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 y profesor de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

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

Ejercicios basados en los ejemplos del seminario práctico

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

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

```
#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 iteracion %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());
}


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>
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("Dentro de la región parallel:\n");
    for(i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]), printf("Single ejecutado por el thread
%d\n",
        omp_get_thread_num());;
    printf("\n");
    }
}


}
```

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer2] 2019-03-22 viernes
$gcc -O2 -fopenmp singleModificado.c -o singleModificado
singleModificado.c:3:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main() {
 ^~~~
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer2] 2019-03-22 viernes
$export OMP_DYNAMIC=FALSE
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer2] 2019-03-22 viernes
$export OMP_NUM_THREADS=8
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer2] 2019-03-22 viernes
$./singleModificado
Introduce valor de inicialización a: 23
Single ejecutada por el thread 1
Dentro de la región parallel:
b[0] = 23       Single ejecutado por el thread 5
b[1] = 23       Single ejecutado por el thread 5
b[2] = 23       Single ejecutado por el thread 5
b[3] = 23       Single ejecutado por el thread 5
b[4] = 23       Single ejecutado por el thread 5
b[5] = 23       Single ejecutado por el thread 5
b[6] = 23       Single ejecutado por el thread 5
b[7] = 23       Single ejecutado por el thread 5
b[8] = 23       Single ejecutado por el thread 5

[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer2] 2019-03-22 viernes
$
```

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>
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("Dentro de la región parallel:\n");
    for(i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]), printf("Single
ejecutado por el thread %d\n",
        omp_get_thread_num());;
    printf("\n");
    }
}

}
```

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer3] 2019-04-02 martes
$gcc -O2 -fopenmp singleModificado2.c -o singleModificado2
singleModificado2.c:3:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main() {
 ^~~~
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer3] 2019-04-02 martes
$export OMP_NUM_THREADS=8
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer3] 2019-04-02 martes
$export OMP_DYNAMIC=FALSE
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer3] 2019-04-02 martes
$./singleModificado2
Introduce valor de inicialización a: 23
Single ejecutada por el thread 1
Dentro de la región parallel:
b[0] = 23       Single ejecutado por el thread 0
b[1] = 23       Single ejecutado por el thread 0
b[2] = 23       Single ejecutado por el thread 0
b[3] = 23       Single ejecutado por el thread 0
b[4] = 23       Single ejecutado por el thread 0
b[5] = 23       Single ejecutado por el thread 0
b[6] = 23       Single ejecutado por el thread 0
b[7] = 23       Single ejecutado por el thread 0
b[8] = 23       Single ejecutado por el thread 0

[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer3] 2019-04-02 martes
$
```

RESPUESTA A LA PREGUNTA: La diferencia es que al usar la directiva master, el thread que se va a ejecutar va a ser siempre el primero (por eso, en cada iteración el single es ejecutado por este thread)

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 se eliminaría la barrera implícita al final, haciendo que la hebra master pueda realizar todas las sumas antes de tiempo razón por la cual se pueden generan errores, y por lo tanto la suma puede que sea errónea.

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 dinámicos**. 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:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer5] 2019-04-02 martes
$time ./SumaVectoresC 10000000
Tama◆o Vectores:10000000 (4 B)
Tiempo:0.036406643       / Tama◆o Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+10000
00.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=200000
0.000000) /

real    0m0,121s
user    0m0,047s
sys     0m0,074s
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer5] 2019-04-02 martes
$
```

   RESPUESTA: La suma del tiempo de CPU del usuario es igual al tiempo real, mostrando el tiempo real que ha transcurrido desde que se ha ejecutado el programa hasta que ha acabado

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores dinámicos** (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):

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer6] 2019-04-03 mi
ércoles
$gcc -O2 -S SumaVectoresC.c -lrt
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:32: warning: format ‘%u’ expects argument of type ‘unsigned i
nt’, but argument 3 has type ‘long unsigned int’ [-Wformat=]
   printf("Tama◆o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                            ~^
                            %lu
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer6] 2019-04-03 mi
ércoles
$
```

Código ensamblador de SumaVectoresC:

