gi2º 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`

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv)
{
  int i,x, n=20, tid;
  int a[n],suma=0,sumalocal;

  if(argc < 3) {
    fprintf(stderr,"[ERROR]-Faltan parametros\n");
    exit(-1);
  }

  n = atoi(argv[1]);
  x = atoi(argv[2]);
  if (n>20) n=20;
  for (i=0; i<n; i++) {
    a[i] = i;
  }

  #pragma omp parallel if(n>4) num_threads(x) 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:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$./if-clauseModificado.eje 6 3
 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 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[5]=5 sumalocal=9
thread master=0 imprime suma=15
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$./if-clauseModificado.eje 6 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 1 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[4]=4 sumalocal=7
 thread 1 suma de a[5]=5 sumalocal=12
thread master=0 imprime suma=15
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$./if-clauseModificado.eje 4 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 master=0 imprime suma=6
```

RESPUESTA:

Como se ve se utilizan el numero de hebras pasadas sin tener que compilar, siempre que el numero de pasadas sea mayor que 4.

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	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	1	1	0	0	0	0
4	0	0	1	1	0	1	0	0	0

5	1	0	1	1	0	1	0	0	0
6	0	1	1	1	0	1	0	0	0
7	1	1	1	1	0	1	0	0	0
8	0	0	0	1	0	0	1	1	1
9	1	0	0	1	0	0	1	1	1
10	0	1	0	1	0	0	1	1	1
11	1	1	0	0	0	0	1	1	1
12	0	0	1	0	0	0	0	0	0
13	1	0	1	0	0	0	0	0	0
14	0	1	1	0	0	0	0	1	0
15	1	1	1	0	0	0	0	1	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	0	0	0	0
1	1	0	0	2	0	0	0	0	0
2	2	1	0	3	1	0	0	0	0
3	3	1	0	1	1	0	0	0	0
4	0	2	1	0	2	3	1	1	3
5	1	2	1	0	2	3	1	1	3
6	2	3	1	0	3	3	1	1	3
7	3	3	1	0	3	3	3	2	3
8	0	0	2	0	0	2	3	2	2
9	1	0	2	0	0	2	3	2	2
10	2	1	2	0	0	2	2	3	2
11	3	1	2	0	0	2	2	3	2
12	0	2	4	0	0	1	0	0	1
13	1	2	4	0	0	1	0	0	1
14	2	3	4	0	0	1	0	0	1
15	3	3	4	0	0	1	0	0	1

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

RESPUESTA:Static todas las hebras hacen las mismas iteraciones a no se que la ultima haga alguna mas. Con dinamic como mínimo hace chunk iteraciones y con guided las hebras se equilibran mas que con dinamic

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`

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

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
  #define omp_set_num_threads(int)
#endif

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

  int i, n=200, chunk, a[n], suma=0;
  omp_sched_t schedule_type;
  int chunk_value;

  if(argc < 3)
  {
    fprintf(stderr,"\nFalta iteraciones o chunk \n");
    exit(-1);
  }

  // omp_set_num_threads(4);

  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)
        {
            printf(" Dentro de 'parallel for':\n");


            printf("   static = 1, dynamic = 2, guided = 3, auto = 4\n");

            omp_get_schedule(&schedule_type, &chunk_value);
            printf("   dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d, chunk: %d\n", \
                omp_get_dynamic(),omp_get_max_threads(), omp_get_thread_limit(), schedule_type, chunk_value);
        }


  }

  printf("Fuera de 'parallel for' suma=%d\n",suma);
  printf("   static = 1, dynamic = 2, guided = 3, auto = 4\n");
    omp_get_schedule(&schedule_type, &chunk_value);
    printf("   dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d, chunk: %d\n", \
        omp_get_dynamic(),omp_get_max_threads(), omp_get_thread_limit(), schedule_type, chunk_value);

}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$./ejercicio3.eje 6 2
 thread 0 suma a[0]=0 suma=0
 Dentro de 'parallel for':
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:2147483647,run-sched-var
: 2, chunk: 1
 thread 0 suma a[1]=1 suma=1
 Dentro de 'parallel for':
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:2147483647,run-sched-var
: 2, chunk: 1
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 3 suma a[2]=2 suma=2
 thread 3 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=9
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:2147483647,run-sched-var
: 2, chunk: 1
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$


[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$export OMP_SCHEDULE="static,3"
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$export OMP_THREAD_LIMIT=10
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$./ejercicio3.eje 6 2
 thread 0 suma a[0]=0 suma=0
 thread 2 suma a[2]=2 suma=2
 thread 2 suma a[3]=3 suma=5
 Dentro de 'parallel for':
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:10,run-sched-var: 1, chu
nk: 3
 thread 0 suma a[1]=1 suma=1
 Dentro de 'parallel for':
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:10,run-sched-var: 1, chu
nk: 3
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
Fuera de 'parallel for' suma=9
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:10,run-sched-var: 1, chu
nk: 3
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
```

RESPUESTA: Se imprimen los mismos valores.

