OpenMP Lab on Nested Parallelism and Tasks
Nested Parallelism
**Nested Parallelism**

- Some OpenMP implementations support nested parallelism
  - A thread within a team of threads may fork spawning a child team of threads

*Image Source: John Mellor-Crummey's Lecture Notes*
More about Nested Parallelism

- It is possible to enable/disable nested parallelism via OpenMP’s internal control variable `nest-var`
  - set either the `OMP_NESTED` environment variable or invoke the library routine `omp_set_nested()`
  - If nested parallelism is enabled, when a thread on a team within a parallel region encounters a new parallel construct, an additional team of threads is forked off of it, and it becomes their master
  - If nested parallelism is disabled, when a thread on a team within a parallel region encounters a new parallel construct, execution continues on this additional single thread only
- The Sun OpenMP implementation provides additional environment variables for controlling nested parallelism:
  - `SUNW_MP_MAX_POOL_THREADS`: Sets limit on the number of threads in the pool that OpenMP maintains for use as slave threads in parallel regions. The default value is 1023.
  - `SUNW_MP_MAX_NESTED_LEVELS`: Sets limit on the number of levels of nesting of parallel regions. Default value is undefined.
**Behaviour of OpenMP Library Functions in Nested Parallel Regions**

- In particular, consider the following OpenMP library calls
  - `omp_set_num_threads()`
  - `omp_get_max_threads()`
  - `omp_set_dynamic()`
  - `omp_get_dynamic()`
  - `omp_set_nested()`
  - `omp_get_nested()`

  - **set** methods affect only parallel regions at same or inner nesting levels encountered by the calling thread
  - **get** methods return values set by the calling thread; upon creation of a team of threads, slave threads inherit values from the master thread.
**Tips on Using Nested Parallelism**

- Obviously, nesting parallel regions is an immediate way to engage more threads in a computation.
- Beware! Nesting parallel regions easily can create large number of threads as their number is the product of the number of threads forked at each level of nested parallelism. This can easily oversubscribe the system. Impose some discipline by setting appropriate values of `SUNW_MP_MAX_POOL_THREADS` and `SUNW_MP_MAX_NESTED_LEVELS`.
- Creating any parallel region will entail some overhead. Overhead from nesting of parallel regions may incur overheads greater than necessary if, for example an outer region could simply employ more threads in a computation. Examples of good reasons to employ nested parallelism are:
  - Insufficient parallelism at outer level
  - Load balance problems
  - Part of a larger parallel region where other code blocks are not sufficiently parallel to warrant more threads.
OpenMP Tasks

- New to OpenMP 3.0
- Introduced to widen the applicability of the OpenMP programming model beyond loop-level and functional parallelism
  - Unbounded loops
  - Recursion
  - Producer/consumer
  - List and tree traversal
- Tasks support parallelisation of irregular problems
- Tasks are work units whose execution may be deferred or performed immediately
- Syntax: `#pragma omp task [clauses...]` above a structured block
- The `task` construct defines an explicit task, and is modified by the following clauses:
  - previously defined `{if, default, private, firstprivate, shared}`
  - `untied`: Controls binding of task to threads
- Tasks may be nested inside parallel regions, other tasks, and other work sharing constructs
Elements of a Task

- Three key elements
  - Code to execute
  - Data (it literally owns its data)
  - An assigned thread that executes the code on the task’s data

- Task processing
  - Each thread encountering a thread construct packages a new instance of a task
  - Some thread in the team executes the task at some later time

- Some other terminology:
  - Task construct: the `task` directive plus the structured block
  - Task: the code plus instructions for allocating data created when a thread encounters a task construct
  - Task region: the dynamic sequence of instructions produced by execution of a task by a thread
Example - Parallel Pointer Chasing Using Tasks

```
#pragma omp parallel
{
    #pragma omp single private(p)
    {
        p = listhead ;
        while (p) {
            #pragma omp task
            process (p)
            p=next (p) ;
        }
    }
}
```

spawn to call process(p)

implicit taskwait synch

point here
Task switching

- In the code example below, tasks are generated, and lots of them!
- Eventually, the generating task is **suspended** and executing thread shifts to performing long task that was previously created
- Other threads work through previously generated tasks and start to starve for work
- With thread switching, the generating task resumes and makes new work

```c
#pragma omp single
{
    #pragma omp task untied
    for (i=0; i<ONEZILLION; i++)
        #pragma omp task
} 
```