```
.file         "SumaVectoresC.c"
              .text
              .section     .rodata.str1.8,"aMS",@progbits,1
              .align 8
.LC0:
              .string      "Faltan n\272 componentes del vector"
              .section     .rodata.str1.1,"aMS",@progbits,1
.LC1:
              .string      "Tama\361o Vectores:%u (%u B)\n"
              .section     .rodata.str1.8
              .align 8
.LC2:
              .string      "No hay suficiente espacio para los vectores "
              .align 8
.LC4:
              .string      "/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n"
              .align 8
.LC5:
              .string      "Tiempo:%11.9f\t / Tama\361o Vectores:%u\t/ V1[0]+V2[0]=V3[0]
(%8.6f+%8.6f=%8.6f) / / V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n"
              .align 8
.LC7:
              .string      "Tiempo:%11.9f\t / Tama\361o Vectores:%u\n"
              .section     .text.startup,"ax",@progbits
              .p2align 4,,15
              .globl       main
              .type        main, @function
main:
.LFB41:
              .cfi_startproc
              pushq        %r15
              .cfi_def_cfa_offset 16
              .cfi_offset 15, -16
              pushq        %r14
              .cfi_def_cfa_offset 24
              .cfi_offset 14, -24
              pushq        %r13
              .cfi_def_cfa_offset 32
              .cfi_offset 13, -32
              pushq        %r12
              .cfi_def_cfa_offset 40
              .cfi_offset 12, -40
              pushq        %rbp
              .cfi_def_cfa_offset 48
              .cfi_offset 6, -48
              pushq        %rbx
              .cfi_def_cfa_offset 56
              .cfi_offset 3, -56
              subq         $56, %rsp
              .cfi_def_cfa_offset 112
              movq         %fs:40, %rax
              movq         %rax, 40(%rsp)
              xorl         %eax, %eax
              cmpl         $1, %edi
              jle          .L26
              movq         8(%rsi), %rdi
              movl         $10, %edx
              xorl         %esi, %esi
              call         strtol@PLT
```

```
                movq          %rax, %r13
                leaq          .LC1(%rip), %rsi
                movl          %eax, %edx
                movl          %r13d, %ebx
                movl          $4, %ecx
                movl          $1, %edi
                leaq          0(,%rbx,8), %r12
                xorl          %eax, %eax
                call          __printf_chk@PLT
                movq          %r12, %rdi
                call          malloc@PLT
                movq          %r12, %rdi
                movq          %rax, %rbp
                call          malloc@PLT
                movq          %r12, %rdi
                movq          %rax, %r14
                call          malloc@PLT
                testq         %rbp, %rbp
                movq          %rax, %r12
                je            .L3
                testq         %r14, %r14
                je            .L3
                testl         %r13d, %r13d
                je            .L27
                pxor          %xmm1, %xmm1
                leal          -1(%r13), %r15d
                xorl          %eax, %eax
                movsd         .LC3(%rip), %xmm3
                cvtsi2sdq     %rbx, %xmm1
                movq          %r15, %rbx
                addq          $1, %r15
                mulsd         %xmm3, %xmm1
                .p2align 4,,10
                .p2align 3
.L8:
                pxor          %xmm0, %xmm0
                movapd        %xmm1, %xmm2
                movapd        %xmm1, %xmm7
                cvtsi2sd      %eax, %xmm0
                mulsd         %xmm3, %xmm0
                addsd         %xmm0, %xmm2
                subsd         %xmm0, %xmm7
                movsd         %xmm2, 0(%rbp,%rax,8)
                movsd         %xmm7, (%r14,%rax,8)
                addq          $1, %rax
                cmpq          %rax, %r15
                jne           .L8
                movq          %rsp, %rsi
                xorl          %edi, %edi
                salq          $3, %r15
                call          clock_gettime@PLT
                xorl          %eax, %eax
                .p2align 4,,10
                .p2align 3
.L9:
                movsd         0(%rbp,%rax), %xmm0
                addsd         (%r14,%rax), %xmm0
                movsd         %xmm0, (%r12,%rax)
                addq          $8, %rax
                cmpq          %r15, %rax
                jne           .L9
                leaq          16(%rsp), %rsi
```

