Grai2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Elena Ortiz Moreno
Grupo de prácticas y profesor de prácticas: B2 Niceto Luque Sola
Fecha de entrega: 22-04-21
Fecha evaluación en clase: 23-04-21

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

Ejercicios basados en los ejemplos del seminario práctico

1. **(a)** Añadir la cláusula `default(none)` a la directiva `parallel` del ejemplo del seminario `shared-clause.c`? ¿Qué ocurre? ¿A qué se debe? **(b)** Resolver el problema generado sin eliminar `default(none)`. Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)

RESPUESTA: Que no compila porque hay que incluir en la directiva shared la variable n a parte de la variable a

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 default(none) shared(a, n)
  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:

Error al intentar compilar el código sin la variable n :

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer1$ gcc -fopenmp -O2 shared-clause.c -o shared-clause
shared-clause.c: In function 'main':
shared-clause.c:15:10: error: 'n' not specified in enclosing 'parallel'
  #pragma omp parallel for default(none) shared(a)
          ^~~
shared-clause.c:15:10: error: enclosing 'parallel'
```

Compilación y ejecución del código modificado añadiendo n:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer1$ gcc -fopenmp -O2 shared-clauseModificado.c -o share
d-clauseModificado
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer1$ ./shared-clauseModificado
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
```

2. **(a)** Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel`. Inicializar `suma` dentro del `parallel` a un valor distinto de 0. Ejecutar varias veces el código ¿Qué imprime el código fuera del `parallel`? (mostrar lo que ocurre con una captura de pantalla) Razonar respuesta. **(b)** Modificar el código del apartado (a) para que se inicialice `suma` fuera del `parallel` en lugar de dentro ¿Qué ocurre? Comparar todo lo que imprime el código ahora con la salida en (a) (mostrar la salida con una captura de pantalla) Razonar respuesta.

(a) RESPUESTA: Si inicializo la variable suma fuera del parallel e imprimo el valor fuera, no tendrá en cuenta lo calculado dentro del parallel ya que es privada y tomará directamente el valor inicial (2).

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.c`

```c
#include <stdio.h>

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

int main()
{
  int i, n = 7;
  int a[n], suma = 2;

  for (i=0; i<n; i++)
    a[i] = i;

  #pragma omp parallel private(suma)
  {
    #pragma omp for
    for (i=0; i<n; i++)
    {
      suma = suma + a[i];
    }
  }

  printf("\n suma fuera del parallel:");
  for (i=0; i<n; i++) {
    printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
  }
  printf("\n* thread %d suma= %d", suma);

  printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer2$ gcc -fopenmp -O2 private-clauseModificado_a.c -o pr
ivate-clauseModificado_a
private-clauseModificado_a.c: In function ‘main’:
private-clauseModificado_a.c:30:31: warning: format ‘%d’ expects a matching ‘int’ argument [-Wformat=]
  printf("\n* thread %d suma= %d", suma);
                            ~^
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer2$ ./private-clauseModificado_a

 suma fuera del parallel:thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] /
thread 0 suma a[4] / thread 0 suma a[5] / thread 0 suma a[6] /
* thread 2 suma= 0
```

(b) RESPUESTA: Si inicializo la variable suma a un valor distinto de 0 dentro del parallel y el resultado lo imprimo fuera, el resultado no será el correcto, ya que esta variable es privada. Se imprimirá un valor aleatorio o basura, ya que fuera del parallel no se ha inicializado.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.c`

```c
#include <stdio.h>

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

int main()
{
  int i, n = 7;
  int a[n], suma;

  for (i=0; i<n; i++)
    a[i] = i;

  #pragma omp parallel private(suma)
  {
    suma = 2;
    #pragma omp for
    for (i=0; i<n; i++)
    {
      suma = suma + a[i];
    }
  }

  printf("\n suma fuera del parallel:");
  for (i=0; i<n; i++) {
    printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
  }
  printf("\n* thread %d suma= %d", suma);

  printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer2$ gcc -fopenmp -O2 private-clauseModificado_b.c -o pr
ivate-clauseModificado_b
private-clauseModificado_b.c: In function ‘main’:
private-clauseModificado_b.c:31:31: warning: format ‘%d’ expects a matching ‘int’ argument [-Wformat=]
  printf("\n* thread %d suma= %d", suma);
                            ~^
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer2$ ./private-clauseModificado_b

 suma fuera del parallel:thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] /
thread 0 suma a[4] / thread 0 suma a[5] / thread 0 suma a[6] /
* thread 0 suma= 0
```

3. **(a)** Eliminar la cláusula `private(suma)` en `private-clause.c`. Ejecutar el código resultante. ¿Qué ocurre? **(b)** ¿A qué es debido?

