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): Jose Teodosio Lorente Vallecillos
Grupo de prácticas: A 3
Fecha de entrega:
Fecha evaluación en clase:

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

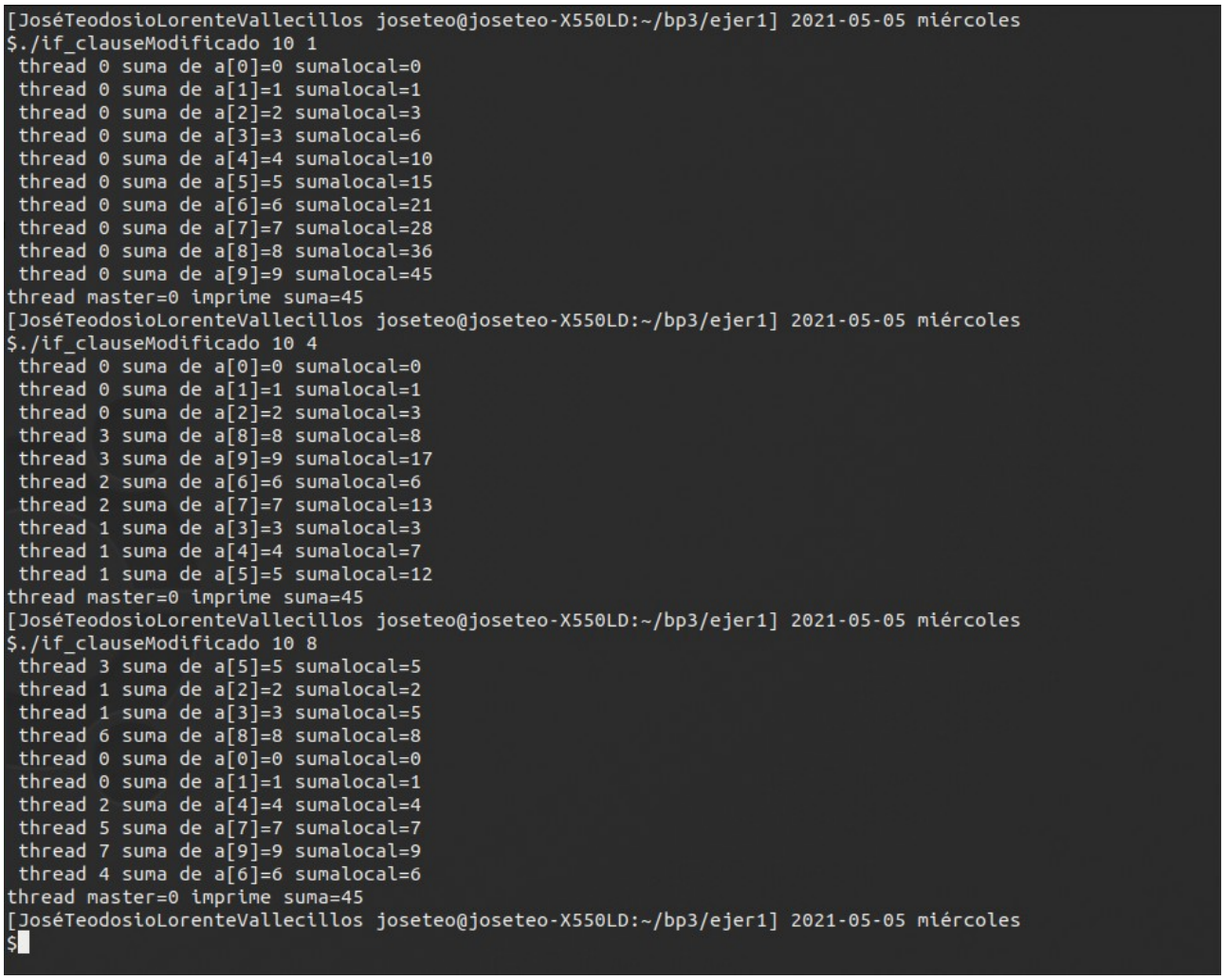
Ejercicios basados en los ejemplos del seminario práctico

1. 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,x;

    if (argc < 2)
    {
        fprintf(stderr,"[ERROR] Falta el numero de iteraciones\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 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:



RESPUESTA:
Mediante esta cláusula lo que se consigue es poder ejecutar el programa con las hebras que se definen según el parámetro de entrada que se le da. El primer parametro son las iteraciones y el segundo son las hebras que lo ejecutan. De este modo tal y como se muestra en la captura de la ejecucion del programa, se puede realizar la suma tanto con 1 hebra como con 8.

