2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

# Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Fernando Vallecillos Ruiz
Grupo de prácticas: A1
Fecha de entrega: 12/05/2018
Fecha evaluación en clase: 14/05/2018

### 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`

```
int main(int argc, char **argv)
{
   int i, n=20, tid, m;
   int a[n],suma=0,sumalocal;
   if(argc < 3)     {
      fprintf(stderr,"[ERROR]-Falta iteraciones y numero threads\n");
      exit(-1);
     }
   n = atoi(argv[1]);
   m = atoi(argv[2]);
    if (n>20) n=20;
   for (i=0; i<n; i++) {
      a[i] = i;
   }

  #pragma omp parallel if(n>4) default(none) \
                    private(sumalocal,tid) shared(a,suma,n) num_threads(m)
  {  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:**

```
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-04-30 Monday
$./if-clause 4 4
 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 0 suma de a[3]=3 sumalocal=6
thread master=0 imprime suma=6
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-04-30 Monday
$./if-clause 5 5
 thread 3 suma de a[3]=3 sumalocal=3
 thread 0 suma de a[0]=0 sumalocal=0
 thread 1 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[2]=2 sumalocal=2
 thread 4 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10
```

**RESPUESTA**:
Esta cláusula hace que en el caso de que no se cumpla la condición dentro de ella, el parallel no se ejecuto en paralelo. Es decir, quita la directiva que permite el paralelismo.
En el ejemplo incluido, esta condicion es *n>4*, por lo que en el primer intento, podemos apreciar que todas las operaciones son ejecutadas por la misma (y única) hebra. Sin embargo, al poner este valor a 5 en la segunda ejecución, podemos ver como las iteraciones son repartidas entre las 5 hebras.

**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 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 9 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 10 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 11 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 12 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 13 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 14 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

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

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

**RESPUESTA**:

La principal diferencia es que *static*, realiza el reparto de forma estática, es decir, el orden es decidido antes de ejecutar el código y por tanto siempre será el mismo. Sin embargo, en *dynamic* y *guided*, el el orden es calculado en tiempo de ejecución. Esto hace que el orden pueda ser diferente en cada ejecución, lo hace mas versátil ya que lo puede repartir según las necesidades de cada momento. Pero tiene que hacer cálculos extra para repartir, asi que es un costo extra también.
La diferencia entre *dynamic* y *guided*, es la forma en la que reparten a lo largo del tiempo. Mientras que *dynamic* mantiene su *chunk* constante, y lo reparte según las necesidades. El *chunk* de *guided*, varía a lo largo del tiempo. Con *guided*, se comienza con un bloque largo y este va menguando, nunca mas pequeño que *chunk* (salvo en la última iteración).

**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`

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

#pragma omp parallel num_threads(4)
 {
     #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
        for (i=0; i<n; i++){
             suma = suma + a[i];
             printf(" thread %d suma a[%d]=%d suma=%d \n",omp_get_thread_num(),i,a[i],suma);
             if(i==0){
                 printf("Dentro del parallel.\n");
                 printf("Variable dyn-var: %s\n",omp_get_dynamic() ? "true" : "false");
                 printf("Variable nthreads-var: %d\n",omp_get_max_threads());
                 printf("Variable thread-limit-var: %d\n",omp_get_thread_limit());
                 omp_get_schedule(&k, &c);
                 if(k == omp_sched_static)
                     printf("Variable run-sched-var: estatica, tamaño chunk: %d\n", c);
                 else if(k == omp_sched_dynamic)
                     printf("Variable run-sched-var: dinamica, tamaño chunk: %d\n", c);
                 else
                     printf("Variable run-sched-var: guided, tamaño chunk: %d\n", c);
             }
         }

 }

 printf("Fuera de 'parallel' suma=%d\n",suma);
 printf("Variable dyn-var: %s\n",omp_get_dynamic() ? "true" : "false");
 printf("Variable nthreads-var: %d\n",omp_get_max_threads());
 printf("Variable thread-limit-var: %d\n",omp_get_thread_limit());
 omp_get_schedule(&k, &c);
 if(k == omp_sched_static)
     printf("Variable run-sched-var: estatica, tamaño chunk: %d\n", c);
 else if(k == omp_sched_dynamic)
     printf("Variable run-sched-var: dinamica, tamaño chunk: %d\n", c);
 else
     printf("Variable run-sched-var: guided, tamaño chunk: %d\n", c);
return(0);
```