```
                xorl    %edi, %edi
                call    clock_gettime@PLT
                movq    24(%rsp), %rax
                subq    8(%rsp), %rax
                pxor    %xmm0, %xmm0
                pxor    %xmm1, %xmm1
                cvtsi2sdq       %rax, %xmm0
                movq    16(%rsp), %rax
                subq    (%rsp), %rax
                cmpl    $9, %r13d
                cvtsi2sdq       %rax, %xmm1
                divsd   .LC6(%rip), %xmm0
                addsd   %xmm1, %xmm0
                jbe     .L28
                movl    %ebx, %eax
                movsd   (%r12), %xmm3
                movsd   (%r12,%rax,8), %xmm6
                leaq    .LC5(%rip), %rsi
                movsd   (%r14,%rax,8), %xmm5
                movl    %ebx, %r9d
                movsd   0(%rbp,%rax,8), %xmm4
                movl    %ebx, %r8d
                movsd   (%r14), %xmm2
                movl    %ebx, %ecx
                movsd   0(%rbp), %xmm1
                movl    %r13d, %edx
                movl    $1, %edi
                movl    $7, %eax
                call    __printf_chk@PLT
.L11:
                movq    %rbp, %rdi
                call    free@PLT
                movq    %r14, %rdi
                call    free@PLT
                movq    %r12, %rdi
                call    free@PLT
                xorl    %eax, %eax
                movq    40(%rsp), %rcx
                xorq    %fs:40, %rcx
                jne     .L29
                addq    $56, %rsp
                .cfi_remember_state
                .cfi_def_cfa_offset 56
                popq    %rbx
                .cfi_def_cfa_offset 48
                popq    %rbp
                .cfi_def_cfa_offset 40
                popq    %r12
                .cfi_def_cfa_offset 32
                popq    %r13
                .cfi_def_cfa_offset 24
                popq    %r14
                .cfi_def_cfa_offset 16
                popq    %r15
                .cfi_def_cfa_offset 8
                ret
.L28:
                .cfi_restore_state
                leaq    .LC7(%rip), %rsi
                leaq    .LC4(%rip), %r15
                movl    %r13d, %edx
                movl    $1, %edi
```

```
                movl          $1, %eax
                leaq          1(%rbx), %r13
                xorl          %ebx, %ebx
                call          __printf_chk@PLT
                .p2align 4,,10
                .p2align 3
.L12:
                movsd         (%r12,%rbx,8), %xmm2
                movl          %ebx, %r8d
                movsd         (%r14,%rbx,8), %xmm1
                movl          %ebx, %ecx
                movsd         0(%rbp,%rbx,8), %xmm0
                movl          %ebx, %edx
                movq          %r15, %rsi
                movl          $1, %edi
                movl          $3, %eax
                addq          $1, %rbx
                call          __printf_chk@PLT
                cmpq          %rbx, %r13
                jne           .L12
                jmp           .L11
.L27:
                movq          %rsp, %rsi
                xorl          %edi, %edi
                call          clock_gettime@PLT
                leaq          16(%rsp), %rsi
                xorl          %edi, %edi
                call          clock_gettime@PLT
                movq          24(%rsp), %rax
                subq          8(%rsp), %rax
                leaq          .LC7(%rip), %rsi
                pxor          %xmm0, %xmm0
                xorl          %edx, %edx
                pxor          %xmm1, %xmm1
                movl          $1, %edi
                cvtsi2sdq     %rax, %xmm0
                movq          16(%rsp), %rax
                subq          (%rsp), %rax
                cvtsi2sdq     %rax, %xmm1
                movl          $1, %eax
                divsd         .LC6(%rip), %xmm0
                addsd         %xmm1, %xmm0
                call          __printf_chk@PLT
                jmp           .L11
.L29:
                call          __stack_chk_fail@PLT
.L3:
                leaq          .LC2(%rip), %rdi
                call          puts@PLT
                movl          $-2, %edi
                call          exit@PLT
.L26:
                leaq          .LC0(%rip), %rdi
                call          puts@PLT
                orl           $-1, %edi
                call          exit@PLT
                .cfi_endproc
.LFE41:
                .size         main, .-main
                .section      .rodata.cst8,"aM",@progbits,8
                .align 8
.LC3:
```

