

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: Juan Carlos Ruiz Fernández

Grupo de prácticas y profesor de prácticas: B3 Dª Mancia Anguita

Fecha de entrega: 13/04/2021

Fecha evaluación en clase: 14/04/2021

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

```
ejer1 > C bucle-forMODIFICADO.c > main(int, char **)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4  int main(int argc, char **argv)
5  {
6      int i, n = 9;
7      if (argc < 2)
8      {
9          fprintf(stderr, "\n[ERROR] - Falta nº iteraciones \n");
10         exit(-1);
11     }
12     n = atoi(argv[1]);
13     #pragma omp parallel for
14     {
15         for (i = 0; i < n; i++)
16             printf("thread %d ejecuta la iteración %d del bucle\n",
17                 omp_get_thread_num(), i);
18     }
19     return (0);
20 }
```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

```
C sectionsMODIFICADO.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3  void funcA()
4  {
5      printf("En funcA: esta sección la ejecuta el thread
6              %d\n",
7              omp_get_thread_num());
8  }
9  void funcB()
10 {
11     printf("En funcB: esta sección la ejecuta el thread
12           %d\n",
13           omp_get_thread_num());
14 }
15 main()
16 {
17     #pragma omp parallel sections
18     {
19         #pragma omp section
20         (void)funcA();
21         #pragma omp section
22         (void)funcB();
23     }
24 }
```

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

```

ejer2 > C singleModificado.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3  main()
4  {
5      int n = 9, i, a, b[n];
6      for (i = 0; i < n; i++)
7          b[i] = -1;
8      #pragma omp parallel
9      {
10         #pragma omp single
11         {
12             printf("Introduce valor de inicialización a : ");
13             scanf("%d", &a);
14             printf("Single ejecutada por el thread %d\n",
15                 omp_get_thread_num());
16         }
17         #pragma omp for
18         for (i = 0; i < n; i++)
19             b[i] = a;
20
21         #pragma omp single
22         {
23             printf("Después de la región parallel con thread %d:\n", omp_get_thread_num());
24             for (i = 0; i < n; i++)
25                 printf("b[%d] = %d\t", i, b[i]);
26             printf("\n");
27         }
28     }
29 }

```

CAPTURAS DE PANTALLA:

```

C:\Windows\system32\wsl.exe
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo/2ºCuatri/AC/Prácticas/BP1/ejer2] 2021-03-20 Saturday
$export OMP_NUM_THREADS=4
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo/2ºCuatri/AC/Prácticas/BP1/ejer2] 2021-03-20 Saturday
$gcc -O2 -fopenmp -o singleModificado singleModificado.c
singleModificado.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
3 | main()
  | ^~~~~
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo/2ºCuatri/AC/Prácticas/BP1/ejer2] 2021-03-20 Saturday
$./singleModificado
Introduce valor de inicialización a : 5
Single ejecutada por el thread 2
Después de la región parallel con thread n0:
b[0] = 5      b[1] = 5      b[2] = 5      b[3] = 5      b[4] = 5      b[5] = 5      b[6] = 5      b[7] = 5
b[8] = 5
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo/2ºCuatri/AC/Prácticas/BP1/ejer2] 2021-03-20 Saturday
$

```

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?

```

ejer3 > C singleModificado2.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3  main()
4  {
5      int n = 9, i, a, b[n];
6      for (i = 0; i < n; i++)
7          b[i] = -1;
8      #pragma omp parallel
9      {
10         #pragma omp single
11         {
12             printf("Introduce valor de inicialización a : ");
13             scanf("%d", &a);
14             printf("Single ejecutada por el thread %d\n",
15                 omp_get_thread_num());
16         }
17         #pragma omp for
18         for (i = 0; i < n; i++)
19             b[i] = a;
20
21         #pragma omp master
22         {
23             printf("Después de la región parallel con thread %d:\n", omp_get_thread_num());
24             for (i = 0; i < n; i++)
25                 printf("b[%d] = %d\t", i, b[i]);
26             printf("\n");
27         }
28     }
29 }

