

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:

Ejercicios basados en los ejemplos del seminario práctico

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

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

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

int main(int argc, char **argv) {
    int i, n = 9;
    if(argc < 2) {
        fprintf(stderr, "\n[ERROR] - Falta no iteraciones \n");
        exit(-1);
    }
    n = atoi(argv[1]);
    #pragma omp parallel for

    for (i=0; i<n; i++)
        printf("thread %d ejecuta la iteración %d del bucle\n",
            omp_get_thread_num(), i);

    return(0);
}
```

RESPUESTA: Captura que muestre el código fuente `sectionsModificado.c`

```
#include <stdio.h>
#include <omp.h>
void funcA() {
    printf("En funcA: esta sección la ejecuta el thread\n", omp_get_thread_num());
}

void funcB() {
    printf("En funcB: esta sección la ejecuta el thread\n", omp_get_thread_num());
}
```

ATCGRID

```

}

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

```

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

RESPUESTA: Captura que muestre el código fuente `singleModificado.c`

```

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

int main() {
    int n = 9, i, a, b[n];
    for (i=0; i<n; i++) b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        { printf("Introduce valor de 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("\n");
        }
    }
}

```

CAPTURAS DE PANTALLA:

```

$./single
Introduce valor de inicialización a: 2
Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 2      b[1] = 2      b[2] = 2      b[3] = 2      b[4] =
2      b[5] = 2      b[6] = 2      b[7] = 2      b[8] = 2

```

- . 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>

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

```

CAPTURAS DE PANTALLA:

ATCGRID

```

$./single
Introduce valor de inicialización a: 2
Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 2      Master ejecutada por el thread 0
b[1] = 2      Master ejecutada por el thread 0
b[2] = 2      Master ejecutada por el thread 0
b[3] = 2      Master ejecutada por el thread 0
b[4] = 2      Master ejecutada por el thread 0
b[5] = 2      Master ejecutada por el thread 0
b[6] = 2      Master ejecutada por el thread 0
b[7] = 2      Master ejecutada por el thread 0
b[8] = 2      Master ejecutada por el thread 0

```

RESPUESTA A LA PREGUNTA:

Que todas las ejecuciones las realiza la hebra master que es la cero, mientras que en la singer puede ser cualquiera de las otras hebras

- . ¿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 barrera las hebras no se esperan en la barrera, probocando que según acaben sobrescriban los resultados

Resto de ejercicios

- . 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 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:

```

[A3estudiante16@atcgrid ~]$ cat STDIN.e71873
real    0m0.017s
user    0m0.005s
sys      0m0.012s
[A3estudiante16@atcgrid ~]$ cat STDIN.o71873
Tiempo(seg.):0.005660497 / Tamaño Vectores:1000000
/ V1[0]+V2[0]=V3[0](100000.000000+100000.000000=200000.000000) / /
V1[999999]+V2[999999]=V3[999999](199999.900000+0.100000=200000.000000)
/

```

- . 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:

```
[A3estudiante16@atcgrid ~]$ cat STDIN.o71887
Tiempo(seg.):0.006127133 / Tamaño Vectores:1000000
/ v1[0]+v2[0]=v3[0](100000.000000+100000.000000=200000.000000) / /
v1[999999]+v2[999999]=v3[999999](199999.900000+0.100000=200000.000000)
/

[A3estudiante16@atcgrid ~]$ echo "time ./programa_secuencial 10" | qsub
-q ac
71894.atcgrid
[A3estudiante16@atcgrid ~]$ cat STDIN.o71894
Tiempo(seg.):0.000002675 / Tamaño Vectores:10
/ v1[0]+v2[0]=v3[0](1.000000+1.000000=2.000000) / /
v1[9]+v2[9]=v3[9](1.900000+0.100000=2.000000) /
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

a)10 iteraciones

$MIPS = NI / (T_{cpu} * 10^6)$.

$MIPS = 63 / 0.000002675 * 10^6 = 19,61$

$MFLOPS = OPERACIONES / (T_{cpu} * 10^6)$.

$MFLOPS = 10 / 0.000002675 * 10^6 = 3,113$

b)1000000

$MIPS = NI / (T_{cpu} * 10^6)$.

$MIPS = 60000003 / 0.006127133 * 10^6 = 1095,30$

$MFLOPS = OPERACIONES / (T_{cpu} * 10^6)$.

$MFLOPS = 10000000 / 0.006127133 * 10^6 = 182,550$

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