```
                .long   2576980378
                .long   1069128089
                .align 8
.LC6:
                .long   0
                .long   1104006501
                .ident  "GCC: (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0"
                .section        .note.GNU-stack,"",@progbits
```

```
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP1/ejer6] 2019-04-03 miércoles
$echo 'time $PBS_O_WORKDIR/SumaVectoresC 10' | qsub -q ac
17608.atcgrid
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP1/ejer6] 2019-04-03 miércoles
$cat STDIN.o17608
Tama◆o Vectores:10 (4 B)
Tiempo:0.000000175      / Tama◆o 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) /
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP1/ejer6] 2019-04-03 miércoles
$cat STDIN.e17608

real    0m0.005s
user    0m0.000s
sys     0m0.002s
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP1/ejer6] 2019-04-03 miércoles
$
```

```
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP1/ejer6] 2019-04-03 miércoles
$cat STDIN.o176
STDIN.o17608  STDIN.o17611
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP1/ejer6] 2019-04-03 miércoles
$cat STDIN.o17611
Tama◆o Vectores:10000000 (4 B)
Tiempo:0.040263560      / Tama◆o Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+100000
0.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.
000000) /
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP1/ejer6] 2019-04-03 miércoles
$
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

La parte del código correspondiente a la suma, y que se halla entre .L8 y .L9 corresponde a **9 instrucciones** sin coma flotante, y a **3 instrucciones** (las 3 siguientes) a coma flotante. El NI que utilizaremos en cada calculo variará de el número de vectores utilizados.

MIPS = (NI*Num_Vectores) / (T_cpu * 10^6)

→ ***Para 10 instrucciones***

MIPS = (9*10) / (0.000000175*10^6) = 514.2857143

MFLOPS = (3*10) / (0.000000175*10^6) = 171.4285714

→ ***Para 100000000 instrucciones***

MIPS = (9*10000000) / (0.40263560*10^6) = 223.5271794

MFLOPS = (3*10000000) / (0.40263560*10^6) = 74.5090598

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

```
        call    clock_gettime@PLT
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L9:
        movsd   0(%rbp,%rax), %xmm0
        addsd   (%r14,%rax), %xmm0
        movsd   %xmm0, (%r12,%rax)
        addq    $8, %rax
        cmpq    %r15, %rax
        jne     .L9
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        movq    24(%rsp), %rax
        subq    8(%rsp), %rax
        pxor    %xmm0, %xmm0
```

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

```
//Inicializar vectores
#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;
  }
}

tiempo_inicio = omp_get_wtime();
//Calcular suma de vectores
#pragma omp parallel
{
  #pragma omp for
    for(i=0; i<N; i++)
      v3[i] = v1[i] + v2[i];
}
tiempo_fin = omp_get_wtime();
tiempo_total = tiempo_fin - tiempo_inicio;


//Imprimir resultado de la suma y el tiempo de ejecuci◆n
if (N<24) {
printf("Tiempo:%11.9f\t / Tama◆o Vectores:%u\n",tiempo_total,N);
for(i=0; i<N; i++)
  printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
         i,i,i,v1[i],v2[i],v3[i]);
}
else
  printf("Tiempo:%11.9f\t / Tama◆o 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",
         tiempo_total,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$gcc -O2 -fopenmp SumaVectoresC.c -o SumaVectoresC
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:34: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument
3 has type ‘long unsigned int’ [-Wformat=]
   printf("Tama◆o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                  ~^
                                  %lu
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$export OMP_DYNAMIC=FALSE
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$export OMP_NUM_THREADS=8
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$./SumaVectoresC 8
Tama◆o Vectores:8 (4 B)
Tiempo:0.000004898       / Tama◆o 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) /
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$
```