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`

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

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
  #define omp_set_num_threads(int)
  #define omp_in_parallel() 0
#endif

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

  int i, n=200, chunk, a[n], suma=0;
  omp_sched_t schedule_type;
  int chunk_value;

  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)
        {
            printf(" Dentro de 'parallel for':\n");

            printf("   static = 1, dynamic = 2, guided = 3, auto = 4\n");
            omp_get_schedule(&schedule_type, &chunk_value);
            printf("   dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d, chunk: %d\n", \
                 omp_get_dynamic(), \
                omp_get_max_threads(), omp_get_thread_limit(), \
                schedule_type, chunk_value);

            printf(" get_num_threads: %d,get_num_procs: %d,in_parallel():%d \n", \
                omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());

        }
  }

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

  printf("   static = 1, dynamic = 2, guided = 3, auto = 4\n");
  omp_get_schedule(&schedule_type, &chunk_value);
  printf("   dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d, chunk: %d\n" \
        , omp_get_dynamic(), \
      omp_get_max_threads(), omp_get_thread_limit(), \
      schedule_type, chunk_value);

  printf(" get_num_threads: %d,get_num_procs: %d,in_parallel():%d \n", \
      omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-14 Monday
$./ejercicio4.eje 6 2
 thread 3 suma a[2]=2 suma=2
 thread 3 suma a[3]=3 suma=5
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 0 suma a[0]=0 suma=0
 Dentro de 'parallel for':
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:10,run-sched-var: 1, chu
nk: 3
 get_num_threads: 4,get_num_procs: 4,in_parallel():1
 thread 0 suma a[1]=1 suma=1
 Dentro de 'parallel for':
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:10,run-sched-var: 1, chu
nk: 3
 get_num_threads: 4,get_num_procs: 4,in_parallel():1
Fuera de 'parallel for' suma=9
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:10,run-sched-var: 1, chu
nk: 3
 get_num_threads: 1,get_num_procs: 4,in_parallel():0
```

RESPUESTA: La única de las nuevas que no cambia es 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.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

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

  int i, n=200, chunk, a[n], suma=0;
  omp_sched_t schedule_type;
  int chunk_value;

  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;

  // Imprimimos antes del cambio
  printf("Antes del cambio\n");
  printf("   static = 1, dynamic = 2, guided = 3, auto = 4\n");
  omp_get_schedule(&schedule_type, &chunk_value);
  printf("   dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d, chunk: %d\n" \
        , omp_get_dynamic(), \
      omp_get_max_threads(), omp_get_thread_limit(), \
      schedule_type, chunk_value);

  printf(" get_num_threads: %d,get_num_procs: %d,in_parallel():%d \n", \
      omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());

  // Cambiamos los valores
  omp_set_dynamic(2);
  omp_set_num_threads(2);
  omp_set_schedule(2, 1);

  #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);

  // Imprimimos despues del cambio
  printf("   static = 1, dynamic = 2, guided = 3, auto = 4\n");
  omp_get_schedule(&schedule_type, &chunk_value);
  printf("   dyn-var: %d, nthreads-var:%d, thread-limit-var:%d,run-sched-var: %d, chunk: %d\n" \
        , omp_get_dynamic(), \
      omp_get_max_threads(), omp_get_thread_limit(), \
      schedule_type, chunk_value);

  printf(" get_num_threads: %d,get_num_procs: %d,in_parallel():%d \n", \
      omp_get_num_threads(),omp_get_num_procs(),omp_in_parallel());
}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-15 Tuesday
$./ejercicio5.eje 6 2
Antes del cambio
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 0, nthreads-var:4, thread-limit-var:10,run-sched-var: 1, chu
nk: 3
 get_num_threads: 1,get_num_procs: 4,in_parallel():0
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[4]=4 suma=5
 thread 0 suma a[5]=5 suma=10
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=10
   static = 1, dynamic = 2, guided = 3, auto = 4
   dyn-var: 1, nthreads-var:2, thread-limit-var:10,run-sched-var: 2, chu
nk: 1
 get_num_threads: 1,get_num_procs: 4,in_parallel():0
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
```

