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): Carmen García Romero Grupo de prácticas y profesor de prácticas:2C-C1 Christian Morillas Fecha de entrega: 12/04/21 Fecha evaluación en clase: 13/04/21

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

bucle-forModificado.c

```c
#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 iteraciones \n");
     exit(-1);
   }

   n = atoi(argv[1]);

   #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);

    return(0);
}
```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

sectionsModificado.c

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

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

void funcB() {
   printf("En funcB: esta sección 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.

   RESPUESTA: Captura que muestre el código fuente singleModificado.c

singleModificado.c

```c
#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 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("Dentro de la región parallel:\n");

        for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
        printf("\n");
        printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
      }
  }
}
```

CAPTURAS DE PANTALLA:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/bp1/
ejer2  2021-03-22 lunes
$./singleModificado
Introduce valor de inicializacion a: 7
Single ejecutada por el thread 5
Dentro de la región parallel:
b[0] = 7        b[1] = 7        b[2] = 7        b[3] = 7        b[4] = 7
b[5] = 7        b[6] = 7 b[7] = 7        b[8] = 7
Single ejecutada por el thread 2
```

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

singleModificado2.c

```c
#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 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("Master dentro de la región parallel:\n");

        for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
        printf("\n");
        printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
      }
  }
}
```

CAPTURAS DE PANTALLA:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/bp
1/ejer3  2021-03-22 lunes
$./singleModificado2
Introduce valor de inicializacion a: 7
Single ejecutada por el thread 5
Master dentro de la región parallel:
b[0] = 7        b[1] = 7        b[2] = 7        b[3] = 7        b[4] = 7
b[5] = 7 b[6] = 7        b[7] = 7        b[8] = 7
Single ejecutada por el thread 0
```

RESPUESTA A LA PREGUNTA: Que la hebra que siempre ejecuta el código es la 0, la hebra 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 barrier hace que todas las hebras se esperen entre sí en ese punto, de forma que si no lo añadimos, las primeras hebras ejecutarían la suma al llegar, y este estaría incompleta.

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:

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer5  2021-03-23 martes
$sbatch -p ac --wrap "time ./SumaVectores 10000000"
Submitted batch job 76175
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer5  2021-03-23 martes
$cat slurm-76175.out
Tiempo:0.037284802      / TamaÃ±o Vectores:10000000    / V1[0]+V2[0]=V3[0](1.061124+0.310553=1.371677) /
 / V1[9999999]+V2[9999999]=V3[9999999](0.665079+0.855890=1.520970) /

real    0m0.596s
user    0m0.549s
sys     0m0.037s
```

RESPUESTA: user + sys = CPUtime, que es el tiempo que tarda en ejecutarse el código de las llamas al sistema y el código del programa.

El tiempo real (elapsed) es mayor ya que este también incluye esperas de E/S, que no forman parte del CPUtime.

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):

   - 10 componentes

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer5  2021-04-06 martes
$sbatch -p ac --wrap "time ./SumaVectores 10"
Submitted batch job 78786
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer5  2021-04-06 martes
$cat slurm-78786.out
Tiempo:0.000000182       / TamaÃ±o Vectores:10  / V1[0]+V2[0]=V3[0](0.459425+3.8
33110=4.292535) / / V1[9]+V2[9]=V3[9](3.475389+1.522045=4.997434) /

real    0m0.025s
user    0m0.000s
sys     0m0.001s
```

   - 10000000 componentes

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer5  2021-04-06 martes
$sbatch -p ac --wrap "time ./SumaVectores 10000000"
Submitted batch job 78790
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer5  2021-04-06 martes
$cat slurm-78790.out
Tiempo:0.037094536       / TamaÃ±o Vectores:10000000    / V1[0]+V2[0]=V3[0](3.56
6779+1.020221=4.587001) / / V1[9999999]+V2[9999999]=V3[9999999](11.978530+0.2989
18=12.277447) /

real    0m0.596s
user    0m0.531s
sys     0m0.054s
```

   RESPUESTA: cálculo de los MIPS y los MFLOPS

   Tenemos 6 instrucciones dentro del bucle, y 3 fuera.

   - Tamaño 10

     NI = 6*10 + 3 = 63

     Tiempo(s) = 0.000000182

     FPO = 3*10 = 30

     MIPS = $63/(0.000000182*10^6) = 346{,}1538461$

MFLOPS = 30/(0.000000182*10^6) =164,835164

- Tamaño 10000000

  NI = 6*10000000 + 3 = 60000003

  Tiempo(s) = 0.037094536

  FPO = 3*10000000 = 30000000

  MIPS = 60000003/(0.037094536*10^6) = 1.617,453045

  MFLOPS = 30000000/(0.037094536*10^6) = 80,874444

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

