2º curso / 2º cuatr.

Grados 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:

Parte I. Ejercicios basados en los ejemplos del seminario práctico

Crear el directorio con nombre `bp0` en atcgrid y en el PC (PC = PC del aula de prácticas o su computador personal).

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 x se debe usar con `sbatch/srun` la opción `--cpus-per-task=x (-cx)`.
- En slurm, por defecto, `cpu` se refiere a cores lógicos (ej. en la opción `-c`), si no se quieren usar cores lógicos hay que añadir la opción `--hint=nomultithread` a `sbatch/srun`. Para que con `sbatch` se tenga en cuenta `---hint=nomultithread` se debe usar `srun` dentro del script delante del ejecutable.
- Para asegurar que solo se crea un proceso hay que incluir `--ntasks=1 (-n1)` en `sbatch/srun`.
- Para que no se ejecute más de un proceso en un nodo de cómputo 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 (partición slurm).
- Las opciones de `sbatch` se pueden especificar también dentro del *script* (usando #SBATCH, ver ejemplos en el script del seminario)
- Se recomienda escribir las órdenes directamente en la ventana de comandos (*shell*) en lugar de usar copy/paste.

1. Ejecutar `lscpu` en el PC, en atcgrid4 (usar en este caso `-p ac4`) y en uno de los restantes nodos de cómputo (atcgrid1, atcgrid2 o atcgrid3, usar en este caso `-p ac`).

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

   RESPUESTA:

```
[ac412@atcgrid practica0]$ srun -p ac4 lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    2
Core(s) per socket:    16
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz
Stepping:              7
CPU MHz:               1188.684
CPU max MHz:           3200,0000
CPU min MHz:           800,0000
BogoMIPS:              4200.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              22528K
NUMA node0 CPU(s):     0-15,32-47
NUMA node1 CPU(s):     16-31,48-63
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 art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb cat_l3 cdp_l3 invpcid_single intel_ppin intel_pt ssbd mba ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm mpx rdt_a avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida arat pln pts pku ospke avx512_vnni md_clear spec_ctrl intel_stibp flush_l1d arch_capabilities
[ac412@atcgrid practica0]$
```

```
[ac412@atcgrid practica0]$ 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:              4799.64
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 f
lexpriority ept vpid dtherm ida arat spec_ctrl intel_stibp flush_l1d
[ac412@atcgrid practica0]$
```

(b) ¿Cuántos cores físicos y cuántos cores lógicos tiene atcgrid4?, ¿cuántos tienen atcgrid1, atcgrid2 y atcgrid3? y ¿cuántos tiene el PC? Razonar las respuestas

RESPUESTA: En vista de las capturas del apartado anterior, atcgrid4 posee 64 cores físicos, y como cada core puede usar dos hilos, 64 * 2 = 128 cores lógicos. En la segunda captura, podemos ver que atcgrid1, atcgrid2 y atcgrid3 poseen 24 cores físicos, y como cada core puede usar dos hilos, 24*2 = 48 cores lógicos. Por último, haciendo lscpu en mi portátil, he podido ver que el PC tiene 8 cores físicos y como cada core puede usar dos hilos, 8*2 = 16 cores lógicos.

2. Compilar y ejecutar en el PC el código `HelloOMP.c` del seminario.

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

RESPUESTA:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/bp0$ gcc -O2 -fopenmp -o HelloOMP HelloOMP.c
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/bp0$ ./HelloOMP
(0:!!!Hello world!!!)(7:!!!Hello world!!!)(3:!!!Hello world!!!)(2:!!!Hello world!!!)(1:!!!Hello world!!!)(4:!!!Hello world!!!)(6:!!!Hello world!!!)(5:!!!Hello world!!!)joshoc7@joshoc7-Aspire-A315-56:~/Es
critorio/bp0$
```

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

RESPUESTA: Por lo visto en el apartado 1, como el PC tiene 8 cores, se imprimen 8 "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 (de 1 a 3) a través de cola ac del gestor de colas utilizando directamente en línea de comandos (no use ningún *script*):

(a) `srun --partition=ac --account=ac --ntasks=1 --cpus-per-task=12 --hint=nomultithread HelloOMP`

(Alternativa: `srun -pac -Aac -n1 -c12 --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:

```
[ac412@atcgrid ejer2]$ srun --partition=ac --account=ac --ntasks=1 --cpus-per-task=12 --hint=nomultithread HelloOMP
(0:!!!Hello world!!!)(8:!!!Hello world!!!)(4:!!!Hello world!!!)(2:!!!Hello world!!!)(6:!!!Hello world!!!)(10:!!!Hello world!!!)(11:!!!Hello world!!!)(5:!!!Hello world!!!)(7:!!!Hello world!!!)(3:!!!Hello
[ac412@atcgrid ejer2]$
```

(b) `srun -pac -Aac -n1 -c24 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:

```
[ac412@atcgrid ejer2]$ srun -pac -Aac -n1 -c24  HelloOMP
(22:!!!Hello world!!!)(6:!!!Hello world!!!)(11:!!!Hello world!!!)(14:!!!Hello world!!!)(12:!!!Hello world!!!)(10:!!!Hello world!!!)(18:!!!Hello world!!!)(17:!!!Hello world!!!)(7:!!!Hello world!!!)(16:!!!
Hello world!!!)(2:!!!Hello world!!!)(5:!!!Hello world!!!)(3:!!!Hello world!!!)(0:!!!Hello world!!!)(21:!!!Hello world!!!)(23:!!!Hello world!!!)(20:!!!Hello world!!!)(1:!!!Hello world!!!)(13:!!!Hello worl
d!!!)(8:!!!Hello world!!!)(4:!!!Hello world!!!)(19:!!!Hello world!!!)(15:!!!Hello world!!!)(9:!!!Hello world!!!)[ac412@atcgrid ejer2]$
```

(c) `srun -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. ¿Qué partición (cola) se está usando?

RESPUESTA: Se está usando una de las tres particiones porque son las que se usan por defecto cuando no se especifica nada.

```
[ac412@atcgrid ejer2]$ srun -n1 HelloOMP
(0:!!!Hello world!!!)(1:!!!Hello world!!!)[ac412@atcgrid ejer2]$
```

(d) ¿Qué orden `srun` usaría para que HelloOMP utilice todos los cores físicos de atcgrid4 (se debe imprimir un único mensaje desde cada uno de ellos)?

Usaría la siguiente orden: srun -pac4 -Aac -n1 -c64 HelloOMP. Tras ejecutarla tenemos:

```
[ac412@atcgrid ejer2]$ srun -pac4 -Aac -n1 -c64  HelloOMP
(5:!!!Hello world!!!)(51:!!!Hello world!!!)(53:!!!Hello world!!!)(50:!!!Hello world!!!)(21:!!!Hello world!!!)(27:!!!Hello world!!!)(52:!!!Hello world!!!)(59:!!!Hello world!!!)(61:!!!Hello world!!!)(16:!!
!Hello world!!!)(47:!!!Hello world!!!)(2:!!!Hello world!!!)(55:!!!Hello world!!!)(32:!!!Hello world!!!)(4:!!!Hello world!!!)(15:!!!Hello world!!!)(11:!!!Hello world!!!)(1:!!!Hello world!!!)(63:!!!Hello w
orld!!!)(45:!!!Hello world!!!)(14:!!!Hello world!!!)(3:!!!Hello world!!!)(35:!!!Hello world!!!)(9:!!!Hello world!!!)(0:!!!Hello world!!!)(25:!!!Hello world!!!)(46:!!!Hello world!!!)(39:!!!Hello world!!!)
(44:!!!Hello world!!!)(49:!!!Hello world!!!)(26:!!!Hello world!!!)(38:!!!Hello world!!!)(36:!!!Hello world!!!)(20:!!!Hello world!!!)(33:!!!Hello world!!!)(62:!!!Hello world!!!)(34:!!!Hello world!!!)(48:!
!!Hello world!!!)(13:!!!Hello world!!!)(41:!!!Hello world!!!)(43:!!!Hello world!!!)(6:!!!Hello world!!!)(30:!!!Hello world!!!)(40:!!!Hello world!!!)(42:!!!Hello world!!!)(24:!!!Hello world!!!)(8:!!!Hello
 world!!!)(18:!!!Hello world!!!)(57:!!!Hello world!!!)(7:!!!Hello world!!!)(29:!!!Hello world!!!)(19:!!!Hello world!!!)(23:!!!Hello world!!!)(17:!!!Hello world!!!)(60:!!!Hello world!!!)(22:!!!Hello world
!!!)(56:!!!Hello world!!!)(12:!!!Hello world!!!)(31:!!!Hello world!!!)(10:!!!Hello world!!!)(58:!!!Hello world!!!)(37:!!!Hello world!!!)(28:!!!Hello world!!!)(54:!!!Hello world!!!)[ac412@atcgrid ejer2]$
```

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 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:

```c
/**
 *
 Compilar con gcc -O2 -fopenmp HelloOMP2.c -o HelloOMP2
 *
 */

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

int main(void) {

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

return(0);

}
```

```
[ac412@atcgrid ~]$ gcc -O2 -fopenmp HelloOMP2.c -o HelloOMP2
[ac412@atcgrid ~]$ sbatch script_helloomp.sh
Submitted batch job 123241
[ac412@atcgrid ~]$ cat slurm-123241.out
Id. usuario del trabajo: ac412
Id. del trabajo: 123241
Nombre del trabajo especificado por usuario: helloOMP
Directorio de trabajo (en el que se ejecuta el script): /home/ac412
Cola: ac
Nodo que ejecuta este trabajo:atcgrid.ugr.es
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)

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

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

  - Para 12 threads:

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

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

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

(0:!!!Hello)
(0:world!!!)             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           121114       ac4 helloOMP    ac413 PD       0:00      2 (PartitionConfig)
           123241        ac helloOMP    ac412  R       0:00      1 atcgrid1
[ac412@atcgrid ~]$
```

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

RESPUESTA: atcgrid1. Nos informa de esto la salida almacenada en un archivo slurm.out, como se puede ver en la captura anterior.

(c) ¿Qué órdenes para el gestor de colas slurm incluye el *script*? Explicar cómo ha obtenido esta información.

RESPUESTA: Observando el código del script, observamos que al principio incluye órdenes para el gestor de carga de trabajo. Estas órdenes dejan predefinidas algunas propiedades a la hora de ejecutar, que son: el nombre del trabajo, la partición y la cuenta en la que se realiza el trabajo, que el trabajo no comparta recursos, y que solo se cree un proceso que pueda utilizar un máximo de 12 núcleos.

También se ha utilizado la orden “srun” para ejecutar el trabajo.

(d) Haga los cambios necesarios en el *script* para que se utilice atcgrid4. Comentar los cambios realizados y los motivos por los que se han hecho.

RESPUESTA: En el script, en la cabecera he cambiado una línea #SBATCH en la que se especifica la partición a usar, cambiando ac por ac4. El motivo de haber hecho esto es que de esta forma la orden sbatch ejecuta el script desde la partición que nosotros deseamos.

NOTA: Utilizar siempre con `sbatch` las opciones `-n1` y `-c`, `--exclusive` y, para usar cores físicos y no lógicos, no olvidar incluir `--hint=nomultithread`. Utilizar siempre con `srun`, si lo usa fuera de un script, las opciones `-n1` y `-c` y, para usar cores físicos y no lógicos, no olvide incluir `--hint=nomultithread`. Recordar que los `srun` dentro de un *script* heredan las opciones incluidas 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 `SumaVectores.c` 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`). 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:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/bp0$ gcc  -O2  SumaVectores.c -o SumaVectores -lrt
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/bp0$ ./SumaVectores 4
Tamaño Vectores:4 (4 B)
Tiempo:0.000000361       / Tamaño Vectores:4
/ V1[0]+V2[0]=V3[0](0.400000+0.400000=0.800000) /
/ V1[1]+V2[1]=V3[1](0.500000+0.300000=0.800000) /
/ V1[2]+V2[2]=V3[2](0.600000+0.200000=0.800000) /
/ V1[3]+V2[3]=V3[3](0.700000+0.100000=0.800000) /
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/bp0$
```

