

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 Martínez Ortiz

Grupo de prácticas: C3

Fecha de entrega:

Fecha evaluación en clase:

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: código fuente `bucle-forModificado.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/  
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/  
/* INTERLINEADO SENCILLO */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>  
  
int main(int argc, char ** argv)  
{  
    int i, n = 9;  
    if(argc < 2) {  
        fprintf(stderr, "\n[ERROR] - Falta no iteraciones \n");  
        exit(-1);  
    }  
  
    n = atoi(argv[1]);  
  
    #pragma omp parallel for  
        for (i=0; i<n; i++)  
            printf("thread %d ejecuta la iteración %d del bucle\n",  
                omp_get_thread_num(), i);  
  
    return(0);  
}
```

RESPUESTA: código fuente `sectionsModificado.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/  
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/  
/* INTERLINEADO SENCILLO */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>  
  
int main(int argc, char ** argv)  
{
```

```
#pragma omp parallel sections
{
    #pragma omp section
    (void) funcA();
    #pragma omp section
    (void) funcB();
}
}
```

2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

RESPUESTA: código fuente `singleModificado.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char ** argv)
{
    int n = 9, i, a, b[n];
    for (i=0; i<n; i++) b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }
        #pragma omp for
        for (i=0; i<n; i++) b[i] = a;

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

CAPTURAS DE PANTALLA:

```

jose@Jose-K55VM: ~/Gil/AC/Practicas_grupo reducido_/Practica1_1
jose@Practica1_1$ ./single
Introduce valor de inicialización a: 23
Single ejecutada por el thread 1
Depués de la región parallel:
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
jose@Practica1_1$

```

- 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 `singleModificado2.c`

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char ** argv)
{
    int n = 9, i, a, b[n];
    for (i=0; i<n; i++) b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }
        #pragma omp for
        for (i=0; i<n; i++)
            b[i] = a;
        #pragma omp master
        {
            printf("Depués de la región parallel:\n");
            for (i=0; i<n; i++) printf("b[%d] = %d\t", i, b[i]);
        }
    }
}

```

```

printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
printf("\n");
}
}
}

```

CAPTURAS DE PANTALLA:

```

jose@Jose-K55VM: ~/GII/AC/Practicas_grupo reducido_/Practica1_1
jose@Practica1_1$ gcc singlemodificado2.c -o single2 -fopenmp
jose@Practica1_1$ ./single2
Introduce valor de inicialización a: 23
Single ejecutada por el thread 5
Depués de la región parallel:
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 0
jose@Practica1_1$

```

RESPUESTA A LA PREGUNTA: Que ahora muestra los resultado el thread master, el 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: Por que al eliminar la directiva barrier los thread no se esperan y cuando llegue el thread master imprimira el valor de la suma en ese momento.

Resto de ejercicios

5. El programa secuencial C del Listado 1 calcula la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i=0, \dots, N-1$). 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 el PC local, 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 mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA: Esto es debido a que tiempo real es la suma de los tiempo de cpu de usuario, del sistema y el tiempo asociado a las esperas debido a E/S o asociado a la ejecución de otros programas, no solo la suma de “user” con “sys”.

```

jose@Jose-K55VM: ~/GII/AC/Practicas_grupo reducido_/Practica1_1
jose@Practica1_1$ time ./SumaVectores 10000000
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)
real    0m0.114s
user    0m0.094s
sys     0m0.020s
jose@Practica1_1$

```

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). Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

CAPTURAS DE PANTALLA:

```

-rw-r--r-- 1 C3estudiante13 0 mar 30 16:11 STDIN.e32420
-rw-r--r-- 1 C3estudiante13 160 mar 2 17:04 STDIN.o26247
-rw-r--r-- 1 C3estudiante13 200 mar 30 16:11 STDIN.o32420
-rw-r--r-- 1 C3estudiante13 240 mar 30 16:05 .Xauthority
[C3estudiante13@atcgrid ~]$ echo 'hello/SumaVectores 10' | qsub -q ac
32436.atcgrid
[C3estudiante13@atcgrid ~]$ ls hello/ -l
total 60
-rwxrwxr-x 1 C3estudiante13 C3estudiante13 8783 mar 1 10:38 HelloOMP
-rw-rw-r-- 1 C3estudiante13 C3estudiante13 741 mar 4 10:52 listado3.sh
-rw-rw-r-- 1 C3estudiante13 C3estudiante13 870 mar 1 11:02 script_helloomp.s
-rw-r--r-- 1 C3estudiante13 C3estudiante13 0 mar 1 11:36 STDIN.e25258
-rw-r--r-- 1 C3estudiante13 C3estudiante13 21150 mar 1 11:36 STDIN.o25258
-rwxrwxr-x 1 C3estudiante13 C3estudiante13 8859 mar 30 16:10 SumaVectores
-rw-rw-r-- 1 C3estudiante13 C3estudiante13 3354 mar 30 16:07 SumaVectoresC.c
[C3estudiante13@atcgrid ~]$ ls hello/ -l
total 60
-rwxrwxr-x 1 C3estudiante13 C3estudiante13 8783 mar 1 10:38 HelloOMP
-rw-rw-r-- 1 C3estudiante13 C3estudiante13 741 mar 4 10:52 listado3.sh
-rw-rw-r-- 1 C3estudiante13 C3estudiante13 870 mar 1 11:02 script_helloomp.s
-rw-r--r-- 1 C3estudiante13 C3estudiante13 0 mar 1 11:36 STDIN.e25258
-rw-r--r-- 1 C3estudiante13 C3estudiante13 21150 mar 1 11:36 STDIN.o25258

```

```

C3estudiante13@atcgrid:~
jose@Practical_1$ ssh -X C3estudiante13@atcgrid.ugr.es
C3estudiante13@atcgrid.ugr.es's password:
Last login: Fri Mar 4 10:51:00 2016 from vpn-p002110.ugr.es
[C3estudiante13@atcgrid ~]$ ls
hello STDIN.e26247 STDIN.o26247
[C3estudiante13@atcgrid ~]$ ls hello/
HelloOMP script_helloomp.sh STDIN.o25258
listado3.sh STDIN.e25258 SumaVectoresC.c
[C3estudiante13@atcgrid ~]$ echo 'hello/SumaVectores 10000000' | qsub -q ac
32420.atcgrid
[C3estudiante13@atcgrid ~]$ qstat
Job id Name User Time Use S Queue
-----
32420.atcgrid STDIN C3estudiante13 00:00:00 C ac
[C3estudiante13@atcgrid ~]$ qstat
Job id Name User Time Use S Queue
-----
32420.atcgrid STDIN C3estudiante13 00:00:00 C ac
[C3estudiante13@atcgrid ~]$ qstat
Job id Name User Time Use S Queue
-----
32420.atcgrid STDIN C3estudiante13 00:00:00 C ac

```

RESPUESTA: cálculo de los MIPS y los MFLOPS. El tiempo de ejecución con $N=10$ ha sido 0.000002522 segundos y con $N=10^7$ es 0.047380682 segundos. El número de instrucciones entre las instrucciones clock_gettime() en el caso de MIPS es $6 \cdot 10^7 + 6$ ($6 \cdot N + 6$) y en el caso de MFLOPS es $3 \cdot 10^7 + 6$. Si hacemos el cálculo $NI / (Tcpu \cdot 10^6)$ obtenemos:

Con $N = 10000000 \Rightarrow$ MIPS = 1266,338990224 MFLOPS = 633,169568982

Con $N = 10 \Rightarrow$ MIPS = 25,773195876 MFLOPS = 11,89532117

RESPUESTA:

código ensamblador generado de la parte de la suma de vectores

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9. */
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ */
/* INTERLINEADO SENCILLO */
/* fichero SumaVectoresC.s */
        .file          "SumaVectoresC.c"
        .section       .rodata.str1.8, "aMS", @progbits, 1
        .align 8

.LC0:
        .string        "Faltan no componentes del vector"
        .align 8

```

```

.LC3:
    .string      "Tiempo(seg.):%11.9f\t / Tama\303\261o Vectores:
%u\t \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"
    .section     .text.startup,"ax",@progbits
    .p2align 4,,15
    .globl      main
    .type        main, @function

main:
.LFB39:
    .cfi_startproc
    pushq       %r12
    .cfi_def_cfa_offset 16
    .cfi_offset 12, -16
    pushq       %rbp
    .cfi_def_cfa_offset 24
    .cfi_offset 6, -24
    pushq       %rbx
    .cfi_def_cfa_offset 32
    .cfi_offset 3, -32
    subq        $32, %rsp
    .cfi_def_cfa_offset 64
    cmpl        $1, %edi
    jle         .L11
    movq        8(%rsi), %rdi
    movl        $10, %edx
    xorl        %esi, %esi
    movl        $33554432, %ebp
    call        strtol
    cmpl        $33554432, %eax
    cmovbe     %eax, %ebp
    testl       %ebp, %ebp
    je          .L3
    cvtsi2sd    %ebp, %xmm1
    leal        -1(%rbp), %r12d
    xorl        %ebx, %ebx
    movsd       .LC1(%rip), %xmm3
    movl        %r12d, %eax
    addq        $1, %rax
    mulsd       %xmm3, %xmm1
    .p2align 4,,10
    .p2align 3

.L5:
    cvtsi2sd    %ebx, %xmm0
    movapd     %xmm1, %xmm7
    mulsd       %xmm3, %xmm0
    movapd     %xmm0, %xmm2
    subsd       %xmm0, %xmm7
    addsd       %xmm1, %xmm2
    movsd       %xmm7, v2(,%rbx,8)
    movsd       %xmm2, v1(,%rbx,8)
    addq        $1, %rbx
    cmpq        %rax, %rbx
    jne         .L5
    movq        %rsp, %rsi
    xorl        %edi, %edi
    salq        $3, %rbx
    call        clock_gettime
    xorl        %eax, %eax
    .p2align 4,,10
    .p2align 3

.L7:

```

```

movsd    v1(%rax), %xmm0
addq     $8, %rax
addsd    v2-8(%rax), %xmm0
movsd    %xmm0, v3-8(%rax)
cmpq     %rbx, %rax
jne      .L7
.L6:
leaq     16(%rsp), %rsi
xorl     %edi, %edi
call     clock_gettime
movq     16(%rsp), %rdx
subq     (%rsp), %rdx
movl     %r12d, %eax
movsd    v3(,%rax,8), %xmm6
movl     %r12d, %r9d
movsd    v2(,%rax,8), %xmm5
movl     %r12d, %r8d
movsd    v1(,%rax,8), %xmm4
movl     %r12d, %ecx
cvtsi2sdq %rdx, %xmm0
movq     24(%rsp), %rdx
subq     8(%rsp), %rdx
movsd    v3(%rip), %xmm3
movsd    v2(%rip), %xmm2
movl     $.LC3, %esi
movl     $1, %edi
movl     $7, %eax
cvtsi2sdq %rdx, %xmm1
movl     %ebp, %edx
divsd    .LC2(%rip), %xmm1
addsd    %xmm1, %xmm0
movsd    v1(%rip), %xmm1
call     __printf_chk
addq     $32, %rsp
.cfi_restore_state
.cfi_def_cfa_offset 32
xorl     %eax, %eax
popq     %rbx
.cfi_def_cfa_offset 24
popq     %rbp
.cfi_def_cfa_offset 16
popq     %r12
.cfi_def_cfa_offset 8
ret
.L3:
.cfi_restore_state
movq     %rsp, %rsi
xorl     %edi, %edi
orl     $-1, %r12d
call     clock_gettime
jmp      .L6
.L11:
movl     $.LC0, %edi
call     puts
orl     $-1, %edi
call     exit
.cfi_endproc
.LFE39:
.size    main, .-main
.comm    v3,268435456,32
.comm    v2,268435456,32
.comm    v1,268435456,32