```

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

CAPTURAS DE PANTALLA:

```

C:\Windows\system32\cmd.exe
[juanca@DESKTOP-RCIHQK2: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP1/ejer3] 2021-03-20 Saturday
$gcc -O2 -fopenmp -o singleModificado2 singleModificado2.c
singleModificado2.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
   3 | main()
     | ^~~~~
[juanca@DESKTOP-RCIHQK2: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP1/ejer3] 2021-03-20 Saturday
$./singleModificado2
Introduce valor de inicialización a : 5
Single ejecutada por el thread 3
Después de la región parallel con thread n0:
b[0] = 5      b[1] = 5      b[2] = 5      b[3] = 5      b[4] = 5      b[5] = 5      b[6] = 5      b[7] = 5
[8] = 5
[juanca@DESKTOP-RCIHQK2: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP1/ejer3] 2021-03-20 Saturday
$

```

RESPUESTA A LA PREGUNTA:

El número de thread se convierte esta vez en 0, que es la hebra master. Antes, aunque fuese la dos, pudo haber sido cualquier otra.

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:

Por que barrier hace esperar a todas las hebras, cada hebra incrementa la variable suma. Si no se pusiera la hebra master (0) podria terminar antes que ninguna e imprimir el resultado sin ser el final esperado.

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 ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i=0, \dots, N-1$). 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:

```

b3estudiante23@atcgrid:~/BP1/ejer5
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer5] 2021-03-30 Tuesday
$ sbatch -pac --wrap "time ./sumavectores 10000000"
Submitted batch job 77073
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer5] 2021-03-30 Tuesday
$ cat slurm-77073.out
Tiempo:0.040126480 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](0.228026+2.027117=2.255143) / / V1[9999999]+
V2[9999999]=V3[9999999](1.514554+4.722218=6.236771) /

real    0m0.580s
user    0m0.516s
sys     0m0.054s
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer5] 2021-03-30 Tuesday
$

```

RESPUESTA:

La suma de *user* y *sys* da como resultado 0.570 que está un poco alejado del *real*. Viene porque el tiempo *elapsed (real)* cuenta además las I/Os y otros programas en ejecución.

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):

```

b3estudiante23@atcgrid:~/BP1/ejer6
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer6] 2021-03-31 Wednesday
$ gcc -O2 -S sumavectores.c
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer6] 2021-03-31 Wednesday
$ gcc -O2 sumavectores.c -o sumavectores
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer6] 2021-03-31 Wednesday
$

```

```

Seleccinar b3estudiante23@atcgrid:~/BP1/ejer6
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer6] 2021-03-31 Wednesday
$ sbatch -pac --wrap "./sumavectores 10"
Submitted batch job 77357
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer6] 2021-03-31 Wednesday
$ cat slurm-77357.out
Tiempo:0.000392074 / Tamaño Vectores:10 / V1[0]+V2[0]=V3[0](1.693424+1.933856=3.627281) / / V1[9]
]+V2[9]=V3[9](0.041325+6.884137=6.925462) /
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer6] 2021-03-31 Wednesday
$ sbatch -pac --wrap "./sumavectores 10000000"
Submitted batch job 77358
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer6] 2021-03-31 Wednesday
$ cat slurm-77358.out
Tiempo:0.039810776 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](0.205866+0.675598=0.881464)
/ / V1[9999999]+V2[9999999]=V3[9999999](1.263491+1.661602=2.925093) /
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer6] 2021-03-31 Wednesday
$

```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Tamaño = 10	Tamaño=10000000
$NI = 6 * 10 + 3 = 63$	$NI = 6 * 10000000 + 3 = 60000003$
$MIPS = \frac{NI}{T_{cpu} * 10^6} = \frac{63}{0.000392074 * 10^6} = 0.160683953$	$MIPS = \frac{NI}{T_{cpu} * 10^6} = \frac{60000003}{0.039810776 * 10^6} = 1507.129703$
$MFLOPS = \frac{OpsFloat}{T_{cpu} * 10^6} = \frac{7 * 10}{0.000392074 * 10^6} = 0.178537725$	$MFLOPS = \frac{OpsFloat}{T_{cpu} * 10^6} = \frac{7 * 100000000}{0.039810776 * 10^6} = 1758.317899$