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="non-monotonic: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	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
2	**2**	**1**	**2**	**2**	**2**	**2**	**0**	**0**
3	**0**	**1**	**0**	**0**	**1**	**2**	**0**	**0**
4	**1**	**2**	**0**	**0**	**1**	**1**	**0**	**0**
5	**2**	**2**	**0**	**0**	**1**	**1**	**0**	**0**
6	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
7	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**
8	**2**	**1**	**0**	**0**	**0**	**1**	**1**	**1**
9	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**1**
10	**1**	**2**	**0**	**0**	**0**	**1**	**2**	**2**
11	**2**	**2**	**0**	**0**	**0**	**1**	**2**	**2**
12	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
13	**1**	**0**	**2**	**0**	**0**	**1**	**0**	**0**
14	**2**	**1**	**2**	**0**	**0**	**1**	**0**	**0**
15	**0**	**1**	**2**	**0**	**0**	**1**	**1**	**0**

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:
La cláusula schedule se encarga de distribuir y asignar el trabajo del bucle entre las distintas hebras. Mediante static dicha distribución se realiza en tiempo de compilación, separando las iteraciones en chunks, de manera que el numero de iteraciones de cada hebra esta ya definido y es equitativo. Mediante dynamic dicha distribución y asignación se realiza en tiempo de ejecución por lo que es desconocido que hebra realizara que iteraciones. Las hebras más rápidas ejecutaran más trabajos, aunque previamente deben ejecutar el chunk predefinido. Mediante guided la realización de la distribución y la asignación es como en dynamic. Comienza con un bloque completo el cual va decreciendo dependiendo del numero de iteraciones que resten, aunque nunca más pequeño que el chunk. Tal y como en dynamic las hebras más rápidas realizaran mas ejecuciones. Al usar monotonic si una hebra ejecutó una iteración i, entonces la hebra debe ejecutar iteraciones mayores que i posteriormente, y para nonmonotonic el orden de ejecución no está sujeta a la restricción monótona.

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.

El valor que OpenMP toma para chunk con static es el tamaño de la division por igual entre los threads de las iteraciones debido a que es el valor asignado por defecto al numero de threads, y el modifier por defecto para static es monotonic, y para los demás su valor por defecto es nonmonotonic. Luego el valor de chunk para dynamic por defecto es 1 y para guided al igual que para dynamic el valor por defecto de chunk es 1.

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, modif, a[n], suma=0;
    omp_sched_t tipo;

    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]);

    int dyn_var=omp_get_dynamic();
    int nthreads_var=omp_get_max_threads();
    int threads_limit_var=omp_get_thread_limit();
    omp_get_schedule(&tipo, & modif);

    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);
    }

    #pragma omp single
      printf("\nThread %d dyn-var %d \n", omp_get_thread_num(), dyn_var);
      printf("Thread %d nthreads-var %d \n", omp_get_thread_num(), nthreads_var);
      printf("Thread %d threads-limit-var %d \n", omp_get_thread_num(), threads_limit_var);
      printf("Thread %d run-sched-var %d \n", omp_get_thread_num(), modif);

    printf("Fuera de 'parallel for' suma=%d\n dyn-var=%d\n nthreads-var=%d\n threads-limit-var=%d\n run-sched-var %d\n",
           suma,dyn_var,nthreads_var, threads_limit_var, modif);
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$gcc -O2 -fopenmp -o scheduled-clauseModificado scheduled-clauseModificado.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$./scheduled-clauseModificado 10 4
 thread 2 suma a[4]=4 suma=4
 thread 2 suma a[5]=5 suma=9
 thread 2 suma a[6]=6 suma=15
 thread 2 suma a[7]=7 suma=22
 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 1 suma a[8]=8 suma=8
 thread 1 suma a[9]=9 suma=17

Thread 0 dyn-var 0
Thread 0 nthreads-var 4
Thread 0 threads-limit-var 2147483647
Thread 0 run-sched-var 0
Fuera de 'parallel for' suma=17
 dyn-var=0
 nthreads-var=4
 threads-limit-var=2147483647
 run-sched-var 0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$
```

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$export OMP_DYNAMIC=TRUE
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$export OMP_NUM_THREADS=8
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$./scheduled-clauseModificado 12 5
 thread 2 suma a[5]=5 suma=5
 thread 2 suma a[6]=6 suma=11
 thread 2 suma a[7]=7 suma=18
 thread 2 suma a[8]=8 suma=26
 thread 2 suma a[9]=9 suma=35
 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 0 suma a[4]=4 suma=10
 thread 1 suma a[10]=10 suma=10
 thread 1 suma a[11]=11 suma=21

Thread 0 dyn-var 1
Thread 0 nthreads-var 8
Thread 0 threads-limit-var 2147483647
Thread 0 run-sched-var 0
Fuera de 'parallel for' suma=21
 dyn-var=1
 nthreads-var=8
 threads-limit-var=2147483647
 run-sched-var 0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$
```