RESPUESTA: Es debido a que la variable suma pasa a ser global y todas las hebras están escribiendo sobre la misma variable. En el caso anterior (suma es privada) cada hebra utiliza su variable suma y luego las juntan en una.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```c
#include <stdio.h>

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

int main()
{
  int i, n = 7;
  int a[n], suma;

  for (i=0; i<n; i++)
    a[i] = i;

  #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:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer3$ gcc -fopenmp -O2 private-clauseModificado2.c -o pri
vate-clauseModificado2
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer3$ ./private-clauseModificado2
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] / thre
ad 0 suma a[0] / thread 0 suma a[1] /
* thread 3 suma= 9
* thread 1 suma= 9
* thread 2 suma= 9
* thread 0 suma= 9
```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el tamaño del vector ¿podría imprimir el código otro valor? Razonar respuesta (añadir capturas de pantalla que muestren lo que ocurre).

(a) RESPUESTA:

Lo que ocurre es que al ser el vector más grande, se acumulan más sumas en cada iteración y aparecen números más grandes.

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer4$ gcc -fopenmp -O2 firstprivate-clauseMod.c -o firstp
rivate-clauseMod
firstprivate-clauseMod.c:8:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main() {
 ^~~~
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer4$ ./firstprivate-clauseMod
 thread 1 suma a[3] suma=3
 thread 1 suma a[4] suma=7
 thread 1 suma a[5] suma=12
 thread 3 suma a[8] suma=8
 thread 3 suma a[9] suma=17
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 2 suma a[6] suma=6
 thread 2 suma a[7] suma=13

Fuera de la construcción parallel suma=0
```

(b) RESPUESTA:

En este caso los números son más pequeños al serlo también el vector. Independientemente de esto, la suma final es 0 en los dos casos ya que está inicializada fuera del parallel y no se modifica en ningún momento.

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer4$ gcc -fopenmp -O2 firstprivate-clause.c -o firstpriv
ate-clause
firstprivate-clause.c:8:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main() {
 ^~~~
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer4$ ./firstprivate-clause
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 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=0
```

5. **(a)** ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? **(b)** ¿A qué cree que es debido? (añadir una captura de pantalla que muestre lo que ocurre)

RESPUESTA: No se inicializa el valor de a, y en consecuencia imprime basura. Esto ocurre porque sin la cláusula copyprivate(a), el valor de inicialización no se copia en todas las hebras mediante difusión, que es lo que hace esta cláusula.

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("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:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer5$ gcc -fopenmp -O2 copyprivate-clauseModificado.c -o
 copyprivate-clauseModificado
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer5$ ./copyprivate-clauseModificado

Introduce valor de inicialización a: 12

Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 0        b[1] = 0        b[2] = 0        b[3] = 12       b[4] = 12       b[5] = 0 b[6] = 0        b[7]
 = 0        b[8] = 0
```

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado (añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA: A la suma anterior se le añaden 10, que es el valor inicial de suma.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.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:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer6$ gcc -fopenmp -O2 reduction-clauseModificado.c -o r
eduction-clauseModificado
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer6$ ./reduction-clauseModificado 10
Tras 'parallel' suma=55
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer6$ ./reduction-clauseModificado 20
Tras 'parallel' suma=200
```

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 añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA: Si no vamos a utilizar la clausula reduction, obligatoriamente la variable suma tiene que ser compartida. Por último, para que las hebras se sincronicen correctamente en el acceso a la variable suma se utiliza la directiva atomic que no es de trabajo compartido.

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=0;

  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
    for (i=0; i<n; i++) suma += a[i];

  printf("Tras 'parallel' suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer7$ gcc -fopenmp -O2 reduction-clauseModificado7.c -o
reduction-clauseModificado7
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer7$ ./reduction-clauseModificado7 8
Tras 'parallel' suma=28
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer7$ ./reduction-clauseModificado7 11
Tras 'parallel' suma=31
```

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

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>
#include <time.h>
#include <malloc.h>

//Comentar uno
#define GLOBAL
//#define DINAMIC

#ifdef GLOBAL
  #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
  if(argc != 2){
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    struct timespec cgt1, cgt2;
    double ncgt;

    int N = atoi(argv[1]);

    #ifdef GLOBAL
      if(N > MAX) N = MAX;
        int matriz[N][N];
        int vector[N];
        int vector_resultado[N];
        printf("Ejecutado GLOBALMENTE\n");
    #endif

