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): Francisco Javier Bolívar Expósito
Grupo de prácticas: D1
Fecha de entrega: 13/05/19
Fecha evaluación en clase: 14/05/19

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

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.

CAPTURA CÓDIGO FUENTE: `if-clauseModificado.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv)
{
    int i, n=20, tid;
    int a[n],suma=0,sumalocal;
    if(argc < 3) {
        fprintf(stderr,"[ERROR]- Usage: <ejecutable> <iteraciones> <num_threads>\n");
        exit(-1);
    }

    n = atoi(argv[1]); if (n>20) n=20;

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

    int x = atoi(argv[2]);
    #pragma omp parallel if(n>4) default(none) \
    private(sumalocal,tid) shared(a,suma,n) num_threads(x)
    { 
        sumalocal=0;
        tid=omp_get_thread_num();
        #pragma omp for private(i) schedule(static) nowait
        for (i=0; i<n; i++)
        {
            sumalocal += a[i];
            printf(" thread %d suma de a[%d]=%d sumalocal=%d \n", tid,i,a[i],sumalocal);
        }
        #pragma omp atomic
        suma += sumalocal;
        #pragma omp barrier
        #pragma omp master
        printf("thread master=%d imprime suma=%d\n",tid,suma);
    }
}
```

CAPTURAS DE PANTALLA:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer1] 2019-05-13 lunes
$./ifclause 6 2
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 1 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[4]=4 sumalocal=7
 thread 1 suma de a[5]=5 sumalocal=12
thread master=0 imprime suma=15
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer1] 2019-05-13 lunes
$./ifclause 6 5
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[3]=3 sumalocal=3
 thread 3 suma de a[4]=4 sumalocal=4
 thread 4 suma de a[5]=5 sumalocal=5
 thread 1 suma de a[2]=2 sumalocal=2
thread master=0 imprime suma=15
```

RESPUESTA:
Las capturas de pantalla muestran el funcionamiento de la cláusula, ya que si hay más de 4 iteraciones se cumple el if y se activa esta. Podemos ver como según el valor que pongamos en la cláusula cambia el número de threads que ejecuta el programa.

2. (a) Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,…15)
- chunck= 1, 2 y 4

Tabla 1 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	1	0	0
4	0	0	1	0	0	0	0	1	0
5	1	0	1	0	0	0	0	1	0
6	0	1	1	1	0	0	1	0	0
7	1	1	1	1	0	0	1	0	0
8	0	0	0	1	1	1	1	0	1
9	1	0	0	1	1	1	1	1	1
10	0	1	0	0	1	1	0	1	1
11	1	1	0	1	1	1	0	1	1
12	0	0	1	1	1	1	0	1	0
13	1	0	1	1	1	1	1	0	0
14	0	1	1	1	0	1	1	0	0
15	1	1	1	1	0	1	0	0	0

(b) Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

Tabla 2 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	1	0	0	0	1	0
1	1	0	0	2	0	0	0	1	0
2	2	1	0	0	1	0	0	1	0
3	3	1	0	3	1	0	2	3	0
4	0	2	1	0	2	2	2	3	2
5	1	2	1	0	2	2	2	3	2
6	2	3	1	0	3	2	3	3	2
7	3	3	1	0	3	2	3	0	2
8	0	0	2	0	0	1	1	0	3
9	1	0	2	2	0	1	1	2	3
10	2	1	2	3	0	1	1	2	3
11	3	1	2	3	0	1	1	2	3
12	0	2	3	3	0	3	0	2	1
13	1	2	3	0	0	3	0	1	1
14	2	3	3	0	0	3	0	1	1
15	3	3	3	0	0	3	0	1	1

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

RESPUESTA: Se reparten las iteraciones en unidades de chunks. En static se distribuyen las unidades en tiempo de ejecución con Round Robin entre todas las hebras. En dynamic se distribuyen en tiempo de ejecución, realizando un chunk mínimo de iteraciones, si una hebra es más rápida o tiene iteraciones más cortas puede realizar más iteraciones En guided cada hebra puede realizar muchas más iteraciones, pero como mínimo tiene que realizar un chunk.

