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): Francisco Javier Bolívar Expósito
Grupo de prácticas y profesor de prácticas:D1 Francisco Barranco Expósito
Fecha de entrega:
Fecha evaluación en clase:

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. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? **(b)** Resuelva el problema generado sin eliminar `default(none)`.Añada el código con la modificación al cuaderno de prácticas. (Añada capturas de pantalla que muestren lo que ocurre)
   RESPUESTA: La variable n no está especificada si es compartida o privada, al tener la claúsula default(none) el código no compila.

   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) shared(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:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2] 2019-04-09 martes
$gcc -fopenmp ejer1/shared-clauseModificado.c
ejer1/shared-clauseModificado.c: En la función 'main':
ejer1/shared-clauseModificado.c:15:10: error: no se especificó 'n' en el 'parallel' que lo contiene
  #pragma omp parallel for default(none) shared(a)
          ^~~
ejer1/shared-clauseModificado.c:15:10: error: 'parallel' contenedora
```

2. Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel` e inicializar `suma` a un valor distinto de 0. Ejecute varias veces el código ¿Qué imprime el código fuera del parallel? (muéstrelo con una captura de pantalla) ¿Qué ocurre si en esta versión de `private-clause.c` se

inicia la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta (añada capturas de pantalla que muestren lo que ocurre). Añadir el código con las modificaciones al cuaderno de prácticas.

RESPUESTA: Suma es privada en la región parallel, por lo que cada hebra tendrá su propia copia de la variable suma que no saldrá fuera de la región. Por lo tanto, al inicializar suma solo dentro de la región parallel el programa imprime un valor basura. Cuando se inicia la variable suma fuera de la región parallel imprime el valor de inicialización.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```
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=10;
        #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\n", omp_get_thread_num(), suma);
}

int main()
{
    int i, n = 7;
    int a[n], suma = 10;

    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("thread %d suma a[%d] / ", omp_get_thread_num(), i);
        }
    }

    printf("\n* thread %d suma= %d\n", omp_get_thread_num(), suma);
}
```

CAPTURAS DE PANTALLA:

```
$gcc -fopenmp private-clauseModificado.c
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer2] 2019-04-18 jueves
$./a.out
thread 6 suma a[6] / thread 0 suma a[0] / thread 2 suma a[2] / thread 1 suma a[1] / thread 4 suma a[4] / thread 3 su
ma a[3] / thread 5 suma a[5] /
* thread 0 suma= 32613
```

```
$gcc -fopenmp private-clauseModificado.c
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer2] 2019-04-18 jueves
$./a.out
thread 6 suma a[6] / thread 0 suma a[0] / thread 1 suma a[1] / thread 3 suma a[3] / thread 4 suma a[4] / thread 5 su
ma a[5] / thread 2 suma a[2] /
* thread 0 suma= 10
```

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

   RESPUESTA: El valor de suma es el mismo para todas las hebras. Al eliminar la cláusula private la variable suma es compartida por todas las hebras, esto hace que todas tengan el mismo valor y que varias sumas al realizarse a la vez puedan perderse.

   CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```c
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:

```
$gcc -fopenmp private-clauseModificado3.c
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer3] 2019-04-18 jueves
$./a.out
thread 3 suma a[3] / thread 0 suma a[0] / thread 5 suma a[5] / thread 4 suma a[4] / thread 2 suma a[2] / thread 1 su
ma a[1] / thread 6 suma a[6] /
* thread 7 suma= 5
* thread 0 suma= 5
* thread 1 suma= 5
* thread 6 suma= 5
* thread 2 suma= 5
* thread 4 suma= 5
* thread 3 suma= 5
* thread 5 suma= 5
```

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 (añada capturas de pantalla que muestren lo que ocurre).

   RESPUESTA: El valor de suma que está inicializado a 0 se difunde debido a la cláusula firstprivate a la variable privada suma de cada hebra. Después, por la cláusula lastprivate, se copia fuera de la región parallel el valor de suma en la última ejecución secuencial, es decir, la última iteración del bucle que realiza suma = suma + a[6], siendo suma = 0 y a[6] = 6. Por lo tanto el código siempre imprime 6 fuera de la región parallel.

