

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 0. Entorno de programación

Estudiante (nombre y apellidos):

Grupo de prácticas y profesor de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

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

Parte I. Ejercicios basados en los ejemplos del seminario práctico

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

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

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

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

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

RESPUESTA:

```
[MariaAguadoMartinez maguado@maguado-HP-Pavilion-x360-Convertible-14-ba1xx:~] 2020-02-24 lunes
$lscpu
Arquitectura:                x86_64
Modo(s) de operación de las CPUs:  32-bit, 64-bit
Orden de los bytes:           Little Endian
CPU(s):                       8
Lista de la(s) CPU(s) en línea:    0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:         4
«Socket(s)»:                   1
Modo(s) NUMA:                  1
ID de fabricante:              GenuineIntel
Familia de CPU:                 6
Modelo:                         142
Nombre del modelo:              Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Revisión:                       10
CPU MHz:                        900.027
CPU MHz máx.:                   4000.0000
CPU MHz mín.:                   400.0000
BogoMIPS:                       3999.93
Virtualización:                 VT-x
Caché L1d:                      32K
Caché L1i:                      32K
Caché L2:                       256K
Caché L3:                       8192K
CPU(s) del nodo NUMA 0:         0-7
Indicadores:                    fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clf
lsh dts acpi mmx fxsr sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg
fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic 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 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsa
vec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear flush_l1d
```

```
[MariaAguado e2estudiante2@atcgrid:~] 2020-02-24 lunes
$lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:             Little Endian
CPU(s):                 8
On-line CPU(s) list:    0-7
Thread(s) per core:     2
Core(s) per socket:     4
Socket(s):              1
NUMA node(s):           1
Vendor ID:              GenuineIntel
CPU family:             6
Model:                  158
Model name:              Intel(R) Xeon(R) CPU E3-1230 v6 @ 3.50GHz
Stepping:                9
CPU MHz:                 1599.395
CPU max MHz:             3900.0000
CPU min MHz:             800.0000
BogoMIPS:                7008.00
Virtualization:          VT-x
L1d cache:               32K
L1i cache:               32K
L2 cache:                256K
L3 cache:                8192K
NUMA node0 CPU(s):       0-7
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 cx
                        16 xtptr pcdm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dno
                        wprefetch epb intel_pt ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle
                        avx2 smep bmi2 erms invpcid rtm mpx rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 dtherm ida arat pln pts h
                        wp hwp_notify hwp_act_window hwp_epp md_clear spec_ctrl intel_stibp flush_l1d
```

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

Según la información obtenida con `lscpu`, podemos decir que el PC tiene 4 cores físicos en un solo socket y 2 cores lógicos por físico, entonces tendríamos 8 cores lógicos.

Sin embargo, en el cluster tenemos 4 cores físicos y de la misma manera 2 lógicos por cada físico.

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.

RESPUESTA:

```
[MariaAguadoMartínez maguado@maguado-HP-Pavilion-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer2] 2020-02-24 lunes
$gcc -O2 -fopenmp -o HelloOMP HelloOMP.c
[MariaAguadoMartínez maguado@maguado-HP-Pavilion-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer2] 2020-02-24 lunes
$chmod +x HelloOMP
[MariaAguadoMartínez maguado@maguado-HP-Pavilion-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer2] 2020-02-24 lunes
$./HelloOMP
(7:!!!Hello world!!!)(1:!!!Hello world!!!)(5:!!!Hello world!!!)(3:!!!Hello world!!!)(0:!!!Hello world!!!)(6:!!!Hello world!!!)(4:!!!Hello world!!!)(2:!!!Hello world!!!)[MariaAguadoMartínez maguado@maguado-HP-Pavilion-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer2] 2020-02-24 lunes
$
```

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

RESPUESTA:

Aparecen 8 Hello world, pues el programa imprime uno por cada hebra, es decir, uno por cada core lógico, y como ya hemos visto, tenemos 8

3. Copiar el ejecutable de `HelloOMP.c` que ha generado anteriormente y que se encuentra en el directorio `ej2` del PC al directorio `ej2` de su home en el *front-end* de `atcgrid`. Ejecutar este código en un nodo de cómputo de `atcgrid` a través de `cola ac` del gestor de colas (no use ningún *script*) utilizando directamente en línea de comandos:

