

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): Néstor Rodríguez Vico

Grupo de prácticas: A1

Fecha de entrega: 12 Mayo

Fecha evaluación en clase: 12 Mayo

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.

CÓDIGO FUENTE: `if-clauseModificado.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, x;
    if(argc < 2) {
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) n=20;
    x = atoi(argv[2]); if (x>4) x=4;

    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(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:

```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario$ ./1-if-clauseModificado 5 1
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 0 suma de a[4]=4 sumalocal=10
thread master=0 imprime suma=10
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario$ ./1-if-clauseModificado 5 4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 1 suma de a[2]=2 sumalocal=2
thread 2 suma de a[3]=3 sumalocal=3
thread 3 suma de a[4]=4 sumalocal=4
thread master=0 imprime suma=10

```

RESPUESTA: Podemos ver que para 1, todo lo ejecuta la hebra 0, ya que solo hay una hebra, la master. En el segundo caso, se crean cuatro hebras, de la 0 a la 3.

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)
- chunk= 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 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 9 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 12 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 13 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 14 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 15 | 1 | 1 | 1 | 0 | 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-clause.d.c | | | schedule-clause.g.c | | |
|-----------|-------------------|---|---|---------------------|---|---|---------------------|---|---|
| | 1 | 2 | 4 | 1 | 2 | 4 | 1 | 2 | 4 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 3 | 3 |
| 1 | 1 | 0 | 0 | 3 | 1 | 0 | 1 | 3 | 3 |
| 2 | 2 | 1 | 0 | 1 | 2 | 0 | 1 | 3 | 3 |
| 3 | 3 | 1 | 0 | 3 | 2 | 0 | 1 | 3 | 3 |
| 4 | 0 | 2 | 1 | 3 | 3 | 2 | 0 | 0 | 1 |
| 5 | 1 | 2 | 1 | 3 | 3 | 2 | 0 | 0 | 1 |
| 6 | 2 | 3 | 1 | 3 | 0 | 2 | 0 | 0 | 1 |
| 7 | 3 | 3 | 1 | 3 | 0 | 2 | 3 | 1 | 1 |
| 8 | 0 | 0 | 2 | 3 | 2 | 1 | 3 | 1 | 0 |
| 9 | 1 | 0 | 2 | 3 | 2 | 1 | 3 | 1 | 0 |
| 10 | 2 | 1 | 2 | 3 | 2 | 1 | 2 | 2 | 0 |
| 11 | 3 | 1 | 2 | 3 | 2 | 1 | 2 | 2 | 0 |
| 12 | 0 | 2 | 3 | 3 | 2 | 3 | 0 | 3 | 3 |
| 13 | 1 | 2 | 3 | 3 | 2 | 3 | 0 | 3 | 3 |
| 14 | 2 | 3 | 3 | 3 | 2 | 3 | 0 | 0 | 3 |
| 15 | 3 | 3 | 3 | 3 | 2 | 3 | 0 | 0 | 3 |

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

RESPUESTA: Con `static`, las iteraciones se dividen en unidades de chunk iteraciones y se asignan en round-robin. Con `dynamic` la unidad de distribución tiene chunk iteraciones, pero si una hebra es más rápido que otra y termina su tarea antes, se le asignará una nueva unidad de chunk iteraciones. En el caso de `guided`, el chunk indica el valor mínimo del tamaño de bloque.

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`

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

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

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

```

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 single
    {
        printf("dyn-var: %d\n",omp_get_dynamic());
        printf("nthreads-var: %d\n",omp_get_max_threads());
        printf("thread-limit-var: %d\n",omp_get_thread_limit());

//https://gcc.gnu.org/onlinedocs/libgomp/omp_005fget_005fschedule.html
        omp_get_schedule(&kind,&chunk_size);
        printf("run-sched-var: (Kind: %d, Modifier: %d) \n",kind,chunk_size);
    }

    #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("dyn-var FUERA de 'parallel for': %d \n",omp_get_dynamic());
    printf("nthreads-var FUERA de 'parallel for': %d\n",omp_get_max_threads());
    printf("thread-limit-var FUERA de 'parallel for': %d\n",omp_get_thread_limit());
    omp_get_schedule(&kind,&chunk_size);
    printf("run-sched-var dentro de 'parallel for'. (Kind: %d. Modifier: %d)\n",kind,chunk_size);

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

