2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 1. Programación paralela I: Directivas OpenMP

Estudiante (nombre y apellidos): Antonio Gámiz Delgado
Grupo de prácticas: B
Fecha de entrega:
Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

1. 1.Usar la directiva `parallel` combinada con directivas de trabajo compartido en los ejemplos `bucle-for.c` y `sections.c` del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

```c
// Antonio Gamiz Delgado

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

int main (int argc, char **argv)
{
    int i, n = 9;
    if(argc < 2)
    {
        fprintf(stderr, "\n[ERROR] - Falta nº de iteraciones\n");
        exit(-1);
    }

    n = atoi(argv[1]);

    #pragma omp parallel for
        for(i=0;i < n; i++)
            printf(" thread %d ejecuta la iteracion %d del bucle \n",
            omp_get_thread_num(), i);
    return(0);
}
```

```c
// Antonio Gamiz Delgados

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

void funcA()
{
    printf("En funcA: esta seccion la ejecuta el thread %d\n", omp_get_thread_num());
}

void funcB()
{
    printf("En funcB: esta seccion la ejecuta el thread %d\n", omp_get_thread_num());
}

main()
{
    #pragma omp parallel sections
    {
        #pragma omp section
            (void) funcA();
        #pragma omp section
            (void) funcB();
    }
}
```

. 2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

```c
//Antonio Gamiz Delgado

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

main()
{
    int n=9, i, a, b[n];

    for(i=0; i<n; i++) b[i]=-1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce el valor de inicializacion a: ");
            scanf("%d", &a);
            printf("Single ejecutada por el thread %d\n",
            omp_get_thread_num());
        }

        #pragma omp for
        for(i=0; i<n; i++)
            b[i]=a;

        #pragma omp single
        {
            printf("Despues de la region parallel: \n");
            for(i=0;i<n; i++) printf("b[%d]=%d\t",i,b[i]);
            printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}

```

```
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ gcc -O2 -fopenmp ./source/singleModificado.c -o ./bin/singleModificado
./source/singleModificado.c:4:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ export OMP_DYNAMIC=FALSE
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ export OMP_NUM_THREADS=8
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ ./bin/singleModificado
Introduce el valor de inicializacion a: 5
Single ejecutada por el thread 0
Despues de la region parallel:
b[0]=5  b[1]=5  b[2]=5  b[3]=5  b[4]=5  b[5]=5  b[6]=5  b[7]=5  b[8]=5
Single ejecutada por el thread 2
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ ./bin/singleModificado
Introduce el valor de inicializacion a: 4
Single ejecutada por el thread 7
Despues de la region parallel:
b[0]=4  b[1]=4  b[2]=4  b[3]=4  b[4]=4  b[5]=4  b[6]=4  b[7]=4  b[8]=4
Single ejecutada por el thread 4
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$
```

. 3.Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `master`. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

```
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ gcc -O2 -fopenmp ./source/singleModificado2.c -o ./bin/singleModificado2
./source/singleModificado2.c:6:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ export OMP_DYNAMIC=FALSE
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ export OMP_NUM_THREADS=8
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ ./bin/singleModificado2
Introduce el valor de inicializacion a: 34
Single ejecutada por el thread 6
Despues de la region parallel:
b[0]=34 b[1]=34 b[2]=34 b[3]=34 b[4]=34 b[5]=34 b[6]=34 b[7]=34 b[8]=34
Single ejecutada por el thread 0
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ ./bin/singleModificado2
Introduce el valor de inicializacion a: 345
Single ejecutada por el thread 6
Despues de la region parallel:
b[0]=345        b[1]=345        b[2]=345        b[3]=345        b[4]=345        b[5]=345        b[6]=345        b[7]=345        b[8]=345
Single ejecutada por el thread 0
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$
```

```c
//Antonio Gamiz Delgado

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

main()
{
    int n=9, i, a, b[n];

    for(i=0; i<n; i++) b[i]=-1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce el valor de inicializacion a: ");
            scanf("%d", &a);
            printf("Single ejecutada por el thread %d\n",
            omp_get_thread_num());
        }

        #pragma omp for
        for(i=0; i<n; i++)
            b[i]=a;

        #pragma omp master
        {
            printf("Despues de la region parallel: \n");
            for(i=0;i<n; i++) printf("b[%d]=%d\t",i,b[i]);
            printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}
```

Vemos que al añadir la directiva master, la hebra que siempre ejecuta el print de salida es la hebra 0, es decir, la master.