**CAPTURAS DE PANTALLA:**

```
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-07 Monday
$./scheduled-clauseModificado 8 4
 thread 1 suma a[0]=0 suma=0
Dentro del parallel.
 thread 2 suma a[4]=4 suma=4
 thread 2 suma a[5]=5 suma=9
 thread 2 suma a[6]=6 suma=15
 thread 2 suma a[7]=7 suma=22
Variable dyn-var: false
Variable nthreads-var: 4
Variable thread-limit-var: 2147483647
Variable run-sched-var: dinamica, tamaño chunk: 1
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=3
 thread 1 suma a[3]=3 suma=6
Fuera de 'parallel' suma=22
Variable dyn-var: false
Variable nthreads-var: 4
Variable thread-limit-var: 2147483647
Variable run-sched-var: dinamica, tamaño chunk: 1
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-07 Monday
$export OMP_THREAD_LIMIT=30
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-07 Monday
$export OMP_DYNAMIC=TRUE
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-07 Monday
$export OMP_NUM_THREADS=15
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-07 Monday
$./scheduled-clauseModificado 8 4
 thread 3 suma a[4]=4 suma=4
 thread 3 suma a[5]=5 suma=9
 thread 3 suma a[6]=6 suma=15
 thread 3 suma a[7]=7 suma=22
 thread 1 suma a[0]=0 suma=0
Dentro del parallel.
Variable dyn-var: true
Variable nthreads-var: 15
Variable thread-limit-var: 30
Variable run-sched-var: dinamica, tamaño chunk: 1
 thread 1 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=3
 thread 1 suma a[3]=3 suma=6
Fuera de 'parallel' suma=22
Variable dyn-var: true
Variable nthreads-var: 15
Variable thread-limit-var: 30
Variable run-sched-var: dinamica, tamaño chunk: 1
```

**RESPUESTA**:

Los resultados que se obtienen son los mismos tanto dentro como fuera de la región paralela. Nosotros estamos modificando y consultando las variables de entorno que son utilizadas por defecto cuando no especificamos nada. Sin embargo, en la región paralela le hemos especificado el tipo de reparto (dinamico y el chunk) y el número de threads. Esto hace que el reparto no cambie y que los valores sean los mismos dentro y fuera del parallel.

**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 num_threads(4)
 {
     #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
        for (i=0; i<n; i++){
             suma = suma + a[i];
             printf(" thread %d suma a[%d]=%d suma=%d \n",omp_get_thread_num(),i,a[i],suma);
             if(i==0){
                 printf("Dentro del parallel.\n");
                 printf("Get numero threads: %d\n",omp_get_num_threads());
                 printf("Get numero procesadores: %d\n",omp_get_num_procs());
                 printf("En paralelo: %s\n",omp_in_parallel() ? "true" : "false");

             }
         }

 }

 printf("Fuera de 'parallel' suma=%d\n",suma);
 printf("Get numero threads: %d\n",omp_get_num_threads());
 printf("Get numero procesadores: %d\n",omp_get_num_procs());
 printf("En paralelo: %s\n",omp_in_parallel() ? "true" : "false");
```

**CAPTURAS DE PANTALLA:**

```
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-07 Monday
$./scheduled-clauseModificado4 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 del parallel.
Get numero threads: 4
Get numero procesadores: 4
En paralelo: true
 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' suma=22
Get numero threads: 1
Get numero procesadores: 4
En paralelo: false
```

**RESPUESTA**:

Tanto en *omp_get_num_threads()* como en *omp_in_parallel()* se obtienen valores distintos fuera y dentro de la región paralela. La primera función nos da el número de hebras que estan siendo utilizadas en un instante, por lo que debe ser distinto, y la segunda función nos dice si estamos dentro o fuera de una región paralela. Sin embargo, la funcion *omp_get_num_procs* nos devuelve el número de procesadores, esta variable no cambia dentro o fuera de la región paralela.

**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 num_threads(4)
 {
     #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
        for (i=0; i<n; i++){
             suma = suma + a[i];
             printf(" thread %d suma a[%d]=%d suma=%d \n",omp_get_thread_num(),i,a[i],suma);
             if(i==0){
                 printf("Variable dyn-var: %s\n",omp_get_dynamic() ? "true" : "false");
                 printf("Variable nthreads-var: %d\n",omp_get_max_threads());
                 omp_get_schedule(&k, &c);
                 if(k == omp_sched_static)
                     printf("Variable run-sched-var: estatica, tamaño chunk: %d\n", c);
                 else if(k == omp_sched_dynamic)
                     printf("Variable run-sched-var: dinamica, tamaño chunk: %d\n", c);
                 else
                     printf("Variable run-sched-var: guided, tamaño chunk: %d\n", c);
             }
         }

 }

 omp_set_dynamic(true);
 omp_set_num_threads(10);
 omp_set_schedule(omp_sched_static, 3);
 printf("Cambio las variables\n.");
 printf("Fuera de 'parallel' suma=%d\n",suma);
 printf("Variable dyn-var: %s\n",omp_get_dynamic() ? "true" : "false");
 printf("Variable nthreads-var: %d\n",omp_get_max_threads());
 omp_get_schedule(&k, &c);
 if(k == omp_sched_static)
     printf("Variable run-sched-var: estatica, tamaño chunk: %d\n", c);
 else if(k == omp_sched_dynamic)
     printf("Variable run-sched-var: dinamica, tamaño chunk: %d\n", c);
 else
     printf("Variable run-sched-var: guided, tamaño chunk: %d\n", c);
```

