Cuaderno de prácticas
de Arquitectura de Computadores
Grado en Ingeniería Informática

Memoria
Bloque Práctico 3

Alumno: Manuel Jesús García Manday
DNI: 48893432-D
Grupo: *D3*

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.

CÓDIGO FUENTE: `if-clauseModificado.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char **argv)
{
		int i, n=20, tid, num;
		int a[n],suma=0,sumalocal;

		if(argc < 3) {
			fprintf(stderr,"[ERROR]-Falta iteraciones o numero de hebras\n");
			exit(-1);
		}

		n = atoi(argv[1]);
		if (n>20)
		    n=20;
		num = atoi(argv[2]);

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

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

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3 $ ./if-clauseModificado 2 5
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=1
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3 $ ./if-clauseModificado 7 5
 thread 2 suma de a[4]=4 sumalocal=4
 thread 2 suma de a[5]=5 sumalocal=9
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 3 suma de a[6]=6 sumalocal=6
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=21
```

RESPUESTA: En la primera captura vemos como sólo trabaja una hebra, esto es debido a que no se ha cumplido la condición de la clausula if, por lo que no se ejecuta la región paralela. Sin embargo, en la segunda captura si se ha cumplido la condición de la clausula if y se pasa a ejecutar la región paralela como podemos ver en la captura.

2. **(a)** Rellenar la tabla (se debe poner en la tabla el *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 `schedule`. En la segunda fila, 1, 2 4 representan el tamaño del chunk

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

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

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

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?

CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

main(int argc, char **argv) {
     int i, n = 16, chunk, a[n], suma = 0, modificador;
      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
      {
           #pragma omp master
            {
                printf("Dentro del parallel dyn-var: %d
\n",omp_get_dynamic());
                printf("Dentro del parallel nthreads-var: %d
\n",omp_get_num_threads());
                printf("Dentro del parallel thread-limit-var: %d
\n",omp_get_max_threads());
                omp_get_schedule(&kind,&modificador);
                printf("Dentro del parallel run-sched-var: %d \n",
kind);
            }
```

```
            #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);
            }
      }

      printf("Fuera de 'parallel for' suma=%d\n",suma);
      printf("Despues del parallel dyn-var: %d \n",omp_get_dynamic());
      printf("Despues del parallel nthreads-var: %d
\n",omp_get_num_threads());
      printf("Despues del parallel thread-limit-var: %d
\n",omp_get_max_threads());
      omp_get_schedule(&kind,&modificador);
      printf("Despues del parallel run-sched-var: %d \n", kind);
}
```

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./scheduled-clauseModificado 10 3
Dentro del parallel dyn-var: 0
Dentro del parallel nthreads-var: 4
Dentro del parallel thread-limit-var: 4
Dentro del parallel run-sched-var: 2
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 2 suma a[9]=9 suma=9
 thread 1 suma a[3]=3 suma=3
 thread 1 suma a[4]=4 suma=7
 thread 1 suma a[5]=5 suma=12
 thread 3 suma a[6]=6 suma=6
 thread 3 suma a[7]=7 suma=13
 thread 3 suma a[8]=8 suma=21
Fuera de 'parallel for' suma=9
Despues del parallel dyn-var: 0
Despues del parallel nthreads-var: 1
Despues del parallel thread-limit-var: 4
Despues del parallel run-sched-var: 2
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ export OMP_SCHEDULE="static"
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./scheduled-clauseModificado 10 3
Dentro del parallel dyn-var: 0
Dentro del parallel nthreads-var: 4
Dentro del parallel thread-limit-var: 4
Dentro del parallel run-sched-var: 1
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 3 suma a[3]=3 suma=3
 thread 3 suma a[4]=4 suma=7
 thread 3 suma a[5]=5 suma=12
 thread 1 suma a[6]=6 suma=6
 thread 1 suma a[7]=7 suma=13
 thread 1 suma a[8]=8 suma=21
 thread 2 suma a[9]=9 suma=9
Fuera de 'parallel for' suma=9
Despues del parallel dyn-var: 0
Despues del parallel nthreads-var: 1
Despues del parallel thread-limit-var: 4
Despues del parallel run-sched-var: 1
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ export OMP_SCHEDULE="guided"
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./scheduled-clauseModificado 10 3
Dentro del parallel dyn-var: 0
Dentro del parallel nthreads-var: 4
Dentro del parallel thread-limit-var: 4
Dentro del parallel run-sched-var: 3
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 1 suma a[9]=9 suma=9
 thread 3 suma a[3]=3 suma=3
 thread 3 suma a[4]=4 suma=7
 thread 3 suma a[5]=5 suma=12
 thread 2 suma a[6]=6 suma=6
 thread 2 suma a[7]=7 suma=13
 thread 2 suma a[8]=8 suma=21
Fuera de 'parallel for' suma=9
Despues del parallel dyn-var: 0
Despues del parallel nthreads-var: 1
Despues del parallel thread-limit-var: 4
Despues del parallel run-sched-var: 3
```

RESPUESTA:

- dyn-var: muestra el mismo valor ya que por defecto esta a 0, y sino es modificado se mantiene tanto dentro como fuera de la región paralela.

- nthreads-var: muestra valores distintos ya que nos indica el número de threads a ejecutar en la región paralela, dentro de la misma nos da 4 porque son las que se estan ejecutando, y fuera nos da 1 porque ya solo tenemos una hebra y se esta ejecutanto el código en secuencial.

- threads-limit-var: muestra el mismo valor, ya que no depende de si esta dentro o fuera de una región paralela, ya que nos indica el número máximo de threads para todo el programa, que coincide con el número de procesadores.

-run-sched-var: muestra el mismo valor, ya que el valor del planificador va a ser el mismo tanto dentro como fuera de la región paralela (siempre y cuando no haya sido modificado).

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.

CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

main(int argc, char **argv) {
       int i, n = 16, chunk, a[n], suma = 0;

       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
        {
            #pragma omp master
            {
                 printf("Dentro del parallel omp_get_num_threads():
%d \n", omp_get_num_threads());
                 printf("Dentro del parallel omp_get_num_procs(): %d
\n", omp_get_num_procs());
                 printf("Dentro del parallel omp_in_parallel(): %d \n",
omp_in_parallel());
            }

            #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);
            }
      }

      printf("Fuera de 'parallel for' suma=%d\n",suma);
      printf("Fuera del parallel omp_get_num_threads(): %d \n",
omp_get_num_threads());
      printf("Fuera del parallel omp_get_num_procs(): %d \n",
omp_get_num_procs());
      printf("Fuera del parallel omp_in_parallel(): %d \n",
omp_in_parallel());

}
```

