

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 Martinez Diaz

Grupo de prácticas y profesor de prácticas: 2 – Juan José Escobar Perez

Fecha de entrega: 05/04/2021

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 bucle-forModificado.c X C bucle-for.c
D: > Usuario > Desktop > Inf + Ade > 2 curso > 2 cuatrimestre > AC > AC_Practicas > bp1_MartinezDiazDavid > ejer1 > C bucle-forModificado.c > main(int, char
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 int main(int argc, char **argv) {
6
7     int i, n = 9;
8     if(argc < 2) {
9
10         fprintf(stderr, "\n[ERROR] - Falta no iteraciones \n");
11         exit(-1);
12     }
13
14     n = atoi(argv[1]);
15     #pragma omp parallel for
16     for (i=0; i<n; i++)
17         printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(), i);
18
19     return(0);
20 }
21
```

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

```
C sections.c X C sectionsModificado.c X
D: > Usuario > Desktop > Inf + Ade > 2 curso > 2 cuatrimestre > AC > AC_Practicas > bp1_MartinezDiazDavid > ejer1 > C sectionsModificado.c > main()
1 #include <stdio.h>
2 #include <omp.h>
3
4 void funcA(){
5
6     printf("En funcA: esta seccion la ejecuta el thread %d\n", omp_get_thread_num());
7 }
8
9 void funcB(){
10
11     printf("En funcB: esta seccion la ejecuta el thread %d\n", omp_get_thread_num());
12 }
13
14
15 int main(){
16
17     #pragma omp parallel sections
18     {
19         #pragma omp section
20         (void)funcA();
21         #pragma omp section
22         (void)funcB();
23     }
24
25     return 0;
26 }
27
28
```

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`

```

1  #include <stdio.h>
2  #include <omp.h>
3
4  int main(){
5      int n = 9, i, a, b[n];
6
7      for(i = 0; i < n; i++) b[i] = -1;
8
9      #pragma omp parallel
10     {
11         #pragma omp single
12         {
13             printf("Introduce valor de inicializacion a: ");
14             scanf("%d", &a);
15             printf("Single(introducir valor) ejecutada por el thread %d\n", omp_get_thread_num());
16         }
17
18         #pragma omp for
19         for(i = 0; i < n; i++)
20             b[i] = a;
21
22         #pragma omp single
23         {
24             printf("\n Impresion ejecutada por el thread %d\n", omp_get_thread_num());
25             for(i = 0; i < n; i++) printf("b[%d] = %d [hebra %d]\n", i, b[i], omp_get_thread_num());
26         }
27     }
28
29     return 0;
30 }
31
32

```

CAPTURAS DE PANTALLA:

```

DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1_MartinezDiazDavid | cd ejer2
DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer2 | ls | 22:15:16
single.c singleModificado singleModificado.c
DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer2 | ./singleModificado
Introduce valor de inicializacion a: 7
Single(introducir valor) ejecutada por el thread 0

Impresion ejecutada por el thread 8
b[0] = 7 [hebra 8]
b[1] = 7 [hebra 8]
b[2] = 7 [hebra 8]
b[3] = 7 [hebra 8]
b[4] = 7 [hebra 8]
b[5] = 7 [hebra 8]
b[6] = 7 [hebra 8]
b[7] = 7 [hebra 8]
b[8] = 7 [hebra 8]
DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer2 | | 22:15:32

```

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`

```

1  #include <stdio.h>
2  #include <omp.h>
3
4  int main(){
5      int n = 9, i, a, b[n];
6
7      for(i = 0; i < n; i++) b[i] = -1;
8
9      #pragma omp parallel
10     {
11         #pragma omp single
12         {
13             printf("Introduce valor de inicializacion a: ");
14             scanf("%d", &a);
15             printf("Single(introducir valor) ejecutada por el thread %d\n", omp_get_thread_num());
16         }
17
18         #pragma omp for
19         for(i = 0; i < n; i++)
20             b[i] = a;
21
22         #pragma omp master
23         {
24             printf("\n Impresion ejecutada por el thread %d\n", omp_get_thread_num());
25             for(i = 0; i < n; i++) printf("b[%d] = %d [hebra %d]\n", i, b[i], omp_get_thread_num());
26         }
27     }
28
29     return 0;
30 }
31
32

```

CAPTURAS DE PANTALLA:

```

DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer3 | ls | 22:16:05
single.c singleModificado2 singleModificado2.c
DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer3 | ./singleModificado2
Introduce valor de inicializacion a: 4
Single(introducir valor) ejecutada por el thread 0

