

2° curso / 2° 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): Pablo Olivares Martínez

Grupo de prácticas: 1

Fecha de entrega: 19/05/2021

Fecha evaluación en clase: 20/05/2021

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`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

int main(int argc, char **argv)
{
    int i, x, n=20, tid;
    int a[n], suma=0, sumalocal;
    if(argc < 2) {
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }

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

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

    x = atoi(argv[2]);
    if (x > 8) x = 8;

    #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);
    }

    return(0);
}
```

**CAPTURAS DE PANTALLA:**

```

Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer1] 2021-05-16 domingo
$ ./if-clauseModificado 8 4
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 3 suma de a[6]=6 sumalocal=6
thread 3 suma de a[7]=7 sumalocal=13
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
thread master=0 imprime suma=28
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer1] 2021-05-16 domingo
$ ./if-clauseModificado 12 8
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread 5 suma de a[9]=9 sumalocal=9
thread 7 suma de a[11]=11 sumalocal=11
thread 3 suma de a[6]=6 sumalocal=6
thread 3 suma de a[7]=7 sumalocal=13
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
thread 6 suma de a[10]=10 sumalocal=10
thread 4 suma de a[8]=8 sumalocal=8
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=66

```

**RESPUESTA:**

Estas instantáneas ilustran el funcionamiento de la cláusula `if` porque como vemos, cuando el número de iteraciones es mayor que 4 (es decir, cuando se aplica la cláusula), vemos que el número de hebras usado es el que hemos indicado como argumento, tal y como se buscaba.

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	2	0	0	0

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

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:** Cuando declaramos el planificador estático, las hebras se distribuyen en tiempo de compilación, mientras que el planificador dinámico las asigna en tiempo de ejecución. Aun así, ambas reparten las iteraciones en chunks. Sin embargo, el planificador guiado, siendo también en tiempo de ejecución, distribuye las iteraciones en bloques grandes cuyo tamaño mengua, sin ser nunca menor que el tamaño de un chunk. En cuanto al uso de `monotonic`, esto implica que la asignación de hebras para las iteraciones se realizan en orden lógico creciente, mientras que en `nonmonotonic` no importa.

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 modificador por defecto es `monotonic` para `static` y `nonmonotonic` para `dynamic` y `guided`. En cuanto al `chunk`, por defecto para `static` se asigna un único chunk por thread, es decir, número de iteraciones entre número de hebras; para `dynamic` se asignan unidades de una iteración y para `guided`, el tamaño por defecto de un chunk es el mismo que el de `static`. Para resolver esta pregunta, he ejecutado el programa del ejercicio anterior quitando los chunks para ver su valor por defecto. Luego para contrastar mis resultados, he investigado por diversas webs hasta que he dado con una bastante fiable (<https://software.intel.com/content/www/us/en/develop/articles/openmp-loop-scheduling.html>).

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`

```

#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, chunk_b, a[n], suma=0;
    omp_sched_t kind;

    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 == 0) {
            omp_get_schedule(&kind, &chunk_b);
            printf("DENTRO: dyn-var -- %d | nthreads-var -- %d | threads-limit-var -- %d | run-sched-var -- %d\n",
                omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_b);
        }
    }

    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("FUERA: dyn-var -- %d | nthreads-var -- %d | threads-limit-var -- %d | run-sched-var -- %d\n",
        omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_b);

    return(0);
}

```

**CAPTURAS DE PANTALLA:**

```

Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer4] 2021-05-16 domingo
$ export OMP_NUM_THREADS=2 && export OMP_DYNAMIC=FALSE && export OMP_SCHEDULE="static,4"
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer4] 2021-05-16 domingo
$ ./scheduled-clauseModificado 8 4
thread 0 suma a[0]=0 suma=0
DENTRO: dyn-var -- 0 | nthreads-var -- 2 | threads-limit-var -- 2147483647 | run-sched-var -- (kind -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: dyn-var -- 0 | nthreads-var -- 2 | threads-limit-var -- 2147483647 | run-sched-var -- (kind -2147483647 | chunk 4)

```

