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): Raúl Rodríguez Pérez
Grupo de prácticas: C1

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 <time.h>
#include <omp.h>

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

  if(argc < 2)    {
    fprintf(stderr,"Error, if-clauseModificado iteraciones num_threads\n");
    exit(-1);
  }

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

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

  x = atoi(argv[2]);

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

```
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer1$ ./if-clauseModificado 8 3
 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 2 suma de a[6]=6 sumalocal=6
 thread 2 suma de a[7]=7 sumalocal=13
 thread 1 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[4]=4 sumalocal=7
 thread 1 suma de a[5]=5 sumalocal=12
thread master=0 imprime suma=28
```

```
                tid,i,a[i],sumalocal);
    }
    #pragma omp atomic
       suma += sumalocal;
    #pragma omp barrier
    #pragma omp master
       printf("thread master=%d imprime suma=%d\n",tid,suma);
  }

  return(0);
}
```

CAPTURAS DE PANTALLA:

RESPUESTA:

La cláusula num_threads() establece el número de hebras las cuales ejecutarán el código correspondiente. Como podemos ver en el ejemplo de la captura de pantalla, hemos ejecutado el código de if-clauseModificado, poniendo que se el for se ejecute 8 iteraciones y que sean 3 hebras las cuales lo ejecutan. Como apreciamos en el resultado, las diferentes ejecuciones de la suma local se han repartido entre las 3 hebras que hemos establecido que tiene. Destacamos que al ser un número impar de hebras, y un numero par de ejecuciones, una de las hebras realiza menos trabajo que las demás.

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

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

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

Iteración	schedule-clause.c			schedule-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
2	0	1	0	1	1	0	0	0	0
3	1	1	0	1	1	0	0	0	0
4	0	0	1	1	0	1	0	0	0
5	1	0	1	1	0	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	1	0	0	1	1	1
11	1	1	0	1	0	0	1	1	1
12	0	0	1	1	0	0	1	1	0
13	1	0	1	1	0	0	1	1	0
14	0	1	1	1	0	0	1	1	0
15	1	1	1	1	0	0	0	1	0

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

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

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

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

RESPUESTA:

En primer lugar, tenemos que el comportamiento de schedule con static, se basa en realizar un reparto estático, es decir, si vemos los ejemplos de las tablas, con 2 hebras y ejecutando con 1 chunk, el reparto se realiza delegando mitad de trabajo para cada una de las hebras. Y en el caso para 4 hebras y 1 chunk, el trabajo se reparte equitativamente entre dichas hebras.

Por otro lado, el comportamiento con dynamic varía en torno a cuanto tarda en realizar el trabajo cada hebra, es decir, se emplea dynamic en los casos en donde las iteraciones no usan un tiempo uniforme, sino que tienes distintos tiempos. Dichas distribuciones entra las hebras no se pueden predecir, ya que se realizan en tiempo de ejecución. Es por eso que los valores o resultados que nos han salido en las tablas, varían siempre que volvemos a ejecutar el script.

Finalmente, el comportamiento que adquiere schedule con guided, podriamos definirlo como, lo mismo

que en dynamic pero de manera mas inteligente, ya que trata de averiguar en tiempo de ejecución el reparto de chunk más óptimo. Destacar también que tanto dynamic como guided, añaden una sobrecarga al variar su reparto en tiempo de ejecución, pero en el caso de guided, la sobrecarga es menor para el mismo número de chunk que en dynamic.

Y para terminar, aclarar que static y dynamic utilizan el valor de chunk como el número de iteraciones que forman el tamaño del bloque, mientras que guided, lo utiliza como el valor del tamaño mínimo del 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?

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, modifier, a[n],suma=0;
  omp_sched_t schedule_type;
   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 for firstprivate(suma) \
           lastprivate(suma) schedule(dynamic,chunk)
   for (i=0; i<n; i++)
   {
     if(i == 0){
       omp_get_schedule(&schedule_type, &modifier);

       printf("Valores dentro de parallel for --> dyn-var = %d   nthreads-var = %d
          thread-limit-var = %d    run-sched-var = %d    chunk = %d\n", omp_get_dynamic(),
          omp_get_max_threads(), omp_get_thread_limit(), schedule_type, modifier);
     }
       suma = suma + a[i];
       //printf(" thread %d suma a[%d]=%d suma=%d \n",
       //      omp_get_thread_num(),i,a[i],suma);
   }

