2º curso / 2º cuatr.

Grado Ingeniería Informática

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 0. Entorno de programación

Estudiante (nombre y apellidos): Nikita Stetskiy
Grupo de prácticas y profesor de prácticas: C3
Fecha de entrega: 23-03-20
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

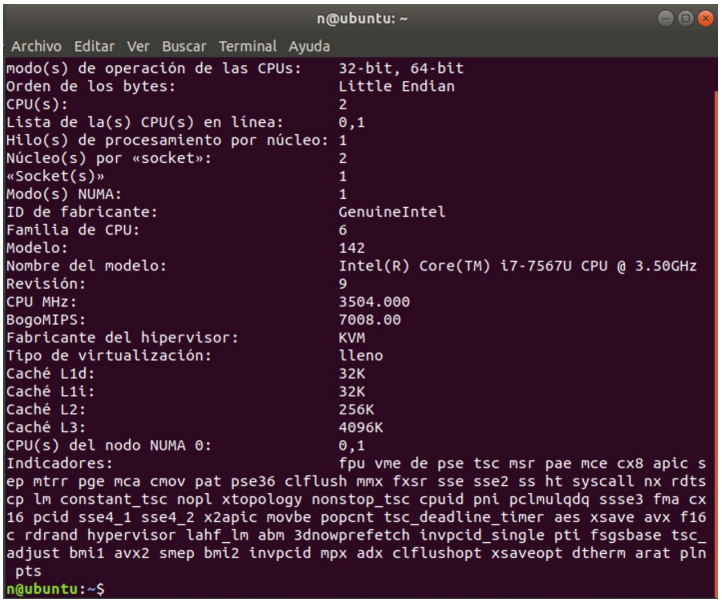
Crear el directorio con nombre bp0 en atcgrid y en el PC local.
NOTA: En las prácticas se usa slurm como gestor de colas. Consideraciones a tener en cuenta:

- 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:



```
[NikitaStetskiy c3estudiante26@atcgrid:~] 2020-03-22 domingo
$srun 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
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse
36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc
 arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni dtes64 mo
nitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 popcnt lahf_lm epb
ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid dtherm ida arat spec_ctrl intel_s
tibp flush_l1d
[NikitaStetskiy c3estudiante26@atcgrid:~] 2020-03-22 domingo
```

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

Atcgrid - Tiene 3 nodos de computo y dos procesadores por cada nodo, por lo tanto cada uno tiene 6 núcleos y 12 lógicos gracias a las dos hebras por core. En total 12 físicos y 24 lógicos por cada nodo.

PC - Tiene un procesador, pero tiene 2 núcleos y 2 lógicos, gracias a 1 hebra por núcleo. En total 2 físicos y 2 lógicos.

RESPUESTA:

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.

```
$g++-8 -fopenmp -O2 HelloOMP.c -o HelloOMP
[NikitaStetskiy nikitastetskiy@vpn-s241208:~/Desktop/COPY/nuevo_testamento1/AC/PRACTICAS/
bp0] 2020-03-22 domingo
$./HelloOMP
(0:!!!Hello world!!!)(2:!!!Hello world!!!)(1:!!!Hello world!!!)(3:!!!Hello world!!!)[Niki
taStetskiy nikitastetskiy@vpn-s241208:~/Desktop/COPY/nuevo_testamento1/AC/PRACTICAS/bp0]
2020-03-22 domingo
```

RESPUESTA:

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

RESPUESTA:

Hace tantos ya que depende de OMP_THREAD_LIMIT , el cual es el número de hebras haciendo la ejecución, es decir el printf (x:!!!Hello world!!!)

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:

```
Uploading HelloOMP to /home/c3estudiante26/bp0/ejer3/HelloOMP
HelloOMP                                      100% 8708    94.6KB/s   00:00
```

(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:

```
[NikitaStetskiy c3estudiante26@atcgrid:~/bp0/ejer3] 2020-03-22 domingo
$srun -p ac -n1 --cpus-per-task=12 --hint=nomultithread HelloOMP
(1:!!!Hello world!!!)(0:!!!Hello world!!!)(10:!!!Hello world!!!)(2:!!!Hello world!!!)(3:!!!H
ello world!!!)(4:!!!Hello world!!!)(7:!!!Hello world!!!)(5:!!!Hello world!!!)(9:!!!Hello wor
ld!!!)(11:!!!Hello world!!!)(6:!!!Hello world!!!)(8:!!!Hello world!!!)[NikitaStetskiy c3estu
diante26@atcgrid:~/bp0/ejer3] 2020-03-22 domingo
```

(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.

