2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Elena Ortiz Moreno
Grupo de prácticas: B2
Fecha de entrega: 13/05/21
Fecha evaluación en clase:14/05/21

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

Ejercicios basados en los ejemplos del seminario práctico

1. 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, n=20, tid;
    int a[n], suma=0, sumalocal;

    if (argc < 2)
    {
        fprintf(stderr,"[ERROR] Falta el nÃºmero de 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 if(n>4) default(none) \
                private(sumalocal,tid) shared(a,suma,n) num_threads(x)
    {
        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:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer1$ gcc -fopenmp -O2 if-clauseModificado.c -o if-clauseModificado
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer1$ ./if-clauseModificado 6 10
 thread 0 suma de a[0]=0 sumalocal=0
 thread 4 suma de a[4]=4 sumalocal=4
 thread 3 suma de a[3]=3 sumalocal=3
 thread 2 suma de a[2]=2 sumalocal=2
 thread 5 suma de a[5]=5 sumalocal=5
 thread 1 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=15
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer1$ ./if-clauseModificado 3 7
 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 master=0 imprime suma=3
```

RESPUESTA: Con estos ejemplos comprobamos que el funcionamiento de la cláusula es correcto, ya que solo se activa a partir de las 4 hebras. Una vez cumplimos esta comprobación y se activa, se reparten las iteraciones de la suma total entre los threads.

2. Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) usando `scheduler-clause.c` con tres *threads* (0,1,2) y un número de iteraciones de 16 (0 a 15 en la tabla). Con este ejercicio se pretende comparar distintas alternativas de planificación de bucles. Se van a usar distintos tipos (`static`, `dynamic`, `guided`), modificadores (`monotonic` y `nonmonotonic`) y tamaños de chunk (x = 1, 2 y 4).

Tabla 1 . Tabla schedule. Rellenar esta tabla ejecutando `scheduler-clause.c` asignando previamente a la variable de entorno `OMP_SCHEDULE` los valores que se indican en la tabla (por ej.: `export OMP_SCHEDULE="nonmonotonic:static,2)`. En la segunda fila, 1, 2 4 representan el tamaño del chunk

Iteración	"monotonic:static,x"		"nonmonotonic:static,x"		"monotonic:dynamic,x"		"monotonic:guided,x"	
	x=1	x=2	x=1	x=2	x=1	x=2	x=1	x=2
0	*1*	2	2	3	1	2	1	1
1	2	1	0	3	2	1	2	2
2	0	0	3	3	3	3	3	0
3	3	3	1	3	0	0	0	3
4	2	2	2	3	1	1	0	1
5	2	2	2	3	1	1	0	1
6	2	2	2	3	1	1	0	1
7	2	2	2	3	1	1	0	1
8	2	2	2	3	1	1	0	1
9	2	2	2	3	1	1	0	1
10	2	2	2	3	1	1	0	1
11	2	2	2	3	1	1	0	1
12	2	2	2	3	1	1	0	1
13	2	2	2	0	1	1	0	1
14	2	2	2	2	1	1	0	1
15	0	2	2	1	1	1	0	1

Destacar las diferencias entre las 4 alternativas de planificación de la tabla, en particular, las que hay entre `static`, `dynamic` y `guided` y las diferencias entre usar `monotonic` y `nonmonotonic`.

RESPUESTA: Con static todas las hebras hacen prácticamente las mismas iteraciones en round-robin. Con dynamic el orden y el reparto no se puede saber, lo único que se puede saber es que como mínimo cada hebra hará chunk iteraciones. Lo mismo pasa con guided, con la diferencia de que las iteraciones están más "equilibradas" entre las hebras y el número de iteraciones que hace cada hebra no es múltiplo de chunk.
El reparto de las iteraciones depende del chunk. Guided funciona mejor que dynamic con el mismo chunk.
La diferencia entre usar monotonic y nonmonotonic es que monotonic solo puede ir en una dirección (hacia delante) y nonmonotonic permite ir en dos direcciones (hacia atrás y hacia delante).

3. ¿Qué valor por defecto usa OpenMP para `chunk` y `modifier` con `static`, `dynamic` y `guided`? Explicar qué ha hecho para contestar a esta pregunta.

Dynamic y guided usan por defecto valor de chunk 1 y static utiliza el 0. Se obtiene con omp_get_schedule.

4. 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
#endif

int main(int argc, char **argv) {
  int i, n=200, chunk, a[n], suma=0, modifier;
  omp_sched_t kind;
   if(argc < 3)     {
      fprintf(stderr,"Falta 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
   {
      if(omp_get_thread_num() == 0)
      {
        printf("Dentro de 'parallel':\n");
        printf("dyn-var: %d\n",omp_get_dynamic());
        printf("nthreads-var:%d\n",omp_get_max_threads());
        printf("thread-limit-var:%d\n",omp_get_thread_limit());
        omp_get_schedule(&kind, &modifier);
        printf("run-sched-var:1ºparámetro:%d,2ºparámetro:%d\n",(int) kind,modifier);
      }
      #pragma omp for firstprivate(suma) \
            lastprivate(suma) schedule(dynamic,chunk)
      int chunk_size;
      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("\tchunk: %d\n", chunk_size);
      }
   }

   printf("Fuera de 'parallel':\n");
   printf("Valor variable suma:%d\n",suma);
   printf("Valor variable dyn-var:%d\n",omp_get_dynamic());
   printf("Valor variable nthreads-var:%d\n",omp_get_max_threads());
   printf("Valor variable thread-limit-var:%d\n",omp_get_thread_limit());
   omp_get_schedule(&kind, &modifier);
   printf("Valor variable run-sched-var:1ºparámetro:%d,2ºparámetro:%d\n",(int) kind,modifier);
   printf("\tchunk: %d\n", chunk_size);

}
```