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`

```c
int main(int argc, char **argv)
{
    int i, j;

    //Leer argumento de entrada
    if(argc < 2)
    {
        fprintf(stderr, "Falta size\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1

    // Inicializamos la matriz triangular (superior)
    int *vector, *result, **matrix;
    vector = (int *) malloc(N*sizeof(int)); // malloc ne
    result = (int *) malloc(N*sizeof(int)); //si no hay
    matrix = (int **) malloc(N*sizeof(int*));

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

    for (i=0; i<N; i++)
    {
        for (j=i; j<N; j++)
            matrix[i][j] = 5;
        vector[i] = 5;
        result[i]=0;
    }

    // Pintamos la matriz
    printf("Matriz:\n");
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            if (j >= i)
                printf("%d ", matrix[i][j]);
            else
                printf("0 ");
        }
        printf("\n");
    }

    // Pintamos el vector
    printf("Vector:\n");
    for (i=0; i<N; i++)
        printf("%d ", vector[i]);
    printf("\n");

    // Obtenemos los resaultados
    for (i=0; i<N; i++)
        for (j=i; j<N; j++)
            result[i] += matrix[i][j] * vector[j];

    // Pintamos los resultados
    printf("Resultado:\n");
    for (i=0; i<N; i++)
        printf("%d ", result[i]);
    printf("\n");

    // Liberamos la memoria
    for (i=0; i<N; i++)
        free(matrix[i]);
    free(matrix);
    free(vector);
    free(result);
```

```
$./ejercicio6.eje 8
Matriz:
5 5 5 5 5 5 5 5
0 5 5 5 5 5 5 5
0 0 5 5 5 5 5 5
0 0 0 5 5 5 5 5
0 0 0 0 5 5 5 5
0 0 0 0 0 5 5 5
0 0 0 0 0 0 5 5
0 0 0 0 0 0 0 5
Vector:
5 5 5 5 5 5 5 5
Resultado:
200 175 150 125 100 75 50 25
```

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: Parce algo mejor static.

a- statict no tiene y dinamyc y guided es 1.

b- chunk * numero de filas

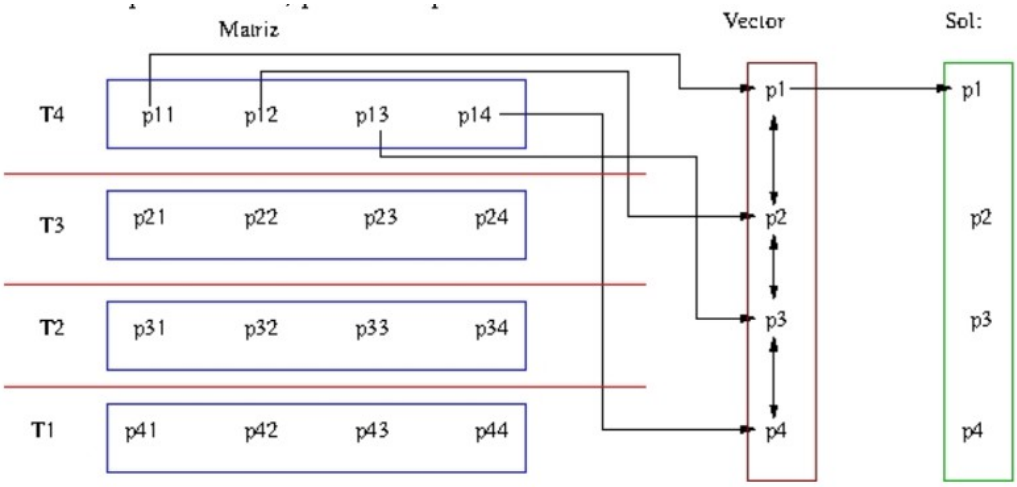
c-

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
// Inicializamos la matriz triangular (superior)
int *vector, *result, **matrix;
vector = (int *) malloc(N*sizeof(int)); // malloc necesita el tamaño en bytes
result = (int *) malloc(N*sizeof(int)); //si no hay espacio suficiente malloc
matrix = (int **) malloc(N*sizeof(int*));

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

for (i=0; i<N; i++)
{
    for (j=i; j<N; j++)
        matrix[i][j] = 2;
    vector[i] = 4;
    result[i]=0;
}

if (debug==1)
{
    // Pintamos la matriz
    printf("Matriz:\n");
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            if (j >= i)
                printf("%d ", matrix[i][j]);
            else
                printf("0 ");
        }
        printf("\n");
    }