```
100         call    clock_gettime@PLT
101         xorl    %eax, %eax
102         .p2align 4,,10
103         .p2align 3
104 .L10:
105         movsd   0(%rbp,%rax,8), %xmm0
106         addsd   (%r14,%rax,8), %xmm0
107         movsd   %xmm0, 0(%r13,%rax,8)
108         addq    $1, %rax
109         cmpl    %eax, %r12d
110         ja      .L10
111         leaq    32(%rsp), %rsi
112         xorl    %edi, %edi
113         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

SumaVectores7.c

```c
#include <time.h> // biblioteca donde se encuentra la funciÃ³n clock_gettime()
#include <omp.h>

int main(int argc, char** argv){

  int i;
  double ncgt, cgt1,cgt2; ; //para tiempo de ejecuciÃ³n
  unsigned int N = atoi(argv[1]); // MÃ¡ximo N =2^32-1=4294967295 (sizeof(unsigned

  printf("TamaÃ±o Vectores:%u (%u B)\n",N, sizeof(unsigned int));

  //Leer argumento de entrada (nÂº de componentes del vector)
  if (argc<2){
    printf("Faltan nÂº componentes del vector\n");
    exit(-1);
  }

  double *v1, *v2, *v3;
  v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaÃ±o en bytes
  v2 = (double*) malloc(N*sizeof(double));
  v3 = (double*) malloc(N*sizeof(double));
  if ((v1 == NULL) || (v2 == NULL) || (v2 == NULL)) {
    printf("No hay suficiente espacio para los vectores \n");
    exit(-2);
  }

  //Inicializacion de los 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;
    }

   cgt1= omp_get_wtime();

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

  cgt2= omp_get_wtime();

  ncgt= cgt2-cgt1;

  //Imprimir resultado de la suma y el tiempo de ejecuciÃ³n
  if (N<12) {
  printf("Tiempo:%11.9f\t / TamaÃ±o Vectores:%u\n",ncgt,N);
  for(i=0; i<N; i++)
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

- **N = 8**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer7  2021-04-12 lunes
$sbatch -p ac --wrap "./SumaVectores7 8"
Submitted batch job 86673
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer7  2021-04-12 lunes
$cat slurm-86673.out
TamaÃ±o Vectores:8 (4 B)
Tiempo:0.000003606       / TamaÃ±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) /
```

- **N = 11**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer7  2021-04-12 lunes
$sbatch -p ac --wrap "./SumaVectores7 11"
Submitted batch job 86674
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer7  2021-04-12 lunes
$cat slurm-86674.out
TamaÃ±o Vectores:11 (4 B)
Tiempo:0.000003669       / TamaÃ±o 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) /
```

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

SumaVectores8.c

```c
int main(int argc, char** argv){

  int i;
  double ncgt, cgt1,cgt2; ; //para tiempo de ejecuciÃ³n
  unsigned int N = atoi(argv[1]); // MÃ¡ximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  printf("TamaÃ±o Vectores:%u (%u B)\n",N, sizeof(unsigned int));

  //Leer argumento de entrada (nÂº de componentes del vector)
  if (argc<2){
    printf("Faltan nÂº componentes del vector\n");
    exit(-1);
  }

  double *v1, *v2, *v3;
  v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaÃ±o en bytes
  v2 = (double*) malloc(N*sizeof(double));
  v3 = (double*) malloc(N*sizeof(double));
  if ((v1 == NULL) || (v2 == NULL) || (v2 == NULL)) {
    printf("No hay suficiente espacio para los vectores \n");
    exit(-2);
  }

  //Inicializacion de los vectores
  #pragma omp parallel sections private(i)
  {
    #pragma omp section
    for (i = 0; i < 1/4*N; i++){
        v1[i] = N * 0.1 + i * 0.1;
        v2[i] = N * 0.1 - i * 0.1;
    }

    #pragma omp section
    for (i = 1/4*N; i < 2/4*N; i++){
        v1[i] = N * 0.1 + i * 0.1;
        v2[i] = N * 0.1 - i * 0.1;
    }

    #pragma omp section
    for (i = 2/4*N; i < 3/4*N; i++){
        v1[i] = N * 0.1 + i * 0.1;
        v2[i] = N * 0.1 - i * 0.1;
    }