```
call clock_gettime
xorl %eax, %eax
.p2align 4,,10
.p2align 3
.L5:
movsd v1(%rax), %xmm0
addq $8, %rax
addsd v2-8(%rax), %xmm0
movsd %xmm0, v3-8(%rax)
cmpq %rax, %rbx
jne .L5
.L6:
leaq 16(%rsp), %rsi
```

ATCGRID

```
xorl %edi, %edi
call clock_gettime
```

4. 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: Captura que muestre el código fuente implementado

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

//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 67108864
double v1[MAX], v2[MAX], v3[MAX];
#endif

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

    double ini,fin;

    double 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

    #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
            //printf("%f\n",v1[i] ); //se lo añadido yo para saber lo que hace
        }

    ini = omp_get_wtime();
    //Calcular suma de vectores

    #pragma omp parallel for
        for(i=0; i<N; i++)
            v3[i] = v1[i] + v2[i];

    fin = omp_get_wtime();

    ncgt= fin-ini;

    //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\n/

```

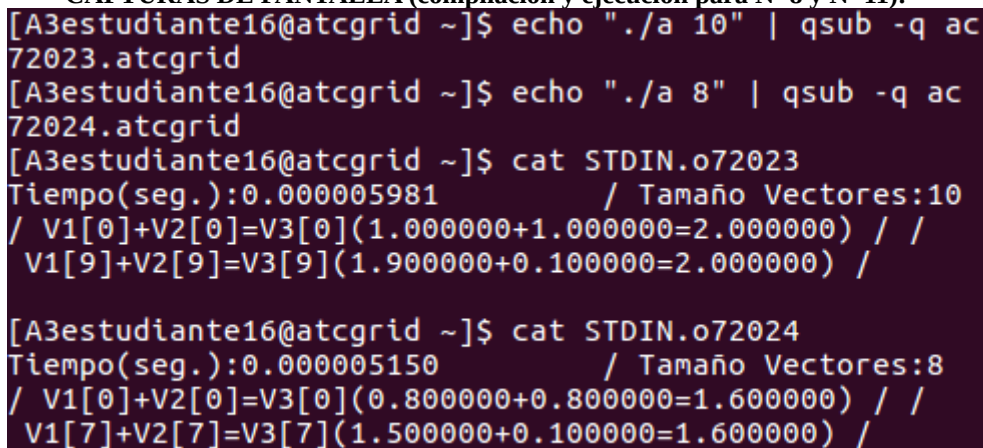
ATCGRID

```

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

        // PRINTF MAS APROPIADO PARA EL EJERCICIO 9
        // printf("Tiempo: %11.9f\t Tamaño en bytes del vector: %u\n\n", ncgt,
N*sizeof(double));
    #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):**


```

[A3estudiante16@atcgrid ~]$ echo "./a 10" | qsub -q ac
72023.atcgrid
[A3estudiante16@atcgrid ~]$ echo "./a 8" | qsub -q ac
72024.atcgrid
[A3estudiante16@atcgrid ~]$ cat STDIN.o72023
Tiempo(seg.):0.000005981          / Tamaño Vectores:10
/ V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) / /
V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /

[A3estudiante16@atcgrid ~]$ cat STDIN.o72024
Tiempo(seg.):0.000005150          / 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) /

```

- . 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: Captura que muestre el código fuente implementado

```

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

```



```

//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 67108864
double v1[MAX], v2[MAX], v3[MAX];
#endif

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

    double ini,fin;

    double 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

```

ATCGRID

```

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

//Dividimos el problema en 4 partes

#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; //los valores dependen de N
            //printf("%f\n",v1[i] ); //se lo añado yo para saber lo que hace
        }

    #pragma omp section
        for(i=N/4; 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
            //printf("%f\n",v1[i] ); //se lo añado yo para saber lo que hace
        }

