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): Elena Ortiz Moreno
Grupo de prácticas y profesor de prácticas: Niceto Rafael Luque Sola B2
Fecha de entrega: 07/04/21
Fecha evaluación en clase: 09/04/21

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

Ejercicios basados en los ejemplos del seminario práctico

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.

RESPUESTA: Captura que muestre el código fuente `bucle-forModificado.c`

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

RESPUESTA: Captura que muestre el código fuente `sectionsModificado.c`

```
#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.

RESPUESTA: Captura que muestre el código fuente `singleModificado.c`

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

int 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 valor de inicialización 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("Depués de la región parallel:\n", omp_get_thread_num());
    for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
    printf("\n");
  }
}
```

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer2$ gcc -O2 -fopenmp singleModificado.c -o singleMod
singleModificado.c: In function ‘main’:
singleModificado.c:24:10: warning: too many arguments for format [-Wformat-extra-args]
   printf("Depués de la región parallel:\n", omp_get_thread_num());
          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer2$ ./singleMod
Introduce valor de inicialización a: 10
Single ejecutada por el thread 1
Depués de la región parallel:
b[0] = 10       b[1] = 10       b[2] = 10       b[3] = 10       b[4] = 10       b[5] = 10       b[6] = 10       b[7] = 10       b[8] = 10
```

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?

   RESPUESTA: Captura que muestre el código fuente `singleModificado2.c`

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

int 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 valor de inicialización 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("(ImprimeValor) single ejecutada por el thread %d\n", omp_get_thread_num());
      for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
      printf("\n");
    }
  }
}
```

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer3$ gcc -O2 -fopenmp singleModificado2.c -o singleMod2
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer3$ ./singleMod2
Introduce valor de inicialización a: 10
Single ejecutada por el thread 0
(ImprimeValor) single ejecutada por el thread 0
b[0] = 10       b[1] = 10       b[2] = 10       b[3] = 10       b[4] = 10       b[5] = 10       b[6] = 10       b[7] = 10       b[8] = 10
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer3$ ./singleMod2
Introduce valor de inicialización a: 10
Single ejecutada por el thread 2
(ImprimeValor) single ejecutada por el thread 0
b[0] = 10       b[1] = 10       b[2] = 10       b[3] = 10       b[4] = 10       b[5] = 10       b[6] = 10       b[7] = 10       b[8] = 10
```

RESPUESTA A LA PREGUNTA:

Como vemos en las capturas anteriores, en este caso al utilizar la directiva master para e último single, esta parte siempre será ejecutada por la hebra 0 (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:

Porque al no indicar la barrera, no habría una barrera implícita tras atomic y se puede ejecutar el printf antes de terminar todas las sumas, por lo que el resultado sería incorrecto.

Comprobación:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer4$ gcc -O2 -fopenmp masterSinBarrier.c -o masterSinBarrier
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer4$ ./masterSinBarrier 10
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 2 suma de a[6]=6 sumalocal=6
 thread 2 suma de a[7]=7 sumalocal=13
 thread 1 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[4]=4 sumalocal=7
 thread 1 suma de a[5]=5 sumalocal=12
 thread 3 suma de a[8]=8 sumalocal=8
 thread 3 suma de a[9]=9 sumalocal=17
thread master=0 imprime suma=28
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer4$ ./masterSinBarrier 5
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 1 suma de a[2]=2 sumalocal=2
 thread 2 suma de a[3]=3 sumalocal=3
 thread 3 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=1
```

1.1.1

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

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:

Ejecución en atcgrid de SumaVectoresC:

```
[b2estudiante27@atcgrid ejer5]$ sbatch -p ac --wrap "time ~/BP1/ejer5/SumaVectoresC 10000000"
Submitted batch job 67235
```

Archivo slurm-67235.out:

Resultado ejecución:

```
Tama◆o Vectores:10000000 (4 B)
Tiempo:0.042454105       / Tama◆o Vectores:10000000     / V1[0]+V2[0]=V3[0]
(1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999]
(1999999.900000+0.100000=2000000.000000) /
```

Tiempos empleados:

```
real    0m0.156s
user    0m0.067s
sys     0m0.045s
```

RESPUESTA:

La suma de user+sys = 0m0.067+0m0.045 = 0m0.112s es menor a los 0m0.156seg del tiempo real de ejecución(elapsed time) porque en esta suma no se cuentan las llamadas al SO y algunos procesos más.

