

Tema 4.5 Programación mediante directivas OpenMP: Tareas

Computación de Altas Prestaciones

Carlos García Sánchez

10 de octubre de 2022

- “Using OpenMP : portable shared memory parallel programming”, Barbara Chapman, et all. 2008
- “OpenMP 5.2”, <https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf>



Outline

- 1 Tareas
- 2 Sincronización
- 3 Clausulas en tareas



¿Qué es una tarea?

- Tareas son unidades de trabajo cuya ejecución
 - puede ser diferidas o...
 - ... se puede ejecutar inmediatamente
- Tareas se componen de
 - **código** a ejecutar, los **datos**, variables de control **interno** (ICVs)
- Las tareas son creadas...
 - ... al llegar a una región paralela: tareas implícitas (por hilo)
 - ... cuando aparece una construcción *task*: tareas explícitas
 - ... cuando aparece una construcción *taskloop*: tareas explícitas por *chunk*
 - ... cuando aparece una construcción *target*: se crea la tarea de destino



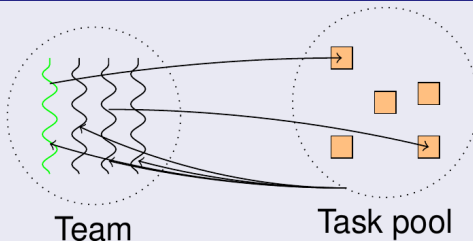
Modelo de ejecución

- Soporta paralelismo no-estructurado
 - Bucles sin límites definidos: `while (<expr>) {...}`
 - Funciones recursivas: `void myfunc(<args>){...; myfunc(<newargs>); ...;}`
- Varios escenarios posibles
 - Creador único, multiples creadores, tareas anidadas...
- Todos los hilos de un “equipo” son candidatos a ejecutar una **tarea**



Modelo de paralelismo de tareas con OpenMP

Task parallelism



- Paralelismo extraído en secciones de código independiente
- Permite para explotar paralelismo no-estructurado
 - Bucles sin límites, funciones recursivas

Modelo de paralelismo de tareas con OpenMP

¿Que es un tarea en OpenMP?

- Tareas = unidades de trabajo (ejecución puede diferirse)
 - También pueden ser ejecutados inmediatamente
- Las tareas se componen de:
 - código para ejecutar
 - un entorno de datos
 - Inicializado en el momento de la creación
 - variables de control interno (ICVs)
- Hilos pueden **cooperar** para ejecutarlas



Construcción task

- `#pragma omp task [clauses]`

Clausulas Datos

- *shared*
- *private*
- *firstprivate*
- *default(shared|none)*
- *in_reduction*

Sincronización

- *depend(dep-type:list)*



¿Cuándo se crean las tareas?

- En una región **parallel**
 - Una tarea implícita se crea para ser asignada al thread
- Cada hilo que entra en la construcción **task**
 - Necesita el código y datos para esa tarea
 - Se crea una nueva tarea específicamente



Compartición de datos entre tareas

Si no hay clausula específica

- Se usan las reglas por defecto
 - Ej: variables globales como *shared*, inicializadas antes de la construcción como *firstprivate*



example_task.c

```
int a;
void foo(){
    int b, c;
    #pragma omp parallel shared(c)
    #pragma omp parallel firstprivate(b)
    {
        int d;
        #pragma omp task
        {
            int e;
            a = //shared
            b = //firstprivate
            c = //shared
            d = //firstprivate
            e = //private
        }
    }
}
```



Planificación de tareas: tied vs untied tasks



- Las tareas son del tipo *tied* por defecto
 - Estas tareas siempre se ejecutan por el mismo hilo
 - Pero puede haber problemas en tareas con características basadas en hilos thread-id, regiones críticas...
- El programador puede especificar tareas como *untied* (planificación relajada)
 - `#pragma omp task untied`
 - Pueden migrar entre hilos



Tied vs untied tasks

- Para indicar puntos de planificación se usa directiva *taskyield*
 - Tareas se pueden suspender (algunos requisitos extra evitan deadlocks)
 - Se puede indicar explícitamente con directiva `#pragma taskyield`



taskyield.c

```
#pragma omp parallel
#pragma omp single
{
    #pragma omp task untied
    {
        foo();
        #pragma omp taskyield
        bar()
    }
}
```



Sincronización de tareas

- Existen dos contrucciones básicas:
 - **barrier**: esperar hasta que lo anterior se complete
 - **taskwait**



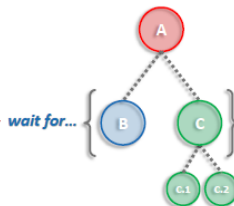
Construcción taskwait



```
#pragma omp taskwait
```

- Suspende la tareas hasta que todos los **hijos** completen la tarea

```
#pragma omp parallel
#pragma omp single
{
    #pragma omp task :A
    {
        #pragma omp task :B
        { ... }
        #pragma omp task :C
        { ... #C.1; #C.2; ... }
        #pragma omp taskwait
    }
} // implicit barrier will wait for C.x
```



Reducciones

- Operación de reducción
- Dos directivas
 - `#pragma omp taskgroup task_reduction(op: list)` registra una reducción en [1], y computa el resultado final después de [3]
 - `#pragma omp task in_reduction(op: list)` para participar en la operación de reducción en [2]

reduction.c

```
int res = 0;
node_t* node = NULL;
...
#pragma omp parallel
{
    #pragma omp single
    {
        #pragma omp taskgroup task_reduction(+: res)
        { // [1]
            while (node) {
                #pragma omp task in_reduction(+: res) \
                    firstprivate(node)
                { // [2]
                    res += node->value;
                }
                node = node->next;
            }
        } // [3]
    }
}
```



Planificación tareas *if*

- Si la cláusula *if* se evalúa como falsa
 - Se suspende la tarea
 - La nueva tarea **se ejecuta inmediatamente**
 - con su propio entorno de datos
 - diferente tarea con respecto a la sincronización.