   CAPTURAS DE PANTALLA:

```
$gcc -fopenmp firstlastprivate-clause.c
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer4] 2019-04-18 jueves
$./a.out
 thread 0 suma a[0] suma=0
 thread 2 suma a[2] suma=2
 thread 1 suma a[1] suma=1
 thread 3 suma a[3] suma=3
 thread 5 suma a[5] suma=5
 thread 4 suma a[4] suma=4
 thread 6 suma a[6] suma=6

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? (añada una captura de pantalla que muestre lo que ocurre)

RESPUESTA: Solo la hebra que ha hecho la lectura guarda el valor de a. Al eliminar la cláusula copyprivate la variable a leída no se copia a la variables privadas del mismo nombre del resto de threads.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```c
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:

```
$gcc -fopenmp copyprivate-clauseModificado.c -o copyprivatemod
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer5] 2019-04-18 jueves
$./copyprivatemod

Introduce valor de inicialización a: 5

Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 5        b[1] = 5        b[2] = 0        b[3] = 0        b[4] = 0        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: Se usa la cláusula reduction para reducir a un único valor las sumas parciales. En las operaciones de suma se inicializan las variables privadas de cada thread a 0 y también se suma el valor de la variable suma global a todos los threads. Por lo tanto para 5 iteraciones vemos como se juntan las sumas de (0 + 1 + 2 + 3 + 4 = 10) que se realizan en el bucle más el valor al que habíamos inicializado suma (10) dando como resultado 10 + 10 = 20

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:

```
$gcc -fopenmp reduction-clauseModificado.c -o reductionclausemod
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer6] 2019-04-18 jueves
$./reductionclausemod 5
Tras 'parallel' suma=20
```

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: Establecemos el ámbito de la variables suma a compartido con “shared(suma)”. Para que las sumas parciales se acumulen sin condiciones de carrera usamos atomic.

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

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

CAPTURAS DE PANTALLA:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer7] 2019-04-18 jueves
$gcc -fopenmp reduction-clauseModificado7.c -o reductionclausemod7
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer7] 2019-04-18 jueves
$./reductionclausemod7 5
Tras 'parallel' suma=10
```

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, v2, 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 <stdlib.h>
#include <stdio.h>
#include <time.h>

//#define MATRIX_GLOBAL
#define MATRIX_DYNAMIC

#ifdef MATRIX_GLOBAL
#define MAX 4096

double m[MAX][MAX], v1[MAX], vf[MAX];
#endif
int main(int argc, char** argv)
{
    if (argc<2)
    {
        printf("Faltan nº columnas de la matriz\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    #ifdef MATRIX_GLOBAL
    if (N>MAX) N=MAX;
    #endif
    #ifdef MATRIX_DYNAMIC
    double *v1, *vf;
    double **m;

    v1 = (double*) malloc(N * sizeof(double));
    vf = (double*) malloc(N * sizeof(double));
    m = (double**) malloc(N * sizeof(double*));

    for (int i=0; i<N; i++)
        m[i] = (double*)malloc(N * sizeof(double));

    if ((v1 == NULL) || (vf == NULL) || (m == NULL))
    {
        printf("No hay suficiente espacio para reservar memoria \n");
        exit(-2);
    }
    #endif

    struct timespec cgt1,cgt2;
    double ncgt;

    printf("Tamaño Matriz:%u x %u (%u B)\n", N, N, sizeof(unsigned int));
    //Inicializacion
    for (int i=0; i<N; ++i){
        v1[i] = N*0.1+i*0.1;
        for (int j=0; j<N; ++j)
            m[i][j] = N*0.1+i*0.1;
    }

    //Multiplicacion
    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (int i=0; i<N; ++i)
        for (int j=0; j<N; ++j)
            vf[i] += m[i][j] * v1[j];

