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): Raul Rodriguez Perez
Grupo de prácticas y profesor de prácticas: C1
Fecha de entrega:
Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la directiva parallel combinada con directivas de trabajo compartido en los ejemplos bucle-for.c y sections.c del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

RESPUESTA: Captura que muestre el código fuente bucle-forModificado.c

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

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

   int i, n = 9;

   if(argc < 2)     {
      fprintf(stderr,"\n[ERROR] - Falta nº iteraciones \n");
      exit(-1);
   }
   n = atoi(argv[1]);

#pragma omp parallel for

   for (i=0; i<n; i++)
      printf("thread %d ejecuta la iteración %d del bucle\n",
            omp_get_thread_num(),i);

   return(0);
}
```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

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

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

int main()  {

    #pragma omp parallel sections
   {
        #pragma omp section
            (void) funcA();
        #pragma omp section
            (void) funcB();
   }
   return 0;
}
```

2. Imprimir los resultados del programa single.c usando una directiva single dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva single incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva single. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

RESPUESTA: Captura que muestre el código fuente `single`Modificado.c

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

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

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

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

     #pragma omp single
     {
     for(i=0;i<n;i++) printf("b[%d] = %d\t",i,b[i]);
     printf("\n");
     }
  }

  return 0;
}
```

CAPTURAS DE PANTALLA:

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/Práctica 1
/1/ejer2$ ./singleModificado
Introduce valor de inicialización a: 4
Single ejecutada por el thread 8
b[0] = 4        b[1] = 4        b[2] = 4        b[3] = 4        b[4] = 4        b[5] = 4b
[6] = 4 b[7] = 4        b[8] = 4
```

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 `single`Modificado2.c

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

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

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

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

    #pragma omp master
    {
    for(i=0;i<n;i++) printf("b[%d] = %d\t",i,b[i]);
    printf("Master ejecutada por el thread %d\n",
            omp_get_thread_num());
    printf("\n");
    }
  }

  return 0;
}
```

CAPTURAS DE PANTALLA:

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/Práctica 1
/1/ejer3$ ./singleModificado2
Introduce valor de inicialización a: 4
Single ejecutada por el thread 9
b[0] = 4        b[1] = 4        b[2] = 4        b[3] = 4        b[4] = 4        b[5] = 4b
[6] = 4 b[7] = 4        b[8] = 4        Master ejecutada por el thread 0
```

RESPUESTA A LA PREGUNTA:

La diferencia con respecto a la ejecución del ejercicio anterior es que en el anterior, al ser utilizada la directiva single, los resultados pueden ser imprimidos por cualquiera de las hebras. En este ejercicio los resultados serán imprimidos siempre por la hebra 0, ya que hemos usado la directiva master.

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 es necesario que la hebra master espere a la ejecución del resto de hebras. Gracias a la barrera conseguimos que la hebra master espere al resto, pero si la quitamos puede ocurrir el caso de que esta acabe su ejecución antes que el resto o que queden hebras sin finalizar su ejecución, en cuyo caso el resultado del cálculo sería incorrecto.

1.1.1

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,…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:

```
[c1estudiante4@atcgrid ejer5]$ time ./SumaVectores 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.014541675       / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000
000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999
](1999999.900000+0.100000=2000000.000000) /

real    0m0.040s
user    0m0.028s
sys     0m0.012s
```

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

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

```
[c1estudiante4@atcgrid ejer6]$ time ./SumaVectores 10
Tamaño Vectores:10 (4 B)
Tiempo:0.000000191       / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.000000+1.0
00000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /

real    0m0.002s
user    0m0.000s
sys     0m0.002s
```