3. Añadir al programa `scheduled-clause.c` lo necesario para que imprima el valor de las variables de control `dyn-var`, `nthreads-var`, `thread-limit-var` y `run-sched-var` dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```
omp_sched_t kind;
int chunk_size;

#pragma omp parallel for firstprivate(suma) \
lastprivate(suma) schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
»   if (i == 0)
»   {
»   »   omp_get_schedule(&kind, &chunk_size);
»   »   printf("Region parallel: %d, %d, %d, %d", omp_get_dynamic(), omp_get_max_threads(),
        omp_get_thread_limit(), kind, chunk_size);
»   }
»   suma = suma + a[i];
»   printf(" thread %d suma a[%d]=%d suma=%d \n", omp_get_thread_num(),i,a[i],suma);
}

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

omp_get_schedule(&kind, &chunk_size);
printf("Fuera de 'parrallel for': %d, %d, %d, %d", omp_get_dynamic(), omp_get_max_threads(),
omp_get_thread_limit(), kind, chunk_size);
```

CAPTURAS DE PANTALLA:

```
$./scheduledmod 5 2
Region parallel: 0, 4, 2147483647, 2, 1 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 2 suma a[4]=4 suma=4
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[3]=3 suma=5
Fuera de 'parallel for' suma=4
Fuera de 'parrallel for': 0, 4, 2147483647, 2, 1[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Univers
idad/AC/Prácticas/bp3/ejer3] 2019-05-13 lunes
$export OMP_NUM_THREADS=8
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer3] 2019-05-13 lunes
$./scheduledmod 5 3
Region parallel: 0, 8, 2147483647, 2, 1 thread 7 suma a[0]=0 suma=0
 thread 7 suma a[1]=1 suma=1
 thread 7 suma a[2]=2 suma=3
 thread 1 suma a[3]=3 suma=3
 thread 1 suma a[4]=4 suma=7
Fuera de 'parallel for' suma=7
Fuera de 'parrallel for': 0, 8, 2147483647, 2, 1[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Univers
idad/AC/Prácticas/bp3/ejer3] 2019-05-13 lunes
```

RESPUESTA: Se imprimen los mismos valores

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

```
#pragma omp parallel for firstprivate(suma) \
lastprivate(suma) schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
»   if (i == 0)
»   {
»   »   printf("Region parallel: %d, %d, %d", 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("Fuera de 'parallel for' suma=%d\n",suma);

printf("Region parallel: %d, %d, %d", omp_get_num_threads(), omp_get_num_procs, omp_in_parallel());
```

CAPTURAS DE PANTALLA:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer4] 2019-05-13 lunes
$gcc -fopenmp scheduled-clauseModificado4.c -o scheduledmod4
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer4] 2019-05-13 lunes
$./scheduledmod4 5 2
Region parallel: 8, 1258905232, 1 thread 4 suma a[0]=0 suma=0
 thread 4 suma a[1]=1 suma=1
 thread 6 suma a[2]=2 suma=2
 thread 6 suma a[3]=3 suma=5
 thread 7 suma a[4]=4 suma=4
Fuera de 'parallel for' suma=4
Region parallel: 1, 1258905232, 0[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Práctic
as/bp3/ejer4] 2019-05-13 lunes
```

RESPUESTA: Se obtienen valores distintos en omp_get_num_threads y omp_in_parallel.

5. Añadir al programa `scheduled-clause.c` lo necesario para modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```
#pragma omp parallel for firstprivate(suma) \
lastprivate(suma) schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
»   if (i == 0)
»   {
»   »   omp_get_schedule(&kind, &chunk_size);
»   »   printf("Region parallel: %d, %d, %d, %d, %d", omp_get_dynamic(), omp_get_max_threads(),
        kind, chunk_size);
»   »
»   »   omp_set_dynamic(0);
»   »   omp_set_num_threads(2);
»   »   omp_set_schedule(0, 4);
»   »
»   »   omp_get_schedule(&kind, &chunk_size);
»   »   printf("Region parallel: %d, %d, %d, %d, %d", omp_get_dynamic(), omp_get_max_threads(),
        kind, chunk_size);
»   }
»   suma = suma + a[i];
»   printf(" thread %d suma a[%d]=%d suma=%d \n", omp_get_thread_num(),i,a[i],suma);
}