```

Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer4] 2021-05-16 domingo
$ export OMP_NUM_THREADS=4 && export OMP_DYNAMIC=TRUE && export OMP_SCHEDULE="dynamic,4"
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer4] 2021-05-16 domingo
$ ./scheduled-clauseModificado 8 4
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
thread 2 suma a[0]=0 suma=0
DENTRO: dyn-var -- 1 | nthreads-var -- 4 | threads-limit-var -- 2147483647 | run-sched-var -- (kind 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
Fuera de 'parallel for' suma=22
FUERA: dyn-var -- 1 | nthreads-var -- 4 | threads-limit-var -- 2147483647 | run-sched-var -- (kind 2 | chunk 4)

```

**RESPUESTA:** Como podemos ver, el valor de éstas tanto dentro como fuera del parallel son el mismo, pues son los valores establecidos en las variables de entorno.

- Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de



prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

#### CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado4.c

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

int main(int argc, char **argv) {
    int i, n=200, chunk, chunk_b, a[n], suma=0;
    omp_sched_t kind;

    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;

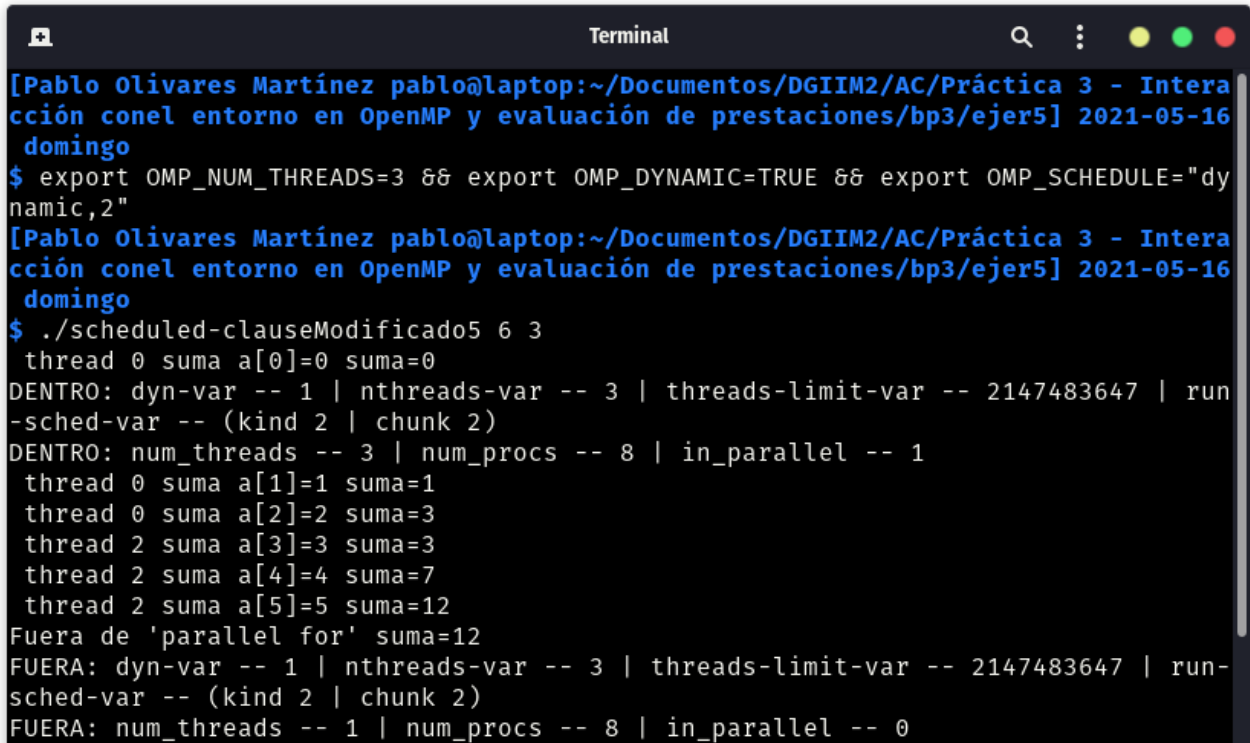
    #pragma omp parallel for firstprivate(suma) \
        lastprivate(suma) schedule(dynamic, chunk)
    for (i=0; i<n; i++)
    {
        suma = suma + a[i];
        printf(" thread %d suma a[%d]=%d suma=%d \n",
            omp_get_thread_num(), i, a[i], suma);

        if (i == 0) {
            omp_get_schedule(&kind, &chunk_b);

            printf("DENTRO: dyn-var -- %d | nthreads-var -- %d | threads-limit-var -- %d | run-s
            omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_b);
            printf("DENTRO: num_threads -- %d | num_procs -- %d | in_parallel -- %d \n",
            omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
        }
    }

    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("FUERA: dyn-var -- %d | nthreads-var -- %d | threads-limit-var -- %d | run-sched-va
    omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_b);
    printf("FUERA: num_threads -- %d | num_procs -- %d | in_parallel -- %d \n",
    omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

    return(0);
}
```