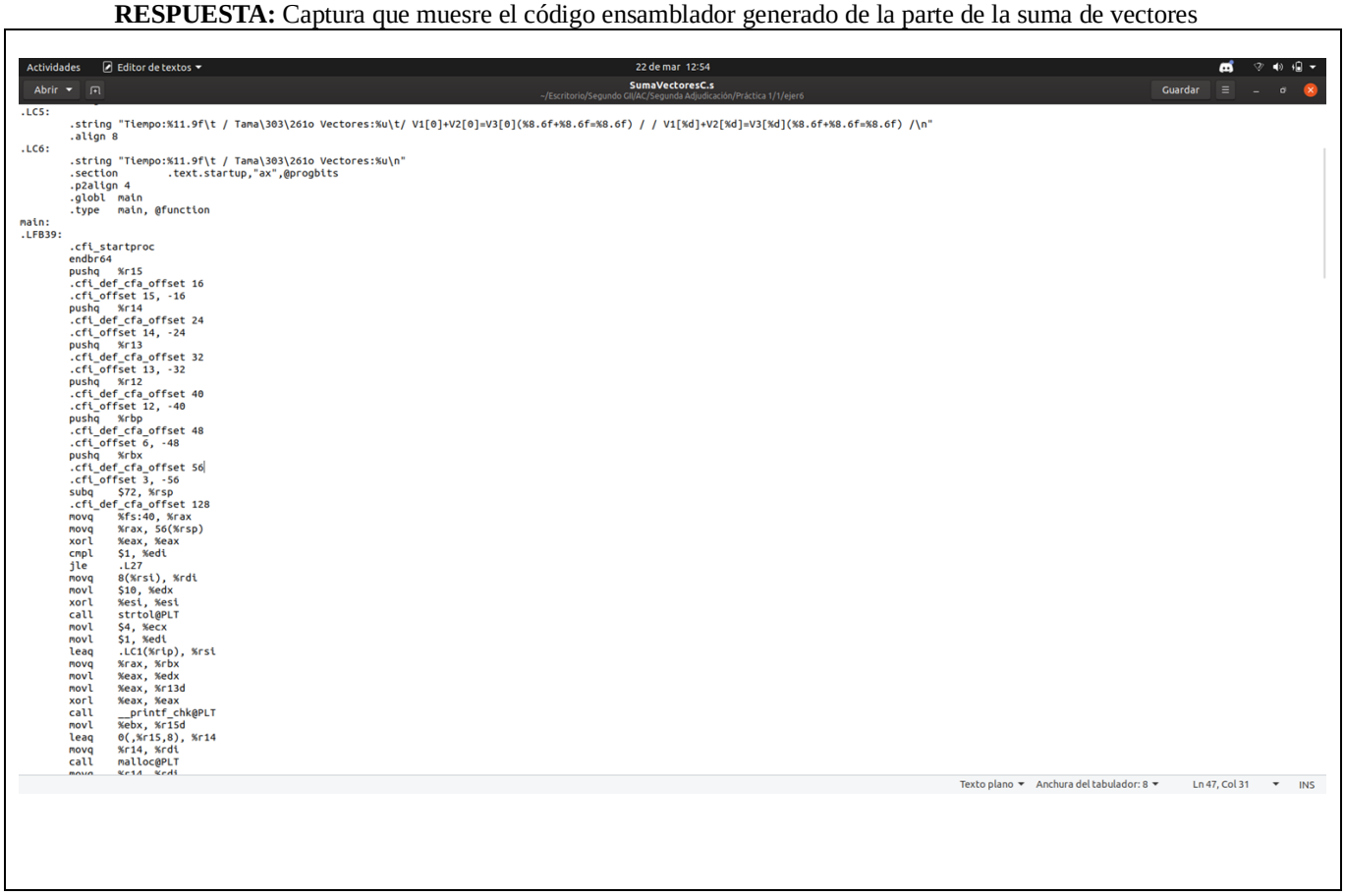
```
[c1estudiante4@atcgrid ejer6]$ time ./SumaVectores 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.013246910       / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000
000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999
](1999999.900000+0.100000=2000000.000000) /

real    0m0.035s
user    0m0.018s
sys     0m0.017s
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

MIPS = (9*(10**7))/(0,033536260*(10**6)) = 2683,66 MIPS

MFLOPS = (3*(10**7))/(0,03353626*(10**6)) = 894,55 MFLOPS

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