CAPTURAS DE PANTALLA:

Con los valores iniciales:

Tras modificar el número de hebras desde terminal:

RESPUESTA: Se imprimen los mismos valores tanto dentro como fuera del parallel.

5. 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
#endif

int main(int argc, char **argv) {
  int i, n=200, chunk, a[n], suma=0, modifier;
  omp_sched_t kind;
   if(argc < 3)     {
      fprintf(stderr,"Falta 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
   {
      if(omp_get_thread_num() == 0)
      {
        printf("Dentro de 'parallel':\n");
        printf("dyn-var: %d\n",omp_get_dynamic());
        printf("nthreads-var:%d\n",omp_get_max_threads());
        printf("thread-limit-var:%d\n",omp_get_thread_limit());
        omp_get_schedule(&kind, &modifier);
        printf("run-sched-var:1ºparámetro:%d,2ºparámetro:%d\n",(int) kind,modifier);
        printf("omp_get_num_threads:%d\n",omp_get_num_threads());
        printf("omp_get_num_procs:%d\n",omp_get_num_procs());
        printf("omp_in_parallel:%d\n",omp_in_parallel());
      }
      #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':\n");
   printf("Valor variable suma:%d\n",suma);
   printf("Valor variable dyn-var:%d\n",omp_get_dynamic());
   printf("Valor variable nthreads-var:%d\n",omp_get_max_threads());
   printf("Valor variable thread-limit-var:%d\n",omp_get_thread_limit());
   omp_get_schedule(&kind, &modifier);
   printf("Valor variable run-sched-var:1ºparámetro:%d,2ºparámetro:%d\n",(int) kind,modifier);
}
```

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer5$ gcc -fopenmp -O2 scheduled-clauseModificado4.c -o scheduled-clauseModificado4
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer5$ ./scheduled-clauseModificado4 5 3
Dentro de 'parallel':
dyn-var: 0
nthreads-var:4
thread-limit-var:2147483647
run-sched-var:1ºparámetro:2,2ºparámetro:1
omp_get_num_threads:4
omp_get_num_procs:4
omp_in_parallel:1
 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 1 suma a[3]=3 suma=3
 thread 1 suma a[4]=4 suma=7
Fuera de 'parallel':
Valor variable suma:7
Valor variable dyn-var:0
Valor variable nthreads-var:4
Valor variable thread-limit-var:2147483647
Valor variable run-sched-var:1ºparámetro:2,2ºparámetro:1
```

RESPUESTA: La única variable constante es omp_get_num_procs(). Las demás varían dependiendo de si estamos dentro de la región paralela o fuera.

