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): Javier Gómez López
Grupo de prácticas y profesor de prácticas: A1 Carlos Megías Nuñez

---

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
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 schedule(static)
   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`

```c
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 directi-

va `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
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("En la región parallel:\n");
   for (i=0; i<n; i++)
        printf(" b[%d] = %d\t",i,b[i]);
   printf("\nSingle ejecutada por la hebra %d\n", omp_get_thread_num());
   }

}
   return(0);
}
```

CAPTURAS DE PANTALLA:

```
javi5454@javi5454-Prestige-15-A10SC:~/Desktop/Github/Linux_PC/II_Curso/II_CUATRI/AC/Practicas/Practica_1/bp1$ ./singleModificado
Introduce valor de inicialización a: 10
Single ejecutada por la hebra 0
En 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
Single ejecutada por la hebra 1
```

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("En de la región parallel:\n");
   for (i=0; i<n; i++)
       printf(" b[%d] = %d\t",i,b[i]);
   printf("\nMaster ejecutada por la hebra %d\n", omp_get_thread_num());
   }
}
   return(0);
}
```

CAPTURAS DE PANTALLA:

```
javi5454@javi5454-Prestige-15-A10SC:~/Desktop/Github/Linux_PC/II_Curso/II_CUATRI/AC/Practicas/Practica_1/bp1$ ./singleModificado2
Introduce valor de inicialización a: 10
Single ejecutada por la hebra 5
En 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
Master ejecutada por la hebra 0
```

RESPUESTA A LA PREGUNTA:

Simplemente que la parte introducida dentro de la región parallel siempre la ejecuta el thread 0 con la directiva master, mientras que con la directiva single puede ejecutarla de manera secuencial igual pero cualquier hebra.

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 sin la directiva barrier, si la hebra master (la hebra 0) termina antes que el resto, entrará en la zona master de la región parallel antes de que el resto hayan terminado su suma local, produciendo un resultado incorrecto.

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:

```
[ac410@atcgrid Executables]$ time srun -pac -n1 -Aac ./SumaVectores 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.060033287       / Tamaño 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    0m0.304s
user    0m0.007s
sys     0m0.008s
[ac410@atcgrid Executables]$
```

RESPUESTA: Vemos que la suma de los tiempos de usuario y sistema es 0.008s+0.007 = 0.015s < 0.304s. Esto se debe a esperas por operaciones, por ejemplo, de E/S.

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

   Tenemos un total de 6 operaciones, 1 de coma flotante, por tanto:

   - MIPS = NI / (T_ejecucion * 10^6)

   - MFLOPS = NFLOPS/ (T_ejecucion * 10^6)

   Para 10 componentes:

   - MIPS = 6 / (0.000469375 * 10^6) = 0.01278 MIPS

   - MFLOPS = 1 / (0.000469375 * 10^6) = 0.00213 MFLOPS

   Para 10000000 componenteS:

   - MIPS = 6 / (0.06 * 10^6) = 0.0001 MIPS

   - MFLOPS = 1 / (0.06 * 10^6) = 0.0000167 MFLOPS

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

```
.L6:
    movsd   0(%rbp,%rax,8), %xmm0
    addsd   0(%r13,%rax,8), %xmm0
    movsd   %xmm0, (%r14,%rax,8)
    addq    $1, %rax
    cmpl    %eax, %r12d
    ja  .L6
```

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`

```
//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; //Se puede usar drand48() para generar los valores de forma aleatoria (drand48_r() para una versión paralela)
}

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

ncgt= omp_get_wtime() - start_time;
```

Se muestra la parte del código que difiere de el código original SumaVectores.c

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[ac410@atcgrid Executables]$ time srun -pac -Aac ./sp-OpenMP-for 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000000000       / 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) /

real    0m0.104s
user    0m0.007s
sys     0m0.008s
```

```
[ac410@atcgrid Executables]$ time srun -pac -Aac ./sp-OpenMP-for 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000000000       / 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) /

real    0m0.093s
user    0m0.008s
sys     0m0.007s
```

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
{
  #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; //Se puede usar drand48() para generar los valores de forma aleatoria (drand48_r() para una versión paralela)
  }

  #pragma omp section
  for(i=N/4; i<2*N/4; i++){
      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=2*N/4; i<3*N/4; i++){
      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=3*N/4; i<N; i++){
      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)
  }
}

double start_time = omp_get_wtime();
//Calcular suma de vectores
#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<2*N/4; i++)
      v3[i] = v1[i] + v2[i];

    #pragma omp section
    for(i=2*N/4; i<3*N/4; i++)
      v3[i] = v1[i] + v2[i];

