3.4 documentación de Francisco Joaquín Murcia Gómez

Francisco Joaquín Murcia Gómez

48734281H

Grado en ingeniería informática UA

Ingeniería de los computadores

Guía OpenMP

3.4 documentación de Francisco Joaquín Murcia Gómez

índice

Directivas

Parallel

Se utiliza para iniciar la programación paralela

```
#pragma omp parallel [clause[ [, ]clause] ...] new-line
        structured-block
```

Clausulas:

- if(scalar-expression)
- num_threads(integer-expression)
- default(shared | none)
- private(list)
- firstprivate(list)
- shared(list)
- copyin(list)
- reduction(operator: list)

For

Se utiliza para los bucles, especifica las iteracionas y como se ejecutara en cada hilo

```
#pragma omp for [clause[[,] clause] ... ] new-line
        for-loops
```

Clausulas:

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator: list)
- schedule(kind[, chunk_size])
- collapse(n)
- ordered
- nowait

```
#pragma omp for
for(i=0;i<(np*2);i++)
{
  printf("Thread %d, contador %d \n",iam,i);
}
```

3.4 documentación de Francisco Joaquín Murcia Gómez

Sections

Un conjunto de bloques que tienen que ser distribuidos y ejecutados por un equipo de hilos.

```
#pragma omp sections [clause[[,] clause] ...] new-line
        {
        [#pragma omp section new-line]
                structured-block
        [#pragma omp section new-line
                structured-block ]
        ...
        }
```

Clausulas:

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator: list)
- nowait

```
#pragma omp sections
{
  #pragma omp section
   printf("Soy el thread %d, en solitario en la seccion 1ª \n",iam);
  #pragma omp section
   printf("Soy el thread %d, en solitario en la sección 2ª \n",iam);
  #pragma omp section
   printf("Soy el thread %d, en solitario en la seccion 3ª \n",iam);
}//sections
```

Single

Especifica que el bloque asociado ha de ser ejecutado por un solo hilo (no tiene por qué ser la maestra)

```
#pragma omp single [clause[[,] clause] ...] new-line
        structured-block
```

Clausula:

- private(list)
- firstprivate(list)
- copyprivate(list)
- nowait

3.4 documentación de Francisco Joaquín Murcia Gómez

```
#pragma omp single                    {
   printf("Soy el thread %d, actuando en solitario dentro del primer
   bloque\n",iam);                      sleep(1);                    }
#pragma omp single                    {
   printf("Soy el thread %d, actuando en solitario dentro ddel
   segundo bloque \n",iam);                       sleep(1);
   }
```

Paralellel for/sections

Funcionan igual que parallel y for o sections pero en una sola línea:

```
#pragma omp parallel for [clause[[,] clause] ...] new-line
        for-loop
#pragma omp parallel sections [clause[ [, ]clause] ...]
        new-line
        {
        [#pragma omp section new-line]
                structured-block
        [#pragma omp section new-line
        structured-block ]
        ...
        }
```

Task

Define una tarea especifica

```
#pragma omp task [clause[ [, ]clause] ...] new-line
        structured-block
```

Clausulas:

- if(scalar-expression)
- untied
- default(shared | none)
- private(list)
- firstprivate(list)
- shared(list)

```
while(my_pointer)
{
    #pragma omp task firstprivate(my_pointer)
    {
        (void) do_independent_work (my_pointer);
    }
    my_pointer = my_pointer->next ;
}
```

Master

Similar al single, pero lo ejecuta la rama principal

```
#pragma omp master new-line
        structured-block
```

```
#pragma omp master                    {
   printf("Soy el thread %d, actuando en solitario dentro del primer
   bloque\n",iam);                      sleep(1);                    }
#pragma omp master                    {
   printf("Soy el thread %d, actuando en solitario dentro ddel
   segundo bloque \n",iam);                      sleep(1);
   }
```