**CAPTURAS DE PANTALLA:**

```
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-07 Monday
$./scheduled-clauseModificado5 8 4
 thread 2 suma a[0]=0 suma=0
Variable dyn-var: false
Variable nthreads-var: 4
Variable run-sched-var: dinamica, tamaño chunk: 1
 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 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
Cambio las variables
.Fuera de 'parallel' suma=22
Variable dyn-var: true
Variable nthreads-var: 10
Variable run-sched-var: estatica, tamaño chunk: 3
```

**RESPUESTA**:

Ya que como hemos visto anteriormente, el resultado de las variables es igual dentor y fuera de la región paralell, decidí modificarlo entre estas dos para que el cambio fuera visible.

## 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.

El anterior código tenía una complejidad de $n^2$ ya que por cada fila ($n$ filas en total) se producen $n$ operaciones. El nuevo código al ser una matriz triangular, por cada fila se disminuye en 1 el número de operaciones. Comenzando desde uno hasta llegar a $n$. Por lo que tenemos la mitad de operaciones en total. Esto produce que la complejidad sea $n*n/2$.

**CAPTURA CÓDIGO FUENTE**: `pmtv-secuencial.c`

```
//Inicializar vectores
  for(i=0; i<N; i++){
    v2[i] = N*0.1-i*0.1;
    v3[i] = 0;
    for (j = i ; j<N; j++)
        v1[i][j] = N*0.1+j*0.1;
  }


clock_gettime(CLOCK_REALTIME,&cgt1);
for(i = 0; i < N; i++)
    for(j= i; j < N; j++)
        v3[i] = v3[i] + v1[i][j]*v2[j];

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

**CAPTURAS DE PANTALLA:**

```
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-09 Wednesday
$./pmtv-secuencial 10


V3
V[0]=7.150000V[1]=6.150000V[2]=5.160000V[3]=4.200000V[4]=3.290000V[5]=2.450000V[6]=1.700000V[7]=1.060000V[8
]=0.550000V[9]=0.190000
Tamaño: 10
Tiempo(seg.):0.000001410
Resultado: v[0]=7.150000 || v[10]=0.190000
```

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**:

a) El valor *chunk* para *static* es 0. Para *dynamic* y *guided* el valor es 1.

Para obtener el valor, he hecho que imprimiese el tipo de *schedule* asignado y el *chunk* correspondiente. Al asignarle *schedule(runtime)*, toma el valor de la variable que el usuario ha asignado. Si el usuario solo asigna el tipo de *schedule* y no el *chunk* este toma el valor por defecto y se imprime.

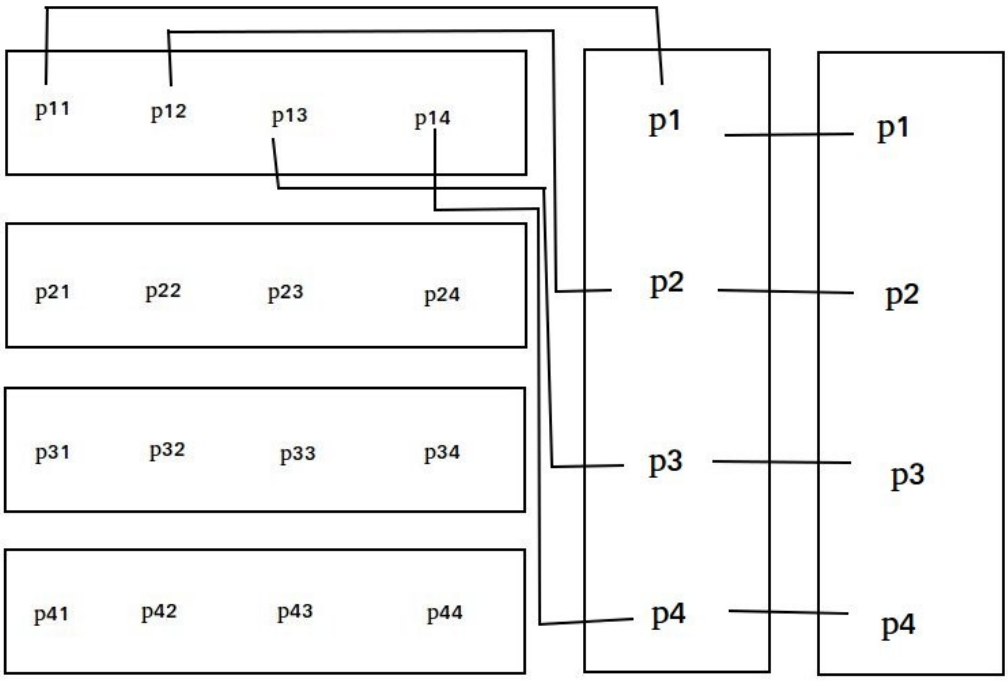
b) El valor por defecto del *chunk* en *static* es 0. Esto produce un reparto de 1792 iteraciones a cada thread. La diferencia con el *chunk* 1 es que en este último, el reparto se produce utilizando *Round Robin*. Para el *chunk* 0, la hebra 0 realiza las iteraciones 0 a 1791. Para *chunk* 1, la hebra 0 realiza la iteración 0, 1792, 3584… Para el *chunk* 64 se realizará un reparto con RR, pero con tamaño 64. Es decir, la hebra 0 realiza las iteraciones 0 a 63, la hebra 1 realiza las iteraciones 64 a 127...

c) Con las asignaciones *dynamic* y *guided*, el reparto de las iteraciones se realiza en tiempo de ejecución. El reparto *dynamic* siempre reparte el mismo *chunk* a la hebra que haya terminado antes. Mientras que el reparto *guided* el *chunk* que reparte irá cambiando a lo largo de la ejecución y se ire haciendo mas pequeño (sin ir a menos que *chunk*).

**CAPTURA CÓDIGO FUENTE**: `pmtv-OpenMP.c`