    #pragma omp section
    for (i = 3/4*N; i < N; i++){
        v1[i] = N * 0.1 + i * 0.1;
        v2[i] = N * 0.1 - i * 0.1;
    }
  }
```

SumaVectores8.c

```
54
55    cgt1= omp_get_wtime();
56
57    //Suma de vectores
58    #pragma omp parallel sections private(i)
59    {
60      #pragma omp section
61      for (i = 0; i < 1/4*N; i++)
62         v3[i] = v1[i] + v2[i];
63
64
65      #pragma omp section
66      for (i = 1/4*N; i < 2/4*N; i++)
67         v3[i] = v1[i] + v2[i];
68
69
70      #pragma omp section
71      for (i = 2/4*N; i < 3/4*N; i++)
72         v3[i] = v1[i] + v2[i];
73
74
75      #pragma omp section
76      for (i = 3/4*N; i < N; i++)
77        v3[i] = v1[i] + v2[i];
78
79
80    }
81
82    cgt2= omp_get_wtime();
83
84    ncgt= cgt2-cgt1;
85
86    //Imprimir resultado de la suma y el tiempo de ejecuciÃ³n
87    if (N<12) {
88    printf("Tiempo:%11.9f\t / TamaÃ±o Vectores:%u\n",ncgt,N);
89    for(i=0; i<N; i++)
90      printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
91             i,i,i,v1[i],v2[i],v3[i]);
92    }
93    else
94      printf("Tiempo:%11.9f\t / TamaÃ±o Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+%8.
95             ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
96
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

- **N = 8**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer8  2021-04-12 lunes
$sbatch -p ac --wrap "./SumaVectores8 8"
Submitted batch job 86686
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer8  2021-04-12 lunes
$cat slurm-86686.out
TamaÃ±o Vectores:8 (4 B)
Tiempo:0.000003569       / TamaÃ±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) /
```

- **N = 11**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer8  2021-04-12 lunes
$sbatch -p ac --wrap "./SumaVectores8 11"
Submitted batch job 86701
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer8  2021-04-12 lunes
$cat slurm-86701.out
TamaÃ±o Vectores:11 (4 B)
Tiempo:0.000003587       / TamaÃ±o 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) /
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer8  2021-04-12 lunes
```

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, la directiva for crea tantas hebras como indique la variable de entorno OMP_NUM_THREADS, de no estar definida se usarán todos los cores o threads disponibles de la máquina.

   Por otro lado, en el ejercicio 8, al dividir el bucle en 4 secciones fijas se usarán 4 hebras y, si hay más, las restantes no harán nada.

10. 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. 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

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer10  2021-04-12 lunes
$cat SumaVectores.sh
#!/bin/bash
#Órdenes para el Gestor de carga de trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaVectores
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#2. Asignar el trabajo a un account
#SBATCH --account=ac