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$export OMP_DYNAMIC=TRUE
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$export OMP_NUM_THREADS=4
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$export OMP_SCHEDULE="static,5"
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$./scheduled-clauseModificado 12 5
 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 0 suma a[4]=4 suma=10
 thread 1 suma a[5]=5 suma=5
 thread 1 suma a[6]=6 suma=11
 thread 1 suma a[7]=7 suma=18
 thread 1 suma a[8]=8 suma=26
 thread 1 suma a[9]=9 suma=35
 thread 2 suma a[10]=10 suma=10
 thread 2 suma a[11]=11 suma=21

Thread 0 dyn-var 1
Thread 0 nthreads-var 4
Thread 0 threads-limit-var 2147483647
Thread 0 run-sched-var 5
Fuera de 'parallel for' suma=21
 dyn-var=1
 nthreads-var=4
 threads-limit-var=2147483647
 run-sched-var 5
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer4] 2021-05-11 martes
$
```

RESPUESTA:

En todas las ejecuciones imprime lo mismo dentro y fuera del parallel, los cuales se corresponden a los valores establecidos previamente en las variables de entorno.

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, modif, a[n], suma=0;
    omp_sched_t tipo;

    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]);

    omp_get_schedule(&tipo, & modif);

    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++)
    {
        if(i==0)
            printf("Dentro del parallel; num_threads=%d, num_procs=%d, omp_in_parallel=%d\n",
                    omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
               omp_get_thread_num(),i,a[i],suma);
    }

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

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer5] 2021-05-11 martes
$gcc -O2 -fopenmp -o scheduled-clauseModificado4 scheduled-clauseModificado4.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer5] 2021-05-11 martes
$./scheduled-clauseModificado4 4 4
Dentro del parallel; num_threads=3, num_procs=4, 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

Fuera de 'parallel for' suma=6
omp_get_num_threads()=1
omp_get_num_procs()=4
omp_in_parallel()=0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer5] 2021-05-11 martes
$
```

RESPUESTA:
Se obtienen diferentes resultados de el num_threads y el omp_parallel.

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, valor_chunk, a[n], suma=0;
    omp_sched_t tipo;

    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(i==(n-2)){
            omp_set_dynamic(0);
            omp_set_num_threads(4);
            omp_set_schedule(1,2);
        }
        if(i==0 || i==(n-2)){
            if(i==0)
                printf("\nAntes del cambio=\n");
            else
                printf("\nDespues del cambio=\n");

            omp_get_schedule(&tipo, &valor_chunk);
            printf("dyn-var=%d, nthreads=%d, run-sched-var=%d \n", omp_get_dynamic(), omp_get_max_threads(), tipo);
        }
    }
    printf("\nFuera de 'parallel for' suma=%d\n",suma);
    printf("dyn-var=%d, nthreads-var=%d, run-sched-var=%d\n",omp_get_dynamic(),omp_get_max_threads(), tipo);
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer6] 2021-05-11 martes
$gcc -O2 -fopenmp -o scheduled-clauseModificado5 scheduled-clauseModificado5.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer6] 2021-05-11 martes
$./scheduled-clauseModificado5 10 10
 thread 0 suma a[0]=0 suma=0

Antes del cambio=
dyn-var=1, nthreads=4, run-sched-var=-2147483647
 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 0 suma a[4]=4 suma=10
 thread 0 suma a[5]=5 suma=15
 thread 0 suma a[6]=6 suma=21
 thread 0 suma a[7]=7 suma=28
 thread 0 suma a[8]=8 suma=36

Despues del cambio=
dyn-var=0, nthreads=4, run-sched-var=1
 thread 0 suma a[9]=9 suma=45

