

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): Fco Javier Merchán Martín

Grupo de prácticas: B2

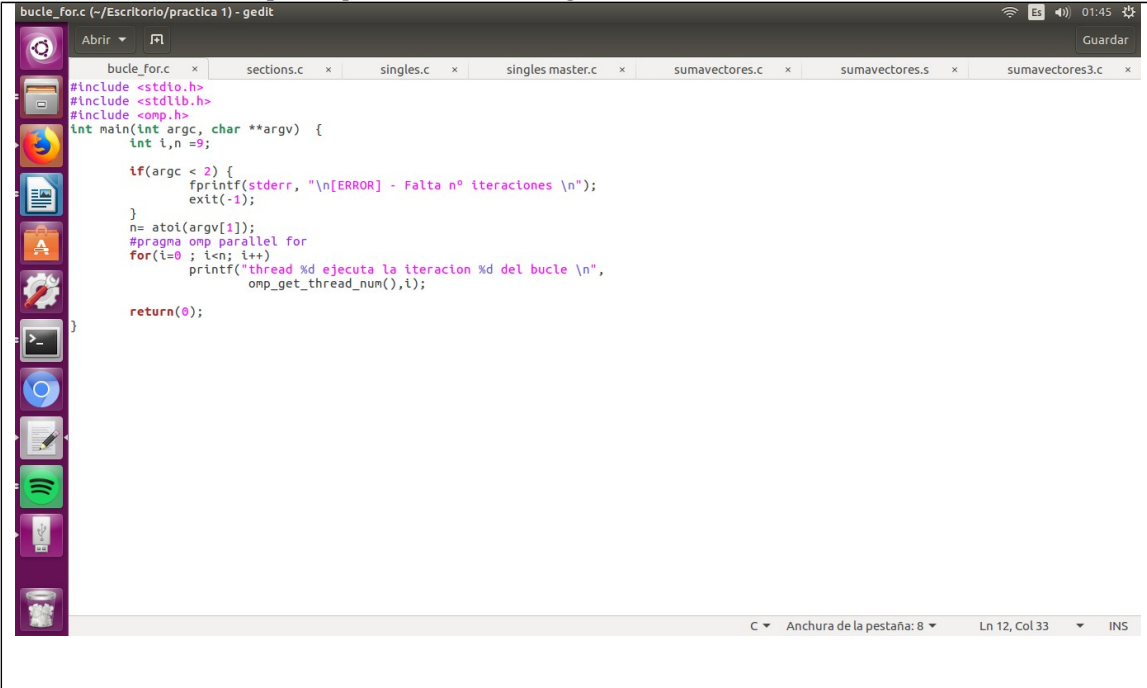
Fecha de entrega: 03-03-2018

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 iteracion %d del bucle \n",
            omp_get_thread_num(), i);

    return(0);
}
```

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

```

sections.c (~/Escritorio/practica 1) - gedit
bucle_for.c x sections.c x singles.c x singles master.c x sumavectores.c x sumavectores.s x sumavectores3.c x
#include <stdio.h>
#include <omp.h>

void funcA() {
    printf("En funcA: esta seccion la ejecuta el thread %d\n",
        omp_get_thread_num());
}

void funcB() {
    printf("En funcB: esta seccion 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();
    }
}
C Anchura de la pestaña: 8 Ln 11, Col 2 INS
    
```

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

```

singles.c (~/Escritorio/practica 1) - gedit
bucle_for.c x sections.c x singles.c x singles master.c x sumavectores.c x sumavectores.s x sumavectores3.c x
#include <stdio.h>
#include <omp.h>

main() {
    int n = 9, i, a, b[n];
    for (i = 0; i < n; i++)
        b[i] = i;

    #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
        {
            for (i = 0; i < n; i++)
                printf("b[%d] = %d\t", i, b[i]);
            printf("\n");
            printf("Single ejecutada por thread %d\n", omp_get_thread_num());
        }
    }
}
C Anchura de la pestaña: 8 Ln 1, Col 1 INS
    
```

CAPTURAS DE PANTALLA:

```

fran@fran-Aspire-5742: ~
$ ./singlesmod
Introduce valor de inicialización a: 50
Single ejecutada por el thread 1
Single ejecutada por thread 0
b[0] = 50 b[1] = 50 b[2] = 50 b[3] = 50 b[4] = 50 b[5] = 50 b[6] = 50 b[7] = 50 b[8] = 50

```

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`