**CAPTURAS DE PANTALLA:**


```

Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer5] 2021-05-16 domingo
$ export OMP_NUM_THREADS=3 && export OMP_DYNAMIC=TRUE && export OMP_SCHEDULE="dynamic,2"
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer5] 2021-05-16 domingo
$ ./scheduled-clauseModificado5 6 3
thread 0 suma a[0]=0 suma=0
DENTRO: dyn-var -- 1 | nthreads-var -- 3 | threads-limit-var -- 2147483647 | run-sched-var -- (kind 2 | chunk 2)
DENTRO: num_threads -- 3 | num_procs -- 8 | in_parallel -- 1
thread 0 suma a[1]=1 suma=1
thread 0 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: dyn-var -- 1 | nthreads-var -- 3 | threads-limit-var -- 2147483647 | run-sched-var -- (kind 2 | chunk 2)
FUERA: num_threads -- 1 | num_procs -- 8 | in_parallel -- 0

```

**RESPUESTA:** En este ejercicio, vemos que obtenemos diferentes resultados tanto en `in_parallel` como en `num_threads` según estemos dentro o fuera de la región `parallel`. Esto se debe a que dentro del `parallel`, el programa utiliza el número de hebras que hemos establecido en este caso en la variable de entorno, pero fuera de la región `parallel` tan solo usa una. Por otro lado `in_parallel` nos dice si está ejecutándose en paralelo el trozo de código donde se encuentra la función, donde es evidente que es `false` fuera y `true` dentro del `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`



```

int main(int argc, char **argv) {
    int i, n=200, chunk, chunk_b, a[n], suma=0;
    omp_sched_t kind, STATIC = 1;

    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 == 1) {
            omp_set_dynamic(0);
            omp_set_num_threads(2);
            omp_set_schedule(STATIC, 2);
            omp_get_schedule(&kind, &chunk_b);

            printf("MODIFICADO:\n");
            printf("DENTRO: dyn-var -- %d | nthreads-var -- %d | threads-limit-var -- %d | run-s\n",
                omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_b);
        } else if (i == 0) {
            omp_get_schedule(&kind, &chunk_b);

            printf("SIN MODIFICAR:\n");
            printf("DENTRO: dyn-var -- %d | nthreads-var -- %d | threads-limit-var -- %d | run-s\n",
                omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_b);
        }
    }

    printf("Fuera de 'parallel for' suma=%d\n", suma);
    printf("FUERA: dyn-var -- %d | nthreads-var -- %d | threads-limit-var -- %d | run-sched-var\n",
        omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), kind, chunk_b);

    return(0);
}

```

**CAPTURAS DE PANTALLA:**

```

[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer6] 2021-05-16 domingo
$ ./scheduled-clauseModificado6 8 4
thread 1 suma a[0]=0 suma=0
SIN MODIFICAR:
DENTRO: dyn-var -- 1 | nthreads-var -- 3 | threads-limit-var -- 2147483647 | run
-sched-var -- (kind 2 | chunk 2)
thread 1 suma a[1]=1 suma=1
MODIFICADO:
DENTRO: dyn-var -- 0 | nthreads-var -- 2 | threads-limit-var -- 2147483647 | run
-sched-var -- (kind 1 | chunk 2)
thread 1 suma a[2]=2 suma=3
thread 1 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: dyn-var -- 1 | nthreads-var -- 3 | threads-limit-var -- 2147483647 | run
-sched-var -- (kind 1 | chunk 2)
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer6] 2021-05-16 domingo

```

**RESPUESTA:** Como vemos claramente, la iteración donde cambiamos estos valores, tienen un valor distinto a los establecidos al inicio del programa.