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

omp_get_schedule(&kind, &chunk_size);
printf("Fuera de 'parrallel for': %d, %d, %d, %d, %d", omp_get_dynamic(), omp_get_max_threads(),
omp_get_thread_limit(), kind, chunk_size);

omp_set_dynamic(0);
omp_set_num_threads(2);
omp_set_schedule(0, 4);

omp_get_schedule(&kind, &chunk_size);
printf("Fuera de 'parrallel for': %d, %d, %d, %d, %d", omp_get_dynamic(), omp_get_max_threads(),
omp_get_thread_limit(), kind, chunk_size);
```

CAPTURAS DE PANTALLA:

RESPUESTA:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer5] 2019-05-13 lunes
$gcc -fopenmp scheduled-clauseModificado5.c -o scheduledmod5
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer5] 2019-05-13 lunes
$./scheduledmod5 16 5
 thread 2 suma a[10]=10 suma=10
 thread 2 suma a[11]=11 suma=21
 thread 2 suma a[12]=12 suma=33
 thread 2 suma a[13]=13 suma=46
 thread 2 suma a[14]=14 suma=60
 thread 7 suma a[15]=15 suma=15
 thread 6 suma a[5]=5 suma=5
 thread 6 suma a[6]=6 suma=11
 thread 6 suma a[7]=7 suma=18
 thread 6 suma a[8]=8 suma=26
 thread 6 suma a[9]=9 suma=35
Region parallel: 0, 8, 2, 1, 1Region parallel: 0, 2, 2, 1, 1705131159 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
 thread 1 suma a[4]=4 suma=10
Fuera de 'parallel for' suma=15
Fuera de 'parrallel for': 0, 8, 2147483647, 2, 1Fuera de 'parrallel for': 0, 2, 2147483647, 2, 1[FranciscoJavierBolí
varExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer5] 2019-05-13 lunes
```

Resto de ejercicios

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

CAPTURA CÓDIGO FUENTE: `pmtv-secuencial.c`

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

//#define MATRIX_GLOBAL
#define MATRIX_DYNAMIC

#ifdef MATRIX_GLOBAL
#define MAX 20000

double m[MAX][MAX], v1[MAX], vf[MAX];
#endif
int main(int argc, char** argv)
{
»   if (argc<2)
»   {»
»   »   printf("Faltan nº columnas de la matriz\n");
»   »   exit(-1);
»   }

»   unsigned int N = atoi(argv[1]);
»
»   #ifdef MATRIX_GLOBAL
»   if (N>MAX) N=MAX;
»   #endif
»   #ifdef MATRIX_DYNAMIC
»   double *v1, *vf;
»   double **m;
»
»   v1 = (double*) malloc(N * sizeof(double));
»   vf = (double*) malloc(N * sizeof(double));
»   m = (double**) malloc(N * sizeof(double*));
»
»   for (int i=0; i<N; i++)
»   »   m[i] = (double*)malloc(N * sizeof(double));
»
»   if ((v1 == NULL) || (vf == NULL) || (m == NULL))
»   {»
»   »   printf("No hay suficiente espacio para reservar memoria \n");
»   »   exit(-2);
»   }
»   #endif

»   struct timespec cgt1,cgt2;
»   double ncgt;
»
»   printf("Tamaño Matriz:%u x %u (%u B)\n", N, N, sizeof(unsigned int));
»   //Inicializacion
»   for (int i=0; i<N; ++i){
»   »   v1[i] = N*0.1+i*0.1;
»   »   for (int j=0; j<N; ++j)
»   »   »   m[i][j] = N*0.1+i*0.1;
»   }
```

```
»   //Multiplicacion
»   clock_gettime(CLOCK_REALTIME,&cgt1);
»
»   for (int i=0; i<N; ++i)
»   »   for (int j=i; j<N; ++j)
»   »   »   vf[i] += m[i][j] * v1[j];