    #pragma omp section
        for(i=N/2; i<4*N/3; i++){
            v1[i] = N*0.1+i*0.1;
            v2[i] = N*0.1-i*0.1; //los valores dependen de N
            //printf("%f\n",v1[i] ); //se lo añado yo para saber lo que hace
        }

    #pragma omp section
        for(i=4*N/3; i<N; i++){
            v1[i] = N*0.1+i*0.1;
            v2[i] = N*0.1-i*0.1; //los valores dependen de N
            //printf("%f\n",v1[i] ); //se lo añado yo para saber lo que hace
        }

}
ini = omp_get_wtime();

```

```

//Calcular suma de vectores

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

}

#pragma omp parallel sections
{
    #pragma omp parallel for
        for(i=N/4; i<N/2; i++)
            v3[i] = v1[i] + v2[i];

}

#pragma omp parallel sections
{
    #pragma omp parallel for
        for(i=N/2; i<4*N/3; i++)
            v3[i] = v1[i] + v2[i];

}

#pragma omp parallel sections
{
    #pragma omp parallel for
        for(i=4*N/3; i<N; i++)
            v3[i
] = v1[i] + v2[i];

}

fin = omp_get_wtime();

ncgt= fin-ini;

//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\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\n",

```

ATCGRID

```

ncgt, N, v1[0], v2[0], v3[0], N-1, N-1, N-1, v1[N-1], v2[N-1], v3[N-1]);

// PRINTF MAS APROPIADO PARA EL EJERCICIO 9
// printf("Tiempo: %11.9f\t Tamaño en bytes del vector: %u\n\n", ncgt,
N*sizeof(double));
#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):

```

[A3estudiante16@atcgrid ~]$ echo "./b 10" | qsub -q ac
72026.atcgrid
[A3estudiante16@atcgrid ~]$ echo "./b 8" | qsub -q ac
72027.atcgrid
[A3estudiante16@atcgrid ~]$ cat STDIN.o72026
Tiempo(seg.):0.015966700          / Tamaño Vectores:10
/ V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) / /
V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /

[A3estudiante16@atcgrid ~]$ cat STDIN.o72027
Tiempo(seg.):0.015382979          / 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) /

```

- . ¿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:

En la versión del ejercicio 7 lo podrán ejecutar tantas hebras como disponga la máquina que lo ejecute. Y en cuanto a cores, dependerá del número de hebras que disponga cada core.

En al versión del ejercicio 8 tal como lo he planteado, lo podrán ejecutar como máximo 4 hebras. Y en cuanto a cores, dependerá de cuantas hebras disponga cada core.

- . 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. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

RESPUESTA:

MI PC

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 4threads/cores	T. paralelo (versión sections) 4threads/cores
16384	0.000102560	0.000204258	0.000329149
32768	0.000198223	0.000396511	0.000207534
65536	0.000480553	0.000333308	0.000423402
131072	0.000864473	0.000644919	0.002532279
262144	0.001497265	0.001288378	0.003925268
524288	0.002615732	0.002889286	0.008618782
1048576	0.004225977	0.004721980	0.006842636
2097152	0.013532305	0.009257841	0.013244836
4194304	0.016962191	0.020610539	0.024575901
8388608	0.035592406	0.034353954	0.047029451
16777216	0.067993580	0.067023493	0.093047791
33554432	0.156815011	0.189389385	0.181544508
67108864	0.362395737	0.406822337	

ATCGRID
ATCGRID

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24threads/cores	T. paralelo (versión sections) 24threads/cores
16384	0.000077100	0.000139022	0.016307745
32768	0.000163176	0.000251246	0.015529985
65536	0.000287508	0.000467504	0.014178379
131072	0.000459156	0.000924023	0.015685344
262144	0.001185846	0.001787399	0.015808880
524288	0.002512682	0.003515960	0.016699555
1048576	0.005765041	0.007416495	0.019655190
2097152	0.011121925	0.013061455	0.026469337
4194304	0.022837296	0.024474751	0.048122738
8388608	0.053917234	0.049820964	0.083120986
16777216	0.100441736	0.103670693	0.150746991
33554432	0.197075850	0.210791107	0.285824574
67108864	0.359280208	0.357872777	0.364596371

- . Rellenar una tabla como la Tabla 3 para atcgrid con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

RESPUESTA:

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados.

