Cuaderno de prácticas de Arquitectura de Computadores

Grado en Ingeniería Informática

Memoria
Bloque Práctico 1

Alumno: Manuel Jesús García Manday
DNI: 48893432D
Grupo: *D3*

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

RESPUESTA: código fuente `bucle-forModificado.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* 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. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

#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();
  }
}
```

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. Tamaño 10.*/
/* 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
      {
        for (i=0; i<n; i++){
          printf("Hebra %d realiza b[%d] =
%d\t",omp_get_thread_num(),i,b[i]);
          printf("\n");
         }
      }
  }
}
```

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practica/Seminario1 $ ./singleModificado
Introduce valor de inicialización a: 9
Single ejecutada por el thread 1
Hebra 1 realiza b[0] = 9
Hebra 1 realiza b[1] = 9
Hebra 1 realiza b[2] = 9
Hebra 1 realiza b[3] = 9
Hebra 1 realiza b[4] = 9
Hebra 1 realiza b[5] = 9
Hebra 1 realiza b[6] = 9
Hebra 1 realiza b[7] = 9
Hebra 1 realiza b[8] = 9
```

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: código fuente `masterModificado2.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* 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
      {
        for (i=0; i<n; i++){
          printf("Hebra %d realiza b[%d] =
%d\t",omp_get_thread_num(),i,b[i]);
          printf("\n");
         }
      }
  }
}
```

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practica/Seminario1 $ ./singleModificado
Introduce valor de inicialización a: 5
Single ejecutada por el thread 1
Hebra 0 realiza b[0] = 5
Hebra 0 realiza b[1] = 5
Hebra 0 realiza b[2] = 5
Hebra 0 realiza b[3] = 5
Hebra 0 realiza b[4] = 5
Hebra 0 realiza b[5] = 5
Hebra 0 realiza b[6] = 5
Hebra 0 realiza b[7] = 5
Hebra 0 realiza b[8] = 5
```

RESPUESTA A LA PREGUNTA: Que en el primer caso (single) cualquier hebra puede solicitar el valor por pantalla y luego imprimirlos, como vemos en la captura es la hebra 1 la que realiza ambas tareas, mientras que en el segundo caso (master) la primera tarea puede seguir siendo realiazada por cualquier hebra, vemos en la captura de pantalla correspondiente que lo realiza la hebra 1, pero vemos que la salida la realiza la hebra 0, esto es lo que diferencia a una claúsula de otra, que en master el código siempre es ejecutado por la hebra 0, además de que no existe barrera implícita.

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 la directiva atomic no tiene implícita la barrera, y de este modo, la primera hebra que llegue a esa directiva y realiza la operación, no esperará a las demás e irá a imprimir directamente el resultado total de la suma, que puede no ser correcto debido a que faltan hebras por sumar su resultado local. A veces puede que el resultado sea el correcto debido a que obligamos a que sea la hebra maestro la que realize la operación de imprimir el resultado, por eso, si por casualidad esa hebra fuese la última en realizar la sumalocal con el total, daría la suma correcta.

Resto de ejercicios

En clase presencial y en casa, usando plataformas (procesadores + SO) de 64 bits**,** se trabajarán los siguientes ejercicios y cuestiones:

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 el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema para el ejecutable en atcgrid y en el PC local. Obtenga los tiempos para vectores con 10000000 componentes.

   CAPTURAS DE PANTALLA:

   pclocal

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practica/Seminario1 $ time ./SumaVectoresGlobal 10000000
Tiempo(seg.):0.047710302


real    0m0.175s
user    0m0.080s
sys     0m0.092s
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practica/Seminario1 $
```

   atcgrid

```
[D3estudiante5@atcgrid hello]$ echo 'time hello/SumaVectoresGlobal 10000000' | qsub -q ac
9513.atcgrid
[D3estudiante5@atcgrid hello]$ cat STDIN.o9513
Tiempo(seg.):0.035953446

[D3estudiante5@atcgrid hello]$ cat STDIN.e9513

real    0m0.115s
user    0m0.066s
sys     0m0.045s
[D3estudiante5@atcgrid hello]$
```

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).