   //printf("Fuera de 'parallel for' suma=%d\n",suma);
   omp_get_schedule(&schedule_type, &modifier);

   printf("Valores fuera de parallel for --> dyn-var = %d   nthreads-var = %d
          thread-limit-var = %d    run-sched-var = %d    chunk = %d\n", omp_get_dynamic(),
          omp_get_max_threads(), omp_get_thread_limit(), schedule_type, modifier);

   return(0);
}
```

```
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer3$ export OMP_SCHEDULE="dynamic
,4"
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer3$ ./scheduled-clauseModificado
 16 2
Dentro de parallel for --> dyn-var = 0   nthreads-var = 12    thread-limit-var =
 2147483647    run-sched-var = 2    chunk = 4
Fuera de parallel for --> dyn-var = 0   nthreads-var = 12    thread-limit-var =
2147483647    run-sched-var = 2    chunk = 4
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer3$ export OMP_SCHEDULE="guided,
2"
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer3$ ./scheduled-clauseModificado
 10 1
Dentro de parallel for --> dyn-var = 0   nthreads-var = 12    thread-limit-var =
 2147483647    run-sched-var = 3    chunk = 2
Fuera de parallel for --> dyn-var = 0   nthreads-var = 12    thread-limit-var =
2147483647    run-sched-var = 3    chunk = 2
```

CAPTURAS DE PANTALLA:

RESPUESTA:

Se imprimen los mismos valores tanto dentro como fuera de la región paralela, la única variable que varía es "run-sched-var" debido a que, dicha variable establece la planificación de bucles para runtime, devolviendo un numero entero por cada uno de los tipos: static = '1', dynamic = '2', guided = '3'.

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, modifier, a[n],suma=0;
  omp_sched_t schedule_type;
   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 for firstprivate(suma) \
         lastprivate(suma) schedule(dynamic,chunk)
   for (i=0; i<n; i++)
```

```
.raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer4$ ./scheduled-clauseModificad
o2 20 4
Dentro de parallel for --> dyn-var = 0   nthreads-var = 12    thread-limit-var =
 2147483647    run-sched-var = 2        chunk = 1   num_threads = 12   num_procs
 = 12   in_parallel = 1
Fuera de parallel for --> dyn-var = 0   nthreads-var = 12    thread-limit-var =
2147483647    run-sched-var = 2        chunk = 1   num_threads = 1   num_procs =
 12   in_parallel = 0
```

```
  {
    if(i == 0){
      omp_get_schedule(&schedule_type, &modifier);
      printf("Dentro de parallel for --> dyn-var = %d   nthreads-var = %d    thread-limit-var = %d    run-sched-var = %d \
      chunk = %d   num_threads = %d   num_procs = %d   in_parallel = %d\n", \
      omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type, modifier,
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
    }
      suma = suma + a[i];
      //printf(" thread %d suma a[%d]=%d suma=%d \n",
      //      omp_get_thread_num(),i,a[i],suma);
  }

  //printf("Fuera de 'parallel for' suma=%d\n",suma);
  omp_get_schedule(&schedule_type, &modifier);
  printf("Fuera de parallel for --> dyn-var = %d   nthreads-var = %d    thread-limit-var = %d    run-sched-var = %d \
     chunk = %d   num_threads = %d   num_procs = %d   in_parallel = %d\n", \
     omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), schedule_type, modifier,
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

  return(0);
}
```

CAPTURAS DE PANTALLA:

RESPUESTA:

Fuera de la región parallel podemos observar que el omp_get:_numthreads() disminuye a 1, por otro lado el resto de funciones se mantienen igual tanto dentro como fuera de la región parallel.

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

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado5.c

```
 #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;
  omp_sched_t tipo;
  int x;

   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;

  printf("Antes de la modificacion:\n");
  printf("dyn-var = %d ; nthreads-var = %d\n", omp_get_dynamic(), omp_get_max_threads());
  omp_get_schedule(&tipo, &x);
  printf("run-sched-var = %d, %d\n", tipo, x);

  omp_set_dynamic(1);
  omp_set_num_threads(6);
  omp_set_schedule(1, 6);

  printf("\nDespues de la modificacion:\n");
  printf("dyn-var = %d ; nthreads-var = %d\n", omp_get_dynamic(), omp_get_max_threads());
  omp_get_schedule(&tipo, &x);
  printf("run-sched-var = %d, %d\n", tipo, x);

   #pragma omp parallel 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);

   return(0);
}
```