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1W6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./scheduled-clau
seModificado4 7 2
Dentro del parallel omp_get_num_threads(): 4
Dentro del parallel omp_get_num_procs(): 4
Dentro del parallel omp_in_parallel(): 1
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 2 suma a[2]=2 suma=2
 thread 2 suma a[3]=3 suma=5
 thread 1 suma a[4]=4 suma=4
 thread 1 suma a[5]=5 suma=9
 thread 3 suma a[6]=6 suma=6
Fuera de 'parallel for' suma=6
Fuera del parallel omp_get_num_threads(): 1
Fuera del parallel omp_get_num_procs(): 4
Fuera del parallel omp_in_parallel(): 0
```

RESPUESTA:

- omp_num_threads(): esta función obtiene valores distintos debido a que devuelve el número de threads que se está usando dentro de una región paralela, vemos como dentro de la región nos devuelve un 4, debido a los 4 cores que tiene la máquina, mientras que fuera de la región paralela ya comienza a ser código secuencial, por lo que el valor devuelto es 1, ya que solo existe una hebra ejecutando ese código.

- omp_get_num_procs(): esta función obtiene el mismo valor indistintamente de donde se invoque, ya que lo que devuelve es el número de procesadores que hay disponibles para el programa.

- omp_in_parallel(): esta función obtiene valores distintos ya que depende del ámbito desde el que se invoque, devuelve 1 cuando esta dentro de una región paralela, y 0 cuando se encuentra fuera de ella como podemos ver en la captura de pantalla.

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.

CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

main(int argc, char **argv) {
     int i, n = 16, chunk, suma = 0, j = 0, modificador;
      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;

      int a[n];
      chunk = atoi(argv[2]);
```

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

      printf("Antes de la modificacion dyn-var: %d
\n",omp_get_dynamic());
      printf("Antes de la modificacion nthreads-var: %d
\n",omp_get_num_threads());
      omp_get_schedule(&kind,&modificador);
      printf("Antes de la modificacion run-sched-var: %d \n",kind);
      omp_set_dynamic(0);
      omp_set_num_threads(2);
      omp_set_schedule(1,modificador);

      #pragma omp parallel
      {
            #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);
            }

            #pragma omp master
            {
                  printf("Despues de la modificacion dyn-var: %d
\n",omp_get_dynamic());
                  printf("Despues de la modificacion nthreads-var:
%d \n",omp_get_num_threads());
                  omp_get_schedule(&kind,&modificador);
                  printf("Despues de la modificacion run-sched-var:
%d \n",kind);
            }
      }
     printf("Fuera de 'parallel for' suma=%d\n",suma);

}
```

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./scheduled-clauseModificado5 10 3
Antes de la modificacion dyn-var: 0
Antes de la modificacion nthreads-var: 1
Antes de la modificacion run-sched-var: 3
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 0 suma a[2]=2 suma=3
 thread 0 suma a[6]=6 suma=9
 thread 0 suma a[7]=7 suma=16
 thread 0 suma a[8]=8 suma=24
 thread 0 suma a[9]=9 suma=33
 thread 1 suma a[3]=3 suma=3
 thread 1 suma a[4]=4 suma=7
 thread 1 suma a[5]=5 suma=12
Despues de la modificacion dyn-var: 0
Despues de la modificacion nthreads-var: 2
Despues de la modificacion run-sched-var: 1
Fuera de 'parallel for' suma=33
```

RESPUESTA:

- dyn-var: muestra el mismo valor porque al modificarla se le ha asignado el valor que tiene por defecto (0).

- nthreads-var: muestra valores distintos, ya que antes de la modificación su valor era 1 debido a que estaba ejecutando código secuencial, mientras que al modificarlo a 2, es el número de hebras que han pasado a ejecutar la región paralela.

- run-sched-var: muestra valores distintos, ya que antes de la modificación mostraba el valor que tenia puesto (guided) mientras que despues de modificarlo a static, vemos como muestra que se ha modificado correctamente.

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un 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 las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CÓDIGO FUENTE: `pmtv-secuencial.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char **argv) {

     int n, i, j, k;
     struct timespec cgt1,cgt2;
     double ncgt;
     n = atoi(argv[1]); /* tomamos el tamaño de filas y columnas de la
matriz */
     int **m1, **m2, **m3; /* declaramos dinamicas las matrices */

     /* Hacemos la reserva de memoria dinamica */
     m1 = (int **) malloc(n * sizeof(int*));
     m2 = (int **) malloc(n * sizeof(int*));
     m3 = (int **) malloc(n * sizeof(int*));

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

```
      if ((m1 == NULL) || (m2 == NULL) || (m3 == NULL)){
          printf("\nError en la reserva de memoria.");
          exit(-1);
      }

     srand(time(NULL));
      /* inicializamos las matrices */
      for(i = 0; i < n; i++){
            for(j = i; j < n; j++){
                m1[i][j] = ((rand() % 4)+1);
                m2[i][j] = ((rand() % 4)+1);
                m3[i][j] = 0;
            }
      }

      /* realizamos la multiplicacion */
      for(i = 0; i < n; i++){
            for(j = 0; j < n; j++)
                  m3[i][j] += m1[i][j] * m2[i][j];
      }

      printf("Valor (0,0): %d \n", m3[0][0]);
      printf("Valor (n-1,n-1): %d \n", m3[n-1][n-1]);
      printf("\n");

}
```

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./pmtv-secuencial 4
Valor (0,0): 4
Valor (n-1,n-1): 2
```

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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. 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 que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 2, 64, 128, 1024 y el chunk por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096 componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 1 .con los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

CÓDIGO FUENTE: `pmtv-OpenMP.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char **argv) {
     int n, i, j, k, num, chunk;
      struct timespec cgt1,cgt2,cgt3,cgt4,cgt5,cgt6;
      double ncgt;
      n = atoi(argv[1]); /* tomamos el tamaño de filas y columnas de
la matriz */
      int *v1, *v3, **m; /* declaramos dinamicas los vectores y la
matriz */
      chunk = atoi(argv[2]); /* para el numero de iteraciones */

      /* Hacemos la reserva de memoria dinamica */
      v1 = (int *) malloc(n * sizeof(int));
      v3 = (int *) malloc(n * sizeof(int));
      m = (int **) malloc(n * sizeof(int*));

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

      if ((v1 == NULL) || (v3 == NULL) || (m == NULL)){
            printf("\nError en la reserva de memoria.");
            exit(-1);
      }
```

```
      srand(time(NULL));
      /* inicializamos la matriz y el vector */
      for(i = 0; i < n; i++){
            v1[i] = (rand() % 3) + 1;
            v3[i] = 0;
            for(j = 0; j < n; j++){
                  if(i == 0)
                      m[i][j] = ((rand() % 4)+1);
                  else{
                      while(i != j){
                        m[i][j] = 0;
                        j++;
                      }

                      while(j < n){
                        m[i][j] = ((rand() % 4)+1);
                        j++;
                      }
                  }
            }
       }

       /* Para que el numero de threads coincida con el número de
cores */
       num = omp_get_num_procs();


       /* realizamos la multiplicacion con planificador static*/
       clock_gettime(CLOCK_REALTIME,&cgt1);
       #pragma omp parallel for shared(m,v1,v3) private(i,j)
num_threads(num) schedule(static,chunk)
       for(i = 0; i < n; i++)
            for(j = i; j < n; j++)
                        v3[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));

       printf("Planificador static: \n");
       printf("Tiempo(seg.):%11.9f\t / Tamaño Vector:%u\n",ncgt,n);
       printf("\n\n");

      /* realizamos la multiplicacion con planificador dynamic*/
       ncgt = 0.0;
       clock_gettime(CLOCK_REALTIME,&cgt3);
       #pragma omp parallel for shared(m,v1,v3) private(i,j)
num_threads(num) schedule(dynamic,chunk)
       for(i = 0; i < n; i++)
            for(j = i; j < n; j++)
                        v3[i] += m[i][j] * v1[j];
```

```
        clock_gettime(CLOCK_REALTIME,&cgt4);
        ncgt=(double) (cgt4.tv_sec-cgt3.tv_sec)+
            (double) ((cgt4.tv_nsec-cgt3.tv_nsec)/(1.e+9));

        printf("Planificador dynamic: \n");
        printf("Tiempo(seg.):%11.9f\t / Tamaño Vector:%u\n",ncgt,n);
        printf("\n\n");

        /* realizamos la multiplicacion con planificador guided*/
        ncgt = 0.0;
        clock_gettime(CLOCK_REALTIME,&cgt5);
        #pragma omp parallel for shared(m,v1,v3) private(i,j)
num_threads(num) schedule(guided,chunk)
        for(i = 0; i < n; i++)
            for(j = i; j < n; j++)
                        v3[i] += m[i][j] * v1[j];

         clock_gettime(CLOCK_REALTIME,&cgt6);
         ncgt=(double) (cgt6.tv_sec-cgt5.tv_sec)+
            (double) ((cgt6.tv_nsec-cgt5.tv_nsec)/(1.e+9));

         printf("Planificador guided: \n");
         printf("Tiempo(seg.):%11.9f\t / Tamaño Vector:%u\n",ncgt,n);
         printf("\n\nValor (0,0): %d \n", m[0][0]);
         printf("Valor (n-1,n-1): %d \n", m[n-1][n-1]);
         printf("\n\n");

}
```