Impresion ejecutada por el thread 0
b[0] = 4 [hebra 0]
b[1] = 4 [hebra 0]
b[2] = 4 [hebra 0]
b[3] = 4 [hebra 0]
b[4] = 4 [hebra 0]
b[5] = 4 [hebra 0]
b[6] = 4 [hebra 0]
b[7] = 4 [hebra 0]
b[8] = 4 [hebra 0]
DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer3 | | 22:16:12

```

RESPUESTA A LA PREGUNTA:

La principal diferencia es que en la directiva `single` se ejecuta cualquier hebra de las disponibles y en la directiva `master` siempre se ejecuta la hebra principal, que en este caso es la hebra 0.

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:

Esto se debe a que la directiva `master` no tiene unas barreras implícitas, si quitamos dicha barrera, la que se pone antes de que se ejecute la directiva `master`, puede que a la hora de calcular la suma se realice antes de que terminen las demás hebras, dando lugar a resultados erróneos. Es decir lo que hace la directiva `barrier` es esperar a que terminen todas las hebras, y luego continua.

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:

```
[e2estudiante14@atcgrid ejer5]$ ls
SumaVectoresC.c
[e2estudiante14@atcgrid ejer5]$ gcc -O2 -fopenmp SumaVectoresC.c -o SumaVectoresC
[e2estudiante14@atcgrid ejer5]$ ls
SumaVectoresC SumaVectoresC.c
[e2estudiante14@atcgrid ejer5]$ sbatch -p ac --wrap "time ./SumaVectoresC 10000000"
Submitted batch job 78201
[e2estudiante14@atcgrid ejer5]$ ls
SumaVectoresC SumaVectoresC.c slurm-78201.out
[e2estudiante14@atcgrid ejer5]$ cat slurm-78201.out
Tamaño Vectores:10000000 (4 B)
Tiempo:0.043746392 / 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    0m0.150s
user    0m0.064s
sys     0m0.050s
[e2estudiante14@atcgrid ejer5]$
```

RESPUESTA:

Esto se debe a que el tiempo real aunque sea mayor, no significa que tarde eso el `cpu` del usuario y la `cpu` del sistema, ya que el programa puede estar esperando sin estar ejecutando nada y aun así el tiempo real estaría contando ese tiempo de espera.

Además existe el caso en el que el tiempo sea menor, ya que se podrían dividir el trabajo los hijos o `threads` que al final se sumarían al usuario y al sistema.

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando `-S` en lugar de `-o`). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para `atcgrid` los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of Floating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Razonar cómo se han obtenido

los valores que se necesitan para calcular los MIPS y MFLOPS. Incorporar **el código ensamblador de la parte de la suma de vectores** (no de todo el programa) en el cuaderno.

CAPTURAS DE PANTALLA (que muestren la generación del código ensamblador y del código ejecutable, y la obtención de los tiempos de ejecución):

```
[e2estudiante14@atcgrid ejer6]$ gcc -O2 -fopenmp SumaVectoresC.c -S SumaVectoresC
gcc: warning: SumaVectoresC: linker input file unused because linking not done
[e2estudiante14@atcgrid ejer6]$ ls
SumaVectoresC SumaVectoresC.c SumaVectoresC.s
[e2estudiante14@atcgrid ejer6]$ sbatch -p ac --wrap "time ./SumaVectoresC 10"
Submitted batch job 78208
[e2estudiante14@atcgrid ejer6]$ cat slurm-78208.out
Tamaño Vectores:10 (4 B)
Tiempo:0.000390236 / 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) /
real    0m0.024s
user    0m0.000s
sys      0m0.003s
[e2estudiante14@atcgrid ejer6]$ sbatch -p ac --wrap "time ./SumaVectoresC 10000000"
Submitted batch job 78209
[e2estudiante14@atcgrid ejer6]$ ls
SumaVectoresC SumaVectoresC.s slurm-78209.out
SumaVectoresC.c slurm-78208.out
[e2estudiante14@atcgrid ejer6]$ cat slurm-78209.out
Tamaño Vectores:10000000 (4 B)
Tiempo:0.041657257 / 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    0m0.143s
user    0m0.056s
sys      0m0.054s
[e2estudiante14@atcgrid ejer6]$
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Caso: N 10;

Nº de Instrucciones: 6;

Tiempo: 0,000390236 s

MIPS = $60 / 0,000390236 * 10^6 \rightarrow 0.1537$;

