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:

Ejercicios basados en los ejemplos del seminario práctico

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.

CAPTURA 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 < 3) {
    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;
  }

  int x = atoi(argv[2]);

  #pragma omp parallel num_threads(x) 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);
  }
}
```

CAPTURAS DE PANTALLA:

```
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./if-clauseM 8 4
 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 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 master=0 imprime suma=28
```

```
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./if-clauseM 8 2
 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 1 suma de a[4]=4 sumalocal=4
 thread 1 suma de a[5]=5 sumalocal=9
 thread 1 suma de a[6]=6 sumalocal=15
 thread 1 suma de a[7]=7 sumalocal=22
thread master=0 imprime suma=28
```

RESPUESTA:

Esta clausurá nos define el numero de threads que se va a usar en el parallel.

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el id del *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)

Iteración	schedule-clause.c			schedule-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	1	0	0	0
1	1	0	0	1	0	1	0	0	0
2	0	1	0	0	1	1	0	0	0
3	1	1	0	0	1	1	0	0	0
4	0	0	1	0	0	0	0	0	0
5	1	0	1	0	0	0	0	0	0
6	0	1	1	0	1	0	0	0	0

7	1	1	1	0	1	0	0	0	0
8	0	0	0	0	1	1	1	1	1
9	1	0	0	0	1	1	1	1	1
10	0	1	0	1	1	1	1	1	1
11	1	1	0	1	1	1	1	1	1
12	0	0	1	0	1	0	1	0	0
13	1	0	1	0	1	0	1	0	0
14	0	1	1	0	1	0	1	0	0
15	1	1	1	0	1	0	1	0	0

(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)

Iteración	**schedule-clause.c**			**schedule-claused.c**			**schedule-clauseg.c**		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	3	3	0	1
1	1	0	0	3	0	3	3	0	1
2	2	1	0	2	1	3	3	0	1
3	3	1	0	1	1	3	3	0	1
4	0	2	1	3	3	2	2	3	3
5	1	2	1	3	3	2	2	3	3
6	2	3	1	3	2	2	2	3	3
7	3	3	1	3	2	2	1	2	3
8	0	0	2	3	0	1	1	2	0
9	1	0	2	3	0	1	1	2	0
10	2	1	2	3	0	1	0	1	0
11	3	1	2	3	0	1	0	1	0
12	0	2	3	3	0	0	2	0	2
13	1	2	3	3	0	0	2	0	2
14	2	3	3	3	0	0	2	0	2
15	3	3	3	3	0	0	2	0	2

Escriba en el cuaderno de prácticas las diferencias en el comportamiento de `schedule()` con `static`, `dynamic` y `guided`.

RESPUESTA:

Static : va a asignar a cada thread un numero de iteraciones en orden.
Dynamic : asigna mas iteraciones a las hebras en función de que hebra trabaja mas rapido
Guided : divide el total de iteraciones entre el numero de hebras, el resultado es el numero de iteraciones para esa hebra. El numero anterior menos el resultado, le aremos el mismo proceso y se le asignará a otra, esta asignación se hace a la thread mas rapida.

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?

CAPTURA 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;
  omp_sched_t kind;
  int modifier;
   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 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);
       if(omp_get_thread_num() == 0)
       {
         omp_get_schedule(&kind,&modifier);
         printf("Dentro de la region paralela:\ndyn-var: %d\nthreads-var:
%d\nthread-limit-var: %d\nrun-sched-var:\n\tkind: %d\n\tmodifier:
%d\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind,modi
fier);
       }
    }

    omp_get_schedule(&kind,&modifier);
    printf("Fuera de la region paralela:\ndyn-var: %d\nthreads-var:
%d\nthread-limit-var: %d\nrun-sched-var: \n\tkind: %d\n\tmodifier:
%d\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind,modi
fier);

    // printf("\nFuera de 'parallel for' suma=%d\n",suma);



   return(0);
}
```