Fuera de 'parallel for' suma=45
dyn-var=1, nthreads-var=4, run-sched-var=1
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp3/ejer6] 2021-05-11 martes
$
```

RESPUESTA:
Lo que tenemos que hacer es que estos valores vienen dados por variables de entorno y son los valores antes de la modificación. Después de modificarlos, la función proporcionará estos valores en tiempo de ejecución. Eso si, las variables dyn-var y nthreads fuera de la operación en paralelo tienen valores proporcionados por el entorno, aunque run-sched conserva la modificación.

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 <time.h>

int main(int argc, char **argv){
  struct timespec cgt1,cgt2;
  double ncgt;

  if(argc<2){
    printf("Faltan n compoentes del vector\n");
    exit(-1);
  }

  unsigned int N=atoi(argv[1]);   //El maximo de N es 2^32 -1 =4294967295 sizeof(unsigned int)=4 B
  int i,j;
  int *v,*mv,**m;

  v=(int*) malloc(N*sizeof(int));
  mv=(int*) malloc(N*sizeof(int));
  m=(int**) malloc(N*sizeof(int *));

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

  if(v==NULL || mv==NULL || m==NULL){
    printf("Fallo en la reserva de espacio para la amtriz y el vector\n");
    exit(-2);
  }

  for(i=0;i<N;i++)
    if(m[i]==NULL){
      printf("Fallo en la reserva de espacio para la amtriz y el vector\n");
      exit(-3);
    }

  for(i=0;i<N;i++){
    for(j=i;j<N;j++)
      m[i][j]=9;
    v[i]=2;
    mv[i]=0;
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  for(i=0;i<N;i++)
    for(j=0;j<N;j++)
      mv[i]+=m[i][j]*v[i];

  clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

  printf("Vector= \n");
  for(i=0;i<N;i++)
    printf("%d ", v[i]);
  printf("\n");

  printf("Resultado= \n");
  for(i=0;i<N;i++)
    printf("%d ", mv[i]);
  printf("\n");

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

  printf("Tamaño de la matriz y vector= %d\t / Tiempo(seg.) %11.9f\n",N,ncgt);

  //liberamos los valores de la matriz y liberamos los espacios de la matriz y los vectores
  for(i=0;i<N;i++)
    free(m[i]);

  free(m);
  free(v);
  free(mv);

  return 0;
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp3/ejer7] 2021-05-12 miércoles
$srun -p ac ./pmtv-secuencial 10
Matriz=
9 9 9 9 9 9 9 9 9 9
0 9 9 9 9 9 9 9 9 9
0 0 9 9 9 9 9 9 9 9
0 0 0 9 9 9 9 9 9 9
0 0 0 0 9 9 9 9 9 9
0 0 0 0 0 9 9 9 9 9
0 0 0 0 0 0 9 9 9 9
0 0 0 0 0 0 0 9 9 9
0 0 0 0 0 0 0 0 9 9
0 0 0 0 0 0 0 0 0 9
Vector=
2 2 2 2 2 2 2 2 2 2
Resultado=
180 162 144 126 108 90 72 54 36 18
Tamaño de la matriz y vector= 10         / Tiempo(seg.) 0.000000377
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp3/ejer7] 2021-05-12 miércoles
$
```

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

int main(int argc, char **argv){
  double tiempo_inicial,tiempo_final, tiempo_total;

  if(argc<2){
    printf("Faltan n compoentes del vector\n");
    exit(-1);
  }

  unsigned int N=atoi(argv[1]);   //El maximo de N es 2^32 -1 =4294967295 sizeof(unsigned int)=4 B
  int i,j,suma;
  int *v,*mv,**m;
  struct drand48_data randBuffer;

  v=(int*) malloc(N*sizeof(int));
  mv=(int*) malloc(N*sizeof(int));
  m=(int**) malloc(N*sizeof(int *));

  if(v==NULL || mv==NULL || m==NULL){
    printf("Fallo en la reserva de espacio para la amtriz y el vector\n");
    exit(-2);
  }

  for(i=0;i<N;i++){
    m[i]=(int *) malloc(N*sizeof(int));
    if(m[i]==NULL){
      printf("Fallo en la reserva de espacio para la amtriz y el vector\n");
      exit(-3);
    }
  }

  for(i=0;i<N;i++)
    for(j=0;j<N;j++)
      m[i][j]=0;

  for(i=0;i<N;i++){
    for(j=i;j<N;j++)
      m[i][j]=9;
    v[i]=2;
    mv[i]=0;
  }
```

```
#pragma omp parallel private(j)
{
  tiempo_inicial= omp_get_wtime();

