2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos):
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

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

Ejercicios basados en los ejemplos del seminario práctico

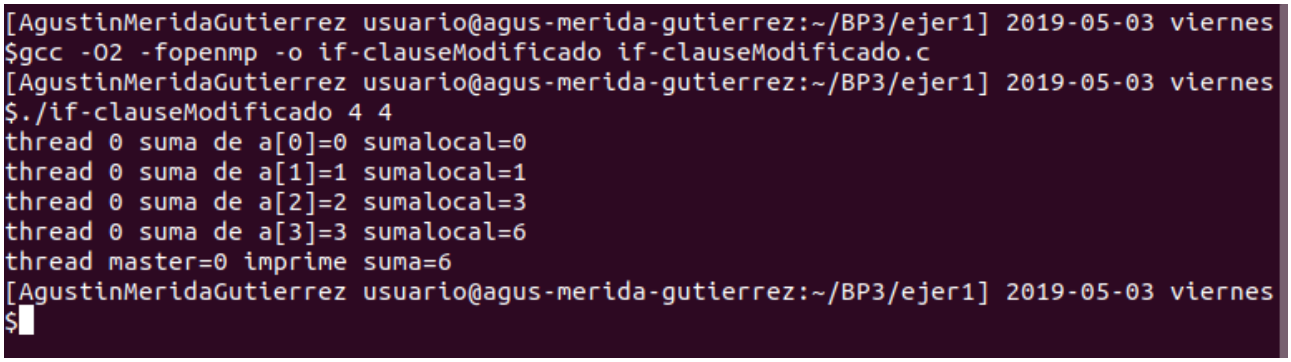
1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad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

CAPTURA CÓDIGO FUENTE: `if-clauseModificado.c`

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

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

    int i, n=20, tid, x;
    int a[n],suma=0,sumalocal;
    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)num_threads(x)\
        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:



RESPUESTA: No ocurre nada con respecto al if-clause original, debido a que el if(n>4) tiene mayor prioridad que num_threads(x). Por lo tanto, el resultado será el mismo.

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

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, asignandose estas en round-robin. Con ***dynamic*** la distribución se realiza en tiempo de ejecución. Es apropiado usarlo si se desconoce el tiempo de ejecución de las iteraciones. La unidad de distribución tiene chunk iteraciones(n/chunk). El problema es que hay sobrecarga adicional. Con ***guided*** la distirbución también se realiza en tiempo de ejecución. Es apropiado usarlo si se desconoce el tiempo de ejecución de las iteraciones o su número. Comienza con bloque largo y el tamaño del bloque va menguando (n.º iteraciones que restan dividido por n.º threads), no más pequeño que chunk (excepto la última). El problema de guided es que se realiza una sobrecarga extra, pero menor que dynamic para el mismo chunk.

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

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```
#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, *chunk_size;
    omp_sched_t kind;

    if(argc<2){
        printf(stderr,"\nFalta 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_max_threads());

            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]suma=%d\n",
            omp_get_thread_num(),i,a[i],suma);
    }

    }

    //Imprimimos fuera de la region parallel todas las variables y la suma
    printf("dyn-var Fuera de parallel: %d\n", omp_get_dynamic());
    printf("nthreads-var Fuera de parallel: %d\n",omp_get_max_threads());
    printf("thread-limit-var Fuera de parallel: %d\n",omp_get_max_threads());

    omp_get_schedule(&kind,&chunk_size);

