

Introduction to OpenMP

Dr. Christian Terboven



Task Scheduling

Dr. Christian Terboven

Introduction to OpenMP



- Default: Tasks are *tied* to the thread that first executes them → not necessarily the creator. Scheduling constraints:
 - Only the thread a task is tied to can execute it
 - A task can only be suspended at task scheduling points
 - Task creation, task finish, `taskwait`, `barrier`, `taskyield`
 - If task is not suspended in a barrier, executing thread can only switch to a direct descendant of all tasks tied to the thread
- Tasks created with the `untied` clause are never tied
 - Resume at task scheduling points possibly by different thread
 - No scheduling restrictions, e.g., can be suspended at any point
 - But: More freedom to the implementation, e.g., load balancing



- Problem: Because `untied` tasks may migrate between threads at any point, thread-centric constructs can yield unexpected results
- Remember when using `untied` tasks:
 - Avoid `threadprivate` variable
 - Avoid any use of thread-ids (i.e., `omp_get_thread_num()`)
 - Be careful with `critical` region and *locks*
- Simple Solution:
 - Create a tied task region with

```
#pragma omp task if(0)
```



- The `taskyield` directive specifies that the current task can be suspended in favor of execution of a different task.
 - Hint to the runtime for optimization and/or deadlock prevention

C/C++

```
#pragma omp taskyield
```

Fortran

```
!$omp taskyield
```

taskyield Example (1/2)

```
#include <omp.h>

void something_useful();
void something_critical();

void foo(omp_lock_t * lock, int n)
{
    for(int i = 0; i < n; i++)
        #pragma omp task
        {
            something_useful();
            while( !omp_test_lock(lock) ) {
                #pragma omp taskyield
            }
            something_critical();
            omp_unset_lock(lock);
        }
}
```




taskyield Example (2/2)

```
#include <omp.h>

void something_useful();
void something_critical();

void foo(omp_lock_t * lock, int n)
{
    for(int i = 0; i < n; i++)
        #pragma omp task
        {
            something_useful();
            while( !omp_test_lock(lock) ) {
                #pragma omp taskyield
            }
            something_critical();
            omp_unset_lock(lock);
        }
}
```



The waiting task may be suspended here and allow the executing thread to perform other work; may also avoid deadlock situations.

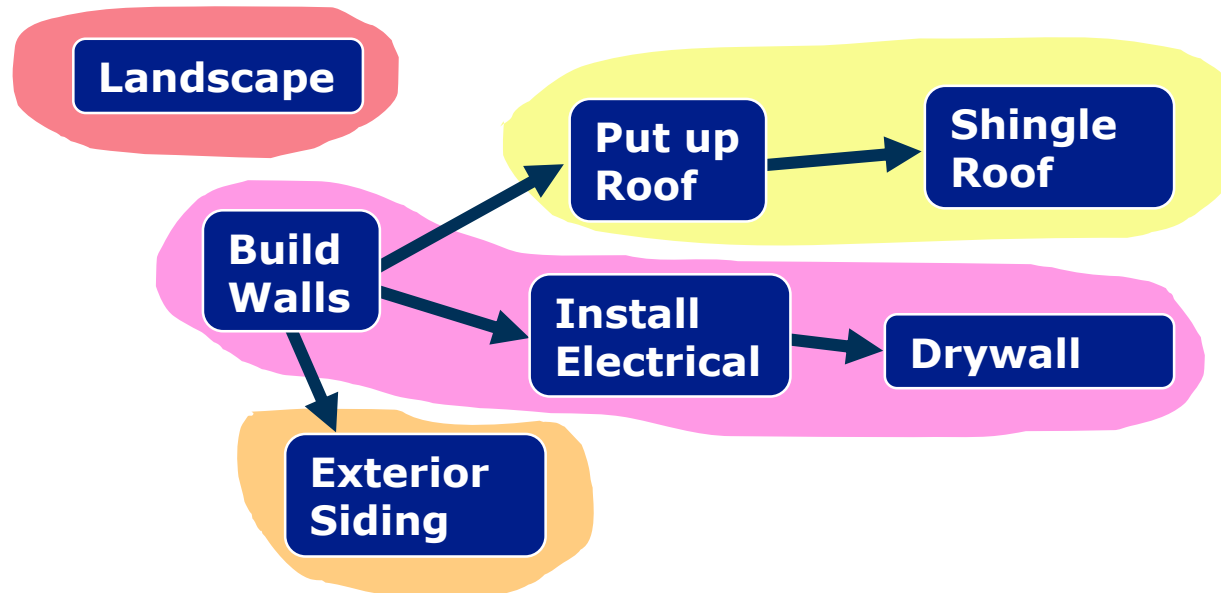
Tasks and Dependencies

Dr. Christian Terboven

Introduction to OpenMP



- Catchy example: Building a house



- Task dependencies constrain execution order and times for tasks
- Fine-grained synchronization of tasks

```
int x = 0;
#pragma omp parallel
#pragma omp single
{
  ● #pragma omp task
  std::cout << x << std::endl;

  #pragma omp taskwait

  ● #pragma omp task
  x++;
}
```

Traditional task wait

```
int x = 0;
#pragma omp parallel
#pragma omp single
{
  ● #pragma omp task depend(in: x)
  std::cout << x << std::endl;

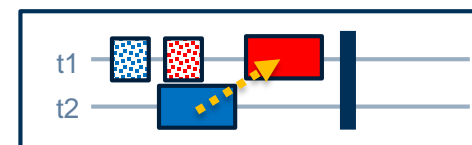
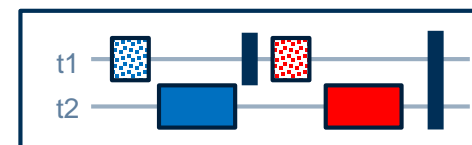
  ● #pragma omp task depend(inout: x)
  x++;
}
```



Dependencies

Task wait



Dependencies



 Task's creation time
 Task's execution time



Questions?