```
    //Inicializar vectores
#pragma omp parallel private(j)
{
    #pragma omp for schedule(runtime)
      for(i=0; i<N; i++){
        v2[i] = N*0.1-i*0.1;
        v3[i] = 0;
        for (j = i ; j<N; j++){
            v1[i][j] = N*0.1+j*0.1;
        }
      }
}


    clock_gettime(CLOCK_REALTIME,&cgt1);
#pragma omp parallel private(j)
{
    #pragma omp for schedule(runtime)
    for(i = 0; i < N; i++){
        for(j= i; j < N; j++){
            v3[i] = v3[i] + v1[i][j]*v2[j];
        }
    }
}

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

    if(N<15){
        /*printf("V1\n");
        for (i = 0; i < N; i ++){
            for (j = 0; j < N; j++)
                printf("%f ", v1[i][j]);
            printf("\n");
        }
        printf("\nV2\n");
        for (i=0; i<N; i++)
            printf("%f", v2[i]);*/
        printf("\nV3\n");
        for (i=0; i<N; i++)
            printf("V[%d]=%f",i ,v3[i]);
        printf("\n");
    }

    printf("Tamaño: %d\n", N);
    printf("Tiempo(seg.):%11.9f\n",ncgt);
    omp_get_schedule(&k, &c);
    if(k == omp_sched_static)
        printf("Variable run-sched-var: estatica, tamaño chunk: %d\n", c);
    else if(k == omp_sched_dynamic)
        printf("Variable run-sched-var: dinamica, tamaño chunk: %d\n", c);
    else
        printf("Variable run-sched-var: guided, tamaño chunk: %d\n", c);
    printf("Resultado: v[0]=%f || v[%d]=%f\n",v3[0],N,v3[N-1]);

}
```

## DESCOMPOSICIÓN DE DOMINIO:




## CAPTURAS DE PANTALLA:

```
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 201
8-05-09 Wednesday
$./pmtv-OpenMP 11

V3
V[0]=9.460000V[1]=8.250000V[2]=7.050000V[3]=5.880000V[4]=4.760000V[5]=3.710000V[6]=2.750000V[7]=1.900000V[8]=1.180000V[9]=0.610000V[10]=0.210000
Tamaño: 11
Tiempo(seg.):0.000124545
Variable run-sched-var: dinamica, tamaño chunk: 1
Resultado: v[0]=9.460000 || v[11]=0.210000
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-09 Wednesday
$export OMP_SCHEDULE="static, 5"
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-09 Wednesday
$./pmtv-OpenMP 11

