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): Jose Luis Pedraza Román
Grupo de prácticas y profesor de prácticas: A2, Christian Morillas
Fecha de entrega: 25/3/2020
Fecha evaluación en clase: X

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`

```
#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`

```
#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() );
}
```

```
void 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: Captura que muestre el código fuente `singleModificado.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 inicializacion 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;
     //}


      //printf("Después de la región parallel:\n");
      #pragma omp single
      {
         for (i=0; i<n; i++)
            printf("b[%d] = %d\t",i,b[i]);
         printf("\n");
         printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
      }
    }


   return(0);
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer2] 2020-03-05 jueves
$gcc -O2 -fopenmp -o singleModificado singleModificado.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer2] 2020-03-05 jueves
$export OMP_DYNAMIC=FALSE
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer2] 2020-03-05 jueves
$export OMP_NUM_THREADS=8
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer2] 2020-03-05 jueves
$./singleModificado
Introduce valor de inicializacion a: 23
Single ejecutada por el thread 6
b[0] = 23       b[1] = 23       b[2] = 23       b[3] = 23       b[4] = 23       b[5] = 23       b
[6] = 23        b[7] = 23       b[8] = 23
Single ejecutada por el thread 2
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer2] 2020-03-05 jueves
$export OMP_NUM_THREADS=3
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer2] 2020-03-05 jueves
$./singleModificado
Introduce valor de inicializacion a: 12
Single ejecutada por el thread 1
b[0] = 12       b[1] = 12       b[2] = 12       b[3] = 12       b[4] = 12       b[5] = 12       b
[6] = 12        b[7] = 12       b[8] = 12
Single ejecutada por el thread 2
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer2] 2020-03-05 jueves
$
```

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`

```
#include <stdio.h>
#include <omp.h>
void 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 inicializacion 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;
     //}


      //printf("Después de la región parallel:\n");
```

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

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer3] 2020-03-05 jueves
$gcc -O2 -fopenmp -o singleModificado2 singleModificado2.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer3] 2020-03-05 jueves
$export OMP_DYNAMIC=FALSE
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer3] 2020-03-05 jueves
$export OMP_NUM_THREADS=8
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer3] 2020-03-05 jueves
$./singleModificado2
Introduce valor de inicializacion a: 23
Single ejecutada por el thread 1
b[0] = 23       b[1] = 23       b[2] = 23       b[3] = 23       b[4] = 23       b[5] = 23       b
[6] = 23        b[7] = 23       b[8] = 23
Master ejecutada por el thread 0
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer3] 2020-03-05 jueves
$export OMP_NUM_THREADS=3
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer3] 2020-03-05 jueves
$./singleModificado2
Introduce valor de inicializacion a: 12
Single ejecutada por el thread 1
b[0] = 12       b[1] = 12       b[2] = 12       b[3] = 12       b[4] = 12       b[5] = 12       b
[6] = 12        b[7] = 12       b[8] = 12
Master ejecutada por el thread 0
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer3] 2020-03-05 jueves
$
```

RESPUESTA A LA PREGUNTA:

La hebra que ejecuta la el printf de la directiva master siempre es la 0, mientras que en el ejercicio anterior podría ser cualquiera de las asignadas para ese proceso.

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 no existiría barrera alguna, ni implicita (dado que master no la tiene) ni explicita, que sincronizase las hebras para asegurar que se realizan correctamente todas las sumas locales por cada una de ellas. Por esto, podrían faltar una o varias sumas locales por realizar y asi alterar el resultado correcto.

Debemos asegurarnos de que todas las demás hebras hayan calculado lo que les corresponde y comprobar siempre el correcto funcionamiento del programa para este tipo de directivas sin barrera implícita.