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

ncgt=omp_get_wtime() - start_time;
```

```
[ac410@atcgrid Executables]$ time srun -pac -Aac ./sp-OpenMP-sections 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000000000       / 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) /

real    0m0.117s
user    0m0.009s
sys     0m0.006s
```

```
[ac410@atcgrid Executables]$ time srun -pac -Aac ./sp-OpenMP-sections 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000000000       / 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) /

real    0m0.097s
user    0m0.009s
sys     0m0.006s
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

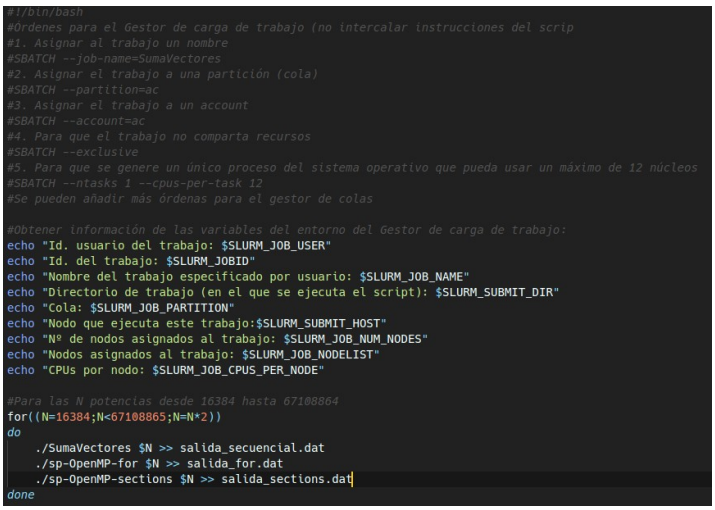
CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

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, podríamos usar un máximo de N threads, donde N es el número de hilos lógicos de los que dispone el computador en el que estemos trabajando. Además este número puede ser modificado por variables globales dentro de la sesión en la que estemos ejecutando nuestros programas (OMP_NUM_THREADS) o en nuestro código los que hayamos asignado. Sin embargo, en el ejercicio 8, como mucho usaremos 4 hebras puesto que hemos dividido nuestras regiones parallel del código en 4 secciones distintas.

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`



```
#!/bin/bash
#Órdenes para el Gestor de carga de trabajo (no intercalar instrucciones del scrip
#1. Asignar al trabajo un nombre
#SBATCH --job-name=SumaVectores
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac
#4. Para que el trabajo no comparta recursos
#SBATCH --exclusive
#5. Para que se genere un único proceso del sistema operativo que pueda usar un máximo de 12 núcleos
#SBATCH --ntasks 1 --cpus-per-task 12
#Se pueden añadir más órdenas para el gestor de colas

#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"
echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"

#Para las N potencias desde 16384 hasta 67108864
for((N=16384;N<67108865;N=N*2))
do
    ./SumaVectores $N >> salida_secuencial.dat
    ./sp-OpenMP-for $N >> salida_for.dat
    ./sp-OpenMP-sections $N >> salida_sections.dat
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

```
Tamaño Vectores:16384 (4 B)
Numero de Threads: 8
Tiempo:0.000082161»   / Tamaño Vectores:16384»    / V1[0]+V2[0]=V3[0](1638.400000+1638.400000=3276.800000) / /
V1[16383]+V2[16383]=V3[16383](3276.700000+0.100000=3276.800000) /
************************************

Tamaño Vectores:32768 (4 B)
Numero de Threads: 8
Tiempo:0.010709959»   / Tamaño Vectores:32768»    / V1[0]+V2[0]=V3[0](3276.800000+3276.800000=6553.600000) / /
V1[32767]+V2[32767]=V3[32767](6553.500000+0.100000=6553.600000) /
********************************
```

Extracto de ejecución PC