    #ifdef DINAMIC
        int **matriz, *vector, *vector_resultado;
        matriz = (int**) malloc(N * sizeof(int*));
        for(int i = 0; i < N; ++i)
            matriz[i] = (int*) malloc(N * sizeof(int));

        vector = (int*) malloc(N * sizeof(int));
        vector_resultado = (int*) malloc(N * sizeof(int));
        printf("Ejecutado DINAMICAMENTE\n");
    #endif

    for(int i = 0; i < N; ++i){
      vector[i] = i;
        for(int j = 0; j < N; ++j)
          matriz[i][j] = i + j;
  }
```

```
  clock_gettime(CLOCK_REALTIME, &cgt2);
  ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) + (double) (cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9);

  printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
  if(N < 11)
    for(int i = 0; i < N; i++){
      printf("VECTOR_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
      printf("\n");
    }
  else{
    printf("VECTOR_RESULTADO[0] = %d  ", vector_resultado[0]);
    printf("VECTOR_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
    printf("\n");
  }

  #ifdef DINAMIC
  for(int i = 0; i < N; i++)
    free(matriz[i]);

  free(matriz); free(vector); free(vector_resultado);
  #endif

  return 0;
}
```

CAPTURAS DE PANTALLA:

DINÁMICO:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer8$ gcc -fopenmp -O2 pmv-secuencial.c -o pmv-secuencial
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer8$ ./pmv-secuencial 8
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.000000725        / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer8$ ./pmv-secuencial 11
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.000001171        / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385  VECTOR_RESULTADO[10] = 935
```

GLOBAL:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer8$ gcc -fopenmp -O2 pmv-secuencial.c -o pmv-secuencial
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer8$ ./pmv-secuencial 8
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000000772        / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer8$ ./pmv-secuencial 11
Ejecutado GLOBALMENTE
Tiempo(seg.): 0.000000897        / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385  VECTOR_RESULTADO[10] = 935
```

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`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include <omp.h>

// Comentar uno
//#define GLOBAL
#define DINAMIC

#ifdef GLOBAL
  #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
  if(argc != 2){
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    double cgt1, cgt2;
    double ncgt;

    int N = atoi(argv[1]);

    #ifdef GLOBAL
      if(N > MAX) N = MAX;
        int matriz[N][N];
        int vector[N];
        int vector_resultado[N];
        printf("Ejecutado GLOBALMENTE\n");
    #endif

    #ifdef DINAMIC
        int **matriz, *vector, *vector_resultado;
        matriz = (int**) malloc(N * sizeof(int*));
        for(int i = 0; i < N; ++i)
            matriz[i] = (int*) malloc(N * sizeof(int));

        vector = (int*) malloc(N * sizeof(int));
        vector_resultado = (int*) malloc(N * sizeof(int));
        printf("Ejecutado DINAMICAMENTE\n");
    #endif
```

```
      #pragma omp parallel for
      for(int i = 0; i < N; ++i){
        vector[i] = i;
        #pragma omp parallel for
          for(int j = 0; j < N; ++j)
            matriz[i][j] = i + j;
    }


  cgt1 = omp_get_wtime();
  //Calculamos el vector resultado
  #pragma omp parallel for
  for(int i = 0; i < N; i++){
    int suma = 0;
    for(int j = 0; j < N; j++)
      suma += matriz[i][j] * vector[j];

    vector_resultado[i] = suma;
  }

  cgt2 = omp_get_wtime();
  ncgt = cgt2-cgt1;

  printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
  if(N < 11)
    for(int i = 0; i < N; i++){
      printf("VECTOR_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
      printf("\n");
    }
  else{
    printf("VECTOR_RESULTADO[0] = %d  ", vector_resultado[0]);
    printf("VECTOR_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
    printf("\n");
  }

  #ifdef DINAMIC
  for(int i = 0; i < N; i++)
    free(matriz[i]);

  free(matriz); free(vector); free(vector_resultado);
  #endif

  return 0;
}
```

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include <omp.h>

// Comentar uno
//#define GLOBAL
#define DINAMIC

#ifdef GLOBAL
  #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
  if(argc != 2){
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    double cgt1, cgt2;
    double ncgt;

    int N = atoi(argv[1]);

    #ifdef GLOBAL
      if(N > MAX) N = MAX;
        int matriz[N][N];
        int vector[N];
        int vector_resultado[N];
        printf("Ejecutado GLOBALMENTE\n");
    #endif

    #ifdef DINAMIC
        int **matriz, *vector, *vector_resultado;
        matriz = (int**) malloc(N * sizeof(int*));
        for(int i = 0; i < N; ++i)
            matriz[i] = (int*) malloc(N * sizeof(int));

        vector = (int*) malloc(N * sizeof(int));
        vector_resultado = (int*) malloc(N * sizeof(int));
        printf("Ejecutado DINAMICAMENTE\n");
    #endif
```