CAPTURAS DE PANTALLA:

```
$export OMP_DYNAMIC=TRUE
[IgnacioMorillasPadial ignacio@ig
08 martes
$export OMP_NUM_THREADS=8
[IgnacioMorillasPadial ignacio@ig
08 martes
$export OMP_NUM_THREADS=6
[IgnacioMorillasPadial ignacio@ig
08 martes
$export OMP_THREAD_LIMIT=12
[IgnacioMorillasPadial ignacio@ig
08 martes
$export OMP_SCHEDULE="dynamic"
[IgnacioMorillasPadial ignacio@ig
08 martes
$./scheduled-clauseM 16 1
Dentro de la region paralela:
dyn-var: 1
threads-var: 6
thread-limit-var: 12
run-sched-var:
        kind: 2
        modifier: 1
Fuera de la region paralela:
dyn-var: 1
threads-var: 6
thread-limit-var: 12
run-sched-var:
        kind: 2
        modifier: 1
```

```
$export OMP_DYNAMIC=FALSE
[IgnacioMorillasPadial ignacio@
-08 martes
$export OMP_NUM_THREADS=8
[IgnacioMorillasPadial ignacio@
-08 martes
$export OMP_THREAD_LIMIT=10
[IgnacioMorillasPadial ignacio@
-08 martes
$export OMP_SCHEDULE="dynamic"
[IgnacioMorillasPadial ignacio@
-08 martes
$./scheduled-clauseM 16 1
Fuera de la region paralela:
dyn-var: 0
threads-var: 8
thread-limit-var: 10
run-sched-var:
        kind: 2
        modifier: 1
```

RESPUESTA:

Como podemos comprobar el resultado es el mismo tanto dentro como fuera de la región paralela.

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.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado4.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;
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 for firstprivate(suma) \
lastprivate(suma) schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
printf(" thread %d suma a[%d]=%d suma=%d \n",

omp_get_thread_num(),i,a[i],suma);
if(omp_get_thread_num() == 0)
printf("Dentro de la region paralela:\nnum_threads: %d\nnum_proc:
%d\nin_parallel:
%d\n",omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
}
printf("\nFuera de la region paralela:\nnum_threads: %d\nnum_proc:
%d\nin_parallel:
%d\n",omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
return(0);
}
```

CAPTURAS DE PANTALLA:

```
$./scheduled-clauseM 16 2
 thread 0 suma a[2]=2 suma=0
Dentro de la region paralela:
num_threads: 3
num_proc:4
in_parallel:1
 thread 0 suma a[3]=3 suma=0
Dentro de la region paralela:
num_threads: 3
num_proc:4
in_parallel:1
 thread 0 suma a[6]=6 suma=0
Dentro de la region paralela:
num_threads: 3
num_proc:4
in_parallel:1
 thread 0 suma a[7]=7 suma=0
 thread 2 suma a[4]=4 suma=0
 thread 1 suma a[0]=0 suma=0
 thread 1 suma a[1]=1 suma=0
 thread 1 suma a[8]=8 suma=0
 thread 1 suma a[9]=9 suma=0
 thread 1 suma a[10]=10 suma=0
 thread 1 suma a[11]=11 suma=0
 thread 1 suma a[12]=12 suma=0
 thread 1 suma a[13]=13 suma=0
 thread 1 suma a[14]=14 suma=0
 thread 1 suma a[15]=15 suma=0
 thread 2 suma a[5]=5 suma=0
Dentro de la region paralela:
num_threads: 3
num_proc:4
in_parallel:1