6. En el código `SumaVectores.c` 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: Contiene un valor numérico con decimales, pues esta variable es de tipo double, que representa el tiempo de ejecución de la suma de vectores, ya que su valor es el de la diferencia entre cgt1 y cgt2, structs timespec en los que se ha almacenado el valor del reloj del sistema antes de empezar la ejecución y justo al acabarla.

(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: Devuelve un dato de tipo timespec. Timespec es una estructura de datos que consiste en un struct que almacena dos datos: tv_sec (es de tipo time_t y almacena segundos) y tv_nsec (es de tipo long y almacena 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: clock_gettime() devuelve en el struct timespec usado como segundo argumento la hora del reloj del sistema en el momento actual (existen otro tipo de opciones para modificar este funcionamiento, como se puede ver en el man de Linux). Siempre devuelve un int que es 0 en caso de éxito y -1 en caso de error.

7. Rellenar una tabla como la Tabla 1 en una hoja de cálculo con los tiempos de ejecución del código SumaVectores.c para vectores locales, globales y dinámicos (se pueden obtener errores en tiempo de ejecución o de compilación, ver ejercicio 9). Obtener estos resultados usando *scripts* (partir del *script* que hay en el seminario). Debe haber una tabla para un nodo de cómputo de atcgrid con procesador Intel Xeon E5645 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:

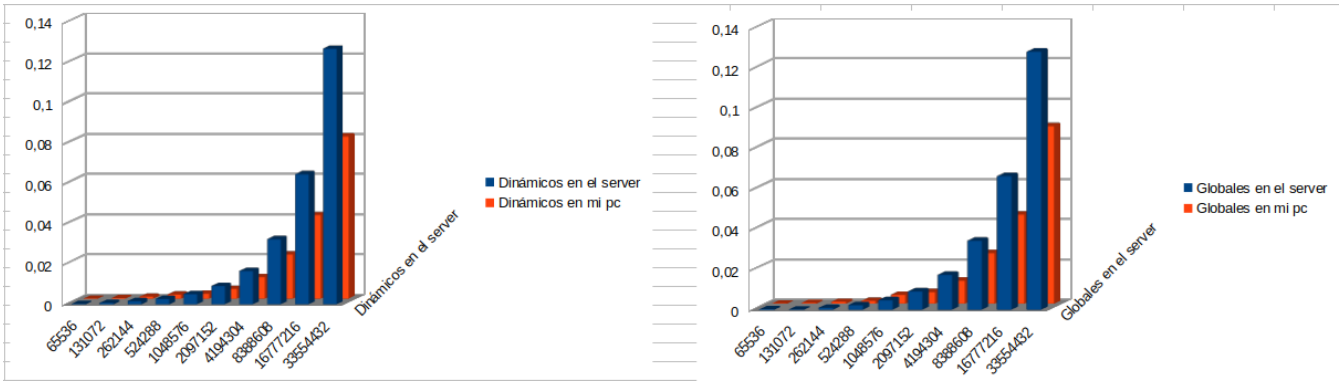
La primera tabla es de los tiempos obtenidos en el PC y la segunda es de los tiempos obtenidos en un nodo de cómputo de atcgrid:

N.º componentes	Bytes de un vector	Tiempo vect.locales	Tiempo vect.globales	Tiempo vect.dinámicos
65536	262144	0.000129273	0.000485253	0.000474735
131072	524288	0.000237571	0.000844608	0.000867588
262144	1048576	0.000268520	0.001702649	0.001366815
524288	2097152	CORE	0.002115613	0.001995662
1048576	4194304	CORE	0.003354219	0.003232513
2097152	8388608	CORE	0.006292948	0.006327733
4194304	16777216	CORE	0.012117733	0.013467573
8388608	33554432	CORE	0.024375785	0.024414714
16777216	67108864	CORE	0.042849282	0.047956784
33554432	134217728	CORE	0.087770935	0.089483960
67108864	268435456	CORE	0.086766337	0.174308643

N.º componentes	Bytes de un vector	Tiempo vect.locales	Tiempo vect.globales	Tiempo vect.dinámicos
65536	262144	0.000480307	0.000522883	0.000488314
131072	524288	0.000970871	0.000499053	0.000968764
262144	1048576	0.001941787	0.001477655	0.001956359
524288	2097152	CORE	0.002783153	0.002953757
1048576	4194304	CORE	0.005481512	0.005384656
2097152	8388608	CORE	0.010135470	0.010188105
4194304	16777216	CORE	0.018605457	0.018641865
8388608	33554432	CORE	0.034007665	0.034171120
16777216	67108864	CORE	0.067082848	0.064971529
33554432	134217728	CORE	0.133857999	0.128535945
67108864	268435456	CORE	0.132435005	0.254932301

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:



Los tiempos en el server son algo más rápidos, pero los del PC y los del server están en el mismo orden por lo que la diferencia es mínima. Esto se debe a que no estamos utilizando ningún paralelismo en la ejecución de SumaVectores.c.

2. Contestar a las siguientes preguntas:

(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:

```
[ac412@atcgrid ~]$ sbatch script_sumavectoresL.sh
Submitted batch job 125434
[ac412@atcgrid ~]$ cat slurm-125434.out
Tamaño Vectores:65536 (4 B)
Tiempo:0.000497771      / Tamaño 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) /
Tamaño Vectores:131072 (4 B)
Tiempo:0.000966021      / Tamaño 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) /
Tamaño Vectores:262144 (4 B)
Tiempo:0.001950350      / Tamaño 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) /
/var/spool/slurmd/job125434/slurm_script: linea 5: 25557 Violación de segmento  (`core' generado) ./SumaVectoresL $CONTADOR
/var/spool/slurmd/job125434/slurm_script: linea 5: 25559 Violación de segmento  (`core' generado) ./SumaVectoresL $CONTADOR
/var/spool/slurmd/job125434/slurm_script: linea 5: 25564 Violación de segmento  (`core' generado) ./SumaVectoresL $CONTADOR
/var/spool/slurmd/job125434/slurm_script: linea 5: 25566 Violación de segmento  (`core' generado) ./SumaVectoresL $CONTADOR
/var/spool/slurmd/job125434/slurm_script: linea 5: 25568 Violación de segmento  (`core' generado) ./SumaVectoresL $CONTADOR
/var/spool/slurmd/job125434/slurm_script: linea 5: 25570 Violación de segmento  (`core' generado) ./SumaVectoresL $CONTADOR
/var/spool/slurmd/job125434/slurm_script: linea 5: 25572 Violación de segmento  (`core' generado) ./SumaVectoresL $CONTADOR
/var/spool/slurmd/job125434/slurm_script: linea 5: 25574 Violación de segmento  (`core' generado) ./SumaVectoresL $CONTADOR
[ac412@atcgrid ~]$
```

