2º curso / 2º cuatr.

Grado Ingeniería Informática

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 0. Entorno de programación

Estudiante (nombre y apellidos): Alejandro Molina Criado
Grupo de prácticas y profesor de prácticas: A2
(Christian Agustín Morillas Gutierrez)

Fecha de entrega: 02 / 03 / 2020
Fecha evaluación en clase: 05 / 03 / 2020

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:

Ejecución en un nodo de cómputo del clúster:

```
[a2estudiante13@atcgrid bp0]$ srun -p ac mkdir ejer1
[a2estudiante13@atcgrid bp0]$ ls
ejer1
[a2estudiante13@atcgrid bp0]$ 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
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 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 monitor 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_ct
rl intel_stibp flush_l1d
[a2estudiante13@atcgrid bp0]$
```

Ejecución en PC local:

```
alex@alex-CX62-6QD:~/bp0$ mkdir ejer1
alex@alex-CX62-6QD:~/bp0$ ls
ejer1
alex@alex-CX62-6QD:~/bp0$ lscpu
Arquitectura:                    x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:              Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          4
Lista de la(s) CPU(s) en línea:      0-3
Hilo(s) de procesamiento por núcleo: 1
Núcleo(s) por «socket»:          4
«Socket(s)»                      1
Modo(s) NUMA:                    1
ID de fabricante:                GenuineIntel
Familia de CPU:                  6
Modelo:                          94
Nombre del modelo:               Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz
Revisión:                        3
CPU MHz:                         1042.212
CPU MHz máx.:                    3200,0000
CPU MHz min.:                    800,0000
BogoMIPS:                        4599.93
Virtualización:                  VT-x
Caché L1d:                       128 KiB
Caché L1i:                       128 KiB
Caché L2:                        1 MiB
Caché L3:                        6 MiB
CPU(s) del nodo NUMA 0:          0-3
Vulnerability Itlb multihit:     KVM: Mitigation: Split huge pages
Vulnerability L1tf:              Mitigation; PTE Inversion; VMX conditional cache flushes, SMT disabled
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT disabled
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, IBPB conditional, IBRS_FW, STIBP disabled, RSB filling
Vulnerability Tsx async abort:   Mitigation; Clear CPU buffers; SMT disabled
Indicadores:                     fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht
                                 tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cp
                                 uid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x
                                 2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid
                                 _single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 hle avx2 s
                                 mep bmi2 erms invpcid rtm mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat
                                 pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear flush_l1d
```

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

RESPUESTA:

En el clúster atcgrid cada nodo de cómputo tiene dos sockets (es decir , podemos introducir dos microprocesadores) con 6 cores físicos y 12 lógicos (ya que hay 2 hilos por core). Por tanto , cada nodo de cómputo tiene **12 cores físicos** (6 cores fisicos * 2 sockets) y **24 cores lógicos** (12 cores logicos * 2 sockets)

En mi PC tengo un socket con **4 cores físicos** , y por la información mostrada por lscpu , tengo un solo hilo por core , por tanto tengo **4 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:

```
alex@alex-CX62-6QD:~/bp0/ejer2$ gcc -O2 -fopenmp -o helloOMP helloOMP.c
alex@alex-CX62-6QD:~/bp0/ejer2$ ./helloOMP
(1:Hola mundo!)(0:Hola mundo!)(2:Hola mundo!)(3:Hola mundo!)
alex@alex-CX62-6QD:~/bp0/ejer2$
```

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

RESPUESTA: Aparecen 4 "Hola mundo", que es el número de cores lógicos que tiene mi máquina , como hemos visto con el comando lscpu. Junto a cada mensaje aparece el número de thread que lo está ejecutando (omp_get_thread_num())

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:

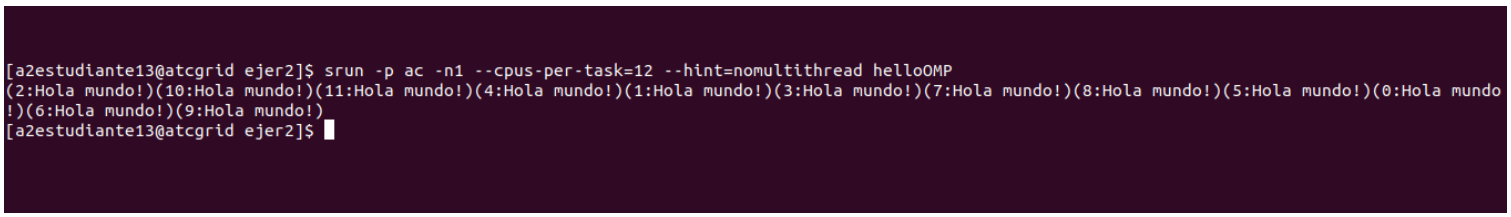
(a) `srun -p ac –n1 --cpus-per-task=12 --hint=nomultithread helloOMP`

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.s

RESPUESTA:

Envío a la cola de ejecución

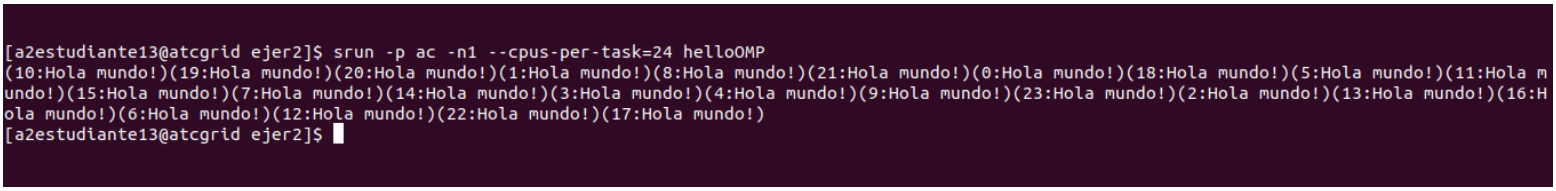
Resultado de la ejecución



(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:



(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:

```
[a2estudiante13@atcgrid ejer2]$ srun -p ac -n1 helloOMP
(1:Hola mundo!)(0:Hola mundo!)
[a2estudiante13@atcgrid ejer2]$
```

(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)?

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

(con -hint=nomultithread forzamos a que no se usen hebras extra en cada core fisico)

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:

Nuevo código (helloOMP2.c)

ejer4 > C helloOMP2.c > main(void)

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

int main(void){

    #pragma omp parallel
    printf("(%d:Hola )" , omp_get_thread_num());
    #pragma omp parallel
    printf("(%d:mundo!)" , omp_get_thread_num());
    return (0);
}

```

```
[a2estudiante13@atcgrid ejer4]$ ls
helloOMP2  script_helloomp.sh
[a2estudiante13@atcgrid ejer4]$ sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread script_helloomp.sh
Submitted batch job 9252
[a2estudiante13@atcgrid ejer4]$ ls
helloOMP2  script_helloomp.sh  slurm-9252.out
[a2estudiante13@atcgrid ejer4]$ cat slurm-9252.out
Id. usuario del trabajo: a2estudiante13
Id. del trabajo: 9252
Nombre del trabajo especificado por usuario: helloOMP
Directorio de trabajo (en el que se ejecuta el script):
/home/a2estudiante13/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):

/var/spool/slurmd/job09252/slurm_script: línea 23: :i: no se encontró la orden
(0:Hola )(3:Hola )(2:Hola )(7:Hola )(1:Hola )(6:Hola )(4:Hola )(8:Hola )(10:Hola )(5:Hola )(11:Hola )(9:Hola )(0:mundo!)(10:mundo!)(8:mundo!)(7:mundo!
)(2:mundo!)(3:mundo!)(9:mundo!)(11:mundo!)(5:mundo!)(6:mundo!)(4:mundo!)(1:mundo!)
 2. Ejecución helloOMP varias veces con distinto no de threads:


