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): Daniel Alconchel Vázquez
Grupo de prácticas: 1
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
int main(int argc, char **argv)
{
    int i, n=20, tid;
    int a[n], suma=0, sumalocal;
    int x;

    if(argc < 2){
        fprintf(stderr,"[ERROR] Falta el número de iteraciones\n");
        exit(EXIT_FAILURE);
    }
    if(argc < 3){
        fprintf(stderr,"[ERROR] Falta el numero de threads\n");
        exit(EXIT_FAILURE);
    }

    n = atoi(argv[1]);
    if (n>20)n=20;
    //MODIFICACION: VARIABLE (int) x
    x = atoi(argv[2]);
    if(x>8) x=8;

   for (i=0; i<n; i++)
     a[i]=i;

    //MODIFICACION: num_threads(x)
    #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);
    }
}
```

NORMAL if-clauseModificado.c c utf-8[uni

CAPTURAS DE PANTALLA:

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer1] 2021-05-11 martes
$ gcc -O2 -fopenmp if-clause.c -o clause
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer1] 2021-05-11 martes
$ gcc -O2 -fopenmp if-clauseModificado.c -o clauseModificado
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer1] 2021-05-11 martes
$ ./clauseModificado 6 4
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[4]=4 sumalocal=4
 thread 3 suma de a[5]=5 sumalocal=5
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
thread master=0 imprime suma=15
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer1] 2021-05-11 martes
$ ./clauseModificado 12 8
 thread 6 suma de a[10]=10 sumalocal=10
 thread 7 suma de a[11]=11 sumalocal=11
 thread 4 suma de a[8]=8 sumalocal=8
 thread 3 suma de a[6]=6 sumalocal=6
 thread 3 suma de a[7]=7 sumalocal=13
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 5 suma de a[9]=9 sumalocal=9
 thread 2 suma de a[4]=4 sumalocal=4
 thread 2 suma de a[5]=5 sumalocal=9
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
thread master=0 imprime suma=66
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer1] 2021-05-11 martes
$
```

RESPUESTA: Como se observa, el segundo argumento pasado al programa hace referencia al número de hebras que se utilizarán (como máximo 8). Con el primer argumento, decimos el número de iteraciones a realizar, que requieren que sean 5 como mínimo para realizar la ejecución en paralelo, ya que, en caso contrario, no sería eficiente la creación, eliminación y sincronización de las hebras.

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

14	2	1	1	2	0	2	2	2
15	2	1	1	2	2	2	2	2

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer3] 2021-05-13 jueves
$ export OMP_NUM_THREADS=3
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer3] 2021-05-13 jueves
$ export OMP_SCHEDULE="monotonic:static,1"
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer3] 2021-05-13 jueves
$ ./sheduler 16
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[4]=4 suma=5
 thread 1 suma a[7]=7 suma=12
 thread 1 suma a[10]=10 suma=22
 thread 1 suma a[13]=13 suma=35
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[3]=3 suma=3
 thread 0 suma a[6]=6 suma=9
 thread 0 suma a[9]=9 suma=18
 thread 0 suma a[12]=12 suma=30
 thread 0 suma a[15]=15 suma=45
 thread 2 suma a[2]=2 suma=2
 thread 2 suma a[5]=5 suma=7
 thread 2 suma a[8]=8 suma=15
 thread 2 suma a[11]=11 suma=26
 thread 2 suma a[14]=14 suma=40
Fuera de 'parallel for' suma=45
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer3] 2021-05-13 jueves
```

`Las demás igual, pero cambiando el export.`

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:
- STATIC: Las hebras se distribuyen en tiempo de compilación, las iteraciones se dividen en unidades de chunk iteraciones y las unidades se asignan en round-robin. El número de iteraciones que realiza cada hebra está predefinido y será equitativo, siempre y cuando el total sea múltiplo de las hebras.

-DYNAMIC: Las hebras se distribuyen en tiempo de ejecución, por lo que no sabemos qué iteraciones realizará cada hebra. La unidad de distribución tiene chunk iteraciones : num unidades = O(n/chunk). Los threads más rápidos ejecutan más unidades, pero como mínimo ejecutarán las chunk iteraciones.

-GUIDED: Las hebras se distribuyen también en tiempo de ejecución. Empezamos con bloque largo, cuyo tamaño va menguando: num_iteraciones_restantes / num_threads, y nunca es más pequeño que chunk. Las hebras más ociosas realizan más ejecuciones.

En cuanto a la diferencia de usar monotonic y nomonotonic, no existe diferencia a la hora de ejecución, solo una diferencia teórica.

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.

Usando la documentación oficial de OpenMP, podemos afirmar que si es static con modifier se establece como monotic. Cualquier otro caso, se establece a nonmonotic.
Las páginas web usadas son:
- https://www.openmp.org/spec-html/5.0/openmpse49.html
- https://www.openmp.org/wp-content/uploads/SC17-Kale-LoopSchedforOMP_BoothTalk.pdf

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`

```
#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 == 0){
      //USO DE OMP_GET_SCHEDULE
      omp_get_schedule(&kind,&chunk_value);

      //MOSTRANDO DENTRO DE PARALLEL LAS VARIABLES...
      printf("DENTRO DE OMP PARALLEL FOR: \n 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(), kind, chunk_value);
    }
}

printf("Fuera de 'parallel for' suma=%d\n",suma);
//MOSTRANDO FUERA DE PARALLEL LAS VARIABLES...
printf("FUERA DE OMP PARALLEL FOR: \n 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(), kind, chunk_value);
```

CAPTURAS DE PANTALLA:

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ gcc -O2 -fopenmp scheduled-clauseModificado.c -o schedule
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ export OMP_NUM_THREADS=2
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ export OMP_DYNAMIC=FALSE
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ export OMP_SCHEDULE="static,4"
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ ./schedule 8 4
 thread 0 suma a[0]=0 suma=0
DENTRO DE OMP PARALLEL FOR:
 dyn-var: 0 | nthreads-var: 2          thread-limit-var: 2147483647, run-sched-var: -2147483647, chunk: 4
 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 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=22
FUERA DE OMP PARALLEL FOR:
 dyn-var: 0 | nthreads-var: 2     thread-limit-var: 2147483647, run-sched-var: -2147483647, chunk: 4
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$
```

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ export OMP_NUM_THREADS=4
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ export OMP_DYNAMIC=TRUE
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ export OMP_SCHEDULE="dynamic,4"
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
$ ./schedule 8 4
 thread 2 suma a[0]=0 suma=0
DENTRO DE OMP PARALLEL FOR:
 dyn-var: 1 | nthreads-var: 4          thread-limit-var: 2147483647, run-sched-var: 2, chunk: 4
 thread 2 suma a[1]=1 suma=1
 thread 2 suma a[2]=2 suma=3
 thread 2 suma a[3]=3 suma=6
 thread 0 suma a[4]=4 suma=4
 thread 0 suma a[5]=5 suma=9
 thread 0 suma a[6]=6 suma=15
 thread 0 suma a[7]=7 suma=22
Fuera de 'parallel for' suma=22
FUERA DE OMP PARALLEL FOR:
 dyn-var: 1 | nthreads-var: 4     thread-limit-var: 2147483647, run-sched-var: 2, chunk: 4
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer4] 2021-05-13 jueves
```

RESPUESTA: Con estas dos ejecuciones vemos que fuera de la región parallel y dentro de la región parallel se muestran exactamente los mismos parámetros, que corresponden con los que hemos definido 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`

```
omp_sched_t kind; int chunk_value;

#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 == 0){
     //USO DE OMP_GET_SCHEDULE
     omp_get_schedule(&kind, &chunk_value);

     //MOSTRANDO DENTRO DE PARALLEL LAS VARIABLES...
     printf("DENTRO DE OMP PARALLEL FOR: \n num_threads: %d\n", omp_get_num_threads());
     printf("DENTRO DE OMP PARALLEL FOR: \n num_procs: %d\n", omp_get_num_procs());
     printf("DENTRO DE OMP PARALLEL FOR: \n in_parallel: %d\n", omp_in_parallel());
     printf("DENTRO DE OMP PARALLEL FOR: \n 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(), kind, chunk_value);
    }
}
```

CAPTURAS DE PANTALLA:

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ export OMP_NUM_THREADS=3
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ export OMP_DYNAMIC=FALSE
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ export OMP_SCHEDULE="dynamic,2"
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ ./modificado 6 3
 thread 1 suma a[0]=0 suma=0
DENTRO DE OMP PARALLEL FOR:
 num_threads: 3
DENTRO DE OMP PARALLEL FOR:
 num_procs: 12
DENTRO DE OMP PARALLEL FOR:
 in_parallel: 1
DENTRO DE OMP PARALLEL FOR:
 dyn-var: 0 | nthreads-var: 3          thread-limit-var: 2147483647, run-sched-var: 2, chunk: 2
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=3
 thread 2 suma a[3]=3 suma=3
 thread 2 suma a[4]=4 suma=7
 thread 2 suma a[5]=5 suma=12
Fuera de 'parallel for' suma=12
FUERA DE OMP PARALLEL FOR:
 num_threads: 1
FUERA DE OMP PARALLEL FOR:
 num_procs: 12
FUERA DE OMP PARALLEL FOR:
 in_parallel: 0
FUERA DE OMP PARALLEL FOR:
 dyn-var: 0 | nthreads-var: 3     thread-limit-var: 2147483647, run-sched-var: 2, chunk: 2
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
```

RESPUESTA: La diferencia que se observa con respecto lo que se imprime dentro de la región parallel y fuera de la región parallel son las variables num_threads y in_parallel. Es algo evidente, en el primer caso, dentro de la región parallel estamos utilizando los 3 threads que hemos definido en la variable de entorno OMP_NUM_THREADS,y fuera de la región parallel nos muestra 1 hebra porque el código ya no se está ejecutando en paralelo. En el caso de in_parallel, no es necesario una explicación exhaustiva. Muestra 1 dentro de parallel (true) y 0 fuera (false).

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`

```
omp_sched_t kind; int chunk_value;

#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);

    //MODIFICACION EJERCICIO 5. MODIFICACION DE VARIABLES
    if(i == 3){
      //CAMBIAMOS DYN-VAR
      omp_set_dynamic(0);
      //CAMBIAMOS NTHREADS-VAR
      omp_set_num_threads(4);
      //CAMBIAMOS RUN-SCHED-VAR
      omp_set_schedule(1,2);
    }
    if(i == 0 || i == 3){
      if(i == 0){
        printf("MOSTRANDO VARIABLES ANTES DE MODIFICACION...\n");
      }
      else{
        printf("MOSTRANDO VARIABLES DESPUES DE MODIFICACION...\n");
      }
      //MOSTRANDO DENTRO DE PARALLEL LAS VARIABLES...
      omp_get_schedule(&kind, &chunk_value);
      printf("DENTRO DE OMP PARALLEL FOR: \n 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(), kind, chunk_value);
    }
}
```

CAPTURAS DE PANTALLA:

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ gcc -O2 -fopenmp scheduled-clauseModificado5.c -o scheduled
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ export OMP_DYNAMIC=TRUE
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ export OMP_NUM_THREADS=2
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ export OMP_SCHEDULE="dynamic,3"
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$ ./scheduled 8 4
 thread 0 suma a[0]=0 suma=0
MOSTRANDO VARIABLES ANTES DE MODIFICACION...
DENTRO DE OMP PARALLEL FOR:
 dyn-var: 1 | nthreads-var: 2          thread-limit-var: 2147483647, run-sched-var: 2, chunk: 3
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 suma a[3]=3 suma=6
MOSTRANDO VARIABLES DESPUES DE MODIFICACION...
DENTRO DE OMP PARALLEL FOR:
 dyn-var: 0 | nthreads-var: 4          thread-limit-var: 2147483647, run-sched-var: 1, chunk: 2
 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
Fuera de 'parallel for' suma=28
FUERA DE OMP PARALLEL FOR:
 dyn-var: 1 | nthreads-var: 2     thread-limit-var: 2147483647, run-sched-var: 1, chunk: 4
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer5] 2021-05-13 jueves
$
```

RESPUESTA: En la iteración 0 del bucle mostramos los parámetros antes del cambio y en la iteración 3 los cambios que se producen. Se modifican dyn_var que pasa a tener valor 0; nthreads_var que pasa a tener valor 4; run-sched-var con valor 1, y chunk con valor 2.

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
//FUNCION PRINCIPAL
int main(int argc, char ** argv){
    //VARIABLES PARA EL CALCULO DEL TIEMPO
    struct timespec cgt1,cgt2;
    double ncgt;

    //PASO 0: COMPROBACION DE ARGUMENTOS
    if(argc < 2){
        printf("ERROR [0] AT %s. NUMERO DE ARGUMENTOS INSUFICIENTE.\n",argv[0]);
        printf("USO: %s [TAM]", argv[0]);
        exit(EXIT_FAILURE);
    }

    //PASO 1: OBTENCION DE DIMENSION
    unsigned int tam = atoi(argv[1]); //OBTENCION DEL TAM DE VECTOR Y MATRIZ
    printf("DIMENSION DE VECTOR Y MATRIZ: %u (%lu B)\n",tam,sizeof(unsigned int));

    //PASO 2: RESERVA DE MEMORIA (DINAMICA)
    #ifdef VAR_DYNAMIC
        //DEFINICION DE VARIABLES
        double ** m;
        double * v;
        double * mv;

        //RESERVA DE MEMORIA DINAMICA
        v = (double *) malloc(tam * sizeof(double)); //RESERVA DE MEMORIA PARA VECTOR
        mv = (double *) malloc(tam * sizeof(double)); //RESERVA DE MEMORIA PARA MULTIPLICACION DE MATRIZ X VECTOR
        m = (double **) malloc(tam * sizeof(double *)); //RESERVA DE MEMORIA PARA MATRIZ

        //COMPROBAMOS ERRORES DE ASIGNACION DE MEMORIA
        if(v == NULL || mv == NULL || m == NULL){
            printf("ERROR [1] AL RESERVAR MEMORIA PARA MATRIZ Y/O VECTOR.\n");
            exit(EXIT_FAILURE);
        }

        //CONTINUAMOS RESERVANDO MEMORIA PARA MATRIZ
        for(int i=0; i<tam;i++){
          m[i] = (double *) malloc(tam * sizeof(double));
          //COMPROBAMOS ERRORES
          if(m[i] == NULL){
              printf("ERROR [2] AL RESERVAR MEMORIA PARA MATRIZ.\n");
              exit(EXIT_FAILURE);
          }
        }
    #endif
```

CAPTURAS DE PANTALLA:

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp3/ejer7] 2021-05-13 jueves
$ gcc -O2 -fopenmp pmtv-secuencial.c -o pmt
```

```
sftp> put pmt
Uploading pmt to /home/e1estudiante1/pmt
pmt                                           100%   17KB  62.1KB/s   00:00
sftp>
```

```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-14 viernes
$ srun -p ac -n1 --cpus-per-task=1 --hint=nomultithread pmt 8
DIMENSION DE VECTOR Y MATRIZ: 8 (4 B)
MATRIZ M:
0.800000000     0.700000000     0.600000000     0.500000000     0.400000000     0.300000000     0.200000000     0.100000000
0.000000000     0.800000000     0.700000000     0.600000000     0.500000000     0.400000000     0.300000000     0.200000000
0.000000000     0.000000000     0.800000000     0.700000000     0.600000000     0.500000000     0.400000000     0.300000000
0.000000000     0.000000000     0.000000000     0.800000000     0.700000000     0.600000000     0.500000000     0.400000000
0.000000000     0.000000000     0.000000000     0.000000000     0.800000000     0.700000000     0.600000000     0.500000000
0.000000000     0.000000000     0.000000000     0.000000000     0.000000000     0.800000000     0.700000000     0.600000000
0.000000000     0.000000000     0.000000000     0.000000000     0.000000000     0.000000000     0.800000000     0.700000000
0.000000000     0.000000000     0.000000000     0.000000000     0.000000000     0.000000000     0.000000000     0.800000000

VECTOR V: [0.800000000  0.900000000     1.000000000     1.100000000     1.200000000     1.300000000     1.400000000     1.500000000     ]
RESULTADO DE MULTIPLICACION: [3.720000000       3.920000000     3.950000000     3.800000000     3.460000000     2.920000000     2.170000000     1.200000000     ]
TIEMPO DE EJECUCION: 0.000000328 || DIMENSION: 8
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-14 viernes
$
```

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`

```
//VARIABLES PARA OBTENER CHUNK & KIND VALUES
omp_sched_t kind; int chunk_value;

//VARIABLES PARA EL CALCULO DEL TIEMPO
double ncgt, t_inicial, t_final;

//PASO 4: CALCULO DEL PRODUCTO
double suma;

#pragma omp parallel
{
  #pragma omp single
    t_inicial = omp_get_wtime();

  #pragma omp for firstprivate(suma) schedule(runtime)
  for(int i = 0; i<tam ; i++){
      #ifdef SEE_CHUNK_DEFAULT
        if(i==0){
            omp_get_schedule(&kind,&chunk_value);
            printf("KIND: %d | CHUNK: %d \n",kind,chunk_value);
        }
      #endif

      suma = 0;
      for(int j=i; j<tam; j++){
          suma += m[i][j] * v[j];
      }
      mv[i] = suma;
  }

  #pragma omp single
    t_final = omp_get_wtime();
}
```

DESCOMPOSICIÓN DE DOMINIO:

resultado[i] = v[0]*M[i][0] + v[1]*M[i][1] + ··· + v[i]*M[i][i] Por ser triangular inferior

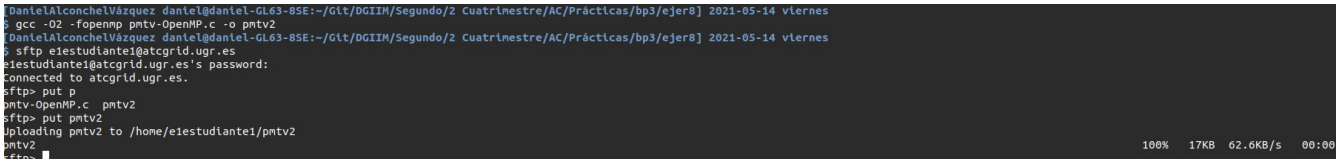
$T_1, \ldots, T_n$

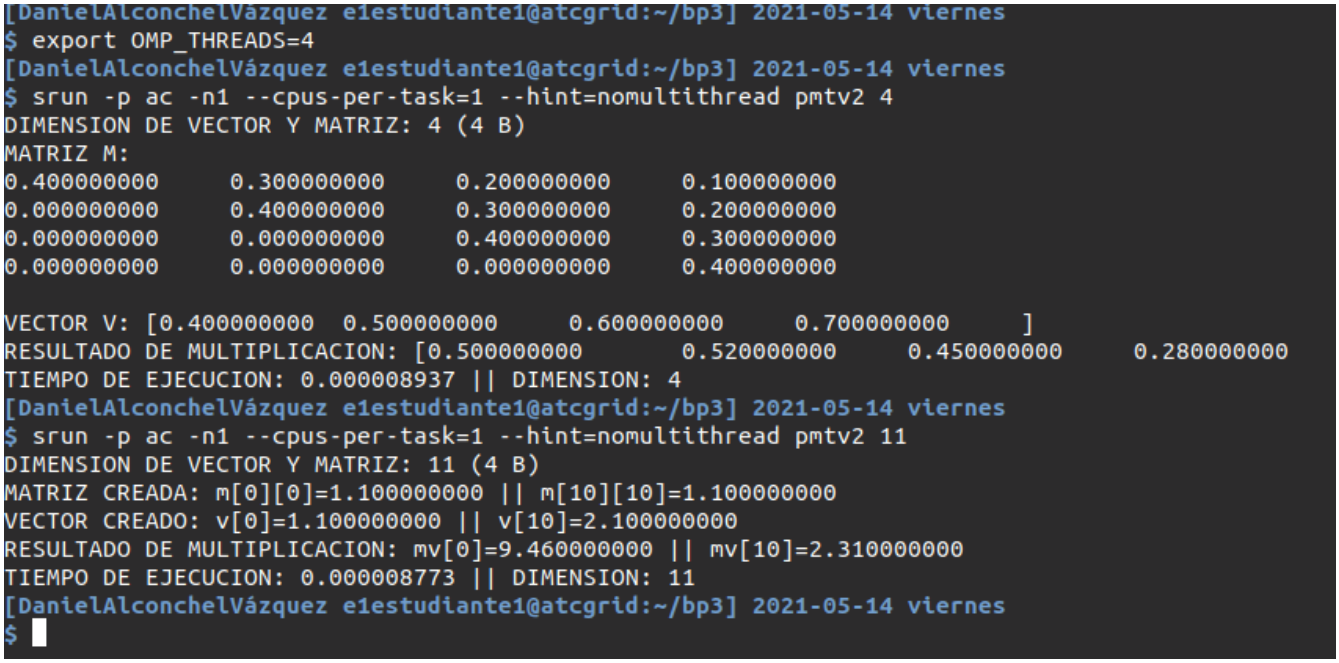
$P_1 \leftrightarrow P_2 \leftrightarrow \cdots \leftrightarrow P_n$

cada tarea es un cálculo de v[i]
cada hebra calcula una entrada
se paralelizan de esta forma todas las filas.

Aquí mostramos un ejemplo de descomposición para una planificación static con valor de chunk 2, para cualquier número de threads. Cada thread se ocuparía de 2 filas consecutivas de la matriz y la multiplicaría por las respectivas componentes del vector.

CAPTURAS DE PANTALLA:





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: Al ser una matriz triangular, el número de iteraciones depende de la fila. La primera realizará n iteraciones e irá disminuyendo de uno en uno hasta llegar a la última fila, donde hará únicamente una iteración.

(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: Usando el ejercicio 2 podremos ver que las operaciones dependerán del número de threads y cuantos haya disponibles. Cada hebra hará un número de operaciones distintas según haya hebras disponibles o no.

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

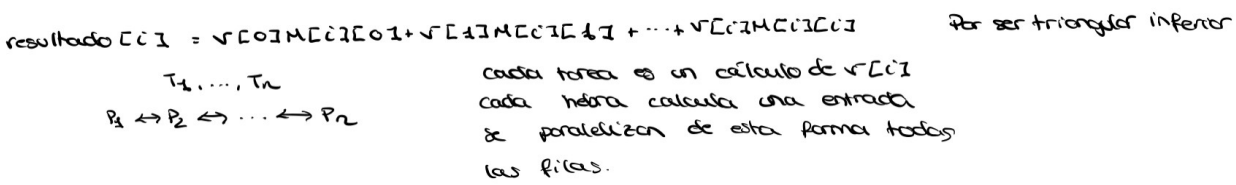
RESPUESTA: Será static, ya que evitaremos la sobrecarga en tiempo de ejecución de dynamic y guided.

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`

```
//VARIABLES PARA OBTENER CHUNK & KIND VALUES
omp_sched_t kind; int chunk_value;

//VARIABLES PARA EL CALCULO DEL TIEMPO
double ncgt, t_inicial, t_final;

//PASO 4: CALCULO DEL PRODUCTO
double suma;

#pragma omp parallel
{
  #pragma omp single
    t_inicial = omp_get_wtime();

  #pragma omp for firstprivate(suma) schedule(runtime)
  for(int i = 0; i<tam ; i++){
      #ifdef SEE_CHUNK_DEFAULT
        if(i==0){
          omp_get_schedule(&kind,&chunk_value);
          printf("KIND: %d | CHUNK: %d \n",kind,chunk_value);
        }
      #endif

      suma = 0;
      for(int j=i; j<tam; j++){
        suma += m[i][j] * v[j];
      }
      mv[i] = suma;
  }

  #pragma omp single
    t_final = omp_get_wtime();
}
```

DESCOMPOSICIÓN DE DOMINIO:



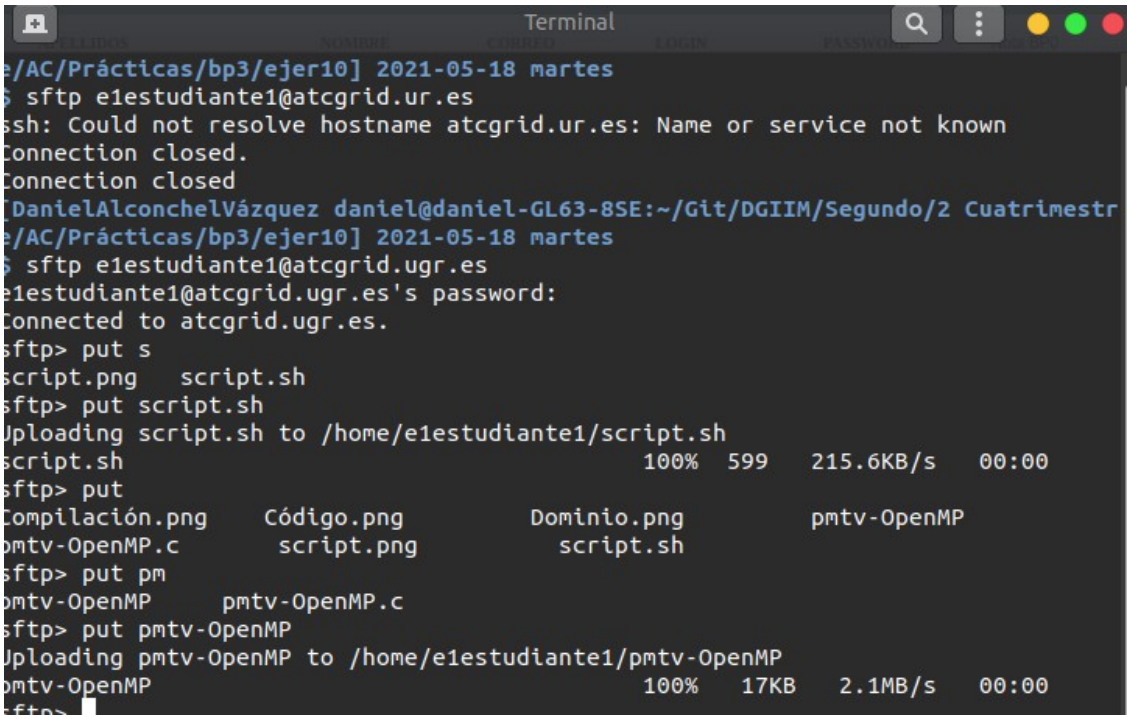
CAPTURAS DE PANTALLA:

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
script.sh                                                    buffers
#!/bin/bash

echo "STATIC MONOTONIC CHUNK=1\n"
export OMP_SCHEDULE="monotonic:static,1"
./pmtv-OpenMP 12800

echo "DYNAMIC MONOTONIC CHUNK=1\n"
export OMP_SCHEDULE="monotonic:dynamic,1"
./pmtv-OpenMP 12800

echo "GUIDED MONOTONIC CHUNK=1\n"
export OMP_SCHEDULE="monotonic:guided,1"
./pmtv-OpenMP 12800

echo "STATIC MONOTONIC CHUNK=64\n"
export OMP_SCHEDULE="monotonic:static,64"
./pmtv-OpenMP 12800

echo "DYNAMIC MONOTONIC CHUNK=64\n"
export OMP_SCHEDULE="monotonic:dynamic,64"
./pmtv-OpenMP 12800
NORMAL  script.sh                                  sh  3% ☰1/26 ㏑:1
```

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimest
e/AC/Prácticas/bp3/ejer10] 2021-05-18 martes
$ gcc -O2 -fopenmp pmtv-OpenMP.c -o pmtv-OpenMP
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimest
e/AC/Prácticas/bp3/ejer10] 2021-05-18 martes
```




```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ sbatch -p ac --hint=nomultithread script.sh
Submitted batch job 106618
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ cat s
script.sh         slurm-106618.out
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ cat slurm-106618.out
STATIC MONOTONIC CHUNK=1\n
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.110043153 || DIMENSION: 12800
DYNAMIC MONOTONIC CHUNK=1\n
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.109675888 || DIMENSION: 12800
GUIDED MONOTONIC CHUNK=1\n
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.114364456 || DIMENSION: 12800
STATIC MONOTONIC CHUNK=64\n
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.109446473 || DIMENSION: 12800
DYNAMIC MONOTONIC CHUNK=64\n
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.109208040 || DIMENSION: 12800
GUIDED MONOTONIC CHUNK=64\n
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.114264246 || DIMENSION: 12800
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$
```

```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ export OMP_SCHEDULE="monotonic:static"
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ ./pmtv-OpenMP 12800
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.027968604 || DIMENSION: 12800
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ export OMP_SCHEDULE="monotonic:dynamic,1"
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ ./pmtv-OpenMP 12800
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.022792988 || DIMENSION: 12800
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ export OMP_SCHEDULE="monotonic:guided,1"
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$ ./pmtv-OpenMP 12800
DIMENSION DE VECTOR Y MATRIZ: 12800 (4 B)
TIEMPO DE EJECUCION: 0.029837679 || DIMENSION: 12800
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp3] 2021-05-18 martes
$
```

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	0.027968604	0.022792988	0.029837679
1	0.110043153	0.109675888	0.114364456
64	0.109446473	0.109208040	0.114264246

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			