A partir de la cuarta iteración, es decir, para vectores de 524288 elementos, nos da un core. Esto se debe a que las variables locales se reservan en la pila y como no se ha borrado el espacio usado en anteriores llamadas, llega un momento en el que no se puede usar más espacio y nos da una 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:

```
[ac412@atcgrid ~]$ sbatch script_sumavectoresG.sh
Submitted batch job 125436
[ac412@atcgrid ~]$ cat slurm-125436.out
Tamaño Vectores:65536 (4 B)
Tiempo:0.000561433      / Tamaño 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) /
Tamaño Vectores:131072 (4 B)
Tiempo:0.000512813      / Tamaño 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) /
Tamaño Vectores:262144 (4 B)
Tiempo:0.001472359      / Tamaño 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) /
Tamaño Vectores:524288 (4 B)
Tiempo:0.002553713      / Tamaño 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) /
Tamaño Vectores:1048576 (4 B)
Tiempo:0.005233975      / 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:0.009799810      / 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:0.018359222      / 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:0.035063426      / 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:0.065783807      / 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:0.132369288      / 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:0.132916644      / 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
) /
[ac412@atcgrid ~]$
```

Aquí no se dan problemas. Lo único que cabe destacar es que en la última iteración, el tamaño es 33554432 elementos a pesar de que se haya introducido 67108864, ya que en el propio programa hay una restricción en la que se fija el valor máximo que puede tomar el n.º de elementos en caso de usar vectores globales.

(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:

```
[ac412@atcgrid ~]$ sbatch script_sumavectoresD.sh
Submitted batch job 125438
[ac412@atcgrid ~]$ cat slurm-125438.out
Tamaño Vectores:65536 (4 B)
Tiempo:0.000456000      / Tamaño 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) /
Tamaño Vectores:131072 (4 B)
Tiempo:0.000981418      / Tamaño 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) /
Tamaño Vectores:262144 (4 B)
Tiempo:0.001987272      / Tamaño 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) /
Tamaño Vectores:524288 (4 B)
Tiempo:0.003068342      / Tamaño 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) /
Tamaño Vectores:1048576 (4 B)
Tiempo:0.005814198      / 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:0.010320614      / 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:0.019311378      / 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:0.034871165      / 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:0.065784193      / 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:0.129511061      / 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:0.255299143      / Tamaño Vectores:67108864     / V1[0]+V2[0]=V3[0](6710886.400000+6710886.400000=13421772.800000) / / V1[67108863]+V2[67108863]=V3[67108863](13421772.700000+0.100000=13421772.800
000) /
[ac412@atcgrid ~]$
```

En el caso de los vectores dinámicos no hay ningún problema ya que al final del programa se libera el espacio reservado en el heap para los vectores dinámicos que se han creado.

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

RESPUESTA:

La variable N es unsigned int, y este tipo comprende rangos entre el 0 y 2^(32)-1, por lo que el máximo valor que podemos almacenar en esta variable es 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:

```
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/bp0$ gcc  -O2  SumaVectoresD.c -o SumaVectoresD -lrt
/tmp/ccNYO3Ot.o: en la función `main':
SumaVectoresD.c:(.text.startup+0x8b): reubicación truncada para ajustar: R_X86_64_PC32 contra el símbolo `v2' definido en la sección COMMON en /tmp/ccNYO3Ot.o
SumaVectoresD.c:(.text.startup+0xda): reubicación truncada para ajustar: R_X86_64_PC32 contra el símbolo `v3' definido en la sección COMMON en /tmp/ccNYO3Ot.o
collect2: error: ld returned 1 exit status
joshoc7@joshoc7-Aspire-A315-56:~/Escritorio/bp0$
```

Nos da un error ya que en caso de querer darle el tamaño máximo a N, necesitaríamos más tamaño de pila del que disponemos.

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. Se generan aleatoriamente las componentes para vectores de tamaño mayor que 8 y se imprimen todas las componentes para vectores menores que 10.

```
/* 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
if (N < 9)
  for (i = 0; i < N; i++)
  {
    v1[i] = N * 0.1 + i * 0.1;
    v2[i] = N * 0.1 - i * 0.1;
  }
else
{
  srand48(time(0));
  for (i = 0; i < N; i++)
  {
    v1[i] = drand48();
    v2[i] = drand48(); //printf("%d:%f,%f/",i,v1[i],v2[i]);
  }
}

  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;
}
```