2º curso / 2º cuatr.

Grados Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 1. Programación paralela I: Directivas OpenMP

Estudiante (nombre y apellidos): QUINTÍN MESA ROMERO
Grupo de prácticas y profesor de prácticas: A1 DGIIM

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
 * ~~~~~~~~~~~~~~~~~~~~
 *
*/
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("Hebra %d ejecuta la iteración %d del bucle\n",
             omp_get_thread_num(),i);


   return(0);
}
```

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

```
 * ~~~~~~~~~~~~~~~~~~~~~
 * sections
 * ~~~~~~~~~~~~~~~~~~~~~
 *
*/
void funcA()
{
   printf("En funcA: esta sectión la ejecuta la hebra %d\n",
        omp_get_thread_num());
}
void funcB()
{
   printf("En funcB: esta sectión la ejecuta la hebra %d\n",
        omp_get_thread_num());
}

void funcC()
{
   printf("En funcC: esta sectión la ejecuta la hebra %d\n",
        omp_get_thread_num());
}

int main()
{
#pragma omp parallel sections
{
      #pragma omp section
        (void) funcA();

      #pragma omp section
        (void) funcB();

      #pragma omp section
        (void) funcC();

}

   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
#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif



int main()
{
   int n = 9;
   int 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 la hebra %d\n",
             omp_get_thread_num());
   }

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

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

```
quintin@quintin-Lenovo-Yoga-S740-14IIL: ~/Documentos/Do...
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica1/bp1$ gcc -O2 -fopenmp -o singleModificado singleModificado.c
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica1/bp1$ ./singleModificado
Introduce valor de inicialización a: 9
Single ejecutada por la hebra 3
Hebra que ejecuta esto: 1
Depués de la región parallel:
 b[0] = 9        b[1] = 9        b[2] = 9        b[3] = 9        b[4] = 9
b[5] = 9        b[6] = 9        b[7] = 9        b[8] = 9
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica1/bp1$
```

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
int main()
{
   int n = 9;
   int 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 la hebra %d\n",
             omp_get_thread_num());
   }

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

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

   return(0);
}
```

```
quintin@quintin-Lenovo-Yoga-S740-14IIL: ~/Documentos/Do...
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica1/bp1$ gcc -O2 -fopenmp -o singleModificado singleModificado.c
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica1/bp1$ ./singleModificado
Introduce valor de inicialización a: 9
Single ejecutada por la hebra 3
Hebra que ejecuta esto: 0
Depués de la región parallel:
 b[0] = 9        b[1] = 9        b[2] = 9        b[3] = 9        b[4] = 9
b[5] = 9        b[6] = 9        b[7] = 9        b[8] = 9
```

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: La directiva **master** no dispone de barrera implícita, por lo que es necesario añadir una directiva **barrier** antes para su correcto funcionamiento. El código de **master.c** realiza la suma de las componentes de un vector en paralelo, para ello, se calcula la suma con un bucle for al que se le aplica la directiva **omp for** , por tanto, cada hebra de las disponibles en el sistema se encarga de realizar una parte de las iteraciones del bucle (las que le asignen). Es posible que alguna de las hebras termina antes que otra, por lo que escribe el valor de **sumalocal** en la variable **suma** , mientras que las demás pueden seguir calculando.

