

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): Elena Cantero Molina

Grupo de prácticas: 2

Fecha de entrega:

Fecha evaluación en clase:

[-RECORDATORIO, quitar todo este texto en rojo del cuaderno definitivo-

1. COMENTARIOS

- 1) Esta plantilla no sustituye al guion de prácticas, se ha preparado para ahorrarles trabajo. Las preguntas de esta plantilla se han extraído del **guion** de prácticas de programación paralela, si tiene dudas sobre el enunciado consulte primero el **guion**.
- 2) Este cuaderno de prácticas se utilizará para asignarle una puntuación durante la evaluación continua de prácticas y también lo utilizará como material de estudio y repaso para preparar el examen de prácticas escrito. Luego redáctelo con cuidado, y sea ordenado y claro.
- 3) No use máquinas virtuales.

2. NORMAS SOBRE EL USO DE LA PLANTILLA

- 1) Usar **interlineado SENCILLO**.
 - 2) Respetar los tipos de letra y tamaños indicados:
 - Calibri-11 o Liberation Serif-11 para el texto
 - **Courier New-10 o Liberation Mono-10 para nombres de fichero, comandos, variables de entorno, etc., cuando se usan en el texto.**
 - Courier New o Liberation Mono de tamaño 8 o 9 para el código fuente en los listados de código fuente.
 - Formatee el código fuente de los listados para que sea legible, limpio y claro. Consulte, como ejemplo, los Listados 1 y 2 del guion (tabule, comente, ...)
 - 3) Insertar las capturas de pantalla donde se pidan y donde se considere oportuno
- Recuerde que debe adjuntar al zip de entrega, el pdf de este fichero, todos los ficheros con código fuente implementados/utilizados y el resto de ficheros que haya implementado/utilizado (scripts, hojas de cálculo, etc.), lea la Sección 1.4 del guion]**

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: código fuente `bucle-forModificado.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/  
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/  
/* INTERLINEADO SENCILLO */  
  
#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: código fuente sectionsModificado.c

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    #pragma omp parallel sections
    {
        #pragma omp section
        (void) funcA();
        #pragma omp section
        (void) funcB();
    }
}

```

- . 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: código fuente singleModificado.c

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    int n = 9, i, a, b[n];

    for (i=0; i<n; i++) b[i] = -1;
    #pragma omp parallel

```

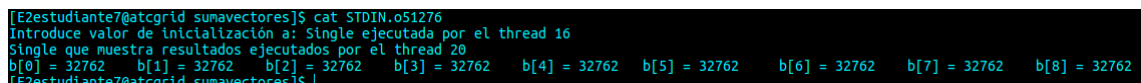
```

{
    #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
    {
        printf("Single que muestra resultados ejecutados por el thread %d\n",
            omp_get_thread_num());
        for (i=0; i<n; i++) printf("b[%d] = %d\t", i, b[i]);
        printf("\n");
    }
}
}

```

CAPTURA DE PANTALLA:


```

[E2estudiante7@atcgird sumavectores]$ cat STDIN.051276
Introduce valor de inicialización a: Single ejecutada por el thread 16
Single que muestra resultados ejecutados por el thread 20
b[0] = 32762   b[1] = 32762   b[2] = 32762   b[3] = 32762   b[4] = 32762   b[5] = 32762   b[6] = 32762   b[7] = 32762   b[8] = 32762
[E2estudiante7@atcgird sumavectores]$

```

4. 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: código fuente `singleModificado2.c`

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

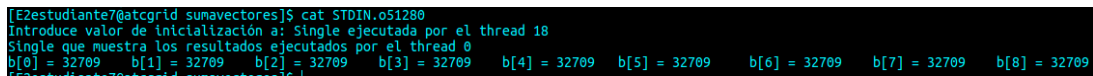
int main(int argc, char ** argv)
{
    int n = 9, i, a, b[n];

    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
{
    printf("Single que muestra los resultados ejecutados por el thread
%d\n", omp_get_thread_num());
    for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
    printf("\n");
}
}
```

CAPTURA DE PANTALLA:


```
[E2estudiante7@atcgird sumavectores]$ cat STDIN.051280
Introduce valor de inicialización a: Single ejecutada por el thread 18
Single que muestra los resultados ejecutados por el thread 0
b[0] = 32769    b[1] = 32769    b[2] = 32769    b[3] = 32769    b[4] = 32769    b[5] = 32769    b[6] = 32769    b[7] = 32769    b[8] = 32769
```

RESPUESTA A LA PREGUNTA:

Al utilizar la directiva master los resultados siempre los pinta el thread 0.

- . ¿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:

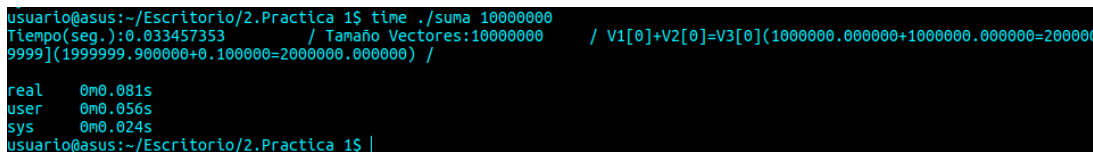
Cuando las hebras actualizan la suma no se esperan entre ellas, por lo que no hay sincronización, en particular, la hebra cero, que hace de master, no espera a las demás hebras por lo que no imprime el último valor de la suma.

Resto de ejercicios

- . 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 el PC local, 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 mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA:

La suma de los tiempos de usuario y sistema es igual al tiempo real porque se está usando sólo un núcleo del procesador.



```
usuario@asus:~/Escritorio/2.Practica 1$ time ./suma 10000000
Tiempo(seg.):0.033457353 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) /
99999(1999999.900000+0.100000=2000000.000000) /
real    0m0.081s
user    0m0.056s
sys     0m0.024s
usuario@asus:~/Escritorio/2.Practica 1$ |
```

- . 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 atcgird 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). Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

CAPTURAS DE PANTALLA:

```
[E2estudiante7@atcgri sumavectores]$ echo 'time sumavectores/sumavectores_globales 10' | qsub -q ac
51343.atcgri
[E2estudiante7@atcgri sumavectores]$ echo 'time sumavectores/sumavectores_globales 1000000' | qsub -q ac
51344.atcgri
[E2estudiante7@atcgri sumavectores]$ qstat
Job ID          Name                    User                    Time Use S Queue
-----
51344.atcgri    STDIN                    E2estudiante7         00:00:00 C ac
[E2estudiante7@atcgri sumavectores]$ cat STDIN.o51343
Tiempo(seg.):0.000002892 / 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) /
[E2estudiante7@atcgri sumavectores]$ cat STDIN.o51344
Tiempo(seg.):0.004834670 / Tamaño Vectores:1000000 / V1[0]+V2[0]=V3[0](100000.000000+100000.000000=200000.000000) / V1[999999]+V2[999999]=V3[999999](
199999.900000+0.100000=200000.000000) /
[E2estudiante7@atcgri sumavectores]$ cat STDIN.e51344
real    0m0.020s
user    0m0.004s
sys     0m0.015s
[E2estudiante7@atcgri sumavectores]$ cat STDIN.e51343
real    0m0.005s
user    0m0.000s
sys     0m0.001s
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

RESPUESTA:

| | | |
|-------------------|---|---|
| Tamaño vector (N) | 10 | 10000000 |
| Tiempo | 0.000002892 | 0.004834670 |
| NI | 60 | 60000000 |
| FPO | 10 | 10000000 |
| MIPS | $60/(0.000002892 \cdot 10^6) = 20.7469$ | $60000000/(0.004834670 \cdot 10^6) = 12410.361$ |
| MFLOPS | $10/(0.000002892 \cdot 10^6) = 3.4578$ | $10000000/(0.004834670 \cdot 10^6) = 2068.3935$ |

código ensamblador generado de la parte de la suma de vectores

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

.L5:
    movsd    v1(%rax), %xmm0
    addq     $8, %rax
    addsd    v2-8(%rax), %xmm0
    movsd    %xmm0, v3-8(%rax)
    cmpq     %rax, %rbx
    jne      .L5
.L6:
    leaq     16(%rsp), %rsi
    xorl     %edi, %edi
    call     clock_gettime
    movq     24(%rsp), %rax
    subq     8(%rsp), %rax
    movl     %r12d, %edx
    pxor     %xmm0, %xmm0
    movl     %r12d, %r8d
    pxor     %xmm1, %xmm1
    movl     %r12d, %ecx
    movsd    v3(,%rdx,8), %xmm6
    movl     %ebp, %esi
```