  #pragma omp for firstprivate(suma) schedule(runtime)
  for(i=0;i<N;i++){
    drand48_r(&randBuffer,&v[i]);
    suma=0;
    for(j=0;j<N;j++){
      suma+=m[i][j]*v[i];
    }
    #pragma single
      mv[i]=suma;
  }

  tiempo_final=omp_get_wtime();
}

tiempo_total= tiempo_final-tiempo_inicial;

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

printf("Vector= \n");
for(i=0;i<N;i++)
  printf("%d ", v[i]);
printf("\n");

printf("Resultado= \n");
for(i=0;i<N;i++)
  printf("%d ", mv[i]);
printf("\n");


printf("Tamaño de la matriz y vector= %d\t / Tiempo(seg.) %11.9f\n",N,tiempo_total);

//liberamos los valores de la matriz y liberamos los espacios de la matriz y los vectores
for(i=0;i<N;i++)
  free(m[i]);
free(m);
free(v);
free(mv);
return 0;
```

DESCOMPOSICIÓN DE DOMINIO:

r = m * v; $c_i = \Sigma_{k=0}^{n-1} m_{i,k} * v_k$, **i = 0,1,2,… n-1**

	0	1	2	3	...	n-1	n		v_0		r_0
0	m_{00}	m_{01}	m_{02}	m_{03}	...	$m_{0\,n-1}$	m_{0n}		v_1		r_1
1	0	m_{11}	m_{12}	m_{13}	...	$m_{1\,n-1}$	m_{1n}		v_2		r_2
2	0	0	m_{22}	m_{23}	...	$m_{2\,n-1}$	m_{2n}	*	v_3	=	r_3
3	0	0	0	m_{33}	...	$m_{3\,n-1}$	m_{3n}		v_4		r_4
...	...	...	...	...	...	...	...		...		...
n-1	0	0	0	0	...	$m_{n-1\,n-1}$	m_{n-1n}		v_{n-1}		r_{n-1}

n	**0**	**0**	**0**	**0**	**...**	**$m_{n\,n-1}$**	**$m_{n\,n}$**		**v_n**		**r_n**

En el caso en el que hiciéramos una asignación estática (RR) y fijáramos el chunk a 2, para N hebras. Cada hebra multiplicaría al vector V dos filas de la matriz de forma consecutiva. En la grafica se distingue la descomposición de la matriz por los colores.

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp3/ejer8] 2021-05-12 miércoles
$srun -p ac ./pmtv-OpenMP 5
Matriz=
9 9 9 9 9
0 9 9 9 9
0 0 9 9 9
0 0 0 9 9
0 0 0 0 9
Vector=
0 1025900544 2041479168 -1666024960 -496598528
Resultado=
0 0 -714637312 76321792 -174419456
Tamaño de la matriz y vector= 5  / Tiempo(seg.) 0.000014901
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp3/ejer8] 2021-05-12 miércoles
$
```

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: Depende, según la fila de la matriz asignada, puesto que la matriz es triangular, de tal forma que la fila 1 hace N operaciones, la fila 2 hace N-1 operaciones, y así sucesivamente hasta la ultima fila que hace 1 operación.

(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: Eso depende de las hebras disponibles, es decir, cada hebra hará x operaciones distintas, dependiendo de la disponibilidad estarán libres u ocupadas.

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

RESPUESTA: La que tiene mejores prestaciones es la planificación guided, puesto que reduce bastante el tiempo de ejecución.

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

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

int main(int argc, char **argv){
  double tiempo_inicial,tiempo_final, tiempo_total;

  if(argc<2){
    printf("Faltan n compoentes del vector\n");
    exit(-1);
  }

  unsigned int N=atoi(argv[1]);   //El maximo de N es 2^32 -1 =4294967295 sizeof(unsigned int)=4 B
  int i,j,suma;
  int *v,*mv,**m;

  v=(int*) malloc(N*sizeof(int));
  mv=(int*) malloc(N*sizeof(int));
  m=(int**) malloc(N*sizeof(int *));

  if(v==NULL || mv==NULL || m==NULL){
    printf("Fallo en la reserva de espacio para la amtriz y el vector\n");
    exit(-2);
  }

  for(i=0;i<N;i++){
    m[i]=(int *) malloc(N*sizeof(int));
    if(m[i]==NULL){
      printf("Fallo en la reserva de espacio para la amtriz y el vector\n");
      exit(-3);
    }
  }

  for(i=0;i<N;i++){
    for(j=0;j<N;j++){
      if(i<=j)
        m[i][j]=9;
      else
        m[i][j]=0;
    }
    v[i]=2;
    mv[i]=0;
  }
```