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:

-PC:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer5] 2020-03-12 jueves
$gcc  -O2  SumaVectores.c -o SumaVectores_globales -lrt
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer5] 2020-03-12 jueves
$time ./SumaVectores_globales 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.033569563       / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.0000
00) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0,109s
user    0m0,058s
sys     0m0,050s
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer5] 2020-03-12 jueves
$
```

-ATCGRID:

```
[a2estudiante18@atcgrid ejer5]$ sbatch -p ac --wrap "time ./SumaVectores_globales 10000000"
Submitted batch job 20244
[a2estudiante18@atcgrid ejer5]$ ls
slurm-20244.out  SumaVectores_globales
[a2estudiante18@atcgrid ejer5]$ cat slurm-20244.out
Tamaño Vectores:10000000 (4 B)
Tiempo:0.041756968       / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.0000
00) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0.163s
user    0m0.048s
sys     0m0.055s
[a2estudiante18@atcgrid ejer5]$
```

En mi PC tenemos que la suma del tiempo de CPU del usuario + el tiempo de CPU del sistema = 0,108s que es menor que el tiempo real = 0,109s.

En un nodo de atcgrid tenemos que la suma del tiempo de CPU del usuario + tiempo de CPU del sistema = 0,103s que es menor que el tiempo real = 0,163s.

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):

```
jos@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer6$ gcc  -O2  SumaVectores.c -o SumaVectores -lrt
jos@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer6$ gcc  -O2  SumaVectores.c -S SumaVectores -lrt
gcc: warning: SumaVectores: linker input file unused because linking not done
jos@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer6$
```

Para tamaño 10:

sumavectores10.out

```
1 Tamaño Vectores:10 (4 B)
2 Tiempo:0.000379245   / 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) /
```

Para tamaño 1000:

sumavectores1000.out

```
1 Tamaño Vectores:1000 (4 B)
2 Tiempo:0.000393727   / Tamaño Vectores:1000 / V1[0]+V2[0]=V3[0](100.000000+100.000000=200.000000) / /
  V1[999]+V2[999]=V3[999](199.900000+0.100000=200.000000) /
```

Para tamaño 100000:

sumavectores100000.out

```
1 Tamaño Vectores:100000 (4 B)
2 Tiempo:0.000618525   / Tamaño Vectores:100000   / V1[0]+V2[0]=V3[0](10000.000000+10000.000000=20000.000000) / /
  V1[99999]+V2[99999]=V3[99999](19999.900000+0.100000=20000.000000) /
```

Para tamaño 10000000:

sumavectores10000000.out

```
1 Tamaño Vectores:10000000 (4 B)
2 Tiempo:0.042026396   / 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) /
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

MIPS NX+y+z/10^6

(contar TODAS)

MFLOPS %xmm_ (solo contar las instrucciones con este operando, que puede

ser de operación de suma, resta... o de carga de registro en coma flotante...)

(.p2alignX no es una instrucción, es una etiqueta y no hay que contarla)

MIPS => 1 inst + n.oit*6 + 2 inst / tiempo*10^6

MIPS(10): 1+(6*10) +2/ (0.000379245*10^6) = 0,166119527

MIPS(1000): 1+(6*1000) +2/ (0.000393727*10^6) = 15,246604881

MIPS(100000): 1+(6*100000) +2/ (0.000618525*10^6) = 970,054565296

MIPS(10000000): 1+(6*10000000)+2 / (0.042026396*10^6) = 1427,674240732

MFLOPS => 3*noit / tiempo*10^6

MFLOPS(10): (3*10) / (0.000379245*10^6) = 0,079104537

MFLOPS(1000): (3*1000) / (0.000393727*10^6) = 7,619492694

MFLOPS(100000): (3*100000) / (0.000618525*10^6) = 485,024857524

MFLOPS(10000000): (3*10000000) / (0.042026396*10^6) = 713,837084674

Por lo que podemos comprobar, con tamaños tan pequeños como 10 no aprovechamos nada el paralelismo que nos puede llegar a ofrecer ATCGRID, por ellos probamos con más valores a parte de 10 y 10000000.

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

```
call            clock_gettime@PLT
                xorl            %eax, %eax
                .p2align 4,,10
                .p2align 3
.L5:
                movsd           (%r12,%rax,8), %xmm0
                addsd           0(%r13,%rax,8), %xmm0
                movsd           %xmm0, (%r14,%rax,8)
                addq            $1, %rax
                cmpl            %eax, %ebp
                ja              .L5
                leaq            16(%rsp), %rsi
                xorl            %edi, %edi
                call            clock_gettime@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

```
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <omp.h>
//#define PRINTF_ALL    //comentar para quitar el printf...