    printf("run-sched-var Fuera de parallel --> Kind: %d, Modifier: %d\n",kind,
chunk_size);
```

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

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer3] 2019-05-14 martes
$export OMP_DYNAMIC=FALSE
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer3] 2019-05-14 martes
$export OMP_NUM_THREADS=2
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer3] 2019-05-14 martes
$export OMP_NUM_THREAD_LIMIT=20
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer3] 2019-05-14 martes
$export OMP_SCHEDULE="dynamic,4"
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer3] 2019-05-14 martes
$./scheduled-clauseModificado 7 3
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2
run-sched-var --> Kind: 2, Modifier: 4
thread 1 suma a[0]suma=0
thread 1 suma a[1]suma=1
thread 1 suma a[2]suma=2
thread 1 suma a[6]suma=6
thread 0 suma a[3]suma=3
thread 0 suma a[4]suma=4
thread 0 suma a[5]suma=5
dyn-var Fuera de parallel: 0
nthreads-var Fuera de parallel: 2
thread-limit-var Fuera de parallel: 2
run-sched-var Fuera de parallel --> Kind: 2, Modifier: 4
Fuera de 'parallel for' suma=9
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer3] 2019-05-14 martes
$./scheduled-clauseModificado 5 3
dyn-var: 0
nthreads-var: 2
thread-limit-var: 2
run-sched-var --> Kind: 2, Modifier: 4
thread 0 suma a[3]suma=3
thread 0 suma a[4]suma=4
thread 1 suma a[0]suma=0
thread 1 suma a[1]suma=1
thread 1 suma a[2]suma=2
dyn-var Fuera de parallel: 0
nthreads-var Fuera de parallel: 2
thread-limit-var Fuera de parallel: 2
run-sched-var Fuera de parallel --> Kind: 2, Modifier: 4
Fuera de 'parallel for' suma=7
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer3] 2019-05-14 martes
$
```

RESPUESTA: Las variables 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.

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

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

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

    if(argc<2){
        printf(stderr,"\nFalta 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_max_threads());

            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]suma=%d\n",
            omp_get_thread_num(),i,a[i],suma);
    }

    }

    //Imprimimos fuera de la region parallel todas las variables y la suma
    printf("dyn-var Fuera de parallel: %d\n", omp_get_dynamic());
    printf("nthreads-var Fuera de parallel: %d\n",omp_get_max_threads());
    printf("thread-limit-var Fuera de parallel: %d\n",omp_get_max_threads());

    omp_get_schedule(&kind,&chunk_size);

    printf("run-sched-var Fuera de parallel --> Kind: %d, Modifier: %d\n",kind,
chunk_size);

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

CAPTURAS DE PANTALLA:

```
◆ ◆◆[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer4] 2019-05-14 martes
$./scheduled-clauseModificado4 5 3
VALORES DE LAS VARIABLES DENTRO DE PARALLEL
omp_get_num_threads: 8
omp_get_num_procs: 8
omp_in_parallel: 1
thread 5 suma a[3]suma=3
thread 5 suma a[4]suma=4
thread 4 suma a[0]suma=0
thread 4 suma a[1]suma=1
thread 4 suma a[2]suma=2
VALORES DE LAS VARIABLES FUERA DE PARALLEL
omp_get_num_threads: 1
omp_get_num_procs: 8
omp_in_parallel: 0
Fuera de 'parallel for' suma=7
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer4] 2019-05-14 martes
$
```

RESPUESTA: Obtenemos resultados diferentes en omp_get_num_threads (dentro de parallel, se muestran todas pero fuera solamente una) y en omp_in_parallel que dentro tiene un valor de 1, y fuera un valor de 0

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`

```
 #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, *chunk_size;
    omp_sched_t *kind;

    if(argc<2){
        printf(stderr,"\nFalta 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("VALORES DE LAS VARIABLES DENTRO DE PARALLEL\n");
            printf("dyn-var: %d\n", omp_get_dynamic());
            //Modificamos la variable dyn y la mostramos por pantalla
            omp_set_dynamic(2);
            printf("Variable dyn-var modificada: %d\n",omp_get_dynamic);
```

```
            printf("nthreads-var: %d\n",omp_get_max_threads());
            //Modificamos la variable nthreads y la mostramos por pantalla
            omp_set_num_threads(4);
            printf("Variable nthreads-var modificada: %d\n",omp_get_max_threads());

            omp_get_schedule(&kind,&chunk_size);
            printf("run-sched-var --> Kind: %d, Modifier: %d\n",kind, chunk_size);
            //Modificamos la variable run-sched y la mostramos por pantalla
            omp_set_schedule(2,4);
            omp_get_schedule(&kind,&chunk_size);
            printf("Variable run-sched-var modificada --> 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]suma=%d\n",
            omp_get_thread_num(),i,a[i],suma);
    }

    }

    //Imprimimos fuera de la region parallel todas las variables y la suma
    printf("VALOR DE LAS VARIABLES FUERA DE PARALLEL");
    printf("dyn-var Fuera de parallel: %d\n", omp_get_dynamic());
    printf("nthreads-var Fuera de parallel: %d\n",omp_get_max_threads());

    omp_get_schedule(&kind,&chunk_size);

    printf("run-sched-var Fuera de parallel --> Kind: %d, Modifier: %d\n",kind,
chunk_size);

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

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer5] 2019-05-14 martes
$./scheduled-clauseModificado5 7 3
VALORES DE LAS VARIABLES DENTRO DE PARALLEL
dyn-var: 0
Variable dyn-var modificada: 1200041856
nthreads-var: 8
Variable nthreads-var modificada: 4
run-sched-var --> Kind: 2, Modifier: 1
Variable run-sched-var modificada --> Kind: 2, Modifier: 4
thread 1 suma a[0]suma=0
thread 1 suma a[1]suma=1
thread 7 suma a[3]suma=3
thread 5 suma a[6]suma=6
thread 7 suma a[4]suma=4
thread 7 suma a[5]suma=5
thread 1 suma a[2]suma=2
VALOR DE LAS VARIABLES FUERA DE PARALLELdyn-var Fuera de parallel: 0
nthreads-var Fuera de parallel: 8
run-sched-var Fuera de parallel --> Kind: 2, Modifier: 1
Fuera de 'parallel for' suma=6
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer5] 2019-05-14 martes
$
```

