

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):

Grupo de prácticas:

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>

void funcA() {
    printf("En funcA: esta sección la ejecuta el thread
%d\n", omp_get_thread_num());
}

void funcB() {
    printf("En funcB: esta sección la ejecuta el thread
%d\n", omp_get_thread_num());
}

main() {
    #pragma omp parallel sections
    {

        #pragma omp section
        (void) funcA();
        #pragma omp section
        (void) funcB();

    }
}

```

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: 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>

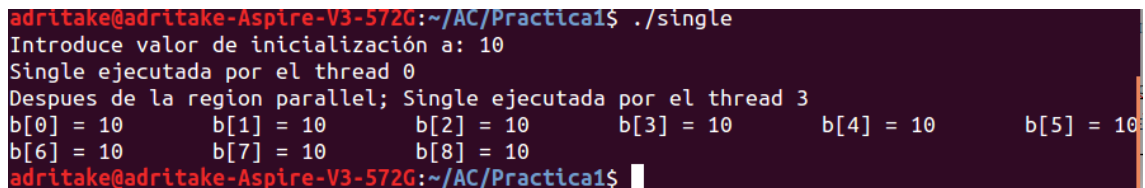
int main(int argc, char ** argv)
{
    int n = 9, i, a, b[n];

    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("Despues de la region parallel; Single ejecutada 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");
        }
    }
}
```

CAPTURAS DE PANTALLA:



```
adritake@adritake-Aspire-V3-572G:~/AC/Practica1$ ./single
Introduce valor de inicialización a: 10
Single ejecutada por el thread 0
Despues de la region parallel; Single ejecutada por el thread 3
b[0] = 10      b[1] = 10      b[2] = 10      b[3] = 10      b[4] = 10      b[5] = 10
b[6] = 10      b[7] = 10      b[8] = 10
adritake@adritake-Aspire-V3-572G:~/AC/Practica1$
```

- 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>

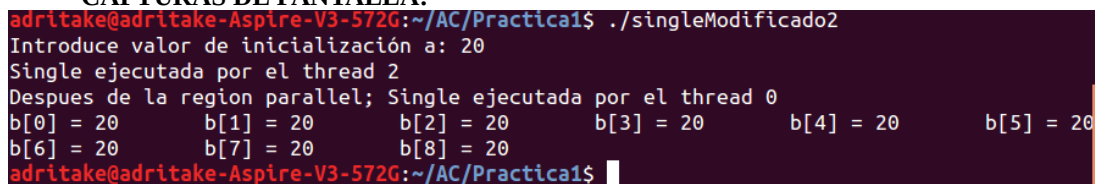
int main(int argc, char ** argv)
{
    int n = 9, i, a, b[n];

    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("Despues de la region parallel; Single ejecutada 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");
        }
    }
}
```

CAPTURAS DE PANTALLA:



```
adritake@adritake-Aspire-V3-572G:~/AC/Practica1$ ./singleModificado2
Introduce valor de inicialización a: 20
Single ejecutada por el thread 2
Despues de la region parallel; Single ejecutada por el thread 0
b[0] = 20      b[1] = 20      b[2] = 20      b[3] = 20      b[4] = 20      b[5] = 20
b[6] = 20      b[7] = 20      b[8] = 20
adritake@adritake-Aspire-V3-572G:~/AC/Practica1$
```

RESPUESTA A LA PREGUNTA: La diferencia es que el segundo single obligamos al programa que sea ejecutado por la hebra padre “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: Porque puede llegar primero la rama `master` a imprimir el resultado antes de que las demás hebras actualicen suma.

Resto de ejercicios

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 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: Es mayor o igual que el tiempo real porque se usan múltiples flujos de control para realizar la ejecución del programa, es decir, se reparte el trabajo y se suman los tiempos de ejecución de cada hebra, pero como se han realizado a la vez el tiempo real es menor.

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

CAPTURAS DE PANTALLA:

RESPUESTA: cálculo de los MIPS y los MFLOPS. Ejecutado en el `atcgrid`

MIPS: $(9 * 10000000) / (0.047227779 * 10^6) = 1905$ MIPS

MFLOPS: $(3 * 10000000) / (0.047227779 * 10^6) = 635$ MFLOPS

RESPUESTA:

Resultado para 10 componentes

Tiempo(seg.):0.000003127 / Tamaño Vectores:10 / $V1[0]+V2[0]=V3[0]$
 $(1.000000+1.000000=2.000000) / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /$

Resultado para 10000000

Tiempo(seg.):0.047227779 / Tamaño Vectores:10000000 / $V1[0]+V2[0]=V3[0]$
 $(1000000.000000+1000000.000000=2000000.000000) /$
 $V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /$

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 */

        call        clock_gettime
        xorl        %eax, %eax
        .p2align 4,,10
        .p2align 3
.L7:
        movsd       v1(%rax), %xmm0
        addq        $8, %rax
```

	addsd	v2-8(%rax), %xmm0
	movsd	%xmm0, v3-8(%rax)
	cmpq	%rbx, %rax
	jne	.L7
.L6:	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 ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i = 0, \dots, N-1$) 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

