

2º curso / 2º cuatr.  
Grado Ing. Inform.  
Doble Grado Ing.  
Inform. y Mat.

## Arquitectura de Computadores (AC)

### Cuaderno de prácticas.

### Bloque Práctico 1. Programación paralela I: Directivas OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

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

**RESPUESTA:** código fuente `bucle-forModificado.c`

```
#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`

```
#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();
    }
    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:** código fuente `singleModificado.c`

```
#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] = 100;
        #pragma omp single
        {
            printf("Dentro de la región parallel:\n");
            for (i=0; i<n; i++)
                printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
            printf("Single ejecutada por el thread %d, (MASTER)\n",omp_get_thread_num());
        }
    }
    return 0;
}
```

#### CAPTURAS DE PANTALLA:

```
rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC/practica2$ ./singleModificado
Introduce valor de inicialización a: Single ejecutada por el thread 2
Dentro de la región parallel:
b[0] = 100    b[1] = 100    b[2] = 100    b[3] = 100    b[4] = 100    b[5] = 100
100
Single ejecutada por el thread 1
```

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`

```
#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] = 100;
        #pragma omp master
        {
            printf("Dentro de la región parallel:\n");
            for (i=0; i<n; i++)
                printf("b[%d] = %d\t", i, b[i]);
            printf("\n");
            printf("Single ejecutada por el thread %d, (MASTER)\n", omp_get_thread_num());
        }
    }
    return 0;
}
```

#### CAPTURAS DE PANTALLA:

```
rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC/practica2$ ./singleMaster
Introduce valor de inicialización a: Single ejecutada por el thread 2
Dentro de la región parallel:
b[0] = 100    b[1] = 100    b[2] = 100    b[3] = 100    b[4] = 100    b[5] = 100    b[6] = 100
100
Single ejecutada por el thread 0, (MASTER)
```

#### RESPUESTA A LA PREGUNTA:

La hebra encargada de ejecutar la sección `master` es siempre la hebra Master (la 0)

4. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

**RESPUESTA:** Porque la directiva `master` no tiene una barrera implícita al final, y por tanto para evitar condiciones de carrera hay que emplear una barrera explícitamente con `barrier`.

### 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 Error: Reference source not found calcula la suma de dos vectores ( $v3 = v1 + v2$ ;  $v3(i) = v1(i) + v2(i)$ ,  $i=0, \dots, N-1$ ). Generar el ejecutable del programa del Error: Reference source not found 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:

##### En el PC Local

```
rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC$ time `./SumaVectores 10000000`
Tiempo(seg.):0.079496976: no se encontró la orden

real    0m0.353s
user    0m0.140s
sys     0m0.204s
```

##### En Atcgrid:

```
[E1estudiante16@atcgrid ~]$ echo 'time `./SumaVectores 10000000`' | qsub -q ac
34610.atcgrid
[E1estudiante16@atcgrid ~]$ cat STDIN.e34610
/var/spool/torque/mom_priv/jobs/34610.atcgrid.SC: línea 1: Tiempo(seg.):0.037031259: no se encontró la
real    0m0.120s
user    0m0.062s
sys     0m0.051s
[E1estudiante16@atcgrid ~]$
```

6. Generar el código ensamblador a partir del programa secuencial C del Error: Reference source not found para **vectores globales** (para generar el código ensamblador tiene que compilar usando `-s` en lugar de `-o`). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of FLOating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Incorpore el código ensamblador de la parte de la suma de vectores en el cuaderno.