MFLOPS = $10 / 0,000390236 * 10^6 \rightarrow 2,56 * 10^{(-3)}$;

Caso: N 10000000;

Nº de Instrucciones: 6;

Tiempo: 0,041657257 s

MIPS = $60000000 / 0,041657257 * 10^6 \rightarrow 144,032$;

MFLOPS = $10000000 / 0,041657257 * 10^6 \rightarrow 240,0542119$;

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

```

.L4:
    pxor    %xmm0, %xmm0
    movapd  %xmm1, %xmm2
    movapd  %xmm1, %xmm7
    cvtsi2sdl    %eax, %xmm0
    mulsd   %xmm3, %xmm0
    addsd   %xmm0, %xmm2
    subsd   %xmm0, %xmm7
    movsd   %xmm2, v1(,%rax,8)
    movsd   %xmm7, v2(,%rax,8)
    addq    $1, %rax
    cmpl    %eax, %ebp
    ja      .L4
    movq    %rsp, %rsi
    xorl    %edi, %edi
    call    clock_gettime
    xorl    %eax, %eax
    .p2align 4,,10
    .p2align 3

.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
    movq    24(%rsp), %rax
    pxor    %xmm0, %xmm0
    subq    8(%rsp), %rax
    cvtsi2sdl    %rax, %xmm0

```

74,7-17 36%

7. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i = 0, \dots, N-1$) 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, $v3$, para varios tamaños pequeños de los vectores (por ejemplo, $N = 8$ y $N = 11$); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de $v1$, $v2$ y $v3$ (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: Captura que muestre el código fuente implementado `sp-OpenMP-for.c`

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

// Creamos variable para el omp_time
double tiempo1 = omp_get_wtime();

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

// Recogemos el tiempo que tarda en calcular la suma de vectores
double tiempo2 = omp_get_wtime();
double tiempo_total = tiempo2 - tiempo1;

//Imprimir resultado de la suma y el tiempo de ejecucion
if (N<10) {
    printf("Tiempo:%11.9f\t / Tamano Vectores:%u\n",tiempo_total,N);
    #pragma omp parallel for
    for(i=0; i<N; i++){
        printf(" / v1[%d]+v2[%d]=v3[%d](%8.6f+%8.6f=%8.6f) /\n",
            i,i,v1[i],v2[i],v3[i]);
    }
}
else
    printf("Tiempo:%11.9f\t / Tamano Vectores:%u\t/ v1[0]+v2[0]=v3[0](%8.6f+%8.6f=%8.6f) / / v1[%d]+v2[%d]
    | | | tiempo_total,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

#ifdef VECTOR_DYNAMIC
free(v1); // libera el espacio reservado para v1
free(v2); // libera el espacio reservado para v2
free(v3); // libera el espacio reservado para v3
#endif
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp1/ejer7 gcc -fopenmp -O2 sp-OpenMP-for.c -o sp-OpenMP-for
DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp1/ejer7 ./sp-OpenMP-for 8 ✓ | 11:29:53
Tamano Vectores:8 (4 B)
Tiempo:0.000867900 / Tamano Vectores:8
/ v1[6]+v2[6]=v3[6](1.400000+0.200000=1.600000) /
/ v1[4]+v2[4]=v3[4](1.200000+0.400000=1.600000) /
/ v1[0]+v2[0]=v3[0](0.800000+0.800000=1.600000) /
/ v1[7]+v2[7]=v3[7](1.500000+0.100000=1.600000) /
/ v1[1]+v2[1]=v3[1](0.900000+0.700000=1.600000) /
/ v1[3]+v2[3]=v3[3](1.100000+0.500000=1.600000) /
/ v1[2]+v2[2]=v3[2](1.000000+0.600000=1.600000) /
/ v1[5]+v2[5]=v3[5](1.300000+0.300000=1.600000) /
DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp1/ejer7 ./sp-OpenMP-for 11 ✓ | 11:29:58
Tamano Vectores:11 (4 B)
Tiempo:0.000012000 / Tamano 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) /
DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp1/ejer7 | ✓ | 11:30:23
```

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, v_3 , 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 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-sections.c`