Resto de ejercicios

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

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

CAPTURA CÓDIGO FUENTE: `pmtv-secuencial.c`

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

#define VECTOR_DYNAMIC  //Como se nos ha indicado en clase, hacemos uso de este tipo de
vectores


int main(int argc, char** argv)
{
    int i, j;
    double t_ini, t_fin, t_total, res;

    //Leemos si el numero de datos introducidos es correcto
    if (argc<2){
        printf("Falta el tamaño de la matriz y e el tamaño del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); //MAXIMO N = 2^32

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

    //Comprobamos que la reserva de memoria se ha realizado correctamente
    if ((v1==NULL) || (v2==NULL) || (M==NULL)){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }

    for(i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if(M[i]==NULL){
            printf("Error al reservar el espacio de la matriz\n");
            exit(-2);
        }
    }

    //Inicializamos la matriz
    for(i=0; i<N; i++){
        v1[i]=1;
    }

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            if(i<=j){
                M[i][j]=1;
```

```
            }else{
                M[i][j]=0;
            }

        }
    }

    //Calculamos v2 = M * v1 y lo que se tarda en realizar esta operación
    t_ini = omp_get_wtime();

    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;
    }

    t_fin = omp_get_wtime();
    t_total = t_fin-t_ini;

    //Mostramos el resultado y su tiempo de ejecución;
    printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f v2[%d]=%8.6f\n", t_total, N,
v2[0], N-1, v2[N-1]);
    if(N<30){
        printf("%f ", v2[i]);
    printf("\n");
    }

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

    free(M);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer6] 2019-05-14 martes
$./pmtv-secuencial 4
Tiempo(seg.): 0.000000560        / Tamaño:4      / V2[0]=4.000000 v2[3]=1.000000
503737752228788171933890038748399105666489241195181639909971847859020864850718256044088048707052890988627388980678772942014736230643311787386122935495987677879163572121942097 92.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer6] 2019-05-14 martes
$
```

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:

APARTADO A

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer7] 2019-05-14 martes
$export OMP_SCHEDULE='static'
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer7] 2019-05-14 martes
$./pmtv-OpenMP 4
run-sched-var --> Kind: 1, Modifier: 0
Tiempo(seg.): 0.000007632        / Tamaño:4     / V2[0]=4.000000 v2[3]=1.000000
503737752228788171933890038748399105666489241195181639909971847859020864850718256044088048707052890988627388980678772942014736230643311787386122935495987677879163572121942097920.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer7] 2019-05-14 martes
$export OMP_SCHEDULE='dynamic'
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer7] 2019-05-14 martes
$./pmtv-OpenMP 4
run-sched-var --> Kind: 2, Modifier: 1
Tiempo(seg.): 0.000008168        / Tamaño:4     / V2[0]=4.000000 v2[3]=1.000000
503737752228788171933890038748399105666489241195181639909971847859020864850718256044088048707052890988627388980678772942014736230643311787386122935495987677879163572121942097920.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer7] 2019-05-14 martes
$export OMP_SCHEDULE='guided'
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer7] 2019-05-14 martes
$./pmtv-OpenMP 4
run-sched-var --> Kind: 3, Modifier: 1
Tiempo(seg.): 0.000029233        / Tamaño:4     / V2[0]=4.000000 v2[3]=1.000000
503737752228788171933890038748399105666489241195181639909971847859020864850718256044088048707052890988627388980678772942014736230643311787386122935495987677879163572121942097920.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer7] 2019-05-14 martes
```