**CAPTURAS DE PANTALLA:**

```

call    clock_gettime
xorl    %eax, %eax
.p2align 4,,10
.p2align 3
.L9:
movsd   0(%rbp,%rax,8), %xmm0
addsd   (%r12,%rax,8), %xmm0
movsd   %xmm0, 0(%r13,%rax,8)
addq    $1, %rax
cmpl    %eax, %ebx
ja      .L9
.L10:
leaq    16(%rsp), %rsi
xorl    %edi, %edi
call    clock_gettime

```

**RESPUESTA:**

El bloque L9 es el bucle for. Como tiene 5 instrucciones que se ejecutan en cada ciclo y el programa tarda 0.62seg en completarse para N=10000000 tenemos que atcgrid da una potencia máxima de aproximadamente  $50000000 / (0.071154736 * 10^6) = 702.69$  MIPS

Para N=10:  $50 / (0.000008656 * 10^6) = 5.77$  MIPS\*

Como solo una instruccion es de tipo float (las movsd) tenemos que atcgrid da una potencia máxima de aproximadamente  $10000000 / (0.071154736 * 10^6) = 140.53$  MFLOPS

Para N=10:  $10 / (0.000008656 * 10^6) = 1.15$  MFLOPS\*

*\*(Deberian tenerse en cuenta todas las instrucciones del programa y no solo el bucle)*

**RESPUESTA:** código ensamblador generado de la parte de la suma de vectores

```

.file      "SumaVectoresC.c"
.section   .rodata.str1.8,"aMS",@progbits,1
.align    8
.LC0:
.string    "Faltan n\302\272 componentes del vector"
.align    8
.LC1:
.string    "Error en la reserva de espacio para los vectores"
.align    8
.LC4:
.string    "Tiempo(seg.):%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"
.section   .text.startup,"ax",@progbits

```

```

        .p2align 4,,15
        .globl      main
        .type       main, @function

main:
.LFB18:

        .cfi_startproc
        pushq       %r14
        .cfi_def_cfa_offset 16
        .cfi_offset 14, -16
        pushq       %r13
        .cfi_def_cfa_offset 24
        .cfi_offset 13, -24
        pushq       %r12
        .cfi_def_cfa_offset 32
        .cfi_offset 12, -32
        pushq       %rbp
        .cfi_def_cfa_offset 40
        .cfi_offset 6, -40
        pushq       %rbx
        .cfi_def_cfa_offset 48
        .cfi_offset 3, -48
        subq        $32, %rsp
        .cfi_def_cfa_offset 80
        cmpl        $1, %edi
        jle         .L15
        movq        8(%rsi), %rdi
        movl        $10, %edx
        xorl        %esi, %esi
        call        strtol
        movl        %eax, %r14d
        movl        %eax, %ebx
        leaq        0(,%r14,8), %r13
        movq        %r13, %rdi
        call        malloc
        movq        %r13, %rdi
        movq        %rax, %rbp
        call        malloc
        movq        %r13, %rdi
        movq        %rax, %r12
        call        malloc
        testq       %r12, %r12
        movq        %rax, %r13
        je          .L3
        testq       %rbp, %rbp
        je          .L3
        testq       %rax, %rax
        je          .L3
        testl       %ebx, %ebx
        .p2align 4,,3
        je          .L16
        cvtsi2sdq   %r14, %xmm1
        xorl        %eax, %eax
        movsd       .LC2(%rip), %xmm3
        mulsd       %xmm3, %xmm1
        .p2align 4,,10
        .p2align 3

.L8:
        cvtsi2sd    %eax, %xmm0
        mulsd       %xmm3, %xmm0
        movapd      %xmm0, %xmm2
        addsd       %xmm1, %xmm2
        movsd       %xmm2, 0(%rbp,%rax,8)