RESPUESTA:

```
[NikitaStetskiy c3estudiante26@atcgrid:~/bp0/ejer3] 2020-03-22 domingo
$srun -p ac -n1 --cpus-per-task=24 HelloOMP
(14:!!!Hello world!!!)(5:!!!Hello world!!!)(0:!!!Hello world!!!)(19:!!!Hello world!!!)(7:!!!
Hello world!!!)(11:!!!Hello world!!!)(17:!!!Hello world!!!)(13:!!!Hello world!!!)(21:!!!Hell
o world!!!)(4:!!!Hello world!!!)(9:!!!Hello world!!!)(15:!!!Hello world!!!)(10:!!!Hello worl
d!!!)(18:!!!Hello world!!!)(23:!!!Hello world!!!)(3:!!!Hello world!!!)(22:!!!Hello world!!!)
(8:!!!Hello world!!!)(6:!!!Hello world!!!)(12:!!!Hello world!!!)(2:!!!Hello world!!!)(1:!!!H
ello world!!!)(20:!!!Hello world!!!)(16:!!!Hello world!!!)[NikitaStetskiy c3estudiante26@atc
grid:~/bp0/ejer3] 2020-03-22 domingo
```

(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:

```
[NikitaStetskiy c3estudiante26@atcgrid:~/bp0/ejer3] 2020-03-22 domingo
$srun -p ac -n1 HelloOMP
(1:!!!Hello world!!!)(0:!!!Hello world!!!)[NikitaStetskiy c3estudiante26@atcgrid:~/bp0/ejer3
] 2020-03-22 domingo
```

(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:

Esta sería la orden:

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

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:

```
sftp> put Hello*
Uploading HelloOMP2 to /home/c3estudiante26/bp0/ejer4/HelloOMP2
HelloOMP2                                     100% 8708    42.3KB/s   00:00
Uploading HelloOMP2.c to /home/c3estudiante26/bp0/ejer4/HelloOMP2.c
HelloOMP2.c                                   100%  182     5.4KB/s   00:00
sftp> put script_helloomp.sh
Uploading script_helloomp.sh to /home/c3estudiante26/bp0/ejer4/script_helloomp.sh
script_helloomp.sh                            100% 1207    35.1KB/s   00:00
```

```c
#include <stdio.h>
#include <omp.h>
int main(void) {

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

}
```

```
HelloOMP2  HelloOMP2.c  script_helloomp2.sh
[NikitaStetskiy c3estudiante26@atcgrid:~/bp0/ejer4] 2020-03-22 domingo
$sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread script_helloomp2.sh
Submitted batch job 24513
[NikitaStetskiy c3estudiante26@atcgrid:~/bp0/ejer4] 2020-03-22 domingo
$cat slurm-24513.out
Id. usuario del trabajo: c3estudiante26
Id. del trabajo: 24513
Nombre del trabajo especificado por usuario: helloOMP
Directorio de trabajo (en el que se ejecuta el script):
/home/c3estudiante26/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):

(5:!!!Hello (10:!!!Hello (8:!!!Hello (2:!!!Hello (7:!!!Hello (4:!!!Hello (0:!!!Hello (11:!!!Hello (3:!!!Hello (9:
!!!Hello (6:!!!Hello (1:!!!Hello world!!!)
 2. Ejecución helloOMP varias veces con distinto no de threads:


  - Para 12 threads:
(2:!!!Hello (1:!!!Hello (0:!!!Hello (9:!!!Hello (5:!!!Hello (8:!!!Hello (10:!!!Hello (11:!!!Hello (7:!!!Hello (4:
!!!Hello (3:!!!Hello (6:!!!Hello world!!!)
  - Para 6 threads:
(0:!!!Hello (1:!!!Hello (2:!!!Hello (3:!!!Hello (5:!!!Hello (4:!!!Hello world!!!)
  - Para 3 threads:
(2:!!!Hello (1:!!!Hello (0:!!!Hello world!!!)
  - Para 1 threads:
(0:!!!Hello world!!!)[NikitaStetskiy c3estudiante26@atcgrid:~/bp0/ejer4] 2020-03-22 domingo
```