CAPTURAS DE PANTALLA:

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

pclocal

Tabla 1 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 4 threads	Dynamic 4 threads	Guided 4 threads
por defecto	0.023631925	0.023041841	0.026898222
2	0.021905677	0.023500586	0.027061117
32	0.028020921	0.022766204	0.027409401
64	0.021501050	0.021672430	0.027837005
2048	0.022492991	0.026412864	0.033573352

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./pmtv-OpenMP 8192 1
Planificador static:
Tiempo(seg.):0.023631925         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.023041841         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.026898222         / Tamaño Vector:8192


Valor (0,0): 3
Valor (n-1,n-1): 3
```

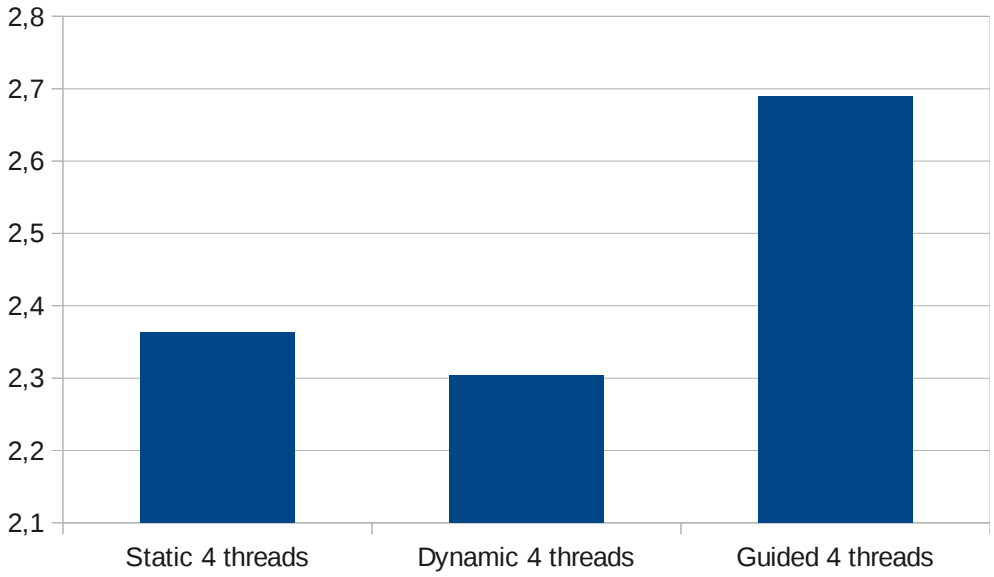
```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./pmtv-OpenMP 8192 2
Planificador static:
Tiempo(seg.):0.021905677         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.023500586         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.027061117         / Tamaño Vector:8192


Valor (0,0): 2
Valor (n-1,n-1): 4
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./pmtv-OpenMP 8192 32
Planificador static:
Tiempo(seg.):0.028020921         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.022766204         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.027409401         / Tamaño Vector:8192


Valor (0,0): 4
Valor (n-1,n-1): 4
```

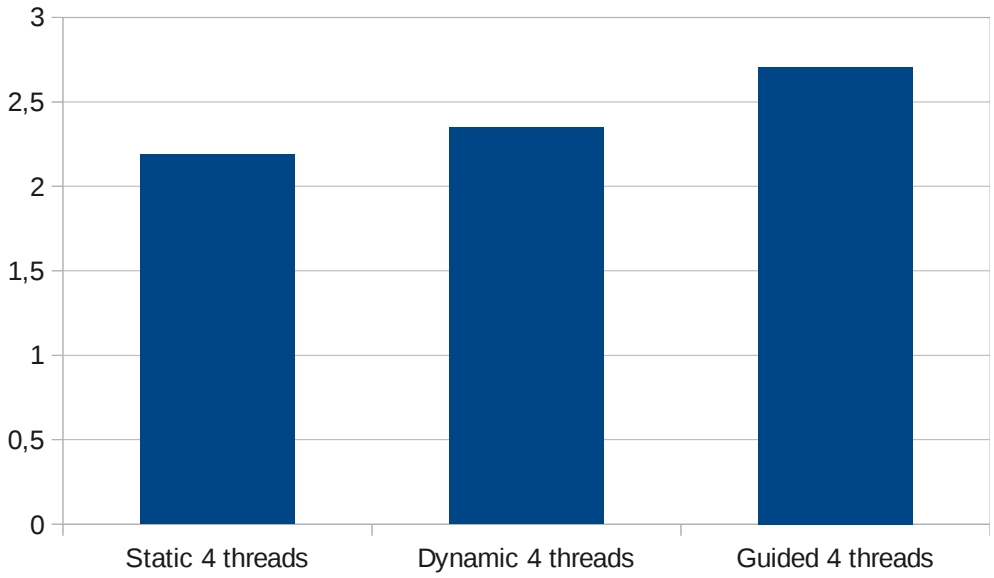
```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./pmtv-OpenMP 8192 64
Planificador static:
Tiempo(seg.):0.021501050         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.021672430         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.027837005         / Tamaño Vector:8192