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




                //Leer argumento de entrada (no de componentes del vector)
                if (argc<2){
                                printf("Faltan no componentes del vector\n");
                                exit(-1);
                }
                unsigned int N = atoi(argv[1]);
```

```
            double *v1, *v2, *v3;
            v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
            v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
            v3 = (double*) malloc(N*sizeof(double));
            if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
                        printf("Error en la reserva de espacio para los vectores\n");
                        exit(-2);
            }
   double cgt1,cgt2, ncgt;
   int i;


            //Inicializar vectores
   #pragma omp parallel for
            for(i=0; i<N; i++){
      //printf("Thread %d ejecuta iteracion %d del bucle de inicializacion\n",
omp_get_thread_num(),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 for
            for(i=0; i<N; i++){
                        v3[i] = v1[i] + v2[i];
      //printf("Thread %d ejecuta iteracion %d del bucle de calculo\n",
omp_get_thread_num(),i);
   }
            cgt2 = omp_get_wtime();


            ncgt=cgt2-cgt1;


            //Imprimir resultado de la suma y el tiempo de ejecución
            #ifdef PRINTF_ALL
            printf("Tiempo(seg.):%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(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]);
            #endif


    //liberar espacio reservado
            free(v1);
            free(v2);
            free(v3);


            return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer7] 2020-03-20 viernes
$gcc -fopenmp -O2 SumaVectores_parallel_for.c -o SumaVectores_parallel_for -lrt
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer7] 2020-03-20 viernes
$./SumaVectores_parallel_for 8
Tiempo(seg.):0.000007423        / 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) /
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer7] 2020-03-20 viernes
$./SumaVectores_parallel_for 11
Tiempo(seg.):0.000007279        / 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) /
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP1/ejer7] 2020-03-20 viernes
```

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

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


#define SECTIONS 4
//#define PRINTF_ALL    //comentar para quitar el printf...


void suma(double *v1, double *v2, double *v3, int ini, int fin){


   for(ini; ini<fin; ini++){
      v3[ini]=v1[ini]+v2[ini];
      //printf("Thread %d ejecuta iteracion %d del bucle de calculo\n",
omp_get_thread_num(),i);
   }


}




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




            //Leer argumento de entrada (no de componentes del vector)
            if (argc<2){
                        printf("Faltan no componentes del vector\n");
                        exit(-1);
            }
```

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


              double *v1, *v2, *v3;
              v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
              v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc
devuelve NULL
              v3 = (double*) malloc(N*sizeof(double));
              if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
                           printf("Error en la reserva de espacio para los vectores\n");
                           exit(-2);
              }
   double cgt1,cgt2, ncgt;
   //int i;


              //Inicializar vectores
   #pragma omp parallel sections /*private(i)*/
   {
                 #pragma omp section
          for(int i=0; i<N; i++){
             //printf("Thread %d ejecuta iteracion %d del bucle de inicializacion\n",
omp_get_thread_num(),i);
                        v1[i] = N*0.1+i*0.1;


          }
       #pragma omp section
          for(int j=0; j<N; j++){
             //printf("Thread %d ejecuta iteracion %d del bucle de inicializacion\n",
omp_get_thread_num(),i);
                        v2[j] = N*0.1-j*0.1; //los valores dependen de N
          }
              }


   cgt1 = omp_get_wtime();
              //Calcular suma de vectores
   #pragma omp parallel sections
   {
      #pragma omp section
         (void) suma(v1,v2,v3,0,N/4);
      #pragma omp section
         (void) suma(v1,v2,v3,N/4,N/2);
      #pragma omp section
         (void) suma(v1,v2,v3,N/2,(3*N)/4);
      #pragma omp section
         (void) suma(v1,v2,v3,(3*N)/4,N);
   }
              cgt2 = omp_get_wtime();


              ncgt=cgt2-cgt1;


              //Imprimir resultado de la suma y el tiempo de ejecución
              #ifdef PRINTF_ALL
              printf("Tiempo(seg.):%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(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]);
            #endif




            free(v1); // libera el espacio reservado para v1
            free(v2); // libera el espacio reservado para v2
            free(v3); // libera el espacio reservado para v3