```
#pragma omp parallel private(j)
{
  tiempo_inicial= omp_get_wtime();

  #pragma omp for firstprivate(suma) schedule(runtime)
  for(i=0;i<N;i++){
    suma=0;
    for(j=0;j<N;j++){
      suma+=m[i][j]*v[i];
    }
    #pragma single
      mv[i]=suma;
  }

  tiempo_final=omp_get_wtime();
}

tiempo_total= tiempo_final-tiempo_inicial;
printf("Tamaño de la matriz y vector= %d\t / Tiempo(seg.) %11.9f\n",N,tiempo_total);

printf("matriz_x_vector[%d]= %d\n",0,mv[0]);
printf("matriz_x_vector[%d]= %d\n",N-1,mv[N-1]);

//liberamos los valores de la matriz y liberamos los espacios de la matriz y los vectores
for(i=0;i<N;i++)
  free(m[i]);
free(m);
free(v);
free(mv);
return 0;
}
```

DESCOMPOSICIÓN DE DOMINIO:

r = m * v; $c_i = \Sigma_{k=0}^{n-1} m_{i,k} * v_k$, **i = 0,1,2,… n-1**

	0	1	2	3	...	n-1	n		v_0		r_0
0	m_{00}	m_{01}	m_{02}	m_{03}	...	$m_{0\,n-1}$	m_{0n}		v_1		r_1
1	0	m_{11}	m_{12}	m_{13}	...	$m_{1\,n-1}$	m_{1n}		v_2		r_2
2	0	0	m_{22}	m_{23}	...	$m_{2\,n-1}$	m_{2n}	*	v_3	=	r_3
3	0	0	0	m_{33}	...	$m_{3\,n-1}$	m_{3n}		v_4		r_4
...	...	...	...	...	...	...	...		...		...
n-1	0	0	0	0	...	$m_{n-1\,n-1}$	m_{n-1n}		v_{n-1}		r_{n-1}
n	0	0	0	0	...	$m_{n\,n-1}$	$m_{n\,n}$		v_n		r_n

En el caso en el que hiciéramos una asignación estática (RR) y fijáramos el chunk a 2, para N hebras. Cada hebra multiplicaría al vector V dos filas de la matriz de forma consecutiva. En la grafica se distingue la descomposición de la matriz por los colores.

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp3/ejer10] 2021-05-14 viernes
$sbatch -p ac ./pmvt-OpenMP-atcgrid.sh
Submitted batch job 105681
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp3/ejer10] 2021-05-14 viernes
$cat slurm-105681.out
static y chunk por defecto ----------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000403143
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18


static y chunk 1 ----------------------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000501219
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18


static y chunk 64 ---------------------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000616442
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18


dynamic y chunk por defecto ----------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000036642
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18


dynamic y chunk 1 ---------------------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000035323
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18


dynamic y chunk 64 --------------------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000036232
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18


guided y chunk por defecto ----------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000037119
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18


guided y chunk 1 ----------------------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000031583
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18
```

```
guided y chunk 64 ---------------------------------------
Tamaño de la matriz y vector= 2200       / Tiempo(seg.) 0.000037152
matriz_x_vector[0]= 39600
matriz_x_vector[2199]= 18


[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp3/ejer10] 2021-05-14 viernes
$
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash

export OMP_NUM_THREADS=12

declare -a planificacion=("static" "dynamic" "guided")

for i in "${planificacion[@]}"
do
  for ((j=0;j<3;++j))
  do
    if [ $j -eq 0 ]
    then
      echo "$i y chunk por defecto ---------------------------- "
      export OMP_SCHEDULE="$i"
    elif [ $j -eq 1 ]
    then
      echo "$i y chunk 1 ------------------------------------- "
      export OMP_SCHEDULE="$i,1"
    else
      echo "$i y chunk 64 ------------------------------------ "
      export OMP_SCHEDULE="$i,64"
    fi
    srun ./pmtv-OpenMP 2200
    echo -e "\n"
  done
done
```

Tabla 2 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N=2200 y 12 threads (solo se ha paralelizado el producto, no la inicialización de los datos).

Chunk	Static	Dynamic	Guided
por defecto	0.000403143	0.000036642	0.000037119
1	0.000501219	0.000035323	0.000031583
64	0.000616442	0.000036232	0.000037152
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.000463527	0.000031915	0.000031132
1	0.000634462	0.000030726	0.000034466
64	0.000527687	0.000030760	0.000032298