```

CAPTURAS DE PANTALLA:

```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario/3$ ./scheduled-clauseModificado 5
3
dyn-var: 0
nthreads-var: 4
thread-limit-var: 2147483647
run-sched-var: (Kind: 2, Modifier: 1)
  thread 3 suma a[0]=0 suma=0
  thread 3 suma a[1]=1 suma=1
  thread 3 suma a[2]=2 suma=3
  thread 0 suma a[3]=3 suma=3
  thread 0 suma a[4]=4 suma=7
Fuera de 'parallel for' suma=7
dyn-var FUERA de 'parallel for': 0
nthreads-var FUERA de 'parallel for': 4
thread-limit-var FUERA de 'parallel for': 2147483647
run-sched-var dentro de 'parallel for'. (Kind: 2, Modifier: 1)
Fuera de 'parallel for' suma=7
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario/3$ export OMP_DYNAMIC=FALSE
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario/3$ export OMP_NUM_THREADS=2
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario/3$ export OMP_THREAD_LIMIT=20
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario/3$ export OMP_SCHEDULE="static,4"
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario/3$ ./scheduled-clauseModificado 5
3
dyn-var: 0
nthreads-var: 2
thread-limit-var: 20
run-sched-var: (Kind: 1, Modifier: 4)
  thread 0 suma a[3]=3 suma=3
  thread 0 suma a[4]=4 suma=7
  thread 1 suma a[0]=0 suma=0
  thread 1 suma a[1]=1 suma=1
  thread 1 suma a[2]=2 suma=3
Fuera de 'parallel for' suma=7
dyn-var FUERA de 'parallel for': 0
nthreads-var FUERA de 'parallel for': 2
thread-limit-var FUERA de 'parallel for': 20
run-sched-var dentro de 'parallel for'. (Kind: 1, Modifier: 4)
Fuera de 'parallel for' suma=7

```

RESPUESTA: No, son iguales.

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`

```

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

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

int main(int argc, char **argv) {
    int i, n=200, chunk, 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 single
        {
            printf("Dentro de Parallel: omp_get_num_threads: %d\n", omp_get_num_threads());
        }
    }
}

```

```

\n",omp_get_num_threads());
    printf("Dentro de Parallel: omp_get_num_procs: %d\n",omp_get_num_procs());
    printf("Dentro de 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: omp_get_num_threads: %d\n",omp_get_num_threads());
printf("Fuera de Parallel: omp_get_num_procs: %d \n",omp_get_num_procs());
printf("Fuera de Parallel: omp_in_parallel: %d \n",omp_in_parallel());

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

```

CAPTURAS DE PANTALLA:

```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario/4$ ./scheduled-clauseModificado4 5 3
Dentro de Parallel: omp_get_num_threads: 4
Dentro de Parallel: omp_get_num_procs: 4
Dentro de Parallel: omp_in_parallel: 1
thread 3 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1
thread 3 suma a[2]=2 suma=3
thread 0 suma a[3]=3 suma=3
thread 0 suma a[4]=4 suma=7
Fuera de Parallel: omp_get_num_threads: 1
Fuera de Parallel: omp_get_num_procs: 4
Fuera de Parallel: omp_in_parallel: 0
Fuera de 'parallel for' suma=7

```

RESPUESTA: Se obtienen distintos resultados en `omp_get_num_threads` y en `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.

CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int main(int argc, char **argv) {
    int i, n=200,chunk,a[n],suma=0;
    int * modifier;
    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 single
    {
        printf("DENTRO DEL PARALLEL.\n\n");
        printf("dyn-var: %d \n",omp_get_dynamic());
        omp_set_dynamic(3);
        printf("Modificamos dyn-var con omp_set_dynamic(1) y el resultado es:
%d\n", omp_get_dynamic());
        printf("nthreads-var: %d \n",omp_get_max_threads());
        omp_set_num_threads(8);
        printf("Modificamos nthreads-var con omp_set_num_threads(8); y el
resultado es: %d\n",omp_get_max_threads());

        omp_get_schedule(&kind,&modifier);
        printf("run-sched-var: (Kind: %d, Modifier: %d)\n",kind,modifier);
        omp_set_schedule(2,2);
        omp_get_schedule(&kind,&modifier);
        printf("Modificamos run-sched-var con omp_set_schedule(2,2) y el
resultado de Kind es %d y el de Modifier es: %d \n",kind,modifier);
    }
    #pragma omp barrier
    #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("\n\nFUERA DEL PARALLEL.\n");
printf("suma=%d\n", suma);
printf("dyn-var: %d \n",omp_get_dynamic());
printf("nthreads-var: %d \n",omp_get_max_threads());
omp_get_schedule(&kind,&modifier);
printf("run-sched-var: (Kind: %d, Modifier: %d)\n",kind,modifier);
return(0);
}

```

CAPTURAS DE PANTALLA:

```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Ej_seminario/5$ ./scheduled-clauseModificado5 5 3
DENTRO DEL PARALLEL.

dyn-var: 0
Modificamos dyn-var con omp_set_dynamic(1) y el resultado es: 1
nthreads-var: 2
Modificamos nthreads-var con omp_set_num_threads(8); y el resultado es: 8
run-sched-var: (Kind: 1, Modifier: 4)
Modificamos run-sched-var con omp_set_schedule(3,2) y el resultado de Kind es 3 y el de Modifier es: 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 1 suma a[3]=3 suma=3
thread 1 suma a[4]=4 suma=7

FUERA DEL PARALLEL.
suma=7
dyn-var: 0
nthreads-var: 2
run-sched-var: (Kind: 1, Modifier: 4)

```

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.

CÓDIGO FUENTE: pmtv-secuencial.c

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
    int i, j;
    double t1, t2, total, res;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
    (sizeof(unsigned int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el
    tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio
    suficiente malloc devuelve NULL
    M = (double**) malloc(N*sizeof(double *));

    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para los
    vectores\n");
        exit(-2);
    }

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if ( M[i]==NULL ){
            printf("Error en la reserva de espacio
    para los vectores\n");
            exit(-2);
        }
    }

    //A partir de aqui se pueden acceder las componentes de la
    matriz como M[i][j]

    //Inicializar matriz y vectores

    for(i=0; i<N; i++) {
        v1[i]=2;
    }
```



```

for(i=0; i<N; i++) {
    for(j=0; j<N; j++) {
        //Superior igualada a 1
        if( i<=j)
            M[i][j]=1;
        //Inferior igualada a 0
        else
            M[i][j]=0;
    }
}

//Medida de tiempo
t1 = omp_get_wtime();

//Calcular producto de matriz por vector v2 = M · v1
for(i=0; i<N; i++) {
    res=0;
    for(j=0; j<N; j++) {
        if( i<=j)
            res=res+(M[i][j]*v1[j]);
    }
    v2[i]=res;
}

//Medida de tiempo
t2 = omp_get_wtime();
total = t2 - t1;

//Imprimir el resultado y el tiempo de ejecución
printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
total,N,v2[0],N-1,v2[N-1]);

    if (N < 30){
for(i=0; i<N; i++)
    printf("%f ", v2[i]);
printf("\n");
}

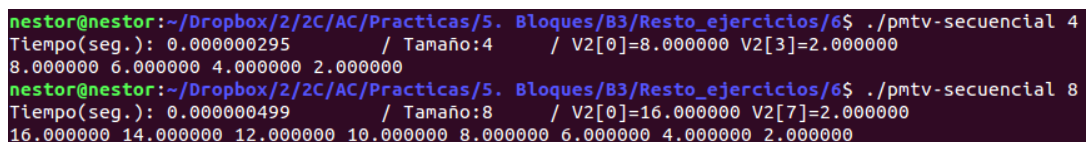
    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    for (i=0; i<N; i++)
        free(M[i]);

    free(M);

return 0;
}