Nº de Componente s	Tiempo secuencial vect. Globales 1 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	real	0m0.003s		real	0m0.129s	
	user	0m0.003s		user	0m0.004s	
	sys	0m0.000s		sys	0m0.002s	
131072	real	0m0.004s		real	0m0.004s	
	user	0m0.004s		user	0m0.009s	
	sys	0m0.000s		sys	0m0.003s	
262144	real	0m0.006s		real	0m0.004s	
	user	0m0.000s		user	0m0.011s	
	sys	0m0.006s		sys	0m0.000s	
524288	real	0m0.008s		real	0m0.009s	
	user	0m0.000s		user	0m0.017s	
	sys	0m0.008s		sys	0m0.004s	
1048576	real	0m0.016s		real	0m0.011s	
	user	0m0.004s		user	0m0.009s	
	sys	0m0.012s		sys	0m0.018s	
2097152	real	0m0.029s		real	0m0.023s	
	user	0m0.011s		user	0m0.041s	
	sys	0m0.018s		sys	0m0.013s	
4194304	real	0m0.057s		real	0m0.042s	
	user	0m0.021s		user	0m0.040s	
	sys	0m0.037s		sys	0m0.048s	
8388608	real	0m0.108s		real	0m0.081s	
	user	0m0.028s		user	0m0.075s	
	sys	0m0.080s		sys	0m0.087s	
16777216	real	0m0.205s		real	0m0.152s	
	user	0m0.064s		user	0m0.110s	
	sys	0m0.140s		sys	0m0.189s	
33554432	real	0m0.415s		real	0m0.290s	
	user	0m0.156s		user	0m0.219s	
	sys	0m0.259s		sys	0m0.361s	
67108864	real	0m0.788s		real	0m0.617s	
	user	0m0.316s		user	0m0.400s	
	sys	0m0.472s		sys	0m0.741s	

ATCGRID

Nº de Componentes	Tiempo secuencial vect. Globales 1 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	Real	0m0.004s		real		0m0.003s
	user	0m0.006s		user		0m0.003s
	sys	0m0.000s		sys		0m0.000s
				real	0m0.004s	
131072	Real	0m0.004s		real		0m0.004s
	user	0m0.006s		user		0m0.001s
	sys	0m0.000s		sys		0m0.003s
				real	0m0.006s	
262144	Real	0m0.004s		real		0m0.006s
	user	0m0.007s		user		0m0.001s
	sys	0m0.000s		sys		0m0.005s
				real	0m0.012s	
524288	Real	0m0.005s		Real	0m0.012s	
	user	0m0.011s		user	0m0.004s	
	sys	0m0.000s		sys	0m0.008s	
1048576	Real	0m0.005s		Real	0m0.022s	
	user	0m0.012s		user	0m0.005s	
	sys	0m0.004s		sys	0m0.016s	
2097152	Real	0m0.011s		Real	0m0.037s	
	user	0m0.019s		user	0m0.017s	
	sys	0m0.011s		sys	0m0.019s	
4194304	Real	0m0.013s		Real	0m0.072s	
	user	0m0.019s		user	0m0.028s	
	sys	0m0.015s		sys	0m0.044s	
8388608	Real	0m0.026s		Real	0m0.144s	
	user	0m0.029s		user	0m0.053s	
	sys	0m0.028s		sys	0m0.088s	
16777216	Real	0m0.053s		Real	0m0.291s	
	user	0m0.041s		user	0m0.086s	
	sys	0m0.065s		sys	0m0.202s	
				real	0m0.604s	
33554432	Real	0m0.097s		Real	0m0.604s	
	user	0m0.072s		user	0m0.209s	
	sys	0m0.136s		sys	0m0.391s	
				r		
67108864	Real	0m0.357s		Real	0m1.201s	
	user	0m0.309s		user	0m0.411s	
	sys	0m0.395s		sys	0m0.772s	