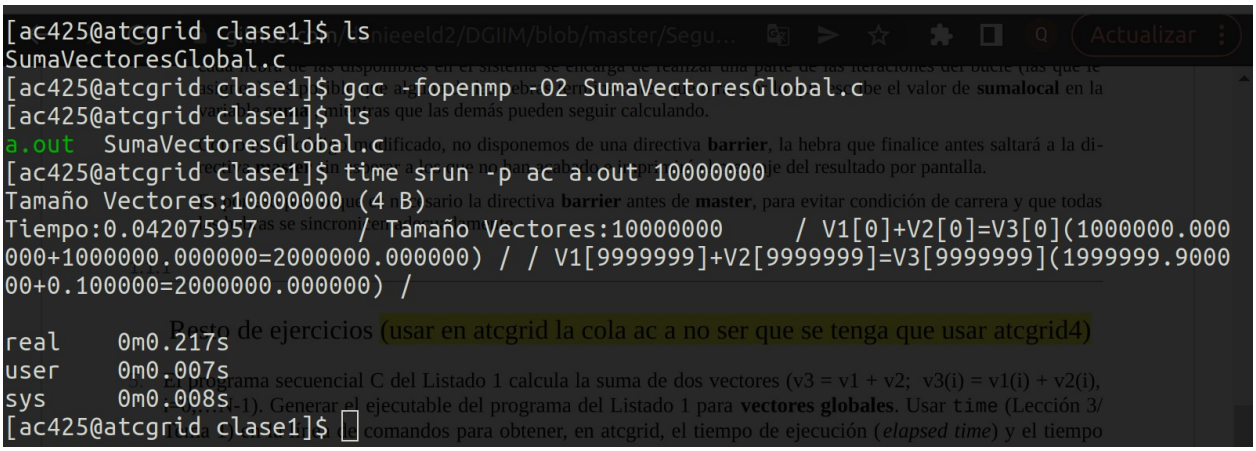
   Como en el código modificado, no disponemos de una directiva **barrier**, la hebra que finalice antes saltará a la directiva **master** sin esperar a los que no han acabado e imprimirá el mensaje del resultado por pantalla.

   Es por eso por lo que es necesario la directiva **barrier** antes de **master**, para evitar condición de carrera y que todas las hebras se sincronicen adecuadamente.

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:



   RESPUESTA: Vemos que el real time es mayor que la suma del user y sys time.

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 (ver cuaderno de BP0) 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):

RESPUESTA: cálculo de los MIPS y los MFLOPS

```
[ac425@atcgrid clase1]$ gcc -fopenmp SumaVectoresGlobal.c
[ac425@atcgrid clase1]$ ls
a.out  SumaVectoresGlobal.c  SumaVectoresGlobal.s
[ac425@atcgrid clase1]$ srun -p ac a.out 10
Tamaño Vectores:10 (4 B)
Tiempo:0.000387946      / 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) /
[ac425@atcgrid clase1]$ srun -p ac a.out 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.060084323      / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000
000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.9000
00+0.100000=2000000.000000) /
[ac425@atcgrid clase1]$
```

Cálculo de MIPS Y MFLOPS

$$MIPS = \frac{NI}{T_{CPU} * 10^6}$$

$$MFLOPS = \frac{nopFlotantes}{T_{CPU} * 10^6}$$

PARA 10:

- MIPS = (10*6+3)/(0.000387946*10⁶) = 0.1623937352

-MFLOPS = (10*1)/0.000387946*10⁶) = 0.02577678337

PARA 10000000:

- MIPS = (10000000*6)/(0.060084323*10⁶) = 998.596589

-MFLOPS = (10000000*1)/0.060084323*10⁶) = 166.4327648

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

```
[ac425@atcgrid clase1]$ gcc -S -fopenmp SumaVectoresGlobal.c
```

```
.L6:
        movsd   v1(,%rax,8), %xmm0
        addsd   v2(,%rax,8), %xmm0
        movsd   %xmm0, v3(,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %ebp
        ja      .L6
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        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); (4) se debe imprimir el tamaño de los vectores y el número de hilos; (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`

```
65  //Inicializar vectores
66
67  #pragma omp parallel for
68  for(i=0; i<N; i++){
69    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //Se puede usar drand48() para
generar los valores de forma aleatoria (drand48_r() para una versión paralela)
70  }
71
72  cgt1 = omp_get_wtime();
73
74  //Calcular suma de vectores
75  #pragma omp parallel for
76  for(i=0; i<N; i++)
77    v3[i] = v1[i] + v2[i];
78
79  cgt2 = omp_get_wtime();
80
81  ncgt=cgt2-cgt1;
82
83  //Imprimir resultado de la suma y el tiempo de ejecución
84  if (N<10) {
85  printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
86  for(i=0; i<N; i++)
87    printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
88           i,i,i,v1[i],v2[i],v3[i]);
89
90  printf("Número de hilos: %d\n", omp_get_num_threads());
91  }
92  else{
93    printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+
%8.6f=%8.6f) / / V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
94           ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
95
96    printf("Número de hilos: %d\n", omp_get_num_threads());
97    }
98
99  return 0;
100 }
```