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

```

.L6:
    movq    %rsp, %rsi
    xorl    %edi, %edi
    call    clock_gettime
    xorl    %eax, %eax
    .p2align 4,,10
    .p2align 3
.L8:
    movsd   v1(,%rax,8), %xmm0
    addsd   v2(,%rax,8), %xmm0
    movsd   %xmm0, v3(,%rax,8)
    addq    $1, %rax
    cmpl    %eax, %ebp
    ja      .L8
    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 ($v_3 = v_1 + v_2$; $v_3(i) = v_1(i) + v_2(i)$, $i = 0, \dots, N-1$) 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, v_3 , 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 v_1 , v_2 y v_3 (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

The image displays three side-by-side screenshots of a C program for vector addition, each showing a different compilation flag and its execution results.

Left Screenshot (C sp-OpenMP-forc.x): Shows the program compiled with `-fopenmp`. The code defines a vector of size `N` (1000) and calculates the sum of two vectors. The output shows the time taken for the calculation: `Time: 11.9f / Tamaño Vectores: 1000`.

Middle Screenshot (C sp-OpenMP-forc.x): Shows the program compiled with `-fopenmp` and `-xopenmp`. The code is identical to the first screenshot. The output shows the time taken for the calculation: `Time: 11.9f / Tamaño Vectores: 1000`.

Right Screenshot (C sp-OpenMP-forc.x): Shows the program compiled with `-fopenmp` and `-xopenmp`. The code is identical to the first screenshot. The output shows the time taken for the calculation: `Time: 11.9f / Tamaño Vectores: 1000`.

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
b3estudiante23@atcggrid:~/BP1/ejer7
[JuanCarlosRuizFernandez b3estudiante23@atcggrid:~/BP1/ejer7] 2021-03-31 Wednesday
$gcc -O2 -fopenmp -o sp-OpenMP-for sp-OpenMP-for.c
[JuanCarlosRuizFernandez b3estudiante23@atcggrid:~/BP1/ejer7] 2021-03-31 Wednesday
$sbatch -pac --wrap "./sp-OpenMP-for 8"
Submitted batch job 77410
[JuanCarlosRuizFernandez b3estudiante23@atcggrid:~/BP1/ejer7] 2021-03-31 Wednesday
$cat slurm-77410.out
Tiempo:0.000743132 / TamaÃ±o Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[JuanCarlosRuizFernandez b3estudiante23@atcggrid:~/BP1/ejer7] 2021-03-31 Wednesday
$sbatch -pac --wrap "./sp-OpenMP-for 11"
Submitted batch job 77411
[JuanCarlosRuizFernandez b3estudiante23@atcggrid:~/BP1/ejer7] 2021-03-31 Wednesday
$cat slurm-77411.out
Tiempo:0.000555348 / TamaÃ±o Vectores:11 / V1[0]+V2[0]=V3[0](0.661862+0.681386=
1.343248) / / V1[10]+V2[10]=V3[10](2.758294+1.304915=4.063209) /
[JuanCarlosRuizFernandez b3estudiante23@atcggrid:~/BP1/ejer7] 2021-03-31 Wednesday
$
```