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). Razonar cómo se han obtenido los valores que se necesitan para calcular los MIPS y MFLOPS. Incorporar **el código ensamblador de la parte de la suma de vectores** (no de todo el programa) en el cuaderno.

   CAPTURAS DE PANTALLA (que muestren la generación del código ensamblador y del código ejecutable, y la obtención de los tiempos de ejecución):

   Generación del código ensamblador:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer6$ gcc -O2 -fopenmp SumaVectoresC.c -S
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:34: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has type ‘long unsigned int’ [-Wformat
=]
   printf("Tama@o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                  ~^
                                  %lu
```

   Generación del código ejecutable:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer6$ gcc -O2 -fopenmp SumaVectoresC.c -o SumaVectoresC
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:34: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has type ‘long unsigned int’ [-Wformat
=]
   printf("Tama@o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                  ~^
                                  %lu
```

   Obtención de tiempos para 10 componentes:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer6$ time ./SumaVectoresC 10
Tama@o Vectores:10 (4 B)
Tiempo:0.000007144       / Tama@o 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) /

real    0m0,003s
user    0m0,002s
sys     0m0,001s
```

   Obtención de tiempos para 10000000 componentes:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer6$ time ./SumaVectoresC 10000000
Tama@o Vectores:10000000 (4 B)
Tiempo:0.040686526       / Tama@o Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]+
V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0,119s
user    0m0,036s
sys     0m0,083s
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Teniendo en cuenta que mi ordenador tiene 2’50GHz (F), sabemos que puede calcular un máximo de 2’5 IPC

CPI = 1/IPC = 0’4

MIPS = $F/CPI*10^9 = 2’5*10^9/0’4*10^9$

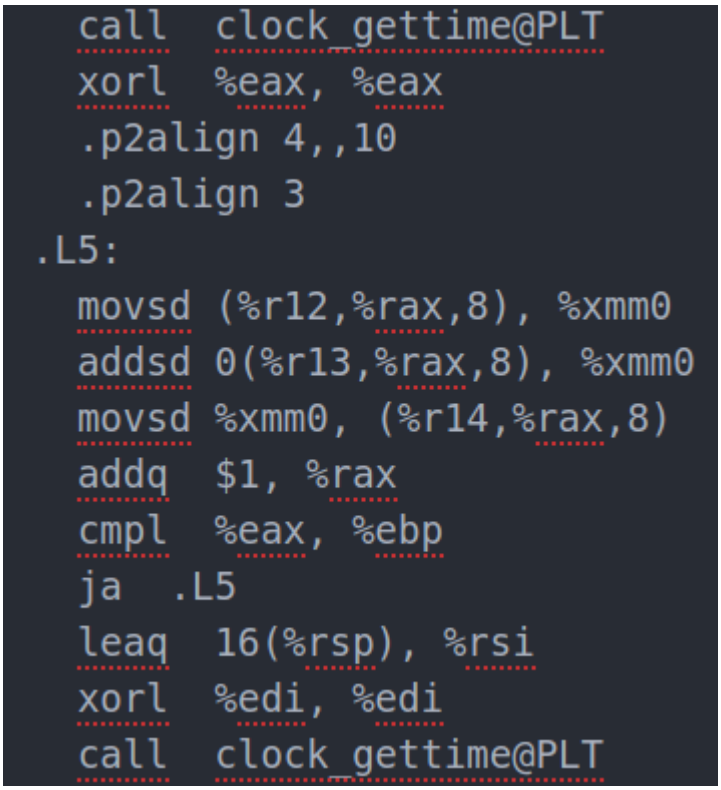
Tenemos 4 operaciones en coma flotante: movsd, addsd, movsd y compl.

GFLOPS = (op_coma_flotante * n_iteraciones) / T

10 componentes → 4 * 10 / 0’000007144 = 5599104’143

10000000 componentes → 4 * 10000000 / 0’040686526 = 983126453’2

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