```

```
62      clock_gettime(CLOCK_REALTIME,&cgt2);
63
64      //Salida del resultado y el tiempo de ejecucion
65      ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
66         (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
67
68      if (N<12)
69      {
70         printf("Tiempo:%11.9f\t / Tamaño columnas:%u\n",ncgt,N);
71         for (int i=0; i<N; ++i)
72         {
73            printf("/ ");
74            for (int j=0; j<N-1; ++j)
75               printf("(%8f*%8f)+", m[i][j], v1[j]);
76            printf("(%8f*%8f) =", m[i][N-1], v1[N-1]);
77            printf(" V2[%d] = %8f /\n", i, vf[i]);
78         }
79      }
80      else
81         printf("Tiempo:%11.9f\t / Tamaño columnas:%u\t/ V2[0] = %8.6f / / V2[%d] = %8.6f /\n", ncgt, N, vf[0],
         N-1, vf[N-1]);
82
83      #ifdef MATRIX_DYNAMIC
84         for(int i = 0; i < N; ++i)
85            free(m[i]);
86         free(m);
87         free(v1);
88         free(vf);
89      #endif
90
91      return 0;
92   }
```

CAPTURAS DE PANTALLA:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer8] 2019-04-18 jueve
s
$./a.out 5
Tamaño Matriz:5 x 5 (4 B)
Tiempo:0.000000912       / Tamaño columnas:5
/ (0.500000*0.500000)+(0.500000*0.600000)+(0.500000*0.700000)+(0.500000*0.800000)+(0.500000*0.900000) = V2[0] = 1
.750000 /
/ (0.600000*0.500000)+(0.600000*0.600000)+(0.600000*0.700000)+(0.600000*0.800000)+(0.600000*0.900000) = V2[1] = 2
.100000 /
/ (0.700000*0.500000)+(0.700000*0.600000)+(0.700000*0.700000)+(0.700000*0.800000)+(0.700000*0.900000) = V2[2] = 2
.450000 /
/ (0.800000*0.500000)+(0.800000*0.600000)+(0.800000*0.700000)+(0.800000*0.800000)+(0.800000*0.900000) = V2[3] = 2
.800000 /
/ (0.900000*0.500000)+(0.900000*0.600000)+(0.900000*0.700000)+(0.900000*0.800000)+(0.900000*0.900000) = V2[4] = 3
.150000 /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer8] 2019-04-18 jueve
s
$./a.out 100
Tamaño Matriz:100 x 100 (4 B)
Tiempo:0.000095628       / Tamaño columnas:100  / V2[0] = 14950.000000 / / V2[99] = 29750.500000 /
```

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

#define MATRIX_GLOBAL
//#define MATRIX_DYNAMIC

#ifdef MATRIX_GLOBAL
#define MAX 4096

double m[MAX][MAX], v1[MAX], vf[MAX];
#endif
int main(int argc, char** argv)
{
   if (argc<2)
   {
      printf("Faltan nº columnas de la matriz\n");
      exit(-1);
   }

   int i, j;
   double cgt1, cgt2, ncgt;
   unsigned int N = atoi(argv[1]);

   #ifdef MATRIX_GLOBAL
   if (N>MAX) N=MAX;
   #endif
   #ifdef MATRIX_DYNAMIC
   double *v1, *vf;
   double **m;

   v1 = (double*) malloc(N * sizeof(double));
   vf = (double*) malloc(N * sizeof(double));
   m = (double**) malloc(N * sizeof(double*));

   for (i=0; i<N; i++)
      m[i] = (double*)malloc(N * sizeof(double));

   if ((v1 == NULL) || (vf == NULL) || (m == NULL))
   {
      printf("No hay suficiente espacio para reservar memoria \n");
      exit(-2);
   }
   #endif

   printf("Tamaño Matriz:%u x %u (%u B)\n", N, N, sizeof(unsigned int));
   //Inicializacion
   #pragma omp parallel for private (j)
   for (i=0; i<N; ++i) {
      v1[i] = N*0.1+i*0.1;
      for (j=0; j<N; ++j)
         m[i][j] = N*0.1+i*0.1;
   }
```

```
59
60      //Multiplicacion
61      cgt1 = omp_get_wtime();
62
63      #pragma omp parallel for private (j)
64      for (i=0; i<N; ++i) {
65          for (j=0; j<N; ++j)
66              vf[i] += m[i][j] * v1[j];
67      }
68
69      cgt2 = omp_get_wtime();
70      ncgt = cgt2 - cgt1;
71
72      //Salida del resultado y el tiempo de ejecucion
73      if (N<12)
74      {
75          printf("Tiempo:%11.9f\t / Tamaño columnas:%u\n",ncgt,N);
76          for (i=0; i<N; ++i)
77          {
78              printf("/ ");
79              for (j=0; j<N-1; ++j)
80                  printf("(%8f*%8f)+", m[i][j], v1[j]);
81              printf("(%8f*%8f) =", m[i][N-1], v1[N-1]);
82              printf(" V2[%d] = %8f /\n", i, vf[i]);
83          }
84      }
85      else
86          printf("Tiempo:%11.9f\t / Tamaño columnas:%u\t/ V2[0] = %8.6f / / V2[%d] = %8.6f /\n", ncgt, N, vf[0],
            N-1, vf[N-1]);
87
88      #ifdef MATRIX_DYNAMIC
89          for(i = 0; i < N; ++i)
90              free(m[i]);
91          free(m);
92          free(v1);
93          free(vf);
94      #endif
95
96      return 0;
97  }
```