CAPTURAS DE PANTALLA:

```
[D3estudiante5@atcgrid hello]$ echo 'time hello/SumaVectoresGlobal 10000000' | qsub -q ac
9513.atcgrid
[D3estudiante5@atcgrid hello]$ cat STDIN.o9513
Tiempo(seg.):0.035953446

[D3estudiante5@atcgrid hello]$ cat STDIN.e9513

real    0m0.115s
user    0m0.066s
sys     0m0.045s
[D3estudiante5@atcgrid hello]$
```

```
    call    clock_gettime
    xorl    %eax, %eax
    .p2align 4,,10
    .p2align 3
.L7:
    movsd   v1(%rax), %xmm0
    addsd   v2(%rax), %xmm0
    movsd   %xmm0, v3(%rax)
    addq    $8, %rax
    cmpq    %rbp, %rax
    jne .L7
.L8:
    leaq    16(%rsp), %rsi
    xorl    %edi, %edi
    call    clock_gettime
```

RESPUESTA:

Para 10 componentes:

NI = 3 + (6 * 10) + 2 = 65 instrucciones

Tcpu = 0.035953446 seg.

MIPS = NI / Tcpu * 10^6 = 1.8 * 10^-3 MIPS

Operaciones_en_coma_flotante = 3 + (3 * 10) + 2 = 35 inst.

Tcpu = 0.035953446 seg.

MFLOPS = Operaciones_en_coma_flotante / Tcpu * 10^6 =

= 9.73 * 10^-4 MFLOPS

Para 10000000 componentes:

NI = 3 + (6 * 10000000) + 2 = 60000005 instrucciones

Tcpu = 0.035953446 seg.

MIPS = NI / Tcpu * 10^6 = 1668.82 MIPS

Operaciones_en_coma_flotante = 3 + (3 * 10000000) + 2 = 30000005 instrucciones.

Tcpu = 0.035953446 seg.

MFLOPS = Operaciones_en_coma_flotante / Tcpu * 10^6 =

= 834.41 MFLOPS

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: código fuente implementado

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
```

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

#define VECTOR_GLOBAL
#ifdef VECTOR_GLOBAL
#define MAX 10000000
double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char ** argv)
{
  int i;
  struct timespec cgt1,cgt2; double t1, t2, tfin; //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)

   //Inicializar vectores en paralelo
   #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; //los valores dependen de N

     }



    t1 = omp_get_wtime();
    //Calcular suma de vectores en paralelo
     #pragma omp parallel for
         for(i=0; i<N; i++)
            v3[i] = v1[i] + v2[i];

    t2 = omp_get_wtime();
    tfin = t2 - t1;

    //Imprimir resultado de la suma y el tiempo de ejecución
    #ifdef PRINTF_ALL
        printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",tfin,N);
     #else
        printf("\nTiempo(seg.):%11.9f\t Tamaño Vectores:%u\t\n\n",
tfin, N);
```

```
        if(N < 12){ /* Solo imprimimos todos los resultados del vector
resultante para valores pequeños, como se indica en el apartado 3) del
ejercicio */
            printf("Suma del vector resultante v3\n\n");
                for(i=0; i < N; i++)
                   printf("V3[%d]==%8.6f\n", i,v3[i]);
         }
         else{
             printf("V1[0]==%8.6f\t\tV2[0]==%8.6f\t\tV3[0]==
%8.6f\n\n", v1[0], v2[0], v3[0]);

            printf("V1[N-1]==%8.6f\t\tV2[N-1]==%8.6f\t\tV3[N-1]==
%8.6f\n\n", v1[N-1], v2[N-1], v3[N-1]);
        }
     #endif

     return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practica/Seminario1 $ ./SumaVectoresGlobal 11

Tiempo(seg.):0.000032464        Tamaño Vectores:11

Suma del vector resultante v3

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
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practica/Seminario1 $ ./SumaVectoresGlobal 14

Tiempo(seg.):0.005245332        Tamaño Vectores:14

V1[0]==1.400000         V2[0]==1.400000         V3[0]==2.800000

V1[N-1]==2.700000               V2[N-1]==0.100000               V3[N-1]==2.800000
```

