, para usar cores físicos y no lógicos, no olvide incluir `--hint=nomultithread`. Utilizar siempre con `srun`, si lo usa fuera de un script, las opciones `–n1` y `--cpus-per-task` y, para usar cores físicos y no lógicos, no olvide incluir `--hint=nomultithread`. Recordar que los `srun` dentro de un script heredan las opciones utilizadas en el `sbatch` que se usa para enviar el script a la cola slurm. Se recomienda usar `sbatch` en lugar de `srun`

para enviar trabajos a ejecutar a través slurm porque éste último deja bloqueada la ventana hasta que termina la ejecución, mientras que usando `sbatch` la ejecución se realiza en segundo plano.

Parte II. Resto de ejercicios

5. Generar en el PC el ejecutable del código fuente C del Listado 1 para vectores locales (para ello antes de compilar debe descomentar la definición de `VECTOR_LOCAL` y comentar las definiciones de `VECTOR_GLOBAL` y `VECTOR_DYNAMIC`). El comentario inicial del código muestra la orden para compilar (siempre hay que usar `-O2` al compilar como se indica en las normas de prácticas). Incorporar volcados de pantalla que demuestren la compilación y la ejecución correcta del código en el PC (leer lo indicado al respecto en las normas de prácticas).

RESPUESTA:

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp0/ejer5] 2020-02-25 martes
$gcc -O2 SumaVectoresC.c -o SumaVectores -lrt
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:32: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has type ‘long unsigned int’ [-Wformat=]
   printf("Tama◆o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                            ~^
                            %lu
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp0/ejer5] 2020-02-25 martes
$./SumaVectores 5
Tama◆o Vectores:5 (4 B)
Tiempo:0.000000080      / Tama◆o Vectores:5
/ V1[0]+V2[0]=V3[0](0.500000+0.500000=1.000000) /
/ V1[1]+V2[1]=V3[1](0.600000+0.400000=1.000000) /
/ V1[2]+V2[2]=V3[2](0.700000+0.300000=1.000000) /
/ V1[3]+V2[3]=V3[3](0.800000+0.200000=1.000000) /
/ V1[4]+V2[4]=V3[4](0.900000+0.100000=1.000000) /
```

6. En el código del Listado 1 se utiliza la función `clock_gettime()` para obtener el tiempo de ejecución del trozo de código que calcula la suma de vectores. El código se imprime la variable `ncgt`,
(a) ¿qué contiene esta variable?
RESPUESTA:

Es una variable de tipo double la cual contiene el tiempo de ejecución. Este lo calcula haciendo la resta entre el instante final y el instante inicial de la ejecucion.

(b) ¿en qué estructura de datos devuelve `clock_gettime()` la información de tiempo (indicar el tipo de estructura de datos, describir la estructura de datos, e indicar los tipos de datos que usa)?

RESPUESTA:

La estructura de datos en la cual se devuelve la información de tiempo es timespec. Esta consta de tv_sec de tipo time_t la cual almacena segundos enteros y tv_nsec de tipo long, donde se almacenan los nanosegundos.

(c) ¿qué información devuelve exactamente la función `clock_gettime()`en la estructura de datos descrita en el apartado (b)? ¿qué representan los valores numéricos que devuelve?

RESPUESTA:

La función clock_gettime () obtiene la hora actual del reloj especificada por el primer parámetro, en este caso, CLOCK_REALTIME y lo pone en el búfer señalado por el segundo parámetro, en el caso de nuestro código sería en cgt1 el inicial y en cgt2 el final.

Puede devolver 2 valores numéricos : 0 si hay éxito y 1 si da error.

7. Rellenar una tabla como la Tabla 1 en una hoja de cálculo con los tiempos de ejecución del código del Listado 1 para vectores locales, globales y dinámicos. Obtener estos resultados usando scripts (partir del script que hay en el seminario). Debe haber una tabla para atcgrid y otra para su PC en la hoja de cálculo. En la columna "Bytes de un vector" hay que poner el total de bytes reservado para un vector. (NOTA: Se recomienda usar en la hoja de cálculo el mismo separador para decimales que usan los códigos al imprimir. Este separador se puede modificar en la hoja de cálculo.)

RESPUESTA:

PC

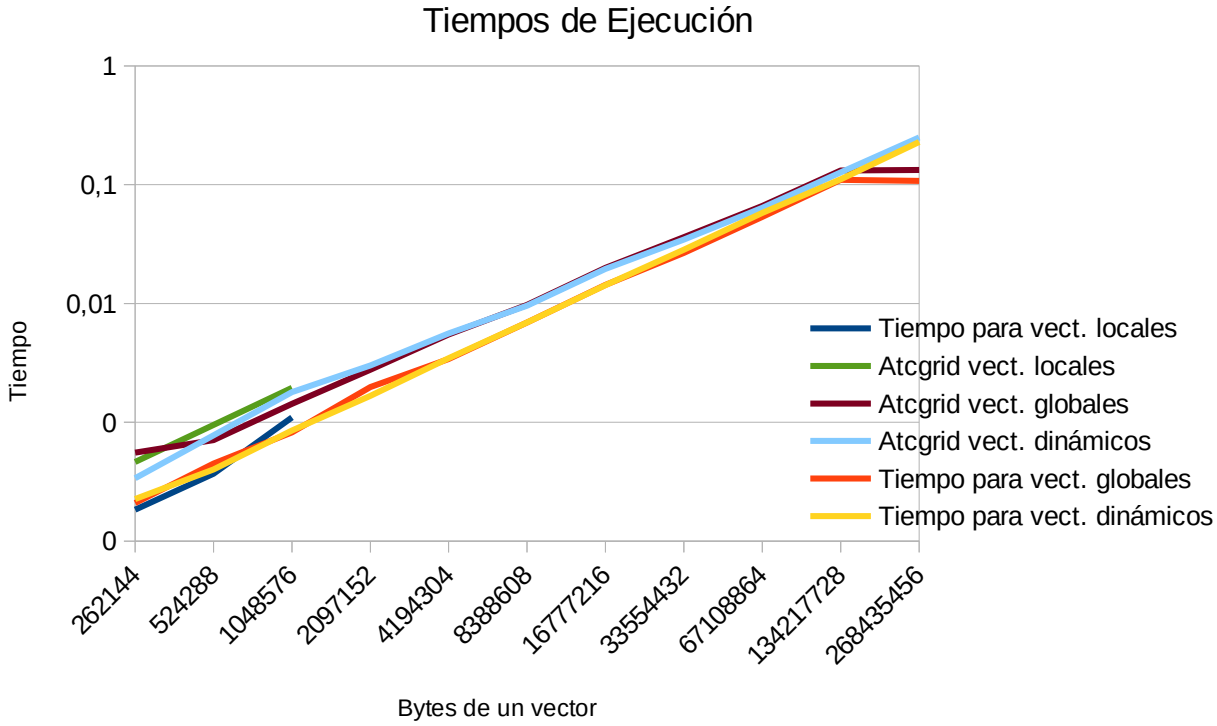
Nº de Componentes	Bytes de un vector	Tiempo para vect. locales	Tiempo para vect. globales	Tiempo para vect. dinámicos
65536	262144	0,000183868	0,000209362	0,000226991
131072	524288	0,00036879	0,000450346	0,00040236
262144	1048576	0,001098819	0,00082307	0,000852985
524288	2097152	Segmentation fault	0,001982827	0,001669821
1048576	4194304	Segmentation fault	0,00341371	0,003473747
2097152	8388608	Segmentation fault	0,006913366	0,00693757
4194304	16777216	Segmentation fault	0,014329695	0,014257481
8388608	33554432	Segmentation fault	0,026739439	0,028314613
16777216	67108864	Segmentation fault	0,053540797	0,057988635
33554432	134217728	Segmentation fault	0,110150686	0,111235242
67108864	268435456	Segmentation fault	0,107620641	0,228787097

Atcgrid

Nº de Componentes	Bytes de un vector	Atcgrid vect. locales	Atcgrid vect. globales	Atcgrid vect. dinámicos
65536	262144	0,000464936	0,000558053	0,000338746
131072	524288	0,000953944	0,000709263	0,000779501
262144	1048576	0,001947317	0,001429279	0,001793606
524288	2097152	Segmentation fault	0,002773627	0,003010333
1048576	4194304	Segmentation fault	0,005495197	0,005590089
2097152	8388608	Segmentation fault	0,009719305	0,009586746
4194304	16777216	Segmentation fault	0,019944515	0,019521349
8388608	33554432	Segmentation fault	0,036228609	0,034515235
16777216	67108864	Segmentation fault	0,066454553	0,063983819
33554432	134217728	Segmentation fault	0,131416843	0,127243822
67108864	268435456	Segmentation fault	0,132831234	0,249897756

1. Con ayuda de la hoja de cálculo representar **en una misma gráfica** los tiempos de ejecución obtenidos en atcgrid y en su PC para vectores locales, globales y dinámicos (eje y) en función del tamaño en bytes de un vector (por tanto, los valores de la segunda columna de la tabla, que están en escala logarítmica, deben estar en el eje x). Utilizar escala logarítmica en el eje de ordenadas (eje y). ¿Hay diferencias en los tiempos de ejecución?

RESPUESTA:



Si hay diferencia que se puede apreciar mejor en la gráfica, mi PC es más rapido que atcgrid pero es una diferencia de tiempos muy pequeña.

2. **(a)** Cuando se usan vectores locales, ¿se obtiene error para alguno de los tamaños?, ¿a qué cree que es debido lo que ocurre? (Incorporar volcados de pantalla como se indica en las normas de prácticas)

RESPUESTA:

Sí, a partir 262144, da segmentation fault tanto en mi PC como en atgrid, esto es debido a que los vecotres locales se almacenan en la pila (stack) y se llena en ese valor, por lo que al intentar meterle el siguiente se desborda.

