Grai2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos):
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? (añada una captura de pantalla que muestre lo que ocurre) **(b)** Resuelva el problema generado sin eliminar `default(none)`.Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA: Nos da error por no especificar el ámbito de la variable n. La añadimos a shared para compartirla con todas las hebras.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```c
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif
int main(){
  int i, n = 7;
  int a[n];
  for (i=0; i<n; i++)
  a[i] = i+1;
  #pragma omp parallel for shared(a, n), default(none)
    for (i=0; i<n; i++) a[i] += i;
      printf("Después de parallel for:\n");
    for (i=0; i<n; i++)
      printf("a[%d] = %d\n",i,a[i]);
}
```

CAPTURAS DE PANTALLA:

```
sharec-clauseEjer1.c:10:11: error: ‘n’ not specified in enclosing parallel
   #pragma omp parallel for shared(a), default(none)
           ^
sharec-clauseEjer1.c:10:11: error: enclosing parallel
```

2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? (inicialice suma a un valor distinto de 0 dentro y fuera de `parallel`) Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA: Al inicializarla fuera el valor que tendría dentro de la parte paralela seria basura, ya que con private los valores tanto al entrar en la zona paralela como al salir de las variables es desconocido(basura).

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```c
int main(){
  int i, n = 7;
  int a[n], suma;
  for (i=0; i<n; i++)
    a[i] = i;
  //suma=3;
  #pragma omp parallel private(suma)
  {
    suma=3;
    #pragma omp for
    for (i=0; i<n; i++){
      suma = suma + a[i];
      printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
    }
    printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
  }
  printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/Codigos] 2018-04-16 Monday
$./private-clauseEjer2.eje
thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[4] / thread 2 suma a[5] / thread 1 suma a[2] /
 thread 1 suma a[3] / thread 3 suma a[6] /
* thread 2 suma= 4196569
* thread 1 suma= 4196565
* thread 3 suma= 4196566
* thread 0 suma= 5
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/Codigos] 2018-04-16 Monday
$gcc -fopenmp -O2 private-clauseEjer2.c -o private-clauseEjer2.eje
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/Codigos] 2018-04-16 Monday
$./private-clauseEjer2.eje
thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 1 suma a[2] / thread 1 suma a[3] /
 thread 2 suma a[4] / thread 2 suma a[5] /
* thread 1 suma= 8
* thread 3 suma= 9
* thread 0 suma= 4
* thread 2 suma= 12
```

3. ¿Qué ocurre si en `private-clause.c` se elimina la cláusula `private(suma)`? ¿A qué cree que es debido?

RESPUESTA: Al estar declarada la variable fuera, esta pasa a ser una variable compartida y todas las hebras trabajan con ella.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```
int main(){
  int i, n = 7;
  int a[n], suma;
  for (i=0; i<n; i++)
    a[i] = i;
  //suma=3;
  #pragma omp parallel
  {
    suma=0;
    #pragma omp for
    for (i=0; i<n; i++){
      suma = suma + a[i];
      printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
    }
    printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
  }
  printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/Codigos] 2018-04-16 Monday
$./private-clauseEjer3.eje
thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 1 suma a[2] / thread 1 suma a[3] /
 thread 2 suma a[4] / thread 2 suma a[5] /
* thread 1 suma= 15
* thread 3 suma= 15
* thread 2 suma= 15
* thread 0 suma= 15
```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. ¿El código imprime siempre 6 fuera de la región `parallel`? Razone su respuesta.

RESPUESTA: Si lastprivate se queda con el valor que daría la ultima hebra a la variable en caso de ejecutarse secuencialmente

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$./firstprivate-clause.eje
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9
 thread 3 suma a[6] suma=6

Fuera de la construcción parallel suma=6
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$./firstprivate-clause.eje
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 3 suma a[6] suma=6
 thread 2 suma a[4] suma=4
 thread 2 suma a[5] suma=9
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5

Fuera de la construcción parallel suma=6
```

5. ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

RESPUESTA: Lo que ocurre es que no se copia el valor pasado a la variable a por teclado en todas las hebras. De modo que solo se modifica el valor de a para la hebra que pide los datos.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```c
#include <stdio.h>
#include <omp.h>
int main() {
  int n = 9, i, b[n];
  for (i=0; i<n; i++) b[i] = -1;
  #pragma omp parallel
    {
      int a;
      #pragma omp single
      {
        printf("\nIntroduce valor de inicialización a: ");
        scanf("%d", &a );
        printf("\nSingle ejecutada por el thread %d\n",
        omp_get_thread_num());
      }
      #pragma omp for
      for (i=0; i<n; i++) b[i] = a;
    }
    printf("Depués de la región parallel:\n");
    for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
      printf("\n");
  }
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$./copyprivate-clause.eje

Introduce valor de inicialización a: 5

Single ejecutada por el thread 0
Depués de la región parallel:
b[0] = 5        b[1] = 5        b[2] = 5        b[3] = 0        b[4] = 0
b[5] = 0        b[6] = 0        b[7] = 0        b[8] = 0
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$
```

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado

RESPUESTA: El programa suma números de 0 a n. Si se cambia suma a 10 el resultado de la suma se sumara con el 10 introducido.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

CAPTURAS DE PANTALLA:

7. En el ejemplo `reduction-clause.c`, elimine `reduction()` de `#pragma omp parallel for reduction(+:suma)` y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector `a` en paralelo sin usar directivas de trabajo compartido .

RESPUESTA:

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int main(int argc, char **argv) {
  int i, n=20, a[n],suma=10;
  if(argc < 2) {
    fprintf(stderr,"Falta iteraciones\n");
    exit(-1);
  }
  n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d",n);}
  for (i=0; i<n; i++)
   a[i] = i;
  #pragma omp parallel for reduction(+:suma)
    for (i=0; i<n; i++)
     suma += a[i];
  printf("Tras 'parallel' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$./reduction-clauseEjer6.eje 5 useEjer6.c
Tras 'parallel' suma=20
```

Resto de ejercicios

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1 (implemente una versión para variables globales y otra para variables dinámicas, use una de estas versiones en los siguientes ejercicios):

$$v2 = M \bullet v1;\ \ v2(i) = \sum_{k=0}^{N-1} M(i,k) \bullet v(k),\ \ i = 0,\ldots N-1$$

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector 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 calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CAPTURA CÓDIGO FUENTE: `pmv-secuencial.c`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif
int main(int argc, char** argv){


  int i, j, N;
  if(argc < 2) {
    fprintf(stderr,"Falta tamaño\n");
    exit(-1);
  }
  N = atoi(argv[1]);
    double **m, *v1, *v2;
    double t1, t2, total;
    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
    m =(double **) malloc(N*sizeof(double *));
    if ( (v1==NULL) || (v2==NULL) || (m==NULL)){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
    }
    for (i = 0; i < N; i++){
      m[i] = malloc(N * sizeof(double));
      if ( m[i] == NULL ){
      printf("Error en la reserva de espacio para los vectores\n");
      exit(-2);
      }
      for(j = 0; j < N; j++){
        m[i][j] = 5.0;
      }
    }

    for(i = 0; i < N; i++){
      v1[i] = 1.0;
      v2[i] = 0;
    }
  t1 = omp_get_wtime();
    for(i = 0; i < N; i++)
      for(j = 0; j < N; j++)
        v2[i] += m[i][j] * v1[i];
  t2 = omp_get_wtime();
  total = t2 - t1;
  if(N <= 11){
    printf("Tiempo(seg.):%11.9f\t  Tamaño Vectores:%u\n",total,N);
      for(i=0; i<N; i++)
        printf("V2[%d] = %f \n",i,v2[i]);
  }else{
    printf("Tiempo(seg.):%11.9f\t Tamaño Vectores:%u\t V2[0] = %f V2[%d] = %f \n",total,N,v2[0],N-1,v2[N-1]);
  }
  free(v2); free(v1);
  for(i = 0; i < N; i++)
    free(m[i]);
  free(m);