 - Para 12 threads:
(5:Hola )(0:Hola )(8:Hola )(6:Hola )(1:Hola )(9:Hola )(7:Hola )(10:Hola )(2:Hola )(11:Hola )(4:Hola )(3:Hola )(5:mundo!)(0:mundo!)(2:mundo!)(9:mundo!)
(1:mundo!)(10:mundo!)(8:mundo!)(7:mundo!)(3:mundo!)(4:mundo!)(6:mundo!)(11:mundo!)
 - Para 6 threads:
(3:Hola )(2:Hola )(1:Hola )(0:Hola )(4:Hola )(5:Hola )(3:mundo!)(5:mundo!)(2:mundo!)(1:mundo!)(4:mundo!)(0:mundo!)
 - Para 3 threads:
(1:Hola )(0:Hola )(2:Hola )(1:mundo!)(0:mundo!)(2:mundo!)
 - Para 1 threads:
(0:Hola )(0:mundo!)[a2estudiante13@atcgrid ejer4]$
```

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

RESPUESTA: atcgrid1 , la ejecución del script nos da la información (**nodos asignados al trabajo: 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: (He ejecutado el programa para tamaños 10 , 20 y 30)

```
alex@alex-CX62-6QD:~/bp0/ejer5$ gcc  -O2  sumavectores.c -o sumavectores -lrt
alex@alex-CX62-6QD:~/bp0/ejer5$ ./sumavectores 10
Tamaño Vectores:10 (4 B)
Tiempo:0.000000427       / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.10
0000=2.000000) /
alex@alex-CX62-6QD:~/bp0/ejer5$ ./sumavectores 20
Tamaño Vectores:20 (4 B)
Tiempo:0.000000452       / Tamaño Vectores:20   / V1[0]+V2[0]=V3[0](2.000000+2.000000=4.000000) / / V1[19]+V2[19]=V3[19](3.900000+0
.100000=4.000000) /
alex@alex-CX62-6QD:~/bp0/ejer5$ ./sumavectores 30
Tamaño Vectores:30 (4 B)
Tiempo:0.000000573       / Tamaño Vectores:30   / V1[0]+V2[0]=V3[0](3.000000+3.000000=6.000000) / / V1[29]+V2[29]=V3[29](5.900000+0
.100000=6.000000) /
alex@alex-CX62-6QD:~/bp0/ejer5$
```

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: Como vemos en la captura adjuntada , la variable ncgt hace la diferencia entre el instante antes de la suma de vectores y el instante tras la suma. Por tanto , ncgt = tiempo de ejecución.

```
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));
```

(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:

int clock_gettime(clockid_t *clk_id*, struct timespec **tp*);

La información del tiempo almacena en la estructura timespec , que tiene 2 campos:

Los **segundos** (del tipo **time_t ,** que es para almacenamiento de tiempos del sistema UNIX) (**8B**)

Los **nanosegundos** de tipo **long**