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

```
//Inicializacion
for (i=0; i<N; ++i) {
    v1[i] = N*0.1+i*0.1;
    #pragma omp parallel for
    for (j=0; j<N; ++j)
        m[i][j] = N*0.1+i*0.1;
}

//Multiplicacion
cgt1 = omp_get_wtime();

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

(El resto de código es igual al de la versión a)

RESPUESTA: En la versión a no he tenido errores de compilación, pero los resultados eran incorrectos ya que al estar declarada la variable j fuera de la región paralela, en el bucle interno se compartía el indice j. Para solucionarlo había que indicar el ámbito privado con: “private (j)”.

En la versión b era importante tener en cuenta que las sumas parciales no se asignaran a la vez perdiéndose algunas de ellas, al igual que en el ejercicio 7 lo resolvemos con un atomic.

CAPTURAS DE PANTALLA:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer9] 2019-04-18 jueves
$gcc -fopenmp pmv-OpenMP-a.c -o pmva
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer9] 2019-04-18 jueves
$gcc -fopenmp pmv-OpenMP-b.c -o pmvb
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer9] 2019-04-18 jueves
$./pmva 5
Tamaño Matriz:5 x 5 (4 B)
Tiempo:0.000006301       / Tamaño columnas:5
/ (0.500000*0.500000)+(0.500000*0.600000)+(0.500000*0.700000)+(0.500000*0.800000)+(0.500000*0.900000) = V2[0] = 1.750000 /
/ (0.600000*0.500000)+(0.600000*0.600000)+(0.600000*0.700000)+(0.600000*0.800000)+(0.600000*0.900000) = V2[1] = 2.100000 /
/ (0.700000*0.500000)+(0.700000*0.600000)+(0.700000*0.700000)+(0.700000*0.800000)+(0.700000*0.900000) = V2[2] = 2.450000 /
/ (0.800000*0.500000)+(0.800000*0.600000)+(0.800000*0.700000)+(0.800000*0.800000)+(0.800000*0.900000) = V2[3] = 2.800000 /
/ (0.900000*0.500000)+(0.900000*0.600000)+(0.900000*0.700000)+(0.900000*0.800000)+(0.900000*0.900000) = V2[4] = 3.150000 /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer9] 2019-04-18 jueves
$./pmvb 5
Tamaño Matriz:5 x 5 (4 B)
Tiempo:0.000022345       / Tamaño columnas:5
/ (0.500000*0.500000)+(0.500000*0.600000)+(0.500000*0.700000)+(0.500000*0.800000)+(0.500000*0.900000) = V2[0] = 1.750000 /
/ (0.600000*0.500000)+(0.600000*0.600000)+(0.600000*0.700000)+(0.600000*0.800000)+(0.600000*0.900000) = V2[1] = 2.100000 /
/ (0.700000*0.500000)+(0.700000*0.600000)+(0.700000*0.700000)+(0.700000*0.800000)+(0.700000*0.900000) = V2[2] = 2.450000 /
/ (0.800000*0.500000)+(0.800000*0.600000)+(0.800000*0.700000)+(0.800000*0.800000)+(0.800000*0.900000) = V2[3] = 2.800000 /
/ (0.900000*0.500000)+(0.900000*0.600000)+(0.900000*0.700000)+(0.900000*0.800000)+(0.900000*0.900000) = V2[4] = 3.150000 /
```