```
 2. Ejecución SumaVectores con vectores locales:

Tama●o Vectores:65536 (4 B)
Tiempo:0.000464936      / Tama●o Vectores:65536       / V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535](13107.100000+0.100000=13107.200000) /
Tama●o Vectores:131072 (4 B)
Tiempo:0.000953944      / Tama●o Vectores:131072      / V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[131071](26214.300000+0.100000=26214.400000) /
Tama●o Vectores:262144 (4 B)
Tiempo:0.001947317      / Tama●o Vectores:262144      / V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) / / V1[262143]+V2[262143]=V3[262143](52428.700000+0.100000=52428.800000) /
srun: error: atcgrid1: task 0: Segmentation fault (core dumped)
srun: error: atcgrid1: task 0: Segmentation fault (core dumped)
srun: error: atcgrid1: task 0: Segmentation fault (core dumped)
srun: error: atcgrid1: task 0: Segmentation fault (core dumped)
srun: error: atcgrid1: task 0: Segmentation fault (core dumped)
srun: error: atcgrid1: task 0: Segmentation fault (core dumped)
srun: error: atcgrid1: task 0: Segmentation fault (core dumped)
srun: error: atcgrid1: task 0: Segmentation fault (core dumped)
```

(b) Cuando se usan vectores globales, ¿se obtiene error para alguno de los tamaños?, ¿a qué cree que es debido lo que ocurre? (Incorporar volcados de pantalla como se indica en las normas de prácticas)

RESPUESTA:

No, no se obtienen ningun error, el problema viene que el máximo numero de comoponentes viene limitado en el código por la variable global que tiene. Entonces cuando metemos un valor mayor al guardado en esa variable global, el iguala ese numero al de la variable, por lo que no nos deja probar con tamaños mayores a ese valor.

```
2. Ejecución SumaVectores con vectores globales:

Tama±o Vectores:65536 (4 B)
Tiempo:0.000558053       / Tama±o Vectores:65536       / V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535](13107.100000+0.100000=13107.200000) /
Tama±o Vectores:131072 (4 B)
Tiempo:0.000709263       / Tama±o Vectores:131072      / V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[131071](26214.300000+0.100000=26214.400000) /
Tama±o Vectores:262144 (4 B)
Tiempo:0.001429279       / Tama±o Vectores:262144      / V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) / / V1[262143]+V2[262143]=V3[262143](52428.700000+0.100000=52428.800000) /
Tama±o Vectores:524288 (4 B)
Tiempo:0.002773627       / Tama±o Vectores:524288      / V1[0]+V2[0]=V3[0](52428.800000+52428.800000=104857.600000) / / V1[524287]+V2[524287]=V3[524287](104857.500000+0.100000=104857.600000) /
Tama±o Vectores:1048576 (4 B)
Tiempo:0.005495197       / Tama±o Vectores:1048576     / V1[0]+V2[0]=V3[0](104857.600000+104857.600000=209715.200000) / / V1[1048575]+V2[1048575]=V3[1048575](209715.100000+0.100000=209715.200000) /
Tama±o Vectores:2097152 (4 B)
Tiempo:0.009719305       / Tama±o Vectores:2097152     / V1[0]+V2[0]=V3[0](209715.200000+209715.200000=419430.400000) / / V1[2097151]+V2[2097151]=V3[2097151](419430.300000+0.100000=419430.400000) /
Tama±o Vectores:4194304 (4 B)
Tiempo:0.019944515       / Tama±o Vectores:4194304     / V1[0]+V2[0]=V3[0](419430.400000+419430.400000=838860.800000) / / V1[4194303]+V2[4194303]=V3[4194303](838860.700000+0.100000=838860.800000) /
Tama±o Vectores:8388608 (4 B)
Tiempo:0.036228609       / Tama±o Vectores:8388608     / V1[0]+V2[0]=V3[0](838860.800000+838860.800000=1677721.600000) / / V1[8388607]+V2[8388607]=V3[8388607](1677721.500000+0.100000=1677721.600000) /
Tama±o Vectores:16777216 (4 B)
Tiempo:0.066454553       / Tama±o Vectores:16777216    / V1[0]+V2[0]=V3[0](1677721.600000+1677721.600000=3355443.200000) / / V1[16777215]+V2[16777215]=V3[16777215](3355443.100000+0.100000=3355443.200000)
 /
Tama±o Vectores:33554432 (4 B)
Tiempo:0.131416843       / Tama±o Vectores:33554432    / V1[0]+V2[0]=V3[0](3355443.200000+3355443.200000=6710886.400000) / / V1[33554431]+V2[33554431]=V3[33554431](6710886.300000+0.100000=6710886.400000)
 /
Tama±o Vectores:67108864 (4 B)
Tiempo:0.132831234       / Tama±o Vectores:33554432    / V1[0]+V2[0]=V3[0](3355443.200000+3355443.200000=6710886.400000) / / V1[33554431]+V2[33554431]=V3[33554431](6710886.300000+0.100000=6710886.400000)
 /
```

(c) Cuando se usan vectores dinámicos, ¿se obtiene error para alguno de los tamaños?, ¿a qué cree que es debido lo que ocurre? (Incorporar volcados de pantalla como se indica en las normas de prácticas)

RESPUESTA:

No, no se obtiene ningun error, funciona perfectamente.

```
2. Ejecución SumaVectores con vectores dinamicos:

Tama±o Vectores:65536 (4 B)
Tiempo:0.000338746       / Tama±o Vectores:65536       / V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535](13107.100000+0.100000=13107.200000) /
Tama±o Vectores:131072 (4 B)
Tiempo:0.000779501       / Tama±o Vectores:131072      / V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[131071](26214.300000+0.100000=26214.400000) /
Tama±o Vectores:262144 (4 B)
Tiempo:0.001793606       / Tama±o Vectores:262144      / V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) / / V1[262143]+V2[262143]=V3[262143](52428.700000+0.100000=52428.800000) /
Tama±o Vectores:524288 (4 B)
Tiempo:0.003019333       / Tama±o Vectores:524288      / V1[0]+V2[0]=V3[0](52428.800000+52428.800000=104857.600000) / / V1[524287]+V2[524287]=V3[524287](104857.500000+0.100000=104857.600000) /
Tama±o Vectores:1048576 (4 B)
Tiempo:0.005590089       / Tama±o Vectores:1048576     / V1[0]+V2[0]=V3[0](104857.600000+104857.600000=209715.200000) / / V1[1048575]+V2[1048575]=V3[1048575](209715.100000+0.100000=209715.200000) /
Tama±o Vectores:2097152 (4 B)
Tiempo:0.009586746       / Tama±o Vectores:2097152     / V1[0]+V2[0]=V3[0](209715.200000+209715.200000=419430.400000) / / V1[2097151]+V2[2097151]=V3[2097151](419430.300000+0.100000=419430.400000) /
Tama±o Vectores:4194304 (4 B)
Tiempo:0.019521349       / Tama±o Vectores:4194304     / V1[0]+V2[0]=V3[0](419430.400000+419430.400000=838860.800000) / / V1[4194303]+V2[4194303]=V3[4194303](838860.700000+0.100000=838860.800000) /
Tama±o Vectores:8388608 (4 B)
Tiempo:0.034515235       / Tama±o Vectores:8388608     / V1[0]+V2[0]=V3[0](838860.800000+838860.800000=1677721.600000) / / V1[8388607]+V2[8388607]=V3[8388607](1677721.500000+0.100000=1677721.600000) /
Tama±o Vectores:16777216 (4 B)
Tiempo:0.063983819       / Tama±o Vectores:16777216    / V1[0]+V2[0]=V3[0](1677721.600000+1677721.600000=3355443.200000) / / V1[16777215]+V2[16777215]=V3[16777215](3355443.100000+0.100000=3355443.200000)
 /
Tama±o Vectores:33554432 (4 B)
Tiempo:0.127243822       / Tama±o Vectores:33554432    / V1[0]+V2[0]=V3[0](3355443.200000+3355443.200000=6710886.400000) / / V1[33554431]+V2[33554431]=V3[33554431](6710886.300000+0.100000=6710886.400000)
 /
Tama±o Vectores:67108864 (4 B)
Tiempo:0.249897756       / Tama±o Vectores:67108864    / V1[0]+V2[0]=V3[0](6710886.400000+6710886.400000=13421772.800000) / / V1[67108863]+V2[67108863]=V3[67108863](13421772.700000+0.100000=13421772.8000
00) /
```

3. **(a)** ¿Cuál es el máximo valor que se puede almacenar en la variable N teniendo en cuenta su tipo? Razonar respuesta.

RESPUESTA:

N es un unsigned int, luego su sizeof es 4B, 4B son 32 bits. El máximo valor que podría almacenar es 2^{32} -1, es decir , 4294967295. El -1 es porque el 0 tambien se cuenta.

(b) Modificar el código fuente C (en el PC) para que el límite de los vectores cuando se declaran como variables globales sea igual al máximo número que se puede almacenar en la variable N y generar el ejecutable. ¿Qué ocurre? ¿A qué es debido? (Incorporar volcados de pantalla que muestren lo que ocurre)

RESPUESTA:

Da fallo de compilación, esto es debido a que el valor que hemos puesto es demasiado grande y seria necesario truncarla.

```
#include <stdlib.h>     // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>      // biblioteca donde se encuentra la función printf()
#include <time.h>       // biblioteca donde se encuentra la función clock_gettime()

//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL  // descomentar para que los vectores sean variables ...
                        // locales (si se supera el tamaño de la pila se ...
                        // generará el error "Violación de Segmento")
#define VECTOR_GLOBAL // descomentar para que los vectores sean variables ...
                        // globales (su longitud no estará limitada por el ...
                        // tamaño de la pila del programa)
//#define VECTOR_DYNAMIC        // descomentar para que los vectores sean variables ...
                        // dinámicas (memoria reutilizable durante la ejecución)

#ifdef VECTOR_GLOBAL
#define MAX 4294967295 //33554432       //=2^25
```

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp0/ejer7] 2020-02-28 viernes
$gcc  -O2  SumaVectoresC.c -o SumaVectores -lrt
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:32: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has type ‘long unsigned int’ [-Wformat=]
   printf("Tama◆o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                               ~^
                               %lu
/tmp/ccL8MDMf.o: En la función `main':
SumaVectoresC.c:(.text.startup+0x76): reubicación truncada para ajustar: R_X86_64_PC32 contra el símbolo `v2' definido en la sección COMMON en /tmp/cc
L8MDMf.o
SumaVectoresC.c:(.text.startup+0xc9): reubicación truncada para ajustar: R_X86_64_PC32 contra el símbolo `v3' definido en la sección COMMON en /tmp/cc
L8MDMf.o
collect2: error: ld returned 1 exit status
```

Entrega del trabajo

Leer lo indicado en las normas de prácticas sobre la entrega del trabajo del bloque práctico en SWAD.