Valor (0,0): 4
Valor (n-1,n-1): 3
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario3/BloquePractico3_GarciaMandayManuelJesus $ ./pmtv-OpenMP 8192 2048
Planificador static:
Tiempo(seg.):0.022492991         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.026412864         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.033573352         / Tamaño Vector:8192


Valor (0,0): 2
Valor (n-1,n-1): 2
```

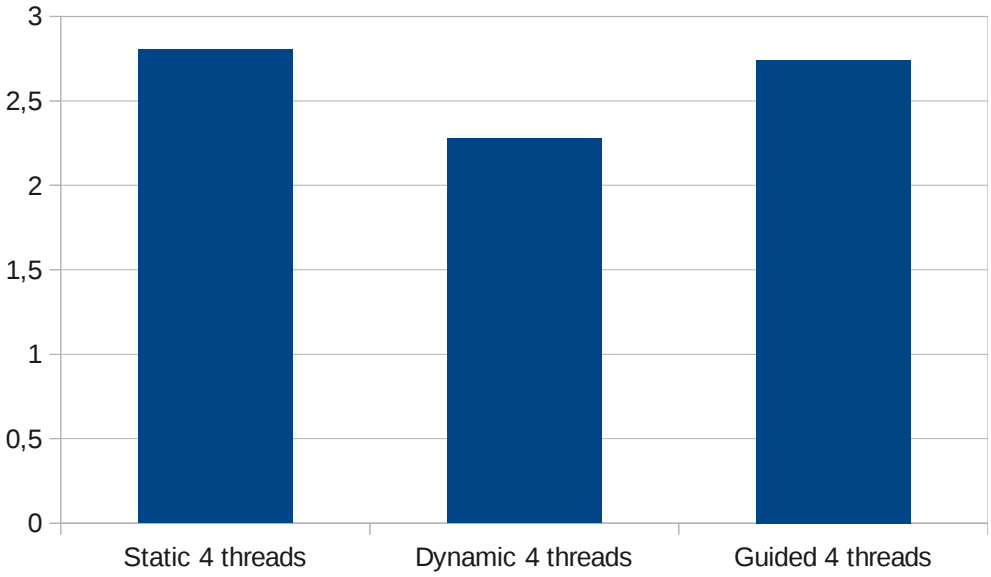
- chunk por defecto



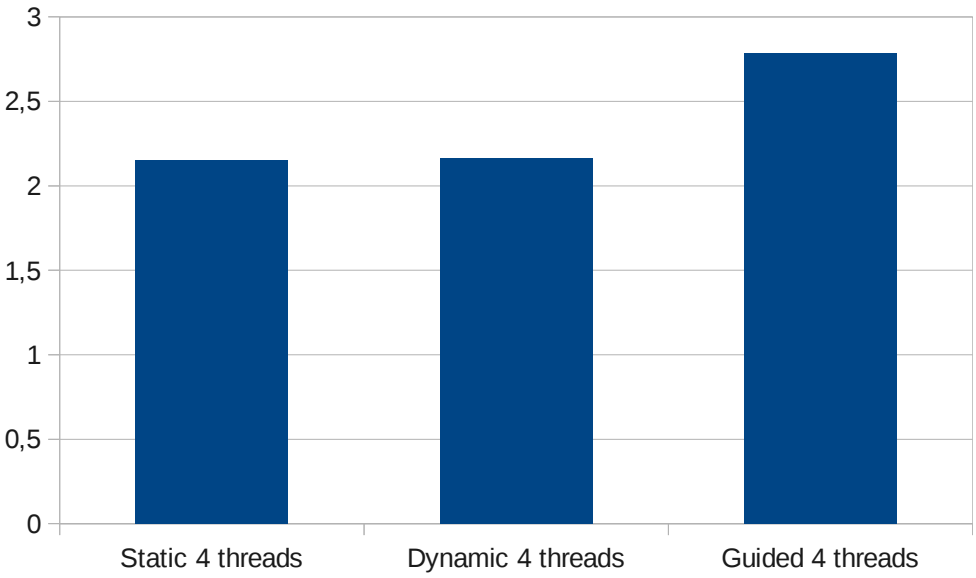
- chunk = 2



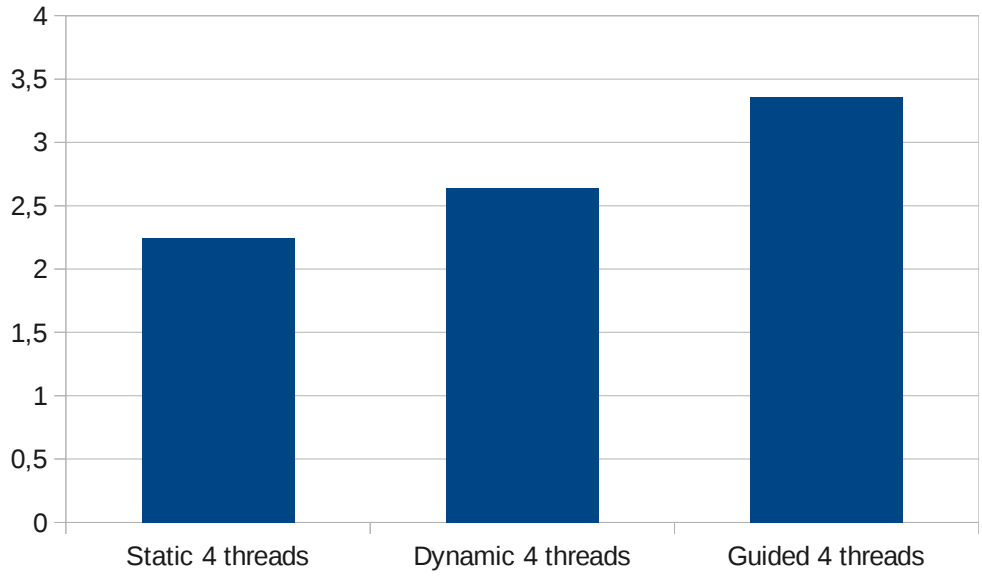
- chunk = 32



- chunk = 64



- chunk = 2048



Gráficas con escala 1:100

atcgrid

Tabla 2 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
por defecto	0.013380559	0.011702854	0.010363755
2	0.012776896	0.012336302	0.011981857
32	0.012623488	0.009613442	0.011695844
64	0.012344271	0.009989740	0.010580940
2048	0.026781251	0.038622795	0.026076317

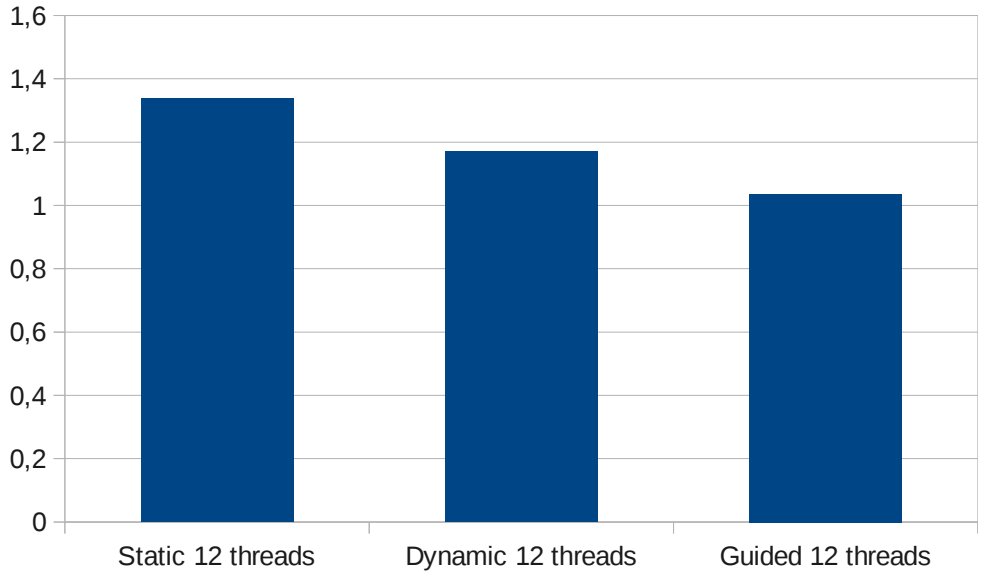
```
[D3estudiante5@atcgrid hello]$ echo 'hello/pmtv-OpenMP 8192 1' | qsub -q ac
18930.atcgrid
[D3estudiante5@atcgrid hello]$ cat STDIN.o18930
Planificador static:
Tiempo(seg.):0.013380559         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.011702854         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.010363755         / Tamaño Vector:8192