```

    cvtsi2sdq    %rax, %xmm0
    movq        16(%rsp), %rax
    subq        (%rsp), %rax
    movsd       v2(%rdx,8), %xmm5
    movsd       v1(%rdx,8), %xmm4
    movl        $.LC3, %edi
    movsd       v3(%rip), %xmm3
    movl        %r12d, %edx
    movsd       v2(%rip), %xmm2
    cvtsi2sdq    %rax, %xmm1
    movl        $7, %eax
    divsd       .LC2(%rip), %xmm0
    addsd       %xmm1, %xmm0
    movsd       v1(%rip), %xmm1
    call        printf
    addq        $32, %rsp
    .cfi_remember_state
    .cfi_def_cfa_offset 32
    xorl        %eax, %eax
    popq        %rbx
    .cfi_def_cfa_offset 24
    popq        %rbp
    .cfi_def_cfa_offset 16
    popq        %r12
    .cfi_def_cfa_offset 8
    ret

```

- . 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: código fuente implementado

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

#define MAX 1000000
double v1[MAX], v2[MAX], v3[MAX];

int main(int argc, char** argv){

```

```

int i;
double cgt1,cgt2;
double ncgt; //para tiempo de ejecución

if (argc<2){
    printf("Faltan nº componentes del vector\n");
    exit(-1);
}

unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

if(N > MAX) N = MAX;

//Inicializar vectores
#pragma omp parallel
{
    #pragma omp for
    for(i=0; i<N; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
    }
}

cgt1 = omp_get_wtime();

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

cgt2 = omp_get_wtime();

ncgt = cgt2-cgt1;

//Imprimir resultado de la suma y el tiempo de ejecución
if(N < 15){
    for(i=0; i<N; i++)
        printf("v3[%i]=%f \n",i,v3[i]);
}
printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0]
(%8.6f+%8.6f=%8.6f) / V1[%d]+V2[%d]=V3[%d] (%8.6f+%8.6f=%8.6f) /\n",
ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

return 0;
}

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)**CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):**

```

[E2estudiante7@atcgird sumavectores]$ gcc -fopenmp -O2 -o SUMA Sumavectores_for.c
[E2estudiante7@atcgird sumavectores]$ echo "sumavectores/SUMA 8" | qsub -q ac
52002.atcgird
[E2estudiante7@atcgird sumavectores]$ echo "sumavectores/SUMA 11" | qsub -q ac
52003.atcgird
[E2estudiante7@atcgird sumavectores]$ cat *.o52002
v3[0]=1.600000
v3[1]=1.600000
v3[2]=1.600000
v3[3]=1.600000
v3[4]=1.600000
v3[5]=1.600000
v3[6]=1.600000
v3[7]=1.600000
Tiempo(seg.):0.004191872 / Tamaño Vectores:8 / V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) / V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[E2estudiante7@atcgird sumavectores]$ cat *.o52003
v3[0]=2.200000
v3[1]=2.200000
v3[2]=2.200000
v3[3]=2.200000
v3[4]=2.200000
v3[5]=2.200000
v3[6]=2.200000
v3[7]=2.200000
v3[8]=2.200000
v3[9]=2.200000
v3[10]=2.200000
Tiempo(seg.):0.003908359 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](1.100000+0.100000=2.200000) / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /

```

- . 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 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: código fuente implementado

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

#define MAX 1000000
double v1[MAX], v2[MAX], v3[MAX];

int main(int argc, char** argv){

    int i;
    double cgt1,cgt2;
    double ncgt; //para tiempo de ejecución

    if (argc<2){
        printf("Faltan nº componentes del vector\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
    (sizeof(unsigned int) = 4 B)

```



```

if(N > MAX) N = MAX;

//Inicializar vectores
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        for(i=0; i<N/4; i++){
            v1[i] = N*0.1+i*0.1;
            v2[i] = N*0.1-i*0.1; //los valores dependen de N
        }
        #pragma omp section
        for(i=N/4; i<N/2; i++){
            v1[i] = N*0.1+i*0.1;
            v2[i] = N*0.1-i*0.1; //los valores dependen de N
        }
        #pragma omp section
        for(i=N/2; i<3*N/4; i++){
            v1[i] = N*0.1+i*0.1;
            v2[i] = N*0.1-i*0.1; //los valores dependen de N
        }
        #pragma omp section
        for(i=3*N/4; i<N; i++){
            v1[i] = N*0.1+i*0.1;
            v2[i] = N*0.1-i*0.1; //los valores dependen de N
        }
    }
}

cgt1 = omp_get_wtime();

//Calcular suma de vectores
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        for(i=0; i<N/4; i++)
            v3[i] = v1[i] + v2[i];

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

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

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