```

```

movapd    %xmm1, %xmm2
subsd     %xmm0, %xmm2
movsd     %xmm2, (%r12,%rax,8)
addq      $1, %rax
cmpl      %eax, %ebx
ja        .L8
movq      %rsp, %rsi
xorl      %edi, %edi
call      clock_gettime
xorl      %eax, %eax
.p2align 4,,10
.p2align 3
.L9:
movsd     0(%rbp,%rax,8), %xmm0
addsd     (%r12,%rax,8), %xmm0
movsd     %xmm0, 0(%r13,%rax,8)
addq      $1, %rax
cmpl      %eax, %ebx
ja        .L9
.L10:
leaq      16(%rsp), %rsi
xorl      %edi, %edi
call      clock_gettime
movq      16(%rsp), %rcx
subq      (%rsp), %rcx
leal      -1(%rbx), %edx
movsd     0(%r13), %xmm3
movl      %ebx, %esi
movl      %edx, %eax
movsd     (%r12), %xmm2
movsd     0(%r13,%rax,8), %xmm6
movl      %edx, %r8d
cvtsi2sdq %rcx, %xmm0
movq      24(%rsp), %rcx
subq      8(%rsp), %rcx
movsd     (%r12,%rax,8), %xmm5
movsd     0(%rbp,%rax,8), %xmm4
movl      $.LC4, %edi
movl      $7, %eax
cvtsi2sdq %rcx, %xmm1
movl      %edx, %ecx
divsd     .LC3(%rip), %xmm1
addsd     %xmm1, %xmm0
movsd     0(%rbp), %xmm1
call      printf
movq      %rbp, %rdi
call      free
movq      %r12, %rdi
call      free
movq      %r13, %rdi
call      free
addq      $32, %rsp
.cfi_remember_state
.cfi_def_cfa_offset 48
xorl      %eax, %eax
popq      %rbx
.cfi_def_cfa_offset 40
popq      %rbp
.cfi_def_cfa_offset 32
popq      %r12
.cfi_def_cfa_offset 24
popq      %r13

```

```
.cfi_def_cfa_offset 16
popq    %r14
.cfi_def_cfa_offset 8
ret

.L16:
.cfi_restore_state
movq    %rsp, %rsi
xorl    %edi, %edi
call    clock_gettime
jmp     .L10

.L3:
movl    $.LC1, %edi
call    puts
movl    $-2, %edi
call    exit

.L15:
movl    $.LC0, %edi
call    puts
orl     $-1, %edi
call    exit
.cfi_endproc

.LFE18:
.size   main, .-main
.section .rodata.cst8,"aM",@progbits,8
.align 8

.LC2:
.long   2576980378
.long   1069128089
.align 8

.LC3:
.long   0
.long   1104006501
.ident  "GCC: (GNU) 4.6.3 20120306 (Red Hat 4.6.3-2)"
.section .note.GNU-stack,"",@progbits
```



7. Implementar un programa en C con OpenMP, a partir del código del Error: Reference source not found, que calcule en paralelo la suma de dos vectores ( $v3 = v1 + v2$ ;  $v3(i)=v1(i) + v2(i)$ ,  $i=0, \dots, N-1$ ) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Error: Reference source not found 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**

```

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

Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt

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>         // biblioteca openmp

#define VECTOR_GLOBAL
#ifdef VECTOR_GLOBAL
#define MAX 33554432     //=2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char** argv){
    int i;
    //Leer argumento de entrada (nº de componentes del vector)
    if (argc<2){
        printf("Faltan nº componentes del vector\n");
        exit(-1);
    }
    //printf("Numero de hebras: %d\n", omp_get_max_threads());
    unsigned int N = atoi(argv[1]);
    printf("N: %s = N:%d\n", argv[1], N);
#ifdef VECTOR_GLOBAL
    if (N>MAX) N=MAX;
#endif