#Obtener información de las variables del entorno del Gestor de carga de trabajo
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"
#Instrucciones del script para ejecutar código:
for ((N=16384; N<67108865; N=N*2))
do
   srun ./SumaVectores8 $N
done
```

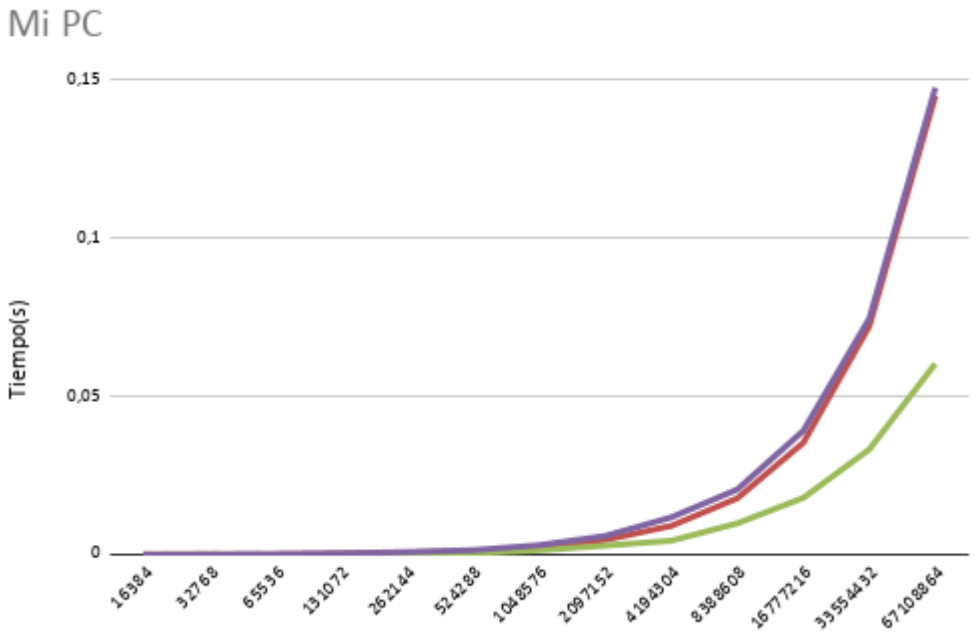
(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

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 y cores lógicos utilizados.

- **Mi PC**

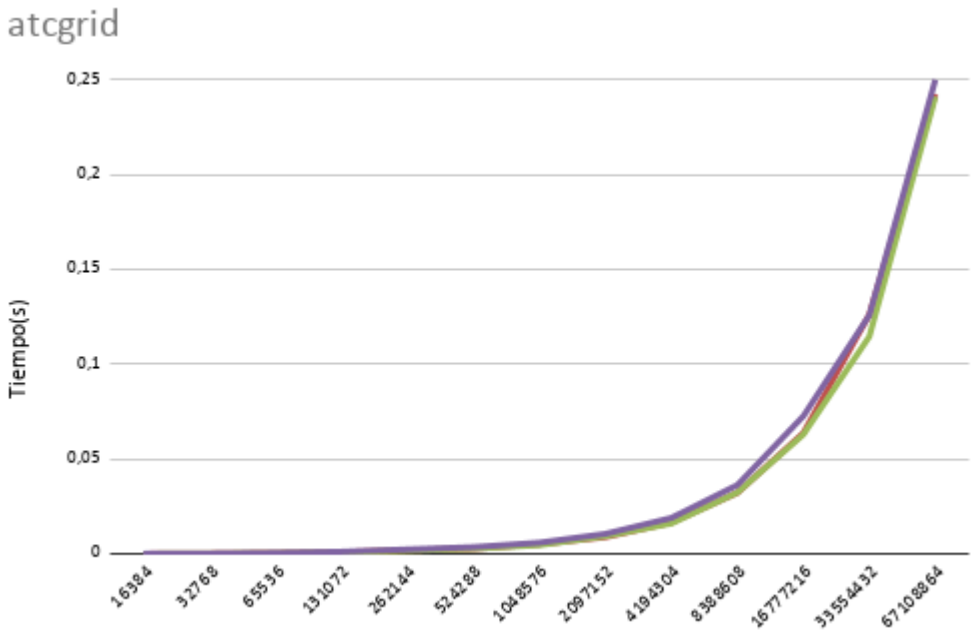
Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) 12 threads = cores lógicos = cores físicos	T. paralelo (versión sections) 12 threads = cores lógicos = cores físicos
16384	0.000157276	0.000056546	0.000050678
32768	0.000064043	0.000168143	0.000102179
65536	0.000126689	0.000033809	0.000213997
131072	0.000370467	0.000063319	0.000437316
262144	0.000618887	0.000117531	0.000861307
524288	0.001073855	0.000434657	0.001458181
1048576	0.002305505	0.001384357	0.003013509
2097152	0.004688445	0.002883310	0.005945611
4194304	0.009121764	0.004446369	0.011829396
8388608	0.017839078	0.009921113	0.020633976
16777216	0.035377732	0.017974196	0.039177762
33554432	0.072221591	0.033298421	0.074656428
67108864	0.144895050	0.060347964	0.147502153



- T. secuencial
- T. paralelo (versión for)
- T paralelo (versión section)

- **ATCGRID**

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) 24 threads = cores lógicos = cores físicos	T. paralelo (versión sections) 24 threads = cores lógicos = cores físicos
16384	0.000097026	0.000089116	0.000109609
32768	0.000234664	0.000188846	0.000210572
65536	0.000461662	0.000337701	0.000542108
131072	0.000949782	0.000716172	0.001154810
262144	0.001924438	0.001724593	0.002385497
524288	0.002654931	0.002769977	0.003579039
1048576	0.005110864	0.004615195	0.005922027
2097152	0.008826609	0.009311676	0.010557633
4194304	0.016537835	0.016022671	0.018954452
8388608	0.032193488	0.032342955	0.036277249
16777216	0.063734282	0.062946208	0.072619148
33554432	0.126149657	0.114578519	0.125839602
67108864	0.242369105	0.240986191	0.249909610



- T. secuencial
- T. paralelo (versión for)
- T paralelo (versión section)

11. 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 (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: Captura del script implementado sp-OpenMP-script11.sh

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer11  2021-04-12 lunes
$cat SumaVectores.sh
#!/bin/bash
#Órdenes para el Gestor de carga de trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaVectores
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#2. Asignar el trabajo a un account
#SBATCH --account=ac

#Obtener información de las variables del entorno del Gestor de carga de trabajo
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"
#Instrucciones del script para ejecutar código:
for ((N=8388608; N<67108865; N=N*2))
do
   time srun ./SumaVectores7 $N
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

- **Secuencial**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer11  2021-04-12 lunes
$cat slurm-87999.out
Id. usuario del trabajo: c1estudiante11
Id. del trabajo: 87999
Nombre del trabajo especificado por usuario: SumaVectores
Directorio de trabajo (en el que se ejecuta el script): /home/c1estudiante11/bp1/ejer11
Cola: ac
Nodo que ejecuta este trabajo:atcgrid.ugr.es
No de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
Tiempo:0.032408285       / TamaÃ±o Vectores:8388608     / V1[0]+V2[0]=V3[0](0.678973+1.135783=1.814756) / / V1[8388607]+V2[8388607]=V3[8388607](0.421070+0.455000=0.876070) /

real    0m0.583s
user    0m0.016s
sys     0m0.004s
Tiempo:0.062766447       / TamaÃ±o Vectores:16777216    / V1[0]+V2[0]=V3[0](1.738428+1.060081=2.798509) / / V1[16777215]+V2[16777215]=V3[16777215](0.098291+0.203956=0.302247)
/

real    0m1.037s
user    0m0.006s
sys     0m0.009s
Tiempo:0.124544533       / TamaÃ±o Vectores:33554432    / V1[0]+V2[0]=V3[0](0.140946+1.050975=1.191920) / / V1[33554431]+V2[33554431]=V3[33554431](0.746379+2.118001=2.864380)
/

real    0m1.951s
user    0m0.008s
sys     0m0.008s
Tiempo:0.238313913       / TamaÃ±o Vectores:67108864    / V1[0]+V2[0]=V3[0](0.411941+2.785539=3.197480) / / V1[67108863]+V2[67108863]=V3[67108863](0.176085+1.263918=1.440003)
/

real    0m3.624s
user    0m0.006s
sys     0m0.009s
```