            return 0;
}
```

Misma estructura que el anterior ejercicio excepto porque es parallel sections.

Repartir el numero de tareas como mínimo a 4 (osea mínimo 4 sections)

#parallel sections

#section

#section

#section

#section

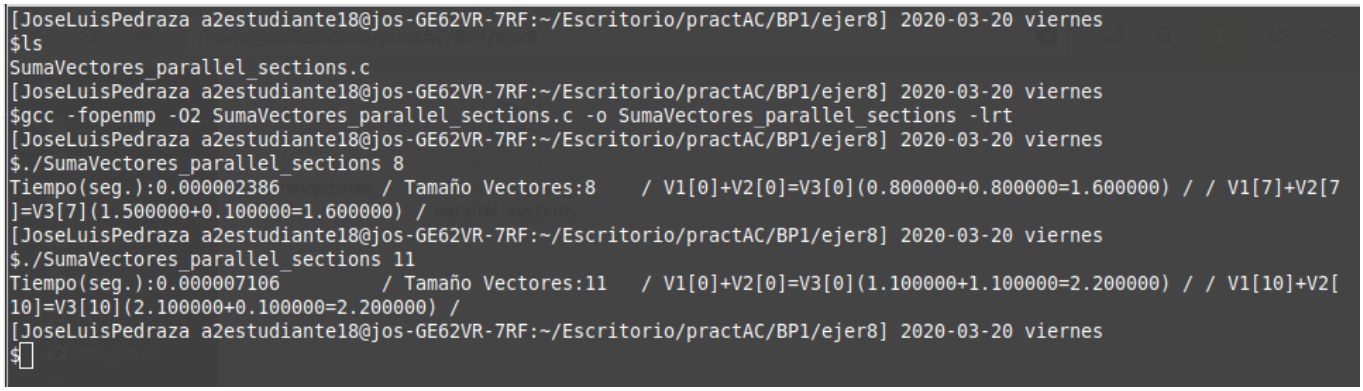
dividir el blucle en 4 partes (0-n/4, n/4-n/2, n/2-3N/4, 3N/4-N)

Recordar!!!!!

podria existir condición de carrera si utilizamos la misma variable i compartida en cada section , lo solucionamos con la clausula parallel sections private(i) o cambiando los indices para cada hebra.

(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:

   Realmente no existiría un límite de threads ni de cores en ninguno de los dos casos. Si podemos considerar el límite máximo útil, en el caso del "parallel-for" dependería del número de iteraciones N que hace el bucle. No sería útil usar un número de threads mayor que N o que el número total de cores. En