Valor (0,0): 2
Valor (n-1,n-1): 1



[D3estudiante5@atcgrid hello]$ echo 'hello/pmtv-OpenMP 8192 2' | qsub -q ac
18931.atcgrid
[D3estudiante5@atcgrid hello]$ cat STDIN.o18931
Planificador static:
Tiempo(seg.):0.012776896         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.012336302         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.011981857         / Tamaño Vector:8192


Valor (0,0): 2
Valor (n-1,n-1): 1
```

```
[D3estudiante5@atcgrid hello]$ echo 'hello/pmtv-OpenMP 8192 32' | qsub -q ac
18932.atcgrid
[D3estudiante5@atcgrid hello]$ cat STDIN.o18932
Planificador static:
Tiempo(seg.):0.012623488         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.009613442         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.011695844         / Tamaño Vector:8192


Valor (0,0): 2
Valor (n-1,n-1): 1
```

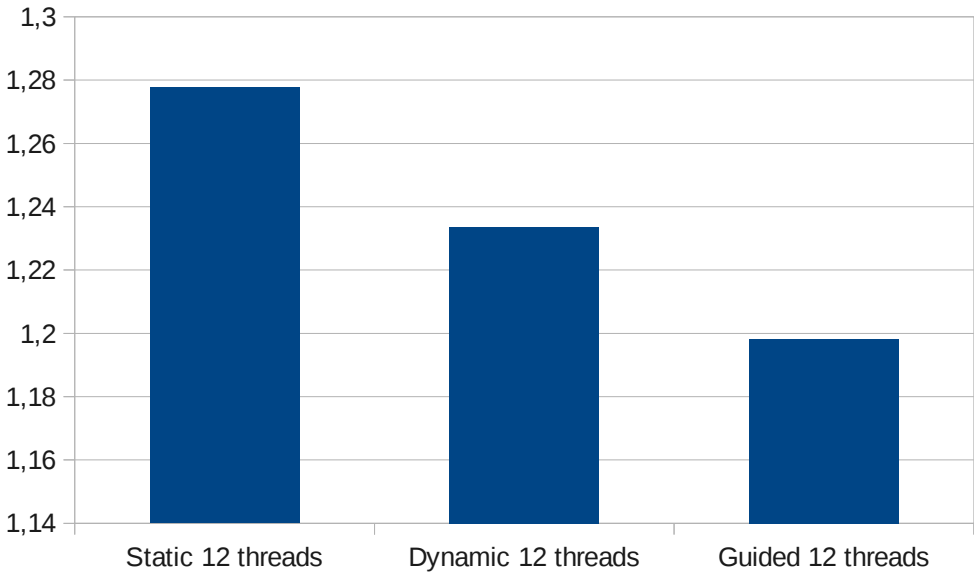
```
[D3estudiante5@atcgrid hello]$ echo 'hello/pmtv-OpenMP 8192 64' | qsub -q ac
18933.atcgrid
[D3estudiante5@atcgrid hello]$ cat STDIN.o18933
Planificador static:
Tiempo(seg.):0.012344271         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.009989740         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.010580940         / Tamaño Vector:8192


Valor (0,0): 4
Valor (n-1,n-1): 1
```

```
[D3estudiante5@atcgrid hello]$ echo 'hello/pmtv-OpenMP 8192 2048' | qsub -q ac
18934.atcgrid
[D3estudiante5@atcgrid hello]$ cat STDIN.o18934
Planificador static:
Tiempo(seg.):0.026781251         / Tamaño Vector:8192