»   clock_gettime(CLOCK_REALTIME,&cgt2);
»
»   //Salida del resultado y el tiempo de ejecucion
»   ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
      (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

»   if (N<12)
»   {
»   »   printf("Tiempo:%11.9f\t / Tamaño columnas:%u\n",ncgt,N);
»   »   for (int i=0; i<N; ++i)
»   »   {
»   »   »   printf("/ ");
»   »   »   for (int j=0; j<N-1; ++j)
»   »   »   »   printf("(%8f*%8f)+", m[i][j], v1[j]);
»   »   »   printf("(%8f*%8f) =", m[i][N-1], v1[N-1]);
»   »   »   printf(" V2[%d] = %8f /\n", i, vf[i]);
»   »   }
»   }
»   else
»   »   printf("Tiempo:%11.9f\t / Tamaño columnas:%u\t/ V2[0] = %8.6f / / V2[%d] = %8.6f /\n", ncgt, N,
        vf[0], N-1, vf[N-1]);

»   #ifdef MATRIX_DYNAMIC
»   »   for(int i = 0; i < N; ++i)
»   »   »   free(m[i]);
»   »   free(m);
»   »   free(v1);
»   »   free(vf);
»   #endif

»   return 0;
}
```

CAPTURAS DE PANTALLA:

```
$./pmtv 100
Tamaño Matriz:100 x 100 (4 B)
Tiempo:0.000068145       / Tamaño columnas:100  / V2[0] = 14950.000000 / / V2[99] = 396.010000 /
```

La complejidad del producto vector por matriz era de NxM (una operación por cada componente de la matriz). Ahora es NxM/2 al solo hacer los calculos de la mitad de la matriz.

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de

tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en atcgrid los tiempos de ejecución del código paralelo (usando, como siempre, -O2 al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 1, 64 y el `chunk` por defecto para la alternativa. Use un tamaño de vector N múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para `chunk` con `static`, `dynamic` y `guided`? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación `static` para cada uno de los `chunks`? (c) Con la asignación `dynamic` y `guided`, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

RESPUESTA:

La alternativa que ofrece mejor resultados es Dynamic, 64. Ya que hay muchas iteraciones que repartir el chunk 64 da menos overhead en todas las alternativas y dynamic proporciona que las hebras más rápidas ejecuten más iteraciones.

a) Usando la funcion omp_get_schedule(&kind, &chunk_size) podemos obtener el chunk que se está usando. Devuelve Static: 0, Dynamic: 1, Guided: 1

b) 15360/2/chunk

c) Depende de la velocidad de los threads cuantas operaciones realizará cada uno

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
#pragma omp parallel for schedule(runtime)
for (int i=0; i<N; ++i)
{
»   for (int j=i; j<N; ++j)
»   »   vf[i] += m[i][j] * v1[j];
}
```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer7] 2019-05-13 lunes
$cat pmtv_script.sh.o21042
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 0Tiempo:0.061796821       / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 1Tiempo:0.077826911       / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 64Tiempo:0.077836986      / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 1Tiempo:0.061697596       / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 1Tiempo:0.062031929       / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 64Tiempo:0.054402862      / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 1Tiempo:0.062630227       / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 1Tiempo:0.062685566       / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
Tamaño Matriz:15360 x 15360 (4 B)
Chunk: 64Tiempo:0.060322813      / Tamaño columnas:15360        / V2[0] = 54357000192.000000 / / V2[15359] = 9436569
.610000 /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer7] 2019-05-13 lunes
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
$cat pmtv_script.sh
#!/bin/bash
export OMP_NUM_THREADS=12