```
struct timespec {
        time_t   tv_sec;        /* seconds */
        long     tv_nsec;       /* nanoseconds */
};
```

(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: Como he dicho en el ejercicio anterior , devuelve los segundos y los nanosegundos (tipo time_t y long respectivamente)

En el programa del listado 1 , utilizamos el reloj CLOCK_REALTIME las dos veces que llamamos a la función , de esa forma , podemos ver el tiempo que ha tardado en hacer la operación con los vectores.

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:

(tiempos en mi pc)				
Nº Componentes	Bytes de un vector	Tiempo para vectores locales	Tiempo para vectores globales	Tiempo para vectores dinámicos
65536	262144	0.000183883	0,000279496	0,000879787
131072	524288	0,00029631	0,000607583	0,000598354
262144	1048576	0,000299024	0,001445371	0,001390112
524288	2097152	Violación de segmento	0,002500327	0,002573779
1048576	4194304	Violación de segmento	0,004914389	0,005003233
2097152	8388608	Violación de segmento	0,009774297	0,009885306
4194304	16777216	Violación de segmento	0,018105702	0,018062792
8388608	33554432	Violación de segmento	0,035796098	0,036204367
16777216	67108864	Violación de segmento	0,070789086	0,072073233
33554432	134217728	Violación de segmento	0,142276727	0,142485696
67108864	268435456	Violación de segmento	0,143043489	0,280331771

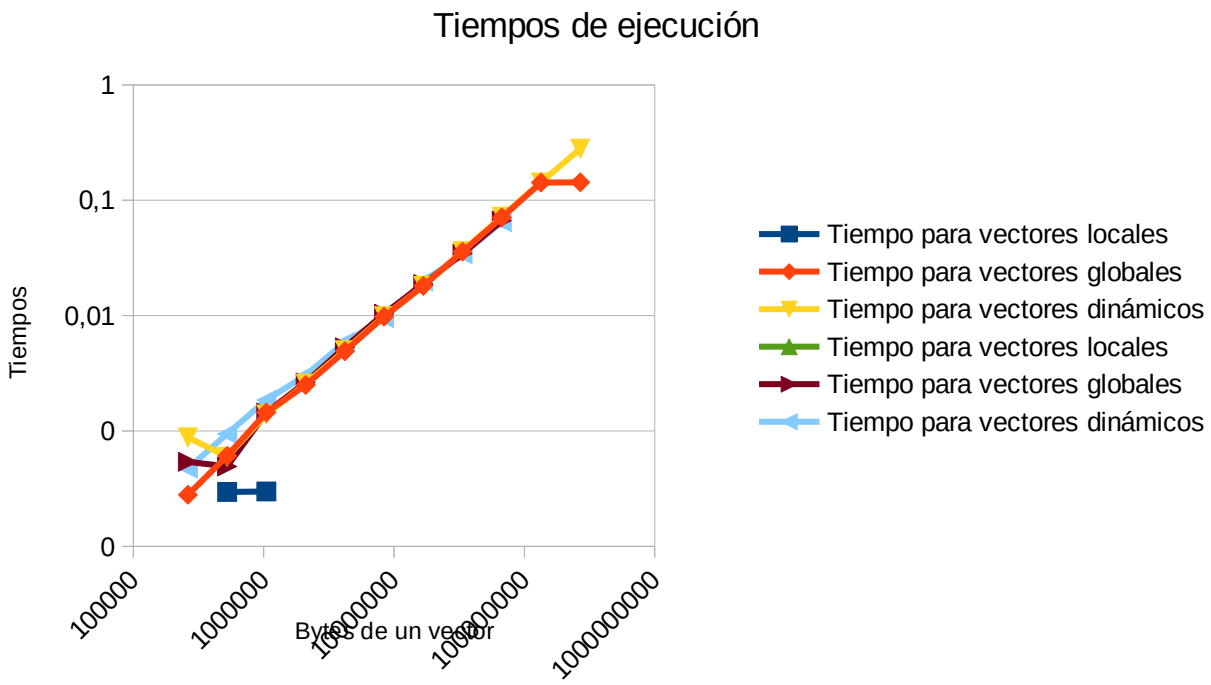
(tiempos en el clúster)				
Nº Componentes	Bytes de un vector	Tiempo para vectores locales	Tiempo para vectores globales	Tiempo para vectores dinámicos
65536	262144	0.000468762	0.000540615	0.000471700
131072	524288	0.000951365	0.000492611	0.000941632
262144	1048576	0.001906719	0.001447402	0.001854927
524288	2097152	Violación de segmento	0.002631435	0.003034562
1048576	4194304	Violación de segmento	0.005304824	0.005896948
2097152	8388608	Violación de segmento	0.010355580	0.009532267
4194304	16777216	Violación de segmento	0.018796092	0.019710399
8388608	33554432	Violación de segmento	0.034454142	0.034078700
16777216	67108864	Violación de segmento	0.066611785	0.064639246
33554432	134217728	Violación de segmento	0.130447177	0.128173886
67108864	268435456	Violación de segmento	0.129817430	0.255327844

Ejecución del script en atcgrid.