```
  call  clock_gettime@PLT
  xorl  %eax, %eax
  .p2align 4,,10
  .p2align 3
.L5:
  movsd (%r12,%rax,8), %xmm0
  addsd 0(%r13,%rax,8), %xmm0
  movsd %xmm0, (%r14,%rax,8)
  addq  $1, %rax
  cmpl  %eax, %ebp
  ja  .L5
  leaq  16(%rsp), %rsi
  xorl  %edi, %edi
  call  clock_gettime@PLT
```

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 `sp-OpenMP-for.c`

```
//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;
    //printf("Ejecutado por hebra %d\n", omp_get_thread_num());
  }

printf("v1[0]=%11.9f\nv1[N-1]=%11.9f\n",v1[0], v1[N-1]);
printf("v2[0]=%11.9f\nv2[N-1]=%11.9f\n",v2[0], v2[N-1]);
//método clock gettime --> preguntar el tiempo clockid(reloj para el que quieres que calcule el tiempo)
double t1 = omp_get_wtime();
//Calcular suma de vectores
#pragma omp parallel for
  for(i=0; i<N; i++){
    v3[i] = v1[i] + v2[i];
    //printf("Ejecutado por hebra (suma) %d\n", omp_get_thread_num());
  }
double t2 = omp_get_wtime();
//ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
//     (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
double tiempo = t2-t1;

//Imprimir resultado de la suma y el tiempo de ejecuci@n
if (N<10) {
printf("Tiempo:%20.9f\t / Tama@o Vectores:%u\n",tiempo,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:%20.9f\t / Tama@o 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",
  tiempo,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

Compilación:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer7$ gcc -fopenmp  -O2  sp-OpenMP-for.c -o sp-OpenMP-for
sp-OpenMP-for.c: In function ‘main’:
sp-OpenMP-for.c:47:34: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has type ‘long unsigned int’ [-Wformat=]
   printf("Tama@o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                 ~^
                                 %lu
```

Ejecución para N=8:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer7$ ./sp-OpenMP-for 8
Tama@o Vectores:8 (4 B)
v1[0]=0.800000000
v1[N-1]=1.500000000
v2[0]=0.800000000
v2[N-1]=0.100000000
Tiempo:          0.000056336      / Tama@o 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) /
```

Ejecución para N=11:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer7$ ./sp-OpenMP-for 11
Tama@o Vectores:11 (4 B)
v1[0]=1.100000000
v1[N-1]=2.100000000
v2[0]=1.100000000
v2[N-1]=0.100000000
Tiempo:          0.000033396      / Tama@o Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000)/ / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
```

8. 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); (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 `sp-OpenMP-sections.c`

```
//Inicializar vectores
#pragma omp parallel sections
{
  #pragma omp section
  {
    for(i=0; i<N/2; i++){
      v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
      //printf("Ejecutado por hebra %d\n", omp_get_thread_num());
    }
    printf("Ejecutado por hebra(1) %d\n", omp_get_thread_num());
  }
  #pragma omp section
  {
    for(i=N/2; i<N; i++){
      v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
      //printf("Ejecutado por hebra %d\n", omp_get_thread_num());
    }
    printf("Ejecutado por hebra(2) %d\n", omp_get_thread_num());
  }
}

printf("v1[0]=%11.9f\nv1[N-1]=%11.9f\n",v1[0], v1[N-1]);
printf("v2[0]=%11.9f\nv2[N-1]=%11.9f\n",v2[0], v2[N-1]);
```

```
//Calcular suma de vectores
#pragma omp parallel sections
{
  #pragma omp section
  for(i=0; i<N/2; i++){
    v3[i] = v1[i] + v2[i];
    //printf("Ejecutado por hebra %d\n", omp_get_thread_num());
  }
  #pragma omp section
  for(i=N/2; i<N; i++){
    v3[i] = v1[i] + v2[i];
    //printf("Ejecutado por hebra %d\n", omp_get_thread_num());
  }
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

Compilación:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer8$ gcc -fopenmp  -O2  sp-OpenMP-sections.c -o sp-OpenMP-sect
ions
sp-OpenMP-sections.c: In function ‘main’:
sp-OpenMP-sections.c:47:34: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has type ‘
long unsigned int’ [-Wformat=]
   printf("Tama@o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                  ~^
                                  %lu
```

Ejecución N=8:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer8$ ./sp-OpenMP-sections 8
Tama@o Vectores:8 (4 B)
Ejecutado por hebra(1) 3
Ejecutado por hebra(2) 1
v1[0]=0.800000000
v1[N-1]=1.500000000
v2[0]=0.800000000
v2[N-1]=0.100000000
Tiempo:          0.003272704      / Tama@o 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) /
```