```

```

cgt2 = omp_get_wtime();

ncgt = cgt2-cgt1;

//Imprimir resultado de la suma y el tiempo de ejecución
if(N < 15){
    for(i=0; i<N; i++){
        printf("v3[%i]=%f \n",i,v3[i]);
    }
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0]
(%8.6f+%8.6f=%8.6f) / V1[%d]+V2[%d]=V3[%d] (%8.6f+%8.6f=%8.6f) /\n",
ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

    return 0;
}

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

[Ezestudiante7@atcgrid sumavectores]$ gcc -fopenmp -O2 -o SUMA Sumavectores_sections.c
[Ezestudiante7@atcgrid sumavectores]$ echo "sumavectores/SUMA 8" | qsub -q ac
52006.atcgrid
[Ezestudiante7@atcgrid sumavectores]$ echo "sumavectores/SUMA 11" | qsub -q ac
52007.atcgrid
[Ezestudiante7@atcgrid sumavectores]$ cat *.o52006
v3[0]=1.600000
v3[1]=1.600000
v3[2]=1.600000
v3[3]=1.600000
v3[4]=1.600000
v3[5]=1.600000
v3[6]=1.600000
v3[7]=1.600000
Tiempo(seg.):0.006144002 / Tamaño Vectores:8 / V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) / V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[Ezestudiante7@atcgrid sumavectores]$ cat *.o52007
v3[0]=2.200000
v3[1]=2.200000
v3[2]=2.200000
v3[3]=2.200000
v3[4]=2.200000
v3[5]=2.200000
v3[6]=2.200000
v3[7]=2.200000
v3[8]=2.200000
v3[9]=2.200000
v3[10]=2.200000
Tiempo(seg.):0.004406251 / 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) /

```

- . ¿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.

RESPUESTA: Como no hemos definido la variable `OMP_NUM_THREADS` en ninguno de los dos códigos, se usarán todos los cores/threads que tenga disponible la máquina usada. Sin embargo, en el anterior ejercicio como mucho se ejecutarán 4 hebras..

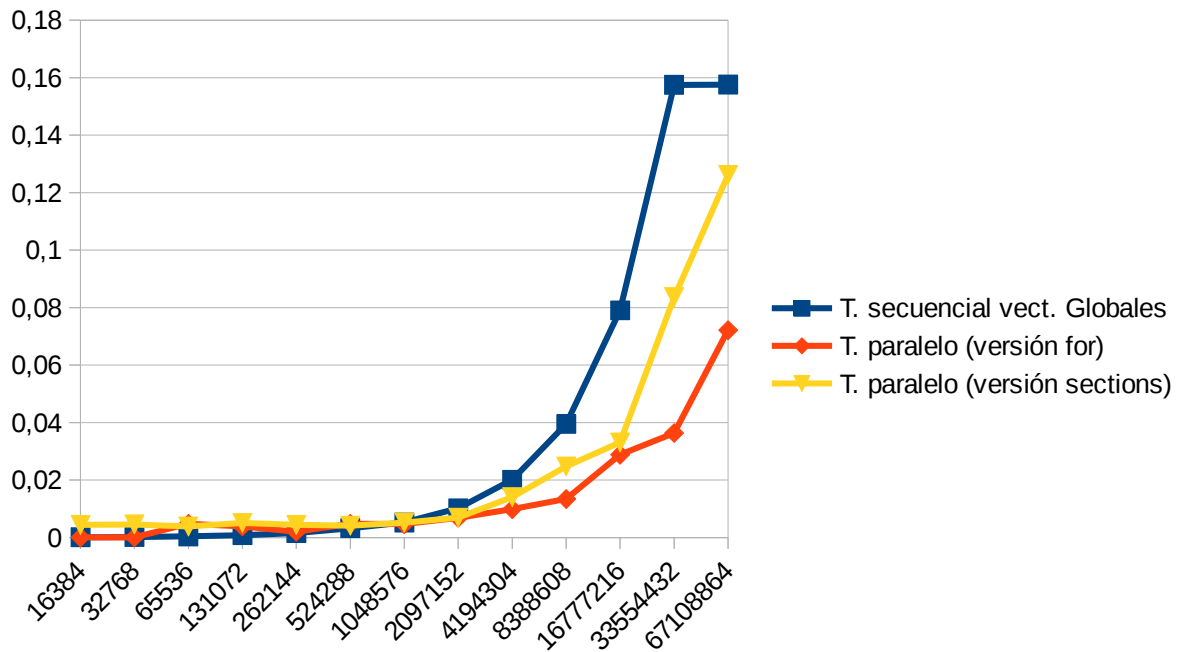
- . Rellenar una tabla como la Tabla 2 para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando `-O2`. En la tabla debe aparecer el tiempo de ejecución del 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. Represente en una gráfica los tres tiempos. **NOTA:** Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

