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): David Gómez Hernández
Grupo de prácticas y profesor de prácticas: C3 María Isabel
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(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`

```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();
    }

    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>

int main()  {
  int n = 9, i, a, b[n];
  for (i=0; i<n; i++)   b[i] = -1;

  #pragma omp parallel
  {
    #pragma omp single
    {  printf("Introduce valor de inicialización a: ");
       scanf("%d", &a );
       printf("Single ejecutada por el thread %d\n",
       omp_get_thread_num());
    }

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

    #pragma omp single
    {
       for (i=0; i<n; i++)
          printf("Thread ID: %d - b[%d] = %d\t",omp_get_thread_num(),i,b[i]);
    }
  }
  return 0;
}
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer2] 2020-03-22 domingo
$sbatch -p ac script_singleModificado.sh
Submitted batch job 24309
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer2] 2020-03-22 domingo
$cat slurm-24309.out
Id. usuario del trabajo: c3estudiante8
Id. del trabajo: 24309
Nombre del trabajo especificado por usuario: singleModificado
Directorio de trabajo (en el que se ejecuta el script):
/home/c3estudiante8/bp1/ejer2
Cola: ac
Nodo que ejecuta este trabajo:atcgrid
No de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
CPUs por nodo: 2

 1. Ejecución singleModificado una vez sin cambiar no de threads (valor
por defecto):

Introduce valor de inicialización a: Single ejecutada por el thread 1
Thread ID: 4 - b[0] = 0 Thread ID: 4 - b[1] = 0 Thread ID: 4 - b[2] = 0 Thread ID: 4 - b[3] = 0 Thread ID
: 4 - b[4] = 0  Thread ID: 4 - b[5] = 0 Thread ID: 4 - b[6] = 0 Thread ID: 4 - b[7] = 0 Thread ID: 4 - b[
8] = 0
 2. Ejecución singleModificado varias veces con distinto no de threads:


 - Para 12 threads:
Introduce valor de inicialización a: Single ejecutada por el thread 1
Thread ID: 3 - b[0] = 0 Thread ID: 3 - b[1] = 0 Thread ID: 3 - b[2] = 0 Thread ID: 3 - b[3] = 0 Thread ID
: 3 - b[4] = 0  Thread ID: 3 - b[5] = 0 Thread ID: 3 - b[6] = 0 Thread ID: 3 - b[7] = 0 Thread ID: 3 - b[
8] = 0
 - Para 6 threads:
Introduce valor de inicialización a: Single ejecutada por el thread 1
Thread ID: 4 - b[0] = 0 Thread ID: 4 - b[1] = 0 Thread ID: 4 - b[2] = 0 Thread ID: 4 - b[3] = 0 Thread ID
: 4 - b[4] = 0  Thread ID: 4 - b[5] = 0 Thread ID: 4 - b[6] = 0 Thread ID: 4 - b[7] = 0 Thread ID: 4 - b[
8] = 0
 - Para 3 threads:
Introduce valor de inicialización a: Single ejecutada por el thread 2
Thread ID: 0 - b[0] = 0 Thread ID: 0 - b[1] = 0 Thread ID: 0 - b[2] = 0 Thread ID: 0 - b[3] = 0 Thread ID
: 0 - b[4] = 0  Thread ID: 0 - b[5] = 0 Thread ID: 0 - b[6] = 0 Thread ID: 0 - b[7] = 0 Thread ID: 0 - b[
8] = 0
 - Para 1 threads:
Introduce valor de inicialización a: Single ejecutada por el thread 0
Thread ID: 0 - b[0] = 0 Thread ID: 0 - b[1] = 0 Thread ID: 0 - b[2] = 0 Thread ID: 0 - b[3] = 0 Thread ID
: 0 - b[4] = 0  Thread ID: 0 - b[5] = 0 Thread ID: 0 - b[6] = 0 Thread ID: 0 - b[7] = 0 Thread ID: 0 - b[
8] = 0  [DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer2] 2020-03-22 domingo
$
```

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
4. anterior?