}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$./ejer8.eje 8
Tiempo(seg.):0.000001000          Tamaño Vectores:8
V2[0] = 40.000000
V2[1] = 40.000000
V2[2] = 40.000000
V2[3] = 40.000000
V2[4] = 40.000000
V2[5] = 40.000000
V2[6] = 40.000000
V2[7] = 40.000000
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$./ejer8.eje 5
Tiempo(seg.):0.000001000          Tamaño Vectores:5
V2[0] = 25.000000
V2[1] = 25.000000
V2[2] = 25.000000
V2[3] = 25.000000
V2[4] = 25.000000
```

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior usando la directiva `for` . Debe implementar dos versiones del código (consulte la lección 5/Tema 2):

    a. una primera que paralelice el bucle que recorre las filas de la matriz y

    b. una segunda que paralelice el bucle que recorre las columnas.

    Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula `reduction`**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:

    - Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
    - Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

    NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada; (2) se debe inicializar la matriz y el vector 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 que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CAPTURA CÓDIGO FUENTE : `pmv-OpenMP-a.c`

```
#pragma omp parallel for private(i,j)
  for (i = 0; i < N; i++){
    m[i] = malloc(N * sizeof(double));
    if ( m[i] == NULL ){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
    }
    for(j = 0; j < N; j++){
      m[i][j] = 5.0;
    }
  }


for(i = 0; i < N; i++){
  v1[i] = 1.0;
  v2[i] = 0;
}
t1 = omp_get_wtime();
#pragma omp parallel for private(i,j,resultado)
for(i = 0; i < N; i++){
  resultado = 0;
  for(j = 0; j < N; j++)
    resultado += m[i][j];
  v2[i] += resultado * v1[i];
}
t2 = omp_get_wtime();
```

CAPTURA CÓDIGO FUENTE: `pmv-OpenMP-b.c`

```
    for (i = 0; i < N; i++){
      m[i] = malloc(N * sizeof(double));
      if ( m[i] == NULL ){
      printf("Error en la reserva de espacio para los vectores\n");
      exit(-2);
      }
      #pragma omp parallel for
      for(j = 0; j < N; j++){
        m[i][j] = 5.0;
      }
    }


  for(i = 0; i < N; i++){
    v1[i] = 1.0;
    v2[i] = 0;
  }
  t1 = omp_get_wtime();

  for(i = 0; i < N; i++){
    #pragma omp parallel
    resultado = 0;
    #pragma omp for
    for(j = 0; j < N; j++)
      resultado += m[i][j] * v1[j];

    #pragma omp critical
    v2[i] += resultado ;
  }
  t2 = omp_get_wtime();
```

RESPUESTA: Me dio error a la hora de inicializar las matrices que soluciones poniendo en privado tanto la i como la j.

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$./ejer9-a.eje 5
Tiempo(seg.):0.000172000          Tamaño Vectores:5
V2[0] = 25.000000
V2[1] = 25.000000
V2[2] = 25.000000
V2[3] = 25.000000
V2[4] = 25.000000
```

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
$./ejer9-b.eje 5
Tiempo(seg.):0.000013000          Tamaño Vectores:5
V2[0] = 25.000000
V2[1] = 25.000000
V2[2] = 25.000000
V2[3] = 25.000000
V2[4] = 25.000000
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-16 Monday
```

10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:
    - Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
    - Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

CAPTURA CÓDIGO FUENTE: `pmv-OpenmMP-reduction.c`

```
for(i = 0; i < N; i++){

    double resultado = 0;
    #pragma omp parallel for reduction(+:resultado)
    for(j = 0; j < N; j++)
      resultado += m[i][j] * v1[j];

    v2[i] += resultado ;
}
```

RESPUESTA:

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-17 Tuesday
$./ejer10.eje 5
.Tiempo(seg.):0.000010000          Tamaño Vectores:5
V2[0] = 25.000000
V2[1] = 25.000000
V2[2] = 25.000000
V2[3] = 25.000000
V2[4] = 25.000000
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica2/
Codigos] 2018-04-17 Tuesday
```

11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar –O2 al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURAS DE PANTALLA (que justifique el código elegido):
Capturas de mi pc

```
Num cores = 1
9-a
Tiempo(seg.):1.781460000        Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):18.382933000       Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):1.776752000        Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):16.209337000       Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.568400000        Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):9.306426000        Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
Num cores = 2
9-a
Tiempo(seg.):0.894372000        Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):9.100007000        Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):1.791229000        Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):15.752641000       Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.308723000        Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):6.236147000        Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
```

```
Num cores = 3
9-a
Tiempo(seg.):0.610589000         Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):7.832427000         Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):1.814462000         Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):13.509553000        Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.222689000         Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):8.754550000         Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
Num cores = 4
9-a
Tiempo(seg.):0.467330000         Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):6.675619000         Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):1.872784000         Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):14.900917000        Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.176615000         Tamaño Vectores:15000   V2[0] = 75000.0
00000 V2[14999] = 75000.000000
Tiempo(seg.):7.885072000         Tamaño Vectores:30000   V2[0] = 150000.
000000 V2[29999] = 150000.000000
```