```

51 // inicializamos los vectores
52 double tiempo_inicial, tiempo_final, tiempo_total;
53 #pragma omp parallel
54 {
55     #pragma omp sections
56     {
57         #pragma omp section
58         for(i=0; i<N/4; i++){
59             v1[i] = N*0.1+i*0.1;
60             v2[i] = N*0.1-i*0.1;
61         }
62
63         #pragma omp section
64         for(j=N/4; j<N/2; j++){
65             v1[j] = N*0.1+j*0.1;
66             v2[j] = N*0.1-j*0.1;
67         }
68
69         #pragma omp section
70         for(k=N/2; k<3*N/4; k++){
71             v1[k] = N*0.1+k*0.1;
72             v2[k] = N*0.1-k*0.1;
73         }
74
75         #pragma omp section
76         for(l=3*N/4; l<N; l++){
77             v1[l] = N*0.1+l*0.1;
78             v2[l] = N*0.1-l*0.1;
79         }
80     }
81
82     #pragma omp single
83     tiempo_inicial = omp_get_wtime();
84
85     #pragma omp sections
86     {
87         #pragma omp section
88         for(i=0; i<N/4; i++){
89             v3[i] = v1[i] + v2[i];
90         }
91
92         #pragma omp section
93         for(j=N/4; j<N/2; j++){
94             v3[j] = v1[j] + v2[j];
95         }
96
97         #pragma omp section
98         for(k=N/2; k<3*N/4; k++){
99             v3[k] = v1[k] + v2[k];
100         }
101
102         #pragma omp section
103         for(l=3*N/4; l<N; l++){
104             v3[l] = v1[l] + v2[l];
105         }
106     }
107
108     #pragma omp single
109     tiempo_final = omp_get_wtime();
110
111     tiempo_total = tiempo_final - tiempo_inicial;
112
113     //Imprimir resultado de la suma y el tiempo de ejecución
114     if (N<10) {
115         printf("Tiempo:%f\t / Tamaño Vectores:%u\n", tiempo_total, N);
116         for(i=0; i<N; i++){
117             printf("/ v1[%d]+v2[%d]=v3[%d](%.6f+%.6f=%.6f) /\n",
118                 i, i, v1[i], v2[i], v3[i]);
119         }
120     }
121     else
122         printf("Tiempo:%f\t / Tamaño Vectores:%u\t / v1[0]+v2[0]=v3[0](%.6f+%.6f=%.6f) / / v1[%d]+v2[%d]=v3[%d](%.6f+%.6f=%.6f) /\n",
123             tiempo_total, N, v1[0], v2[0], v3[0], N-1, N-1, v1[N-1], v2[N-1], v3[N-1]);
124 }

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer8 gcc -O2 -fopenmp
sp-OpenMP-sections.c -o sp-OpenMP-sections
DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer8 ./sp-OpenMP-sections 8
Tiempo:0.000003900 / 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) /
DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer8 ./sp-OpenMP-sections 11
Tiempo:0.000051500 / 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) /
DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer8

```

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuántos 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, vamos a poder utilizar todos los cores que tenga mi pc, ya que en el código no se ha implementado la función “omp-set-num-threads”, que establece el número de threads que vamos a utilizar en el programa.

Sin embargo en el ejercicio 8, como cada bucle for lo ejecuta una sola hebra, su grado de paralelización ya no es el mismo, por lo que no se llegará a utilizar el máximo creo yo, que en este caso, serían 4 cores físicos, uno para cada bucle for.

10. Rellenar una tabla como la Tabla 210 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

#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vlocales
#PBS -N SumaVectoresC_vlocales
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"

echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
# FIN del trozo que deben incluir todos los scripts
for (( i = 16384; i <= 67108864; i*=2 )); do
    ./$1 $i
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