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

```
File Edit Selection View Go Run Terminal Help
C sp-OpenMP-section.c x ...
ejér8 > C sp-OpenMP-section.c > main(int, char **)
20 //define MAX 4294967295 //~2^32 -1
21
22 double v1[MAX], v2[MAX], v3[MAX];
23 #endif
24 int main(int argc, char **argv)
25 {
26
27     int i;
28
29     double cgt1, cgt2, ncgt;
30
31     //Leer argumento de entrada (nÃº de componentes
32     if (argc < 2){
33         printf("Faltan nÃº componentes del vector\n");
34         exit(-1);
35     }
36
37     unsigned int N = atoi(argv[1]); // MÃ¡ximo N=2^
38     //printf("TamaÃ±o Vectores:\u000a %u B)\n",N, sizeof
39     #ifdef VECTOR_LOCAL
40         double v1[N], v2[N], v3[N]; // TamaÃ±o variable
41         // disponible en C a partir de C99
42     #endif
43     #ifdef VECTOR_GLOBAL
44         if (N > MAX)
45             N = MAX;
46     #endif
47     #ifdef VECTOR_DYNAMIC
48         double *v1, *v2, *v3;
49         v1 = (double *)malloc(N * sizeof(double)); //
50         v2 = (double *)malloc(N * sizeof(double)); //
51         v3 = (double *)malloc(N * sizeof(double)); //
52         if ((v1 == NULL) || (v2 == NULL) || (v3 == NULL)
53             {
54             printf("No hay suficiente espacio para los v
55             exit(-2);
56         }
57     #endif
58     srand(time(0));
59     #pragma omp parallel sections private(i)
60     {
61         #pragma omp section
62         {
63             for (i = 0; i < N/4; i++)// ##
64             {
65                 v1[i] = rand() / ((double)rand());
66                 v2[i] = rand() / ((double)rand());
67             }
68         }
69         #pragma omp section
70         {
71             for (i = N/4; i < N/2; i++) // ##
72             {
73                 v1[i] = rand() / ((double)rand());
74                 v2[i] = rand() / ((double)rand());
75             }
76         }
77         #pragma omp section
78         {
79             for (i = N/2; i < 3*N/4; i++) // ##
80             {
81                 v1[i] = rand() / ((double)rand());
82                 v2[i] = rand() / ((double)rand());
83             }
84         }
85         #pragma omp section
86         {
87             for (i = 3*N/4; i < N; i++)// ##
88             {
89                 v1[i] = rand() / ((double)rand());
90                 v2[i] = rand() / ((double)rand());
91             }
92         }
93     }
94     cgt1 = omp_get_wtime();
95
96     #pragma omp parallel sections private(i)
97     {
98         #pragma omp section // ##
99         {
100             for (i = 0; i < N/4; i++)
101                 v3[i] = v1[i] + v2[i];
102         }
103         #pragma omp section // ##
104         {
105             for (i = N/4; i < N/2; i++)
106                 v3[i] = v1[i] + v2[i];
107         }
108         #pragma omp section // ##
109         {
110             for (i = N/2; i < 3*N/4; i++)
111                 v3[i] = v1[i] + v2[i];
112         }
113         #pragma omp section // ##
114         {
115             for (i = 3*N/4; i < N; i++)
116                 v3[i] = v1[i] + v2[i];
117         }
118     }
119     cgt2 = omp_get_wtime();
120     ncgt = cgt2 - cgt1;
121
122     //Imprimir resultado de la suma y el tiempo de e
123     if (N < 10)
124     {
125         printf("Tiempo:\u0011.9f\t / TamaÃ±o Vectores:\u000a
126         for (i = 0; i < N; i++)
127             printf("/ v1[%d]+v2[%d]=v3[%d]\u000a.6f+.8f+%.
128             i, i, i, v1[i], v2[i], v3[i]);
129     }
130     else
131         printf("Tiempo:\u0011.9f\t / TamaÃ±o Vectores:\u000a
132         ncgt, N, v1[0], v2[0], v3[0], N - 1, N
133
134     #ifdef VECTOR_DYNAMIC
135         free(v1); // libera el espacio reservado para v1
136         free(v2); // libera el espacio reservado para v2
137     #endif
138 }
```