```

singlemaster.c (-/Escritorio/practica 1) - gedit
#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
        {
            for (i = 0; i < n; i++)
                printf("b[%d] = %d\t", i, b[i]);
            printf("\n");
            printf("Single ejecutada por thread %d\n", omp_get_thread_num());
        }
    }
}

```

CAPTURAS DE PANTALLA:

```

fran@fran-Aspire-5742: ~
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$gcc -O2 -fopenmp singlesmaster.c -o singlesmod -lrt
singlesmaster.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
main() {
^
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$./singlesmod
Introduce valor de inicialización a: 50
Single ejecutada por el thread 3
b[0] = 50      b[1] = 50      b[2] = 50      b[3] = 50      b[4] = 50      b[5] = 50      b[6] = 50      b[7] = 50      b[8] = 50
Single ejecutada por thread 0
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$

```

RESPUESTA A LA PREGUNTA: Al usar la directiva master, los resultados siempre los da la thread 0, es decir la master.

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

RESPUESTA: Por que al eliminar el barrier eliminas la barrera que hace que todas las hebras se unan, pudiendo dar resultado la hebra master antes de que acaben las otras hebras.

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 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: Es menor al tiempo real, ya que se ejecuta desde varios nucleos

```

B2estudiante10@atcgrid:~$ time echo './SumaVectores 10000000' | qsub -q ac
71152.atcgrid
real    0m0.034s
user    0m0.013s
sys      0m0.008s
[B2estudiante10@atcgrid ~]$ cat
.bash_history .bash_profile .joe_state .mozilla/ STDIN.e71151 STDIN.o71151 SumaVectores
.bash_logout .bashrc .local/ .ssh/ STDIN.e71152 STDIN.o71152 .Xauthority
[B2estudiante10@atcgrid ~]$ cat S
STDIN.e71151 STDIN.e71152 STDIN.o71151 STDIN.o71152 SumaVectores
[B2estudiante10@atcgrid ~]$ cat STDIN.o71152
Tiempo(seg.):0.000000134 / Tamaño Vectores:29 / V1[0]+V2[0]=V3[0](2.900000+2.900000+5.800000) / / V1[28]+V2[28]=V3[28](5.700000+0.100
000+5.800000) /
[B2estudiante10@atcgrid ~]$

```

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:

```

fran@fran-Aspire-5742:~$ ./SumaVectores 10000000
Tiempo(seg.):0.000000271 / Tamaño Vectores:29 / V1[0]+V2[0]=V3[0](2.900000+2.900000+5.800000) / / V1[28]+V2[28]=V3[28](5.700000+0.100
000+5.800000) /
fran@fran-Aspire-5742:~$ ./SumaVectores 10
Tiempo(seg.):0.000000350 / Tamaño Vectores:29 / V1[0]+V2[0]=V3[0](2.900000+2.900000+5.800000) / / V1[28]+V2[28]=V3[28](5.700000+0.100
000+5.800000) /
fran@fran-Aspire-5742:~$

```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Tamaño vector = 10: NI: 63 ($6 \cdot 10 + 3$) FPO: 30 ($3 \cdot 10$) Tiempo(seg.): 0.000000350 MIPS: $63 / (0.000000350 \cdot 10^6) = 180$ MFLOPS: $30 / (0.000000350 \cdot 10^6) = 85,714$	Tamaño vector = 10000000 : NI: 60000003 ($6 \cdot 10000000 + 3$) FPO: 30000000 ($3 \cdot 10000000$) Tiempo(seg.): 0.000000271 MIPS: $60000003 / (0.048853831 \cdot 10^6) = 221402225,1$ MFLOPS: $30000000 / (0.048853831 \cdot 10^6) = 110701118,1$
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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