```

```

        .section      .rodata.cst8,"aM",@progbits,8
        .align 8
.LC1:
        .long         2576980378
        .long         1069128089
        .align 8
.LC2:
        .long         0
        .long         1104006501
        .ident        "GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.1) 4.8.4"
        .section      .note.GNU-stack,"",@progbits

```

7. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i=0, \dots, N-1$) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función `omp_get_wtime()`, que proporciona el estándar OpenMP, en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, $v3$, para varios tamaños pequeños de los vectores (por ejemplo, $N = 8$ y $N=11$); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de $v1$, $v2$ y $v3$ (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: código fuente implementado

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
/* fichero SumasOmp.c */
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

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();

//Muestra los valores de v1,v2,v3

    printf("%d \t %f \t %f \t %f \n",0,v1[0],v2[0],v3[0]);
    printf("%d \t %f \t %f \t %f \n",1,v1[1],v2[1],v3[1]);
    ncgt= cgt2 - cgt1;

    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
#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):

```

jose@jose-K55VM: ~/GII/AC/Practicas_grupo reducido_/Practica1_1
jose@Practica1_1$ gcc -O2 SumasOMP.c -o SumasOMP -lrt -fopenmp
jose@Practica1_1$ ./SumasOMP 8
Posicion V1      V2      V3
0      0.800000    0.800000    1.600000
1      0.900000    0.700000    1.600000
2      1.000000    0.600000    1.600000
3      1.100000    0.500000    1.600000
4      1.200000    0.400000    1.600000
5      1.300000    0.300000    1.600000
6      1.400000    0.200000    1.600000
7      1.500000    0.100000    1.600000
-----
Tiempo(seg.):0.001157514      / 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)
jose@Practica1_1$ ./SumasOMP 11
Posicion V1      V2      V3
0      1.100000    1.100000    2.200000
1      1.200000    1.000000    2.200000
2      1.300000    0.900000    2.200000
3      1.400000    0.800000    2.200000
4      1.500000    0.700000    2.200000
5      1.600000    0.600000    2.200000
6      1.700000    0.500000    2.200000
7      1.800000    0.400000    2.200000
8      1.900000    0.300000    2.200000
9      2.000000    0.200000    2.200000
10     2.100000    0.100000    2.200000
-----
Tiempo(seg.):0.004712179      / 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)
jose@Practica1_1$

```

8. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las `parallel` y `sections/section` (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva `for`); es decir, hay que repartir el trabajo (tareas) entre varios threads usando `sections/section`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función `omp_get_wtime()` en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, `v3`, para tamaños pequeños de los vectores (por ejemplo, $N = 8$ y $N=11$); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de `v1`, `v2` y `v3` (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: código fuente implementado

```

/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
/* fichero SumasOmp2.c */
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

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
        for(i=0; i<N/2; i++){
            v1[i] =
N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
        }
    #pragma omp section
        for(i=(N/2)+1; 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 sections
{
    #pragma omp section
        for(i=0; i<N/2; i++){
            v3[i] =
v1[i] + v2[i];
        }
    #pragma omp section
        for(i=(N/2)+1; i<N; i++){
            v3[i] = v1[i] + v2[i];
        }
}

cgt2 = omp_get_wtime();

//Muestra los valores de v1,v2,v3
printf("Posicion V1 \t\t V2 \t\t V3 \n");
for(i=0; i<N; i++)

```

```

        printf("%d \t %f \t %f \t %f
\n",i,v1[i],v2[i],v3[i]);
        printf("-----\n");

        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 \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",ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-
1]);
        #endif

#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):