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

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

    int *v1, *v2, *v3;
    int i;
    if(argc < 2) {
        fprintf(stderr, "\n[ERROR] - Falta el numero de elementos. \n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    v1 = malloc(N*sizeof(int));
    v2 = malloc(N*sizeof(int));
    v3 = malloc(N*sizeof(int));

    double tiempo_ini, tiempo_fin, tiempo_total;

    #pragma omp parallel
    {
```

```
#pragma omp for

    for( i = 0; i < N; i++){

        v1[i] = i;
        v2[i] = i;

    }


#pragma omp single
    tiempo_ini = omp_get_wtime();

#pragma omp for

    for( i = 0; i < N; i++){

        v3[i]=v1[i]+v2[i];

    }

#pragma omp single

    tiempo_fin = omp_get_wtime();

}


tiempo_total = tiempo_fin - tiempo_ini;

printf("\nVector 1:");

for( i =0; i< N; i++)
    printf("%d ", v1[i]);
```

```

printf("\nVector 2:");

for( i =0; i< N; i++)
    printf("%d ", v2[i]);

printf("\nVector del resultado:");

for( i =0; i< N; i++)
    printf("%d ", v3[i]);

printf("\nEl tiempo en realizar la suma ha sido %f\n", tiempo_total);

exit (0);

}

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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 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


```

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

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

    int *v1, *v2, *v3;
    int i,j,k,l;
    if(argc < 2) {
        fprintf(stderr, "\n[ERROR] - Falta el numero de elementos. \n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    v1 = malloc(N*sizeof(int));
    v2 = malloc(N*sizeof(int));
    v3 = malloc(N*sizeof(int));

    double tiempo_ini, tiempo_fin, tiempo_total;

    #pragma omp parallel
    {
        #pragma omp sections
        {

            #pragma omp section
            for( i = 0; i< (N/4); i++){
                v1[i]=i;
                v2[i]=i;
            }

            #pragma omp section
            for( j = (N/4); j < (N/4)*2; j++){
                v1[j]=j;
                v2[j]=j;
            }

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

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

```

```

    }

}

#pragma omp single
    tiempo_ini = omp_get_wtime();

#pragma omp sections
{

    #pragma omp section
        for( i = 0; i < (N/4); i++){
            v3[i]=v1[i]+v2[i];
        }

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

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

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

}

#pragma omp single

    tiempo_fin = omp_get_wtime();

}

tiempo_total = tiempo_fin - tiempo_ini;

printf("\nVector 1:");