```

CAPTURAS DE PANTALLA:



```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/6$ ./pmtv-secuencial 4
Tiempo(seg.): 0.000000295 / Tamaño:4 / V2[0]=8.000000 V2[3]=2.000000
8.000000 6.000000 4.000000 2.000000
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/6$ ./pmtv-secuencial 8
Tiempo(seg.): 0.000000499 / Tamaño:8 / V2[0]=16.000000 V2[7]=2.000000
16.000000 14.000000 12.000000 10.000000 8.000000 6.000000 4.000000 2.000000

```

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: El valor por defecto del chunk lo podemos ver en la siguiente imagen:

```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ export OMP_SCHEDULE="static"
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ ./pmtv-OpenMP 5
run-sched-var: (Kind: 1, Modifier: 0)
Tiempo(seg.): 0.000080286 / Tamaño:5 / V2[0]=10.000000 V2[4]=2.000000
10.000000 8.000000 6.000000 4.000000 2.000000
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ export OMP_SCHEDULE="dynamic"
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ ./pmtv-OpenMP 5
run-sched-var: (Kind: 2, Modifier: 1)
Tiempo(seg.): 0.000126539 / Tamaño:5 / V2[0]=10.000000 V2[4]=2.000000
10.000000 8.000000 6.000000 4.000000 2.000000
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ export OMP_SCHEDULE="guided"
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ ./pmtv-OpenMP 5
run-sched-var: (Kind: 3, Modifier: 1)
Tiempo(seg.): 0.000074549 / Tamaño:5 / V2[0]=10.000000 V2[4]=2.000000
10.000000 8.000000 6.000000 4.000000 2.000000

```

Para obtener dicho valor, he usado la función: `omp_get_schedule(&kind,&chunk_size)`. Para el caso de static, el valor obtenido por la función es 0, pero esto quiere decir que el chunk es calculado en tiempo de ejecución: $\text{chunk} = \text{num_iteraciones} / \text{num_hebras}$.

CÓDIGO FUENTE: `pmtv-OpenMP.c`

```

// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
    int i, j;
    double t1, t2, total, res;
    int chunk_size;
    omp_sched_t kind;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Uso: ./pmtv-OpenMP <tamaño>\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
    (sizeof(unsigned int) = 4 B)

    double *v1, *v2, **M;
    v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el

```

```

tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio
suficiente malloc devuelve NULL
    M = (double**) malloc(N*sizeof(double *));

    if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
        printf("Error en la reserva de espacio para los
vectores\n");
        exit(-2);
    }

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if ( M[i]==NULL ){
            printf("Error en la reserva de espacio
para los vectores\n");
            exit(-2);
        }
    }

    //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

    //Inicializar matriz y vectores

    for(i=0; i<N; i++) {
        v1[i]=2;
    }

    for(i=0; i<N; i++) {
        for(j=0; j<N; j++) {
            //Superior igualada a 1
            if(i<=j)
                M[i][j]=1;
            //Inferior igualada a 0
            else
                M[i][j]=0;
        }
    }

    //Calcular producto de matriz por vector v2 = M . v1
    #pragma omp parallel shared(M,v1,v2) private(i,j)
    {
        #pragma omp single
        {
            omp_get_schedule(&kind,&chunk_size);
            printf("run-sched-var: (Kind: %d, Modifier: %d) \n",kind,chunk_size);
            //Medida de tiempo
            t1 = omp_get_wtime();
        }
        #pragma omp for schedule (runtime)
        for(i=0; i<N; i++){
            printf("thread %d ejecuta la iteración %d del
bucle\n",omp_get_thread_num(),i);
            v2[i] = 0;
            for(j=i; j<N; j++){
                v2[i] = v2[i] + M[i][j] * v1[j];
            }
        }

        #pragma omp single
        {