```
.LC5:
        .string "Tiempo:%11.9f\t / Tama\303\261o 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"
        .align 8
.LC6:
        .string "Tiempo:%11.9f\t / Tama\303\261o Vectores:%u\n"
        .section        .text.startup,"ax",@progbits
        .p2align 4
        .globl  main
        .type   main, @function
main:
.LFB39:
        .cfi_startproc
        endbr64
        pushq   %r15
        .cfi_def_cfa_offset 16
        .cfi_offset 15, -16
        pushq   %r14
        .cfi_def_cfa_offset 24
        .cfi_offset 14, -24
        pushq   %r13
        .cfi_def_cfa_offset 32
        .cfi_offset 13, -32
        pushq   %r12
        .cfi_def_cfa_offset 40
        .cfi_offset 12, -40
        pushq   %rbp
        .cfi_def_cfa_offset 48
        .cfi_offset 6, -48
        pushq   %rbx
        .cfi_def_cfa_offset 56
        .cfi_offset 3, -56
        subq    $72, %rsp
        .cfi_def_cfa_offset 128
        movq    %fs:40, %rax
        movq    %rax, 56(%rsp)
        xorl    %eax, %eax
        cmpl    $1, %edi
        jle     .L27
        movq    8(%rsi), %rdi
        movl    $10, %edx
        xorl    %esi, %esi
        call    strtol@PLT
        movl    $4, %ecx
        movl    $1, %edi
        leaq    .LC1(%rip), %rsi
        movq    %rax, %rbx
        movl    %eax, %edx
        movl    %eax, %r13d
        xorl    %eax, %eax
        call    __printf_chk@PLT
        movl    %ebx, %r15d
        leaq    0(,%r15,8), %r14
        movq    %r14, %rdi
        call    malloc@PLT
```

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,…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

```
unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
printf("Tamaño Vectores:%u (%lu B)\n",N, sizeof(unsigned int));
#ifdef VECTOR_LOCAL
double v1[N], v2[N], v3[N];   // Tamaño variable local en tiempo de ejecución ...
    // disponible en C a partir de C99
#endif
#ifdef VECTOR_GLOBAL
if (N>MAX) N=MAX;
#endif
#ifdef VECTOR_DYNAMIC
double *v1, *v2, *v3;
v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
v2 = (double*) malloc(N*sizeof(double));
v3 = (double*) malloc(N*sizeof(double));
if ((v1 == NULL) || (v2 == NULL) || (v2 == NULL)) {
  printf("No hay suficiente espacio para los vectores \n");
  exit(-2);
}
#endif
#pragma omp parallel
{
  #pragma omp for
  //Inicializar vectores
  for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
  }

  #pragma omp single
  clock_gettime(CLOCK_REALTIME,&cgt1);
  #pragma omp for
  //Calcular suma de vectores
  for(i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];

  #pragma omp single
  clock_gettime(CLOCK_REALTIME,&cgt2);
}

  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
       (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

  //Imprimir resultado de la suma y el tiempo de ejecución
  if (N<10) {
  printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
  for(i=0; i<N; i++)
    printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
           i,i,i,v1[i],v2[i],v3[i]);
  }
  else
    printf("Tiempo:%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]);

#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
return 0;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/P
ráctica 1/1/ejer7$ ./SumaVectoresC 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000000892       / 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) /
```

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/P
ráctica 1/1/ejer7$ ./SumaVectoresC 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000004966       / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.1
00000=2.200000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
```

8. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las parallel y sections/section (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva for); es decir, hay que repartir el trabajo (tareas) entre varios threads usando sections/section. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función omp_get_wtime() en lugar de clock_gettime(). NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

   RESPUESTA: Captura que muestre el código fuente implementado

```
        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_inicio, tiempo_final, 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_inicio = 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(k = (N/4)*2; k < (N/4)*3; k++){
            v3[k] = v1[k] + v2[k];
        }
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/P
ráctica 1/1/ejer8$ ./Suma2Vectores 8

Vector 1: 0
1
2
3
4
5
6
7

Vector 2: 0
1
2
3
4
5
6
7

Vector del resultado: 0
2
75856176
0
8
10
12
14

El tiempo en realizar la suma ha sido 0.000008
```

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 pueden declarar tantas hebras como tamaño le des a los vectores, ya que solo se aprovecha una hebra por elemento. En el ejercicio 8, lo he programado de forma que se divida el trabajo en 4 partes, así que se pueden usar 4 hebras.

10. Rellenar una tabla como la Tabla 212 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos (use el máximo número de cores físicos del computador que como máximo puede aprovechar el código, no use un número de threads superior al número de cores físicos). Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

    RESPUESTA:

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 del computador que como máximo puede aprovechar el código.

11. Rellenar una tabla como la 12 Tabla 3 para atcgrid con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con time para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.
RESPUESTA:

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión** for **¿? Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						