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

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

int main()  {
  int n = 9, i, a, b[n];

  for (i=0; i<n; i++)   b[i] = -1;
  #pragma omp parallel
  {
    #pragma omp single
    {  printf("Introduce valor de inicialización a: ");
       scanf("%d", &a );
       printf("Single ejecutada por el thread %d\n",
       omp_get_thread_num());
    }

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

    #pragma omp master
    {
       for (i=0; i<n; i++)
           printf("Thread ID: %d - b[%d] = %d\t",omp_get_thread_num(),i,b[i]);
    }
  }
  return 0;
}
```

CAPTURAS DE PANTALLA:

```
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer3] 2020-03-22 domingo
$sbatch -p ac script_singleModificado.sh
Submitted batch job 24310
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer3] 2020-03-22 domingo
$cat slurm-24310.out
Id. usuario del trabajo: c3estudiante8
Id. del trabajo: 24310
Nombre del trabajo especificado por usuario: singleModificado2
Directorio de trabajo (en el que se ejecuta el script):
/home/c3estudiante8/bp1/ejer3
Cola: ac
Nodo que ejecuta este trabajo:atcgrid
No de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
CPUs por nodo: 2

 1. Ejecución singleModificados una vez sin cambiar no de threads (valor
por defecto):

Introduce valor de inicialización a: Single ejecutada por el thread 1
Thread ID: 0 - b[0] = 0 Thread ID: 0 - b[1] = 0 Thread ID: 0 - b[2] = 0 Thread ID: 0 - b[3] = 0 Thread I
: 0 - b[4] = 0  Thread ID: 0 - b[5] = 0 Thread ID: 0 - b[6] = 0 Thread ID: 0 - b[7] = 0 Thread ID: 0 - b
8] = 0
 2. Ejecución singleModificados varias veces con distinto no de threads:


 - Para 12 threads:
Introduce valor de inicialización a: Single ejecutada por el thread 1
Thread ID: 0 - b[0] = 0 Thread ID: 0 - b[1] = 0 Thread ID: 0 - b[2] = 0 Thread ID: 0 - b[3] = 0 Thread I
: 0 - b[4] = 0  Thread ID: 0 - b[5] = 0 Thread ID: 0 - b[6] = 0 Thread ID: 0 - b[7] = 0 Thread ID: 0 - b
8] = 0
 - Para 6 threads:
Introduce valor de inicialización a: Single ejecutada por el thread 1
Thread ID: 0 - b[0] = 0 Thread ID: 0 - b[1] = 0 Thread ID: 0 - b[2] = 0 Thread ID: 0 - b[3] = 0 Thread I
: 0 - b[4] = 0  Thread ID: 0 - b[5] = 0 Thread ID: 0 - b[6] = 0 Thread ID: 0 - b[7] = 0 Thread ID: 0 - b
8] = 0
 - Para 3 threads:
Introduce valor de inicialización a: Single ejecutada por el thread 2
Thread ID: 0 - b[0] = 0 Thread ID: 0 - b[1] = 0 Thread ID: 0 - b[2] = 0 Thread ID: 0 - b[3] = 0 Thread I
: 0 - b[4] = 0  Thread ID: 0 - b[5] = 0 Thread ID: 0 - b[6] = 0 Thread ID: 0 - b[7] = 0 Thread ID: 0 - b
8] = 0
 - Para 1 threads:
Introduce valor de inicialización a: Single ejecutada por el thread 0
Thread ID: 0 - b[0] = 0 Thread ID: 0 - b[1] = 0 Thread ID: 0 - b[2] = 0 Thread ID: 0 - b[3] = 0 Thread I
: 0 - b[4] = 0  Thread ID: 0 - b[5] = 0 Thread ID: 0 - b[6] = 0 Thread ID: 0 - b[7] = 0 Thread ID: 0 - b
8] = 0  [DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer3] 2020-03-22 domingo
$
```

RESPUESTA A LA PREGUNTA:

Como el single lleva dentro una directiva master, esa parte del single será ejecutada por la hebra master, es decir, la hebra número 0. Si no llevara el master (como en el ejercicio anterior), cualquier hebra podría ejecutar esa parte del single.

5. ¿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:

Es por la característica de la directiva master al final, ya que esta no incluye una barrera implcíta, por lo tanto la hebra master puede adelantarse al resto de hebras que están realizando la operación de suma imprimiendo el resultado antes de que el resto haya acabado.

Resto de ejercicios

6. 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:

```
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer5] 2020-03-22 domingo
$sbatch -p ac --wrap "time ./SumaVectores 1000000"
Submitted batch job 24326
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer5] 2020-03-22 domingo
$cat slurm-24326.out
Tamaño Vectores:1000000 (4 B)
Tiempo:0.004907422       / Tamaño Vectores:1000000      / V1[0]+V2[0]=V3[0](100000.000000+100000.000000=2
00000.000000) / / V1[999999]+V2[999999]=V3[999999](199999.900000+0.100000=200000.000000) /

real    0m0.045s
user    0m0.006s
sys     0m0.007s
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer5] 2020-03-22 domingo
$
```

La suma de los tiempos de CPU del usuario y del sistema es menos que el tiempo real.

7. 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. Incorpore **el código ensamblador de la parte de la suma de vectores** 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):

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Github/AC/Práctica 1/ejer6]
03/22/20 17:36:16
$ gcc -O2 -S SumaVectoresC.c
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:33: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has t
ype ‘long unsigned int’ [-Wformat=]
   printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                ~^
                                %lu
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Github/AC/Práctica 1/ejer6]
03/22/20 17:36:41
$
```

```
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer6] 2020-03-22 domingo
$sbatch -p ac --wrap "./SumaVectores 10"
Submitted batch job 24436
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer6] 2020-03-22 domingo
$cat slurm-24436.out
Tamaño Vectores:10 (4 B)
Tiempo:0.000401759       / 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) /
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer6] 2020-03-22 domingo
$rm slurm-24436.out
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer6] 2020-03-22 domingo
$sbatch -p ac --wrap "./SumaVectores 10000000"
Submitted batch job 24438
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer6] 2020-03-22 domingo
$cat slurm-24438.out
Tamaño Vectores:10000000 (4 B)
Tiempo:0.043516163       / 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) /
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer6] 2020-03-22 domingo
$
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Tenemos 9 instrucciones entre ambas llamadas al getTime, teniendo en cuenta que estandoe n un for, se van a ejecutar todas las veces que el for dictamine.

Para calcular los MIPS, → numero de instrucciones / tiempo de ejecucion (10^6)

Para N = 10

((6*10)+3)/0.000401759 * 10^6 = 1,50 MIPS

Para N = 10000000

((6*10000000)+3)/0.043516163 * 10^6 = 13787.98 MIPS

Para calcular los MOPS → número de instrucciones de coma flotante (%xmn0) / tiempo ejecución (10^6)

Para N = 10

3*10/0.000401759 * 10^6 = 0,074 MOPS

Para N = 10000000

3*10000000/0.043516163 * 10^6 = 689,4 MOPS

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

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

8. 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

```
//Inicialializamos los vectores
#pragma omp parallel for
  for(int i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
  }

tiempo1 = omp_get_wtime();
//Realizamos la suma de los vectores
#pragma omp parallel for
  for(int i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];

tiempo2 = omp_get_wtime() - tiempo1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Github/AC/Práctica 1/ejer7]
03/22/20 18:25:12
$ gcc -O2 -fopenmp SumaVectoresOMP.c -o SumaVectoresOMP
SumaVectoresOMP.c: In function ‘main’:
SumaVectoresOMP.c:46:33: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has
 type ‘long unsigned int’ [-Wformat=]
   printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                               ~^
                               %lu
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Github/AC/Práctica 1/ejer7]
03/22/20 18:25:42
$
```