V3
V[0]=9.460000V[1]=8.250000V[2]=7.050000V[3]=5.880000V[4]=4.760000V[5]=3.710000V[6]=2.750000V[7]=1.900000V[8]=1.180000V[9]=0.610000V[10]=0.210000
Tamaño: 11
Tiempo(seg.):0.000101721
Variable run-sched-var: estatica, tamaño chunk: 5
Resultado: v[0]=9.460000 || v[11]=0.210000
```

**TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid**

**SCRIPT:** pmvt-OpenMP_PCaula.sh

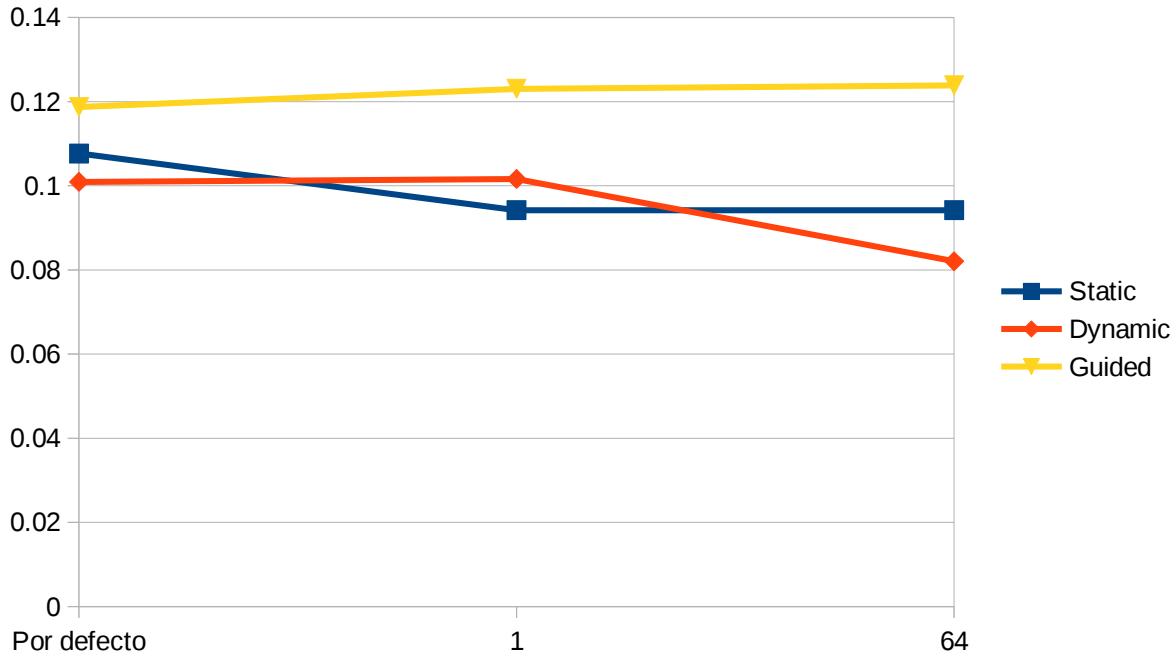
```
#!/bin/bash
export OMP_DYNAMIC="false"
export OMP_SCHEDULE="static"
./pmtv-OpenMP 21504
export OMP_SCHEDULE="static, 1"
./pmtv-OpenMP 21504
export OMP_SCHEDULE="static, 64"
./pmtv-OpenMP 21504

export OMP_SCHEDULE="dynamic"
./pmtv-OpenMP 21504
export OMP_SCHEDULE="dynamic, 1"
./pmtv-OpenMP 21504
export OMP_SCHEDULE="dynamic, 64"
./pmtv-OpenMP 21504