6. Añadir al programa `scheduled-clause.c` lo necesario para, usando funciones, modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` dentro de la región paralela y fuera de la región paralela. En la modificación de `run-sched-var` se debe usar un valor de `kind` distinto al utilizado en la cláusula `schedule()`. Añadir lo necesario para imprimir el contenido de estas variables antes y después de cada una de las dos modificaciones. Comentar los resultados.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado5.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=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;

  int modifier; omp_sched_t kind;

  #pragma omp parallel for firstprivate(suma) \
  lastprivate(suma) schedule(dynamic,chunk)
  for (i=0; i<n; i++) {
    if ( i == 0 )
    {
      printf("En parallel: dyn-var=%d, nthreads-var=%d, threadlimit-var=%d\n",
        omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit());
      printf("dyn-var: %d \n",omp_get_dynamic());
       omp_set_dynamic(3);
       printf("Modificamos dyn-var con omp_set_dynamic(1) y el resultado es: %d\n", omp_get_dynamic());
       printf("nthreads-var: %d \n",omp_get_max_threads());
       omp_set_num_threads(8);
       printf("Modificamos nthreads-var con omp_set_num_threads(8); y el resultado es: %d\n",omp_get_max_threads());

       omp_get_schedule(&kind,&modifier);
       printf("run-sched-var: (Kind: %d, Modifier: %d)\n",kind,modifier);
       omp_set_schedule(2,2);
       omp_get_schedule(&kind,&modifier);
       printf("Modificamos run-sched-var con omp_set_schedule(2,2) y el resultado de Kind es %d y el de Modifier es: %d \n",kind,modifier);
    }
    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("Además, dyn-var=%d, nthreads-var=%d, threadlimit-var=%d ",
    omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit());
  omp_get_schedule(&kind,&modifier);
  printf("run-sched-var: (Kind: %d. Modifier: %d)\n",kind,modifier);
}
```

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer6$ gcc -fopenmp scheduled-clauseModificado5.c -o schedule
d-clauseModificado5
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer6$ ./scheduled-clauseModificado5 10 4
En parallel: dyn-var=0, nthreads-var=4, threadlimit-var=2147483647
dyn-var: 0
Modificamos dyn-var con omp_set_dynamic(1) y el resultado es: 1
nthreads-var: 4
Modificamos nthreads-var con omp_set_num_threads(8); y el resultado es: 8
run-sched-var: (Kind: 2, Modifier: 1)
Modificamos run-sched-var con omp_set_schedule(2,2) y el resultado de Kind es 2 y el de Modifier es: 2
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 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 de 'parallel for' suma=17
Además, dyn-var=0, nthreads-var=4, threadlimit-var=2147483647 run-sched-var: (Kind: 2. Modifier: 1)
```

RESPUESTA: Desde la línea de comandos se leen las opciones correspondientes y se modifican con las funciones correspondientes.

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

7. Implementar un programa secuencial en C que multiplique una matriz triangular inferior por un vector (use variables dinámicas y tipo de datos double). Comparar el orden de complejidad y el número total de operaciones (sumas y productos) de este código respecto al 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
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
#include <string.h>

int main ( int argc, char **argv ){
  if (argc < 2){
    perror("Falta tamaño de la matriz\n");
    return EXIT_FAILURE;
  }

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

  if(N == 0){
    perror("Tamaño invalido");
    return EXIT_FAILURE;
  }

  double *vector, *result, **matriz;
  vector = (double *) malloc(N*sizeof(double));
  result = (double *) malloc(N*sizeof(double));
  matriz = (double **) malloc(N*sizeof(double*));
  int i, j;

  if (vector == NULL || result == NULL || matriz == NULL){
    perror("Error en la asignación de memoria");
    return EXIT_FAILURE;
  }

  for (int i = 0; i < N; i++){
    if ((matriz[i] = (double*) malloc(N*sizeof(double))) == NULL){
      perror("Error en la asignación de memoria para la matriz");
      return EXIT_FAILURE;
    }
  }

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

  for (i = 0; i < N; i++){
    for(j = 0; j < N; j++)
      matriz[i][j] = 3.0;
    vector[i] = 5.0;
    result[i] = 0.0;
  }

  printf("Matriz:\n");
  for (i = 0; i < N; i++){
    for (j = 0; j < N; j++){
      if (j >= i)
        printf("%5.2f ", matriz[i][j]);
      else
        printf("0 ");
    }
    printf("\n");
  }

  printf("vector:\n");
  for (i = 0; i < N; i++)
    printf("%5.2f ", vector[i]);
  printf("\n");

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

  result[N-1] = matriz[N-1][N-1] * vector[N-1];

  printf("Resultado:\n");
  for (i = 0; i < N; i++)
    printf("%5.2f ", result[i]);
  printf("\n");