```
[e2estudiante14@atcgrid ejer10]$ sbatch -p ac ./sp-OpenMP-script10.sh
Submitted batch job 78338
[e2estudiante14@atcgrid ejer10]$ cat slurm-78338.out
Tamaño Vectores:16384 (4 B)
Tiempo:0.000432747 / 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=3276.8
00000) /
Tamaño Vectores:32768 (4 B)
Tiempo:0.000881658 / 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=6553.6
00000) /
Tamaño Vectores:65536 (4 B)
Tiempo:0.000488072 / Tamaño Vectores:65536 / V1[0]+V2[0]=V3[0](6553.600000
+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535](13107.100000+0.100000=1310
7.200000) /
Tamaño Vectores:131072 (4 B)
Tiempo:0.000603504 / Tamaño Vectores:131072 / V1[0]+V2[0]=V3[0](13107.20000
0+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[131071](26214.300000+0.100000
=26214.400000) /
```

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.

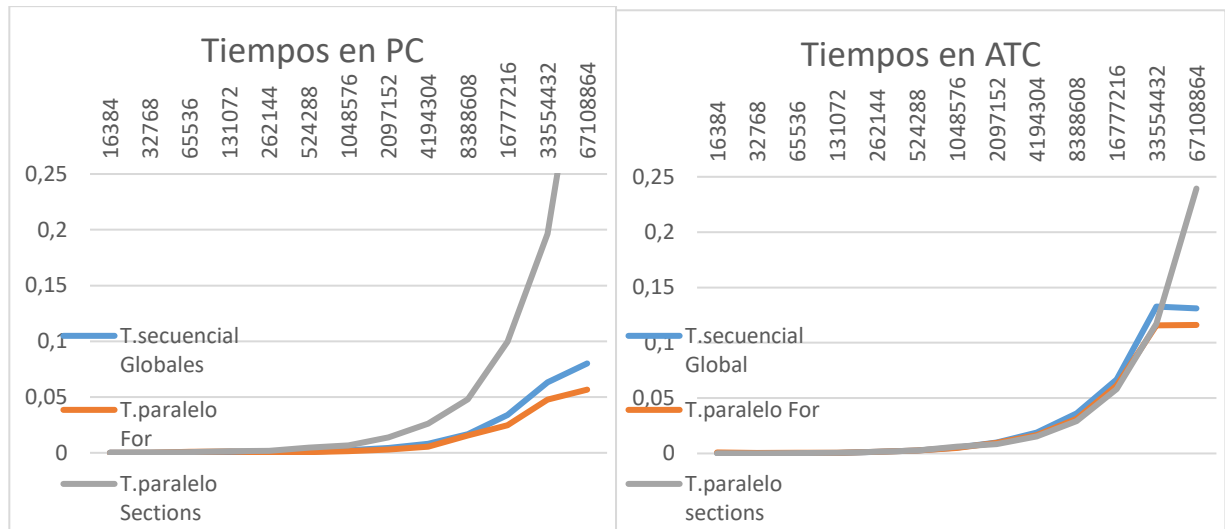
Para PC:

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) 6 threads = cores lógicos = cores físicos	T. paralelo (versión sections) 6 threads = cores lógicos = cores físicos
16384	0.000097600	0.000031400	0.000135800
32768	0.000140400	0.000055600	0.000320000
65536	0.000132900	0.000879900	0.000524700
131072	0.000283000	0.001213900	0.001614100
262144	0.000495600	0.000312000	0.001664800
524288	0.000967200	0.000600500	0.004641700
1048576	0.002228500	0.001345800	0.006590600
2097152	0.004278200	0.003030000	0.013758000
4194304	0.008132800	0.005573200	0.026234300
8388608	0.016687300	0.015657900	0.048014300
16777216	0.033854200	0.024733300	0.099617300
33554432	0.063229200	0.047648600	0.196372300
67108864	0.080136900	0.056642900	0.413890500

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.

Para ATC:

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) 12 threads = cores lógicos = cores físicos	T. paralelo (versión sections) 12 threads = cores lógicos = cores físicos
16384	0.000430682	0.000893410	0.000103626
32768	0.000418737	0.000469036	0.000179239
65536	0.000391069	0.000509299	0.000409625
131072	0.000443401	0.000736352	0.000749394
262144	0.001443790	0.001429945	0.001445636
524288	0.002761257	0.002568111	0.002555914
1048576	0.005237209	0.005067702	0.006035995
2097152	0.009870978	0.009554561	0.008564293
4194304	0.018729252	0.016653799	0.015439760
8388608	0.036239037	0.031263638	0.029247530
16777216	0.066939411	0.061516255	0.058253612
33554432	0.132744833	0.115652569	0.117332138
67108864	0.131187172	0.116130535	0.239396375



11. Rellenar una tabla como la 12Tabla 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

#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vlocales
#PBS -N SumaVectoresC_vlocales
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"

echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
# FIN del trozo que deben incluir todos los scripts
for (( i = 8388608; i <= 67108864 ; i*=2 )); do
    time ./SumaVectoresC $i
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