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

```

b3estudiante23@atcgrid:~/BP1/ejer8
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer8] 2021-04-07 Wednesday
$gcc -O2 -fopenmp sp-OpenMP-section.c -o sp-OpenMP-section
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer8] 2021-04-07 Wednesday
$ll
total 24
-rwxrwxr-x 1 b3estudiante23 b3estudiante23 12992 Apr 7 12:58 sp-OpenMP-section
-rwxrwxr-x 1 b3estudiante23 b3estudiante23 4169 Apr 7 12:57 sp-OpenMP-section.c
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer8] 2021-04-07 Wednesday
$sbatch -pac --wrap "./sp-OpenMP-section 8"
Submitted batch job 79491
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer8] 2021-04-07 Wednesday
$cat slurm-79491.out
Tiempo:0.000499386 / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](3.457864+0.389709=3.847573) /
/ V1[1]+V2[1]=V3[1](0.946117+13.582312=14.528429) /
/ V1[2]+V2[2]=V3[2](3.125780+1.382228=4.508008) /
/ V1[3]+V2[3]=V3[3](1.190013+0.276875=1.466888) /
/ V1[4]+V2[4]=V3[4](1.820279+3.792062=5.612341) /
/ V1[5]+V2[5]=V3[5](0.301206+0.623295=0.924501) /
/ V1[6]+V2[6]=V3[6](1.890967+0.065135=1.956102) /
/ V1[7]+V2[7]=V3[7](1.472121+0.625883=2.098004) /
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer8] 2021-04-07 Wednesday
$sbatch -pac --wrap "./sp-OpenMP-section 11"
Submitted batch job 79493
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP1/ejer8] 2021-04-07 Wednesday
$cat slurm-79493.out
Tiempo:0.000555195 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](0.685158+0.752811=1.437968) / / V1[10]+V2[10]=V3[10](0.645434+2.606071=3.251506) /

```


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**. ¿sections?

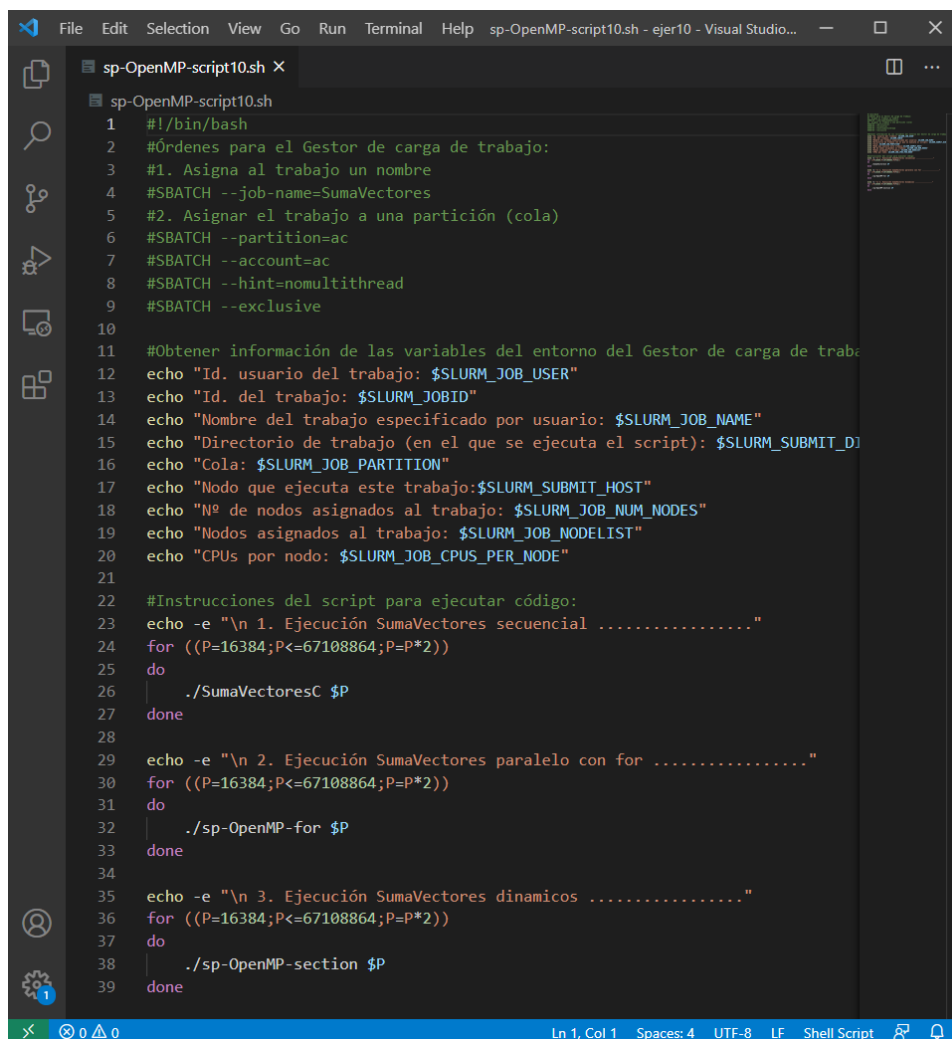
RESPUESTA:

Para el ejercicio 7 daría igual cuantos se usen ya que al usar *parallel for* se reparte entre todos. Como mucho dejaría de hacer efecto a partir de repartir una hebra por cada elemento del vector.