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

  free(matriz);
  free(vector);
  free(result);

  return EXIT_SUCCESS;
}
```

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer7$ gcc -fopenmp -O2 pmtv-secuencial.c -o pmtv-secuencial
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer7$ ./pmtv-secuencial 3
Matriz:
 3.00  3.00  3.00
0  3.00  3.00
0 0  3.00
vector:
 5.00  5.00  5.00
Resultado:
45.00 45.00 15.00
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer7$ ./pmtv-secuencial 8
Matriz:
 3.00  3.00  3.00  3.00  3.00  3.00  3.00  3.00
0  3.00  3.00  3.00  3.00  3.00  3.00  3.00
0 0  3.00  3.00  3.00  3.00  3.00  3.00
0 0 0  3.00  3.00  3.00  3.00  3.00
0 0 0 0  3.00  3.00  3.00  3.00
0 0 0 0 0  3.00  3.00  3.00
0 0 0 0 0 0  3.00  3.00
0 0 0 0 0 0 0  3.00
vector:
 5.00  5.00  5.00  5.00  5.00  5.00  5.00  5.00
Resultado:
120.00 120.00 120.00 120.00 120.00 120.00 120.00 15.00
```

8. Implementar en paralelo la multiplicación de una matriz triangular inferior 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. La inicialización de los datos la debe hacer el thread 0. 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`. Mostrar en una captura de pantalla que el código resultante funciona correctamente. NOTA: usar para generar los valores aleatorios, por ejemplo, `drand48_r()`.

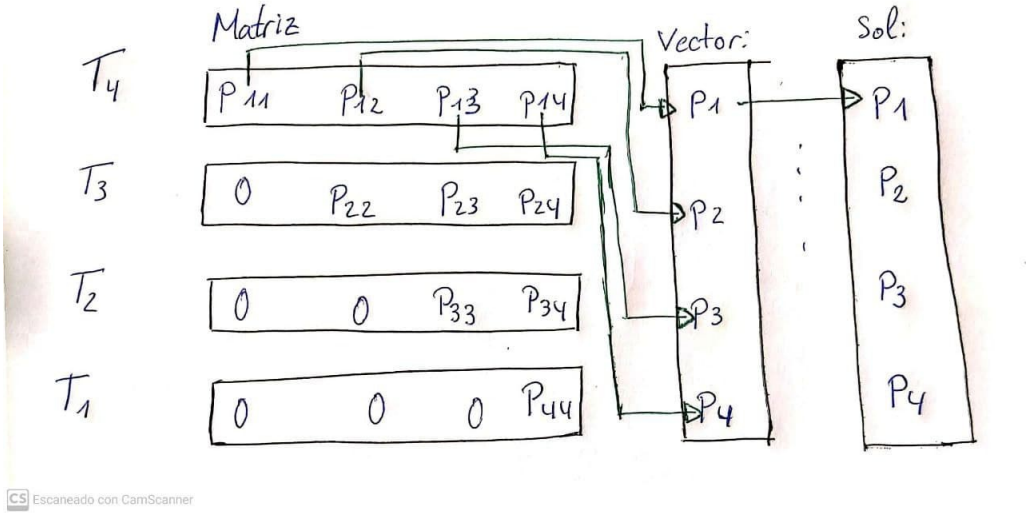
CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #include <omp.h>
#endif
#include <string.h>

#define FINAL

int main ( int argc, char **argv ){
  if (argc < 2){
    perror("Falta tamaño de la matriz\n");
    return EXIT_FAILURE;
  }

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

  if(N == 0){
    perror("Tamaño invalido");
    return EXIT_FAILURE;
  }

  double *vector, *result, **matriz;
  vector = (double *) malloc(N*sizeof(double));
  result = (double *) malloc(N*sizeof(double));
  matriz = (double **) malloc(N*sizeof(double*));
  int i, j;

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

  for (i=0; i<N; i++) {
    for (j=i; j<N; j++)
      matriz[i][j] = 3;
    vector[i] = 5;
    result[i] = 0;
  }
```

```c
#ifdef TESTING
printf("Matriz:\n");
for (i=0; i<N; i++) {
    for (j=0; j<N; j++){
        if (j >= i)
            printf("%5.2f ", matriz[i][j]);
        else
            printf("0 ");
    }
    printf("\n");
}

  printf("vector:\n");
  for (i=0; i<N; i++)
      printf("%5.2f ", vector[i]);
  printf("\n");
  #endif

