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): Brian Sena Simons
Grupo de prácticas y profesor de prácticas: 2ºB – B2
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

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

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

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

  int i, n = 9;

  if(argv < 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 itración %d del bucle\n", omp_get_thread_num(),i);

  return(0);
}
```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

```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());
}

int main(){
#pragma omp parallel sections
  {
#pragma omp section
    (void)funcA();
#pragma omp section
    (void)funcB();
  }
  return 0;
}
```

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>

// gcc -O2 -fopenmp -o single single.c

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[1]=a;


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

}
```

CAPTURAS DE PANTALLA:

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer2] 2021-03-13 sábado
$./singleModificado
introduce valor de inicialización a:1000
Single ejecutada por el thread 3
Despues de la región parallel:
b[0] = -1 (2)   b[1] = 1000 (2) b[2] = -1 (2)   b[3] = -1 (2)   b[4] = -1 (2)   b[5] = -1 (2)   b[6] = -
1 (2)   b[7] = -1 (2)   b[8] = -1 (2)
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer2] 2021-03-13 sábado
$./singleModificado 10000
introduce valor de inicialización a:10000
Single ejecutada por el thread 1
Despues de la región parallel:
b[0] = -1 (6)   b[1] = 10000 (6)        b[2] = -1 (6)   b[3] = -1 (6)   b[4] = -1 (6)   b[5] = -1 (6)
b[6] = -1 (6)   b[7] = -1 (6)   b[8] = -1 (6)
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer2] 2021-03-13 sábado
$./singleModificado 100000
introduce valor de inicialización a:100000
Single ejecutada por el thread 3
Despues de la región parallel:
b[0] = -1 (1)   b[1] = 100000 (1)       b[2] = -1 (1)   b[3] = -1 (1)   b[4] = -1 (1)   b[5] = -1 (1)
b[6] = -1 (1)   b[7] = -1 (1)   b[8] = -1 (1)
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer2] 2021-03-13 sábado
```

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>

// gcc -O2 -fopenmp -o single single.c

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[1]=a;


#pragma omp master
    {
      printf("Despues de la región parallel:\n");
      for(i=0;i<n;++i)
        printf("b[%d] = %d (%d)\t",i,b[i],omp_get_thread_num());
      printf("\n");
    }
  }
  return 0;

}
```

CAPTURAS DE PANTALLA:

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer3] 2021-03-13 sábado
$./singleModificado2
introduce valor de inicialización a:10000
Single ejecutada por el thread 7
Despues de la región parallel:
b[0] = -1 (0)   b[1] = 10000 (0)        b[2] = -1 (0)   b[3] = -1 (0)   b[4] = -1 (0)   b[5] = -1 (0)
b[6] = -1 (0)   b[7] = -1 (0)   b[8] = -1 (0)
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer3] 2021-03-13 sábado
$./singleModificado2
introduce valor de inicialización a:100000
Single ejecutada por el thread 0
Despues de la región parallel:
b[0] = -1 (0)   b[1] = 100000 (0)       b[2] = -1 (0)   b[3] = -1 (0)   b[4] = -1 (0)   b[5] = -1 (0)
b[6] = -1 (0)   b[7] = -1 (0)   b[8] = -1 (0)
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer3] 2021-03-13 sábado
$./singleModificado2
introduce valor de inicialización a:1000
Single ejecutada por el thread 7
Despues de la región parallel:
b[0] = -1 (0)   b[1] = 1000 (0) b[2] = -1 (0)   b[3] = -1 (0)   b[4] = -1 (0)   b[5] = -1 (0)   b[6] = -
1 (0)   b[7] = -1 (0)   b[8] = -1 (0)
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer3] 2021-03-13 sábado
```

RESPUESTA A LA PREGUNTA:

Hemos podido observar que en ambas ejecuciones, apenas una hebra es la que finaliza el código después de la región paralela. Sin embargo, utilizando la directiva single esa hebra es escogida aleatoriamente según

esté ocupada o ociosa. Mientras que en la directiva master ocurre que siempre es la hebra 0 (Maestra). Lo único es que no tiene barreras por lo que podría ocasionar valores erróneos debido a la falta de sincronización entre hebras.

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:

Al quitar la directiva *barrier*, la hebra maestra (0), y cualquiera de las demás, si ha finalizado sus cálculos seguirá la ejecución secuencial del código y terminará imprimiendo por pantalla antes de que las que aún estén pendientes por terminar hayan registrado la suma.

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:

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer5] 2021-03-23 martes
$ls
SumaVectoresG
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer5] 2021-03-23 martes
$time ./SumaVectoresG 10000000
Tiempo(seg.):0.032046775        / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1.053404+2.06573
119137) /           V1[9999999]+V2[9999999]=V3[9999999](2.956480+5.060154=8.016635) /

real    0m0.302s
user    0m0.240s
sys     0m0.062s
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer5] 2021-03-23 martes
$time ./SumaVectoresG 10000000
Tiempo(seg.):0.032077658        / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1.724202+0.72053
444739) /           V1[9999999]+V2[9999999]=V3[9999999](8.508197+2.357509=10.865706) /

real    0m0.302s
user    0m0.233s
sys     0m0.069s
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer5] 2021-03-23 martes
```

