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): Nikita Stetskiy
Grupo de prácticas y profesor de prácticas: C3
Fecha de entrega: 20-03-20
Fecha evaluación en clase: 20-03-20

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

  {
    #pragma omp 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 section
    {
        #pragma omp 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 `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
    {
    printf("Depués de la región parallel:\n");
    for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
    printf("\n");

    printf("Identificador de thread: %d\n", omp_get_thread_num());
    }
}
}
```

CAPTURAS DE PANTALLA:

```
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer1]
2020-03-23 lunes
$./singleModificado
Introduce valor de inicialización a: 9
Single ejecutada por el thread 0
Depués de la región parallel:
b[0] = 9        b[1] = 9        b[2] = 9        b[3] = 9        b[4] = 9        b[5] = 9        b[6]
 = 9    b[7] = 9        b[8] = 9
Identificador de thread: 2
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer1]
2020-03-23 lunes
```

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
    {
    printf("Depués de la región parallel:\n");
    for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
    printf("\n");

    printf("Identificador de thread: %d\n", omp_get_thread_num());
    }
}
}
```

CAPTURAS DE PANTALLA:

```
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer1]
2020-03-23 lunes
$./singleModificado
Introduce valor de inicialización a: 9
Single ejecutada por el thread 1
Depués de la región parallel:
b[0] = 9        b[1] = 9        b[2] = 9        b[3] = 9        b[4] = 9        b[5] = 9        b[6]
 = 9    b[7] = 9        b[8] = 9
Identificador de thread: 0
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer1]
2020-03-23 lunes
```

RESPUESTA A LA PREGUNTA:

Al ser master, la hebra que se encarga de realizar ese proceso va a ser siempre la hebra master, es decir la ultima parte tendrá siempre el identificador 0 ya que es la hebra 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:

Sin barrera no hay ningún tipo de espera para las hebras, por lo que se pueden solapar las soluciones. Es decir, gracias a la barrare se espera a que se realicen todas las sumas antes del printf.

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:

```
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer5]
2020-03-23 lunes
$time ./listado 10000000
Tiempo(seg.):0.036574000         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1
000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=200000
0.000000) /

real    0m0.122s
user    0m0.060s
sys     0m0.059s
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer5]
2020-03-23 lunes
```

No es el mismo ya que hay programas de fondo o esperando a ser ejecutados. Es decir, hay tiempo asociado a la esperas de I/O ó la ejecución de otros programas.

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):

```
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer5]
2020-03-23 lunes
$time ./listado 10
Tiempo(seg.):0.000001000         / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.000000+1.000000=2.0000
00) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /

real    0m0.008s
user    0m0.002s
sys     0m0.003s
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer5]
2020-03-23 lunes
```

```
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer5]
2020-03-23 lunes
$g++-8 -fopenmp -O2 listado.c -S listado
g++-8: warning: listado: linker input file unused because linking not done
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer5]
2020-03-23 lunes
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

10 COMPONENTES:

MIPS = 5 / (0,000001000 *10^6) = 5
MFLOPS = 3 / (0,000001000 *10^6) = 3

10000000 COMPONENTES:

MIPS = 5 / (0,036574000 * 10^6) = 0.00013670913

MFLOPS = 3 / (0,036574000 *10^6) = 0.00008202548

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

```
                call    _clock_gettime
                xorl    %eax, %eax
                .align 4
L7:
                movsd   (%r12,%rax,8), %xmm0
                addsd   0(%r13,%rax,8), %xmm0
                movsd   %xmm0, (%r14,%rax,8)
                addq    $1, %rax
                cmpl    %eax, %ebp
                ja      L7
                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,…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

```c
/* SumaVectoresC.c
 Suma de dos vectores: v3 = v1 + v2

 Para compilar usar (-lrt: real time library, no todas las versiones de gcc necesitan que se incluya -lrt):
      gcc  -O2 SumaVectores.c -o SumaVectores -lrt
      gcc  -O2 -S SumaVectores.c -lrt   //para generar el código ensamblador

 Para ejecutar use: SumaVectoresC longitud
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>     // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>
//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL    // descomentar para que los vectores sean variables ...
                          // locales (si se supera el tamaño de la pila se ...
                          // generará el error "Violación de Segmento")
#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
                          // globales (su longitud no estará limitada por el ...
                          // tamaño de la pila del programa)
//#define VECTOR_DYNAMIC    // descomentar para que los vectores sean variables ...
                          // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432         //=2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif

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

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

  //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]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  #ifdef VECTOR_LOCAL
  double v1[N], v2[N], v3[N];   // Tamaño variable local en tiempo de ejecución ...
                                // disponible en C a partir de actualización 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)); //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);
  }
  #endif

  //Inicializar vectores
  #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
  }

  cgt1=omp_get_wtime();
  //Calcular suma de vectores
  #pragma omp parallel for
  for(i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];

  cgt2=omp_get_wtime();
  ncgt=cgt2-cgt1;

  //Imprimir resultado de la suma y el tiempo de ejecución
  if (N<10) {
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%lu\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]);


  #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):

