2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP

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

---

1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.

CÓDIGO FUENTE: `if-clauseModificado.c`

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

int main(int argc, char **argv)
{
   int i, n=20, tid;
   int a[n],suma=0,sumalocal;
   if(argc < 2)     {
      fprintf(stderr,"[ERROR]-Falta iteraciones\n");
      exit(-1);
     }
   n = atoi(argv[1]); if (n>20) n=20;
   for (i=0; i<n; i++) {
      a[i] = i;
   }

  #pragma omp parallel if(n>4) default(none) \
                     private(sumalocal,tid) shared(a,suma,n)
  {  sumalocal=0;
     tid=omp_get_thread_num();
     #pragma omp for private(i) schedule(static) nowait
     for (i=0; i<n; i++)
     {   sumalocal += a[i];
         printf(" thread %d suma de a[%d]=%d sumalocal=%d \n",
                     tid,i,a[i],sumalocal);
     }
     #pragma omp atomic
         suma += sumalocal;
     #pragma omp barrier
     #pragma omp master
         printf("thread master=%d imprime suma=%d\n",tid,suma);
  }
  return(0);
}
```

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./if-clause 10 5
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 2 suma de a[4]=4 sumalocal=4
 thread 2 suma de a[5]=5 sumalocal=9
 thread 3 suma de a[6]=6 sumalocal=6
 thread 3 suma de a[7]=7 sumalocal=13
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 4 suma de a[8]=8 sumalocal=8
 thread 4 suma de a[9]=9 sumalocal=17
thread master=0 imprime suma=45
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./if-clause 10 7
 thread 4 suma de a[7]=7 sumalocal=7
 thread 5 suma de a[8]=8 sumalocal=8
 thread 2 suma de a[4]=4 sumalocal=4
 thread 2 suma de a[5]=5 sumalocal=9
 thread 3 suma de a[6]=6 sumalocal=6
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 6 suma de a[9]=9 sumalocal=9
thread master=0 imprime suma=45
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./if-clause 10 1
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 0 suma de a[3]=3 sumalocal=6
 thread 0 suma de a[4]=4 sumalocal=10
 thread 0 suma de a[5]=5 sumalocal=15
 thread 0 suma de a[6]=6 sumalocal=21
 thread 0 suma de a[7]=7 sumalocal=28
 thread 0 suma de a[8]=8 sumalocal=36
 thread 0 suma de a[9]=9 sumalocal=45
thread master=0 imprime suma=45
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$
```