. 4. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

RESPUESTA:

Si eliminámaramos la directiva barrier del programa, los threads no se sincronizarían correctamente, es decir, como los threads se ejecutan en ordénes aleatorioas, podría darse el caso de que el primer thread que terminara fuera el 0, sin haber terminado los otros, por lo que el resultado que imprimiría sería incorrecto por los demás threads no han actuado todavía sobre la variable suma.

En cambio, si ponemos la directiva barrier, expresamos explicitamente que los threads que terminen antes se esperen hasta que terminen los demás, y así obtener el resultado correcto.

Resto de ejercicios

. 5. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener, en atcgrid, el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema generado. Obtenga los tiempos para vectores con 10000000 componentes. ¿La suma de los tiempos de CPU del usuario y del sistema es menor, mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ gcc -O2 ./source/listado1.c -o ./bin/listado1 -lrt
```

Compilamos el programa.

```
sftp> lcd ArquitecturaDeComputadores/Práctica\ 2/
sftp> lpwd
Local working directory: /home/antonio/ArquitecturaDeComputadores/Práctica 2
sftp> pwd
Remote working directory: /home/E2estudiante14
sftp> cd practica2/
sftp> ls
sftp> put ./bin/listado1
Uploading ./bin/listado1 to /home/E2estudiante14/practica2/listado1
./bin/listado1                                                100% 8968     8.8KB/s   00:00
sftp>
```

Subimos el programa a atcgrid.

```
[E2estudiante14@atcgrid practica2]$ ls
listado1
[E2estudiante14@atcgrid practica2]$ echo 'time practica2/listado1 10000000' | qsub -q ac
68311.atcgrid
[E2estudiante14@atcgrid practica2]$
```

Ejecutamos la orden ‘time’,

```
sftp> get STDIN.o68311
Fetching /home/E2estudiante14/practica2/STDIN.o68311 to STDIN.o68311
/home/E2estudiante14/practica2/STDIN.o68311                   100%  205     0.2KB/s   00:00
sftp> get STDIN.e68311
Fetching /home/E2estudiante14/practica2/STDIN.e68311 to STDIN.e68311
/home/E2estudiante14/practica2/STDIN.e68311                   100%   42     0.0KB/s   00:00
sftp>
```

Traemos los ficheros desde atcgrid a local.

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ cat ./outputs/STDIN.o68311
Tiempo (seg.): 0.054919933      / Tamaño vectores: 10000000    / v1[0]+v2[0]=v3[0](1000000.000000+1000000.000000=2000
000.000000) / / v1[9999999]+v2[9999999]=v3[9999999](1999999.900000+0.100000=2000000.000000) /
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ cat ./outputs/STDIN.e68311

real    0m0.163s
user    0m0.063s
sys     0m0.096s
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶
```

Como vemos, la orden ‘time’ muestra su resultado en la salido de error.

Vemos que el CPU time es 0.159s (=user+sys=0.063+0.096), que es mayor que el tiempo real, 0.163s.

Esto es debido a que el tiempo ‘user’ indica el tiempo que ha pasado el programa en tiempo de ejecución de espacio del usuario, y ‘sys’ en el nivel kernel del SO, mientras que el elapsed time también tiene ese valor más el asociado a interrupciones que sufre el programa, ya sea por la espera de I/O o ejecuaciones de otros procesos del SO.