RESPUESTA:

La suma de los **tiempos de CPU del usuario y del sistema** son siempre igual o menor que al tiempo real en programas secuenciales, en mi caso son iguales que el tiempo real. Porque el tiempo real es una aproximación de lo que puede tardar el programa; El tiempo de usuario es lo que tarda que el usuario le da enter hasta que recupera el control de la terminal y el tiempo de sistema es lo que hace por debajo del programa usando recursos del sistema.

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):

```
2021-03-23-07:46 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer5$ gcc -O2 -fopenmp -S SumaVectoresC.c
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:89:54: warning: format ‘%lu’ expects argument of type ‘long unsigned int’, but argument
3 has type ‘unsigned int’ [-Wformat=]
   89 |   printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%lu\n",ncgt,N);
      |                                                     ~~^         ~
      |                                                       |         |
      |                                                       |         unsigned int
      |                                                       long unsigned int
      |                                                     %u
2021-03-23-07:57 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer5$ ls
SumaVectoresC.c  SumaVectoresC.s  SumaVectoresG
2021-03-23-07:57 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer5$ gcc -O2 -fopenmp -o SumaVectoresG Sum
aVectoresC.c
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:89:54: warning: format ‘%lu’ expects argument of type ‘long unsigned int’, but argument
3 has type ‘unsigned int’ [-Wformat=]
   89 |   printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%lu\n",ncgt,N);
      |                                                     ~~^         ~
      |                                                       |         |
      |                                                       |         unsigned int
      |                                                       long unsigned int
      |                                                     %u
2021-03-23-07:58 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer5$ ls
SumaVectoresC.c  SumaVectoresC.s  SumaVectoresG
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer5] 2021-03-23 martes
$time ./SumaVectoresG 10000000
Tiempo(seg.):0.035793519         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](11.984104+1.833060=1
3.817163) /           V1[9999999]+V2[9999999]=V3[9999999](0.373530+0.329370=0.702900) /

real    0m0.311s
user    0m0.251s
sys     0m0.060s
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer5] 2021-03-23 martes
$time ./SumaVectoresG 10
Tiempo(seg.):0.000002134         / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](0.497424+0.199950=0.697373)
/           V1[9]+V2[9]=V3[9](1.390424+0.313308=1.703732) /

real    0m0.001s
user    0m0.000s
sys     0m0.001s
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer5] 2021-03-23 martes
```

Si recordamos el código, tiene 6 instrucciones por bucle y 3 para inicializar las variables. Con los tiempos obtenidos del programa puedo calcular los MIPS que tiene. Para los MFLOPS se aplica la misma lógica pero teniendo en cuenta que ahora hablamos de operaciones en coma flotante.

Para 1000:

MIPS = NI/Tcpu = F / CPI = 3 + (6*10000000)/(0,035793519*10^6) =

MFLOPS = Oper_coma_flotante / Tcpu = 7*10000000/(0,035793519*10^6) =

Para 10:

MIPS = NI/Tcpu = F / CPI = 3 + (6*10)/(0,000002134*10^6) =

MFLOPS = Oper_coma_flotante / Tcpu = 7*10/(0,000002134*10^6) =

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

```
29   xorl  %eax, %eax
28   .p2align 4,,10
27   .p2align 3
26 .L8:
25   movsd (%r15,%rax,8), %xmm0
24   addsd 0(%r13,%rax,8), %xmm0
23   movsd %xmm0, 0(%rbp,%rax,8)
22   addq  $1, %rax
21   cmpl  %eax, %r12d
20   ja  .L8
19   leaq  32(%rsp), %rsi
18   xorl  %edi, %edi
17   call  clock_gettime@PLT
16   movq  40(%rsp), %rax
```