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,…15)
- chunck= 1, 2 y 4

Tabla 1 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Para que veas que uso make y scripts:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ls
if-clause.c  Makefile           scheduled-clause.c            scheduleg-clause.c  tabla_shcedule.ods
logs         schedule-clause.c  scheduled-clauseModificado.c  script.sh
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ make &> /dev/null
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ls
if-clause    logs      schedule-clause    scheduled-clause    scheduled-clauseModificado    scheduleg-clause    script.sh
if-clause.c  Makefile  schedule-clause.c  scheduled-clause.c  scheduled-clauseModificado.c  scheduleg-clause.c  tabla_shcedule.ods
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./script.sh
Iter    Chunk 1 Chunk 2 Chunk 4 Chunk 1 Chunk 2 Chunk 4 Chunk 1 Chunk 2 Chunk 4
0       0       0       0       0       1       0       1       1       1
1       1       0       0       0       1       0       1       1       1
2       1       0       0       0       1       0       1       1       1
3       1       0       0       0       1       0       1       1       1
4       1       0       0       0       1       0       1       1       1
5       1       0       0       0       1       0       1       1       1
6       1       0       0       0       1       0       1       1       1
7       1       0       0       0       1       0       1       1       1
8       1       1       1       0       1       0       1       1       1
9       0       1       1       0       1       0       1       1       1
10      0       1       1       0       1       0       1       1       1
11      0       1       1       0       1       0       1       1       1
12      0       1       1       0       1       1       0       0       0
13      0       1       1       0       1       1       0       0       0
14      0       1       1       0       0       1       0       0       0
15      0       1       1       1       0       1       0       0       0
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$
```

Las tres primeras columnas son las de schedule-clause, las tres siguientes de scheduled-claused y las tres ultimas de scheduleg-clause.

(b) Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

Tabla 2 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ls
if-clause.c  schedule-clause4.c   scheduled-clause.c            scheduleg-clause.c  tabla_shcedule.ods
logs         schedule-clause.c    scheduled-clauseModificado.c  script_4Hebras.sh
Makefile     scheduled-clause4.c  scheduleg-clause4.c           script.sh
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ make &> /dev/null
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ls
if-clause    schedule-clause     scheduled-clause     scheduled-clauseModificado    scheduleg-clause4.c  tabla_shcedule.ods
if-clause.c  schedule-clause4    scheduled-clause4    scheduled-clauseModificado.c  scheduleg-clause.c
logs         schedule-clause4.c  scheduled-clause4.c  scheduleg-clause              script_4Hebras.sh
Makefile     schedule-clause.c   scheduled-clause.c   scheduleg-clause4             script.sh
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./script_4Hebras.sh
Iter    Chunk 1 Chunk 2 Chunk 4 Chunk 1 Chunk 2 Chunk 4 Chunk 1 Chunk 2 Chunk 4
0       3       2       1       3       0       0       0       1       3
1       3       3       1       3       0       2       1       3       3
2       3       2       1       3       1       2       1       3       3
3       3       2       1       3       1       2       1       3       3
4       0       2       3       3       2       0       1       3       0
5       0       3       3       3       2       2       1       3       0
6       0       3       3       3       1       3       1       3       0
7       0       3       3       3       1       3       1       0       0
8       1       0       0       3       1       3       1       0       2
9       1       0       0       3       1       3       1       0       2
10      1       0       0       3       2       2       0       0       2
11      1       0       0       3       2       2       0       1       2
12      2       1       2       3       0       2       0       1       1
13      2       1       2       1       0       2       2       2       1
14      2       1       2       2       3       0       2       2       1
15      2       1       2       0       3       0       2       2       1
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$
```

Para que veas que uso los script y el make...

3. Añadir al programa `scheduled-clause.c` lo necesario para que imprima el valor de las variables de control `dyn-var`, `nthreads-var`, `thread-limit-var` y `run-sched-var` dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

CÓDIGO FUENTE: `scheduled-clauseModificado.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=200,chunk,a[n],suma=0;
   int * modifier;
   omp_sched_t * kind;
   if(argc < 3){
      fprintf(stderr,"\nFalta iteraciones o chunk \n");
      exit(-1);
   }
   n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);

   for (i=0; i<n; i++)
       a[i] = i;
   #pragma omp parallel
   {
   #pragma omp single
   {
   printf("dyn-var (ajuste dinámico del numero de threads) dentro de la región
paralela: %d \n",omp_get_dynamic());
   printf("nthreads-var(threads en la siguiente ejecución paralela) dentro de
la región paralela: %d \n",omp_get_max_threads());
   printf("thread-limit-var (máximo numero de threads para todo el programa)
dentro de la región paralela: %d \n",omp_get_thread_limit());
   omp_get_schedule(&kind,&modifier);
   printf("run-sched-var (planificación de bucles para runtime) dentro de la
región paralela. Kind: %d, Modifier: %d \n",kind,modifier);
   }
   #pragma omp for firstprivate(suma) \
            lastprivate(suma) schedule(dynamic,chunk)
   for (i=0; i<n; i++){
       suma = suma + a[i];
       printf(" thread %d suma a[%d]=%d suma=%d
\n",omp_get_thread_num(),i,a[i],suma);
   }
   }
   printf("Fuera de 'parallel for' suma=%d\n",suma);
   printf("dyn-var FUERA de la región paralela: %d \n",omp_get_dynamic());
   printf("nthreads-var FUERA de la región paralela: %d
\n",omp_get_max_threads());
   printf("thread-limit-var FUERA de la región paralela: %d
\n",omp_get_thread_limit());
   omp_get_schedule(&kind,&modifier);
   printf("run-sched-var dentro de la región paralela. Kind: %d. Modifier:
%d \n",kind,modifier);
   return(0);
}
```

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./scheduled-clauseModificado 10 2
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: (Kind: 2, Modifier: 1)
 thread 2 suma a[0]=0 suma=0
 thread 2 suma a[1]=1 suma=1
 thread 3 suma a[6]=6 suma=6
 thread 3 suma a[7]=7 suma=13
 thread 0 suma a[4]=4 suma=4
 thread 0 suma a[5]=5 suma=9
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[3]=3 suma=5
 thread 2 suma a[8]=8 suma=9
 thread 2 suma a[9]=9 suma=18
Fuera de 'parallel for' suma=18
dyn-var FUERA de 'parallel for': 0
nthreads-var FUERA de 'parallel for': 4
thread-limit-var FUERA de 'parallel for': 2147483647
run-sched-var dentro de 'parallel for'. (Kind: 2. Modifier: 1)
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$
```

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

```
Scheduled-clausemodificado4
```

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./scheduled-clauseModificado4 10 4
[Fuera Parallel] omp_get_num_threads: 4
[Parallel] omp_get_num_procs: 4
[Parallel] omp_in_parallel: 1
thread 1 suma a[0]=0 suma=0
thread 1 suma a[1]=1 suma=1
thread 1 suma a[2]=2 suma=3
thread 1 suma a[3]=3 suma=6
thread 0 suma a[4]=4 suma=4
thread 0 suma a[5]=5 suma=9
thread 0 suma a[6]=6 suma=15
thread 0 suma a[7]=7 suma=22
thread 2 suma a[8]=8 suma=8
thread 2 suma a[9]=9 suma=17
[Fuera Parallel] suma=17
[Fuera Parallel] omp_get_num_threads: 1
[Fuera Parallel] omp_get_num_procs: 4
[Fuera Parallel] omp_in_parallel: 0
```

RESPUESTA: Claramente se obtienen valores distintos en el booleano que indica si estamos dentro o fuera de la seccion paralela (omp_in_parallel) y tambien en el que indica el numero de hebras activas (omp_get_num_procs).

5. Añadir al programa `scheduled-clause.c` lo necesario para modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

CÓDIGO FUENTE: `scheduled-clauseModificado5.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=200,chunk,a[n],suma=0;
    int * modifier;
    omp_sched_t * kind;

   if(argc < 3)     {
      fprintf(stderr,"\nFalta iteraciones o chunk \n");
      exit(-1);
     }
   n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);

   for (i=0; i<n; i++)
      a[i] = i;
   #pragma omp parallel
   {
   #pragma omp single
   {
      printf("[Parallel] dyn-var: %d \n",omp_get_dynamic());
      printf("[Parallel] Modificamos dyn-var: (hacemos omp_set_dynamic(3);)
\n");
      omp_set_dynamic(3);
      printf("[Parallel] dyn-var despues de la modificacion: %d
\n",omp_get_dynamic());

      printf("[Parallel] nthreads-var: %d \n",omp_get_max_threads());
      printf("[Parallel] Modificar nthreads-var: (lo ponemos a 8) \n");
      omp_set_num_threads(8);
      printf("[Parallel] nthreads-var despues de la modificacion: %d
\n",omp_get_max_threads);
      omp_get_schedule(&kind,&modifier);
      printf("[Parallel] run-sched-var: (Kind: %d, Modifier: %d)
\n",kind,modifier);
      printf("[Parallel] Modificar run-sched-var: \n");
      omp_set_schedule(3,2);
      omp_get_schedule(&kind,&modifier);
      printf("[Parallel] Kind despues la modificacion: %d, Modifier despues la
modificacion: %d \n",kind,modifier);

     }

     #pragma omp barrier
     #pragma omp for firstprivate(suma) \
     lastprivate(suma) schedule(dynamic,chunk)
         for (i=0; i<n; i++){
              suma = suma + a[i];
              printf(" thread %d suma a[%d]=%d suma=%d \n",
```

```
omp_get_thread_num(),i,a[i],suma);                    }
    }

    printf("[Fuera Parallel] suma=%d\n",suma);
    printf("[Fuera Parallel] dyn-var: %d \n",omp_get_dynamic());
    printf("[Fuera Parallel] nthreads-var: %d \n",omp_get_max_threads());
    omp_get_schedule(&kind,&modifier);
    printf("[Fuera Parallel] run-sched-var: (Kind: %d. Modifier: %d)
\n",kind,modifier);

    return(0);
}
```

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ls
if-clause          schedule-clause4.c    scheduled-clauseModificado      scheduled-clauseModificado.c
if-clause.c        schedule-clause.c     scheduled-clauseModificado4     scheduleg-clause
logs               scheduled-clause      scheduled-clauseModificado4.c   scheduleg-clause4
Makefile           scheduled-clause4     scheduled-clauseModificado4.c~  scheduleg-clause4.c
schedule-clause    scheduled-clause4.c   scheduled-clauseModificado5     scheduleg-clause.c
schedule-clause4   scheduled-clause.c    scheduled-clauseModificado5.c   script_4Hebras.sh
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./*5 10 4
[Parallel] dyn-var: 0
[Parallel] Modificamos dyn-var: (hacemos omp_set_dynamic(3);)
[Parallel] dyn-var despues de la modificacion: 1
[Parallel] nthreads-var: 4
[Parallel] Modificar nthreads-var: (lo ponemos a 8)
[Parallel] nthreads-var despues de la modificacion: 4197040
[Parallel] run-sched-var: (Kind: 2, Modifier: 1)
[Parallel] Modificar run-sched-var:
[Parallel] Kind despues la modificacion: 3, Modifier despues la modificacion: 2
 thread 3 suma a[0]=0 suma=0
 thread 3 suma a[1]=1 suma=1
 thread 3 suma a[2]=2 suma=3
 thread 3 suma a[3]=3 suma=6
 thread 2 suma a[8]=8 suma=8
 thread 2 suma a[9]=9 suma=17
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 1 suma a[6]=6 suma=15
 thread 1 suma a[7]=7 suma=22
[Fuera Parallel] suma=17
[Fuera Parallel] dyn-var: 0
[Fuera Parallel] nthreads-var: 4
[Fuera Parallel] run-sched-var: (Kind: 2. Modifier: 1)
```

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: `pmtv-secuencial.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 11000

int main(int argc, char **argv){
             int i, j;
             int filas_columnas;
             int suma;
             suma = 0;

             int *vector;
             int **matriz;
             int *resultado;

             if (argc < 2){
                          printf("Falta el número de componentes\n");
                          return(-1);
             }
             filas_columnas = atoi(argv[1]);

             if (filas_columnas > MAX){
                          printf("Tamaño de la matriz demasiado grande.
Superar %d provoca fallos\n",MAX);
                          return(-1);
             }

    //Reservamos memoria para los vectores y la matriz
             vector = (int *)malloc(filas_columnas * sizeof(int));
             resultado = (int *)malloc(filas_columnas * sizeof(int));
             matriz = (int **)malloc(filas_columnas * sizeof(int *));

             for (i=0; i<filas_columnas; i++){
                          matriz[i] = (int *)malloc(filas_columnas *
sizeof(int));
             }

//Inicializamos matriz (TRIANGULAR INFERIOR) y vector
// i= columnas de la matriz
// j= filas de la matriz

             for (j=0; j<filas_columnas; j++){
                          for (i=0; i<filas_columnas;i++){
                                       if (j>i)
                                                    matriz[j][i] = 0;
                                       else
                                                    matriz[j][i] = j*i;
                          }
                          vector[j] = i;
             }

//Multiplicamos la matriz por el vector
```

```
            for (i=0; i<filas_columnas; i++){
                        suma=0;
                        for (j=0; j<filas_columnas; j++){
                                    suma+=(matriz[j][i]*vector[i]);
                        }
                        resultado[i] = suma;
            }
//Hacemos un par de printf para que el optimizador de codigo no se salte el
programa entero...
            printf ("Componente(0,0) del resultado del producto: matriz
triangular por vector = %d\n",resultado[0]);
            printf ("Componente(N-1,N-1) del resultado del producto: matriz
triangular por vector = %d\n",resultado[filas_columnas-1]);
    free(matriz);
    free(resultado);
    free(vector);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ls
if-clause          pmtv-secuencial.c~  scheduled-clause4             scheduled-clauseModificado4.c~  scheduleg-clause4.c
if-clause.c        schedule-clause     scheduled-clause4.c           scheduled-clauseModificado5     scheduleg-clause.c
logs               schedule-clause4    scheduled-clause.c            scheduled-clauseModificado5.c   script_4Hebras.sh
Makefile           schedule-clause4.c  scheduled-clauseModificado    scheduled-clauseModificado.c    script.sh
pmtv-secuencial    schedule-clause.c   scheduled-clauseModificado4   scheduleg-clause                tabla_shcedule.ods
pmtv-secuencial.c  scheduled-clause    scheduled-clauseModificado4.c scheduleg-clause4
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./pmtv-secuencial 1000
Componente(0,0) del resultado del producto: matriz triangular por vector = 0
Componente(N-1,N-1) del resultado del producto: matriz triangular por vector = 784293664
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./pmtv-secuencial 10000
Componente(0,0) del resultado del producto: matriz triangular por vector = 0
Componente(N-1,N-1) del resultado del producto: matriz triangular por vector = 1714839680
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ time(./pmtv-secuencial 10000)
Componente(0,0) del resultado del producto: matriz triangular por vector = 0
Componente(N-1,N-1) del resultado del producto: matriz triangular por vector = 1714839680

real    0m4.082s
user    0m3.425s
sys     0m0.653s
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$
```

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno OMP_SCHEDULE. Obtener en atcgrid los tiempos de ejecución del código paralelo que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 2, 64, 128, 1024 y el chunk por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096 componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 con los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

CÓDIGO FUENTE: pmtv-OpenMP.c

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

#define MAX 11000

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

             int i, j;
             int filas_columnas;
             double tiempo;
             int suma;
             suma = 0;

             int *vector;
             int **matriz;
             int *resultado;

             int chunk;

             if (argc < 2){
       printf("Falta chunk y número de componentes\n");
                return(-1);
             }

             if(argc < 3)     {
                           printf("Falta chunk\n");
         return(-1);
    }

    chunk = atoi(argv[2]);

    //El número de filas y columnas será un argumento de entrada
    //Al ser una matriz triangular, sabemos que es cuadrado (no filas = no
columnas)

    filas_columnas = atoi(argv[1]);

             if (filas_columnas > MAX){
                           printf("Tamaño demasiado grande. Superar %d produce
errores\n",MAX);
                           return(-1);
             }

    //Reservamos la matriz, el vector, y el resultado de la multiplicacion

             vector = (int *)malloc(filas_columnas * sizeof(int));
             resultado = (int *)malloc(filas_columnas * sizeof(int));
             matriz = (int **)malloc(filas_columnas * sizeof(int *));

             for (i=0; i<filas_columnas; i++){
                           matriz[i] = (int *)malloc(filas_columnas *
sizeof(int));
             }


    //Inicializamos matriz (TRIANGULAR INFERIOR) y vector
    // i= columnas de la matriz
    // j= filas de la matriz
```