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

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define VECTOR_GLOBAL
#ifdef VECTOR_GLOBAL

#define MAX 10000000
double v1[MAX], v2[MAX], v3[MAX];
int tama;
#endif

void inicializaVector1(){
   int i;
   for(i = 0; i < tama; i++)
         v1[i] = tama*0.1+i*0.1;
}

void inicializaVector2(){
   int i;
   for(i = 0; i < tama; i++)
        v2[i] = tama*0.1-i*0.1;
}

int main(int argc, char ** argv)
{
   int i, c;
   struct timespec cgt1,cgt2;
   double t1, t2, tfin; //para tiempo de ejecución
```

```
    if (argc<2){//Leer argumento de entrada (no de componentes del
vector)
         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)
     tama = N;
     c = N/4 + 1;

     // Dividimos la inicializacion en secciones
     #pragma omp parallel sections
     {
            #pragma omp section
                (void) inicializaVector1();
            #pragma omp section
                 (void) inicializaVector2();
     }

     t1 = omp_get_wtime();
     //Calcular suma de vectores en secciones
     #pragma omp parallel sections private (i)
     {
            #pragma omp section
                for(i = 0; i < c; i++)
                   v3[i] = v1[i] + v2[i];
            #pragma omp section
                for(i = c; i < 2*c ; i++)
                   v3[i] = v1[i] + v2[i];
            #pragma omp section
                for(i = 2*c; i < 3*c; i++)
                   v3[i] = v1[i] + v2[i];
            #pragma omp section
                for(i = 3*c; i < N; i++)
                   v3[i] = v1[i] + v2[i];
     }

     t2 = omp_get_wtime();
     tfin = t2 - t1;

     //Imprimir resultado de la suma y el tiempo de ejecución
     #ifdef PRINTF_ALL
       printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",tfin,N);
     #else
       printf("\nTiempo(seg.):%11.9f\t Tamaño Vectores:%u\t\n\n",
tfin, N);
       if(N < 12){ /* Solo imprimimos todos los resultados del vector
resultante para valores pequeños, como se indica en el apartado 3) del
ejercicio */
            printf("Suma del vector resultante v3\n\n");
               for(i=0; i < N; i++)
                   printf("V3[%d]==%8.6f\n", i,v3[i]);
       }
```

```
        else{

             printf("V1[0]==%8.6f\t\tV2[0]==%8.6f\t\tV3[0]==%8.6f\n\n",
v1[0], v2[0], v3[0]);
             printf("V1[N-1]==%8.6f\t\tV2[N-1]==%8.6f\t\tV3[N-1]==
%8.6f\n\n", v1[N-1], v2[N-1], v3[N-1]);
       }
      #endif
      return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practica/Seminario1 $ ./SumaVectoresGlobal_Ejercicio8 15

Tiempo(seg.):0.000030178         Tamaño Vectores:15

V1[0]==1.500000         V2[0]==1.500000         V3[0]==3.000000

V1[N-1]==2.900000               V2[N-1]==0.100000               V3[N-1]==3.000000

jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practica/Seminario1 $ ./SumaVectoresGlobal_Ejercicio8 11