. 6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando `-S` en lugar de `-o`). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of FLOating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

CAPTURAS DE PANTALLA:

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ gcc -O2 -S ./source/listado1.c
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ ls
bin  guion  listado1.s  outputs  screenshots  source
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶
```

```
[E2estudiante14@atcgrid practica2]$ echo 'time practica2/listado1 10' | qsub -q ac
68312.atcgrid
[E2estudiante14@atcgrid practica2]$
```

```
sftp> get STDIN.o68312
Fetching /home/E2estudiante14/practica2/STDIN.o68312 to STDIN.o68312
/home/E2estudiante14/practica2/STDIN.o68312                   100%  151     0.2KB/s   00:00
sftp> get STDIN.e68312
Fetching /home/E2estudiante14/practica2/STDIN.e68312 to STDIN.e68312
/home/E2estudiante14/practica2/STDIN.e68312                   100%   42     0.0KB/s   00:00
sftp>
```

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ cat ./outputs/STDIN.o68312
Tiempo (seg.): 0.000002262      / Tamaño vectores: 10  / v1[0]+v2[0]=v3[0](1.000000+1.000000=2.000000) / / v1[9]+v2[9
]=v3[9](1.900000+0.100000=2.000000) /
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ cat ./outputs/STDIN.e68312

real    0m0.004s
user    0m0.001s
sys     0m0.000s
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶
```

Generamos el código ensamblador asociado al listado1.c con la opción -O2.

Ejecutamos el programa listado1 con n=10 ya que para el caso 10000000 podemos usar los datos del ejercicio 5

RESPUESTA: cálculo de los MIPS y los MFLOPS

Para el cálculo de los MIPS necesitamos el número de instrucciones, por lo que contamos el número de líneas del código ensamblador que se ejecuta.

He contado 40 líneas (.L6), por lo que la fórmula de los MIPS de teoría nos dice que:

MIPS(n=10)=40/(0.001*10^6)=0,04 MIPS

MIPS(n=10000000)=40/(0.159*10^6)=0,0003 MIPS

Las MFLOPS no se cómo calcularlas debido a que no se cuántas operaciones en coma flotante se realiazan. Si las supiera, aplicaria la misma fórmula de antes pero sustituyendo el número de instrucciones por la cantidad de operaciones en coma flotante.

RESPUESTA: Captura que muestre el código ensamblador generado de la parte de la suma de vectores

```
                call    clock_gettime
                xorl    %eax, %eax
                .p2align 4,,10
                .p2align 3
.L5:
                movsd   v1(%rax), %xmm0
                addq    $8, %rax
                addsd   v2-8(%rax), %xmm0
                movsd   %xmm0, v3-8(%rax)
                cmpq    %rax, %rbx
                jne     .L5
.L6:
                leaq    16(%rsp), %rsi
                xorl    %edi, %edi
                call    clock_gettime
                movq    24(%rsp), %rax
                subq    8(%rsp), %rax
                movl    %r12d, %edx
                pxor    %xmm0, %xmm0
                movl    %r12d, %ecx
                movsd   v3(,%rdx,8), %xmm6
                movl    %r12d, %r9d
                movsd   v2(,%rdx,8), %xmm5
                movl    %r12d, %r8d
                cvtsi2sdq %rax, %xmm0
                movq    16(%rsp), %rax
                subq    (%rsp), %rax
                movsd   v1(,%rdx,8), %xmm4
                movsd   v3(%rip), %xmm3
                movl    %ebp, %edx
                movsd   v2(%rip), %xmm2
                movl    $.LC3, %esi
                movl    $1, %edi
                movapd  %xmm0, %xmm1
                pxor    %xmm0, %xmm0
                divsd   .LC2(%rip), %xmm1
                cvtsi2sdq %rax, %xmm0
```

```
                movl    $7, %eax
                addsd   %xmm1, %xmm0
                movsd   v1(%rip), %xmm1
                call    __printf_chk
                xorl    %eax, %eax
                movq    40(%rsp), %rcx
                xorq    %fs:40, %rcx
                jne     .L15
                addq    $48, %rsp
                .cfi_remember_state
                .cfi_def_cfa_offset 32
                popq    %rbx
                .cfi_def_cfa_offset 24
                popq    %rbp
                .cfi_def_cfa_offset 16
                popq    %r12
                .cfi_def_cfa_offset 8
                ret
.L3:
                .cfi_restore_state
                movq    %rsp, %rsi
                xorl    %edi, %edi
                orl     $-1, %r12d
                call    clock_gettime
```