    // Pintamos el vector
    printf("Vector:\n");
    for (i=0; i<N; i++)
        printf("%d ", vector[i]);
    printf("\n");
}

double start, end, total;
start = omp_get_wtime();


// Obtenemos los resultados

// Usamos runtime para poder variarlo luego con la variable OMP_SCHEDULE
#pragma omp parallel for private(j) schedule(runtime)
for (i=0; i<N; i++)
    for (j=i; j<N; j++)
        result[i] += matrix[i][j] * vector[j];

end = omp_get_wtime();
total = end - start;

if (debug==1)
{
    // Pintamos los resultados
    printf("Resultado:\n");
    for (i=0; i<N; i++)
        printf("%d ", result[i]);
    printf("\n");
}
```

CAPTURAS DE PANTALLA:

DESCOMPOSICIÓN DE DOMINIO:



SCRIPT: pmvt-OpenMP_PCaula.sh

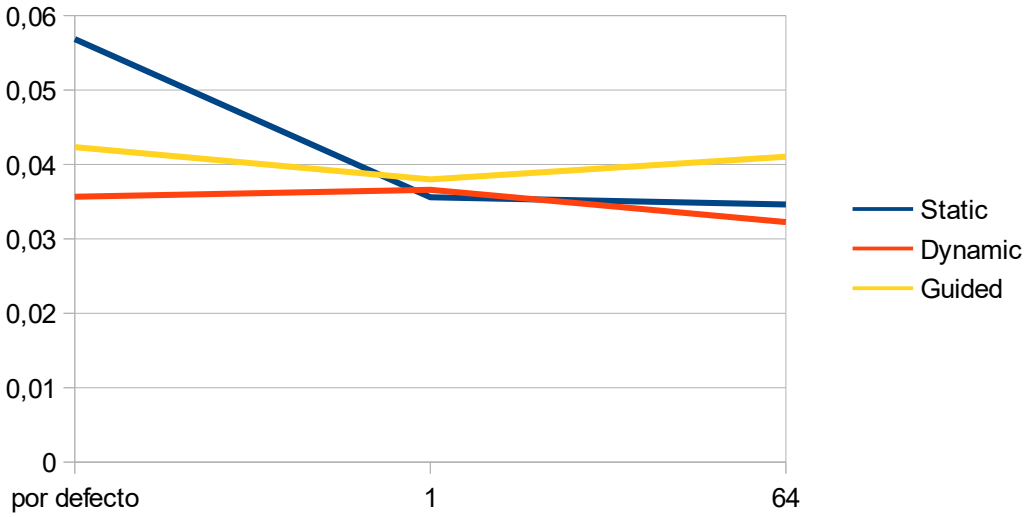
```
export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
$PBS_O_WORKDIR/ejercicio7.eje 15360

export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
$PBS_O_WORKDIR/ejercicio7.eje 15360

export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
$PBS_O_WORKDIR/ejercicio7.eje 15360

export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
$PBS_O_WORKDIR/ejercicio7.eje 15360

export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
$PBS_O_WORKDIR/ejercicio7.eje 15360

export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
$PBS_O_WORKDIR/ejercicio7.eje 15360

export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
$PBS_O_WORKDIR/ejercicio7.eje 15360

export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
$PBS_O_WORKDIR/ejercicio7.eje 15360