el caso del "parallel-sections" al fijar el número de sections a 4, tampoco sería útil usar un numero mayor que este de threads (en el mejor caso sería usar 4 threads).

10. Rellenar una tabla como la Tabla 2 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.

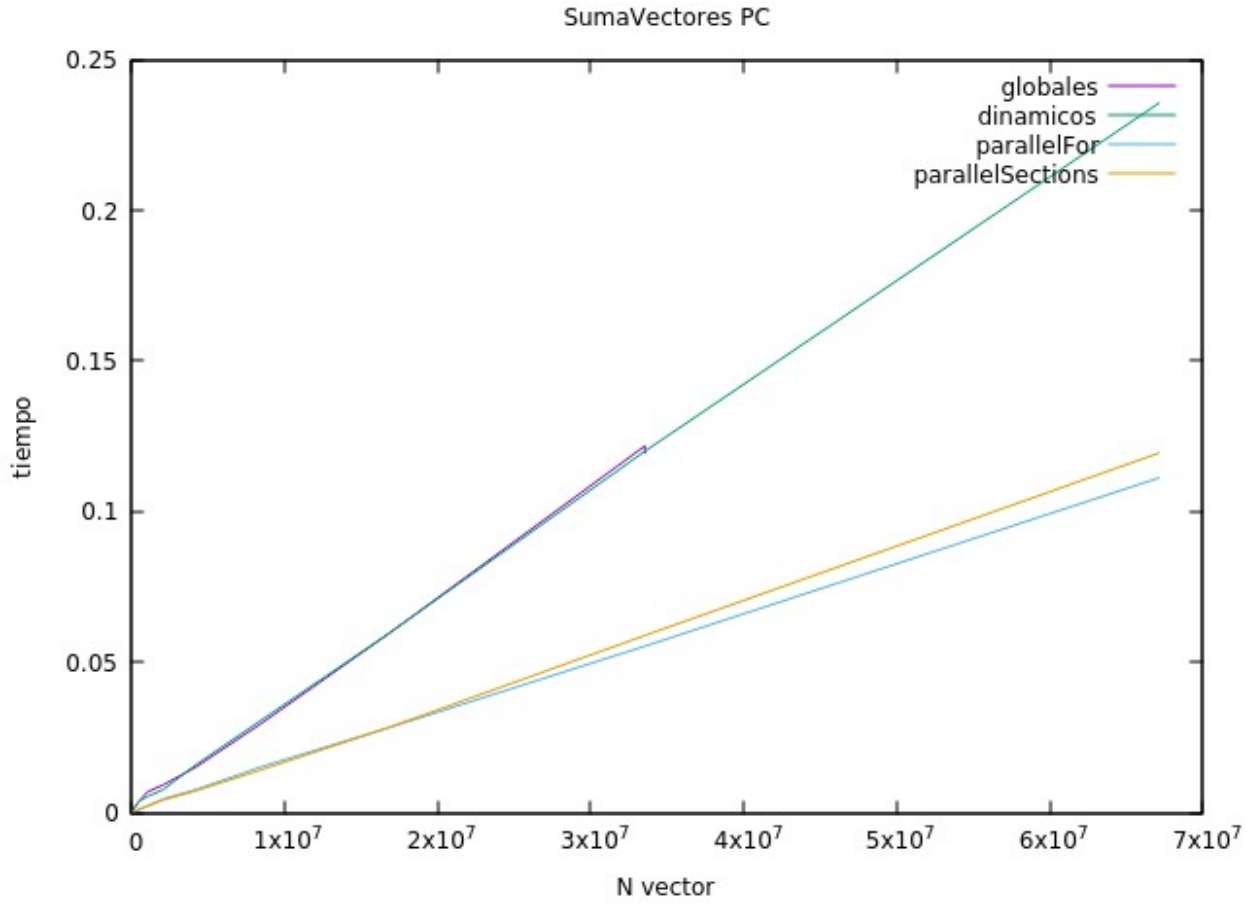
    RESPUESTA:

    -PC:

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.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. secuencial vect. Dinamicos 1 thread/core	T. paralelo (versión `for`) 4 threads/cores	T. paralelo (versión `sections`) 4 threads/cores
16384	0.000121335	0.000120206	0.000032610	0.000051444
32768	0.000242062	0.000286663	0.000096090	0.000102542
65536	0.000472600	0.000501045	0.000118152	0.000318588
131072	0.000926012	0.001197821	0.000176009	0.000303087
262144	0.001978174	0.002049762	0.000463533	0.000508991
524288	0.003713204	0.003862543	0.000871513	0.001199924
1048576	0.006837743	0.005303155	0.002135288	0.002108884