```c
    #pragma omp parallel for
    for(int i = 0; i < N; ++i){
      vector[i] = i;
      #pragma omp parallel for
        for(int j = 0; j < N; ++j)
          matriz[i][j] = i + j;
  }

  cgt1 = omp_get_wtime();
  //Calculamos el vector resultado
  for(int i = 0; i < N; i++){
    int suma = 0;
    #pragma omp parallel for
    for(int j = 0; j < N; j++)
      #pragma omp atomic
      suma += matriz[i][j] * vector[j];

    vector_resultado[i] = suma;
  }

  cgt2 = omp_get_wtime();
  ncgt = cgt2-cgt1;

  printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
  if(N < 11)
    for(int i = 0; i < N; i++){
      printf("VECTOR_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
      printf("\n");
    }
  else{
    printf("VECTOR_RESULTADO[0] = %d  ", vector_resultado[0]);
    printf("VECTOR_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
    printf("\n");
  }

  #ifdef DINAMIC
  for(int i = 0; i < N; i++)
    free(matriz[i]);

  free(matriz); free(vector); free(vector_resultado);
  #endif

  return 0;
}
```

RESPUESTA: En el apartado b, al no inicializar suma a 0 antes del for, se suma el contenido de las posiciones anteriores del vector con el actual. Los resultados varían debido a una falta de sincronización entre hebras al insertar en el vector los resultados que obtiene cada una.

CAPTURAS DE PANTALLA:

Compilación y ejecución versión a:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer9$ gcc -fopenmp -O2 pmv-OpenMP-a.c -o pmv-OpenMP-a
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer9$ ./pmv-OpenMP-a 8
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.000007196       / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer9$ ./pmv-OpenMP-a 11
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.014756816       / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385  VECTOR_RESULTADO[10] = 935
```

Compilación y ejecución versión b:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer9$ gcc -fopenmp -O2 pmv-OpenMP-b.c -o pmv-OpenMP-b
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer9$ ./pmv-OpenMP-b 8
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.000057664       / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer9$ ./pmv-OpenMP-b 11
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.000066011       / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385  VECTOR_RESULTADO[10] = 935
```

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`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include <omp.h>

// Comentar uno
//#define GLOBAL
#define DINAMIC

#ifdef GLOBAL
  #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
  if(argc != 2){
        printf("Error de argumentos %s", argv[0]);
        return(EXIT_FAILURE);
    }

    double cgt1, cgt2;
    double ncgt;

    int N = atoi(argv[1]);

    #ifdef GLOBAL
      if(N > MAX) N = MAX;
        int matriz[N][N];
        int vector[N];
        int vector_resultado[N];
        printf("Ejecutado GLOBALMENTE\n");
    #endif

    #ifdef DINAMIC
        int **matriz, *vector, *vector_resultado;
        matriz = (int**) malloc(N * sizeof(int*));
        for(int i = 0; i < N; ++i)
            matriz[i] = (int*) malloc(N * sizeof(int));

        vector = (int*) malloc(N * sizeof(int));
        vector_resultado = (int*) malloc(N * sizeof(int));
        printf("Ejecutado DINAMICAMENTE\n");
    #endif
```

```
    #pragma omp parallel for
    for(int i = 0; i < N; ++i){
      vector[i] = i;
      #pragma omp parallel for
        for(int j = 0; j < N; ++j)
          matriz[i][j] = i + j;
  }

  cgt1 = omp_get_wtime();
  //Calculamos el vector resultado
  for(int i = 0; i < N; i++){
    int suma = 0;
    #pragma omp parallel for reduction(+:suma)
    for(int j = 0; j < N; j++)
      suma += matriz[i][j] * vector[j];

    vector_resultado[i] = suma;
  }

  cgt2 = omp_get_wtime();
  ncgt = cgt2-cgt1;

  printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
  if(N < 11)
    for(int i = 0; i < N; i++){
      printf("VECTOR_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
      printf("\n");
    }
  else{
    printf("VECTOR_RESULTADO[0] = %d  ", vector_resultado[0]);
    printf("VECTOR_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
    printf("\n");
  }

  #ifdef DINAMIC
  for(int i = 0; i < N; i++)
    free(matriz[i]);

  free(matriz); free(vector); free(vector_resultado);
  #endif

    return 0;
}
```

RESPUESTA: He juntado las claúsulas parallel y for, y sustituido la directiva atomic por la cláusula reduction para hacer la suma.