export OMP_SCHEDULE="guided"
./pmtv-OpenMP 21504
export OMP_SCHEDULE="guided, 1"
./pmtv-OpenMP 21504
export OMP_SCHEDULE="guided, 64"
./pmtv-OpenMP 21504
```

**Tabla 3 .**Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= 21504 , 12 threads

| Chunk | Static | Dynamic | Guided |
|---|---|---|---|
| por defecto | | | |
| 1 | | | |
| 64 | | | |

| Chunk | Static | Dynamic | Guided |
|---|---|---|---|
| por defecto | | | |
| 1 | | | |
| 64 | | | |

| Chunk | Static | Dynamic | Guided |
|---|---|---|---|
| **Por defecto** | 0.107660301 | 0.100904226 | 0.118709529 |
| **1** | 0.09420721 | 0.101605135 | 0.12305295 |
| **64** | 0.094207973 | 0.082060297 | 0.12386462 |




| Chunk | Static | Dynamic | Guided |
|---|---|---|---|
| **Por defecto** | 0.094947922 | 0.100784972 | 0.1285511 |
| **1** | 0.090612236 | 0.102142688 | 0.110861663 |
| **64** | 0.093941733 | 0.083325352 | 0.10578094 |

**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`

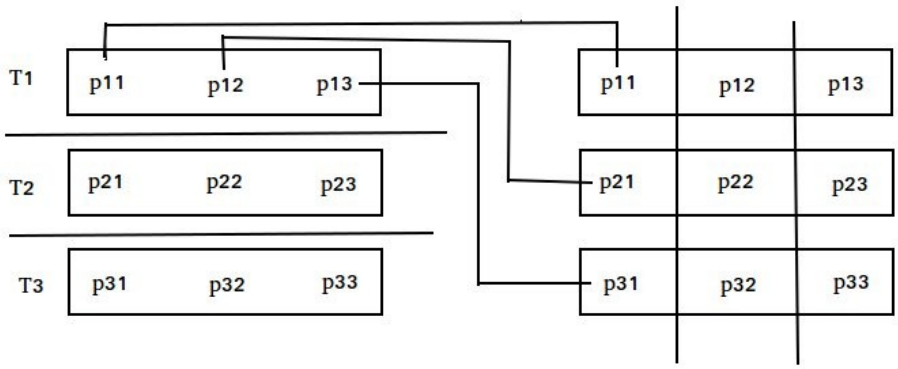
```
#ifdef VECTOR_DYNAMIC
  double **v1 = (double**) malloc(N*sizeof(double*));
  double **v2 = (double**) malloc(N*sizeof(double*));
  double **v3 = (double**) malloc(N*sizeof(double*));
  for(i = 0; i < N; i++){
    v1[i] = (double*) malloc(N*sizeof(double));// malloc necesita el tamano en bytes
    v2[i] = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc devuelve
NULL
    v3[i] = (double*) malloc(N*sizeof(double));
    }
    if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
  }
  #endif
    //Inicializar vectores

      for(i=0; i<N; i++){
        for (j = 0 ; j<N; j++){
            v1[i][j] = N*0.1+j*0.1;
            v2[i][j] = N*0.1-i*0.1;
            v3[i][j] = 0;
        }
      }

double aux = 0;
    clock_gettime(CLOCK_REALTIME,&cgt1);
{
    for(i = 0; i < N; i++){
        for(j= 0; j < N; j++){
            for(k = 0; k < N; k++)
                v3[i][j] += (v1[i][k]*v2[k][j]);
        }
    }
}
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
```

**CAPTURAS DE PANTALLA:**

```
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 2018-05-09 Wednesday
$./pmm-secuencial 3
V1
0.300000 0.400000 0.500000
0.300000 0.400000 0.500000
0.300000 0.400000 0.500000

V2
0.300000 0.300000 0.300000
0.200000 0.200000 0.200000
0.100000 0.100000 0.100000

V3
V[0][0]=0.220000V[0][1]=0.220000V[0][2]=0.220000V[1][0]=0.220000V[1][1]=0.220000V[1][2]=0.220000V[2][0]=0.220000V[2][1]=0.220000V[2][2]=0.220000
Tamaño: 3
Tiempo(seg.):0.000000930
Resultado: v[0][0]=0.220000 || v[2][2]=0.220000
```

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`

```
#pragma omp parallel for private(j) schedule(guided,20)
      for(i=0; i<N; i++){
        for (j = 0 ; j<N; j++){
            v1[i][j] = N*0.1+j*0.1;
            v2[i][j] = N*0.1-i*0.1;
            v3[i][j] = 0;
        }
      }

double aux = 0;
    clock_gettime(CLOCK_REALTIME,&cgt1);

#pragma omp parallel for private(j) schedule(guided,20)
    for(i = 0; i < N; i++){
        for(j= 0; j < N; j++){
            for(k = 0; k < N; k++)
                v3[i][j] += (v1[i][k]*v2[k][j]);
        }
    }
```

**CAPTURAS DE PANTALLA:**

```
[FernandoVallecillosRuiz falconskull@FalconskullNotebook-PC:~/Desktop/ac/S3] 201
8-05-09 Wednesday
$./pmm-OpenMP 3 4
V1
0.300000 0.400000 0.500000
0.300000 0.400000 0.500000
0.300000 0.400000 0.500000