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

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

RESPUESTA:

```
$srun -p ac -n1 --cpus-per-task=12 --hint=nomultithread HelloOMP
H(0:!!!Hello world!!!)(4:!!!Hello world!!!)(11:!!!Hello world!!!)(5:!!!Hello world!!
s!)(9:!!!Hello world!!!)(2:!!!Hello world!!!)(3:!!!Hello world!!!)(7:!!!Hello world!
!!)(1:!!!Hello world!!!)(10:!!!Hello world!!!)(8:!!!Hello world!!!)(6:!!!Hello worl
d!!!)[MaríaAguado e2estudiante2@atcgrid:~/bp0/ejer2] 2020-02-20 jueves
$
```

(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:

```
s[MaríaAguado e2estudiante2@atcgrid:~/bp0/ejer2] 2020-02-20 jueves
R$srun -p ac -n1 --cpus-per-task=24 HelloOMP
s(21:!!!Hello world!!!)(17:!!!Hello world!!!)(3:!!!Hello world!!!)(5:!!!Hello world!
U!)(16:!!!Hello world!!!)(14:!!!Hello world!!!)(23:!!!Hello world!!!)(11:!!!Hello w
orld!!!)(20:!!!Hello world!!!)(12:!!!Hello world!!!)(15:!!!Hello world!!!)(18:!!!He
llo world!!!)(9:!!!Hello world!!!)(7:!!!Hello world!!!)(13:!!!Hello world!!!)(6:!!!
HHello world!!!)(4:!!!Hello world!!!)(19:!!!Hello world!!!)(22:!!!Hello world!!!)(0:
s!!!Hello world!!!)(8:!!!Hello world!!!)(10:!!!Hello world!!!)(2:!!!Hello world!!!)(
1:!!!Hello world!!!)[MaríaAguado e2estudiante2@atcgrid:~/bp0/ejer2] 2020-02-20 juev
es
```

(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:

```
$srun -p ac -n1 HelloOMP
(1:!!!Hello world!!!)(0:!!!Hello world!!!)[MaríaAguado e2estudiante2@atcgrid:~/bp0/
ej2] 2020-02-20 jueves
```

(d) ¿Qué orden sr un 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)?

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

Sin incluir el `-hint=nomultithread`, para que utilice los nodos que queremos.

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:

Código nuevo:

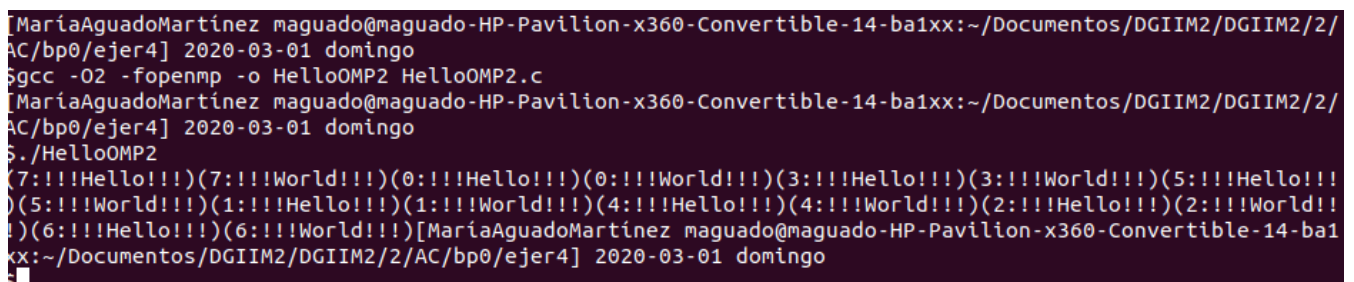
```
/*Compilar con: gcc-02 -fopenmp-o HelloOMPHelloOMP.c */
#include<stdio.h>
#include<omp.h>

int main(void){
#pragma omp parallel
{

printf("%d:!!!Hello!!!", omp_get_thread_num());
printf("%d:!!!World!!!", omp_get_thread_num());
}
return(0);

}
```

Compilación :



```
[MaríaAguadoMartínez maguado@maguado-HP-Pavilion-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer4] 2020-03-01 domingo
$gcc -02 -fopenmp -o HelloOMP2 HelloOMP2.c
[MaríaAguadoMartínez maguado@maguado-HP-Pavilion-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer4] 2020-03-01 domingo
$./HelloOMP2
(7:!!!Hello!!!)(7:!!!World!!!)(0:!!!Hello!!!)(0:!!!World!!!)(3:!!!Hello!!!)(3:!!!World!!!)(5:!!!Hello!!!)(5:!!!World!!!)(1:!!!Hello!!!)(1:!!!World!!!)(4:!!!Hello!!!)(4:!!!World!!!)(2:!!!Hello!!!)(2:!!!World!!!)(6:!!!Hello!!!)(6:!!!World!!!)[MaríaAguadoMartínez maguado@maguado-HP-Pavilion-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer4] 2020-03-01 domingo
```