```

sumavectores.s (~/Escritorio/practica 1) - gedit
bucle_for.c x sections.c x singles.c x sumavectores.c x sumavectores.s x sumavectores3.c x singlesmaster.c x sumavectores.c x
movapd %xmm0, %xmm2
subsd %xmm0, %xmm7
addsd %xmm1, %xmm2
movsd %xmm7, v2(,%rax,8)
movsd %xmm2, v1(,%rax,8)
addq $1, %rax
cmpl $29, %rax
jne .L3
movq %rsp, %r10
xorl %edi, %edi
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)
cmpl $232, %rax
jne .L5
leaq 16(%rsp), %r10
xorl %edi, %edi
call clock_gettime
movq 24(%rsp), %rax
subq 8(%rsp), %rax
movl $28, %ecx
pxor %xmm0, %xmm0
movl $28, %r9d
pxor %xmm1, %xmm1
movl $28, %r8d
movsd v3+224(%rip), %xmm6
movl $29, %edx
cvttsd2sq %rax, %xmm0
movq 16(%rsp), %rax
subq (%rsp), %rax
movsd v2+224(%rip), %xmm5
movsd v1+224(%rip), %xmm4

```

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: Captura que muestre el código fuente implementado

```

sumavectores3.c (~/Escritorio/practica 1) - gedit
Abrir Guardar
bucle_for.c x sections.c x singles.c x sumavectores.c x sumavectores.s x sumavectores3.c x singlesmaster.c x sumavectores.c x
/* SumaVectores.c
*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <omp.h>

#define PRINTF_ALL
#define VECTOR_LOCAL
#define VECTOR_GLOBAL
#define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 2732-1
double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char** argv){
    int i;
    struct timespec cgt1,cgt2; double ncgt;
    double t1, t2, total;

    if (argc<2){
        printf("Faltan n1 componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    #ifdef VECTOR_LOCAL
    double v1[N], v2[N], v3[N];
    #endif
    #ifdef VECTOR_GLOBAL

#ifdef VECTOR_GLOBAL
    if (N>MAX) N=MAX;
#endif
    #ifdef VECTOR_DYNAMIC
    double *v1, *v2, *v3;
    v1 = (double*) malloc(N*sizeof(double));
    v2 = (double*) malloc(N*sizeof(double));
    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
    {
        //Inicializar vectores
        #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;
        }

        //clock_gettime(CLOCK_REALTIME,&cgt1);
        #pragma omp single
        {
            t1 = omp_get_wtime();
        }

        #pragma omp for
        for(i=0; i<N; i++){
            v3[i] = v1[i] + v2[i];
        }
        //clock_gettime(CLOCK_REALTIME,&cgt2);
        #pragma omp single
        {
            t2 = omp_get_wtime();
        }
    }

    total = t2- t1;
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    #ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",diferencia,N);
    for(i=0; i < N; i++){
        printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) \n", i,i,i,v1[i],v2[i],v3[i]);
    }
    #else
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f+%8.6f) \n", ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
    #endif

    #ifdef VECTOR_DYNAMIC
    free(v1);
    free(v2);
    free(v3);
    #endif
    return 0;
}

```


(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

fran@fran-Aspire-5742: ~
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$gcc -O2 -fopenmp singlesmaster.c -o singlesmod -lrt
singlesmaster.c:3:11: warning: return type defaults to 'int' [-Wimplicit-int]
    main() {
    ^
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$./singlesmod
Introduce valor de inicialización a: 50
Single ejecutada por el thread 3
b[0] = 50      b[1] = 50      b[2] = 50      b[3] = 50      b[4] = 50      b[5] = 50      b[6] = 50      b[7] = 50      b[8] = 50
Single ejecutada por thread 0
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$gcc -O2 -fopenmp sumavectores3.c -o sumavectores3 -lrt
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$./sumavectores3 8
Tiempo(seg.):0.000000000 / Tamaño Vectores:29 / V1[0]+V2[0]=V3[0](2.900000+2.900000+5.800000) / V1[28]+V2[28]=V3[28](5.700000+0.100
000+5.800000) /
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$./sumavectores3 11
Tiempo(seg.):0.000000000 / Tamaño Vectores:29 / V1[0]+V2[0]=V3[0](2.900000+2.900000+5.800000) / V1[28]+V2[28]=V3[28](5.700000+0.100
000+5.800000) /
[FCo Javier Merchan Martin fran@fran-Aspire-5742:~/Escritorio/practica 1] 2018-04-03 martes
$

```

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: Captura que muestre el código fuente implementado

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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:

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

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) ¿?threads/cores	T. paralelo (versión sections) ¿?threads/cores
16384			
32768			
65536			
131072			
262144			
524288			
1048576			
2097152			
4194304			
8388608			
16777216			
33554432			
67108864			

RESPUESTA:

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.

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 ¿? Threads/cores		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
65536						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						