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

```
#pragma omp parallel for
    //Inicializar vectores
    for (i = 0; i < N; i++)
    {
      v1[i] = N * 0.1 + i * 0.1;
      v2[i] = N * 0.1 - i * 0.1;
    }
  double time_start = omp_get_wtime();
    //srand(time(0));
#pragma omp parallel for
  //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));*/

  ncgt = (double)((omp_get_wtime() - time_start)*1000.0);
  //Imprimir resultado de la suma y el tiempo de ejecución
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
2021-03-23-08:35 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer7$ nvim sp-OpenMP-for.c
2021-03-23-08:35 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer7$ gcc -O2 -fopenmp -o sp-OpenMP-for sp-
OpenMP-for.c
2021-03-23-08:35 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer7$ ./sp-OpenMP-for 8
Tiempo(seg.):1.449800000         / 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) /
2021-03-23-08:35 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer7$ ./sp-OpenMP-for 11
Tiempo(seg.):0.506100000         / Tamaño 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) /
2021-03-23-08:35 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer7$
```

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

```
#pragma omp parallel sections
{
    //Inicializar vectores
#pragma omp section
    for (i = 0; i < N/4; i++){
      v1[i] = N * 0.1 + i * 0.1; v2[i] = N * 0.1 - i * 0.1;
    }
#pragma omp section
    for (i = N/4; i < N/2; i++){
      v1[i] = N * 0.1 + i * 0.1; v2[i] = N * 0.1 - i * 0.1;
    }
#pragma omp section
    for (i = N/2; i < N*3/4; i++){
      v1[i] = N * 0.1 + i * 0.1; v2[i] = N * 0.1 - i * 0.1;
    }
#pragma omp section
    for (i = N*3/4; i < N; i++){
      v1[i] = N * 0.1 + i * 0.1; v2[i] = N * 0.1 - i * 0.1;
    }
}
  double time_start = omp_get_wtime();
    //srand(time(0));
#pragma omp parallel sections
{
#pragma omp section
    for (i = 0; i < N/4; i++){
    v3[i] = v1[i] + v2[i];
    }
#pragma omp section
    for (i = N/4; i < N/2; i++){
    v3[i] = v1[i] + v2[i];
    }
#pragma omp section
    for (i = N/2; i < N*3/4; i++){
    v3[i] = v1[i] + v2[i];
    }
#pragma omp section
    for (i = N*3/4; i < N; i++){
    v3[i] = v1[i] + v2[i];
    }
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
2021-03-23-08:38 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer8$ nvim sp-OpenMP-sections.c
2021-03-23-08:54 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer8$ gcc -O2 -fopenmp -o sp-OpenMP-section
 sp-OpenMP-sections.c
2021-03-23-08:54 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer8$ ./sp-OpenMP-section 8
Tiempo(seg.):0.970600000         / 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) /
2021-03-23-08:54 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer8$ ./sp-OpenMP-section 11
Tiempo(seg.):0.007500000         / Tamaño 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) /
2021-03-23-08:54 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP1/ejer8$
```

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** como usábamos **"omp parallel for",** automáticamente el código se ejecuta con el máximo número de hebras del sistema, siempre y cuando no hayamos establecido un máximo manualmente con **"omp_set_num_threads(x)".**

   En el caso del **ejercicio 8**, al utilizar sections y apenas haber desarrollado 4 de ellas, la computadora ejecutará apenas 4 hebras concurrentemente.

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

Pd: He cambiado los prints de los archivos para qué solo me impriman el tiempo de ejecución y así sea más fácil copiar y pegar en la tabla.

Ubuntu 20.04 LTS

```
#!/bin/bash

echo "Tiempo ejecución secuencial:"
for i in $(seq 14 26); do
    A=`echo "2^$i" | bc `
    ./SumaVectoresG $A
done

echo "Tiempo ejecución de OpemMP-FOR:"
for i in $(seq 14 26); do
    A=`echo "2^$i" | bc `
    ./sp-OpenMP-for $A
done

echo "Tiempo ejecución de OpenMP-sections:"
for i in $(seq 14 26); do
    A=`echo "2^$i" | bc `
    ./sp-OpenMP-sections $A
done
```