```
[ac410@atcgrid Practica_1]$ sbatch ./sp-OpenMP-script10.sh
Submitted batch job 137680
```

Extracto de ejecución del grid

TABLA PC

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.

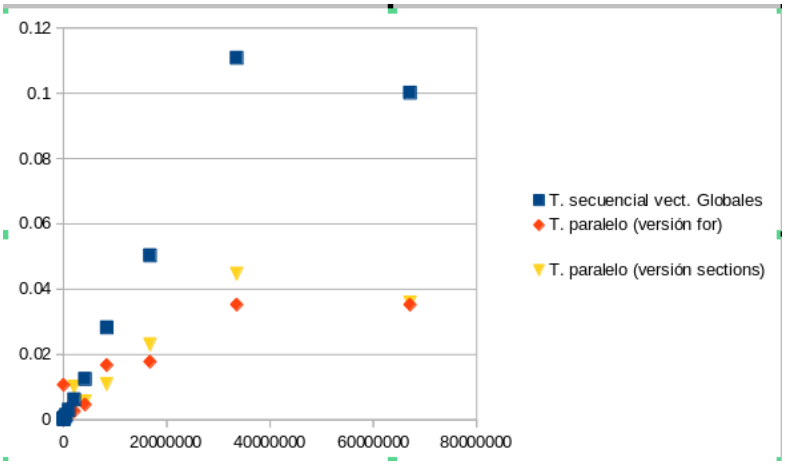
Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión `for`) 8 threads = cores lógicos = cores físicos	T. paralelo (versión `sections`) 4 threads = cores lógicos = cores físicos
16384	0.000208316	0.000082161	0.000122886
32768	0.000415447	0.010709959	0.000064403
65536	0.000290607	0.000106417	0.000105045
131072	0.000501287	0.000128263	0.000134848
262144	0.000865775	0.000190305	0.000258907
524288	0.001489720	0.000458396	0.000653148
1048576	0.003074520	0.003711719	0.001418718
2097152	0.006239798	0.002676859	0.010152828
4194304	0.012523048	0.004665856	0.005696965
8388608	0.028290708	0.016757004	0.010931184
16777216	0.050360756	0.017792153	0.023039213
33554432	0.110850010	0.035277641	0.044704074
67108864	0.100174393	0.035250088	0.035956083

TABLA ATCGRID

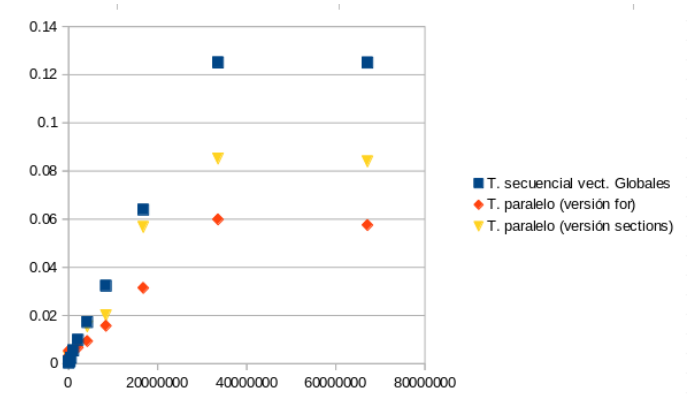
Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos y cores lógicos utilizados.

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión `for`) 12threads = cores lógicos = cores físicos	T. paralelo (versión `sections`) 4threads = cores lógicos = cores físicos
16384	0.000427654	0.003045879	0.000620098
32768	0.000860156	0.005308498	0.000670481
65536	0.000779235	0.003045489	0.002754011
131072	0.000955099	0.003367930	0.000690087
262144	0.001148898	0.003493435	0.001160005
524288	0.002470590	0.004262983	0.003816748
1048576	0.005517596	0.005110912	0.004421277
2097152	0.009912027	0.006722465	0.007672386
4194304	0.017301183	0.009362149	0.015492428
8388608	0.032359722	0.015738714	0.020077078
16777216	0.063957614	0.031449211	0.056751861
33554432	0.125092874	0.059906489	0.085180072
67108864	0.125036677	0.057552531	0.083990583

Grafica PC



Gráfica GRID



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`

```
#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"
echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"

#Redirigimos la salida para que nos muestre por pantalla solo el resultado de time

#Para las N potencias desde 16384 hasta 67108864
for((N=16384;N<67108865;N=N*2))
do
    echo "Tiempo Secuencial para $N:"
    time ./SumaVectores $N >> salida_secuencial.dat

    echo "Tiempo para paralelo $N:"
    time ./sp-OpenMP-for $N >> salida_for.dat

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/core**			**Tiempo secuencial vect. Globales 12 thread/core**		
	Elapsed	***CPU-user***	***CPU_sys***	***Elapsed***	***CPU-user***	***CPU_sys***
65536	0.003s	0.001s	0.002s	0.011s	0.068s	0.138s
131072	0.003s	0.002s	0.001s	0.010s	0.032s	0.155s
262144	0.006s	0.002s	0.004s	0.015s	0.106s	0.187s
524288	0.009s	0.001s	0.008s	0.015s	0.106s	0.167s
1048576	0.0015s	0.006s	0.009s	0.018s	0.141s	0.212s
2097152	0.027s	0.013s	0.014s	0.024s	0.266s	0.197s
4194304	0.046s	0.023s	0.023s	0.028s	0.358s	0.195s
8388608	0.084s	0.048s	0.036s	0.042s	0.662s	0.258s
16777216	0.160s	0.088s	0.071s	0.086s	1.246s	0.513s
33554432	0.316s	0.166s	0.150s	0.160s	1.246s	2.367s
67108864	0.316s	0.186s	0.130s	0.159s	2.475s	0.960s