Envío a la cola del atcgrid:

```
[MaríaAguado e2estudiante2@atcgrid:~/bp0/ejer4] 2020-03-01 domingo
$chmod +rwx script_helloomp.sh
[MaríaAguado e2estudiante2@atcgrid:~/bp0/ejer4] 2020-03-01 domingo
$sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread script_helloomp.sh
Submitted batch job 15278
[MaríaAguado e2estudiante2@atcgrid:~/bp0/ejer4] 2020-03-01 domingo
$
```

Resultado del fichero slurm-15278.out:

```
[MaríaAguado e2estudiante2@atcgrid:~/bp0/ejer4] 2020-03-01 domingo
$cat slurm-15278.out
Id: usuario del trabajo: e2estudiante2
Id: del trabajo: 15278
Nombre del trabajo especificado por usuario: helloOMP
Directorio de trabajo (en el que se ejecuta el script): /home/e2estudiante2/bp0/ejer4
Cola: ac
Nodo que ejecuta este trabajo: atcgrid1
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):
2:!!!Hello!!!(2:!!!World!!!)(0:!!!Hello!!!)(0:!!!World!!!)(10:!!!Hello!!!)(10:!!!World!!!)(1:!!!Hello!!!)(1:!!!World!!!)(6:!!!Hello!!!)(6:!!!World!!!)(7:!!!Hello!!!)(7:!!!World!!!)(4:!!!Hello!!!)(4:!!!World!!!)(5:!!!Hello!!!)(5:!!!World!!!)(9:!!!Hello!!!)(9:!!!World!!!)(3:!!!Hello!!!)(3:!!!World!!!)(11:!!!Hello!!!)(11:!!!World!!!)(8:!!!Hello!!!)(8:!!!World!!!)
2. Ejecución helloOMP varias veces con distinto nº de threads:

- Para 12 threads:
0:!!!Hello!!!(0:!!!World!!!)(8:!!!Hello!!!)(8:!!!World!!!)(2:!!!Hello!!!)(2:!!!World!!!)(5:!!!Hello!!!)(5:!!!World!!!)(9:!!!Hello!!!)(9:!!!World!!!)(6:!!!Hello!!!)(6:!!!World!!!)(7:!!!Hello!!!)(7:!!!World!!!)(10:!!!Hello!!!)(10:!!!World!!!)(3:!!!Hello!!!)(3:!!!World!!!)(4:!!!Hello!!!)(4:!!!World!!!)(1:!!!Hello!!!)(1:!!!World!!!)(11:!!!Hello!!!)(11:!!!World!!!)
- Para 6 threads:
1:!!!Hello!!!(1:!!!World!!!)(5:!!!Hello!!!)(5:!!!World!!!)(4:!!!Hello!!!)(4:!!!World!!!)(2:!!!Hello!!!)(2:!!!World!!!)(3:!!!Hello!!!)(3:!!!World!!!)(0:!!!Hello!!!)(0:!!!World!!!)
- Para 3 threads:
1:!!!Hello!!!(1:!!!World!!!)(2:!!!Hello!!!)(2:!!!World!!!)(0:!!!Hello!!!)(0:!!!World!!!)
- Para 1 threads:
0:!!!Hello!!!(0:!!!World!!!)[MaríaAguado e2estudiante2@atcgrid:~/bp0/ejer4] 2020-03-01 domingo
$
```

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

RESPUESTA:

El nodo que ha ejecutado el script es el nodo atcgrid1, como podemos observar si leemos el archivo de salida de la ejecución anterior (ver captura anterior).

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:

```

MariaAguado@maguado-HP-Pavillon-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer7$ 2020-02-27 jueves
$gcc -O2 sumavectores.c -o SumaVectores_loc -lrt
MariaAguado@maguado-HP-Pavillon-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer7$ 2020-02-27 jueves
$chmod +rxw SumaVectores_loc
MariaAguado@maguado-HP-Pavillon-x360-Convertible-14-ba1xx:~/Documentos/DGIIM2/DGIIM2/2/AC/bp0/ejer7$ 2020-02-27 jueves
$./SumaVectores_loc 38
Tamaño Vectores:38 (4 B)
Tiempo:0.000000460 / Tamaño Vectores:38 / V1[0]+V2[0]=V3[0](3.800000+3.800000=7.600000) / / V1[37]+V2[37]=V3[37](7.500000+0.100000=7.600000) /

```

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:

La variable `ncgt` contiene el tiempo que se ha tardado en ejecutar el bucle `for`, es decir, la diferencia entre los tiempos representados por `cgt1` y `cgt2`.