Caputras atcgrid

```
$cat STDIN.o74090
Num cores = 1
9-a
Tiempo(seg.):0.972926310         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):3.405013252         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.968857639         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):3.419982498         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.358859780         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):1.493193823         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
Num cores = 2
9-a
Tiempo(seg.):0.489748043         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):2.199621169         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.488986026         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):2.316597681         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.210340803         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):1.192478689         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
Num cores = 3
9-a
Tiempo(seg.):0.325231529         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):1.433115488         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.309702475         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):1.455958991         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.164378283         Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.791982216         Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
Num cores = 4
```

```
Num cores = 4
9-a
Tiempo(seg.):0.245649796        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):1.112555492        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.243899045        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):1.108098744        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.139415651        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.688251907        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
Num cores = 5
9-a
Tiempo(seg.):0.195224196        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.861183089        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.195830050        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.875239477        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.156044853        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.570421765        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
Num cores = 6
9-a
Tiempo(seg.):0.166385802        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):1.042076656        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.165269745        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.799434516        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.130799683        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.546885673        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
```

```
Num cores = 10
9-a
Tiempo(seg.):0.102310276        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.427538152        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.100921392        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.423312819        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.129885486        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.543530554        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
Num cores = 11
9-a
Tiempo(seg.):0.092430844        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.473746663        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.098798654        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.387599419        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.149220153        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.622454079        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
Num cores = 12
9-a
Tiempo(seg.):0.085682151        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.387063839        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
9-b
Tiempo(seg.):0.090523656        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.383468572        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
10
Tiempo(seg.):0.150609330        Tamaño Vectores:15000   V2[0] = 75000.000000 V2[14999] = 75000.000000
Tiempo(seg.):0.621135714        Tamaño Vectores:30000   V2[0] = 150000.000000 V2[29999] = 150000.000000
```

TABLA Y GRÁFICA (por *ejemplo* para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 30000 y 100000, y otro entre 5000 y 30000):

PC tamaño 15000

	1 núcleo	2 núcleo	3 núcleo	4 núcleo
9-a	1.781460000	0.894372000	0.610589000	0.467330000
9-b	1.776752000	1.791229000	1.814462000	1.872784000
10	0.568400000	0.308723000	0.222689000	0.176615000

PC 30000

	1 núcleo	2 núcleo	3 núcleo	4 núcleo
9-a	18.382933000	9.100007000	7.832427000	6.675619000
9-b	16.209337000	15.752641000	13.509553000	14.900917000
10	9.306426000	6.236147000	8.754550000	7.885072000

ATCGRID 15000

	2	4	6	8	10	12
9-a	0.48974804	0.24564979	0.16638580	0.12374200	0.10231027	0.08568215
9-b	0.48898602	0.24389904	0.16526974	0.12885729	0.10092139	0.09052365
10	0.21034080	0.13941565	0.13079968	0.13507221	0.12988548	0.15060933

ATCGRID 30000

	2	4	6	8	10	12
9-a	2.19962116	1.11255549	1.04207665	0.54409360	0.42753815	0.38706383
9-b	2.31659768	1.10809874	0.79943451	0.53662613	0.42331281	0.38346857
10	1.19247868	0.68825190	0.54688567	0.52670343	0.62245407	0.62113571

COMENTARIOS SOBRE LOS RESULTADOS:

En mi ordenador el mas rápido con tamaño de 15000 es el ejercicio 10 en cambio con tamaño algo mas grande (30000) el ejercicio 9a y el 10 se igualan mucho. En el pc se va disminuyendo el tiempo conforme se aumentan procesadores.

En atcgrid en tamaño 15000 el mas rápido con menos de 6 núcleos es el ejercicio 10 y con mas núcleos los ejercicios 9 a y b tienden a ser mas rápidos, pasando lo mismo con el tamaño 30000,