```
            for (j=0; j<filas_columnas; j++){
                        for (i=0; i<filas_columnas;i++){
                                    if (j>i)
                                                matriz[j][i] = 0;
                                    else
                                                matriz[j][i] = j*i;
                        }
                        vector[j] = i;
            }

            //Multiplicamos la matriz por el vector(paralelizamos)
            tiempo = omp_get_wtime();
            #pragma omp parallel for private(suma) schedule(guided,chunk)
            for (i=0; i<filas_columnas; i++){
                        suma=0;
                        for (j=0; j<filas_columnas; j++){
                                    suma+=(matriz[j][i]*vector[i]);
                        }
                        resultado[i] = suma;
            }
            tiempo = omp_get_wtime() - tiempo;
            printf ("El tiempo de ejecucion es: %f\n", tiempo);

            printf ("Componente(0,0) del resultado del producto: matriz
triangular por vector = %d \n",resultado[0]);
            printf ("Componente(N-1,N-1) del resultado del producto: matriz
triangular por vector = %d \n",resultado[filas_columnas-1]);

    //Liberamos la memoria
    free(matriz);
    free(resultado);
    free(vector);

    return 0;
}
```

CAPTURAS DE PANTALLA:

Tabla 3 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3$ ./script7.sh
Chunk   Static          Dynamic         Guided
1       0.006550        0.002587        0.001168
2       0.005302        0.011447        0.000500
32      0.012856        0.003903        0.000436
64      0.009597        0.002026        0.000445
2048    0.064957        0.062842        0.072979
```

8 . Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

$$A = B \bullet C;\ A(i,j) = \sum_{k=0}^{N-1} B(i,k) \bullet C(k,j),\ i,j = 0,\ldots N-1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: `pmm-secuencial.c`

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

#define MAX 4096

int main(int argc, char **argv){
             int i, j, k;
             int dimension_matrices;
             int suma;
             suma = 0;

             int **matrizB;
             int **matrizC;
             int **matrizA;

             if (argc < 2){
                          printf("Falta el número de componentes\n");
                          return(1);
             }
 //El producto de matrices cuadradas requiere que ambas sean de la misma
 //dimension por tanto solo necesitamos un parametro para reservar la memoria
             dimension_matrices = atoi(argv[1]);

             if (dimension_matrices > MAX){
                          printf("Tamaño demasiado grande. No superar
%d\n\n",MAX);
                          return(1);
             }
//Creamos las matrices
// matrizA = matrizB * matrizC
             matrizB = (int **)malloc(dimension_matrices * sizeof(int*));
             matrizC = (int **)malloc(dimension_matrices * sizeof(int*));
             matrizA = (int **)malloc(dimension_matrices * sizeof(int*));

             for (i=0; i<dimension_matrices; i++){
                          matrizB[i] = (int *)malloc(dimension_matrices *
sizeof(int));
                          matrizC[i] = (int *)malloc(dimension_matrices *
sizeof(int));
                          matrizA[i] = (int *)malloc(dimension_matrices *
sizeof(int));
             }
//Inicializamos las matrices
             for (j=0; j<dimension_matrices; j++){
                          for (i=0; i<dimension_matrices; i++){
                                       matrizB[j][i] = j+i;
                                       matrizC[j][i] = j*i;
                          }
             }
```

```
//Multiplicamos las matrices B y C

            for (i=0; i<dimension_matrices; i++){
                        for(j=0; j<dimension_matrices; j++){
                                    suma = 0;
                                    for (k=0; k<dimension_matrices; k++){
                                                suma += (matrizB[i]
[k]*matrizC[k][j]);
                                    }
                                    matrizA[i][j]=suma;
                        }
            }
//Imprimimos resultados
            printf ("Resultado[0][0] = %d\n",matrizA[0][0]);
            printf ("Componente(N-1,N-1) del resultado de la multiplicación
de ambas matrices=%d\n",matrizA[dimension_matrices-1][dimension_matrices-1]);