- **For**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp1/ejer11  2021-04-12 lunes
$cat slurm-88019.out
Id. usuario del trabajo: c1estudiante11
Id. del trabajo: 88019
Nombre del trabajo especificado por usuario: SumaVectores
Directorio de trabajo (en el que se ejecuta el script): /home/c1estudiante11/bp1/ejer11
Cola: ac
Nodo que ejecuta este trabajo:atcgrid.ugr.es
No de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
TamaÃ±o Vectores:8388608 (4 B)
Tiempo:0.031058073       / TamaÃ±o 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) /

real    0m0.181s
user    0m0.010s
sys     0m0.011s
TamaÃ±o Vectores:16777216 (4 B)
Tiempo:0.056237552       / TamaÃ±o Vectores:16777216    / V1[0]+V2[0]=V3[0](1677721.600000+1677721.600000=3355443.200000) / / V1[16777215]+V2[16777215]=V3[16777215](3355443.10
0000+0.100000=3355443.200000) /

real    0m0.235s
user    0m0.007s
sys     0m0.009s
TamaÃ±o Vectores:33554432 (4 B)
Tiempo:0.116499405       / TamaÃ±o Vectores:33554432    / V1[0]+V2[0]=V3[0](3355443.200000+3355443.200000=6710886.400000) / / V1[33554431]+V2[33554431]=V3[33554431](6710886.30
0000+0.100000=6710886.400000) /

real    0m0.346s
user    0m0.006s
sys     0m0.009s
TamaÃ±o Vectores:67108864 (4 B)
Tiempo:0.226960879       / TamaÃ±o 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) /

real    0m0.618s
user    0m0.008s
sys     0m0.006s
```

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 = 1 core lógico = 1 core físico**			**Tiempo paralelo/versión** for **12 Threads = cores lógicos=cores físicos**		
	Elapsed	***CPU-user***	***CPU-sys***	***Elapsed***	***CPU-user***	***CPU- sys***
8388608	0m0.583s	0m0.016s	0m0.004s	0m0.181s	0m0.010s	0m0.011s
16777216	0m1.037s	0m0.006s	0m0.009s	0m0.235s	0m0.007s	0m0.009s
33554432	0m1.951s	0m0.008s	0m0.008s	0m0.346s	0m0.006s	0m0.009s
67108864	0m3.624s	0m0.006s	0m0.009s	0m0.618s	0m0.008s	0m0.006s