- (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:

Se devuelve en una estructura llamada `timespec`, que contiene el tiempo transcurrido desde el 1 de enero de 1970 hasta el actual. Se conforma de dos campos, el primero almacena el número de segundos (`time_t`) y el segundo el número de nanosegundos(`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 ya he mencionado, devuelve la medición de tiempo desde el 01/01/1970, almacenando en `tv_sec` el número de segundos y en `tv_nsec` el número de nanosegundos.

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 . TABLA DEL PC

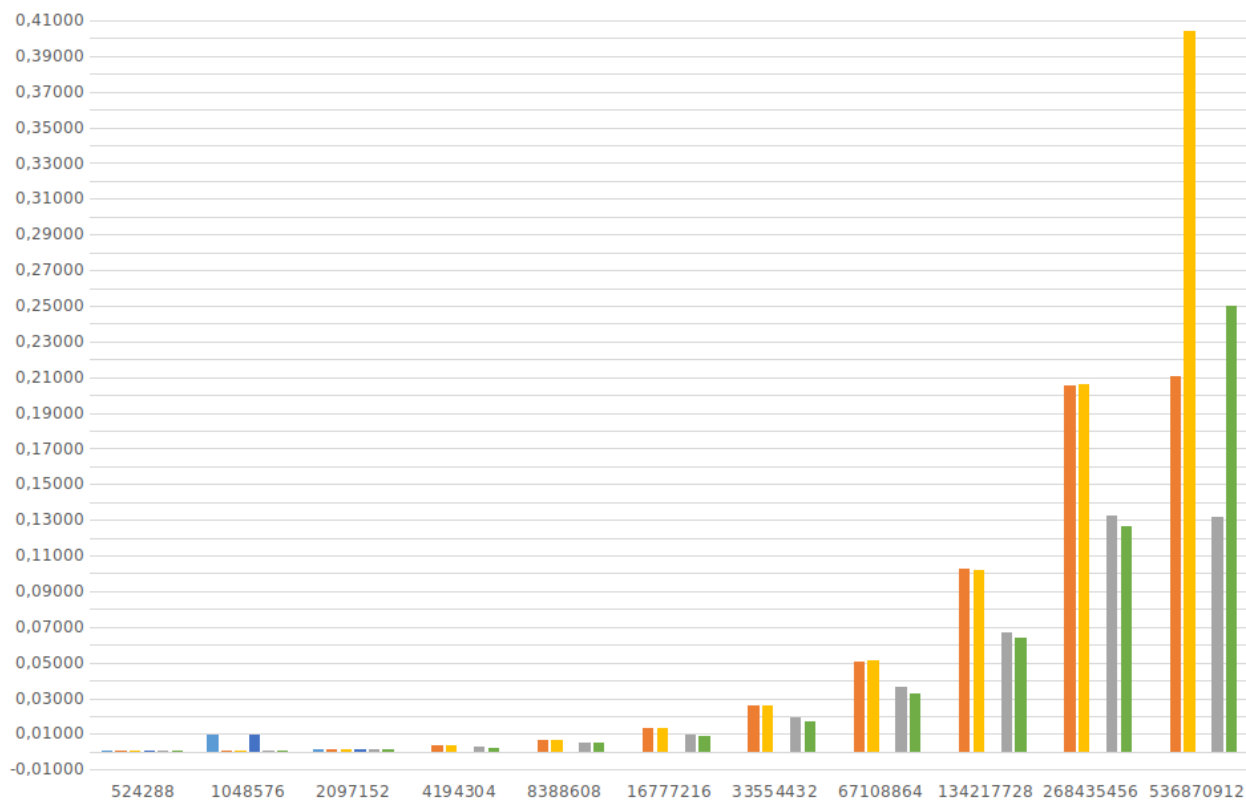
Nº de Componentes	Bytes de un vector	Tiempo para vector local	Tiempo para vector global	Tiempo para vector dinámico
65536	524288	0.000769465	0.000389026	0.000989366
131072	1048576	0.00990347	0.000799095	0.000775098
262144	2097152	0.001178043	0.001599682	0.001577445
524288	4194304	CORE	0.003343599	0.003294619
1048576	8388608	CORE	0.006559414	0.006673985
2097152	16777216	CORE	0.013090321	0.013491148
4194304	33554432	CORE	0.025915476	0.025733662
8388608	67108864	CORE	0.050796960	0.050921144
16777216	134217728	CORE	0.102524590	0.102025548
33554432	268435456	CORE	0.205260097	0.206425768
67108864	536870912	CORE	0.210375211	0.403702962