V2
0.300000 0.300000 0.300000
0.200000 0.200000 0.200000
0.100000 0.100000 0.100000

V3
V[0][0]=0.220000V[0][1]=0.220000V[0][2]=0.220000V[1][0]=0.220000V[1][1]=0.220000
V[1][2]=0.220000V[2][0]=0.220000V[2][1]=0.220000V[2][2]=0.220000
Tamaño: 3
Tiempo(seg.):0.000009845
Resultado: v[0][0]=0.220000 || v[2][2]=0.220000
```

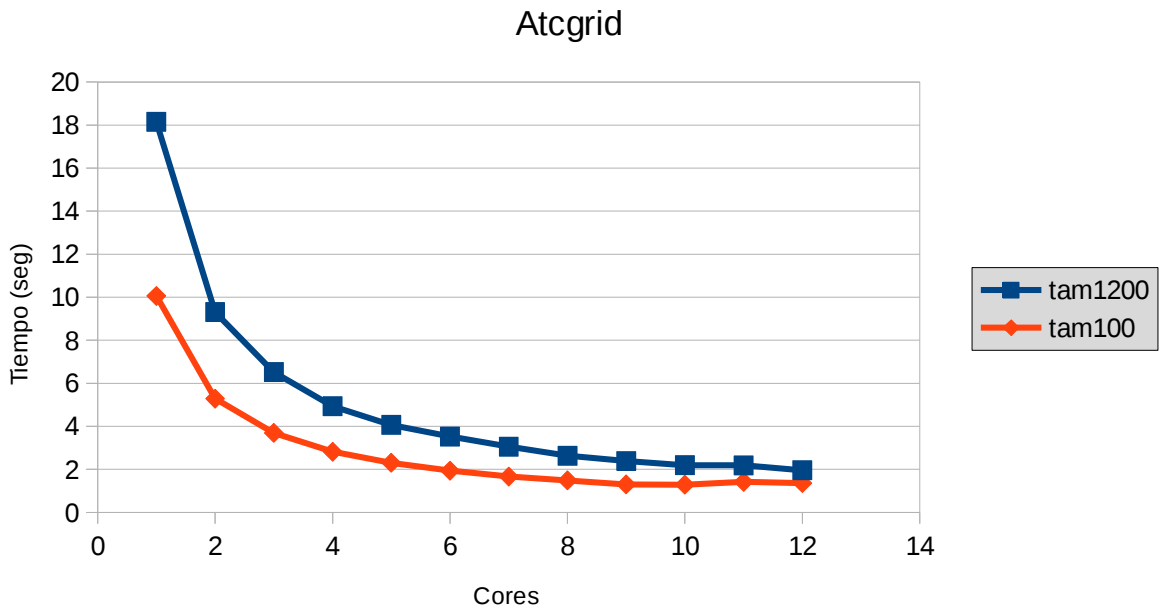
10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos 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.

**SCRIPT:** pmm-OpenMP_atcgrid.sh

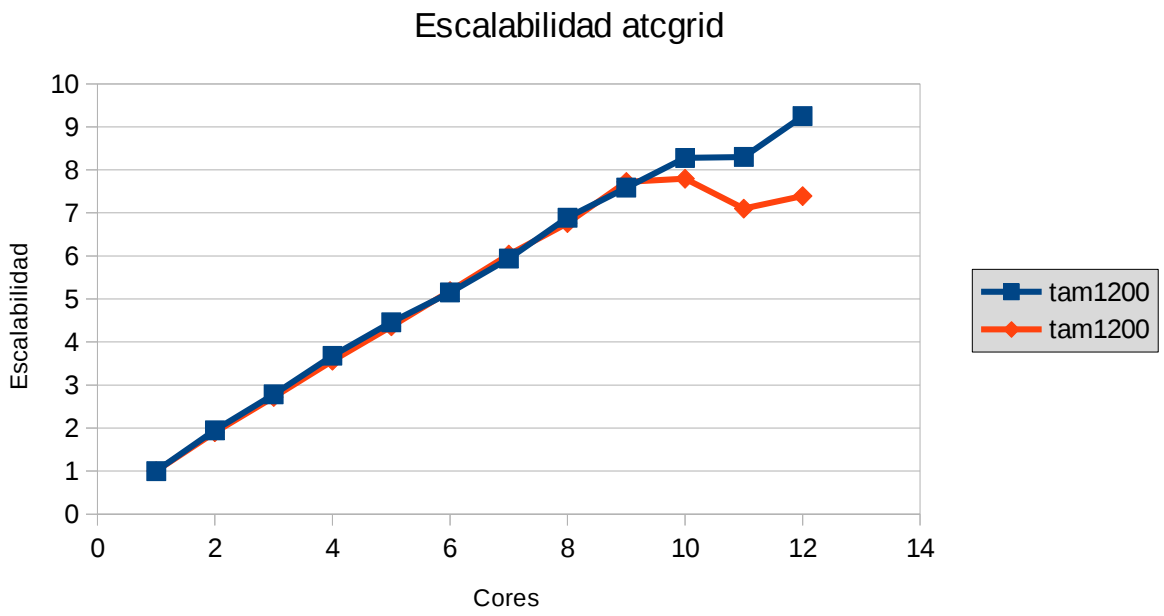
```
#!/bin/bash