```

```

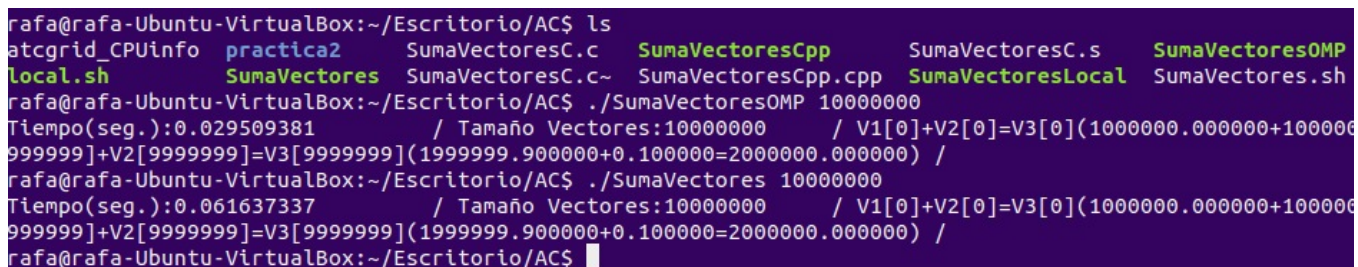
#pragma omp parallel
{
    #pragma omp for
    for(i=0; i<N; i++){
        v1[i] = N*0.1+i*0.1;
        v2[i] = N*0.1-i*0.1;
    }
}
double start = omp_get_wtime();
//Calcular suma de vectores tambien en 4 hebras aprovechando la arquitectura del PC

#pragma omp parallel
{
    #pragma omp for
    for(i=0; i<N; i++){
        v3[i] = v1[i] + v2[i];
    }
}
double end = omp_get_wtime();
double time = end - start;

//Imprimir resultado de la suma y el tiempo de ejecución
printf("Tiempo(seg.):%11.9f\n / Tamaño Vectores:%u\n/ V1[0]+V2[0]=V3[0] (%8.6f+
%8.6f=%8.6f)\n V1[%d]+V2[%d]=V3[%d] (%8.6f+%8.6f=%8.6f) /\n",
time,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

if (N < 50){
    for(i=0; i<N; i++){
        printf("%f + %f = %f\n",v1[i], v2[i], v3[i]);
    }
}
return 0;
}

```

**CAPTURAS DE PANTALLA:**


```

rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC$ ls
atcgrid_CPUinfo practica2 SumaVectoresC.c SumaVectoresCpp SumaVectoresC.s SumaVectoresOMP
local.sh SumaVectores SumaVectoresC.c~ SumaVectoresCpp.cpp SumaVectoresLocal SumaVectores.sh
rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC$ ./SumaVectoresOMP 1000000
Tiempo(seg.):0.029509381 / Tamaño Vectores:1000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000)=V3[999999](1999999.900000+0.100000=2000000.000000) /
rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC$ ./SumaVectores 1000000
Tiempo(seg.):0.061637337 / Tamaño Vectores:1000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000)=V3[999999](1999999.900000+0.100000=2000000.000000) /
rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC$

```

8. Implementar un programa en C con OpenMP, a partir del código del Error: Reference source not found, 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

```

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

Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt

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>     // biblioteca openmp

#define VECTOR_GLOBAL

#ifdef VECTOR_GLOBAL
#define MAX 33554432 // = 2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char** argv){
    //Indices para cada hebra
    int i, j, k, l;
    //Leer argumento de entrada (nº de componentes del vector)
    if (argc<2){
        printf("Faltan nº componentes del vector\n");
        exit(-1);
    }
    printf("Numero de hebras: %d\n", omp_get_max_threads());

    unsigned int N = atoi(argv[1]);
    printf("N: %s = N:%d\n", argv[1], N);
    #ifdef VECTOR_GLOBAL
    if (N>MAX) N=MAX;
    #endif

    printf("Inicializacion:\n");
    //Inicializar vectores en cuatro partes (Aprovechando 4 hebras)