Planificador dynamic:
Tiempo(seg.):0.038622795         / Tamaño Vector:8192


Planificador guided:
Tiempo(seg.):0.026076317         / Tamaño Vector:8192


Valor (0,0): 1
Valor (n-1,n-1): 4
```

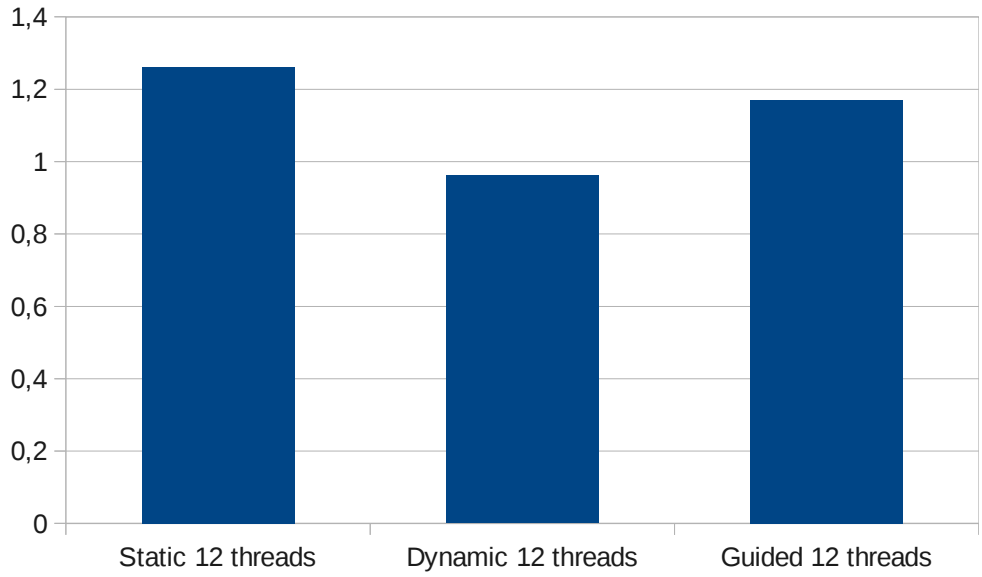
- chunk por defecto



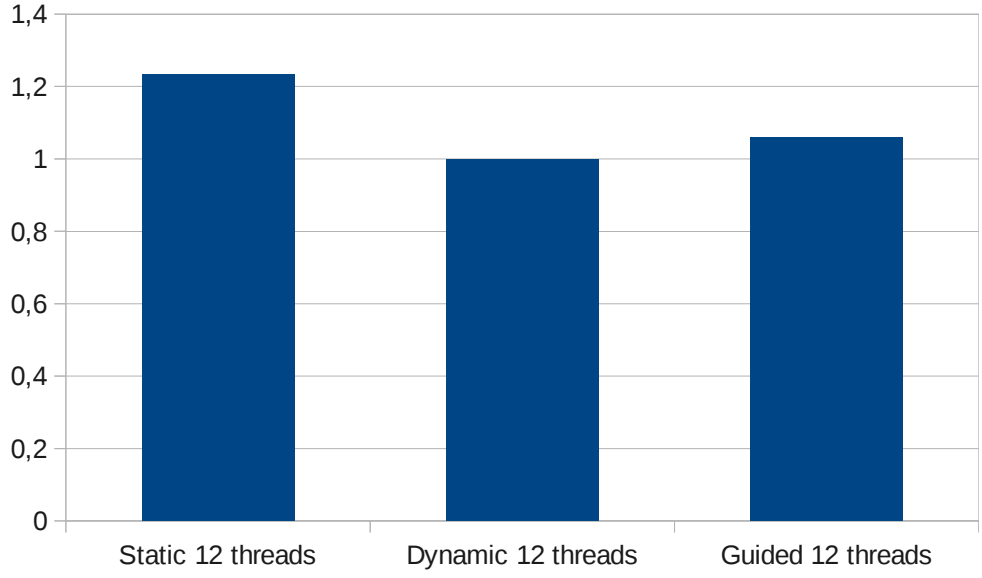
- chunk = 2



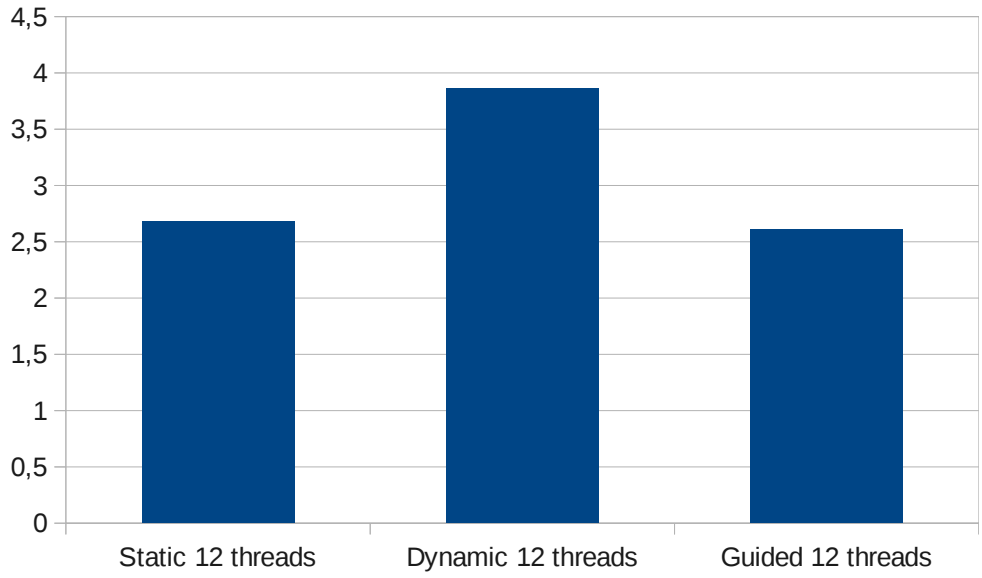
- chunk = 32



- chunk = 64



- chunk = 2048



Gráficas con escala 1:100

Según que tipo chunk y que cantidad de threads podemos decir cual nos puede interesar mas, si es un chunk pequeño y un número de threads mayor, la planificación guided es la mas idónea, como podemos ver en los ejemplos sobre atcgrid, pero en circunstancias contrarias, es decir, con un chunk mayor y un número de threads menor los tiempos se disparan con este tipo de planificador, siendo mejor un planificador static o dynamic, sabiendo que el guided tiene un funcionamiento similiar al dynamic.

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

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

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.

CÓDIGO FUENTE: `pmm-secuencial.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char **argv) {
      int n, i, j, k;
       struct timespec cgt1,cgt2;
       double ncgt;
       n = atoi(argv[1]); /* tomamos el tamaño de filas y columnas de
la matriz */
       int **m1, **m2, **m3; /* declaramos dinamicas las matrices */

       /* Hacemos la reserva de memoria dinamica */
       m1 = (int **) malloc(n * sizeof(int*));
       m2 = (int **) malloc(n * sizeof(int*));
       m3 = (int **) malloc(n * sizeof(int*));

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

       if ((m1 == NULL) || (m2 == NULL) || (m3 == NULL)){
            printf("\nError en la reserva de memoria.");
            exit(-1);
       }

       srand(time(NULL));
       /* inicializamos las matrices */
       for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                 m1[i][j] = ((rand() % 4)+1);
                 m2[i][j] = ((rand() % 4)+1);
                 m3[i][j] = 0;
            }
       }
```

```
        /* realizamos la multiplicacion */
        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++)
                m3[i][j] += m1[i][j] * m2[i][j];
        }

        printf("Valor (0,0): %d \n", m3[0][0]);
        printf("Valor (n-1,n-1): %d \n", m3[n-1][n-1]);
        printf("\n");
}
```

CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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 utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2).

CÓDIGO FUENTE: `pmm-OpenMP.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char **argv) {
      int n, i, j, k, nump;
       struct timespec cgt1,cgt2;
       double ncgt;
       n = atoi(argv[1]); /* tomamos el tamaño de filas y columnas de
la matriz */
       int **m1, **m2, **m3; /* declaramos dinamicas las matrices */

       /* Hacemos la reserva de memoria dinamica */
       m1 = (int **) malloc(n * sizeof(int*));
       m2 = (int **) malloc(n * sizeof(int*));
       m3 = (int **) malloc(n * sizeof(int*));