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer10$ gcc -fopenmp -O2 pmv-OpenMP-reduction.c -o pmv-Ope
nMP-reduction
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer10$ ./pmv-OpenMP-reduction 8
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.000045192        / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140
VECTOR_RESULTADO[1] = 168
VECTOR_RESULTADO[2] = 196
VECTOR_RESULTADO[3] = 224
VECTOR_RESULTADO[4] = 252
VECTOR_RESULTADO[5] = 280
VECTOR_RESULTADO[6] = 308
VECTOR_RESULTADO[7] = 336
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer10$ ./pmv-OpenMP-reduction 11
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.000063298        / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385  VECTOR_RESULTADO[10] = 935
```

11. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid4, en uno de los nodos de la cola ac 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):

Versión a:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer11$ gcc -fopenmp -O2 pmv-OpenMP-a.c -o pmv-OpenMP-a
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer11$ ./pmv-OpenMP-a 7000
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.017594451        / Tamaño vectores: 7000
VECTOR_RESULTADO[0] = -1655282492  VECTOR_RESULTADO[6999] = -2002970832
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer11$ ./pmv-OpenMP-a 20000
Ejecutado DINAMICAMENTE
Tiempo(seg.): 0.132958470        / Tamaño vectores: 20000
VECTOR_RESULTADO[0] = -708020816  VECTOR_RESULTADO[19999] = 277436608
```

Versión b:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer11$ gcc -fopenmp -O2 pmv-OpenMP-b.c -o pmv-OpenMP-b
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer11$ ./pmv-OpenMP-b 7000
Ejecutado DINAMICAMENTE
Tiempo(seg.): 1.557654626        / Tamaño vectores: 7000
VECTOR_RESULTADO[0] = -1655282492  VECTOR_RESULTADO[6999] = -2002970832
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP2/ejer11$ ./pmv-OpenMP-b 20000
Ejecutado DINAMICAMENTE
Tiempo(seg.): 17.120899594       / Tamaño vectores: 20000
VECTOR_RESULTADO[0] = -708020816  VECTOR_RESULTADO[19999] = 277436608
```

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:

He elegido la versión a, ya que tarda mucho menos tiempo.

```
#pragma omp parallel for
for(int i = 0; i < N; i++){
  int suma = 0;
  for(int j = 0; j < N; j++)
    suma += matriz[i][j] * vector[j];

  vector_resultado[i] = suma;
}

cgt2 = omp_get_wtime();
```

```
55    for(int i = 0; i < N; i++){
56      int suma = 0;
57      #pragma omp parallel for
58      for(int j = 0; j < N; j++)
59        #pragma omp atomic
60        suma += matriz[i][j] * vector[j];
61
62      vector_resultado[i] = suma;
63    }
64
```

Esto se debe a que en la versión a, donde calculamos el vector resultado, el parallel for engloba los dos for, inicializando así la suma dentro. Por ello, en este trozo de código no se ejecuta nada de manera secuencial, y cada hebra realiza los for.

CAPTURA DE PANTALLA del script `pmv-OpenmMP-script.sh`

Script.sh (PC):

```
#!/bin/bash
for((i=1; i<=32;i=i+1))
do
  OMP_NUM_THREADS=$i
  export OMP_NUM_THREADS
  ./$1 7000 >> datosPC7000
done

for((i=1; i<=32;i=i+1))
do
  OMP_NUM_THREADS=$i
  export OMP_NUM_THREADS
  ./$1 20000 >> datosPC20000
done
```

script_atcgrid.sh:

```
#!/bin/bash
# Órdenes para el sistema de colas:
# 1. Asigna al trabajo un nombre
# SBATCH --nombre-trabajo = ATCGRID
# 2. Asignar el trabajo a una cola (partición)
# SBATCH --partición = ac
# 2. Asignar el trabajo a un account
# SBATCH --cuenta = ac

# Obtener información de las variables del entorno del sistema de colas:
echo  " Id. usuario del trabajo: $ SLURM_JOB_USER "
echo  " Id. del trabajo: $ SLURM_JOBID "
echo  " Nombre del trabajo especificado por usuario: $ SLURM_JOB_NAME "
echo  " Directorio de trabajo (en el que se ejecuta el script):
$ SLURM_SUBMIT_DIR "
echo  " Cola: $ SLURM_JOB_PARTITION "
echo  " Nodo que ejecuta este trabajo: $ SLURM_SUBMIT_HOST "
echo  " No de nodos asignados al trabajo: $ SLURM_JOB_NUM_NODES "
echo  " Nodos asignados al trabajo: $ SLURM_JOB_NODELIST "
echo  " CPU por nodo: $ SLURM_JOB_CPUS_PER_NODE "

for((i=1; i<=32;i=i+1))
do
  OMP_NUM_THREADS=$i
  export OMP_NUM_THREADS
  srun -p ac ./$1 7000 >> datosATCGRID7000
done

for((i=1; i<=32;i=i+1))
do
  OMP_NUM_THREADS=$i
  export OMP_NUM_THREADS
  srun -p ac ./$1 20000 >> datosATCGRID20000
done
```

script_atcgrid4.sh:

```
#!/bin/bash
# Órdenes para el sistema de colas:
# 1. Asigna al trabajo un nombre
# SBATCH --nombre-trabajo = ATCGRID4
# 2. Asignar el trabajo a una cola (partición)
# SBATCH --partición = ac
# 2. Asignar el trabajo a un account
# SBATCH --cuenta = ac

# Obtener información de las variables del entorno del sistema de colas:
echo  " Id. usuario del trabajo: $ SLURM_JOB_USER "
echo  " Id. del trabajo: $ SLURM_JOBID "
echo  " Nombre del trabajo especificado por usuario: $ SLURM_JOB_NAME "
echo  " Directorio de trabajo (en el que se ejecuta el script):
$ SLURM_SUBMIT_DIR "
echo  " Cola: $ SLURM_JOB_PARTITION "
echo  " Nodo que ejecuta este trabajo: $ SLURM_SUBMIT_HOST "
echo  " No de nodos asignados al trabajo: $ SLURM_JOB_NUM_NODES "
echo  " Nodos asignados al trabajo: $ SLURM_JOB_NODELIST "
echo  " CPU por nodo: $ SLURM_JOB_CPUS_PER_NODE "

for((i=1; i<=32;i=i+1))
do
  OMP_NUM_THREADS=$i
  export OMP_NUM_THREADS
  srun -p ac4 ./$1 7000 >> datosATCGRID47000
done