```
[a2estudiante13@atcgrid ejer7]$ ls
script_atcgrid.sh  SumaVectores  sumavectores.c
[a2estudiante13@atcgrid ejer7]$ sbatch -p ac script_atcgrid.sh
Submitted batch job 11280
[a2estudiante13@atcgrid ejer7]$ cat slurm-11280.out
Id. usuario del trabajo:
Id. del trabajo:
Nombre  del trabajo especificado por usuario:
Nodo que ejecuta qsub:
Directorio en el que se ha ejecutado qsub:
Cola:
Nodos asignados al trabajo:
Tamaño Vectores:65536 (4 B)
Tiempo:0.000468762       / Tamaño Vectores:65536       / V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535](13
107.100000+0.100000=13107.200000) /
Tamaño Vectores:131072 (4 B)
Tiempo:0.000951365       / Tamaño Vectores:131072       / V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[13107
1](26214.300000+0.100000=26214.400000) /
Tamaño Vectores:262144 (4 B)
Tiempo:0.001906719       / Tamaño Vectores:262144       / V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) / / V1[262143]+V2[262143]=V3[26214
3](52428.700000+0.100000=52428.800000) /
/var/spool/slurmd/job11280/slurm_script: línea 22:  7694 Violación de segmento  (`core' generado) ./SumaVectores $N
/var/spool/slurmd/job11280/slurm_script: línea 22:  7696 Violación de segmento  (`core' generado) ./SumaVectores $N
/var/spool/slurmd/job11280/slurm_script: línea 22:  7698 Violación de segmento  (`core' generado) ./SumaVectores $N
/var/spool/slurmd/job11280/slurm_script: línea 22:  7700 Violación de segmento  (`core' generado) ./SumaVectores $N
/var/spool/slurmd/job11280/slurm_script: línea 22:  7702 Violación de segmento  (`core' generado) ./SumaVectores $N
/var/spool/slurmd/job11280/slurm_script: línea 22:  7704 Violación de segmento  (`core' generado) ./SumaVectores $N
/var/spool/slurmd/job11280/slurm_script: línea 22:  7706 Violación de segmento  (`core' generado) ./SumaVectores $N
/var/spool/slurmd/job11280/slurm_script: línea 22:  7708 Violación de segmento  (`core' generado) ./SumaVectores $N
[a2estudiante13@atcgrid ejer7]$
```

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:



Nota: Los tiempos azul , naranja y amarillo son los tiempos de ejecución en el PC local.

Como podemos observar , hay diferencias en los tiempos de ejecución , siendo los del atcgrid menores debido a que funciona a más velocidad que mi pc (2.4 GHZ frente a 2.3 GHZ)

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:

```
---tam = 524288---
Tamaño Vectores:524288 (4 B)
scriptLocal.sh: línea 6:  8952 Violación de segmento  (`core' generado) ./sumavectores $N
---tam = 1048576---
Tamaño Vectores:1048576 (4 B)
scriptLocal.sh: línea 6:  8954 Violación de segmento  (`core' generado) ./sumavectores $N
---tam = 2097152---
Tamaño Vectores:2097152 (4 B)
scriptLocal.sh: línea 6:  8956 Violación de segmento  (`core' generado) ./sumavectores $N
---tam = 4194304---
Tamaño Vectores:4194304 (4 B)
scriptLocal.sh: línea 6:  8958 Violación de segmento  (`core' generado) ./sumavectores $N
---tam = 8388608---
Tamaño Vectores:8388608 (4 B)
scriptLocal.sh: línea 6:  8960 Violación de segmento  (`core' generado) ./sumavectores $N
---tam = 16777216---
Tamaño Vectores:16777216 (4 B)
scriptLocal.sh: línea 6:  8962 Violación de segmento  (`core' generado) ./sumavectores $N
---tam = 33554432---
Tamaño Vectores:33554432 (4 B)
scriptLocal.sh: línea 6:  8964 Violación de segmento  (`core' generado) ./sumavectores $N
---tam = 67108864---
```

Los vectores locales se almacenan en la pila , cuando esta llega a su máximo , se produce la violación de segmento.

(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 se obtiene ningún error con los vectores globales, esto es debido a que no están limitados por el tamaño de la pila. Reserva espacio en tiempo de compilación.

Descomentamos la definicion de vector global y comentamos la de vector estático , ejecutamos con sbatch y obtenemos:

```
[a2estudiante13@atcgrid ejer7]$ gcc  -O2  sumavectores.c -o SumaVectores -lrt
[a2estudiante13@atcgrid ejer7]$ ls
script_atcgrid.sh  slurm-11280.out  slurm-11328.out  slurm-11357.out  SumaVectores  sumavectores.c
[a2estudiante13@atcgrid ejer7]$ sbatch -p ac script_atcgrid.sh
Submitted batch job 15612
[a2estudiante13@atcgrid ejer7]$ ls
script_atcgrid.sh  slurm-11280.out  slurm-11328.out  slurm-11357.out  slurm-15612.out  SumaVectores  sumavectores.c
[a2estudiante13@atcgrid ejer7]$ cat slurm-15612.out
```

```
Tamaño Vectores:65536 (4 B)
Tiempo:0.000554499      / Tamaño Vectores:65536       / V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535](13
107.100000+0.100000=13107.200000) /
Tamaño Vectores:131072 (4 B)
Tiempo:0.000499919      / Tamaño Vectores:131072      / V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[13107
1](26214.300000+0.100000=26214.400000) /
Tamaño Vectores:262144 (4 B)
Tiempo:0.001431144      / Tamaño Vectores:262144      / V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) / / V1[262143]+V2[262143]=V3[26214
3](52428.700000+0.100000=52428.800000) /
Tamaño Vectores:524288 (4 B)
Tiempo:0.002688714      / Tamaño Vectores:524288      / V1[0]+V2[0]=V3[0](52428.800000+52428.800000=104857.600000) / / V1[524287]+V2[524287]=V3[5242
87](104857.500000+0.100000=104857.600000) /
Tamaño Vectores:1048576 (4 B)
Tiempo:0.005365987      / Tamaño 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) /
Tamaño Vectores:2097152 (4 B)
Tiempo:0.010141044      / Tamaño 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) /
Tamaño Vectores:4194304 (4 B)
Tiempo:0.018862076      / Tamaño 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) /
Tamaño Vectores:8388608 (4 B)
Tiempo:0.034769449      / Tamaño 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) /
Tamaño Vectores:16777216 (4 B)
Tiempo:0.066615277      / Tamaño 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) /
Tamaño Vectores:33554432 (4 B)
Tiempo:0.132431457      / Tamaño 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) /
Tamaño Vectores:67108864 (4 B)
Tiempo:0.132081945      / Tamaño 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:

Tampoco se obtiene ningún error , ya que los vectores dinámicos pueden ir reservando espacio en memoria durante la ejecución del programa.