```

[estudiante@l4atgrid ejor11]$ sbatch -p ac --wrap "time ./sp-OpenMP-script11.sh"
Submitted batch job 78371
[estudiante@l4atgrid ejor11]$ cat slurm-78371.out
Id. usuario del trabajo:
Id. del trabajo:
Nombre del trabajo especificado por usuario:
Modo que ejecuta qsub:
Directorio en el que se ha ejecutado qsub:
Cola:
Modos asignados al trabajo:
Tamaño Vectores:8388608 (4 B)
Tiempo:0.835216782 / Tamaño Vectores:8388608 / Vi[0]-V2[0]-Vi[0](838860.800000+838860.800000-1677721.600000) // Vi[8388607]-V2[8388607]-Vi[8388607](1677721.500000+0.100000-1677721.600000) /
/
real 0m0.112s
user 0m0.099s
sys 0m0.063s
Tamaño Vectores:16777216 (4 B)
Tiempo:0.862159889 / Tamaño Vectores:16777216 / Vi[0]-V2[0]-Vi[0](1677721.600000+1677721.600000-3355443.200000) // Vi[16777215]-V2[16777215]-Vi[16777215](3355443.100000+0.100000-3355443.200000) /
/
real 0m0.165s
user 0m0.174s
sys 0m0.111s
Tamaño Vectores:33554432 (4 B)
Tiempo:0.121398861 / Tamaño Vectores:33554432 / Vi[0]-V2[0]-Vi[0](3355443.200000+3355443.200000-6710886.400000) // Vi[33554431]-V2[33554431]-Vi[33554431](6710886.300000+0.100000-6710886.400000) /
/
real 0m0.293s
user 0m0.226s
sys 0m0.222s
Tamaño Vectores:67108864 (4 B)
Tiempo:0.131620217 / Tamaño Vectores:67108864 / Vi[0]-V2[0]-Vi[0](6710886.400000+6710886.400000-13421772.800000) // Vi[67108863]-V2[67108863]-Vi[67108863](13421772.800000+0.100000-13421772.900000) /
/
real 0m0.296s
user 0m0.219s
sys 0m0.226s
Tamaño Vectores:134217728 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:134217728 / Vi[0]-V2[0]-Vi[0](13421772.900000+13421772.900000-26843545.600000) // Vi[134217727]-V2[134217727]-Vi[134217727](26843545.600000+0.100000-26843545.700000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:268435456 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:268435456 / Vi[0]-V2[0]-Vi[0](26843545.700000+26843545.700000-53687091.200000) // Vi[268435455]-V2[268435455]-Vi[268435455](53687091.200000+0.100000-53687091.300000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:536870912 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:536870912 / Vi[0]-V2[0]-Vi[0](53687091.300000+53687091.300000-107374182.400000) // Vi[536870911]-V2[536870911]-Vi[536870911](107374182.400000+0.100000-107374182.500000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:1073741824 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:1073741824 / Vi[0]-V2[0]-Vi[0](107374182.500000+107374182.500000-214748364.800000) // Vi[1073741823]-V2[1073741823]-Vi[1073741823](214748364.800000+0.100000-214748364.900000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:2147483648 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:2147483648 / Vi[0]-V2[0]-Vi[0](214748364.900000+214748364.900000-4294967296.000000) // Vi[2147483647]-V2[2147483647]-Vi[2147483647](4294967296.000000+0.100000-4294967296.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:4294967296 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:4294967296 / Vi[0]-V2[0]-Vi[0](4294967296.100000+4294967296.100000-8589934592.000000) // Vi[4294967295]-V2[4294967295]-Vi[4294967295](8589934592.000000+0.100000-8589934592.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:8589934592 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:8589934592 / Vi[0]-V2[0]-Vi[0](8589934592.100000+8589934592.100000-17179869184.000000) // Vi[8589934591]-V2[8589934591]-Vi[8589934591](17179869184.000000+0.100000-17179869184.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:17179869184 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:17179869184 / Vi[0]-V2[0]-Vi[0](17179869184.100000+17179869184.100000-34359738368.000000) // Vi[17179869183]-V2[17179869183]-Vi[17179869183](34359738368.000000+0.100000-34359738368.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:34359738368 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:34359738368 / Vi[0]-V2[0]-Vi[0](34359738368.100000+34359738368.100000-68719476736.000000) // Vi[34359738367]-V2[34359738367]-Vi[34359738367](68719476736.000000+0.100000-68719476736.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:68719476736 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:68719476736 / Vi[0]-V2[0]-Vi[0](68719476736.100000+68719476736.100000-137438953472.000000) // Vi[68719476735]-V2[68719476735]-Vi[68719476735](137438953472.000000+0.100000-137438953472.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:137438953472 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:137438953472 / Vi[0]-V2[0]-Vi[0](137438953472.100000+137438953472.100000-274877906944.000000) // Vi[137438953471]-V2[137438953471]-Vi[137438953471](274877906944.000000+0.100000-274877906944.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:274877906944 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:274877906944 / Vi[0]-V2[0]-Vi[0](274877906944.100000+274877906944.100000-549755813888.000000) // Vi[274877906943]-V2[274877906943]-Vi[274877906943](549755813888.000000+0.100000-549755813888.