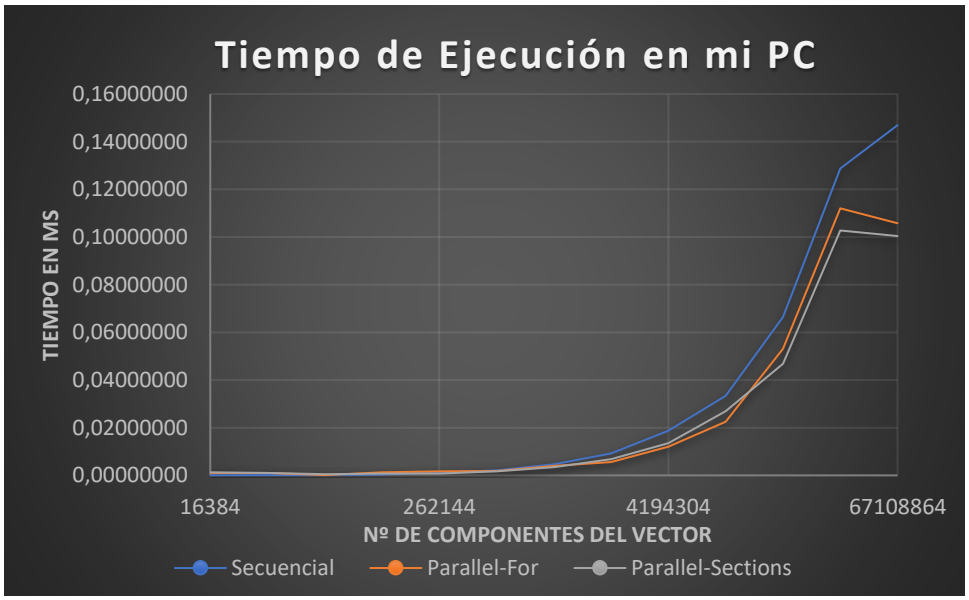
```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer10] 2021-03-30 martes
$sbatch -p ac allSizes.sh
Submitted batch job 77048
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer10] 2021-03-30 martes
$ls
allSizes.sh  slurm-77048.out  sp-OpenMP-for  sp-OpenMP-sections  SumaVectoresG
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer10] 2021-03-30 martes
$
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

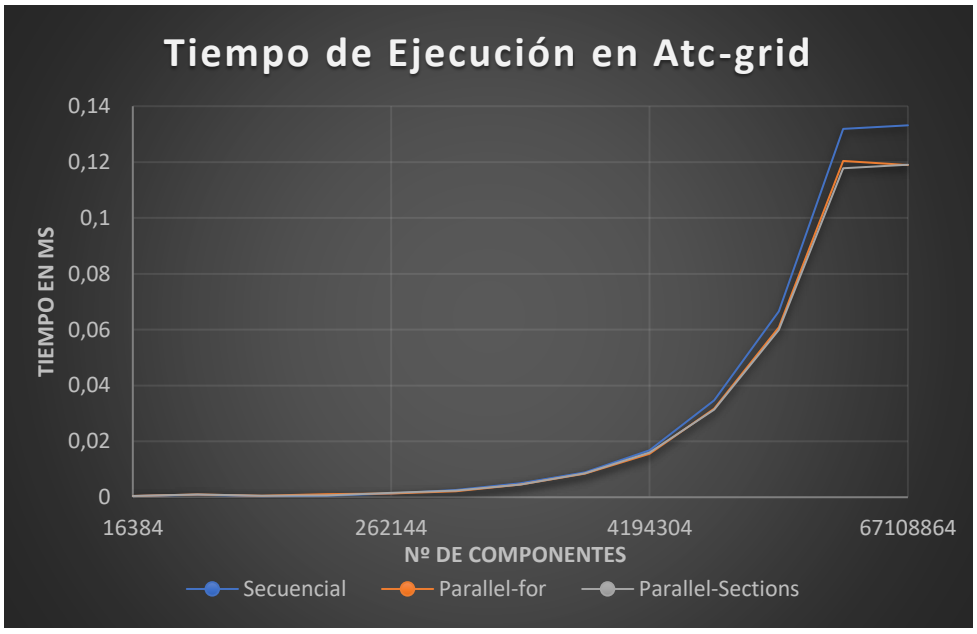
Tiempo ejecución en mi PC:

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) ¿?threads = cores lógicos = cores físicos	T. paralelo (versión sections) ¿?threads = cores lógicos = cores físicos
16384	0.000041200	0.001006600	0.001321500
32768	0.000059100	0.001043800	0.000992500
65536	0.000131500	0.000109100	0.000485200
131072	0.000318200	0.001294200	0.000740100
262144	0.001326300	0.001727800	0.000722500
524288	0.002134000	0.001788000	0.001733000
1048576	0.004699100	0.003950200	0.003379900
2097152	0.009201100	0.005618900	0.006826000
4194304	0.018782900	0.012008700	0.013549800
8388608	0.033418900	0.022461100	0.026907500
16777216	0.066456700	0.053086700	0.046812400
33554432	0.128733100	0.112055300	0.102757800
67108864	0.146928700	0.105811300	0.100362300



Tiempo de Ejecución en ATC-GRID:

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) ¿?threads = cores lógicos = cores físicos	T. paralelo (versión sections) ¿?threads = cores lógicos = cores físicos
16384	0.000430140	0.000458203	0.000451583
32768	0.000884755	0.000985712	0.000976808
65536	0.000385149	0.000497684	0.000568420
131072	0.000526832	0.001067813	0.000624850
262144	0.001340818	0.001326110	0.001573656
524288	0.002575860	0.002114229	0.002370473
1048576	0.005019039	0.004555743	0.004605968
2097152	0.008938022	0.008390326	0.008610044
4194304	0.016864381	0.015554648	0.016047191
8388608	0.034699956	0.031688750	0.031323101
16777216	0.066598025	0.060837433	0.059916690
33554432	0.131836073	0.120414563	0.117755380
67108864	0.133162577	0.118927237	0.119066168



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