Tabla 2 . TABLA DEL ATCGRID

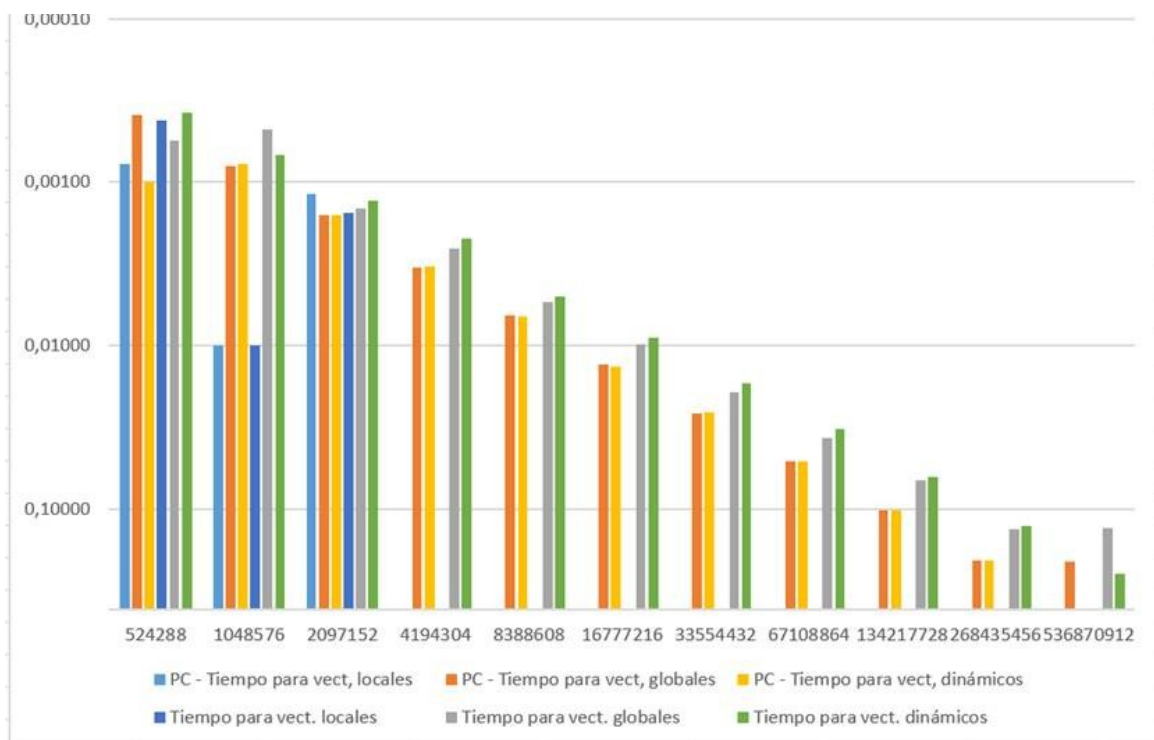
Nº de Componentes	Bytes de un vector	Tiempo para vector local	Tiempo para vector global	Tiempo para vector dinámico
65536	524288	0.000421774	0.000558406	0.000375303
131072	1048576	0.00990347	0.000475119	0.000676183
262144	2097152	0.001535539	0.001439079	0.001300848
524288	4194304	CORE	0.002547102	0.002212443
1048576	8388608	CORE	0.005394058	0.004974851
2097152	16777216	CORE	0.009776898	0.008985083
4194304	33554432	CORE	0.019236193	0.017043057
8388608	67108864	CORE	0.036451562	0.032496980
16777216	134217728	CORE	0.066956614	0.063680829
33554432	268435456	CORE	0.132072706	0.126381743
67108864	536870912	CORE	0.131318266	0.250147421

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:



Con la escala logarítmica:



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:

```

VERSIÓN LOCAL

Tamaño Vectores:65536 (4 B)
Tiempo:0.000769465 / 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:0.0009998347 / 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:0.001178043 / 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)
./script.sh: línea 12: 16444 Violación de segmento ('core' generado) ./SumaVectores_loc $N
Tamaño Vectores:1048576 (4 B)
./script.sh: línea 12: 16446 Violación de segmento ('core' generado) ./SumaVectores_loc $N
Tamaño Vectores:2097152 (4 B)
./script.sh: línea 12: 16448 Violación de segmento ('core' generado) ./SumaVectores_loc $N
Tamaño Vectores:4194304 (4 B)
./script.sh: línea 12: 16450 Violación de segmento ('core' generado) ./SumaVectores_loc $N
Tamaño Vectores:8388608 (4 B)
./script.sh: línea 12: 16452 Violación de segmento ('core' generado) ./SumaVectores_loc $N
Tamaño Vectores:16777216 (4 B)
./script.sh: línea 12: 16454 Violación de segmento ('core' generado) ./SumaVectores_loc $N
Tamaño Vectores:33554432 (4 B)
./script.sh: línea 12: 16456 Violación de segmento ('core' generado) ./SumaVectores_loc $N
Tamaño Vectores:67108864 (4 B)
./script.sh: línea 12: 16458 Violación de segmento ('core' generado) ./SumaVectores_loc $N

```

En los vectores locales se produce una violación de segmento a partir del tamaño 52428, pues utilizar este tamaño supondría usar $512K \times 8 = 2^{22} = 4M$ por cada vector, y al ser tres vectores, tendríamos 12M de pila. Sin embargo, analizando la pila con la orden `ulimit -a`, vemos que solamente tenemos 8M. Por tanto, como las variables locales se almacenan en la pila, no hay espacio suficiente para almacenar los tres vectores.

(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:

```

VERSIÓN GLOBAL

Tamaño Vectores:65536 (4 B)
Tiempo:0.000389026 / 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:0.000799095 / 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:0.00159062 / 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:0.003343599 / 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:0.006559414 / 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:0.013098321 / 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:0.025915476 / 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:0.050796969 / 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:0.102524590 / 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:0.205260097 / 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:0.210375211 / 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=13421772.800000) /

```

Con los vectores globales, no tenemos ningún error de memoria, pues para las variables globales se reserva un espacio en una zona concreta del programa, para vectores con MAX elementos. Sin embargo, cabe destacar que en el último tamaño N utilizado, al ser éste mayor que MAX, no reserva N componentes, sino que mantiene MAX.

(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:

```

VERSIÓN DINÁMICA

Tamaño Vectores:65536 (4 B) / 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) /
Tiempo:0.000406965 / Tamaño Vectores:131072 (4 B) / 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) / V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) / / V1[262143]+V2[262143]=V3[262143](52428.700000+0.100000=52428.800000) /
Tiempo:0.001597253 / Tamaño Vectores:524288 (4 B) / 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) / V1[0]+V2[0]=V3[0](104857.600000+104857.600000=209715.200000) / / V1[1048575]+V2[1048575]=V3[1048575](209715.100000+0.100000=209715.200000) /
Tiempo:0.006658841 / Tamaño Vectores:2097152 (4 B) / 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) / V1[0]+V2[0]=V3[0](419430.400000+419430.400000=838860.800000) / / V1[4194303]+V2[4194303]=V3[4194303](838860.700000+0.100000=838860.800000) /
Tiempo:0.025759380 / Tamaño Vectores:8388608 (4 B) / 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) / V1[0]+V2[0]=V3[0](1677721.600000+1677721.600000=3355443.200000) / / V1[16777215]+V2[16777215]=V3[16777215](3355443.100000+0.100000=3355443.200000) /
Tiempo:0.101933749 / Tamaño Vectores:33554432 (4 B) / 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) / V1[0]+V2[0]=V3[0](6710886.400000+6710886.400000=13421772.800000) / / V1[67108863]+V2[67108863]=V3[67108863](13421772.700000+0.100000=13421772.800000) /
Tiempo:0.403489608 / Tamaño Vectores:134217728 (4 B) / V1[0]+V2[0]=V3[0](13421772.800000+13421772.800000=268435456.000000) / / V1[134217727]+V2[134217727]=V3[134217727](268435455.900000+0.100000=268435456.000000) /

```

En los vectores dinámicos no se produce ningún tipo de error, pues la reserva se produce en tiempo de ejecución, y se almacenan en el heap, situado en la RAM.

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

RESPUESTA: $2^{32} - 1$

(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:

En la versión global de un error, que se llama reubicación truncada, que viene del ensamblador, pues estamos intentando coger un espacio de memoria superior al que está disponible, al declarar el vector.

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)

#ifndef 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;
}

```