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$export OMP_DYNAMIC=FALSE
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$export OMP_NUM_THREADS=11
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$./SumaVectoresC 11
Tama◆o Vectores:11 (4 B)
Tiempo:0.000070695       / Tama◆o Vectores:11
/ V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2](1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3](1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4](1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5](1.600000+0.600000=2.200000) /
/ V1[6]+V2[6]=V3[6](1.700000+0.500000=2.200000) /
/ V1[7]+V2[7]=V3[7](1.800000+0.400000=2.200000) /
/ V1[8]+V2[8]=V3[8](1.900000+0.300000=2.200000) /
/ V1[9]+V2[9]=V3[9](2.000000+0.200000=2.200000) /
/ V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer7] 2019-04-03 miércoles
$
```

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

```
//Mostramos el numero de hebras, para saber con cuantas trabajamos
printf("Numero de hebras: %d\n", omp_get_max_threads());

//Creamos una variables que utilizaremos para medir el tiempo
double tiempo_inicio, tiempo_fin, tiempo_total;
//Inicializar vectores
#pragma omp parallel sections
{
    #pragma omp section
    {
      for(i=0; i<N/4; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
      }
    }

    #pragma omp section
    {
      for(i=0; i<N/4; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
      }
    }

    #pragma omp section
    {
      for(i=0; i<N/4; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
      }
    }

    #pragma omp section
    {
      for(i=0; i<N/4; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
      }
    }
}
```

Código (parte 1) usando la directiva Section

```
tiempo_inicio = omp_get_wtime();
//Calcular suma de vectores
#pragma omp parallel sections
{
  #pragma omp section
  {
      for(i=0; i<N/4; i++)
      v3[i] = v1[i] + v2[i];
  }

  #pragma omp section
  {
      for(i=0; i<N/4; i++)
      v3[i] = v1[i] + v2[i];
  }

  #pragma omp section
  {
      for(i=0; i<N/4; i++)
      v3[i] = v1[i] + v2[i];
  }

  #pragma omp section
  {
      for(i=0; i<N/4; i++)
      v3[i] = v1[i] + v2[i];
  }
}
tiempo_fin = omp_get_wtime();
tiempo_total = tiempo_fin - tiempo_inicio;

//Imprimir resultado de la suma y el tiempo de ejecuci♦n
if (N<24) {
printf("Tiempo:%11.9f\t / Tama♦o Vectores:%u\n",tiempo_total,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]);
}
```

Código (parte 2) usando la directiva Section

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer8] 2019-04-03 miércoles
$gcc -O2 -fopenmp SumaVectoresC.c -o SumaVectoresC
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:34: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument
3 has type ‘long unsigned int’ [-Wformat=]
   printf("Tama♦o Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                 ~^
                                 %lu
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer8] 2019-04-03 miércoles
$export OMP_DYNAMIC_FALSE=false
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer8] 2019-04-03 miércoles
$export OMP_NUM_THREADS=4
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer8] 2019-04-03 miércoles
$./SumaVectoresC 8
Tama♦o Vectores:8 (4 B)
Numero de hebras: 4
Tiempo:0.000004000       / Tama♦o 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](0.000000+0.000000=0.000000) /
/ V1[3]+V2[3]=V3[3](0.000000+0.000000=0.000000) /
/ V1[4]+V2[4]=V3[4](0.000000+0.000000=0.000000) /
/ V1[5]+V2[5]=V3[5](0.000000+0.000000=0.000000) /
/ V1[6]+V2[6]=V3[6](0.000000+0.000000=0.000000) /
/ V1[7]+V2[7]=V3[7](0.000000+0.000000=0.000000) /
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer8] 2019-04-03 miércoles
$
```