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

RESPUESTA:

El nodo se encuentra en $SLURM_JOB_NODELIST, en este caso es el atcgrid1

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:

```
$g++-8 -fopenmp -O2 SumaVectoresC2.c -o SumaVectoresC2
[NikitaStetskiy nikitastetskiy@vpn-s241208:~/Desktop/COPY/nuevo_testamento1/AC/PRACTICAS/
bp0] 2020-03-22 domingo
$./SumaVectoresC2 1234 4 56
Tiempo(seg.):0.000003000         / Tamaño Vectores:1234 / V1[0]+V2[0]=V3[0](123.400000+12
3.400000=246.800000) / / V1[1233]+V2[1233]=V3[1233](246.700000+0.100000=246.800000) /
[NikitaStetskiy nikitastetskiy@vpn-s241208:~/Desktop/COPY/nuevo_testamento1/AC/PRACTICAS/
bp0] 2020-03-22 domingo
```

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:

Tiempo: 0.000003000. La resta de las dos tomas de tiempo.

(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:

Es un struct de segundos y nanosegundos. Datos de tipo time_t y long.

(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:

Devuelve el tiempo de ejecución, clock_gettime() devuelve exactamente el instante en que nos encontramos actualmente.

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:

Tabla 1 . Copiar la tabla de la hoja de cálculo utilizada
PC LOCAL

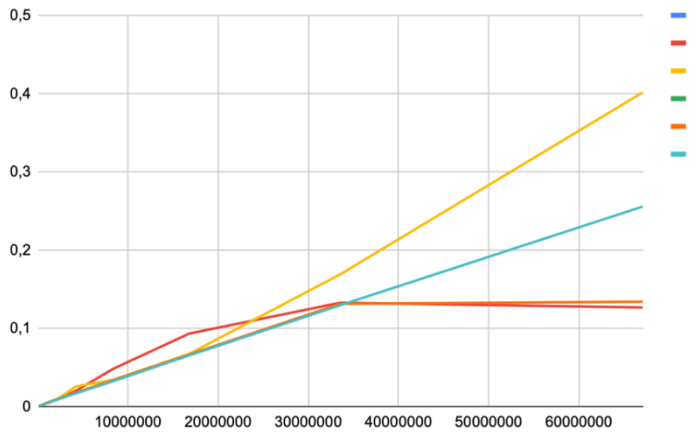
Nº de Componentes	Bytes de un vector	Tiempo para vect. locales	Tiempo para vect. globales	Tiempo para vect. dinámicos
65536	4	0.000268000	0.000305000	0.000254000
131072	4	0.000545000	0.000509000	0.000541000
262144	4	0.001070000	0.001441000	0.001147000
524288	4		0.002481000	0.002259000
1048576	4		0.004363000	0.004220000
2097152	4		0.008883000	0.008932000
4194304	4		0.020186000	0.025715000
8388608	4		0.048537000	0.034285000
16777216	4		0.093272000	0.067301000
33554432	4		0.132734000	0.169154000
67108864	4		0.126612000	0.401567000

ATCGRID

Nº de Componentes	Bytes de un vector	Tiempo para vect. locales	Tiempo para vect. globales	Tiempo para vect. dinámicos
65536	4	0.000347350	0.000553068	0.000353891
131072	4	0.000729218	0.000707976	0.000690712
262144	4	0.001715939	0.001458415	0.001363693
524288	4		0.002596810	0.002303558
1048576	4		0.005272029	0.004711992
2097152	4		0.009632141	0.008731386
4194304	4		0.017910770	0.016863930
8388608	4		0.034092677	0.032639006
16777216	4		0.066517364	0.065425251
33554432	4		0.131373431	0.129587369
67108864	4		0.134036288	0.255643984

8. 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:



Azul – PC LOCAL V. LOCAL

Rojo – PC LOCAL V. GLOBAL

Amarillo – PC LOCAL V. DINAMICO

Verde – ATCGRID V. LOCAL

Naranja – ATCGRID V. GLOBAL

Azul claro – ATCGRID V. DINAMICO

9. **(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: Si, ocurre con los vectores locales debido a que se acaba llenando la pila y por lo consiguiente da core

(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: Si, en el ultimo se ha truncado al tamaño especifico que tienen. Por eso el ultimo y el penúltimo son tan parecidos.