for((i=1; i<=32;i=i+1))
do
  OMP_NUM_THREADS=$i
  export OMP_NUM_THREADS
  srun -p ac4 ./$1 20000 >> datosATCGRID420000
done
```

CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

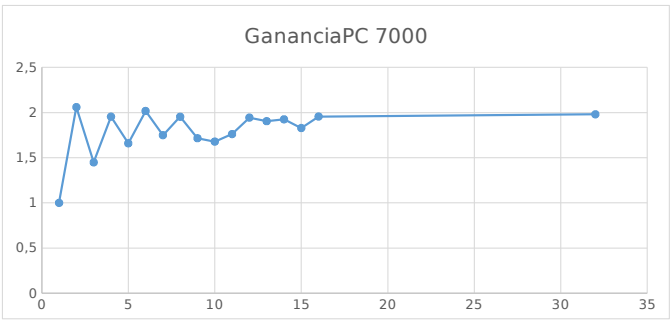
```
[b2estudiante27@atcgrid BP2]$ ./script_atcgrid.sh pmv-OpenMP-a
 Id. usuario del trabajo: $ SLURM_JOB_USER
 Id. del trabajo: $ SLURM_JOBID
 Nombre del trabajo especificado por usuario: $ SLURM_JOB_NAME
 Directorio de trabajo (en el que se ejecuta el script):
$ SLURM_SUBMIT_DIR
 Cola: $ SLURM_JOB_PARTITION
 Nodo que ejecuta este trabajo: $ SLURM_SUBMIT_HOST
 No de nodos asignados al trabajo: $ SLURM_JOB_NUM_NODES
 Nodos asignados al trabajo: $ SLURM_JOB_NODELIST
 CPU por nodo: $ SLURM_JOB_CPUS_PER_NODE
```

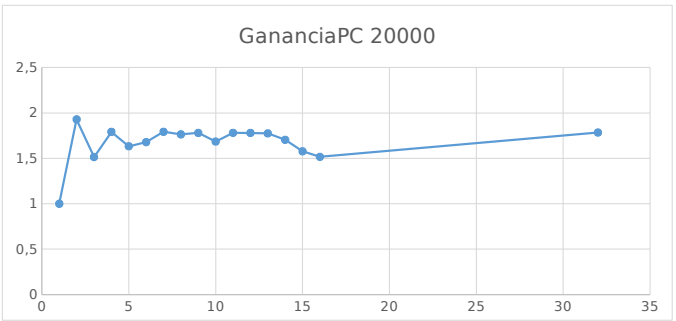
```
[b2estudiante27@atcgrid BP2]$ ./script_atcgrid4.sh pmv-OpenMP-a
 Id. usuario del trabajo: $ SLURM_JOB_USER
 Id. del trabajo: $ SLURM_JOBID
 Nombre del trabajo especificado por usuario: $ SLURM_JOB_NAME
 Directorio de trabajo (en el que se ejecuta el script):
$ SLURM_SUBMIT_DIR
 Cola: $ SLURM_JOB_PARTITION
 Nodo que ejecuta este trabajo: $ SLURM_SUBMIT_HOST
 No de nodos asignados al trabajo: $ SLURM_JOB_NUM_NODES
 Nodos asignados al trabajo: $ SLURM_JOB_NODELIST
 CPU por nodo: $ SLURM_JOB_CPUS_PER_NODE
```

TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia):

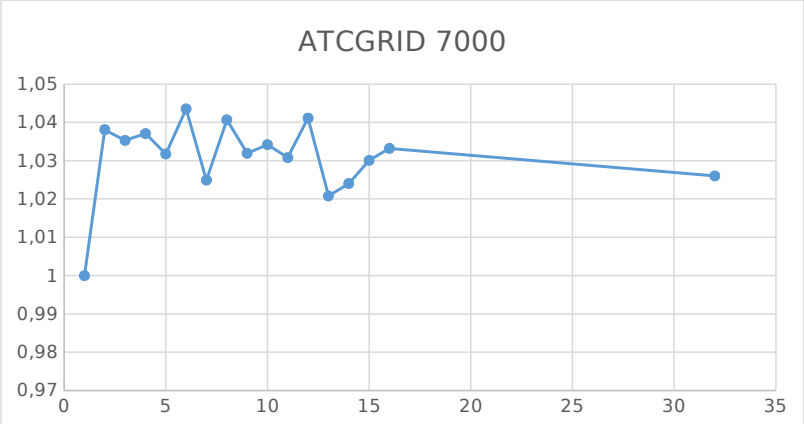
PC LOCAL

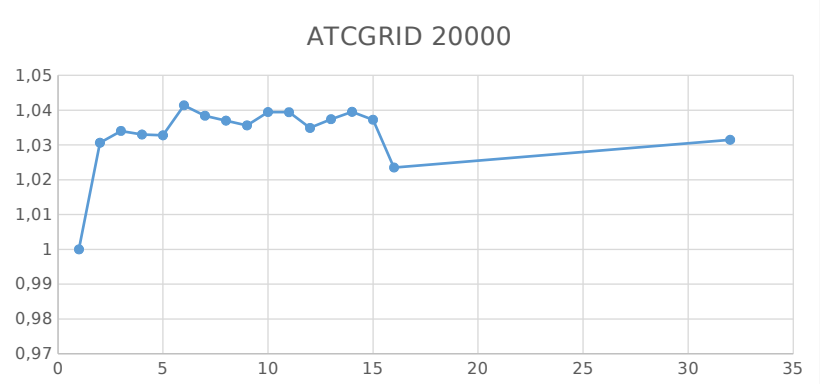
Tamaño	Version A	n threads	Ganancia
7000	0,032343938	1	1
7000	0,015704751	2	2,0595002
7000	0,022332248	3	1,4483064
7000	0,016548684	4	1,9544719
7000	0,019494018	5	1,6591725
7000	0,01603881	6	2,0166046
7000	0,01849296	7	1,7489865
7000	0,016559987	8	1,9531379
7000	0,018853737	9	1,7155187
7000	0,019275892	10	1,6779477
7000	0,018363405	11	1,7613257
7000	0,016637265	12	1,9440658
7000	0,016985738	13	1,9041821
7000	0,016806563	14	1,9244826
7000	0,017691203	15	1,8282498
7000	0,016539987	16	1,9554996
7000	0,016334715	32	1,9800736
20000	0,233334357	1	1
20000	0,120931588	2	1,929474
20000	0,15410425	3	1,5141332
20000	0,130257043	4	1,7913377
20000	0,142906313	5	1,6327785
20000	0,139023055	6	1,6783861
20000	0,130172598	7	1,7924998
20000	0,132288004	8	1,7638361
20000	0,131073634	9	1,7801777
20000	0,138492618	10	1,6848144
20000	0,131039536	11	1,7806409
20000	0,131109519	12	1,7796904