Critical

Restringe la ejecución a un solo hilo

```
#pragma omp critical [(name)] new-line
        structured-block
```

```
#pragma omp critical
{
     printf("Soy el thread %d, al inicio de la seccion critica
     \n",iam);
     sleep(1);
     printf("\t\tSoy el thread %d, al final de la seccion critica
     \n",iam);
}
```

Barrier

Especifica un punto de unión de los hilos

```
#pragma omp barrier new-line
```

```
printf("Soy el thread %d, antes del barrier \n",iam);
#pragma omp barrier
printf("\t\tSoy el thread %d, despues del barrier \n",iam);
```

Taskwait

Especifica el lugar de la finalización de las tareas secundarias

```
#pragma omp taskwait newline
```

```
#pragma omp taskwait
fn = fnm1 + fnm2;
return(fn);
```

Atomic

Hace que un lugar de almacenamiento se actualiza automáticamente con lecturas y escrituras simultaneas

```
#pragma omp atomic new-line
        expression-stmt

expression-stmt: one of the following forms:
        x binop = expr
        x++
        ++x
        x--
        --x
```

Flush

Crea un hilo temporal

```
#pragma omp flush [(list)] new-line
```

Ordered

El bloque se ejecuta en el orden en que se ejecutaría de manera secuencial

```
#pragma omp ordered new-line
        structured-block
```

```
#pragma omp ordered
{
        printf("Soy el thread %d, actuando en la iteracion
%d\n",iam,i);
        sleep(1);                    }
}
```

Threadprivate

Especifica las variables que se replican, teniendo cada hilo una copia

```
#pragma omp threadprivate(list) new-line
```

```
#pragma omp threadprivate(x) //x privada a cada thread, no se copia valor del master
int main()  {
   int iam =0, np = 1, i=0,j=0;
   x=9999; // lo ponemos en el master
   #pragma omp parallel private(iam, np,i) copyin(x)
        //sin copyin en cada tread x=0, salvo el master
   {
        printf("Soy el thread %d, antes de actualizar, con x=%d \n",iam,x);
        x=1000+iam;
        printf("\t\tSoy el thread %d, despues de que actualice, con x=%d \n",iam,x);
   }//parallel

   printf("\n Despues de pragma parallel x=%d \n\n",x); //x=1000
}//main
```

Clausulas

Cláusula de compartición de datos

default(shared|none)

Controla el atributo por defecto de comparticion de datos para variables referenciasdas en el contructor de la región paralela o tarea

shared(list)

Declara uno o mas elementos que se comparten por las tareas generadas por el contructor de la región paralela

private(list)

Declaro uno o mas elementos privados para la tarea

firstprivate(list)

Declara uno o más elementos privados para la tarea, e inicializados cada una a su correspondiente valor original en el momento que se encuentra el constructor

lastprivate(list)

Declara uno o mas elementos privados implícitos a la tarea, y causa que el elemento original se actualiza después de terminar la región

reduction(operator:list)

Declara una acumulación a partir de una lista de elementos usando un operador asociativo indicado.

nowait

Evita la sincronización implícita de las hebras al terminar el bloque de una directiva

Cláusulas de copia de datos

copyin(list)

Copia los valores de las variables privadas del hilo maestro en cada una de las otras pertenecientes al equipo de ejecución de la región paralela

copyprivate(list)

Transmite el valor de un entorno de datos de una tarea a otro entorno de datos de las tareas pertenecientes a la región paralela

Schedule

Static

Se dividen las iteraciones en partes, esta se asigna a los hilos siguiendo el método de planificación round-robin sehun en los hilos

Dynamic

Cada hilo ejecuta una parte de las iteraciones, después pide otra parte así hasta que se acabe

Guided

Como la dinámica, pero las partes comienzan siendo grandes y se van reduciendo con forme se avanza