Tiempo(seg.):0.000044837         Tamaño Vectores:11

Suma del vector resultante v3

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
```

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 la nueva versión del ejercicio 7 se podrá utilizar como máximo el número de threads y cores que soporte la máquina en la que se ejecute el programa tanto en la inicializaciónde los vectores como en la suma (tomando por defecto que la varibale OMP_DYNAMIC esta a TRUE). En la nueva versión del ejercicio 8 en la parte de inicialización solo se utilizarán 2 threads, ya que en este caso lo hemos paralelizado en dos secciones, mientras que en la suma de los vectores utilizaremos tantos threads como secciones tengamos, en nuestro caso 4.

10. Rellenar una tabla 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 (*elapsed time*) 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.

RESPUESTA:

pclocal

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) 4 threads/cores**	**T. paralelo (versión `sections`) 4 threads/cores**
65536	0.000675225	0.000397201	0.000373322
131072	0.001491632	0.000860371	0.000823237
262144	0.002406193	0.001557094	0.001409803
524288	0.002800876	0.003103279	0.002272950
1048576	0.005507122	0.004856963	0.007111249
2097152	0.010709475	0.006731538	0.007250409
4194304	0.021280478	0.006731538	0.013340933
8388608	0.042461757	0.026382429	0.026251207
16777216	0.084534825	0.051427400	0.052074169
33554432	0.169001700	0.102897256	0.106090207
67108864	0.324546567	0.205116094	0.207716935

atcgrid

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`) 24 threads/cores**	**T. paralelo (versión `sections`) 4 threads/cores**
65536	0.000578655	0.004140011	0.003473490
131072	0.000610976	0.006144360	0.002713520
262144	0.001425565	0.005044994	0.004338708
524288	0.002845649	0.005732458	0.003854355
1048576	0.004621343	0.005976548	0.005584468
2097152	0.008365676	0.008645637	0.008767267
4194304	0.016245768	0.011048306	0.013596497
8388608	0.031658700	0.015819224	0.026466083
16777216	0.062490906	0.029107602	0.042110082
33554432	0.125234788	0.057185204	0.079755234
67108864	0.245657989	0.108581803	0.202189255

11. Rellenar una tabla para atcgrid y otra 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.
RESPUESTA:

pclocal

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core** ***Elapsed CPU-user CPU- sys***	**Tiempo paralelo/versión `for` 4 Threads/cores** ***Elapsed CPU-user CPU- sys***
65536	0.001155083---0.000---0.004	0.000390997---0.012---0.004
131072	0.002339463---0.000---0.008	0.000807701---0.000---0.012
262144	0.001898739---0.008---0.004	0.001612484---0.004---0.016
524288	0.003742406---0.012---0.008	0.003211003---0.008---0.032
1048576	0.007352790---0.016---0.020	0.003525950---0.024---0.032
2097152	0.014247701---0.040---0.024	0.006900592---0.032---0.048
4194304	0.028294783---0.048---0.064	0.015311851---0.064---0.084
8388608	0.055740123---0.128---0.088	0.026878270---0.140---0.124
16777216	0.110554190---0.256---0.164	0.053246653---0.224---0.268
33554432	0.221201113---0.476---0.360	0.104906779---0.448---0.488
67108864	0.439209950---1.004---0.636	0.208832396---0.744---1.080

atcgrid

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core** ***Elapsed CPU-user CPU- sys***	**Tiempo paralelo/versión** `for` **24 Threads/cores** ***Elapsed CPU-user CPU- sys***
65536	0.000720522---0.000---0.003	0.003347323---0.174---0.116
131072	0.001179376---0.003---0.002	0.004608388---0.167---0.137
262144	0.002617988---0.005---0.005	0.003625809---0.092---0.146
524288	0.004307977---0.011---0.005	0.003971558---0.088---0.187
1048576	0.006646340---0.018---0.010	0.004829246---0.151---0.177
2097152	0.012920141---0.033---0.013	0.008454072---0.306---0.182
4194304	0.025554737---0.067---0.020	0.009236538---0.329---0.218
8388608	0.050802415---0.130---0.039	0.016350789---0.603---0.277
16777216	0.100608117---0.258---0.072	0.029959452---1.075---0.435
33554432	0.200932439---0.506---0.149	0.056073902---2.080---0.951
67108864	0.400114744---1.012---0.293	0.108696727---4.059---1.746