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer8] 2019-04-03 miércoles
$./SumaVectoresC 11
Tama◆o Vectores:11 (4 B)
Numero de hebras: 4
Tiempo:0.000003321       / Tama◆o Vectores:11
/ V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2](0.000000+0.000000=0.000000) /
/ V1[3]+V2[3]=V3[3](0.000000+0.000000=0.000000) /
/ V1[4]+V2[4]=V3[4](0.000000+0.000000=0.000000) /
/ V1[5]+V2[5]=V3[5](0.000000+0.000000=0.000000) /
/ V1[6]+V2[6]=V3[6](0.000000+0.000000=0.000000) /
/ V1[7]+V2[7]=V3[7](0.000000+0.000000=0.000000) /
/ V1[8]+V2[8]=V3[8](0.000000+0.000000=0.000000) /
/ V1[9]+V2[9]=V3[9](0.000000+0.000000=0.000000) /
/ V1[10]+V2[10]=V3[10](0.000000+0.000000=0.000000) /
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP1/ejer8] 2019-04-03 miércoles
$
```

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: El ejercicio 7 el número de threads y de cores depende del valor que le introduzcamos a los vectores (N=8 por ejemplo). En el ejercicio 8, sin embargo, el numero de threads y cores depende del número de núcleos por <<socket>> que tenga el computador. En mi caso, el computador dispone de 4, siendo por lo tanto el máximo en este ejercicio.

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:

Hacer uso de :

export OMP_DYNAMIC_FALSE

export OMP_NUM_THREADS(4 en PC y 12 en ATCGRID)

→TABLA CON DATOS DEL PC

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **4 threads/cores**	**T. paralelo (versión `sections`)** **4 threads/cores**
16384	0,000094960	0,000028765	0,000033469
32768	0,000181735	0,000057573	0,000059479
65536	0,000367860	0,000113416	0,000119295
131072	0,000773206	0,000233800	0,000208920
262144	0,001748746	0,000581434	0,000399781
524288	0,003232394	0,001273814	0,000800347
1048576	0,003305917	0,002526455	0,001659110
2097152	0,006569197	0,004054721	0,003405522
4194304	0,013443020	0,008199552	0,004324149
8388608	0,025887633	0,017402412	0,007535157
16777216	0,051320830	0,032585639	0,014850405
33554432	0,104486113	0,061110496	0,028549010
67108864	0,214336856	0,127441064	0,056547080

→ TABLA CON DATOS DE ATCGRID

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **12threads/cores**	**T. paralelo (versión `sections`)** **12threads/cores**
16384	0,000119219	0,000035411	0,000051428
32768	0,000232782	0,000047206	0,000080694
65536	0,000477205	0,000081960	0,000147715
131072	0,000940098	0,000117214	0,000298917
262144	0,001880805	0,000242207	0,000597123
524288	0,002824804	0,001293042	0,001598231
1048576	0,005389496	0,003110324	0,001595822
2097152	0,009671872	0,004833140	0,003334964
4194304	0,017648462	0,010531129	0,006707495
8388608	0,033712477	0,015823744	0,011312882
16777216	0,065913061	0,036602360	0,021468347
33554432	0,132120411	0,060225785	0,040317787
67108864	0,261578341	0,119963197	0,109638616

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.

Nº de Componentes	**Tiempo secuencial vect. Dinámicos 1 thread/core**			**Tiempo paralelo/versión `for` 12 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0,005	0,000	0,002	0,005	0,007	0,000
131072	0,002	0,001	0,001	0,002	0,000	0,008
262144	0,003	0,000	0,003	0,003	0,003	0,008
524288	0,004	0,002	0,002	0,003	0,012	0,004
1048576	0,007	0,000	0,007	0,004	0,014	0,010
2097152	0,011	0,003	0,008	0,007	0,037	0,024
4194304	0,016	0,007	0,009	0,010	0,049	0,047
8388608	0,027	0,016	0,011	0,014	0,101	0,048
16777216	0,046	0,022	0,024	0,024	0,166	0,081
33554432	0,079	0,037	0,042	0,044	0,270	0,162
67108864	0,154	0,091	0,062	0,085	0,520	0,287

RESPUESTA: En el caso de ejecutarlo en secuencial, el tiempo es el mismo. Sin embargo, en paralelo lo que ocurre es que el tiempo es mayor. Esto se debe a que en paralelo debemos crear y destruir todas las hebras/cores que utilizamos, y en secuencial no ocurre esto.