Auto

La planificación del reparto de tareas es decidida por el compilador

Runtime

El planificador y el tamaño de las partes las toma de run-sched-var

Rutinas

Rutinas de ejecución

set_num_threads

Indica el numero de hilos a utilizar una región paralela

```
void omp_set_num_threads(int num_threads);
```

get_num_threads

Obtiene el número de hilos que se están usando en una región paralela

```
int omp_get_num_threads(void)
```

get_max_threads

Obtiene la máxima cantidad posible de hilos

```
int omp_get_max_threads(void);
```

get_thread_num

Devuelve el número del hilo

```
int omp_get_thread_num(void);
```

get_num_procs

Devuelve el máximo número de procesadores que se pueden asignar al programa

```
int omp_get_num_procs(void);
```

3.4 documentación de Francisco Joaquín Murcia Gómez

omp_in_parallel(void);

Devuelve valor distinto de cero si se ejecuta dentro de una región paralela

```
int omp_in_parallel(void);
```

set_dynamic

Permite poner o quitar el que el número de threads se pueda ajustar dinámicamente en las regiones paralelas.

```
void omp_set_dynamic(void);
```

omp_get_dynamic(void);

Devuelve un valor distinto de cero si está permitido el ajuste dinámico del número de hilo

```
int omp_get_dynamic(void);
```

set_nested

Para permitir o desautorizar el paralelismo anidado

```
void omp_set_nested(int nested);
```

get_nested

Devuelve un valor distinto de cero si está permitido el paralelismo anidado

```
int omp_get_nested(void);
```

set_schedule

Afecta al planificador por defecto usado en las rutinas, modifica la variable run-sched-var

```
void omp_set_schedule(omp_sched_t kind, int modifier);
```

get_schedule

Devuelve el planificador aplicado en las rutinas de planificación

```
void omp_get_schedule(omp_sched_t *kind,int *modifier);
```

get_thread_limit

Devuelve el número máximo de hilos disponibles para el programa

```
int omp_get_thread_limit(void)
```

set_max_active_levels

Limita el numero regiones paralelas anidadas, modifica la variable max-active-levels-var (VIC).

```
int omp_get_max_active_levels(void);
```

get_level

Devuelve el número de regiones paralelas anidadas que contiene la llamada.

get_ancestor_thread_num

Devuelve, para un determinado nivel anidado de la hebra actual, el numero de hebra padre o al que actualmente pertenece

```
int omp_get_ancestor_thread_num(int level);
```

get_team_size

Devuelve, para un determinado nivel anidado de la hebra actual, el tamaño del equipo de la hebra padre o al que actualmente pertenece

```
int omp_get_team_size(int level);
```

get_active_level

Devuelve, para un determinado nivel anidado de la hebra actual, el numero de hebra del padre.

```
int omp_get_active_level(void);
```

Cerrojos

init_[nest]_lock

Esta rutina inicializa los cerrojos de OpenMP

```
void omp_init_lock(omp_lock_t *lock);
void omp_init_nest_lock(omp_nest_lock_t *lock);
```

destroy_[nest]_lock

Estas rutinas de asegurar que el bloqueo de OpenMP no está inicializado.

```
void omp_destroy_lock(omp_lock_t *lock);
void omp_destroy_nest_lock(omp_nest_lock_t *lock);
```

set_[nest]_lock

Estas rutinas proporcionan un medio de establecer un bloqueo de OpenMP.

```
void omp_set_lock(omp_lock_t *lock);
void omp_set_nest_lock(omp_nest_lock_t *lock);
```

unset_[nest]_lock

Estas rutinas proporcionan un medio de establecer un bloqueo de OpenMP.

```
void omp_unset_lock(omp_lock_t *lock);
void omp_unset_nest_lock(omp_nest_lock_t *lock);
```

test_[nest]_lock

Estas rutinas comprueban si el cerrojo esta bloqueado, no suspendiendo la ejecución de la tarea.

```
int omp_test_lock(omp_lock_t *lock);
int omp_test_nest_lock(omp_nest_lock_t *lock);
```

Rutinas de tiempo

get_wtime

Devuelve el valor del reloj en segundos.

```
double omp_get_wtime(void);
```

get_wtick

Devuelve la precisión del reloj

```
double omp_get_wtick(void);
```