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

  #pragma omp parallel for private(j) schedule(runtime)
  for (i=0; i<N; i++)
      for (j=i; j<N; j++)
          result[i] += matriz[i][j] * vector[j];

  result[N-1] = matriz[N-1][N-1] * vector[N-1];
  end = omp_get_wtime();
  tiempo = end - start;

  printf("Matriz:\n");
  for (i = 0; i < N; i++){
    for (j = 0; j < N; j++){
      if (j >= i)
        printf("%5.2f ", matriz[i][j]);
      else
        printf("0 ");
    }
    printf("\n");
  }
```

```c
    printf("vector:\n");
    for (i = 0; i < N; i++)
      printf("%5.2f ", vector[i]);
    printf("\n");

    printf("Resultado:\n");
    for (i=0; i<N; i++)
        printf("%5.2f ", result[i]);
    printf("\n");


    #ifdef FINAL
    printf("Tiempo = %11.9f\t Primera = %5.2f\t Ultima=%5.2f\n",
    tiempo,result[0],result[N-1]);
    #endif


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

  free(matriz);
  free(vector);
  free(result);

  return EXIT_SUCCESS;
}
```

DESCOMPOSICIÓN DE DOMINIO:



La inicialización de los datos la debe hacer el thread 0 → significa que la inicialización se debe hacer de forma secuencial, es decir, fuera del for de OpenMP

CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer8$ gcc -fopenmp -O2 pmtv-OpenMP.c -o pmtv-OpenMP
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer8$ ./pmtv-OpenMP 3
Matriz:
 3.00  3.00  3.00
0  3.00  3.00
0 0  3.00
vector:
 5.00  5.00  5.00
Resultado:
45.00 30.00 15.00
Tiempo = 0.009971317      Primera = 45.00         Ultima=15.00
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer8$ ./pmtv-OpenMP 8
Matriz:
 3.00  3.00  3.00  3.00  3.00  3.00  3.00  3.00
0  3.00  3.00  3.00  3.00  3.00  3.00  3.00
0 0  3.00  3.00  3.00  3.00  3.00  3.00
0 0 0  3.00  3.00  3.00  3.00  3.00
0 0 0 0  3.00  3.00  3.00  3.00
0 0 0 0 0  3.00  3.00  3.00
0 0 0 0 0 0  3.00  3.00
0 0 0 0 0 0 0  3.00
vector:
 5.00  5.00  5.00  5.00  5.00  5.00  5.00  5.00
Resultado:
120.00 105.00 90.00 75.00 60.00 45.00 30.00 15.00
Tiempo = 0.000273529      Primera = 120.00        Ultima=15.00
```

9. Contestar a las siguientes preguntas sobre el código del ejercicio anterior:

(a) ¿Qué número de operaciones de multiplicación y qué número de operaciones de suma realizan cada uno de los threads en la asignación `static` con `monotonic` y un `chunk` de 1?

RESPUESTA:

OpenMP divide las iteraciones entre las hebras con chunks de tamaños 0, 1 y 64 y los distrubuye entre las hebras de manera en que si quedan chunks y se llega a la última hebra el reparto comienza de nuevo por la hebra principal. Si tenemos chunk=0 el reparto es equitativo, con chunk=1 cada hebra se encarga de una iteración y con chunk=64 cada una se encarga de 64 iteraciones.

(b) 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:

El reparto no será uniforme, a diferencia de lo que ocurrirá en static.

(c) ¿Qué alternativa ofrece mejores prestaciones? Razonar la respuesta.

RESPUESTA:

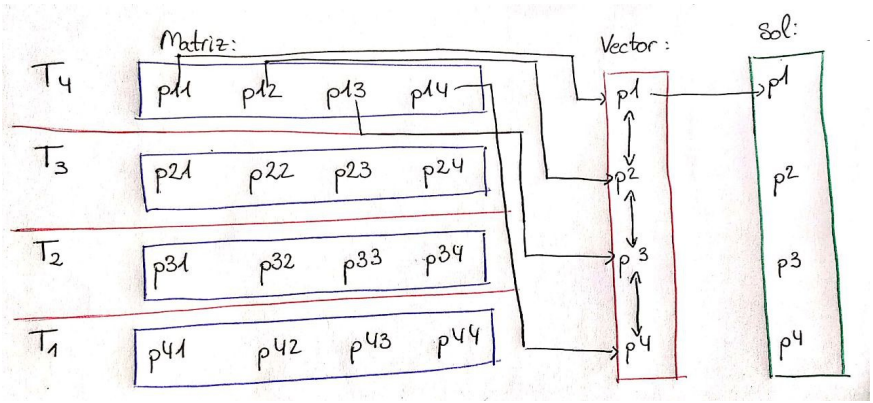
Depende del chunk que se utilice y del tamaño del vector.