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

```

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

    if (argc < 2) {
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

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

#ifdef VECTOR_GLOBAL
    if (N > MAX) N = MAX;
#endif

#ifdef VECTOR_DYNAMIC
    double *v1, *v2, **M;
    v1 = (double *)malloc(N * sizeof(double));
    v2 = (double *)malloc(N * sizeof(double));
    M = (double **)malloc(N * sizeof(double *));

    // Reservamos memoria ahora para las componentes de la matriz
    for (i = 0; i < N; i++) {
        M[i] = (double *)malloc(N * sizeof(double));
    }
    if ((v1 == NULL) || (v2 == NULL) || (M == NULL)) {
        printf("No hay suficiente espacio para los vectores \n");
        exit(-2);
    }
#endif

    // Inicializamos la matriz y los vectores
    srand48(time(NULL));
    for (i = 0; i < N; i++) {
        v1[i] = drand48();
        v2[i] = 0;
        for (j = 0; j < N; j++) {
            if (j <= i)
                M[i][j] = drand48();
            else
                M[i][j] = 0;
        }
    }

    // Inicializamos la primera variable de tiempo
    clock_gettime(CLOCK_REALTIME, &cgt1);

```

```

// Inicializamos la primera variable de tiempo
clock_gettime(CLOCK_REALTIME,&cgt1);

// Calculamos el producto
for (i = 0; i < N; i++)
    for (j = 0; j <= i; j++)
        v2[i] += M[i][j] * v1[j];

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

// Para tamaños pequeños (hasta N=10), imprimimos todas las componentes del vector resul
// y el tiempo de ejecución
if (N <= 10) {
    printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n", N, ncgt);

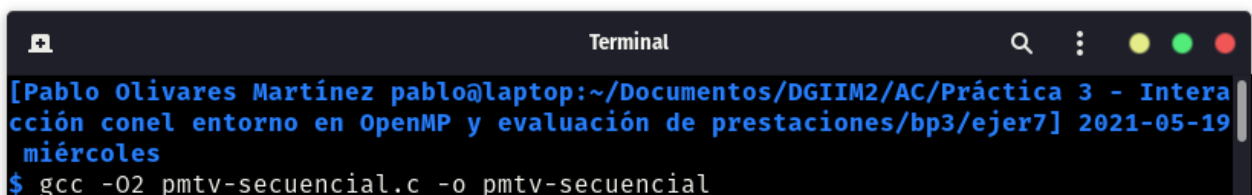
    for (i = 0; i < N; i++) {
        printf("v2[%i] = %f\n", i, v2[i]);
    }
}

// Para tamaños superiores, imprimimos el tiempo de ejecución y la primera y última comp
else {
    printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n Primera componente: %f\n Últ
        v2[0], v2[N - 1]);
}

#ifdef VECTOR_DYNAMIC
    free(v1);
    free(v2);
    for (i = 0; i < N; i++) {
        free(M[i]);
    }
    free(M);
#endif

return 0;
}

```

**CAPTURAS DE PANTALLA:**


```

Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer7] 2021-05-19
miércoles
$ gcc -O2 pmtv-secuencial.c -o pmtv-secuencial

```

```
e1estudiante21@atcgrid:~/bp3/ejer7
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp3/ejer7] 2021-05-19 miércoles
$ srun -p ac -n1 --cpus-per-task=1 --hint=nomultithread ./pmtv-secuencial 8
Tamaño vectores: 8
Tiempo de ejecución: 0.000000
v2[0] = 0.057534
v2[1] = 1.410381
v2[2] = 0.963545
v2[3] = 0.375341
v2[4] = 1.683424
v2[5] = 1.133315
v2[6] = 1.287757
v2[7] = 2.881652
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp3/ejer7] 2021-05-19 miércoles
```

**RESPUESTA:** Mientras que el programa de la anterior práctica tenía que hacer  $N^2$  operaciones, siendo  $N$  el tamaño de la matriz (por tanto siendo de orden  $O(N^2)$ ), este algoritmo tan solo requiere la suma de  $i=0$  hasta  $N$  de la suma de  $j=0$  hasta  $i$ , siendo dicho valor  $(N^2 + N) / 2$ , que, aún siendo  $O(N^2)$  también, es más eficiente pues requiere de menos operaciones.