```
[ac425@atcgrid clase1]$ gcc -fopenmp Ejercicio7.c -o Ejercicio7
[ac425@atcgrid clase1]$ srun -p ac Ejercicio7 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000442662      / 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) /
Número de hilos: 1
[ac425@atcgrid clase1]$ srun -p ac Ejercicio7 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000521471      / 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) /
Número de hilos: 1
[ac425@atcgrid clase1]$
```

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); (4) se debe imprimir el tamaño de los vectores y el número de hilos; (5) 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 private(i)
{
       #pragma omp section
        for(i=0; i<N; i+=4){
          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //Se puede usar drand48()
para generar los valores de forma aleatoria (drand48_r() para una versión paralela)
        }

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

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

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

      cgt1 = omp_get_wtime();
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
  //Calcular suma de vectores

  #pragma omp parallel sections private(i)
  {
        #pragma omp section
         for(i=0; i<N; i+=4){
           v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //Se puede usar drand48()
para generar los valores de forma aleatoria (drand48_r() para una versión paralela)
         }

        #pragma omp section
         for(i=1; i<N; i+=4){
           v3[i] = v1[i] + v2[i];
         }

         #pragma omp section
         for(i=2; i<N; i+=4){
           v3[i] = v1[i] + v2[i];
         }

         #pragma omp section
         for(i=3; i<N; i+=4){
           v3[i] = v1[i] + v2[i];
         }
  }


       cgt2 = omp_get_wtime();

  ncgt=cgt2-cgt1;
```

```
[ac425@atcgrid clase1]$ gcc -fopenmp Ejercicio8.c -o Ejercicio8
[ac425@atcgrid clase1]$ srun -p ac Ejercicio8 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000459491      / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=0.000000) /
/ 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=0.000000) /
/ 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) /
Número de hilos: 1
[ac425@atcgrid clase1]$ srun -p ac Ejercicio8 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000445759      / Tamaño Vectores:11    / V1[0]+V2[0]=V3[0](1.100000+1.100000=0
.000000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
Número de hilos: 1
[ac425@atcgrid clase1]$
```

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 máximo se podría ejecutar en N hebras. Este número estará limitado por el número de cores lógicos de los que nuestro computador tenga.

   Por su parte, el ejercicio 8 solo se podrá ejecutar como máximo en un número de hebras igual al número de secciones en el que hayamos dividido el bucle; esto es en mi caso, 4.

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`

    (RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

```
#!/bin/bash
sizes=(16384 32768 65536 131072 262144 524288 1048576 2097152 4194304 8388608
16777216 33554432 67108864)

for i in "${sizes[@]}"; do
        ./Ejercicio8 $i
done
```

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.

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		0.000020778	0.000098606
32768		0.000110188	0.000132078
65536		0.000291160	0.000271794
131072		0.000431649	0.000515855
262144		0.000469066	0.001022459
524288		0.000584733	0.002238415
1048576		0.001205289	0.004819661
2097152		0.002380499	0.009069730
4194304		0.004520271	0.016975840
8388608		0.008867558	0.032223247
16777216		0.017468642	0.070531539
33554432		0.035115987	0.138721694
67108864		0.034029526	0.122079070

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
sizes=(8388608 16777216 33554432 67108864
)

for i in "${sizes[@]}"; do
        time ./Ejercicio7 $i
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

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	0m0.096s	0m0.034s	0m0.062s	0m0.040s	0m0.091s	0m0.111s
16777216	0m0.184s	0m0.092s	0m0.091s	0m0.070s	0m0.153s	0m0.201s
33554432	0m0.356s	0m0.133s	0m0.223s	0m0.135s	0m0.306s	0m0.390s
67108864	0m0.359s	0m0.130s	0m0.229s	0m0.136s	0m0.313s	0m0.394s