Ejecución N=11:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP1/ejer8$ ./sp-OpenMP-sections 11
Tama@o Vectores:11 (4 B)
Ejecutado por hebra(1) 0
Ejecutado por hebra(2) 1
v1[0]=1.100000000
v1[N-1]=2.100000000
v2[0]=1.100000000
v2[N-1]=0.100000000
Tiempo:          0.000045478      / Tama@o Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000)/ / V1[10]+V
2[10]=V3[10](2.100000+0.100000=2.200000) /
```

9. ¿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. NOTA: Al contestar piense sólo en el código, no piense en el computador en el que lo va a ejecutar.

RESPUESTA:

En el ejercicio 7, al utilizar un for, el máximo será el número de hebras del que dispongamos, el trabajo se repartirá entre todas ellas.

En cambio, en el ejercicio 8, el máximo necesario será el número de secciones, aunque podemos disponer de tantas hebras como queramos, solo se utilizarían 2 en mi caso, pues lo he dividido en 2 secciones.

10. Rellenar una tabla como la Tabla 2Error: no se encontró el origen de la referencia 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. Escribir un script para realizar las ejecuciones necesarias utilizando como base el script del seminario de BP0 (se deben imprimir en el script al menos las variables de entorno que ya se imprimen en el script de BP0). 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 (use el máximo número de cores físicos del computador que como máximo puede aprovechar el código, no use un número de threads superior al número de cores físicos). Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado. Observar que el número de componentes en la tabla llega hasta **67108864.**

RESPUESTA: Captura del script implementado `sp-OpenMP-script10.sh`

```
#! / bin / bash
# Órdenes para el sistema de colas:
# 1. Asigna al trabajo un nombre
# SBATCH --nombre-trabajo = holaOMP
# 2. Asignar el trabajo a una cola (partición)
# SBATCH --partición = ac
# 2. Asignar el trabajo a un account
# SBATCH --cuenta = ac

# Obtener información de las variables del entorno del sistema de colas:
echo  " Id. usuario del trabajo: $ SLURM_JOB_USER "
echo  " Id. del trabajo: $ SLURM_JOBID "
echo  " Nombre del trabajo especificado por usuario: $ SLURM_JOB_NAME "
echo  " Directorio de trabajo (en el que se ejecuta el script):
$ SLURM_SUBMIT_DIR "
echo  " Cola: $ SLURM_JOB_PARTITION "
echo  " Nodo que ejecuta este trabajo: $ SLURM_SUBMIT_HOST "
echo  " No de nodos asignados al trabajo: $ SLURM_JOB_NUM_NODES "
echo  " Nodos asignados al trabajo: $ SLURM_JOB_NODELIST "
echo  " CPU por nodo: $ SLURM_JOB_CPUS_PER_NODE "

#para N potencia de 2 desde 2^16 a 2^26

echo ""
echo "SUMAVECTORES FOR: "
echo ""

for ((N=16384;N<67108865;N=N*2))
do
  ./sp-OpenMP-for $N >> datosFOR.txt
done

echo ""
echo "SUMAVECTORES SECTIONS: "
echo ""

for ((N=16384;N<67108865;N=N*2))
do
  ./sp-OpenMP-sections $N >> datosSECTIONS.txt
done

echo ""
echo "SUMAVECTORES SECUENCIAL: "
echo ""

for ((N=16384;N<67108865;N=N*2))
do
  ./SumaVectoresC $N >> datosSECUENCIAL.txt
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