for ((N=1;N<13;N=N+1))
do
    ./pmm-OpenMP 1200 $N
done
```

**ESTUDIO DE ESCALABILIDAD EN atcgrid:**

| | Tiempo | |
|---|---|---|
| Cores | 1200 | 1000 |
| 1 | 18.147089392 | 10.058921122 |
| 2 | 9.309682185 | 5.2859488303 |
| 3 | 6.520895633 | 3.6964816954 |
| 4 | 4.931913759 | 2.8231832065 |
| 5 | 4.068912435 | 2.3067855136 |
| 6 | 3.523082024 | 1.9401685757 |
| 7 | 3.056590049 | 1.6677829418 |
| 8 | 2.634881724 | 1.4874862219 |
| 9 | 2.391656839 | 1.3033301981 |
| 10 | 2.192147053 | 1.2906589511 |
| 11 | 2.186671723 | 1.4169954631 |
| 12 | 1.962114004 | 1.361213031 |



| | Escalabilidad | |
|---|---|---|
| Cores | 1200 | 1000 |
| 1 | 1 | 1 |
| 2 | 1.9492705585 | 1.9029546908 |
| 3 | 2.7829136385 | 2.7212149149 |
| 4 | 3.6795228544 | 3.5629714356 |
| 5 | 4.4599360841 | 4.3605792836 |
| 6 | 5.150913112 | 5.184560377 |
| 7 | 5.9370373852 | 6.0313131104 |
| 8 | 6.8872500905 | 6.7623625511 |
| 9 | 7.587664374 | 7.7178608589 |
| 10 | 8.278226302 | 7.7936321703 |
| 11 | 8.2989546172 | 7.0987673454 |
| 12 | 9.248743628 | 7.389674425 |

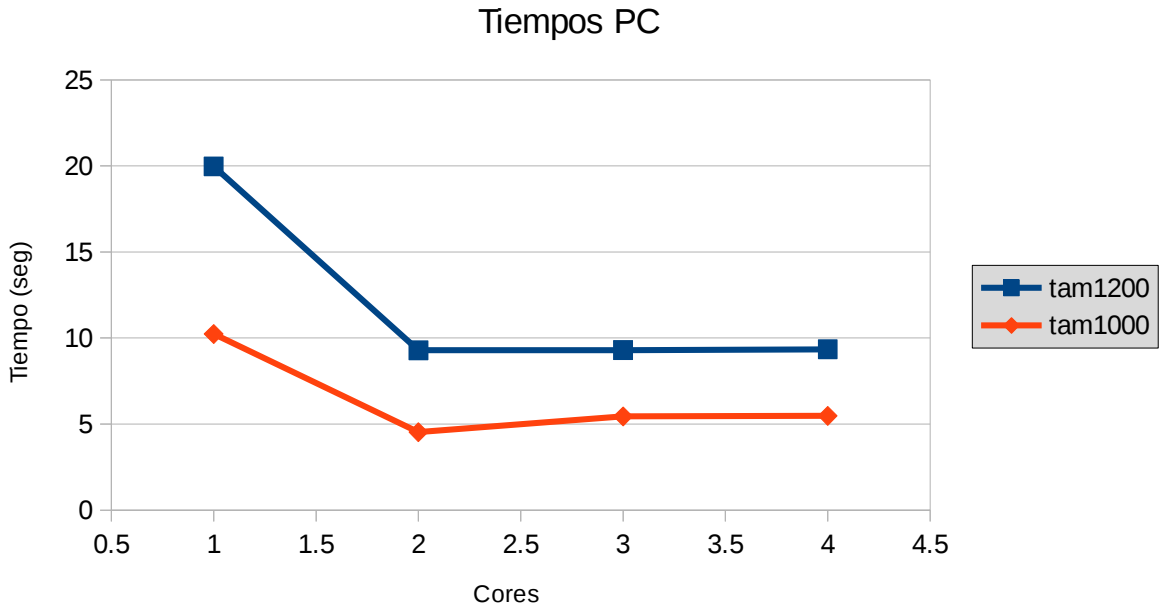



**SCRIPT:** pmm-OpenMP_pclocal.sh

```
#!/bin/bash


for ((N=1;N<5;N=N+1))
do
    ./pmm-OpenMP 1000 $N
done
```

**ESTUDIO DE ESCALABILIDAD EN PCLOCAL:**

| | Tiempo | |
|---|---|---|
| Cores | 1200 | 1000 |
| 1 | 19.967902692 | 10.231332165 |
| 2 | 9.291343974 | 4.531372872 |
| 3 | 9.301718267 | 5.447748683 |
| 4 | 9.344617954 | 5.473751572 |




| | Escalabilidad | |
|---|---|---|
| Cores | 1200 | 1000 |
| 1 | 1 | 1 |
| 2 | 2.1490865851 | 2.2578879413 |
| 3 | 2.1466896888 | 1.8780844639 |