```
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer7] 2020-03-22 domingo
$sbatch -p ac --wrap "./SumaVectoresOMP 8"
Submitted batch job 24739
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer7] 2020-03-22 domingo
$cat slurm-24739.out
Tamaño Vectores:8 (4 B)
Tiempo:0.000056624       / 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) /
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer7] 2020-03-22 domingo
$rm slurm-24739.out
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer7] 2020-03-22 domingo
$sbatch -p ac --wrap "./SumaVectoresOMP 11"
Submitted batch job 24744
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer7] 2020-03-22 domingo
$cat slurm-24744.out
Tamaño Vectores:11 (4 B)
Tiempo:0.000054993       / 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) /
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer7] 2020-03-22 domingo
$
```

9. 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

```
//Inicialializamos los vectores dividiendo las tareas entre 2
#pragma omp parallel sections private(i)
{

  #pragma omp section
    for(i=0; i<N/2; i++)
      v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;

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

}
tiempo1 = omp_get_wtime();
//Realizamos la suma de los vectores dividiendo las tareas entre 2
#pragma omp parallel sections private(i)
{
  #pragma omp section
    for(i=0; i<N/2; i++)
      v3[i] = v1[i] + v2[i];

  #pragma omp section
    for(i=N/2; i<N; i++)
      v3[i] = v1[i] + v2[i];
}
tiempo2 = omp_get_wtime() - tiempo1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Github/AC/Práctica 1/ejer8]
03/22/20 18:47:47
$ gcc -O2 -fopenmp SumaVectoresOMP2.c -o SumaVectoresOMP2
SumaVectoresOMP2.c: In function ‘main’:
SumaVectoresOMP2.c:46:33: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 ha
s type ‘long unsigned int’ [-Wformat=]
   printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                               ~^
                               %lu
[DavidGómezHernández david@david-HP-Pavilion-Notebook:/home/david/Escritorio/Github/AC/Práctica 1/ejer8]
03/22/20 18:47:48
$
```