//Liberamos las matrices
    free(matrizA);
    free(matrizB);
    free(matrizC);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ ls
Makefile          pmtv-OpenMP          pmtv-OpenMP-dynamic.c  pmtv-OpenMP-static    pmtv-secuencial.c
pmm-secuencial    pmtv-OpenMP.c        pmtv-OpenMP-guided     pmtv-OpenMP-static.c
pmm-secuencial.c  pmtv-OpenMP-dynamic  pmtv-OpenMP-guided.c   pmtv-secuencial
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ ./pmm-secuencial 10
Componente(0,0) del resultado de la multiplicación de ambas matrices=0
Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=6210
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ ./pmm-secuencial 100
Componente(0,0) del resultado de la multiplicación de ambas matrices=0
Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=81021600
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ ./pmm-secuencial 1000
Componente(0,0) del resultado de la multiplicación de ambas matrices=0
Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=2073477872
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ time(./pmm-secuencial 10)
Componente(0,0) del resultado de la multiplicación de ambas matrices=0
Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=6210

real    0m0.004s
user    0m0.002s
sys     0m0.002s
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ time(./pmm-secuencial 100)
Componente(0,0) del resultado de la multiplicación de ambas matrices=0
Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=81021600

real    0m0.005s
user    0m0.003s
sys     0m0.003s
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ time(./pmm-secuencial 1000)
Componente(0,0) del resultado de la multiplicación de ambas matrices=0
Componente(N-1,N-1) del resultado de la multiplicación de ambas matrices=2073477872

real    0m19.614s
user    0m18.073s
sys     0m1.520s
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$
```

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2,Lección 5/Tema 2).

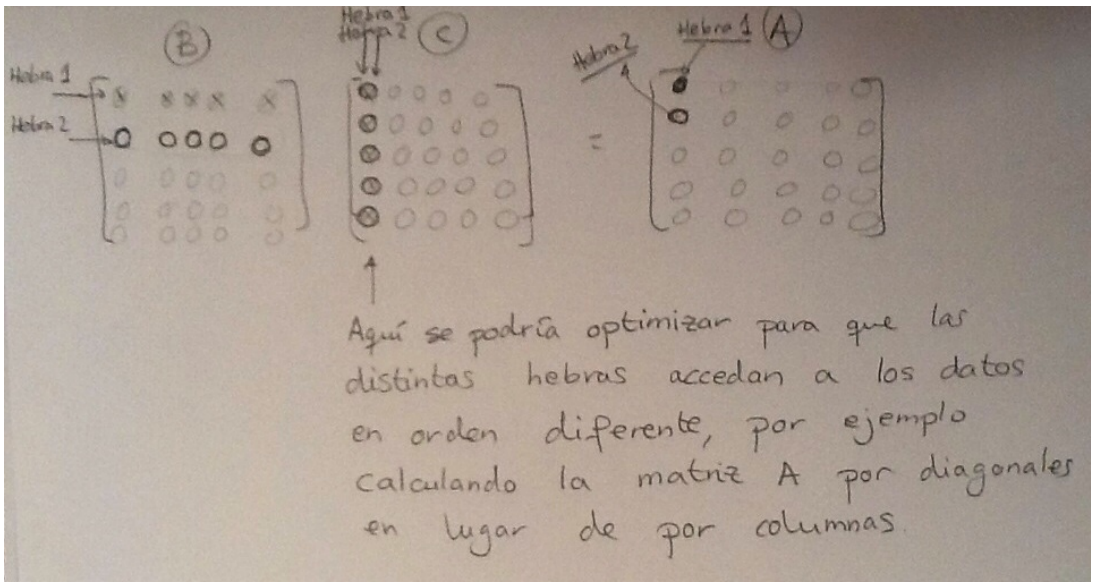
CÓDIGO FUENTE: `pmm-OpenMP.c`

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

#define MAX 4096

int main(int argc, char **argv){
             int i, j, k;
             int dimension_matrices;
             int suma;
             suma = 0;

             int **matrizB;
             int **matrizC;
             int **matrizA;

             if (argc < 2){
                          printf("Falta el número de componentes\n");
                          return(1);
             }

    //El producto de matrices cuadradas requiere que ambas sean de la misma
    //dimension por tanto solo necesitamos un parametro para reservar la
memoria
             dimension_matrices = atoi(argv[1]);

             if (dimension_matrices > MAX){
                          printf("Tamaño demasiado grande. No superar
%d\n\n",MAX);
                          return(1);
             }

//Creamos las matrices
// matrizA = matrizB * matrizC

             matrizB = (int **)malloc(dimension_matrices * sizeof(int*));
             matrizC = (int **)malloc(dimension_matrices * sizeof(int*));
             matrizA = (int **)malloc(dimension_matrices * sizeof(int*));

             for (i=0; i<dimension_matrices; i++){
                          matrizB[i] = (int *)malloc(dimension_matrices *
sizeof(int));
                          matrizC[i] = (int *)malloc(dimension_matrices *
sizeof(int));
                          matrizA[i] = (int *)malloc(dimension_matrices *
sizeof(int));
             }

//Inicializamos las matrices

             for (j=0; j<dimension_matrices; j++){
                          for (i=0; i<dimension_matrices; i++){
                                       matrizB[j][i] = j+i;
```

```
                                        matrizC[j][i] = j*i;
                                }
                        }

//Multiplicamos las matrices B y C (paralelizamos)
                        #pragma omp for private(i,j,k) schedule (runtime)
                        for (i=0; i<dimension_matrices; i++){
                                for(j=0; j<dimension_matrices; j++){
                                        suma = 0;
                                        for (k=0; k<dimension_matrices; k++){
                                                suma += (matrizB[i]
[k]*matrizC[k][j]);
                                        }
                                        matrizA[i][j]=suma;
                                }
                        }

//Imprimimos resultados

                        printf ("Resultado[0][0] = %d\n",matrizA[0][0]);
                        printf ("Resultado[N-1][N-1] = %d\n",matrizA[dimension_matrices-
1][dimension_matrices-1]);

//Liberamos las matrices
    free(matrizA);
    free(matrizB);
    free(matrizC);

    return 0;

}
```