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`

```
//Multiplicacion
cgt1 = omp_get_wtime();

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

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

Resto de código igual al desarrollado en el 9b

RESPUESTA: Solo necesitamos añadir la cláusula reduction al parallel for indicando que reduzca las sumas de vf y el programa ya funciona correctamente dando la misma salida que el resto.

CAPTURAS DE PANTALLA:

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):

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer9] 2019-04-18 jueves
$gcc -fopenmp -O2 pmv-OpenMP-a.c -o pmv-a
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer9] 2019-04-18 jueves
$gcc -fopenmp -O2 pmv-OpenMP-b.c -o pmv-b
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer10] 2019-04-18 jueves
$gcc -fopenmp -O2 pmv-OpenMP-reduction.c -o pmv-reduction
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer10] 2019-04-18 jueves
$./pmv-a 4096
Tamaño Matriz:4096 x 4096 (4 B)
Tiempo:0.007857677       / Tamaño columnas:4096 / V2[0] = 1030708264.960000 / / V2[4095] = 2061164892.160000 /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer10] 2019-04-18 jueves
$./pmv-b 4096
Tamaño Matriz:4096 x 4096 (4 B)
Tiempo:5.472407511       / Tamaño columnas:4096 / V2[0] = 1030708264.960000 / / V2[4095] = 2061164892.160000 /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp2/ejer10] 2019-04-18 jueves
$./pmv-reduction 4096
Tamaño Matriz:4096 x 4096 (4 B)
Tiempo:0.198159444       / Tamaño columnas:4096 / V2[0] = 1030708264.960000 / / V2[4095] = 2061164892.160000 /
```

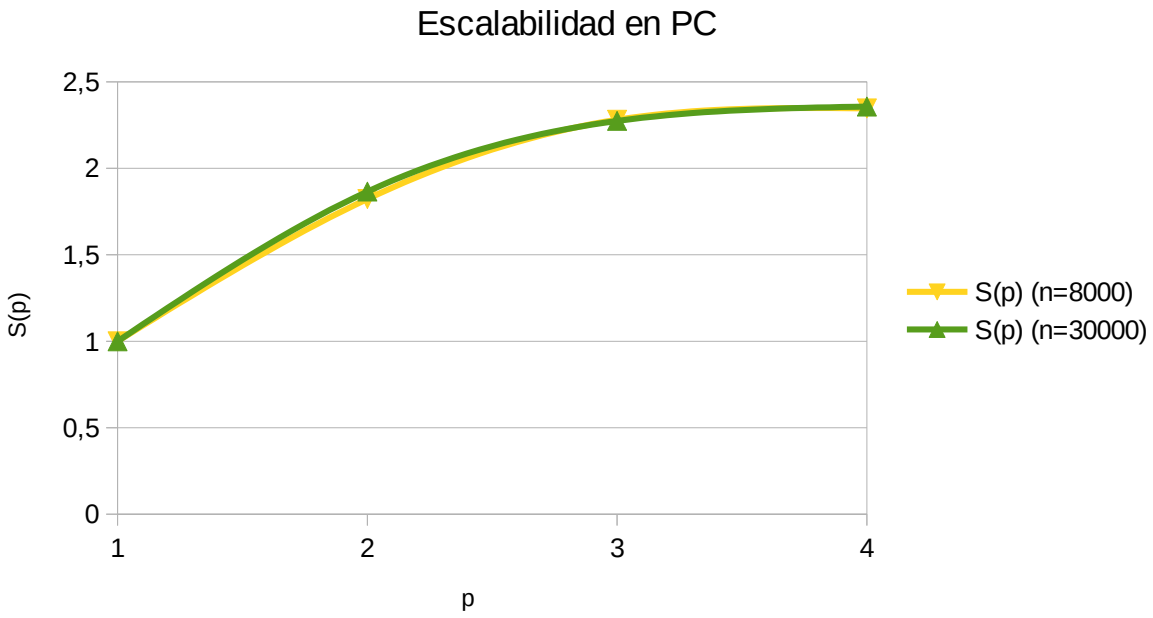
TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia) (para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 20000 y 100000, y otro entre 5000 y 20000):