export OMP_SCHEDULE="static"
./pmtv 15360

export OMP_SCHEDULE="static,1"
./pmtv 15360


export OMP_SCHEDULE="static,64"
./pmtv 15360


export OMP_SCHEDULE="dynamic"
./pmtv 15360


export OMP_SCHEDULE="dynamic,1"
./pmtv 15360


export OMP_SCHEDULE="dynamic,64"
./pmtv 15360


export OMP_SCHEDULE="guided"
./pmtv 15360

export OMP_SCHEDULE="guided,1"
./pmtv 15360

export OMP_SCHEDULE="guided,64"
./pmtv 15360
```

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.061796821	0.061697596	0.062630227
1	0.077826911	0.062031929	0.062685566
64	0.077836986	0.054402862	0.060322813

Chunk	Static	Dynamic	Guided
por defecto	0.062198657	0.062218764	0.063549875
1	0.077598435	0.062354446	0.061568765
64	0.077754981	0.054215465	0.060122423

8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

$$A = B \bullet C;\ A(i,j) = \sum_{k=0}^{N-1} B(i,k) \bullet C(k,j),\ i,j = 0,\ldots N-1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (`N`-1, `N`-1) del resultado antes de que termine el programa.

CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

```
#ifdef MATRIX_DYNAMIC
double **m1, **m2, **m3;

m1 = (double**) malloc(N * sizeof(double*));
m2 = (double**) malloc(N * sizeof(double*));
m3 = (double**) malloc(N * sizeof(double*));

for (int i=0; i<N; i++)
{
»   m1[i] = (double*)malloc(N * sizeof(double));
»   m2[i] = (double*)malloc(N * sizeof(double));
»   m3[i] = (double*)malloc(N * sizeof(double));
}

if ((m1 == NULL) || (m2 == NULL) || (m3 == NULL))
{»
»   printf("No hay suficiente espacio para reservar memoria \n");
»   exit(-2);
}
#endif

struct timespec cgt1,cgt2;
double ncgt;

printf("Tamaño Matriz:%u x %u (%u B)\n", N, N, sizeof(unsigned int));
//Inicializacion
for (int i=0; i<N; ++i){
»   for (int j=0; j<N; ++j)
»   {
»   »   m1[i][j] = N*0.1+i*0.1;
»   »   m2[i][j] = N*0.1+i*0.1;
»   }
}

//Multiplicacion
clock_gettime(CLOCK_REALTIME,&cgt1);

for (int i=0; i<N; ++i)
»   for (int j=0; j<N; ++j)
»   »   for (int k=0; k < N; ++k)
»   »   »   m3[i][j] += m1[i][k] * m2[k][j];

clock_gettime(CLOCK_REALTIME,&cgt2);

//Salida del resultado y el tiempo de ejecucion
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
   (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));


printf("Tiempo:%11.9f\t / Tamaño columnas:%u\t/ m3[0] = %8.6f / / m3[%d] = %8.6f /\n", ncgt, N, m3[0], N-
m3[N-1]);

#ifdef MATRIX_DYNAMIC
»   for(int i = 0; i < N; ++i)
»   {
»   »   free(m1[i]);
»   »   free(m2[i]);
»   »   free(m3[i]);
»   }
»   free(m1);
»   free(m2);
»   free(m3);
```

CAPTURAS DE PANTALLA:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer8] 2019-05-13 lunes
$gcc pmm-secuencial.c -O2 -o pmm
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp3/ejer8] 2019-05-13 lunes
$./pmm 100
Tamaño Matriz:100 x 100 (4 B)
Tiempo:0.003603113       / Tamaño columnas:100  / m3[0] = 14950.000000 / / m3[99] = 29750.500000 /
```

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2,Lección 5/Tema 2).

DESCOMPOSICIÓN DE DOMINIO:

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

```
int j, k;
#pragma omp parallel for private(j)
for (int i=0; i<N; ++i){
»   for (j=0; j<N; ++j)
»   {
»   »   m1[i][j] = N*0.1+i*0.1;
»   »   m2[i][j] = N*0.1+i*0.1;
»   }
}

//Multiplicacion
cgt1 = omp_get_wtime();

#pragma omp parallel for private(j,k)
for (int i=0; i<N; ++i)
{
»   for (j=0; j<N; ++j)
»   {
»   »   for (k=0; k < N; ++k)
»   »   »   m3[i][j] += m1[i][k] * m2[k][j];
»   }
}
```

CAPTURAS DE PANTALLA:

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos del computador. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

ESTUDIO DE ESCALABILIDAD EN atcgrid:

SCRIPT: pmm-OpenMP_atcgrid.sh

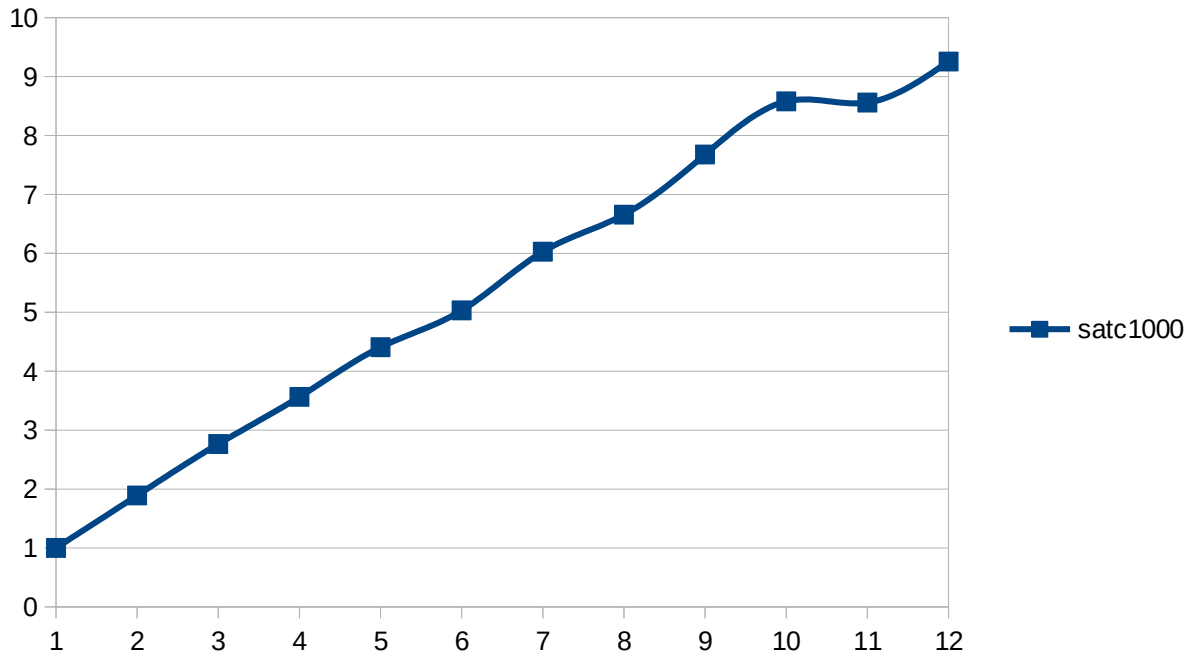
```
#!/bin/bash
for ((N=1;N<13;N=N+1)) do
        export OMP_NUM_THREADS=$N
        ./pmm-openmp 1000
        ./pmm-openmp 150
done
```

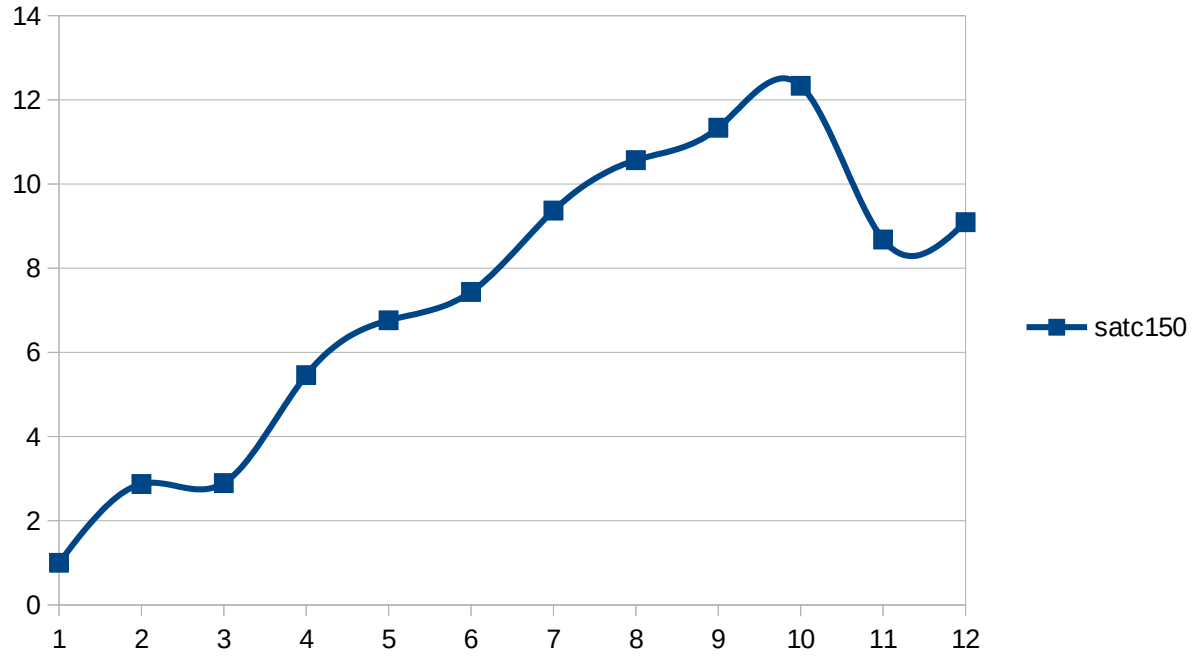
Atcgrid(1500)