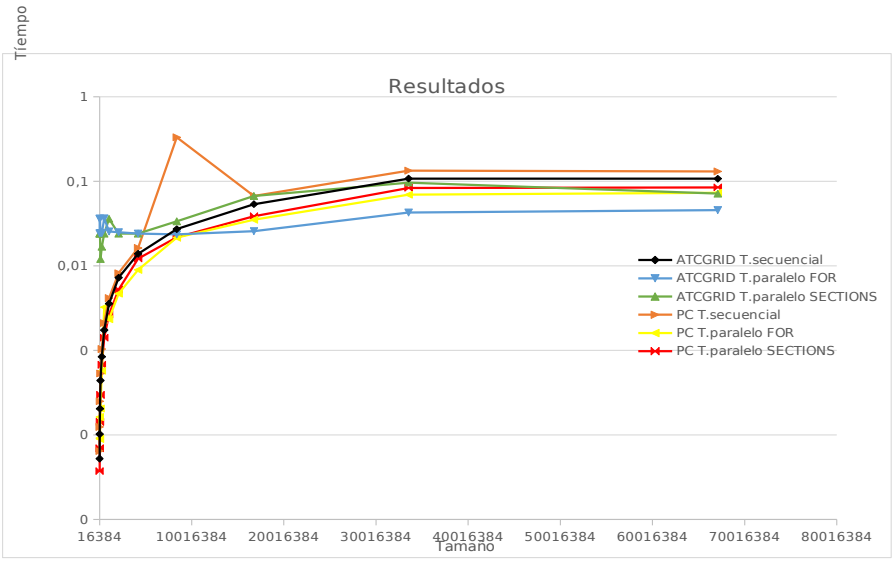
CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

```
[b2estudiante27@atcgrid ejer10]$ sh script.sh
 Id. usuario del trabajo: $ SLURM_JOB_USER
 Id. del trabajo: $ SLURM_JOBID
 Nombre del trabajo especificado por usuario: $ SLURM_JOB_NAME
 Directorio de trabajo (en el que se ejecuta el script):
$ SLURM_SUBMIT_DIR
 Cola: $ SLURM_JOB_PARTITION
 Nodo que ejecuta este trabajo: $ SLURM_SUBMIT_HOST
 No de nodos asignados al trabajo: $ SLURM_JOB_NUM_NODES
 Nodos asignados al trabajo: $ SLURM_JOB_NODELIST
 CPU por nodo: $ SLURM_JOB_CPUS_PER_NODE
```

******EJECUTAR LISTADO 1, EJER7 Y EJER8 EN MI PC Y EN ATCGRID**

RESULTADOS DE LAS EJECUCIONES EN EL PC			
Nº de Componentes	T.secuencial	T.paralelo FOR	T.paralelo SECTIONS
	1 thread/core	8 thread/ 4 cores	8 thread/ 4 cores
16384	6,4427E-05	9,1899E-05	3,7383E-05
32768	0,000124253	0,000162778	6,9509E-05
65536	0,000250559	8,9711E-05	0,000143043
131072	0,000532911	0,000205169	0,000297579
262144	0,001037255	0,00057333	0,000675834
524288	0,00210836	0,003257224	0,001412149
1048576	0,004147164	0,002338049	0,002668816
2097152	0,008204096	0,004708484	0,005246469
4194304	0,016264602	0,008966647	0,01221105
8388608	0,33167158	0,021659977	0,021736004
16777216	0,06727397	0,03514056	0,038601139
33554432	0,133767048	0,069371048	0,083688318
67108864	0,131165061	0,073408301	0,084888329

RESULTADOS DE LAS EJECUCIONES EN EL ATCGRID			
Nº de Componentes	T.secuencial	T.paralelo FOR	T.paralelo SECTIONS
	1 thread/core	24 thread/ 12 cores	24 thread/ 12 cores
16384	5,2306E-05	0,024260238	0,024204925
32768	0,000102349	0,036195356	0,024020419
65536	0,000204354	0,036172174	0,024060655
131072	0,000440063	0,024516881	0,012027457
262144	0,000843477	0,024339557	0,016816854
524288	0,001740623	0,036465995	0,024149653
1048576	0,003575966	0,025521882	0,036058258
2097152	0,00725248	0,024992675	0,024122383
4194304	0,01387663	0,0241131	0,024077654
8388608	0,027209301	0,023524757	0,033687167
16777216	0,053768779	0,02573783	0,067055982
33554432	0,107983342	0,042738255	0,096761093
67108864	0,107289898	0,04568556	0,071480528



11. Rellenar una tabla como la Error: no se encontró el origen de la referenciaTabla 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 (que debe coincidir con el número cores físicos y lógicos) que usan los códigos. Escribir un script para realizar las ejecuciones necesarias utilizando como base el script del seminario de BP0 (se deben imprimir en el script al menos las variables de entorno que ya se imprimen en el script de BP0) ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

    RESPUESTA: El tiempo de CPU en la versión secuencial es menor que el tiempo real debido a que solo se usa un procesador.