2097152	0.009177675	0.007598233	0.004517402	0.004225286
4194304	0.014988892	0.015703095	0.007590048	0.007171591
8388608	0.029237130	0.030333538	0.015009493	0.014011632
16777216	0.059237476	0.059343152	0.028090356	0.028180349
33554432	0.121720940	0.119970502	0.055225890	0.058827540
67108864		0.235537776	0.111061894	0.119326381

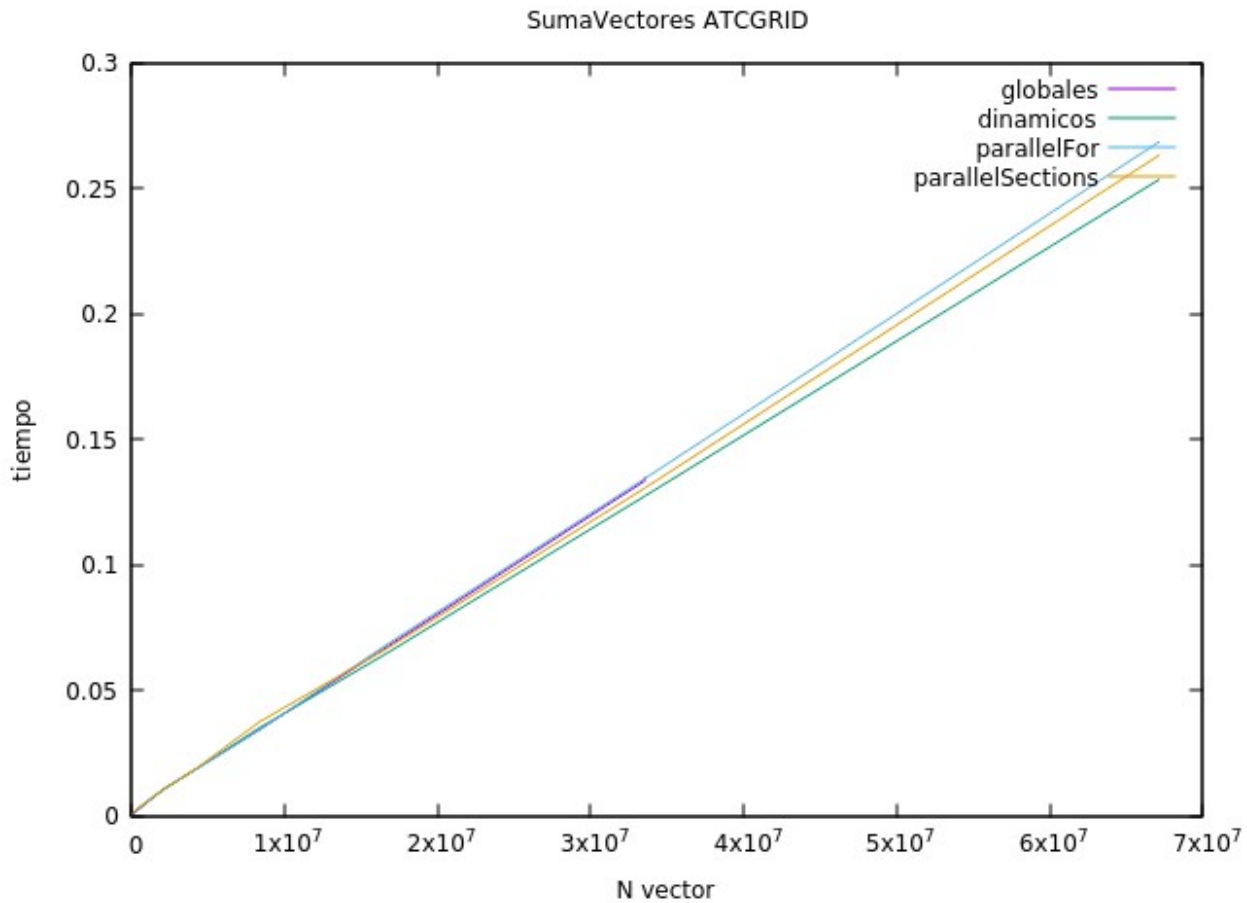
El ultimo tamaño para globales no lo hace por el limite de tamaño (repite el anterior, 3355… con tiempo 0.110725241 por esto lo borro)


SumaVectores PC
globales
dinamicos
parallelFor
parallelSections
tiempo
N vector
0.25
0.2
0.15
0.1
0.05
0
0
1x10^7
2x10^7
3x10^7
4x10^7
5x10^7
6x10^7
7x10^7

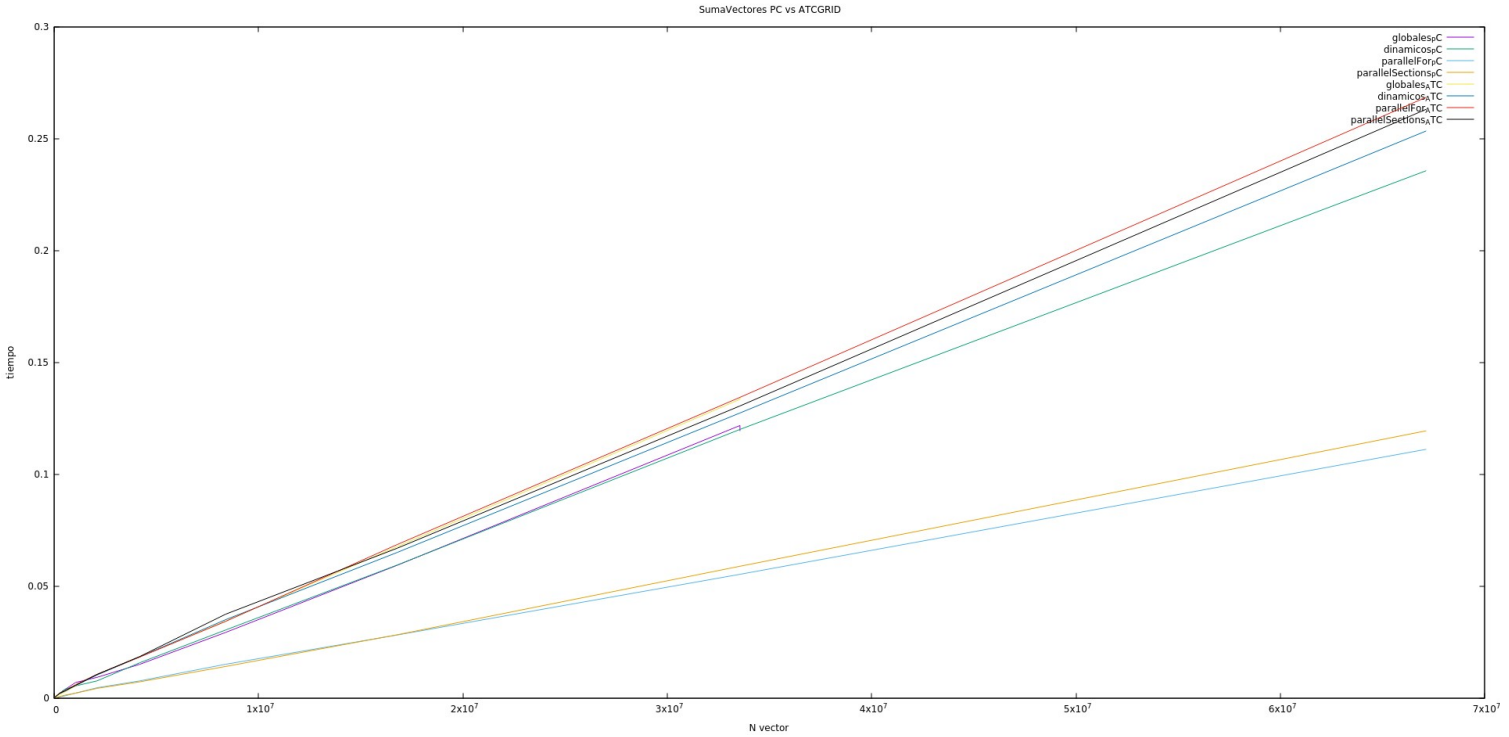
-ATCGRID:

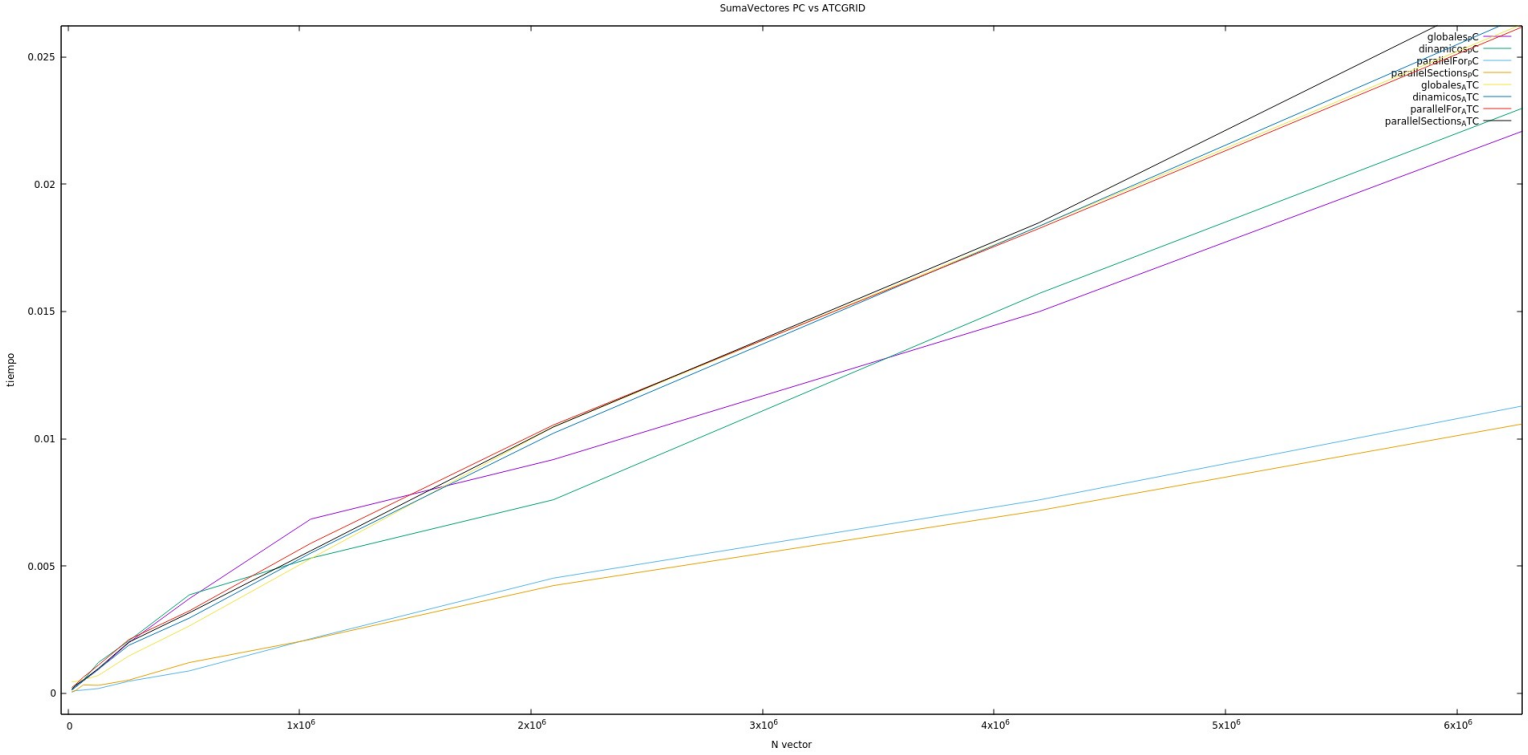
Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos del computador que como máximo puede aprovechar el código.