RESPUESTA:

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 utilizados.

ATCGRID

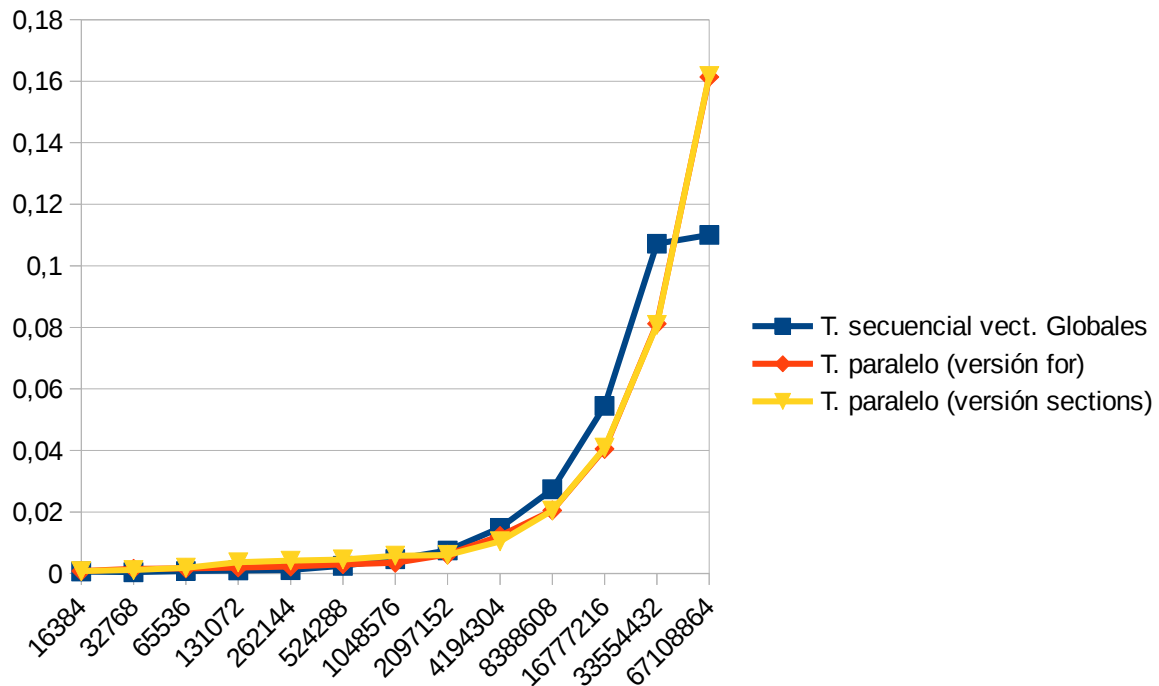
| Nº de Componentes | T. secuencial vect. Globales 1 thread/core | T. paralelo (versión for) 24 threads/ 12 cores | T. paralelo (versión sections) 24 threads/ 12 cores |
|-------------------|---|---|--|
| 16384 | 0.000100908 | 0.000042405 | 0.004398519 |
| 32768 | 0.000195444 | 0.000063788 | 0.004549379 |
| 65536 | 0.000415998 | 0.004803067 | 0.003841128 |
| 131072 | 0.000790579 | 0.003907705 | 0.005085746 |
| 262144 | 0.001572509 | 0.002002141 | 0.004402053 |
| 524288 | 0.003211480 | 0.004817808 | 0.004138734 |
| 1048576 | 0.005240308 | 0.004683742 | 0.005219677 |
| 2097152 | 0.010197316 | 0.006827010 | 0.006933197 |
| 4194304 | 0.020113659 | 0.009871710 | 0.014058851 |
| 8388608 | 0.039500037 | 0.013402974 | 0.024763916 |
| 16777216 | 0.079046547 | 0.028823564 | 0.033075666 |
| 33554432 | 0.157492056 | 0.036349228 | 0.083601041 |
| 67108864 | 0.157591808 | 0.072151635 | 0.126195299 |