```
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer7] 2020-03-
23 lunes
$g++-8 -fopenmp -O2 Listado1.c -o Listado1
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer7] 2020-03-
23 lunes
$time ./Listado1 8
Tiempo(seg.):0.000054000         / 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) /

real    0m0.008s
user    0m0.003s
sys     0m0.003s
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer7] 2020-03-
23 lunes
$time ./Listado1 11
Tiempo(seg.):0.000086000         / 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) /

real    0m0.008s
user    0m0.003s
sys     0m0.003s
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer7] 2020-03-
23 lunes
```

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

```c
/* SumaVectoresC.c
 Suma de dos vectores: v3 = v1 + v2

 Para compilar usar (-lrt: real time library, no todas las versiones de gcc necesitan que se incluya -lrt):
      gcc  -O2 SumaVectores.c -o SumaVectores -lrt
      gcc  -O2 -S SumaVectores.c -lrt   //para generar el código ensamblador

 Para ejecutar use: SumaVectoresC longitud
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>     // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>
//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL    // descomentar para que los vectores sean variables ...
                          // locales (si se supera el tamaño de la pila se ...
                          // generará el error "Violación de Segmento")
#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
                          // globales (su longitud no estará limitada por el ...
                          // tamaño de la pila del programa)
//#define VECTOR_DYNAMIC    // descomentar para que los vectores sean variables ...
                          // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432         //=2^25
#define SECTIONS 4
double v1[MAX], v2[MAX], v3[MAX];
#endif

void funcion(int num, double *v1, double *v2, double *v3, int N) {
  int i;
  for(i=num; i<N; i=i+SECTIONS)
    v3[i] = v1[i] + v2[i];
}


void inicializar1(int N, double *v1){
  int i;
  for(i=0; i<N; i++){
   v1[i] = N*0.1-i*0.1;
  }
}

void inicializar2(int N, double *v2){
  int i;
  for(i=0; i<N; i++){
   v2[i] = N*0.1-i*0.1;
  }
}

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

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

  //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]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  #ifdef VECTOR_LOCAL
  double v1[N], v2[N], v3[N];   // Tamaño variable local en tiempo de ejecución ...
                                // disponible en C a partir de actualización 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)); //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);
  }
  #endif

  // Inicializar vectores
  #pragma omp parallel sections
  {
    #pragma omp section
     (void)inicializar1(N,v1);
    #pragma omp section
     (void)inicializar2(N,v2);
  }

  cgt1=omp_get_wtime();
    #pragma omp parallel sections
    {
    #pragma omp section
      (void)funcion(0,v1,v2,v3,N);
    #pragma omp section
      (void)funcion(1,v1,v2,v3,N);
    #pragma omp section
      (void)funcion(2,v1,v2,v3,N);
    #pragma omp section
      (void)funcion(3,v1,v2,v3,N);
    }
  cgt2=omp_get_wtime();
  ncgt=cgt2-cgt1;

  //Imprimir resultado de la suma y el tiempo de ejecución
  if (N<10) {
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%lu\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]);


  #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):

```
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer8] 2020-03-
23 lunes
$g++-8 -fopenmp -O2 Listado2.c -o Listado2
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer8] 2020-03-
23 lunes
$time ./Listado2 8
Tiempo(seg.):0.000038000         / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.700000+0.700000=1.400000) /
/ V1[2]+V2[2]=V3[2](0.600000+0.600000=1.200000) /
/ V1[3]+V2[3]=V3[3](0.500000+0.500000=1.000000) /
/ V1[4]+V2[4]=V3[4](0.400000+0.400000=0.800000) /
/ V1[5]+V2[5]=V3[5](0.300000+0.300000=0.600000) /
/ V1[6]+V2[6]=V3[6](0.200000+0.200000=0.400000) /
/ V1[7]+V2[7]=V3[7](0.100000+0.100000=0.200000) /

real    0m0.007s
user    0m0.002s
sys     0m0.004s
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer8] 2020-03-
23 lunes
$time ./Listado2 11
Tiempo(seg.):0.000039000         / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / /
V1[10]+V2[10]=V3[10](0.100000+0.100000=0.200000) /

real    0m0.007s
user    0m0.002s
sys     0m0.004s
[NikitaStetskiy nikitastetskiy@MacBook-Pro-de-Nikita:~/Desktop/COPY/nuevo_testamento1/AC/bp1/ejer8] 2020-03-
23 lunes
```

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:

No existe un límite fijo de hebras o cores en estos casos. Aunque es absurdo utilizar más hebras o cores que el número de iteraciones del ejercicio 7 o sections del ejercicio 8. Es decir, en el ejercicio 7 depende de las iteraciones que haga el bucle. En el ejercicio 8, al fijar el sections, lo ideal seria usar el mismo numero de hebras que sections.