```
#!/bin/bash

echo "Tiempo ejecución secuencial:"
for i in $(seq 23 26); do
    A=`echo "2^$i" | bc `
    time ./SumaVectoresG $A
done

echo "Tiempo ejecución de OpemMP-FOR:"
for i in $(seq 23 26); do
    A=`echo "2^$i" | bc `
    time ./sp-OpenMP-for $A
done

echo "Tiempo ejecución de OpenMP-sections:"
for i in $(seq 23 26); do
    A=`echo "2^$i" | bc `
    time ./sp-OpenMP-sections $A
done
```

```
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer11] 2021-03-30 martes
$sbatch -p ac allSizes2.sh
Submitted batch job 77051
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer11] 2021-03-30 martes
$ls
allSizes2.sh  slurm-77051.out  sp-OpenMP-for  sp-OpenMP-sections  SumaVectoresG
[BrianSenaSimons b2estudiante23@atcgrid:~/BP1/ejer11] 2021-03-30 martes
$
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

Tiempo Ejecución en mi PC:

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread = 1 core lógico = 1 core físico			Tiempo paralelo/versión for ¿? Threads = cores lógicos=cores físicos		
	Elapsed	*CPU-user*	*CPU- sys*	*Elapsed*	*CPU-user*	*CPU- sys*
8388608	real 0m0,289s	user 0m0,266s	sys 0m0,016s	real 0m0,067s	user 0m0,141s	sys 0m0,094s
16777216	real 0m0,558s	user 0m0,516s	sys 0m0,031s	real 0m0,122s	user 0m0,438s	sys 0m0,266s
33554432	real 0m1,124s	user 0m1,016s	sys 0m0,109s	real 0m0,268s	user 0m0,750s	sys 0m0,703s
67108864	real 0m1,111s	user 0m1,031s	sys 0m0,078s	real 0m0,268s	user 0m0,969s	sys 0m0,516s

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 **¿? Threads = cores lógicos=cores físicos**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
8388608	real 0m0.544s			real 0m0.083s		
	user 0m0.459s			user 0m0.081s		
	sys 0m0.044s			sys 0m0.054s		
16777216	real 0m0.996s			real 0m0.149s		
	user 0m0.900s			user 0m0.163s		
	sys 0m0.070s			sys 0m0.111s		
33554432	real 0m1.927s			real 0m0.281s		
	user 0m1.738s			user 0m0.326s		
	sys 0m0.139s			sys 0m0.213s		
67108864	real 0m1.899s			real 0m0.276s		
	user 0m1.740s			user 0m0.322s		
	sys 0m0.148s			sys 0m0.205s		

En el caso de la ejecución secuencial el tiempo total, elapsed, es >= que la suma de "user" + "sys" debido a que como no tiene una ejecución en paralela el tiempo total coincide con lo que ha tardado el programa. Pero en el caso del "Omp-for" el tiempo elapsed es siempre < que la suma "user" + "sys" debido a que tiene una ejecución en paralelo. Time lo que hace es sumar el tiempo que ha tardado cada hebra dando un tiempo que no coincide con lo que ha tardado realmente.