PC LOCAL

| Nº de Componentes | T. secuencial vect. Globales 1 thread/core | T. paralelo (versión for) 4 threads/cores | T. paralelo (versión sections) 4 threads/cores |
|-------------------|---|--|---|
| 16384 | 0.000662893 | 0.000763443 | 0.00076776 |
| 32768 | 0.000428588 | 0.001505181 | 0.001135505 |
| 65536 | 0.000798973 | 0.001648852 | 0.001872409 |
| 131072 | 0.001013074 | 0.001954453 | 0.003644727 |
| 262144 | 0.001145456 | 0.002257646 | 0.004175213 |
| 524288 | 0.002526714 | 0.002980489 | 0.00455717 |
| 1048576 | 0.004654744 | 0.003483928 | 0.005783547 |
| 2097152 | 0.007470967 | 0.006145993 | 0.006027816 |

| | | | |
|-----------------|-------------|-------------|-------------|
| 4194304 | 0.014822318 | 0.012252084 | 0.010439391 |
| 8388608 | 0.027362306 | 0.020508792 | 0.020445009 |
| 16777216 | 0.05446513 | 0.040488235 | 0.040901917 |
| 33554432 | 0.107199635 | 0.081184456 | 0.080813464 |
| 67108864 | 0.110037222 | 0.161410645 | 0.161730871 |



00. Rellenar una tabla como la Tabla 3 para el PC local 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 2 Threads/ 2 cores | | |
|-------------------|---|-----------------|-----------------|---|-----------------|-----------------|
| | <i>Elapsed</i> | <i>CPU-user</i> | <i>CPU- sys</i> | <i>Elapsed</i> | <i>CPU-user</i> | <i>CPU- sys</i> |
| 65536 | real 0m0.004s user 0m0.000s sys 0m0.004s | | | real 0m0.003s user 0m0.004s sys 0m0.000s | | |
| 131072 | real 0m0.006s user 0m0.000s sys 0m0.008s | | | real 0m0.005s user 0m0.008s sys 0m0.000s | | |
| 262144 | real 0m0.010s user 0m0.008s sys 0m0.000s | | | real 0m0.009s user 0m0.016s sys 0m0.004s | | |
| 524288 | real 0m0.013s user 0m0.008s | | | real 0m0.012s user 0m0.024s | | |

| | | | |
|-----------------|---------------|--|---------------|
| | sys 0m0.004s | | sys 0m0.008s |
| 1048576 | real 0m0.012s | | real 0m0.014s |
| | user 0m0.008s | | user 0m0.036s |
| | sys 0m0.000s | | sys 0m0.004s |
| 2097152 | real 0m0.022s | | real 0m0.019s |
| | user 0m0.008s | | user 0m0.048s |
| | sys 0m0.012s | | sys 0m0.012s |
| 4194304 | real 0m0.043s | | real 0m0.033s |
| | user 0m0.036s | | user 0m0.068s |
| | sys 0m0.004s | | sys 0m0.036s |
| 8388608 | real 0m0.071s | | real 0m0.056s |
| | user 0m0.052s | | user 0m0.120s |
| | sys 0m0.016s | | sys 0m0.044s |
| 16777216 | real 0m0.132s | | real 0m0.104s |
| | user 0m0.088s | | user 0m0.268s |
| | sys 0m0.040s | | sys 0m0.096s |
| 33554432 | real 0m0.251s | | real 0m0.194s |
| | user 0m0.196s | | user 0m0.472s |
| | sys 0m0.052s | | sys 0m0.252s |
| 67108864 | real 0m0.254s | | real 0m0.367s |
| | user 0m0.140s | | user 0m0.968s |
| | sys 0m0.112s | | sys 0m0.456s |