. 7. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función `omp_get_wtime()`, que proporciona el estándar OpenMP, en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para varios tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: Captura que muestre el código fuente implementado

```
//Inicializar vectores
#pragma omp parallel for
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
}

#pragma omp barrier

double start = omp_get_wtime();

//Calcular suma de vectores
#pragma omp parallel for
for(i= 0; i< N; i++)
  v3[i]= v1[i] + v2[i];
#pragma omp barrier

double end = omp_get_wtime();

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
printf("Tiempo(seg): %11.9f\t / tamaño vectores: %u\n", (float)(end-start), 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]);
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ ./bin/listado1_omp 8
Tiempo(seg): 0.000018881         / tamaño vectores: 8
/V1[0] + V2[0] = V3[0] (0.800000 + 0.800000 = 1.600000) /
/V1[1] + V2[1] = V3[1] (0.900000 + 0.700000 = 1.600000) /
/V1[2] + V2[2] = V3[2] (1.000000 + 0.600000 = 1.600000) /
/V1[3] + V2[3] = V3[3] (1.100000 + 0.500000 = 1.600000) /
/V1[4] + V2[4] = V3[4] (1.200000 + 0.400000 = 1.600000) /
/V1[5] + V2[5] = V3[5] (1.300000 + 0.300000 = 1.600000) /
/V1[6] + V2[6] = V3[6] (1.400000 + 0.200000 = 1.600000) /
/V1[7] + V2[7] = V3[7] (1.500000 + 0.100000 = 1.600000) /
```

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ ./bin/listado1_omp 11
Tiempo(seg): 0.000005690         / tamaño vectores: 11
/V1[0] + V2[0] = V3[0] (1.100000 + 1.100000 = 2.200000) /
/V1[1] + V2[1] = V3[1] (1.200000 + 1.000000 = 2.200000) /
/V1[2] + V2[2] = V3[2] (1.300000 + 0.900000 = 2.200000) /
/V1[3] + V2[3] = V3[3] (1.400000 + 0.800000 = 2.200000) /
/V1[4] + V2[4] = V3[4] (1.500000 + 0.700000 = 2.200000) /
/V1[5] + V2[5] = V3[5] (1.600000 + 0.600000 = 2.200000) /
/V1[6] + V2[6] = V3[6] (1.700000 + 0.500000 = 2.200000) /
/V1[7] + V2[7] = V3[7] (1.800000 + 0.400000 = 2.200000) /
/V1[8] + V2[8] = V3[8] (1.900000 + 0.300000 = 2.200000) /
/V1[9] + V2[9] = V3[9] (2.000000 + 0.200000 = 2.200000) /
/V1[10] + V2[10] = V3[10] (2.100000 + 0.100000 = 2.200000) /
```

. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las `parallel` y `sections/section` (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva `for`); es decir, hay que repartir el trabajo (tareas) entre varios threads usando `sections/section`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función `omp_get_wtime()` en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: Captura que muestre el código fuente implementado

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA:

. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan

los códigos. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) ¿?threads/cores	T. paralelo (versión `sections`) ¿?threads/cores
16384			
32768			
65536			
131072			
262144			
524288			
1048576			
2097152			
4194304			
8388608			
16777216			
33554432			
67108864			

RESPUESTA:

. Rellenar una tabla como la Tabla 3 para atcgrid con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.
RESPUESTA:

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión `for` ¿? Threads/cores		
	Elapsed	*CPU-user*	*CPU- sys*	*Elapsed*	*CPU-user*	*CPU- sys*
65536						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						