10. Rellenar una tabla como la Tabla 210 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:

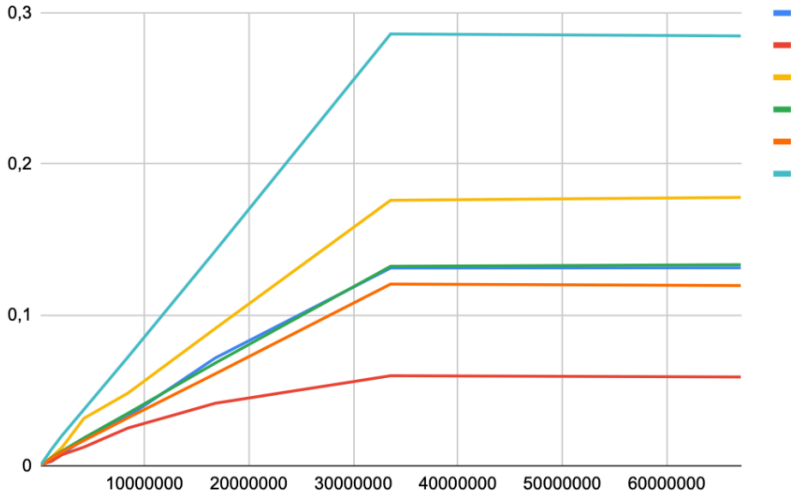
PC LOCAL

Nº de Componentes	T. secuencial vect. Globales	T. paralelo (versión for)	T. paralelo (versión sections)
	1 thread/core	4 threads/cores	4 threads/cores
65536	0.000291000	0.000286000	0.000481000
131072	0.000465000	0.000414000	0.000801000
262144	0.001068000	0.000747000	0.001501000
524288	0.002226000	0.001861000	0.002581000
1048576	0.003825000	0.002902000	0.005553000
2097152	0.007903000	0.007380000	0.012305000
4194304	0.017503000	0.012428000	0.031494000
8388608	0.032614000	0.024994000	0.048104000
16777216	0.071493000	0.041464000	0.091076000
33554432	0.131049000	0.059662000	0.175933000
67108864	0.131188000	0.058844000	0.177834000

ATCGRID

Nº de Componentes	T. secuencial vect. Globales	T. paralelo (versión for)	T. paralelo (versión sections)
	1 thread/core	24 threads/cores	24 threads/cores
65536	0.000484945	0.000698647	0.001080483
131072	0.000702447	0.000554038	0.001354463
262144	0.001397619	0.001543926	0.002733272
524288	0.002666679	0.002492901	0.005324172
1048576	0.005410149	0.004997749	0.010632569
2097152	0.009843726	0.009283407	0.020177528
4194304	0.018503611	0.016791804	0.037392203
8388608	0.034761419	0.031746577	0.071949951
16777216	0.068127978	0.061007543	0.142545601
33554432	0.132191383	0.120399320	0.286104102
67108864	0.133233851	0.119411185	0.284801764

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.



Azul – PC LOCAL V. GLOBAL
Rojo – PC LOCAL V. FOR
Amarillo – PC LOCAL V. SECTIONS
Verde – ATCGRID V. GLOBAL
Naranja – ATCGRID V. FOR
Azul claro – ATCGRID V. SECTIONS

11. Rellenar una tabla como la 11Tabla 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	0m0.045s	0m0.000s	0m0.003s	0m0.054s	0m0.005s	0m0.003s
131072	0m0.056s	0m0.000s	0m0.004s	0m0.076s	0m0.005s	0m0.004s
262144	0m0.089s	0m0.001s	0m0.005s	0m0.078s	0m0.006s	0m0.007s
524288	0m0.089s	0m0.005s	0m0.004s	0m0.078s	0m0.008s	0m0.010s
1048576	0m0.067s	0m0.010s	0m0.005s	0m0.100s	0m0.021s	0m0.009s
2097152	0m0.097s	0m0.017s	0m0.011s	0m0.096s	0m0.031s	0m0.016s
4194304	0m0.107s	0m0.026s	0m0.026s	0m0.114s	0m0.053s	0m0.031s
8388608	0m0.115s	0m0.054s	0m0.041s	0m0.100s	0m0.100s	0m0.055s
16777216	0m0.190s	0m0.097s	0m0.072s	0m0.157s	0m0.160s	0m0.114s
33554432	0m0.372s	0m0.168s	0m0.149s	0m0.293s	0m0.337s	0m0.176s
67108864	0m0.359s	0m0.149s	0m0.163s	0m0.298s	0m0.330s	0m0.196s

Al igual que en el ejercicio anterior, no es el mismo ya que hay programas de fondo o esperando a ser ejecutados. Es decir, hay tiempo asociado a la esperas de I/O ó la ejecución de otros programas.