20000	0,131468067	13	1,7748368
20000	0,136884031	14	1,7046134
20000	0,147898427	15	1,5776663
20000	0,153879607	16	1,5163436
20000	0,130766127	32	1,7843639



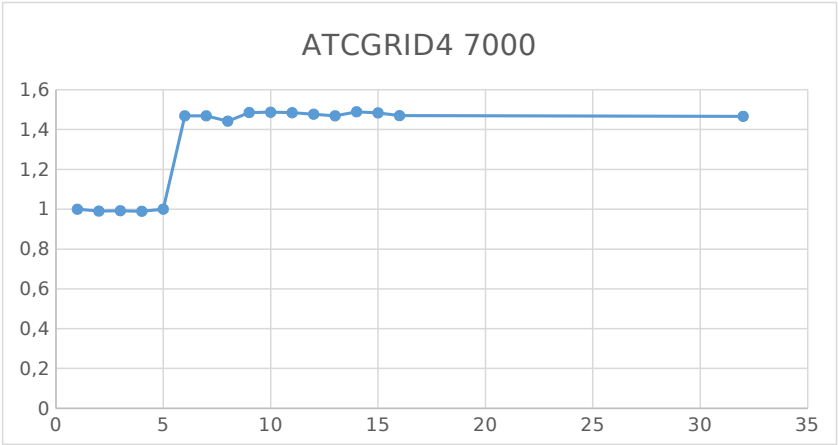


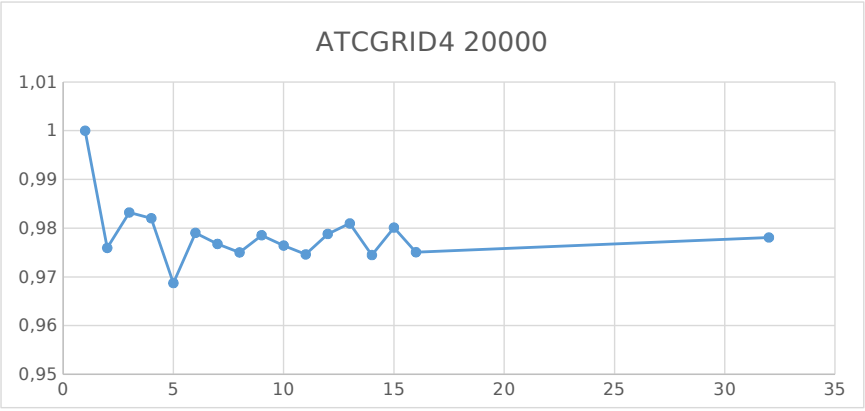
ATCGRID			
Tamaño	Version A	n threads	Ganancia
7000	0,065519	1	1
7000	0,063114	2	1,038099
7000	0,063286	3	1,035276
7000	0,063178	4	1,037059
7000	0,063503	5	1,031739
7000	0,062787	6	1,043516
7000	0,063926	7	1,024912
7000	0,06296	8	1,040651
7000	0,063493	9	1,031916
7000	0,063354	10	1,034174
7000	0,063562	11	1,030792
7000	0,062933	12	1,041091
7000	0,064185	13	1,020785
7000	0,063981	14	1,024031
7000	0,063606	15	1,030076
7000	0,063413	16	1,033204
7000	0,063858	32	1,026009
20000	0,532951	1	1
20000	0,517105	2	1,030644
20000	0,515408	3	1,034037
20000	0,515923	4	1,033005
20000	0,516038	5	1,032774
20000	0,511763	6	1,041403
20000	0,513242	7	1,038401
20000	0,513939	8	1,036993
20000	0,514613	9	1,035634
20000	0,512713	10	1,039471
20000	0,512736	11	1,039426
20000	0,514981	12	1,034895
20000	0,513733	13	1,037409
20000	0,51269	14	1,039519
20000	0,513813	15	1,037246
20000	0,520703	16	1,023521
20000	0,516685	32	1,031481





ATCGRID4			
Tamaño	Version A	n threads	Ganancia
7000	0,039898	1	1
7000	0,040286	2	0,990353
7000	0,040198	3	0,992522
7000	0,040313	4	0,989694
7000	0,039883	5	1,000359
7000	0,02716	6	1,468998
7000	0,027156	7	1,469179
7000	0,027662	8	1,442322
7000	0,026856	9	1,485588
7000	0,02683	10	1,487033
7000	0,026876	11	1,484521
7000	0,027006	12	1,477365
7000	0,027157	13	1,469127
7000	0,026787	14	1,489431
7000	0,026889	15	1,483761
7000	0,027143	16	1,4699
7000	0,027208	32	1,466397
20000	0,213176	1	1
20000	0,21843	2	0,975946
20000	0,216813	3	0,983224
20000	0,217077	4	0,982028
20000	0,220055	5	0,968738
20000	0,21774	6	0,979039
20000	0,218247	7	0,976764
20000	0,218637	8	0,975023
20000	0,217849	9	0,978548
20000	0,21832	10	0,976436
20000	0,218729	11	0,974614
20000	0,217788	12	0,978824
20000	0,217312	13	0,980968
20000	0,218759	14	0,97448
20000	0,217501	15	0,980116
20000	0,218626	16	0,97507
20000	0,217953	32	0,978085





COMENTARIOS SOBRE LOS RESULTADOS:

Después de varias ejecuciones, no consigo mejorar los resultados. En atcgrid4 se obtienen mejores resultados con una hebra que con varias. Sin embargo en atcgrid, a partir de 16 hebras ya no merece la pena seguir aumentando el número de estas.