1	2	3	4	5	6	7	8	9	10	11	12
10,144829012	5,36280736	3,669835722	2,846487691	2,301858243	2,016264952	1,682934848	1,524095904	1,321363399	1,182510866	1,185393052	1,096106865
1	1,8917011802	2,7643823268	3,5639813389	4,407234478	5,0314959857	6,028058082	6,6562930754	7,6775465551	8,5790577522	8,5581984768	9,2553284136

Atcgrid(150)

1	2	3	4	5	6	7	8	9	10	11	12
0,009443914	0,003288508	0,003261792	0,001729489	0,001396795	0,00126999	0,001008117	0,000893567	0,000832952	0,000765679	0,001088004	0,001038513
1	2,8717929225	2,8953146001	5,4605227324	6,7611310178	7,4362113088	9,367874959	10,568781076	11,337885016	12,334038154	8,6800361028	9,0936887646





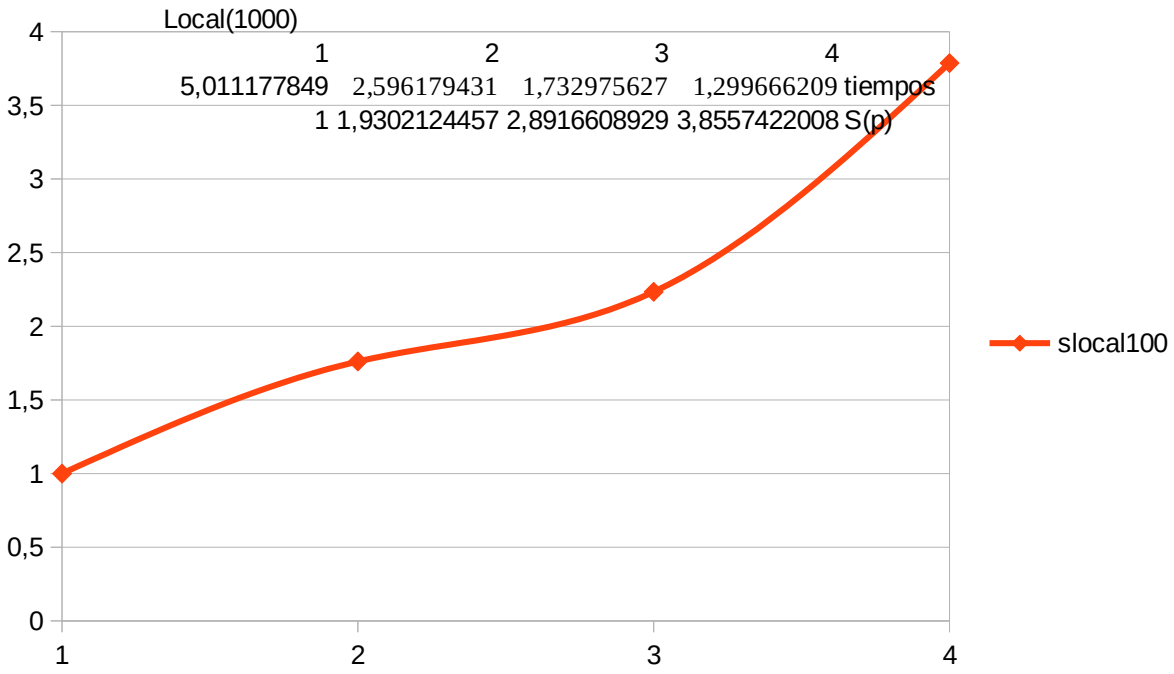
ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