(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 ocurre

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

RESPUESTA: 4294967295, es el valor máximo para un entero con signo de 32 bits en informática. Por lo tanto, es el valor máximo para una variable declarada como un entero sin signo en muchos lenguajes de programación que se ejecutan en computadoras modernas. La presencia del valor puede reflejar un error, una condición de desbordamiento o un valor perdido.

(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:

```
[NikitaStetskiy nikitastetskiy@vpn-s241208:~/Desktop/COPY/nuevo_testamento1/AC/PRACTICAS/bp0] 2020-03-22 domingo
$g++-8 -fopenmp -O2 SumaVectoresC2.c -o SumaVectoresC2
final section layout:
    __TEXT/__text_startup addr=0x100000BB0, size=0x00000223, fileOffset=0x00000BB0, type=1
    __TEXT/__stubs addr=0x100000DD4, size=0x0000001E, fileOffset=0x00000DD4, type=28
    __TEXT/__stub_helper addr=0x100000DF4, size=0x00000042, fileOffset=0x00000DF4, type=32
    __TEXT/__cstring addr=0x100000E38, size=0x0000010D, fileOffset=0x00000E38, type=13
    __TEXT/__const addr=0x100000F48, size=0x00000010, fileOffset=0x00000F48, type=0
    __TEXT/__eh_frame addr=0x100000F58, size=0x000000A8, fileOffset=0x00000F58, type=19
    __DATA/__nl_symbol_ptr addr=0x100001000, size=0x00000010, fileOffset=0x00001000, type=29
    __DATA/__la_symbol_ptr addr=0x100001010, size=0x00000028, fileOffset=0x00001010, type=27
    __DATA/__pu_bss5 addr=0x100001038, size=0x00000000, fileOffset=0x00000000, type=25
    __DATA/__huge addr=0x100001040, size=0x17FFFFFF8, fileOffset=0x00000000, type=25
ld: 32-bit RIP relative reference out of range (34359739451 max is +/-2GB): from _main (0x100000BB0) to _v2 (0x900001040) in '_main' from
/var/folders/cj/cg78w8tx02b70gdnmdx8268m0000gq/T//ccGkleEU.o for architecture x86_64
collect2: error: ld returned 1 exit status
[NikitaStetskiy nikitastetskiy@vpn-s241208:~/Desktop/COPY/nuevo_testamento1/AC/PRACTICAS/bp0] 2020-03-22 domingo
```

Entrega del trabajo

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

Listado 1 . Código C que suma dos vectores

```
/* SumaVectoresC.c
 Suma de dos vectores: v3 = v1 + v2

 Para compilar usar (-lrt: real time library, no todas las versiones de gcc necesitan que se incluya -lrt):
      gcc  -O2 SumaVectores.c -o SumaVectores –lrt
      gcc  -O2 –S SumaVectores.c –lrt   //para generar el código ensamblador

 Para ejecutar use: SumaVectoresC longitud
*/
```

```
#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 33554432       //=2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char** argv){

  int i;
  struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

  //Leer argumento de entrada (nº de componentes del vector)
  if (argc<2){
    printf("Faltan nº componentes del vector\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  #ifdef VECTOR_LOCAL
  double v1[N], v2[N], v3[N];   // Tamaño variable local en tiempo de ejecución ...
                        // disponible en C a partir de actualización C99
  #endif
  #ifdef VECTOR_GLOBAL
  if (N>MAX) N=MAX;
  #endif
  #ifdef VECTOR_DYNAMIC
  double *v1, *v2, *v3;
  v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
  v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc devuelve NULL
  v3 = (double*) malloc(N*sizeof(double));
    if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
  }
  #endif

  //Inicializar vectores
  for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);
  //Calcular suma de vectores
  for(i=0; i<N; i++)
```

```
  v3[i] = v1[i] + v2[i];

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
     (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

//Imprimir resultado de la suma y el tiempo de ejecución
if (N<10) {
printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%lu\n",ncgt,N);
for(i=0; i<N; i++)
  printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
         i,i,i,v1[i],v2[i],v3[i]);
}
else
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /
         V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
         ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);


#ifdef VECTOR_DYNAMIC
free(v1); // libera el espacio reservado para v1
free(v2); // libera el espacio reservado para v2
free(v3); // libera el espacio reservado para v3
#endif
return 0;
}
```