```

```

#pragma omp parallel sections
{
    #pragma omp section
    {
        for(i=0; i<N/2; i++){
            v1[i] = N*0.1+i*0.1;
            //printf("v1[%d] - Hebra: %d: %f\n",i, omp_get_thread_num(), v1[i]);
        }
    }
    #pragma omp section
    {
        for(j=0; j<N/2; j++){
            v2[j] = N*0.1-j*0.1;
            //printf("v2[%d] - Hebra: %d: %f\n",j, omp_get_thread_num(), v2[j]);
        }
    }
    #pragma omp section
    {
        for(k=N/2; k<N; k++){
            v1[k] = N*0.1+k*0.1;
            //printf("v1[%d] - Hebra: %d: %f\n",k, omp_get_thread_num(), v1[k]);
        }
    }
    #pragma omp section
    {
        for(l=N/2; l<N; l++){
            v2[l] = N*0.1-l*0.1;
            //printf("v2[%d] - Hebra: %d: %f\n",l, omp_get_thread_num(), v2[l]);
        }
    }
}
//#pragma omp barrier No hace falta ponerla porque sections tiene barrera implicita

double start = omp_get_wtime();
//Calcular suma de vectores tambien en 4 hebras aprovechando la arquitectura del PC

printf("Suma:\n");
#pragma omp parallel sections
{
    #pragma omp section
    {
        for(i=0; i<N/4; i++){
            v3[i] = v1[i] + v2[i];
        }
    }
    #pragma omp section
    {
        for(j=N/4; j<N/2; j++){
            v3[j] = v1[j] + v2[j];
        }
    }
    #pragma omp section
    {
        for(k=N/2; k<3*N/4; k++){
            v3[k] = v1[k] + v2[k];
        }
    }
    #pragma omp section
    {
        for(l=3*N/4; l<N; l++){
            v3[l] = v1[l] + v2[l];
        }
    }
}

double end = omp_get_wtime();
double time = end - start;

```

```

//Imprimir resultado de la suma y el tiempo de ejecución
printf("Tiempo(seg.):%11.9f\n / Tamaño Vectores:%u\n/ V1[0]+V2[0]=V3[0](%8.6f+
%8.6f=%8.6f)\n V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
time,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

if (N < 50){
    for(i=0; i<N; i++){
        printf("v1[%d] + v2[%d] = v3[%d]\t %f + %f = %f\n",i,i,i, v1[i], v2[i], v3[i]);
    }
}

return 0;
}

```

**CAPTURAS DE PANTALLA:**

```

rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC$ ./SumaVectoresSections 5
Numero de hebras: 4
N: 5 = N:5
Inicializacion:
v1[0] - Hebra: 1: 0.500000
v2[2] - Hebra: 0: 0.300000
v2[3] - Hebra: 0: 0.200000
v2[4] - Hebra: 0: 0.100000
v1[1] - Hebra: 1: 0.600000
v2[0] - Hebra: 2: 0.500000
v2[1] - Hebra: 2: 0.300000
v1[2] - Hebra: 3: 0.700000
v1[3] - Hebra: 3: 0.800000
v1[4] - Hebra: 3: 0.900000
Suma:
v1[0] + v2[0] = v3[0]    0.500000 + 0.500000 = 1.000000
v1[1] + v2[1] = v3[1]    0.600000 + 0.400000 = 1.000000
v1[2] + v2[2] = v3[2]    0.700000 + 0.300000 = 1.000000
v1[3] + v2[3] = v3[3]    0.800000 + 0.200000 = 1.000000
v1[4] + v2[4] = v3[4]    0.900000 + 0.100000 = 1.000000
Tiempo(seg.):0.000022070
/ Tamaño Vectores:5
/ V1[0]+V2[0]=V3[0](0.500000+0.500000=1.000000)
V1[4]+V2[4]=V3[4](0.900000+0.100000=1.000000) /
rafa@rafa-Ubuntu-VirtualBox:~/Escritorio/AC$

```

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:**

Threds puedes utilizar tantas como quieras pero cores solo puedes utilizar los que tengas en tu ordenador (y si estas en una maquina virtual solo puedes usar los que la maquina tenga asignados)