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`



```
// Reservamos memoria ahora para las componentes de la matriz e inicializamos
#pragma omp parallel
{
    #pragma omp master
    {
        for (i = 0; i < tam; i++)
            M[i] = (double *)malloc(tam * sizeof(double));

        srand48(time(NULL));

        for (i = 0; i < tam; i++) {
            v1[i] = drand48();
            v2[i] = 0;
        }

        for (i = 0; i < tam; i++) {
            for (j = 0; j < tam; j++) {
                if (j <= i)
                    M[i][j] = drand48();
                else
                    M[i][j] = 0;
            }
        }
    }

    // Inicializamos la primera variable de tiempo
    #pragma omp single
    t1 = omp_get_wtime();

    // Calculamos el producto
    #pragma omp parallel for private(j) schedule(runtime) // Para paralelizar el for de
    for (i = 0; i < tam; i++) {
        for (j = 0; j <= i; j++) {
            v2[i] += M[i][j] * v1[j];
        }
    }

    // Obtenemos el tiempo total transcurrido
    #pragma omp single
    total = omp_get_wtime() - t1;
}
```

**DESCOMPOSICIÓN DE DOMINIO:**

M <sub>00</sub>	0	0	...	0	x	V <sub>0</sub>	=	V' <sub>0</sub>
M <sub>10</sub>	M <sub>11</sub>	0	...	0		V <sub>1</sub>		V' <sub>1</sub>
M <sub>20</sub>	M <sub>21</sub>	M <sub>22</sub>	...	0		V <sub>2</sub>		V' <sub>2</sub>
...	...	...	...	...		...		...
M <sub>N0</sub>	M <sub>N1</sub>	M <sub>N2</sub>	...	M <sub>NN</sub>		V <sub>N</sub>		V' <sub>N</sub>

Aquí hemos realizado una descomposición de dominio sobre la matriz M tal y como representan los colores escogidos. En este caso se habría hecho con una planificación static con valor de chunk 1 asignada en tiempo



de ejecución.

### CAPTURAS DE PANTALLA:

```
Terminal
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer8] 2021-05-19 miércoles
$ gcc -O2 -fopenmp pmtv-OpenMP.c -o pmtv-OpenMP
[Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 3 - Interacción con el entorno en OpenMP y evaluación de prestaciones/bp3/ejer8] 2021-05-19 miércoles
```

```
e1estudiante21@atcgrid:~/bp3/ejer8
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp3/ejer8] 2021-05-19 miércoles
$ srun -p ac -n1 --cpus-per-task=1 --hint=nomultithread ./pmtv-OpenMP 8
Tamaño vectores: 8
Tiempo de ejecución: 0.000016
v2[0] = 0.013575
v2[1] = 0.262602
v2[2] = 0.278245
v2[3] = 1.972599
v2[4] = 1.590697
v2[5] = 2.141332
v2[6] = 2.306218
v2[7] = 2.104434
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp3/ejer8] 2021-05-19 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:** como está establecido en `static` y `monotonic` con tamaño de `chunk` 1, cada hebra ejecutará una iteración en orden ascendente. De esta forma, tendremos que la hebra 0 realizará una iteración, la segunda 2 y así hasta la última fila que ejecutará  $N$  iteraciones, es decir, la suma de la serie aritmética hasta  $N$  que es  $(N^2+N)/2$ .

(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 número de operaciones por thread dependerá de la disponibilidad de las hebras, pues se van asignando en tiempo de ejecución. Esto hará que cada hebra realice un número diferente de operaciones. Sin embargo, mientras que cada thread ejecutará un múltiplo de `chunk` operaciones, en `guided` las asignaciones son de tamaño igual o mayor a `chunk`, por lo que probablemente cierta hebra con `guided` ejecute más operaciones que la que más ejecute `dynamic`, pero como hemos dicho, esto no es posible saberlo con certeza ya que se asignan según su disponibilidad.

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

**RESPUESTA:** En este caso, como tenemos iteraciones que crecen de tamaño y por tanto de tamaño diferente, considero que la mejor opción podría ser `guided`, seguida por `dynamic`, pues cuando una hebra acabe su ocupación puede tomar la siguiente iteración o `chunk` de iteraciones. Además, `guided` cada vez asigna `chunks` más pequeños y como cada vez tenemos más iteraciones, podría conseguir una distribución más uniforme.

**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 utilizados en el cuaderno de prácticas.  
**NOTA:** Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

#### CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

El código fuente empleado es exactamente el mismo que el del ejercicio 8.

**DESCOMPOSICIÓN DE DOMINIO:** La misma que el ejercicio 8.

**CAPTURAS DE PANTALLA:**

```
e1estudiante21@atcgrid:~/bp3/ejer10
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp3/ejer10] 2021-05-19 miércoles
$ sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread ./pmtv-OpenMP_atcgrid.sh
Submitted batch job 106978
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp3/ejer10] 2021-05-19 miércoles
$ sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread ./pmtv-OpenMP_atcgrid.sh
Submitted batch job 106983
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp3/ejer10] 2021-05-19 miércoles
$
```

#### TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

**SCRIPT:** pmtv-OpenMP\_atcgrid.sh

```
N=15360

declare -a list=("monotonic:static" "monotonic:dynamic" "monotonic:guided")

for i in "${list[@]}"
do
    for (( j = 0; j < 3; ++j )); do
        case $j in
            0)
                echo "PLANIFICACION: $i | CHUNK: DEFAULT"
                export OMP_SCHEDULE=$i
                ;;
            1)
                echo "PLANIFICACION: $i | CHUNK: 1"
                export OMP_SCHEDULE="$i,1"
                ;;
            2)
                echo "PLANIFICACION: $i | CHUNK: 64"
                export OMP_SCHEDULE="$i,64"
                ;;
        esac
    done
done
srun ./pmtv-OpenMP $N
```

**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	5.262942	4.993390	5.200945
1	5.402290	5.203451	4.239911
64	5.231029	5.184271	5.238659
Chunk	Static	Dynamic	Guided
por defecto	5.250700	5.186347	5.320422
1	5.184097	5.099935	5.232407
64	5.173724	5.100471	5.161540

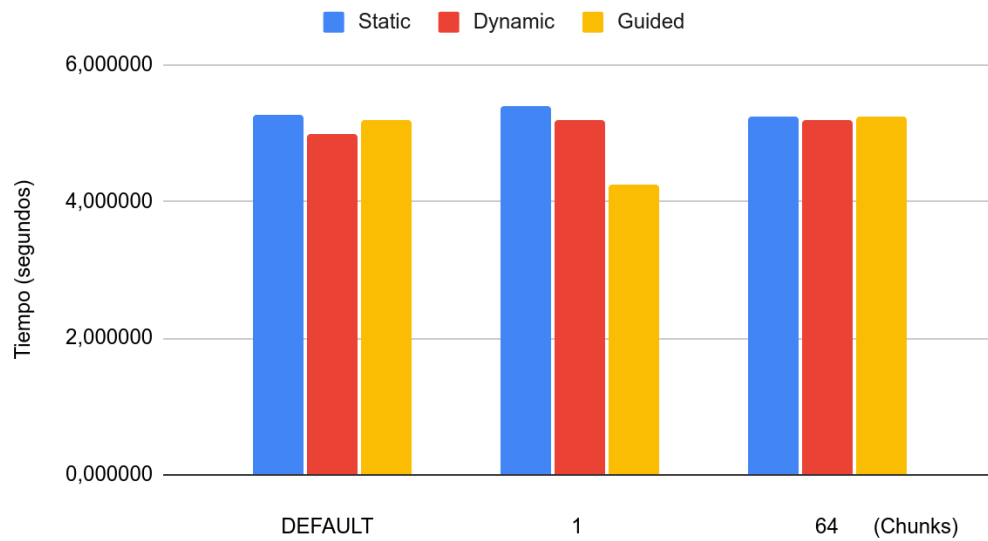
```