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:549755813888 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:549755813888 / Vi[0]-V2[0]-Vi[0](549755813888.100000+549755813888.100000-1099511627776.000000) // Vi[549755813887]-V2[549755813887]-Vi[549755813887](1099511627776.000000+0.100000-1099511627776.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:1099511627776 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:1099511627776 / Vi[0]-V2[0]-Vi[0](1099511627776.100000+1099511627776.100000-2199023255552.000000) // Vi[1099511627775]-V2[1099511627775]-Vi[1099511627775](2199023255552.000000+0.100000-2199023255552.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:2199023255552 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:2199023255552 / Vi[0]-V2[0]-Vi[0](2199023255552.100000+2199023255552.100000-4398046511104.000000) // Vi[2199023255551]-V2[2199023255551]-Vi[2199023255551](4398046511104.000000+0.100000-4398046511104.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:4398046511104 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:4398046511104 / Vi[0]-V2[0]-Vi[0](4398046511104.100000+4398046511104.100000-8796093022208.000000) // Vi[4398046511103]-V2[4398046511103]-Vi[4398046511103](8796093022208.000000+0.100000-8796093022208.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:8796093022208 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:8796093022208 / Vi[0]-V2[0]-Vi[0](8796093022208.100000+8796093022208.100000-17592186044416.000000) // Vi[8796093022207]-V2[8796093022207]-Vi[8796093022207](17592186044416.000000+0.100000-17592186044416.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:17592186044416 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:17592186044416 / Vi[0]-V2[0]-Vi[0](17592186044416.100000+17592186044416.100000-35184372088832.000000) // Vi[17592186044415]-V2[17592186044415]-Vi[17592186044415](35184372088832.000000+0.100000-35184372088832.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:35184372088832 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:35184372088832 / Vi[0]-V2[0]-Vi[0](35184372088832.100000+35184372088832.100000-70368744177664.000000) // Vi[35184372088831]-V2[35184372088831]-Vi[35184372088831](70368744177664.000000+0.100000-70368744177664.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:70368744177664 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:70368744177664 / Vi[0]-V2[0]-Vi[0](70368744177664.100000+70368744177664.100000-140737488355328.000000) // Vi[70368744177663]-V2[70368744177663]-Vi[70368744177663](140737488355328.000000+0.100000-140737488355328.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:140737488355328 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:140737488355328 / Vi[0]-V2[0]-Vi[0](140737488355328.100000+140737488355328.100000-281474976710656.000000) // Vi[140737488355327]-V2[140737488355327]-Vi[140737488355327](281474976710656.000000+0.100000-281474976710656.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:281474976710656 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:281474976710656 / Vi[0]-V2[0]-Vi[0](281474976710656.100000+281474976710656.100000-562949953421312.000000) // Vi[281474976710655]-V2[281474976710655]-Vi[281474976710655](562949953421312.000000+0.100000-562949953421312.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:562949953421312 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:562949953421312 / Vi[0]-V2[0]-Vi[0](562949953421312.100000+562949953421312.100000-1125899906842624.000000) // Vi[562949953421311]-V2[562949953421311]-Vi[562949953421311](1125899906842624.000000+0.100000-1125899906842624.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:1125899906842624 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:1125899906842624 / Vi[0]-V2[0]-Vi[0](1125899906842624.100000+1125899906842624.100000-2251799813685248.000000) // Vi[1125899906842623]-V2[1125899906842623]-Vi[1125899906842623](2251799813685248.000000+0.100000-2251799813685248.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:2251799813685248 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:2251799813685248 / Vi[0]-V2[0]-Vi[0](2251799813685248.100000+2251799813685248.100000-4503599627370496.000000) // Vi[2251799813685247]-V2[2251799813685247]-Vi[2251799813685247](4503599627370496.000000+0.100000-4503599627370496.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:4503599627370496 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:4503599627370496 / Vi[0]-V2[0]-Vi[0](4503599627370496.100000+4503599627370496.100000-9007199254740992.000000) // Vi[4503599627370495]-V2[4503599627370495]-Vi[4503599627370495](9007199254740992.000000+0.100000-9007199254740992.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:9007199254740992 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:9007199254740992 / Vi[0]-V2[0]-Vi[0](9007199254740992.100000+9007199254740992.100000-18014398509481984.000000) // Vi[9007199254740991]-V2[9007199254740991]-Vi[9007199254740991](18014398509481984.000000+0.100000-18014398509481984.