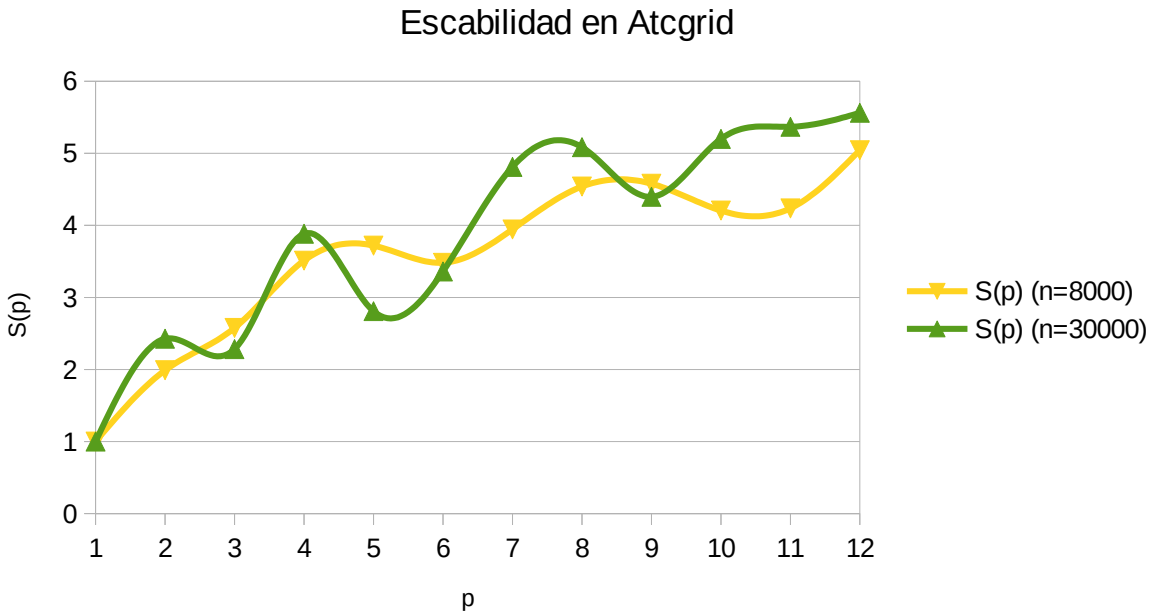
PC

p	Tiempo (n=8000)	Tiempo (n=30000)	S(p) (n=8000)	S(p) (n=30000)
1	0,056942291	0,810517179	1	1
2	0,031249279	0,434562784	1,8221953537	1,86513251673
3	0,024970092	0,356489575	2,2804197518	2,27360696032
4	0,024271977	0,343897644	2,3460095978	2,35685586436

Atcgrid

p	Tiempo (n=8000)	Tiempo (n=30000)	S(p) (n=8000)	S(p) (n=30000)
1	0,14170598	4,371753426	1	1
2	0,071225393	1,801869105	1,989542971	2,4262325237
3	0,055004085	1,915379832	2,5762810162	2,28244724778
4	0,040368908	1,126014442	3,5102752841	3,88250209139
5	0,038106391	1,5565092	3,7186932764	2,80869102862
6	0,040707204	1,30124789	3,4811032465	3,35966225928





COMENTARIOS SOBRE LOS RESULTADOS: El mejor código que elegimos es la versión a, ya que el código b no puede realizar varias sumas en paralelo teniendo un rendimiento penoso y el código reduction tiene que realizar operaciones adicionales para juntar las sumas paralelas. Podemos ver como la ganancia sube muy rápido al principio pero cada vez más lento cuantos más procesadores.