```

```

for( i =0; i< N; i++)
    printf("%d ", v1[i]);

printf("\nVector 2:");

for( i =0; i< N; i++)
    printf("%d ", v2[i]);

printf("\nVector del resultado:");

for( i =0; i< N; i++)
    printf("%d ", v3[i]);

printf("El tiempo en realizar la suma ha sido %f\n", tiempo_total);

exit (0);
}

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA: En el ejercicio 7 se podrán usar tantas threads como tamaño tengan los vectores, ya que solo se podran aprovechar una hebra por elemento. En el ejercicio 8 se podrán usar 4 hebras ya que lo he programado de tal manera que se reparta la tarea en 4 partes

10. 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:

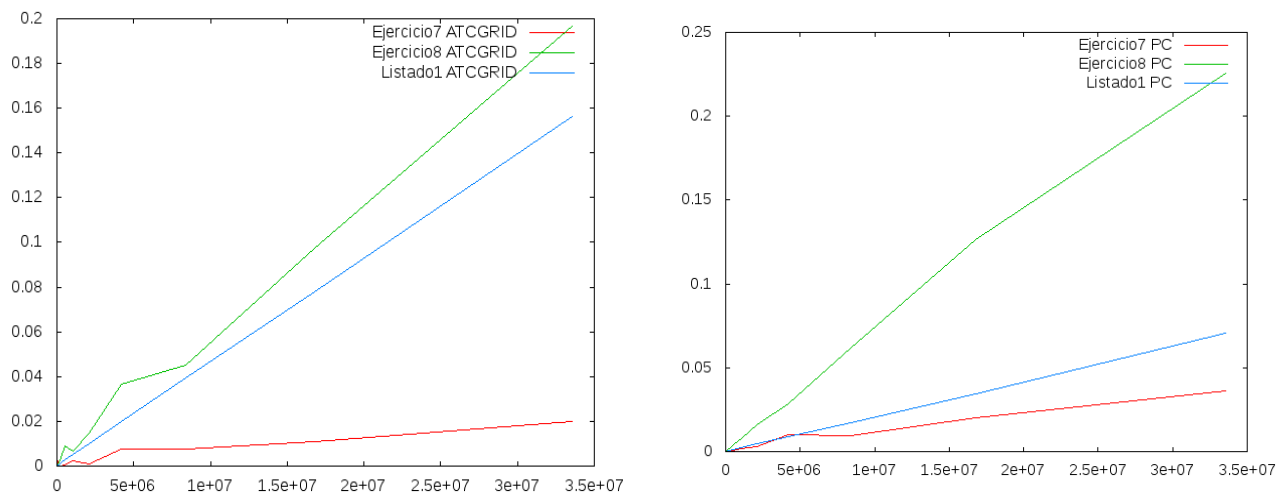
Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “?” por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

ATCGRID

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24 threads/cores	T. paralelo (versión sections) 4 threads/cores
1024	0.000010	0.001332	0.002934
2048	0.000016	0.000019	0.000009
4096	0.000025	0.000029	0.000016
8192	0.000053	0.000070	0.000967
16384	0.000109	0.000204	0.000042
32768	0.000189	0.000428	0.005119
65536	0.000384	0.000422	0.000066
131072	0.000674	0.001297	0.000093
262144	0.001262	0.003635	0.000230
524288	0.002672	0.008898	0.000374
1048576	0.005041	0.006805	0.002288
2097152	0.009957	0.014834	0.000883
4194304	0.019764	0.036765	0.007815
8388608	0.039217	0.045146	0.007478
16777216	0.077898	0.097215	0.011103
33554432	0.156117	0.196606	0.020039

PC

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 4 threads/cores	T. paralelo (versión sections) 4 threads/cores
1024	0.000124	0.000005	0.003981
2048	0.000125	0.000013	0.000018
4096	0.000144	0.000798	0.000028
8192	0.000157	0.000016	0.000057
16384	0.000163	0.000032	0.000089
32768	0.000215	0.000059	0.001993
65536	0.000212	0.000103	0.000294
131072	0.000311	0.000205	0.000898
262144	0.000711	0.000392	0.001943
524288	0.001205	0.000849	0.004070
1048576	0.002319	0.001689	0.007990
2097152	0.004595	0.002913	0.015882
4194304	0.008677	0.009932	0.028272
8388608	0.017420	0.009339	0.061928
16777216	0.034355	0.020105	0.126252
33554432	0.068414	0.036466	0.225746



11. 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: En el caso de un thread es igual pero en el caso de 4 el tiempo real es menor al de CPU

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 Componente s	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 4 Threads/cores		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
1024	real	0m0.001s		real	0m0.002s	
	user	0m0.000s		user	0m0.000s	
	sys	0m0.001s		sys	0m0.002s	
2048	real	0m0.001s		real	0m0.002s	
	user	0m0.000s		user	0m0.000s	
	sys	0m0.001s		sys	0m0.002s	
4096	real	0m0.001s		real	0m0.002s	
	user	0m0.000s		user	0m0.000s	
	sys	0m0.001s		sys	0m0.002s	
8192	real	0m0.001s		real	0m0.002s	
	user	0m0.000s		user	0m0.000s	
	sys	0m0.001s		sys	0m0.002s	
16384	real	0m0.001s		real	0m0.002s	
	user	0m0.000s		user	0m0.000s	
	sys	0m0.001s		sys	0m0.004s	
32768	real	0m0.001s		real	0m0.007s	
	user	0m0.000s		user	0m0.010s	
	sys	0m0.001s		sys	0m0.000s	
65536	real	0m0.001s		real	0m0.001s	
	user	0m0.000s		user	0m0.000s	
	sys	0m0.001s		sys	0m0.002s	
131072	real	0m0.002s		real	0m0.003s	
	user	0m0.000s		user	0m0.007s	
	sys	0m0.001s		sys	0m0.000s	
262144	real	0m0.004s		real	0m0.002s	
	user	0m0.004s		user	0m0.000s	
	sys	0m0.000s		sys	0m0.005s	
524288	real	0m0.005s		real	0m0.005s	
	user	0m0.000s		user	0m0.011s	
	sys	0m0.005s		sys	0m0.003s	
1048576	real	0m0.009s		real	0m0.005s	
	user	0m0.006s		user	0m0.004s	
	sys	0m0.003s		sys	0m0.013s	
2097152	real	0m0.016s		real	0m0.006s	
	user	0m0.012s		user	0m0.015s	
	sys	0m0.004s		sys	0m0.005s	
4194304	real	0m0.030s		real	0m0.012s	
	user	0m0.026s		user	0m0.024s	
	sys	0m0.004s		sys	0m0.019s	
8388608	real	0m0.057s		real	0m0.025s	
	user	0m0.050s		user	0m0.049s	
	sys	0m0.008s		sys	0m0.034s	
16777216	real	0m0.118s		real	0m0.046s	
	user	0m0.094s		user	0m0.115s	
	sys	0m0.024s		sys	0m0.053s	
33554432	real	0m0.223s		real	0m0.087s	
	user	0m0.171s		user	0m0.236s	
	sys	0m0.052s		sys	0m0.086s	