Abrir  file:///home/pablo/Documentos/DCIIM2/ACIP...evaluacion%20de%20prestaciones/bp3/ejer10  Guardar
1 Id. usuario del trabajo: elestudiente21
2 Id. del trabajo: 106978
3 Nombre del trabajo especificado por usuario: pmtv-OpenMP
4 Directorio de trabajo (en el que se ejecuta el script): /home/elestudiente21/bp3/ejer10
5 Cola: ac
6 Nodo que ejecuta este trabajo: atcgrid.ugr.es
7 Nº de nodos asignados al trabajo: 1
8 Nodos asignados al trabajo: atcgrid1
9 CPUs por nodo: 24
10 PLANIFICACION: monotonic:static | CHUNK: DEFAULT
11 Tamaño vectores: 15360
12 Tiempo de ejecución: 5.262942
13 Primera componente: 0.158611
14 Última componente: 7664.048613
15 PLANIFICACION: monotonic:static | CHUNK: 1
16 Tamaño vectores: 15360
17 Tiempo de ejecución: 5.402290
18 Primera componente: 0.760686
19 Última componente: 7664.869059
20 PLANIFICACION: monotonic:static | CHUNK: 64
21 Tamaño vectores: 15360
22 Tiempo de ejecución: 5.231029
23 Primera componente: 0.545220
24 Última componente: 7804.364528
25 PLANIFICACION: monotonic:dynamic | CHUNK: DEFAULT
26 Tamaño vectores: 15360
27 Tiempo de ejecución: 4.993390
28 Primera componente: 0.847109
29 Última componente: 7741.998351
30 PLANIFICACION: monotonic:dynamic | CHUNK: 1
31 Tamaño vectores: 15360
32 Tiempo de ejecución: 5.203451
33 Primera componente: 5.236392
34 Última componente: 7802.006152
35 PLANIFICACION: monotonic:dynamic | CHUNK: 64
36 Tamaño vectores: 15360
37 Tiempo de ejecución: 5.184271
38 Primera componente: 0.291302
39 Última componente: 7762.884423
40 PLANIFICACION: monotonic:guided | CHUNK: DEFAULT
41 Tamaño vectores: 15360
42 Tiempo de ejecución: 5.200945
43 Primera componente: 0.093230
44 Última componente: 4077.432828
45 PLANIFICACION: monotonic:guided | CHUNK: 1
46 Tamaño vectores: 15360
47 Tiempo de ejecución: 4.239911
48 Primera componente: 0.233503
49 Última componente: 7714.502133
50 PLANIFICACION: monotonic:guided | CHUNK: 64
51 Tamaño vectores: 15360
52 Tiempo de ejecución: 5.238659

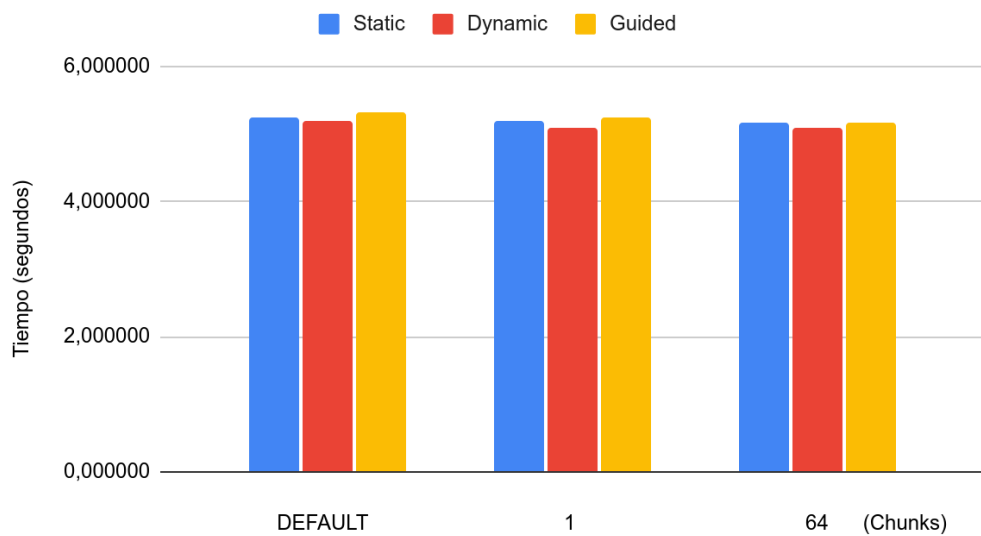
```

*Muestra de parte de los datos obtenidos con el script.*

### Comparación de tiempos pmtv-OpenMP (1ª Ejecución)



### Comparación de tiempos pmtv-OpenMP (2ª Ejecución)



#### *Resultado de las dos ejecuciones y sus correspondientes gráficas*

Según los resultados, podemos ver que guided ha obtenido el mejor resultado puntual, sin embargo, dynamic se ha mantenido por debajo de guided y static en la gran mayoría de las ejecuciones.