CAPTURAS DE PANTALLA:

```
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer5$ ./scheduled-clauseModificado5 20 5
Antes de la modificacion:
dyn-var = 0 ; nthreads-var = 12
run-sched-var = 2, 1

Despues de la modificacion:
dyn-var = 1 ; nthreads-var = 6
run-sched-var = 1, 6
 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[3]=3 suma=6
 thread 3 suma a[15]=15 suma=15
 thread 3 suma a[16]=16 suma=31
 thread 3 suma a[17]=17 suma=48
 thread 3 suma a[18]=18 suma=66
 thread 3 suma a[19]=19 suma=85
 thread 1 suma a[5]=5 suma=5
 thread 1 suma a[6]=6 suma=11
 thread 1 suma a[7]=7 suma=18
 thread 1 suma a[8]=8 suma=26
 thread 1 suma a[9]=9 suma=35
 thread 0 suma a[4]=4 suma=10
 thread 2 suma a[10]=10 suma=10
 thread 2 suma a[11]=11 suma=21
 thread 2 suma a[12]=12 suma=33
 thread 2 suma a[13]=13 suma=46
 thread 2 suma a[14]=14 suma=60
Fuera de 'parallel for' suma=85
```

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

int main(int argc, char **argv){
  int componentes;
  int *vec;
  int **m;
  int *res;
  int MAX = 9999;

  if (argc < 2){
    printf("ERROR: DEBE INTRODUCIR EL NUMERO DE COMPONENTES\n");
    return(-1);
  }

  componentes = atoi(argv[1]);

  if (componentes > MAX){
    printf("El tamaño de la m excede el máximo permitido");
    return(-1);
  }

  vec = (int *)malloc(componentes * sizeof(int));
  res = (int *)malloc(componentes * sizeof(int));
  m = (int **)malloc(componentes * sizeof(int *));

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


  for (int j=0; j<componentes; j++){
    for (int i=0; i<componentes;i++){
      if (j>i){
        m[j][i] = 0;
      }
      else{
        m[j][i] = j+i;
      }
    }
    vec[j] = i;
  }

  for (int i=0; i<componentes; i++){
    int suma=0;
    for (int j=0; j<componentes; j++){
      suma+=(m[j][i]*vec[i]);
    }
    res[i] = suma;
```

```
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer6$ ./producto-matriz-vector 8 8
Primer componente = 0
Ultimo componente = 672
```

```
  }

  printf ("Primer componente = %d\n",res[0]);
  printf ("Ultimo componente = %d\n",res[componentes-1]);

  free(m);
  free(res);
  free(vec);

  return 0;
}
```

CAPTURAS DE PANTALLA:

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:

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
   #include <omp.h>
#else
   #define omp_get_thread_num() 0
   #define omp_get_num_threads() 1
   #define omp_set_num_threads(int)
   #define omp_in_parallel() 0
   #define omp_set_dynamic(int)
#endif

int main(int argc, char **argv){
   struct timespec t_ini,t_fin; double t_total;
   int chunk_size;
   omp_sched_t kind;
   int *vec;
   int *res;
   int **m;
```

```
  if(argc < 2){
    fprintf(stderr, "ERROR: FALTA POR INTRODUCIR EL TAMANIO\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);
  vec = (int *) malloc(N*sizeof(int));
  res = (int *) malloc(N*sizeof(int));
  m = (int **) malloc(N*sizeof(int*));

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

  for (i=0; i<N; i++){
    for (j=i; j<N; j++){
      m[i][j] = 1;
    }
    vec[i] = 2;
    res[i]= 0;
  }


  clock_gettime(CLOCK_REALTIME,&t_ini);

  #pragma omp parallel for private(j) schedule(runtime)
  for (i=0; i<N; i++){
    for (j=i; j<N; j++){
      res[i] += m[i][j] * vec[j];
    }
  }

  clock_gettime(CLOCK_REALTIME,&t_fin);
  t_total=(double) (t_fin.tv_sec-t_ini.tv_sec)+(double) ((t_fin.tv_nsec-t_ini.tv_nsec)/(1.e+9));

  omp_get_schedule(&kind,&chunk_size);
  printf("run-sched-var: (Kind: %d, Modifier: %d) \n",kind,chunk_size);
  printf("Tiempo: \%11.9f\n",t_total);
  printf("Primer componente: %d, Ultimo componente: %d\n",res[0],res[N-1]);

  for (i=0; i<N; i++){
    free(m[i]);
  }
  free(m);
  free(vec);
  free(res);

  return 0;
}
```

DESCOMPOSICIÓN DE DOMINIO:
Cada thread recorre una fila diferente de la matriz, por lo que calcula un valor del vector final

CAPTURAS DE PANTALLA:

```
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer7$ ./pmtv-OpenMP 20
run-sched-var: (Kind: 2, Modifier: 1)
Tiempo: 0.001012408
Primer componente: 160, Ultimo componente: 8
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer7$ ./pmtv-OpenMP 80
run-sched-var: (Kind: 2, Modifier: 1)
Tiempo: 0.002179305
Primer componente: 640, Ultimo componente: 8
```

TABLA RESULTADOS, SCRIPT atcgrid SCRIPT

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N=15000 , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.001983251	0.148464790	0.158364734
1	0.145862139	0.148264283	0.151237489
64	0.152603834	0.149273849	0.151283049
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.002245674	0.148364859	0.151678956
1	0.148769086	0.148984940	0.151728394
64	0.148736274	0.149838495	0.151463738

SCRIPT ATCGRID

```
#!/bin/bash

echo "################################################################################"
echo "Id. usuario del trabajo: $SLURM_JOB_USER"
echo "Id. del trabajo: $SLURM_JOBID"
echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
echo "Directorio de trabajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
echo "Cola: $SLURM_JOB_PARTITION"
echo "Host de envío del trabajo:$SLURM_SUBMIT_HOST"
echo "Nº de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
echo "################################################################################"

export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
$SLURM_SUBMIT_DIR/pmtv-OpenMP 1500
$SLURM_SUBMIT_DIR/pmtv-OpenMP 1500

export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000

export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
```

```
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000

export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000

export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000

export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000

export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000

export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000

export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
$SLURM_SUBMIT_DIR/pmtv-OpenMP 15000
```

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

int main(int argc, char **argv){
  int **m1,**m2, **m3;

  if(argc < 2){
    fprintf(stderr, "ERROR FALTA INTRODUCIR EL TAMANIO\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);
  m1 = (int **) malloc(N*sizeof(int*));
  m2 = (int **) malloc(N*sizeof(int*));
  m3 = (int **) malloc(N*sizeof(int*));
```

```
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer8$ ./producto-matrices 5
Primer elemento = 0
Ultimo elemento = 80
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer8$
```

```
  for (int 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));
  }
  for (int i=0; i<N; i++){
    for (int j=0; j<N; j++){
      m1[i][j] = i;
      m2[i][j] = 0;
      m3[i][j] = j;
    }
  }

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

  printf("Primer elemento = %d\n", a[0][0]);
  printf("Ultimo elemento = %d\n", a[N-1][N-1]);

  for (int 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:

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:

Cada hebra recorre diferentes filas de la primera matriz, todas su columnas, y luego para la otra matriz todas sus filas y columnas

CAPTURA CÓDIGO FUENTE: pmm-OpenMP.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 **m1, **m2, **m3;

  if(argc < 2){
    fprintf(stderr, "Falta tamanio\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);
  m1 = (int **) malloc(N*sizeof(int*));
  m2= (int **) malloc(N*sizeof(int*));
  m3 = (int **) malloc(N*sizeof(int*));

  for (int 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));
  }

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

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


  printf("Primer elemento = %d\n",m1[0][0]);
  printf("Ultimo elemento = %d\n",m1[N-1][N-1]);

  for (int i=0; i<N; i++){
    free(m1[i]);
    free(m2[i]);
    free(m3[i]);
  }
  free(m1);
  free(m2);
  free(m3);

  return 0;
}
```

```
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer9$ ./producto-matrices-paralelo
 7
Primer elemento = 0
Ultimo elemento = 252
raulrguez@usuario-GF63-Thin-9SC:~/Escritorio/ejer9$
```

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

echo "################################################################################"
echo "Id. usuario del trabajo: $SLURM_JOB_USER"
echo "Id. del trabajo: $SLURM_JOBID"
echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
echo "Directorio de trabajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
echo "Cola: $SLURM_JOB_PARTITION"
echo "Host de envío del trabajo:$SLURM_SUBMIT_HOST"
echo "Nº de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
echo "################################################################################"

echo "paralelo 1 thread:"
export OMP_NUM_THREADS=1
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500

echo "paralelo 2 threads:"
export OMP_NUM_THREADS=2
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500

echo "paralelo 4 threads:"
export OMP_NUM_THREADS=4
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500
```

TABLA ATCGRID

Tamaño/threads	1 thread	2 thread	4 thread
100	0.002126411	0.001277373	0.001225568
500	0.383295482	1.275026644	1.100001745
1000	8.267114808	9.461235906	9.060441530
1500	14.274871209	19.781322127	20.781943701

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
#!/bin/bash

echo "paralelo 1 thread:"
export OMP_NUM_THREADS=1
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500

echo "paralelo 2 threads:"
export OMP_NUM_THREADS=2
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500

echo "paralelo 4 threads:"
export OMP_NUM_THREADS=4
./pmm-OpenMP 100
./pmm-OpenMP 500
./pmm-OpenMP 1000
./pmm-OpenMP 1500
```

TABLA PC/LOCAL

Tamaño/threads	1 thread	2 thread	4 thread
100	0.002944524	0.000332651	0.000168804
500	0.121812357	0.051172430	0.028308320

1000	1.015855766	0.466393597	0.231591890
1500	7.263051371	3.564431454	1.785296565