Para el ejercicio 8 es claramente diferente ya que al usar *sections* se usa una hebra por cada seccion. He usado cuatro secciones pues serian 4 threads. Ya que a partir de ahí no se usarían más thread.

10. Rellenar una tabla como la Tabla 2 para ategrid 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



```

1  #!/bin/bash
2  #Órdenes para el Gestor de carga de trabajo:
3  #1. Asigna al trabajo un nombre
4  #SBATCH --job-name=SumaVectores
5  #2. Asignar el trabajo a una partición (cola)
6  #SBATCH --partition=ac
7  #SBATCH --account=ac
8  #SBATCH --hint=nomultithread
9  #SBATCH --exclusive
10
11 #Obtener información de las variables del entorno del Gestor de carga de trabajo
12 echo "Id. usuario del trabajo: $SLURM_JOB_USER"
13 echo "Id. del trabajo: $SLURM_JOBID"
14 echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
15 echo "Directorio de trabajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
16 echo "Cola: $SLURM_JOB_PARTITION"
17 echo "Nodo que ejecuta este trabajo: $SLURM_SUBMIT_HOST"
18 echo "Nº de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
19 echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
20 echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
21
22 #Instrucciones del script para ejecutar código:
23 echo -e "\n 1. Ejecución SumaVectores secuencial ....."
24 for ((P=16384;P<=67108864;P*=2))
25 do
26     ./SumaVectoresC $P
27 done
28
29 echo -e "\n 2. Ejecución SumaVectores paralelo con for ....."
30 for ((P=16384;P<=67108864;P*=2))
31 do
32     ./sp-OpenMP-for $P
33 done
34
35 echo -e "\n 3. Ejecución SumaVectores dinámicos ....."
36 for ((P=16384;P<=67108864;P*=2))
37 do
38     ./sp-OpenMP-section $P
39 done
  
```

PC

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) 6 threads = cores lógicos = cores físicos	T. paralelo (versión sections) 4threads = cores lógicos = cores físicos
16384	0.000250700	0.000051800	0.000146800
32768	0.000251800	0.000105700	0.000269400
65536	0.000517500	0.000329100	0.000679600
131072	0.001054800	0.000440100	0.001329900
262144	0.002239300	0.000872600	0.000677700
524288	0.004352900	0.001792300	0.001433200
1048576	0.009188600	0.003600300	0.002829400
2097152	0.017812200	0.005588200	0.005579700
4194304	0.035427400	0.011081100	0.011042700
8388608	0.070997500	0.022124800	0.022143800
16777216	0.143583800	0.047111200	0.043986200
33554432	0.289759900	0.087971200	0.088945100
67108864	0.565269100	0.087723200	0.088919700

ATCGRID

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) 6 threads = cores lógicos = cores físicos	T. paralelo (versión sections) 4threads = cores lógicos = cores físicos
16384	0.000074812	0.003720209	0.000772145
32768	0.000189204	0.004406378	0.000765312
65536	0.000339872	0.002773285	0.000694402
131072	0.000682281	0.004620351	0.000811465
262144	0.001394084	0.006682668	0.001600377
524288	0.002554423	0.006162725	0.002713896
1048576	0.004584026	0.007651083	0.004400183
2097152	0.008412979	0.008470867	0.007871754
4194304	0.016319425	0.015543226	0.014997181
8388608	0.031789341	0.028322849	0.029561102
16777216	0.063030133	0.062946230	0.056317754
33554432	0.125763755	0.101037581	0.113159031
67108864	0.255399517	0.115689002	0.114760682