```

```

        //Medida de tiempo
        t2 = omp_get_wtime();
    }
}

    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=%8.6f\n",
total,N,v2[0],N-1,v2[N-1]);

        if (N < 30){
    for(i=0; i<N; i++)
        printf("%f ", v2[i]);
    printf("\n");
}

        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2
        for (i=0; i<N; i++)
            free(M[i]);

        free(M);

    return 0;
}

```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ export OMP_SCHEDULE="static"
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ ./pmtv-OpenMP 5
run-sched-var: (Kind: 1, Modifier: 0)
Tiempo(seg.): 0.000080286 / Tamaño:5 / V2[0]=10.000000 V2[4]=2.000000
10.000000 8.000000 6.000000 4.000000 2.000000
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ export OMP_SCHEDULE="dynamic"
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ ./pmtv-OpenMP 5
run-sched-var: (Kind: 2, Modifier: 1)
Tiempo(seg.): 0.000126539 / Tamaño:5 / V2[0]=10.000000 V2[4]=2.000000
10.000000 8.000000 6.000000 4.000000 2.000000
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ export OMP_SCHEDULE="guided"
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/7$ ./pmtv-OpenMP 5
run-sched-var: (Kind: 3, Modifier: 1)
Tiempo(seg.): 0.000074549 / Tamaño:5 / V2[0]=10.000000 V2[4]=2.000000
10.000000 8.000000 6.000000 4.000000 2.000000

```

TABLA RESULTADOS, SCRIPT Y GRÁFICA ATCGRID

SCRIPT: pmtv-OpenMP_atcgrid.sh

```

#!/bin/bash
#Se asigna al trabajo el nombre pmtv-OpenMP
#PBS -N pmtv-OpenMP
#Se asigna al trabajo la cola ac
#PBS -q ac

export OMP_NUM_THREADS=12
export OMP_SCHEDULE="static"
$PBS_O_WORKDIR/pmtv-OpenMP 24832
export OMP_SCHEDULE="static, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 24832
export OMP_SCHEDULE="static, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 24832

export OMP_NUM_THREADS=12

```

```

export OMP_SCHEDULE="dynamic"
$PBS_O_WORKDIR/pmtv-OpenMP 24832
export OMP_SCHEDULE="dynamic, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 24832
export OMP_SCHEDULE="dynamic, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 24832

export OMP_NUM_THREADS=12
export OMP_SCHEDULE="guided"
$PBS_O_WORKDIR/pmtv-OpenMP 24832
export OMP_SCHEDULE="guided, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 24832
export OMP_SCHEDULE="guided, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 24832

```

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

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

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

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.

CÓDIGO FUENTE: pmm-secuencial.c

```

// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
    int i, j, k;
    double t1, t2, total;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc < 2) {
        printf("Falta tamaño de matriz\n");
        exit(-1);
    }
}

```

```

    }

    unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1=4294967295
    (sizeof(unsigned int) = 4 B)

    double **M1, **M2, **M3;
    //M1 = M2 * M3;
    M1 = (double**) malloc(N*sizeof(double *));
    M2 = (double**) malloc(N*sizeof(double *));
    M3 = (double**) malloc(N*sizeof(double *));

    if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
        printf("Error en la reserva de espacio para las
matrices\n");
        exit(-2);
    }

    for (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[i]==NULL || M2[i]==NULL || M3[i]==NULL ){
            printf("Error en la reserva de espacio
para los matrices\n");
            exit(-2);
        }
    }

    //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

    //Inicializar matriz y vectores

    for (j=0; j<N; j++){
        for (i=0; i<N; i++){
            M1[i][j] = 0;
            M2[i][j] = 2;
            M3[i][j] = 2;
        }
    }