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

```
        if ((m1 == NULL) || (m2 == NULL) || (m3 == NULL)){
            printf("\nError en la reserva de memoria.");
            exit(-1);
        }

        /* obtenemos el numero de procesadores para obligar a que se
use el maximo número de threads posible */
       nump = omp_get_num_procs();

      /* inicializamos las matrices */
      clock_gettime(CLOCK_REALTIME,&cgt1);
      #pragma omp parallel for shared(m1,m2,m3) private(i,j)
num_threads(nump) schedule(guided)
       for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                  m1[i][j] = ((rand() % 4)+1);
                  m2[i][j] = ((rand() % 4)+1);
                  m3[i][j] = 0;
            }
       }
       clock_gettime(CLOCK_REALTIME,&cgt2);
       ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
            (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

       printf("Tiempo de inicializacion(seg.):%11.9f\t / Tamaño
Vector:%u\n",ncgt,n);
       printf("\n");

       /* realizamos la multiplicacion */
       clock_gettime(CLOCK_REALTIME,&cgt1);
       #pragma omp parallel for shared(m1,m2,m3) private(i,j)
num_threads(nump) schedule(guided)
       for(i = 0; i < n; i++){
            for(j = 0; j < n; j++)
                  m3[i][j] += m1[i][j] * m2[i][j];
       }
       clock_gettime(CLOCK_REALTIME,&cgt2);
       ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
            (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

       printf("Tiempo de multiplicacion(seg.):%11.9f\t / Tamaño
Vector:%u\n",ncgt,n);
       printf("\n\n\n");

       printf("Valor (0,0): %d \n", m3[0][0]);
       printf("Valor (n-1,n-1): %d \n", m3[n-1][n-1]);
       printf("\n");

}

/* Comentar que tanto en la paralelización de la inicialización y de
la multiplicación el planificador que lo lleva a cabo es de tipo
guided, ya que al no conocer el número de iteraciones  es el que mejor
tiempo de ejecución puede dar. Tiene menos sobrecarga que dynamic. */
```

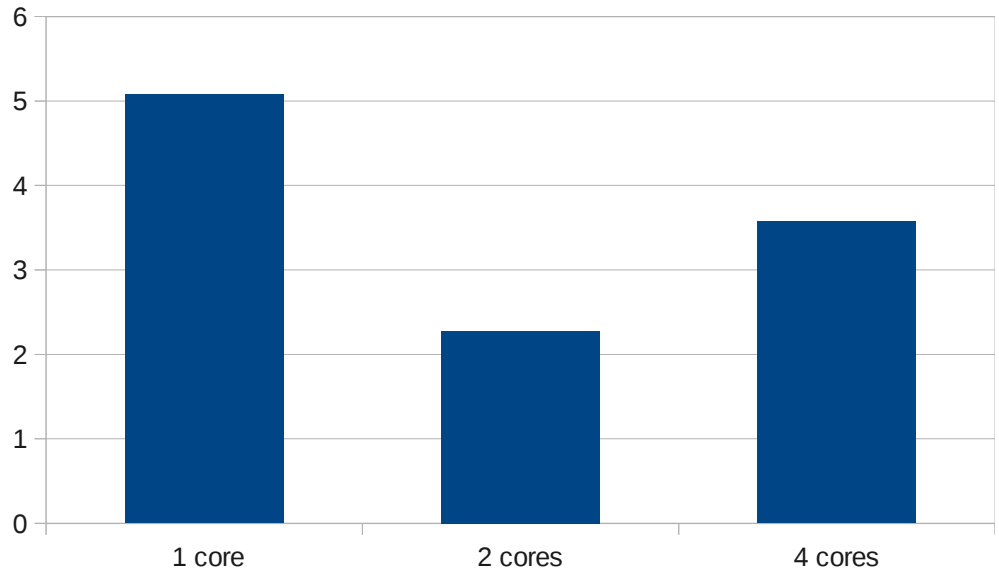
CAPTURAS DE PANTALLA:
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores, variando entre 1, 2, 4, 6, 8, y 12) en atcgrid y en el PC local del código paralelo implementado para tres tamaños de las matrices (N = 100, 1000 y 1500). Presente los resultados del estudio en tablas de valores y en gráficas. Consulte la Lección 6/Tema 2.

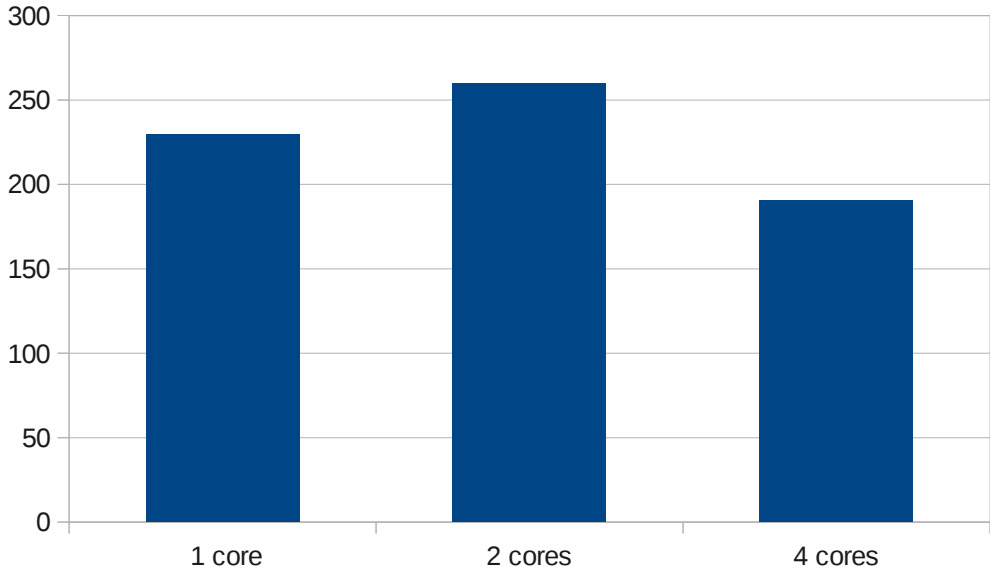
pclocal

N	1 core	2 cores	4 cores	6 cores	8 cores	12 cores
100	0.000050790	0.000022747	0.000035766			
1000	0.002294342	0.002595887	0.001902926			
1500	0.004995730	0.004532448	0.004368870			