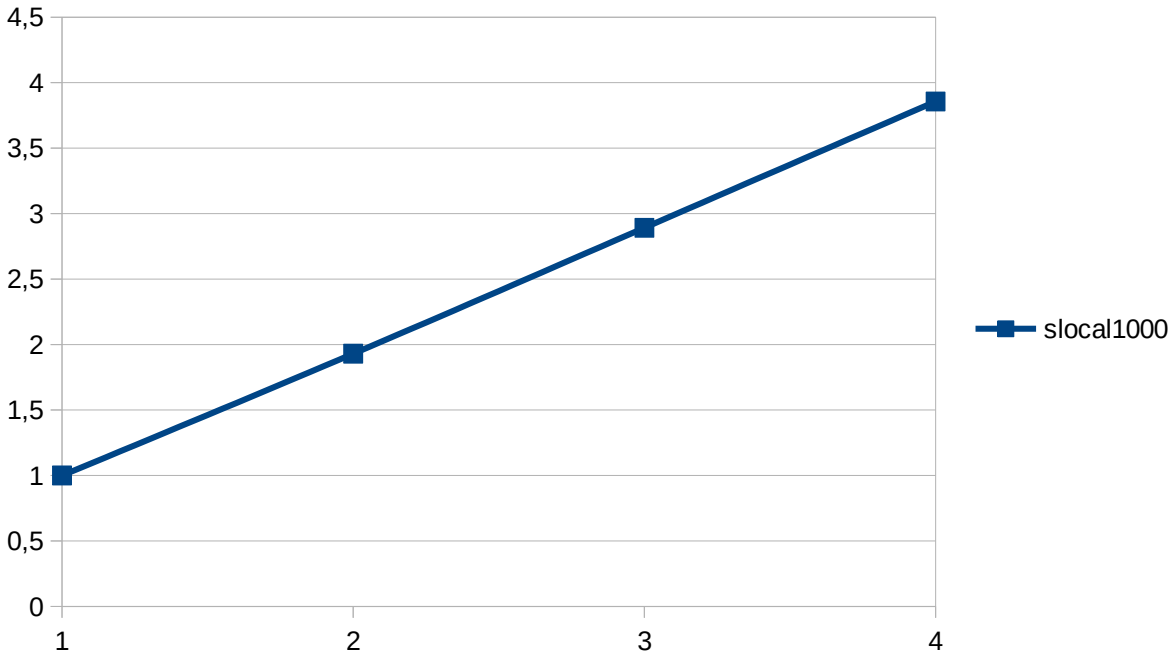
SCRIPT: pmm-OpenMP_pclocal.sh

```
$cat pmm-OpenMP_script_local.sh
#!/bin/bash
for ((N=1;N<5;N=N+1)) do
        export OMP_NUM_THREADS=$N
        ./pmm-openmp 1000
        ./pmm-openmp 100
done
```

Local(100)

	1	2	3	4	
	0,000942859	0,000535434	0,00042209	0,000248985	tiempos
	1	1,7609247825	2,2337866332	3,7868104504	S(p)


Local(1000)
1 2 3 4
5,011177849 2,596179431 1,732975627 1,299666209 tiempos
1 1,9302124457 2,8916608929 3,8557422008 S(p)
4
3,5
3
2,5
2
1,5
1
0,5
0
1 2 3 4
slocal100


4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
1 2 3 4
slocal1000