```
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer8] 2020-03-22 domingo
$sbatch -p ac --wrap "./SumaVectoresOMP2 8"
Submitted batch job 24893
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer8] 2020-03-22 domingo
$cat slurm-24893.out
Tamaño Vectores:8 (4 B)
Tiempo:0.000060532       / 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) /
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer8] 2020-03-22 domingo
$rm slurm-24893.out
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer8] 2020-03-22 domingo
$sbatch -p ac --wrap "./SumaVectoresOMP2 11"
Submitted batch job 24894
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer8] 2020-03-22 domingo
$cat slurm-24894.out
Tamaño Vectores:11 (4 B)
Tiempo:0.000058502       / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[1
]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
[DavidGómezHernández c3estudiante8@atcgrid:~/bp1/ejer8] 2020-03-22 domingo
$
```

10. ¿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.

    RESPUESTA:

    Ejercicio 7 → El número máximo de hebras será igual al número de componentes del vector.

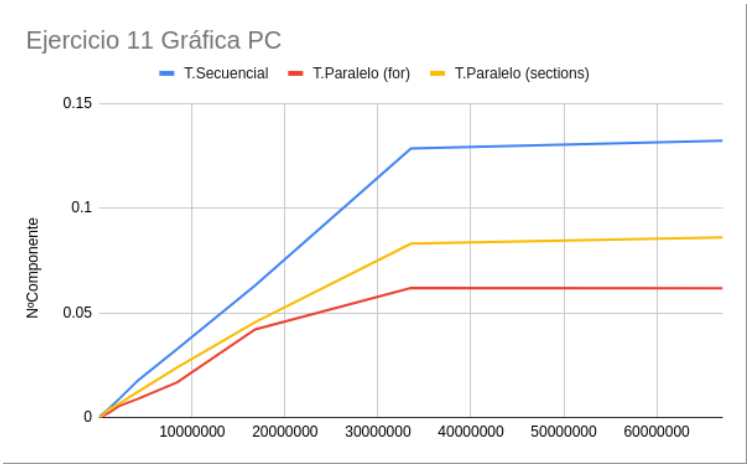
    Ejercicio 8 → El número máximo de hebras será igual al número de sections

11. 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. 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.

    RESPUESTA:

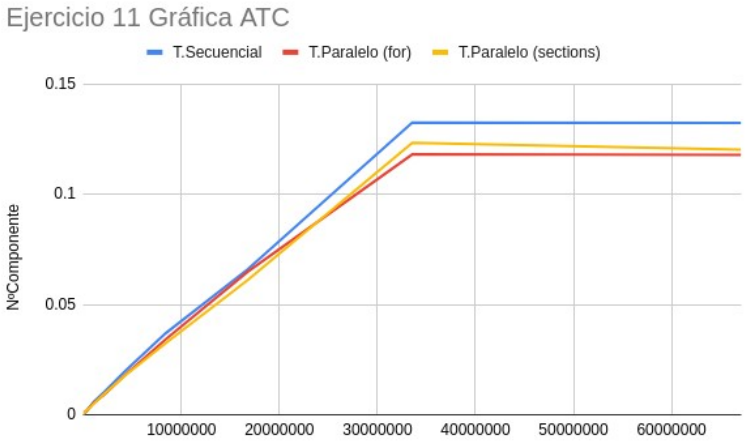
    Ejecución en PC

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) 12 threads/cores**	**T. paralelo (versión `sections`) 12 threads/cores**
16384	0.00021792	0.000056023	0.000050559
32768	0.000511147	0.000046448	0.000081763
65536	0.000385159	0.000057556	0.000259514
131072	0.000566244	0.000201193	0.000297516
262144	0.001330452	0.000303113	0.00070254
524288	0.002151147	0.001062782	0.001809325
1048576	0.004216493	0.002190362	0.003975835
2097152	0.00864958	0.005419278	0.006399797
4194304	0.017812393	0.008921464	0.012291357
8388608	0.032658137	0.016817574	0.023920015
16777216	0.063099412	0.042059784	0.045473678
33554432	0.128531816	0.061872015	0.083006099
67108864	0.132187019	0.061677633	0.086068187



Ejecución en ATCGRID

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) 12 threads/cores	T. paralelo (versión `sections`) 12 threads/cores
16384	0.000434959	0.00047685	0.000497639
32768	0.00046505	0.000531886	0.000895092
65536	0.000966399	0.000859173	0.00062195
131072	0.000695465	0.001386119	0.000986136
262144	0.00142324	0.001339113	0.001797987
524288	0.00290557	0.002690492	0.002627574
1048576	0.005779061	0.005341476	0.004932012
2097152	0.010183198	0.008976862	0.009322997
4194304	0.019501167	0.017764885	0.017561182
8388608	0.03694794	0.03414768	0.032392286
16777216	0.066006314	0.064901359	0.061147999
33554432	0.132569835	0.118114967	0.123366429
67108864	0.132325957	0.117817586	0.120286968



12. 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/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.
RESPUESTA:

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 paralelo/versión `for` 12 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	00.00	0.00	0.00	00.00	0.00	0.00
131072	00.00	0.00	0.00	00.00	0.00	0.00
262144	00.00	0.00	0.00	00.00	0.00	0.00
524288	00.00	0.00	0.00	00.00	0.00	0.00
1048576	00.00	0.00	0.00	00.00	0.00	0.00
2097152	0.02	0.01	0.01	00.02	0.01	0.01
4194304	00.04	0.03	0.01	00.04	0.02	0.02
8388608	00.09	0.04	0.04	00.09	0.05	0.03
16777216	00.16	0.09	0.07	00.15	0.08	0.07
33554432	00.31	0.16	0.14	00.28	0.33	0.21
67108864	00.32	0.15	0.17	00.28	0.33	0.22