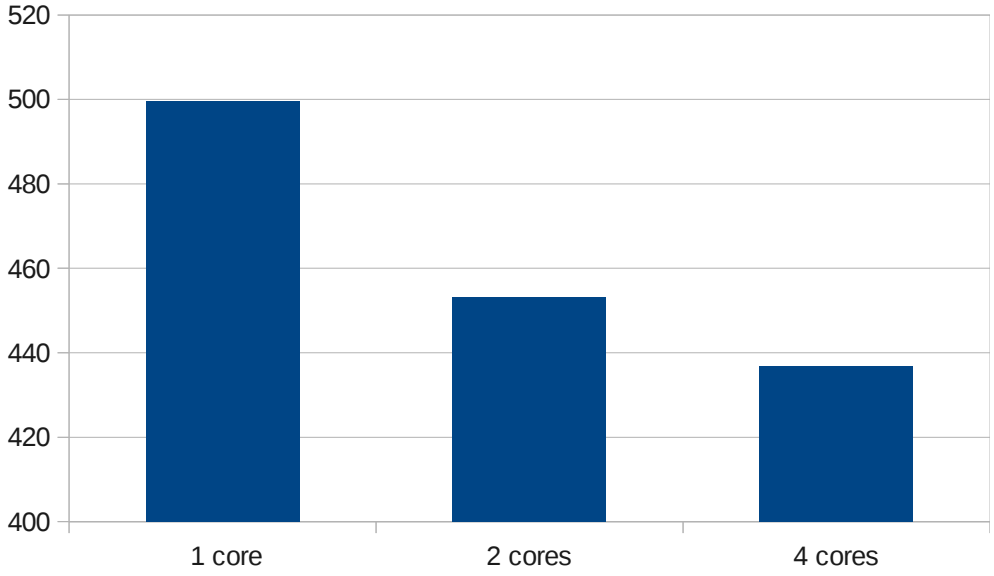
N = 100



N = 1000



N = 1500

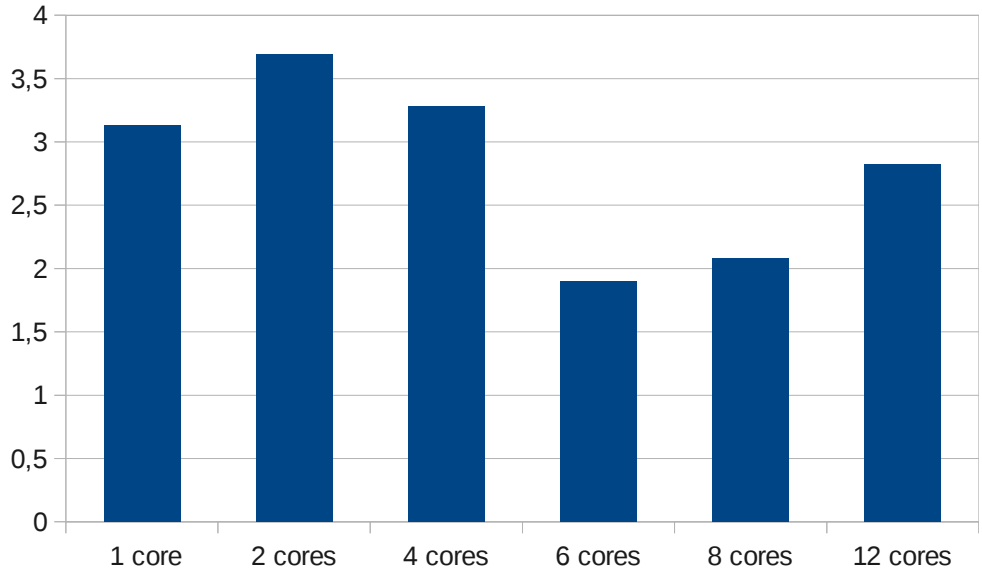


Gráficas con escala 1:100000

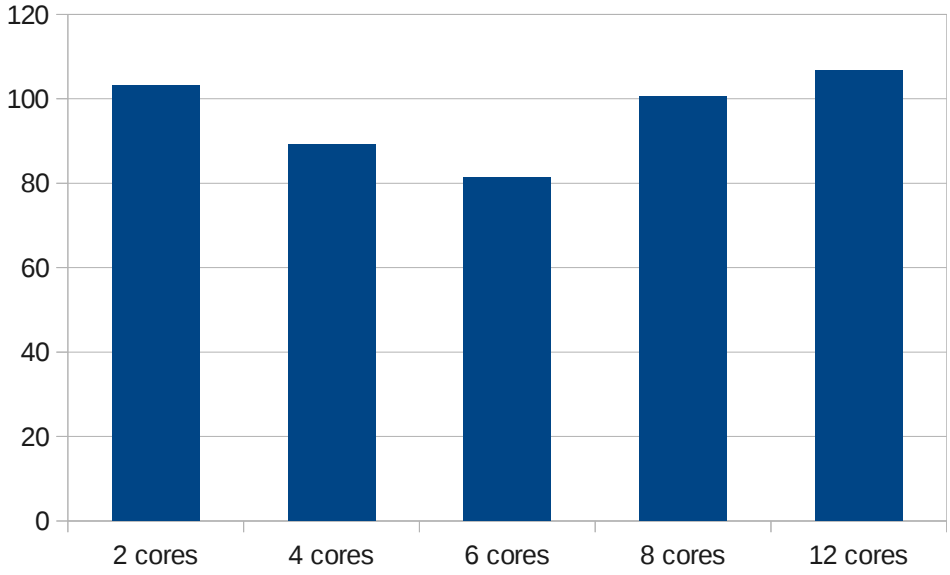
atcgrid

N	1 core	2 cores	4 cores	6 cores	8 cores	12 cores
100	0.000031314	0.000036879	0.000032849	0.000018967	0.000020808	0.000028207
1000	0.001910384	0.001031281	0.000892866	0.000814603	0.001005904	0.001067617
1500	0.004881577	0.002757703	0.002936023	0.002956914	0.002519343	0.002573324

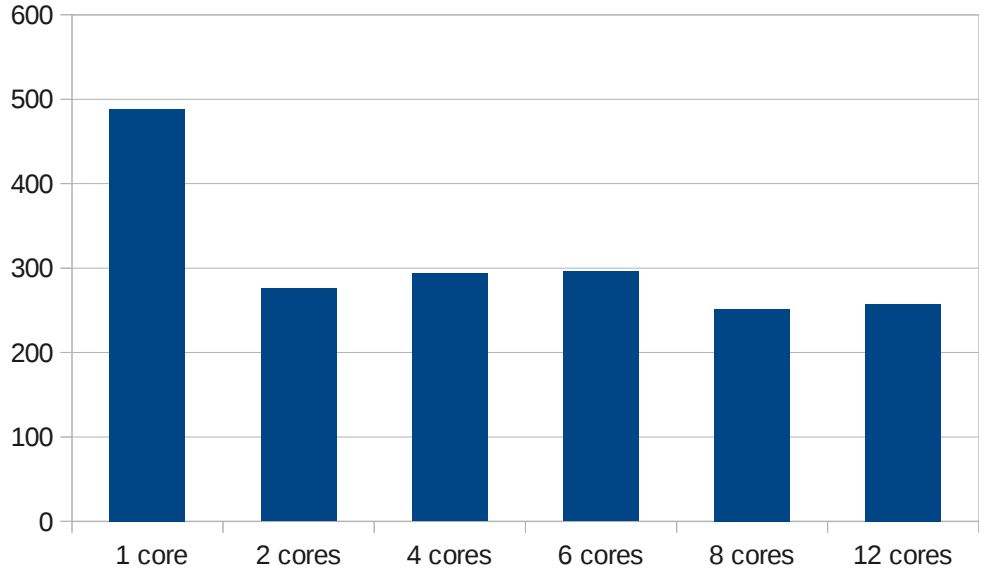
N = 100



N = 1000


120
100
80
60
40
20
0
2 cores
4 cores
6 cores
8 cores
12 cores

N = 1500


600
500
400
300
200
100
0
1 core
2 cores
4 cores
6 cores
8 cores
12 cores