    En la versión paralela el tiempo de cpu es mayor que el tiempo real debido a que el tiempo de cpu es lasuma de los tiempos de cpu en cada uno de los procesadores, el tiempo real es lo que ha tardado en ejecutarse el programa.

    Captura del script implementado `sp-OpenMP-script11.sh`

```
#! / bin / bash
# Órdenes para el sistema de colas:
# 1. Asigna al trabajo un nombre
# SBATCH --nombre-trabajo = holaOMP
# 2. Asignar el trabajo a una cola (partición)
# SBATCH --partición = ac
# 2. Asignar el trabajo a un account
# SBATCH --cuenta = ac

# Obtener información de las variables del entorno del sistema de colas:
echo  " Id. usuario del trabajo: $ SLURM_JOB_USER "
echo  " Id. del trabajo: $ SLURM_JOBID "
echo  " Nombre del trabajo especificado por usuario: $ SLURM_JOB_NAME "
echo  " Directorio de trabajo (en el que se ejecuta el script):
$ SLURM_SUBMIT_DIR "
echo  " Cola: $ SLURM_JOB_PARTITION "
echo  " Nodo que ejecuta este trabajo: $ SLURM_SUBMIT_HOST "
echo  " No de nodos asignados al trabajo: $ SLURM_JOB_NUM_NODES "
echo  " Nodos asignados al trabajo: $ SLURM_JOB_NODELIST "
echo  " CPU por nodo: $ SLURM_JOB_CPUS_PER_NODE "

#para N potencia de 2 desde 2^16 a 2^26

echo ""
echo "SUMAVECTORES FOR: "
echo ""

for ((N=16384;N<67108865;N=N*2))
do
   time ./sp-OpenMP-for $N >> datosFOR.txt
done

echo ""
echo "SUMAVECTORES SECUENCIAL: "
echo ""

for ((N=16384;N<67108865;N=N*2))
do
  time ./SumaVectoresC $N >> datosSECUENCIAL.txt
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

```
[b2estudiante27@atcgrid ejer11]$ sh sp-OpenMP-script11.sh
 Id. usuario del trabajo: $ SLURM_JOB_USER
 Id. del trabajo: $ SLURM_JOBID
 Nombre del trabajo especificado por usuario: $ SLURM_JOB_NAME
 Directorio de trabajo (en el que se ejecuta el script):
$ SLURM_SUBMIT_DIR
 Cola: $ SLURM_JOB_PARTITION
 Nodo que ejecuta este trabajo: $ SLURM_SUBMIT_HOST
 No de nodos asignados al trabajo: $ SLURM_JOB_NUM_NODES
 Nodos asignados al trabajo: $ SLURM_JOB_NODELIST
 CPU por nodo: $ SLURM_JOB_CPUS_PER_NODE
```

Nº de Componentes	T.secuencial 1 thread/core			T.paralelo FOR 24 thread/ 12 cores		
	Elapsed	CPU-user	CPU-sys	Elapsed	CPU-user	CPU-sys
16384	0,001	0,000	0,001	0,003	0,002	0,003
32768	0,001	0,001	0,000	0,002	0,002	0,003
65536	0,001	0,000	0,001	0,006	0,026	0,001
131072	0,002	0,001	0,001	0,003	0,006	0,001
262144	0,003	0,000	0,003	0,004	0,008	0,005
524288	0,006	0,001	0,005	0,006	0,011	0,012
1048576	0,011	0,007	0,004	0,026	0,148	0,014
2097152	0,022	0,011	0,011	0,020	0,096	0,027
4194304	0,043	0,013	0,030	0,023	0,089	0,047
8388608	0,084	0,035	0,048	0,040	0,151	0,084
16777216	0,165	0,058	0,107	0,059	0,137	0,188
33554432	0,328	0,121	0,207	0,129	0,384	0,347
67108864	0,328	0,129	0,199	0,129	0,353	0,372