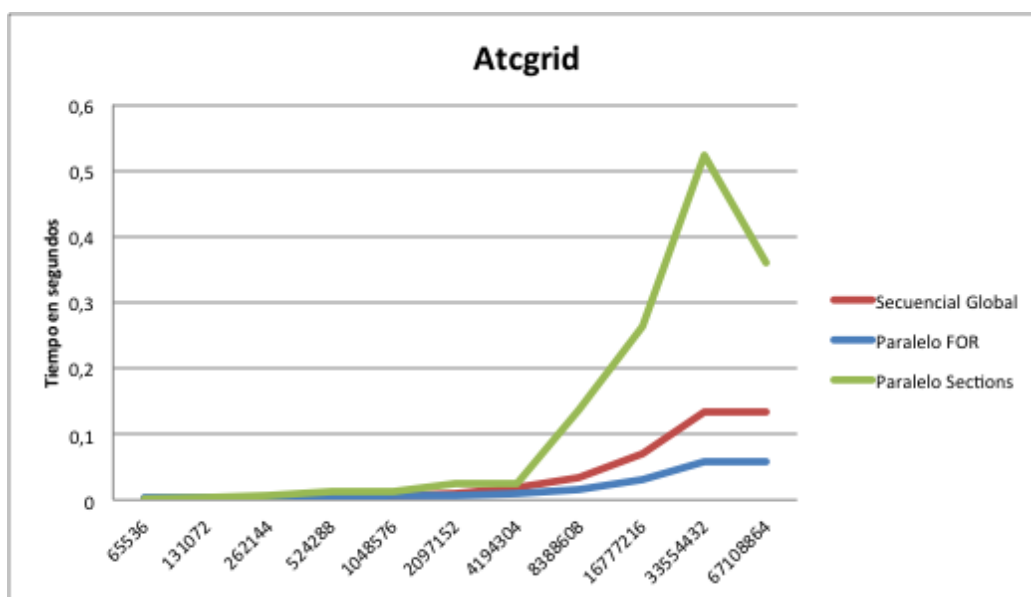
10. Rellenar una tabla como la para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Error: Reference source not found. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de 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:**

**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

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	0.001119930	0.000194996	0.000605384
131072	0.000938974	0.000820776	0.001763912
262144	0.002263957	0.000718527	0.003397638
524288	0.003985906	0.004123709	0.004492471
1048576	0.008259308	0.002313981	0.008681269
2097152	0.012739410	0.007083302	0.018681144
4194304	0.024119238	0.011102808	0.035399192
8388608	0.048427849	0.023746592	0.093761916
16777216	0.105067384	0.047288412	0.148922279
33554432	0.268460655	0.093336444	0.360296630
67108864	0.201799026	0.093858019	0.359798802

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24threads/cores	T. paralelo (versión sections) 24 threads/cores
65536	0.000546472	0.003257220	0.001533863
131072	0.000486419	0.003268032	0.003156201
262144	0.001444043	0.003934552	0.005439613
524288	0.002656301	0.004253729	0.011551572
1048576	0.005286185	0.005563400	0.011709568
2097152	0.008720559	0.007253630	0.025428127
4194304	0.017671053	0.009826105	0.024905960
8388608	0.034306461	0.016426054	0.137164485
16777216	0.068669328	0.031660163	0.262161369
33554432	0.133166682	0.057222648	0.523739028
67108864	0.133720562	0.058756163	0.359933530



11. Rellenar una tabla como la 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 Error: Reference source not found. Ponga en la tabla el número de threads/cores que usan los códigos.

**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 4 Threads/cores		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
65536	0,00	0,00	0,00	0,00	0,00	0,00
131072	0,00	0,00	0,00	0,00	0,00	0,00
262144	0,01	0,00	0,00	0,00	0,00	0,01
524288	0,02	0,01	0,01	0,00	0,00	0,00
1048576	0,05	0,02	0,02	0,01	0,02	0,01
2097152	0,09	0,05	0,04	0,02	0,03	0,05
4194304	0,17	0,08	0,08	0,05	0,07	0,10
8388608	0,33	0,22	0,11	0,07	0,05	0,21
16777216	0,68	0,40	0,27	0,14	0,09	0,41
33554432	1,32	0,90	0,42	0,27	0,24	0,70
67108864	1,31	0,86	0,45	0,27	0,26	0,68