export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
$PBS_O_WORKDIR/ejercicio7.eje 15360
```

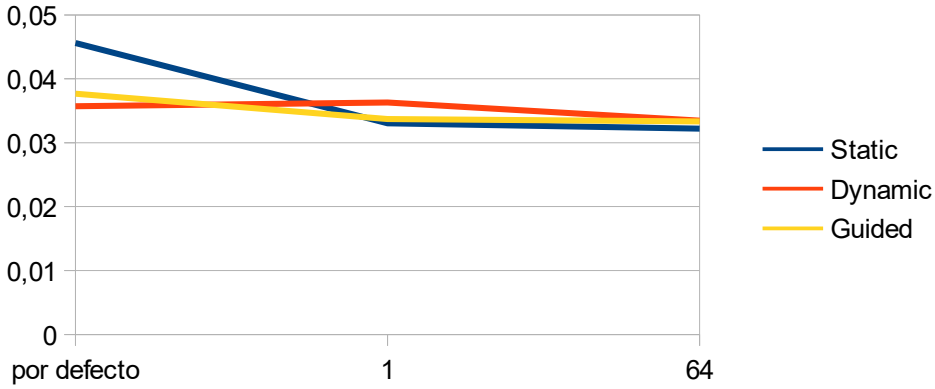
TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

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	0,045611616	0,035698784	0,037658477
1	0,033047001	0,036286262	0,033698742
64	0,032233385	0,033451399	0,033354582

Chunk	Static	Dynamic	Guided
por defecto	0,056859401	0,035658425	0,042333559
1	0,035580593	0,036584586	0,038002658
64	0,034628693	0,032245697	0,041025485





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`

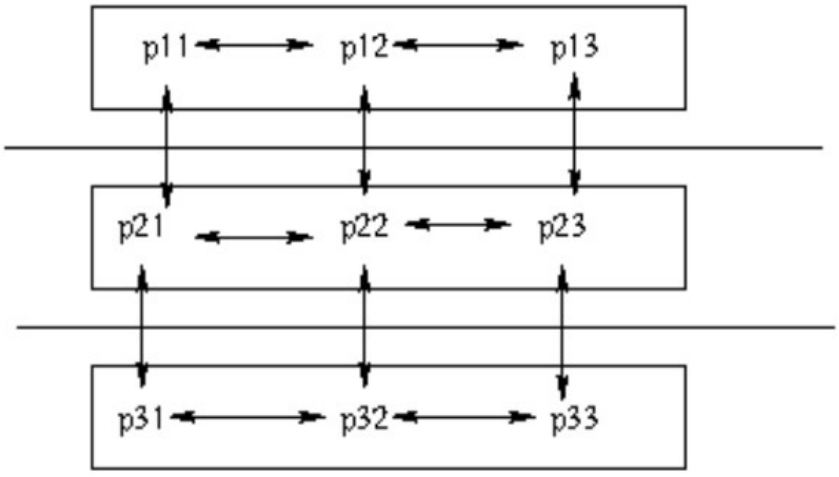
```c
int main(int argc, char const *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "ERROR: falta numero de filas y columnas\n");
        exit(1);
    }

    unsigned n, i, j, k;
    n = strtol(argv[1], NULL, 10);

    int **a, **b, **c;
    a = (int **) malloc(n*sizeof(int*));
    b = (int **) malloc(n*sizeof(int*));
    c = (int **) malloc(n*sizeof(int*));
    for (i=0; i<n; i++) {
        a[i] = (int *) malloc(n*sizeof(int));
        b[i] = (int *) malloc(n*sizeof(int));
        c[i] = (int *) malloc(n*sizeof(int));
    }

    // Inicializcion
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            a[i][j] = 0;
            b[i][j] = /*i+1*/2;
            c[i][j] = /*j+1*/1;
        }
    }

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);
    // Multiplicacion
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            for (k=0; k<n; k++)
                a[i][j] += b[i][k] * c[k][j];
    clock_gettime(CLOCK_REALTIME,&cgt2);

    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    if (n < 15) {
        printf("M1:\n");
        printMatriz(n, b);
        printf("M2:\n");
        printMatriz(n, c);
        printf("Sol:\n");
        printMatriz(n, a);
    }
    else
        printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[n-1][n-1]);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-15 Tuesday
$./ejercicio8.eje 8
M1:
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
M2:
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
Sol:
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-15 Tuesday
```

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`

```c
int main(int argc, char **argv)
{
    unsigned i, j, k;

    if(argc < 2)
    {
        fprintf(stderr, "falta size\n");
        exit(-1);
    }


    unsigned int N = atoi(argv[1]); // Maximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    int **a, **b, **c;
    a = (int **) malloc(N*sizeof(int*));
    b = (int **) malloc(N*sizeof(int*));
    c = (int **) malloc(N*sizeof(int*));

    for (i=0; i<N; i++)
    {
        a[i] = (int *) malloc(N*sizeof(int));
        b[i] = (int *) malloc(N*sizeof(int));
        c[i] = (int *) malloc(N*sizeof(int));
    }

    // Inicializamos las matrices
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            a[i][j] = 0;
            b[i][j] = 2;
            c[i][j] = 2;
        }
    }

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);
    // Multiplicacion
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            for (k=0; k<N; k++)
                a[i][j] += b[i][k] * c[k][j];
    clock_gettime(CLOCK_REALTIME,&cgt2);

    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    // Pitamos la primera y la ultima linea de la matriz resultante
    printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[N-1][N-1]);

    // Liberamos la memoria
    for (i=0; i<N; i++)
    {
        free(a[i]);
        free(b[i]);
        free(c[i]);
    }
    free(a);
    free(b);
    free(c);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica3/
Codigos] 2018-05-15 Tuesday
$./ejercicio9.eje 15
Tiempo = 0.000004900     Primera = 60    Ultima=60
```