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calcular producto de matriz por matriz M1 = M2 * M3

    for (i=0; i<N; i++){
        for(j=0; j<N; j++){
            for (k=0; k<N; k++){
                M1[i][j] = M1[i][j] +
(M2[i][k]*M3[k][j]);
            }
        }
    }

    //Medida de tiempo
    t2 = omp_get_wtime();
    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ M1[0][0]=%8.6f M1[%d][%d]=
%8.6f\n", total,N,M1[0][0],N-1,N-1,M1[N-1][N-1]);

```

```

        for (i=0; i<N; i++){
            free(M1[i]); free(M2[i]); free(M3[i]);
        }

        free(M1); free(M2); free(M3);

    return 0;
}

```

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

```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/8$ ./compilar
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/8$ ./pmm-secuencial 8
Tiempo(seg.): 0.000001622      / Tamaño:8      / M1[0][0]=32.000000 M1[7][7]=32.000000
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/8$ ./pmm-secuencial 11
Tiempo(seg.): 0.000002752      / Tamaño:11     / M1[0][0]=44.000000 M1[10][10]=44.000000

```

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:

CÓDIGO FUENTE: pmm-OpenMP.c

```

// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv) {
    int i, j, k;
    double t1, t2, total, suma;

    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Falta tamaño de matriz\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
    (sizeof(unsigned int) = 4 B)

    double **M1, **M2, **M3;
    //M1 = M2 * M3;
    M1 = (double**) malloc(N*sizeof(double *));
    M2 = (double**) malloc(N*sizeof(double *));
    M3 = (double**) malloc(N*sizeof(double *));

    if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
        printf("Error en la reserva de espacio para las
matrices\n");
        exit(-2);
    }

    for (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[i]==NULL || M2[i]==NULL || M3[i]==NULL ){
            printf("Error en la reserva de espacio
para los matrices\n");
            exit(-2);
        }
    }

    //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

    //Inicializar matriz
#pragma omp parallel shared(M2,M3) private(i,j)
    {
        #pragma omp for schedule (runtime)
        for(i=0;i<N; i++){
            for(j=0;j< N; j++){
                M1[i][j]=0;
            }
        }
        #pragma omp for schedule (runtime)
        for(i=0;i<N; i++){
            for(j=0;j<N; j++){
                M2[i][j] = 2;
            }
        }

        #pragma omp for schedule (runtime)
        for(i=0;i<N; i++){
            for(j=0;j<N; j++){
                M3[i][j] = 2;
            }
        }
    }

    //Calcular producto de matriz por matriz M1 = M2 * M3
#pragma omp parallel shared(M1,M2,M3) private(i,j,k)
    {
        //Medida de tiempo
        #pragma omp single
        {
            t1 = omp_get_wtime();
        }
        #pragma omp for schedule (runtime)
        for(i=0;i<N; i++){
            for(j=0;j<N; j++){
                for(k=0;k<N; k++){
                    M1[i][j] = M1[i][j] + M2[i][k] * M3[k][j];
                }
            }
        }
        //Medida de tiempo
        #pragma omp single
        {
            t2 = omp_get_wtime();
        }
    }

    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ M1[0][0]=%8.6f M1[%d][%d]=

```



```

%8.6f\n", total, N, M1[0][0], N-1, N-1, M1[N-1][N-1]);

        for (i=0; i<N; i++){
            free(M1[i]); free(M2[i]); free(M3[i]);
        }

        free(M1); free(M2); free(M3);

    return 0;
}

```

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

```

nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/9$ ./pmm-OpenMP 8
Tiempo(seg.): 0.000005634 / Tamaño:8 / M1[0][0]=32.000000 M1[7][7]=32.000000
nestor@nestor:~/Dropbox/2/2C/AC/Practicas/5. Bloques/B3/Resto_ejercicios/9$ ./pmm-OpenMP 11
Tiempo(seg.): 0.000009046 / Tamaño:11 / M1[0][0]=44.000000 M1[10][10]=44.000000

```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. 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

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pcllocal.sh