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

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

#define VECTOR_DYNAMIC  //Como se nos ha indicado en clase, hacemos uso de este tipo de
vectores


int main(int argc, char** argv)
{
    int i, j, *chunk_size;
    double t_ini, t_fin, t_total, res;
```

```
    omp_sched_t kind;

    //Leemos si el numero de datos introducidos es correcto
    if (argc<2){
        printf("Falta el tamaño de la matriz y e el tamaño del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); //MAXIMO N = 2^32-1

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

    //Comprobamos que la reserva de memoria se ha realizado correctamente
    if ((v1==NULL) || (v2==NULL) || (M==NULL)){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }

    for(i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
        if(M[i]==NULL){
            printf("Error al reservar el espacio de la matriz\n");
            exit(-2);
        }
    }

    //Inicializamos la matriz
    for(i=0; i<N; i++){
        v1[i]=1;
    }

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            if(i<=j){
                M[i][j]=1;
            }else{
                M[i][j]=0;
            }

        }
    }

    //Calculamos v2 = M * v1 y lo que se tarda en realizar esta operación
    #pragma omp parallel shared(v1,v2,M)
    {
        #pragma omp single
        {
            omp_get_schedule(&kind,&chunk_size);
            printf("run-sched-var --> Kind: %d, Modifier: %d\n",kind,chunk_size);
            t_ini = omp_get_wtime();
        }
        #pragma omp for schedule(runtime)
            for(i=0; i<N; i++){
                v2[i]=0;
                for(j=0; j<N; j++){
                    if(i<=j){
                        v2[i]=v2[i]+(M[i][j]*v1[j]);
                    }
                }
```

```
            }

        #pragma omp single
        {
            t_fin = omp_get_wtime();
        }
    }

    t_total = t_fin-t_ini;

    //Mostramos el resultado y su tiempo de ejecución;
    printf("Tiempo(seg.): %11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f v2[%d]=%8.6f\n", t_total, N,
v2[0], N-1, v2[N-1]);
    if(N<30){
        printf("%f ", v2[i]);
    printf("\n");
    }

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

    free(M);

    return 0;
}
```

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP3/ejer7] 2019-05-14 martes
$cat script_ejer7.sh.o21237
run-sched-var --> Kind: 1, Modifier: 0
Tiempo(seg.): 0.169086047       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 1, Modifier: 1
Tiempo(seg.): 0.160486598       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 1, Modifier: 64
Tiempo(seg.): 0.143553145       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 2, Modifier: 1
Tiempo(seg.): 0.220598287       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 2, Modifier: 1
Tiempo(seg.): 0.220322213       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 2, Modifier: 64
Tiempo(seg.): 0.126528397       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 3, Modifier: 1
Tiempo(seg.): 0.194157495       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 3, Modifier: 1
Tiempo(seg.): 0.182217230       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 3, Modifier: 64
Tiempo(seg.): 0.159465800       / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP3/ejer7] 2019-05-14 martes
$
```

```
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP3/ejer7] 2019-05-14 martes
$cat script_ejer7.sh.o21239
run-sched-var --> Kind: 1, Modifier: 0
Tiempo(seg.): 0.255149868        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 1, Modifier: 1
Tiempo(seg.): 0.134669052        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 1, Modifier: 64
Tiempo(seg.): 0.151859375        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 2, Modifier: 1
Tiempo(seg.): 0.220302060        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 2, Modifier: 1
Tiempo(seg.): 0.221236420        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 2, Modifier: 64
Tiempo(seg.): 0.126551175        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 3, Modifier: 1
Tiempo(seg.): 0.153425092        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 3, Modifier: 1
Tiempo(seg.): 0.171560790        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
run-sched-var --> Kind: 3, Modifier: 64
Tiempo(seg.): 0.166773154        / Tamaño:20000 / V2[0]=20000.000000 v2[19999]=1.000000
[AgustinMeridaGutierrez A1estudiante14@atcgrid:~/BP3/ejer7] 2019-05-14 martes
$
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-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

#Asignamos 12 threads y ejecutamos pmtv-OpenMP en static
export OMP_NUM_THREADS=12
export OMP_SCHEDULE="static"
$PBS_O_WORKDIR/pmtv-OpenMP 20000
export OMP_SCHEDULE="static, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 20000
export OMP_SCHEDULE="static, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 20000

#Asignamos 12 threads y ejecutamos pmtv-OpenMP en dynamic
export OMP_NUM_THREADS=12
export OMP_SCHEDULE="dynamic"
$PBS_O_WORKDIR/pmtv-OpenMP 20000
export OMP_SCHEDULE="dynamic, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 20000
export OMP_SCHEDULE="dynamic, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 20000

#Asignamos 12 threads y ejecutamos pmtv-OpenMP en guided
export OMP_NUM_THREADS=12
export OMP_SCHEDULE="guided"
$PBS_O_WORKDIR/pmtv-OpenMP 20000
export OMP_SCHEDULE="guided, 1"
$PBS_O_WORKDIR/pmtv-OpenMP 20000
export OMP_SCHEDULE="guided, 64"
$PBS_O_WORKDIR/pmtv-OpenMP 20000
```