```

jose@jose-K55VM: ~/GII/AC/Practicas_grupo reducido /Practica1_1
jose@Practica1_1$ gcc -O2 SumasOMP2.c -o SumasOMP2 -lrt -fopenmp
jose@Practica1_1$ ./SumasOMP2 8
Posicion V1      V2      V3
0      0.800000    0.800000    1.600000
1      0.900000    0.700000    1.600000
2      1.000000    0.600000    1.600000
3      1.100000    0.500000    1.600000
4      0.000000    0.000000    0.000000
5      1.300000    0.300000    1.600000
6      1.400000    0.200000    1.600000
7      1.500000    0.100000    1.600000
-----
Tiempo(seg.):0.004479073 / 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)
jose@Practica1_1$ ./SumasOMP2 11
Posicion V1      V2      V3
0      1.100000    1.100000    2.200000
1      1.200000    1.000000    2.200000
2      1.300000    0.900000    2.200000
3      1.400000    0.800000    2.200000
4      1.500000    0.700000    2.200000
5      0.000000    0.000000    0.000000
6      1.700000    0.500000    2.200000
7      1.800000    0.400000    2.200000
8      1.900000    0.300000    2.200000
9      2.000000    0.200000    2.200000
10     2.100000    0.100000    2.200000
-----
Tiempo(seg.):0.000320072 / 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)
jose@Practica1_1$

```

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 cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA: Como máximo se podría utilizar 8 thread y 4 cores, ya que mi pc local tiene un Intel i7 3610QM y tiene 4 cores fisicos y un total de 8 thread logicos.

En la caso del ejercicio 8 igual que para el ejercicio7 ya que la maquina fisica es la misma y se pueden utilizar el mismo número de threads

10. Rellenar una tabla como la Tabla 2Error: Reference source not found para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución 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. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

PC LOCAL			
Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 2 threads/cores	T. paralelo (versión sections) 2 threads/cores
16384	0.000416901	0.001488180	0.000882708
32768	0.000816674	0.001138798	0.000335236
65536	0.001552995	0.004354044	0.000677035
131072	0.002841389	0.007242692	0.000505278
262144	0.004059699	0.001679301	0.000987031
524288	0.006770054	0.002204862	0.004792929
1048576	0.011151692	0.007047004	0.011633327
2097152	0.018281742	0.005773935	0.002658425
4194304	0.031842276	0.009225012	0.012781238
8388608	0.063187880	0.018060645	0.022983799
16777216	0.000416901	0.035801184	0.039648724
33554432	0.000816674	0.070496969	0.064772528
67108864	0.001552995	0.070982531	0.064772528

ATC GRID			
Nº de Componentes	T. secuencial vect. Globales 4 thread/core	T. paralelo (versión for) 4 threads/cores	T. paralelo (versión sections) 4 threads/cores
16384	0.000100402	0.000097658	0.005695588
32768	0.000197059	0.000197976	0.000130200
65536	0.000388863	0.000394189	0.000257377
131072	0.000774980	0.000768573	0.000560542
262144	0.001477882	0.001368752	0.001095353
524288	0.002329666	0.002276422	0.004828924
1048576	0.006155159	0.005003889	0.006298008
2097152	0.010133266	0.010103624	0.008646455
4194304	0.020023555	0.019857669	0.014940373
8388608	0.039683815	0.040182111	0.026884592
16777216	0.079367433	0.079625968	0.052027259
33554432	0.159158478	0.159894174	0.112880543
67108864	0.158306600	0.158619380	0.090991759

RESPUESTA:

11. Rellenar una tabla como la Tabla 3 para el PC local con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿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.

PCLOCAL

Nº de Componente s	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 2Threads/cores		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
65536	003	000	003	011	058	005
131072	004	004	000	012	073	000
262144	008	004	004	016	090	000
524288	011	006	005	012	052	026
1048576	018	018	002	022	105	005
2097152	028	023	004	018	079	037
4194304	044	040	004	037	141	036
8388608	069	061	008	047	283	034
16777216	111	078	033	087	471	169
33554432	205	164	042	165	909	356
67108864						

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.

ATCGRID

Nº de Componente s	Tiempo secuencial vect. Globales 4 thread/core			Tiempo paralelo/versión for 4Threads/cores		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
65536	006	000	003	052	000	002
131072	002	000	002	055	000	002
262144	003	001	001	100	002	001
524288	004	002	002	055	001	003
1048576	006	002	004	055	003	004
2097152	009	004	005	067	006	004
4194304	018	008	011	078	012	009
8388608	036	009	026	088	015	024
16777216	073	030	043	122	037	035
33554432	139	053	085	188	057	081
67108864	275	108	165	331	115	159