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ ls
Makefile      pmm-OpenMP.c~     pmtv-OpenMP          pmtv-OpenMP-dynamic.c  pmtv-OpenMP-static
pmm-OpenMP    pmm-secuencial    pmtv-OpenMP.c        pmtv-OpenMP-guided     pmtv-OpenMP-static.c
pmm-OpenMP.c  pmm-secuencial.c  pmtv-OpenMP-dynamic  pmtv-OpenMP-guided.c   pmtv-secuencial
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ ./pmm-OpenMP 10
Resultado[0][0] = 0
Resultado[N-1][N-1] = 6210
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ ./pmm-OpenMP 100
Resultado[0][0] = 0
Resultado[N-1][N-1] = 81021600
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ ./pmm-OpenMP 1000
Resultado[0][0] = 0
Resultado[N-1][N-1] = 2073477872
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ time(./pmm-OpenMP 100)
Resultado[0][0] = 0
Resultado[N-1][N-1] = 81021600

real    0m0.005s
user    0m0.002s
sys     0m0.002s
rafa@RNog-Ubuntu14:~/Escritorio/AC/p3/pmv's$ time(./pmm-OpenMP 1000)
Resultado[0][0] = 0
Resultado[N-1][N-1] = 2073477872

real    0m14.422s
user    0m13.201s
sys     0m1.210s
```


B
Hebra 1
Hebra 2
Hebra 1
Hebra 2
C
Hebra 2
Hebra 1
A
=
Aquí se podría optimizar para que las distintas hebras accedan a los datos en orden diferente, por ejemplo calculando la matriz A por diagonales en lugar de por columnas.