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

Chunk	Static	Dynamic	Guided
por defecto	0.169086047	0.220598287	0.194157495
1	0.160486598	0.220322213	0.182217230
64	0.143553145	0.126528397	0.159465800
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.255149868	0.221236420	0.153425092
1	0.134669052	0.221236420	0.171560790
64	0.151859375	0.126551175	0.166773154

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`

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

#define VECTOR_DYNAMIC  //Como se nos ha indicado en clase, hacemos uso de este tipo de
vectores


int main(int argc, char** argv)
{
    int i, j, k;
    double t_ini, t_fin, t_total;

    //Leemos si el numero de datos introducidos es correcto
    if (argc<2){
        printf("Falta el tamaño de la matriz y e el tamaño del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); //MAXIMO N = 2^32

    double **A, **B, **C;
    A = (double**) malloc(N*sizeof(double *));
    B = (double**) malloc(N*sizeof(double *));
    C = (double**) malloc(N*sizeof(double *));
```

```
//Comprobamos que la reserva de memoria se ha realizado correctamente
//MATRIZ M1
    if ((A==NULL) || (B==NULL) || (C==NULL)){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }

    for(i=0; i<N; i++){
        A[i] = (double*) malloc(N*sizeof(double));
        B[i] = (double*) malloc(N*sizeof(double));
        C[i] = (double*) malloc(N*sizeof(double));

        if(A[i]==NULL || B[i]==NULL || C[i]==NULL){
            printf("Error al reservar el espacio de la matriz\n");
            exit(-2);
        }
    }
//*************************************************+


//Inicializamos las matriz
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            A[i][j]=0;
            B[i][j]=1;
            C[i][j]=1;
        }
    }

    //Calculamos A = B*C y lo que se tarda en realizar esta operación
    t_ini = omp_get_wtime();

    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            for(k=0; k<N; k++){
                A[i][j] = A[i][j] + (B[i][j]*C[i][j]);
            }
        }
    }

    t_fin = omp_get_wtime();
    t_total = t_fin-t_ini;

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

    //Liberamos el espacio reservado de las matrices A, B y C
    for(i=0; i<N; i++) {free(A[i]); free(B[i]); free(C[i]);}