Descomentamos la definición de vector dinámico y comentamos la de vector global , ejecutamos con sbatch y obtenemos:

```
Nodos asignados al trabajo:
Tamaño Vectores:65536 (4 B)
Tiempo:0.000475928       / Tamaño Vectores:65536       / V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535](13
107.100000+0.100000=13107.200000) /
Tamaño Vectores:131072 (4 B)
Tiempo:0.000793022       / Tamaño Vectores:131072      / V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[13107
1](26214.300000+0.100000=26214.400000) /
Tamaño Vectores:262144 (4 B)
Tiempo:0.001858018       / Tamaño Vectores:262144      / V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) / / V1[262143]+V2[262143]=V3[26214
3](52428.700000+0.100000=52428.800000) /
Tamaño Vectores:524288 (4 B)
Tiempo:0.002969253       / Tamaño Vectores:524288      / V1[0]+V2[0]=V3[0](52428.800000+52428.800000=104857.600000) / / V1[524287]+V2[524287]=V3[5242
87](104857.500000+0.100000=104857.600000) /
Tamaño Vectores:1048576 (4 B)
Tiempo:0.005600087       / Tamaño 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) /
Tamaño Vectores:2097152 (4 B)
Tiempo:0.010233880       / Tamaño 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) /
Tamaño Vectores:4194304 (4 B)
Tiempo:0.019034602       / Tamaño 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) /
Tamaño Vectores:8388608 (4 B)
Tiempo:0.034504183       / Tamaño 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) /
Tamaño Vectores:16777216 (4 B)
Tiempo:0.063930874       / Tamaño 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) /
Tamaño Vectores:33554432 (4 B)
Tiempo:0.128239917       / Tamaño 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) /
Tamaño Vectores:67108864 (4 B)
Tiempo:0.255969770       / Tamaño Vectores:67108864    / V1[0]+V2[0]=V3[0](6710886.400000+6710886.400000=13421772.800000) / / V1[67108863]+V2[6710886
3]=V3[67108863](13421772.700000+0.100000=13421772.800000) /
[a2estudiante13@atcgrid ejer7]$
```

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

```
unsigned int  N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
```

N es un unsigned int (4 Bytes = 32 bits) por tanto podremos representar hasta el número **2^32 – 1 = 4294967295.**

(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:

```
#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, //= 2^32 - 1
```

```
[a2estudiante13@atcgrid ejer7]$ gcc  -O2  sumavectores.c -o SumaVectores -lrt
/tmp/ccQ5VAva.o: En la función `main':
sumavectores.c:(.text.startup+0x9d): reubicación truncada para ajustar: R_X86_64_32S contra el símbolo `v2' definido en la sección COMMON en /tmp/ccQ5
VAva.o
sumavectores.c:(.text.startup+0xd1): reubicación truncada para ajustar: R_X86_64_32S contra el símbolo `v2' definido en la sección COMMON en /tmp/ccQ5
VAva.o
sumavectores.c:(.text.startup+0xda): reubicación truncada para ajustar: R_X86_64_32S contra el símbolo `v3' definido en la sección COMMON en /tmp/ccQ5
VAva.o
sumavectores.c:(.text.startup+0x12f): reubicación truncada para ajustar: R_X86_64_32S contra el símbolo `v3' definido en la sección COMMON en /tmp/ccQ
5VAva.o
sumavectores.c:(.text.startup+0x13e): reubicación truncada para ajustar: R_X86_64_32S contra el símbolo `v2' definido en la sección COMMON en /tmp/ccQ
5VAva.o
sumavectores.c:(.text.startup+0x155): reubicación truncada para ajustar: R_X86_64_PC32 contra el símbolo `v3' definido en la sección COMMON en /tmp/cc
Q5VAva.o
sumavectores.c:(.text.startup+0x15d): reubicación truncada para ajustar: R_X86_64_PC32 contra el símbolo `v2' definido en la sección COMMON en /tmp/cc
Q5VAva.o
sumavectores.c:(.text.startup+0x1b4): reubicación truncada para ajustar: R_X86_64_32S contra el símbolo `v3' definido en la sección COMMON en /tmp/ccQ
5VAva.o
sumavectores.c:(.text.startup+0x1c7): reubicación truncada para ajustar: R_X86_64_32S contra el símbolo `v2' definido en la sección COMMON en /tmp/ccQ
5VAva.o
```

El error es lanzado por el enlazador , ya que el tamaño del vector global excede el rango de lo direcionable por una instrucción de direccionamiento de 32-bit , que es 2GB.

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;
}
```