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:18014398509481984 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:18014398509481984 / Vi[0]-V2[0]-Vi[0](18014398509481984.100000+18014398509481984.100000-36028797018963968.000000) // Vi[18014398509481983]-V2[18014398509481983]-Vi[18014398509481983](36028797018963968.000000+0.100000-36028797018963968.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:36028797018963968 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:36028797018963968 / Vi[0]-V2[0]-Vi[0](36028797018963968.100000+36028797018963968.100000-72057594037927936.000000) // Vi[36028797018963967]-V2[36028797018963967]-Vi[36028797018963967](72057594037927936.000000+0.100000-72057594037927936.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:72057594037927936 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:72057594037927936 / Vi[0]-V2[0]-Vi[0](72057594037927936.100000+72057594037927936.100000-144115188075855872.000000) // Vi[72057594037927935]-V2[72057594037927935]-Vi[72057594037927935](144115188075855872.000000+0.100000-144115188075855872.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:144115188075855872 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:144115188075855872 / Vi[0]-V2[0]-Vi[0](144115188075855872.100000+144115188075855872.100000-288230376151711744.000000) // Vi[144115188075855871]-V2[144115188075855871]-Vi[144115188075855871](288230376151711744.000000+0.100000-288230376151711744.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:288230376151711744 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:288230376151711744 / Vi[0]-V2[0]-Vi[0](288230376151711744.100000+288230376151711744.100000-576460752303423488.000000) // Vi[288230376151711743]-V2[288230376151711743]-Vi[288230376151711743](576460752303423488.000000+0.100000-576460752303423488.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:576460752303423488 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:576460752303423488 / Vi[0]-V2[0]-Vi[0](576460752303423488.100000+576460752303423488.100000-1152921504606846976.000000) // Vi[576460752303423487]-V2[576460752303423487]-Vi[576460752303423487](1152921504606846976.000000+0.100000-1152921504606846976.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:1152921504606846976 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:1152921504606846976 / Vi[0]-V2[0]-Vi[0](1152921504606846976.100000+1152921504606846976.100000-2305843009213693952.000000) // Vi[1152921504606846975]-V2[1152921504606846975]-Vi[1152921504606846975](2305843009213693952.000000+0.100000-2305843009213693952.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:2305843009213693952 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:2305843009213693952 / Vi[0]-V2[0]-Vi[0](2305843009213693952.100000+2305843009213693952.100000-4611686018427387904.000000) // Vi[2305843009213693951]-V2[2305843009213693951]-Vi[2305843009213693951](4611686018427387904.000000+0.100000-4611686018427387904.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:4611686018427387904 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:4611686018427387904 / Vi[0]-V2[0]-Vi[0](4611686018427387904.100000+4611686018427387904.100000-9223372036854775808.000000) // Vi[4611686018427387903]-V2[4611686018427387903]-Vi[4611686018427387903](9223372036854775808.000000+0.100000-9223372036854775808.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:9223372036854775808 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:9223372036854775808 / Vi[0]-V2[0]-Vi[0](9223372036854775808.100000+9223372036854775808.100000-18446744073709551616.000000) // Vi[9223372036854775807]-V2[9223372036854775807]-Vi[9223372036854775807](18446744073709551616.000000+0.100000-18446744073709551616.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:18446744073709551616 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:18446744073709551616 / Vi[0]-V2[0]-Vi[0](18446744073709551616.100000+18446744073709551616.100000-36893488147419103232.000000) // Vi[18446744073709551615]-V2[18446744073709551615]-Vi[18446744073709551615](36893488147419103232.000000+0.100000-36893488147419103232.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:36893488147419103232 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:36893488147419103232 / Vi[0]-V2[0]-Vi[0](36893488147419103232.100000+36893488147419103232.100000-73786976294838206464.000000) // Vi[36893488147419103231]-V2[36893488147419103231]-Vi[36893488147419103231](73786976294838206464.000000+0.100000-73786976294838206464.100000) /
/
real 0m0.330s
user 0m0.169s
sys 0m0.141s
Tamaño Vectores:73786976294838206464 (4 B)
Tiempo:0.141620217 / Tamaño Vectores:73786976294838206464 / Vi[0]-V2[0]-Vi[0](73786976294838206464.100000+73786976294838206464.100000-147573952589676412928.000000) // Vi[73786976294838206463]-V2[73786976294838206463]-Vi[73786976294838206463](147573952589676412928.000
```