    free(A);
    free(B);
    free(C);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer8] 2019-05-14 martes
$./pmm-secuencial 3
Tiempo(seg.): 0.000004408       / Tamaño:3      / A[0][0]=3.000000 A[2][-1240157216]=3.000000
[AgustinMeridaGutierrez usuario@agus-merida-gutierrez:~/BP3/ejer8] 2019-05-14 martes
$
```

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).

10.

DESCOMPOSICIÓN DE DOMINIO:

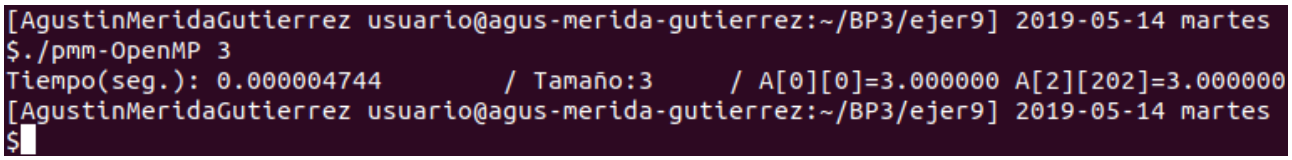
CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

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

#define VECTOR_DYNAMIC  //Como se nos ha indicado en clase, hacemos uso de este tipo de
vectores


int main(int argc, char** argv)
{
    int i, j, k;
    double t_ini, t_fin, t_total, res;

    //Leemos si el numero de datos introducidos es correcto
    if (argc<2){
        printf("Falta el tamaño de la matriz y e el tamaño del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); //MAXIMO N = 2^32

    double **A, **B, **C;
    A = (double**) malloc(N*sizeof(double *));
    B = (double**) malloc(N*sizeof(double *));
    C = (double**) malloc(N*sizeof(double *));

//Comprobamos que la reserva de memoria se ha realizado correctamente
//MATRIZ M1
    if ((A==NULL) || (B==NULL) || (C==NULL)){
        printf("Error al reservar el espacio de la matriz\n");
        exit(-2);
    }

    for(i=0; i<N; i++){
        A[i] = (double*) malloc(N*sizeof(double));
        B[i] = (double*) malloc(N*sizeof(double));
        C[i] = (double*) malloc(N*sizeof(double));

        if(A[i]==NULL || B[i]==NULL || C[i]==NULL){
            printf("Error al reservar el espacio de la matriz\n");
            exit(-2);
        }
    }
//*************************************************+


//Inicializamos las matriz
    #pragma omp parallel shared(B,C)
```

```
    {
        //MATRIZ A
        #pragma omp for schedule(runtime)
            for(i=0; i<N; i++){
                for(j=0; j<N; j++){
                    A[i][j]=0;
                }
            }


        //MATRIZ B
        #pragma omp for schedule(runtime)
            for(i=0; i<N; i++){
                for(j=0; j<N; j++){
                    B[i][j]=1;
                }
            }

        //MATRIZ C
        #pragma omp for schedule(runtime)
            for(i=0; i<N; i++){
                for(j=0; j<N; j++){
                    C[i][j]=1;
                }
            }
    }
    //Calculamos A = B*C y lo que se tarda en realizar esta operación
    #pragma omp parallel shared (A,B,C)
    {
        #pragma omp single
        {
            t_ini = 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++){
                        A[i][j] = A[i][j] + (B[i][j]*C[i][j]);
                    }
                }
            }

        #pragma omp single
        {
            t_fin = omp_get_wtime();
        }
    }

    t_total = t_fin-t_ini;

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

    //Liberamos el espacio reservado de las matrices A, B y C
    for(i=0; i<N; i++) {free(A[i]); free(B[i]); free(C[i]);}

    free(A);
    free(B);
    free(C);
```

```
    return 0;
}
```

CAPTURAS DE PANTALLA:



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

ESTUDIO DE ESCALABILIDAD EN atcgrid:

SCRIPT: pmm-OpenMP_atcgrid.sh

```
    #!/bin/bash
#Se asigna al trabajo el nombre pmm-OpenMP
#PBS -N pmm-OpenMP
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
#echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
#echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
#echo "Id. del trabajo: $PBS_JOBID"
#echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
#echo "Nodo que ejecuta qsub: $PBS_O_HOST"
#echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
#echo "Cola: $PBS_QUEUE"
#echo "Nodos asignados al trabajo:"
#cat $PBS_NODEFILE

for ((N=1;N<13;N=N+1))
do
   export OMP_NUM_THREADS=$N
   $PBS_O_WORKDIR/pmm-OpenMP 100
done

for ((N=1;N<13;N=N+1))
do
   export OMP_NUM_THREADS=$N
   $PBS_O_WORKDIR/pmm-OpenMP 1500
done
```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
    !/bin/bash
```

```
for((N=1; N<5; N=N+1))
do
    export OMP_NUM_THREADS=$N
    ./pmm-OpenMP 100
done

for ((N=1; N<5; N=N+1))
do
    export OMP_NUM_THREADS=$N
    ./pmm-OpenMP 1500
done
```