10. 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 (con `monotonic` en todos los casos). Usar un tamaño de vector N múltiplo del número de cores y de 64 que esté entre 11520 y 23040. El número de threads en las ejecuciones debe coincidir con el número de núcleos del computador. 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 (representar los valores de las dos tablas). Incluir los scripts utilizado en el cuaderno de prácticas.
NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #include <omp.h>
#endif
#include <string.h>

// #define TESTING
#define FINAL

int main(int argc, char const *argv[])
{
    if (argc < 2) {
        fprintf(stderr, "Falta el tamanio de la matriz\n");
        exit(-1);
    }

    int N = atoi(argv[1]);

    int **matriz;
    int *v1;
    int *v2;
    int i, j;

    matriz = (int **) malloc(N*sizeof(int*));
    v1 = (int *) malloc(N*sizeof(int));
    v2 = (int *) malloc(N*sizeof(int));
    for (i=0; i<N; i++)
        matriz[i] = (int*) malloc(N*sizeof(int));

    #pragma omp parallel for private(j) schedule(runtime)
    for (i=0; i<N; i++) {
        for (j=i; j<N; j++)
            matriz[i][j] = 3;
        v1[i] = 5;
        v2[i] = 0;
    }

    #ifdef TESTING
    printf("Matriz:\n");
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++){
            if (j >= i)
                printf("%d ", matriz[i][j]);
            else
                printf("0 ");
        }
        printf("\n");
    }
    printf("v1:\n");
    for (i=0; i<N; i++)
        printf("%d ", v1[i]);
    printf("\n");
    #endif

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

    #pragma omp parallel for private(j) schedule(runtime)
    for (i=0; i<N; i++)
        for (j=i; j<N; j++)
            v2[i] += matriz[i][j] * v1[j];

    v2[N-1] = matriz[N-1][N-1] * v1[N-1];
    end = omp_get_wtime();
    tiempo = end - start;

    #ifdef TESTING
    printf("Resultado:\n");
    for (i=0; i<N; i++)
        printf("%d ", v2[i]);
    printf("\n");
    #endif

    #ifdef FINAL
    printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",tiempo,v2[0],v2[N-1]);
    #endif
}
```

DESCOMPOSICIÓN DE DOMINIO:



CAPTURAS DE PANTALLA:

```
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer10$ gcc -O2 -fopenmp pmtv-OpenMP.c -o pmtv-OpenMP
elena@elena97om:~/Escritorio/6AÑO/AC/practicas/BP3/ejer10$ ./pmtv-OpenMP 4

Ejecutando de forma DINAMICA

Matriz inicial:
 1  9  17  25
 0  8  16  24
 0  0  15  23
 0  0  0  22

Vector inicial:  | 5 | 7 | 9 | 11 |
Tiempo matriz tamanio 4: 0.000023720
Vector resultado:  | 496 | 464 | 388 | 242 |
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

```
[b2estudiante27@atcgrid ejer10]$ sh pmtv-OpenMP_atcgrid.sh >> resultados.txt
[b2estudiante27@atcgrid ejer10]$
```

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash

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

#Instrucciones del script para ejecutar código:
export OMP_NUM_THREADS=12
export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
srun ./pmtv7 15360

export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
srun ./pmtv7 15360

export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
srun ./pmtv7 15360

export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
srun ./pmtv7 15360

export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
srun ./pmtv7 15360

export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
srun ./pmtv7 15360

export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
srun ./pmtv7 15360

export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
srun ./pmtv7 15360

export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
srun ./pmtv7 15360
```

Tabla 2 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N= (solo se ha paralelizado el producto, no la inicialización de los datos).

Chunk	Static	Dynamic	Guided
por defecto	0,029105295	0,031229568	0,031179004
1	0,031237901	0,025257783	0,028858418
64	0,029003302	0,031205580	0,031288572
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0,134498794	0,123585165	0,123420683
1	0,134599909	0,134399926	0,117883167
64	0,134519795	0,134376322	0,134479214