Fuera de la region paralela:
num_threads: 1
num_proc:4
in_parallel:0
```

```
$./scheduled-clauseM 16 2
 thread 3 suma a[6]=6 suma=0
 thread 3 suma a[7]=7 suma=0
 thread 0 suma a[4]=4 suma=0
Dentro de la region paralela:
num_threads: 4
num_proc:4
in_parallel:1
 thread 0 suma a[5]=5 suma=0
Dentro de la region paralela:
num_threads: 4
num_proc:4
in_parallel:1
 thread 0 suma a[10]=10 suma=0
Dentro de la region paralela:
num_threads: 4
num_proc:4
in_parallel:1
 thread 0 suma a[11]=11 suma=0
Dentro de la region paralela:
num_threads: 4
num_proc:4
in_parallel:1
 thread 0 suma a[12]=12 suma=0
Dentro de la region paralela:
num_threads: 4
num_proc:4
in_parallel:1
 thread 0 suma a[13]=13 suma=0
Dentro de la region paralela:
num_threads: 4
num_proc:4
in_parallel:1
 thread 0 suma a[14]=14 suma=0
Dentro de la region paralela:
num_threads: 4
num_proc:4
in_parallel:1
 thread 0 suma a[15]=15 suma=0
Dentro de la region paralela:
num_threads: 4
num_proc:4
in_parallel:1
 thread 1 suma a[0]=0 suma=0
 thread 1 suma a[1]=1 suma=0
 thread 2 suma a[2]=2 suma=0
 thread 2 suma a[3]=3 suma=0
 thread 3 suma a[8]=8 suma=0
 thread 3 suma a[9]=9 suma=0

Fuera de la region paralela:
num_threads: 1
num_proc:4
in_parallel:0
```

RESPUESTA:

Las funciones que obtienen distintos valores dependiendo de si están dentro o fuera de la región paralela son omp_get_num_threads() y omp_in_parallel(). Sin embargo la función omp_get_num_procs() se mantiene siempre igual.

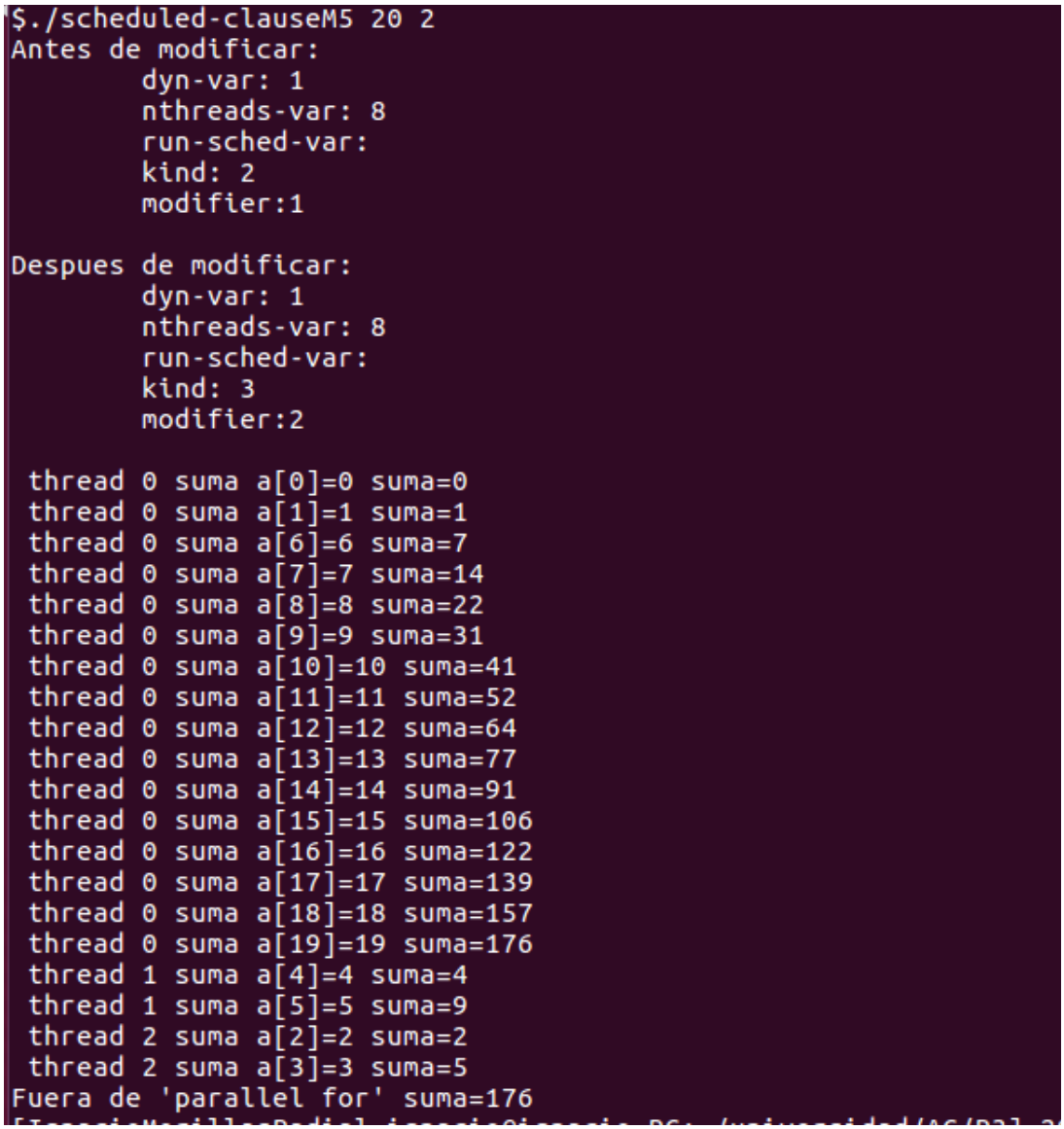
5. Añadir al programa `scheduled-clause.c` lo necesario para modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados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

CAPTURA 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;
omp_sched_t kind;
int modifier;
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;
// Antes de modificar
omp_get_schedule(&kind, &modifier);
printf("Antes de modificar:\n\tdyn-var: %d\n\tnthreads-var: %d\n\trun-sched-
var:\n\tkind: %d\n\tmodifier:
%d\n",omp_get_dynamic(),omp_get_max_threads(),kind,modifier);
// static = 1 ; dynmic = 2 ; guided = 3 ; auto = 4
// Realizamos los cambios
omp_set_dynamic(3);
omp_set_num_threads(8);
omp_set_schedule(3,2);
// Despues de modificar
omp_get_schedule(&kind, &modifier);
printf("\nDespues de modificar:\n\tdyn-var: %d\n\tnthreads-var: %d\n\trun-
sched-var:\n\tkind: %d\n\tmodifier:
%d\n\n",omp_get_dynamic(),omp_get_max_threads(),kind,modifier);
#pragma omp parallel 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);
return(0);
}
```