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. secuencial vect. Dinamicos 1 thread/core	T. paralelo (versión `for`) 12 threads/cores	T. paralelo (versión `sections`) 4 threads/cores
16384	0.000442552	0.000117315	0.000218660	0.000151256
32768	0.000451356	0.000237032	0.000336312	0.000313343
65536	0.000525760	0.000476262	0.000619119	0.000510037
131072	0.000690704	0.000933258	0.001093803	0.000965107
262144	0.001454796	0.001878188	0.002101824	0.001996884
524288	0.002637257	0.002947555	0.003230212	0.003155904
1048576	0.005282412	0.005500491	0.005885677	0.005594173
2097152	0.010461592	0.010214040	0.010536956	0.010458514
4194304	0.018345022	0.018346566	0.018259874	0.018487521
8388608	0.034309161	0.034937369	0.034175912	0.037385233
16777216	0.067565789	0.064980680	0.068495795	0.066860136
33554432	0.133452062	0.127338278	0.134306267	0.130467681
67108864		0.253301830	0.268395569	0.262991346



-ATCGRIDvsPC





11. Rellenar una tabla como la 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. Dinamicos 1 thread/core**			**Tiempo paralelo/versión `for` 12 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0m0.090s	0m0.008s	0m0.010s	0m0.097s	0m0.009s	0m0.009s
131072	0m0.090s	0m0.009s	0m0.009s	0m0.101s	0m0.011s	0m0.007s
262144	0m0.092s	0m0.009s	0m0.009s	0m0.121s	0m0.009s	0m0.008s
524288	0m0.085s	0m0.009s	0m0.010s	0m0.120s	0m0.009s	0m0.009s
1048576	0m0.105s	0m0.010s	0m0.008s	0m0.135s	0m0.012s	0m0.006s
2097152	0m0.112s	0m0.013s	0m0.005s	0m0.132s	0m0.009s	0m0.010s
4194304	0m0.110s	0m0.008s	0m0.010s	0m0.135s	0m0.007s	0m0.011s
8388608	0m0.168s	0m0.008s	0m0.011s	0m0.317s	0m0.011s	0m0.008s
16777216	0m0.246s	0m0.012s	0m0.007s	0m0.229s	0m0.009s	0m0.009s
33554432	0m0.415s	0m0.007s	0m0.012s	0m0.387s	0m0.006s	0m0.012s
67108864	0m0.689s	0m0.008s	0m0.010s	0m0.739s	0m0.008s	0m0.011s

La suma del tiempo de CPU del usuario + el tiempo de CPU del sistema es menor que el tiempo real.