CAPTURAS DE PANTALLA:

```
$./scheduled-clauseM5 16 1
Antes de modificar:
        dyn-var: 1
        nthreads-var: 8
        run-sched-var:
        kind: 2
        modifier:1

Despues de modificar:
        dyn-var: 1
        nthreads-var: 8
        run-sched-var:
        kind: 3
        modifier:2

 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[3]=3 suma=3
 thread 0 suma a[4]=4 suma=7
 thread 0 suma a[5]=5 suma=12
 thread 0 suma a[6]=6 suma=18
 thread 0 suma a[7]=7 suma=25
 thread 0 suma a[8]=8 suma=33
 thread 0 suma a[9]=9 suma=42
 thread 0 suma a[10]=10 suma=52
 thread 0 suma a[11]=11 suma=63
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[13]=13 suma=15
 thread 1 suma a[14]=14 suma=29
 thread 0 suma a[12]=12 suma=75
 thread 1 suma a[15]=15 suma=44
 thread 2 suma a[1]=1 suma=1
Fuera de 'parallel for' suma=44
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05-08 martes
```



RESPUESTA:

Resto de ejercicios

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por 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 la primera y última componente del resultado antes de que termine el programa.

CAPTURA CÓDIGO FUENTE: `pmtv-secuencial.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char const *argv[]) {
  int **matriz;
```

```
  int *vector, *resultado;
  int tam, suma_aux;
  time_t t;
  // Inicializamos la semilla del rand
  srand((unsigned) time(&t));
  if(argc < 2){
    fprintf(stderr, "\nFalta tamaño de filas/columnas\n");
    exit(-1);
  }
  tam = atoi(argv[1]);
  // Reservo memoria
  vector = (int *) malloc(tam*sizeof(int));
  resultado = (int *) malloc(tam*sizeof(int));
  matriz = (int **) malloc(tam*sizeof(int*));
  // Inicializar vector y reservar memoria para matriz
  for (int i = 0; i < tam; i++) {
    matriz[i] = (int *) malloc(tam*sizeof(int));
  }
  // Inicializar matriz a 0
  for (int i = 0; i < tam; i++) {
    for (int j = 0; j < tam; j++) {
      matriz[i][j] = 0;
    }
  }
  // Inicializar valores por encima de diagonal principal de la matriz
  // e inicializar valores del vector
  for (int i = 0; i < tam; i++) {
    vector[i] = rand() % 20;
    for (int j = 0 + i; j < tam; j++) {
      matriz[i][j] = rand() % 20;
    }
  }
  // Realizar calculo
  for (int i = 0; i < tam; i++) {
    suma_aux = 0;
    for (int j = 0 + i; j < tam; j++) {
      suma_aux += vector[j] * matriz[i][j];
    }
    resultado[i] = suma_aux;
  }
  // Mostrar resultado
  printf("Primer elemento resultado: %d\n",resultado[0]);
  printf("Ultimo elemento resultado: %d\n",resultado[tam-1]);
  //Liberar memoria
  for (int i = 0; i < tam; i++) {
    free(matriz[i]);
  }
  free(matriz);
  free(vector);
  free(resultado);
  return 0;
}
```

CAPTURAS DE PANTALLA:

```
$gcc -O2 -fopenmp -o pmtv-secuencial pmtv-secuencial.c
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3
-08 martes
$./pmtv-secuencial 4
Primer elemento resultado: 256
Ultimo elemento resultado: 0
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3
-08 martes
$./pmtv-secuencial 4
Primer elemento resultado: 155
Ultimo elemento resultado: 20
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3
-08 martes
$./pmtv-secuencial 4
Primer elemento resultado: 155
Ultimo elemento resultado: 20
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3
-08 martes
$./pmtv-secuencial 4
Primer elemento resultado: 442
Ultimo elemento resultado: 90
```

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. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. 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 (usando, como siempre, -O2 al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 1, 64 y el `chunk` por defecto para la alternativa. Use un tamaño de vector N múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para `chunk` con `static`, `dynamic` y `guided`? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación `static` para cada uno de los `chunks`? (c) Con la asignación `dynamic` y `guided`, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

RESPUESTA:

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_PCaula.sh

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= , 12 threads

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			

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.

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char const *argv[]) {
  int **matrizA, **matrizB, **resultado;
  int tam;
  time_t t;
  // Inicializamos la semilla del rand
  srand((unsigned) time(&t));
  if(argc < 2){
    fprintf(stderr, "\nFalta tamaño de filas/columnas\n");
    exit(-1);
  }
  tam = atoi(argv[1]);
  // Reservo memoria
```

```
  matrizA = (int **) malloc(tam*sizeof(int*));
  matrizB = (int **) malloc(tam*sizeof(int*));
  resultado = (int **) malloc(tam*sizeof(int*));
  // Reservo memoria para matriz
  for (int i = 0; i < tam; i++) {
    matrizA[i] = (int *) malloc(tam*sizeof(int));
    matrizB[i] = (int *) malloc(tam*sizeof(int));
    resultado[i] = (int *) malloc(tam*sizeof(int));
  }
  // Inicializar matrices
  for (int i = 0; i < tam; i++) {
    for (int j = 0; j < tam; j++) {
      matrizA[i][j] = rand() % 10;
      matrizB[i][j] = rand() % 10;
    }
  }
  // Realizar calculo matrizA x matrizB
  for (int i = 0; i < tam; i++) {
    for (int j = 0; j < tam; j++) {
      for (int k = 0; k < tam; k++) {
        resultado[i][j] += matrizA[i][k] * matrizB[k][j];
      }
    }
  }
  // Mostrar resultado
  printf("Primer elemento resultado: %d\n",resultado[0][0]);
  printf("Ultimo elemento resultado: %d\n",resultado[tam-1][tam-1]);
  // Liberar memoria
  for (int i = 0; i < tam; i++) {
    free(matrizA[i]);
    free(matrizB[i]);
    free(resultado[i]);
  }
  free(matrizA);
  free(matrizB);
  free(resultado);
  return 0;
}
```

CAPTURAS DE PANTALLA:

```
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./pmm-secuencial 3
Primer elemento resultado: 69
Ultimo elemento resultado: 15
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./pmm-secuencial 3
Primer elemento resultado: 54
Ultimo elemento resultado: 104
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./pmm-secuencial 3
Primer elemento resultado: 55
Ultimo elemento resultado: 9
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./pmm-secuencial 3
Primer elemento resultado: 36
Ultimo elemento resultado: 47
```

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).

DESCOMPOSICIÓN DE DOMINIO:

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int main(int argc, char const *argv[]) {
  int **matrizA, **matrizB, **resultado;
  int tam,j,k;
  time_t t;
  double inicio, tiempo;
  // Inicializamos la semilla del rand
  srand((unsigned) time(&t));
  if(argc < 2){
    fprintf(stderr, "\nFalta tamaño de filas/columnas\n");
    exit(-1);
  }
  tam = atoi(argv[1]);
  // Reservo memoria
  matrizA = (int **) malloc(tam*sizeof(int*));
  matrizB = (int **) malloc(tam*sizeof(int*));
  resultado = (int **) malloc(tam*sizeof(int*));
  // Reservo memoria para matriz
  for (int i = 0; i < tam; i++) {
    matrizA[i] = (int *) malloc(tam*sizeof(int));
    matrizB[i] = (int *) malloc(tam*sizeof(int));
    resultado[i] = (int *) malloc(tam*sizeof(int));
  }
  // Inicializar matrices
  for (int i = 0; i < tam; i++) {
    for (int j = 0; j < tam; j++) {
      matrizA[i][j] = rand() % 10;
      matrizB[i][j] = rand() % 10;
    }
  }
  inicio = omp_get_wtime();
  // Realizar calculo matrizA x matrizB
  #pragma omp parallel for private(j,k) schedule(runtime)
  for (int i = 0; i < tam; i++) {
    for (j = 0; j < tam; j++) {
      for (k = 0; k < tam; k++) {
        resultado[i][j] += matrizA[i][k] * matrizB[k][j];
      }
    }
  }
  tiempo = omp_get_wtime() - inicio;
  // Mostrar resultado
  printf("Tiempo empleado: %11.9f\n",tiempo);
  printf("Primer elemento resultado: %d\n",resultado[0][0]);
```

```
  printf("Ultimo elemento resultado: %d\n",resultado[tam-1][tam-1]);
  // Liberar memoria
  for (int i = 0; i < tam; i++) {
    free(matrizA[i]);
    free(matrizB[i]);
    free(resultado[i]);
  }
  free(matrizA);
  free(matrizB);
  free(resultado);
  return 0;
}
```

CAPTURAS DE PANTALLA:

```
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./pmm-OpenMP 500
Tiempo empleado: 0.119535124
Primer elemento resultado: 9793
Ultimo elemento resultado: 10726
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./pmm-OpenMP 1000
Tiempo empleado: 1.287554785
Primer elemento resultado: 20604
Ultimo elemento resultado: 20601
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
-08 martes
$./pmm-OpenMP 1500
Tiempo empleado: 5.749824552
Primer elemento resultado: 30073
Ultimo elemento resultado: 31364
[IgnacioMorillasPadial ignacio@ignacio-PC:~/universidad/AC/P3] 2018-05
```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos del computador. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

ESTUDIO DE ESCALABILIDAD EN atcgrid:

SCRIPT: pmm-OpenMP_atcgrid.sh

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh