ICHEC
Irish Centre for High-End Computing

New Features after OpenMP 2.5

OpenMP

OpenMP Specifications

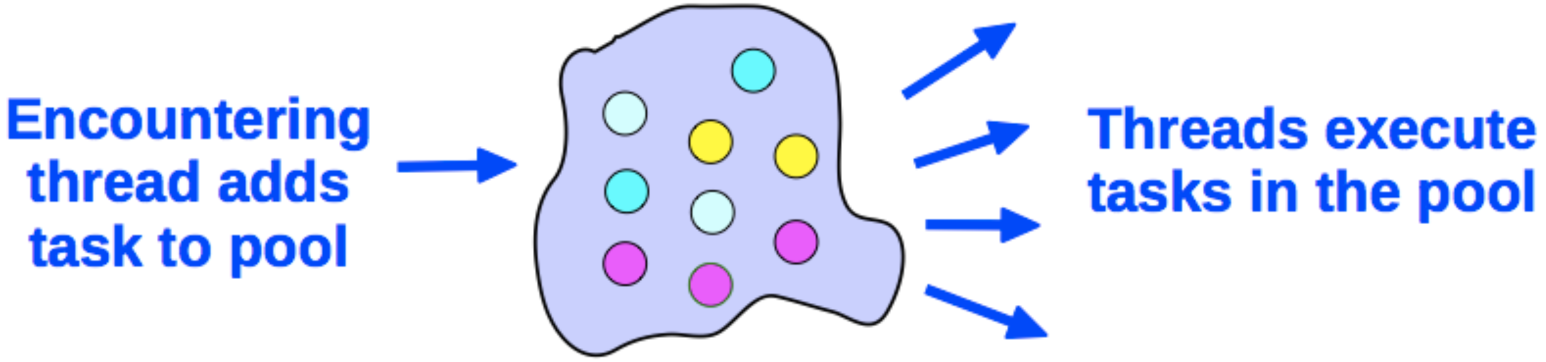
- Version 3.0 released in May 2008
  - New task level parallelism
  - Improvements to loop and nested parallelism
  - Additional Clauses, runtime functions and environment variables

- Version 3.1 released in July 2011
  - Additional Clauses
  - Improvements to task parallelism
  - Initial support for thread binding

Version 3.0 - Task Parallelism

- New feature in OpenMP
- Tasks: Work units executed by the encountering thread or deferred for execution by any other thread.
- Tasks are composed of
  - Code
  - Data
  - ICVs

- Two activities: packaging and execution


Encountering thread adds task to pool
Threads execute tasks in the pool

Task Construct

C/C++:

```
#pragma omp task [clauses]
{
   …
}
```

Fortran:

```
!$omp task [clauses]
   …
!$omp end task
```

- Clauses: if, untied, data clauses

ICHEC
Irish Centre for High-End Computing

Data Clauses:

- Implicit rules apply
- Otherwise …

C:

default(shared|none), private(list), firstprivate(list), shared(list)

Fortran:

default(private|firstprivate|shared|none), private(list), firstprivate(list), shared(list)

Nesting:

- Can be nested
  - Inside parallel regions
  - Inside other tasks
  - Inside work-sharings

Task Synchronisation:

- taskwait: #pragma omp taskwait
  !$omp taskwait
- Task behaviour
- Barrier: #pragma omp barrier
  !$omp barrier
- Implicit barrier

Fibonacci Sequence:

fib(0)=0

fib(1)=1

fib(n)=fib(n-1)+fib(n-2), n>1

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

```
#pragma omp parallel num_threads(n)
{
#pragma omp task
   function_A();
#pragma omp barrier
#pragma omp single
   {
   #pragma omp task
   function_B();
   }
}
```

```
int main(){
   int n=30;

   omp_set_dynamic(0);
   omp_set_num_threads(4);

   #pragma omp parallel shared(n)
   {
      #pragma omp single
      printf("fib(%d)=%d\n", n,
fib(n));
   }
}
```

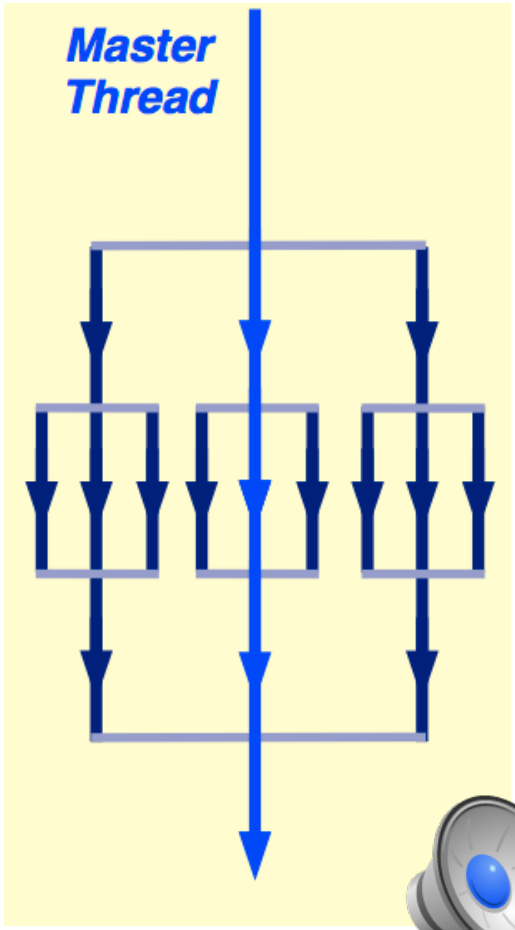
```
int fib(int n)
{
int i, j;
if(n<2) return n;
else{
   #pragma omp task shared(i)
firstprivate(n)
   i=fib(n-1);
   #pragma omp task shared(j)
firstprivate(n)
   j=fib(n-2);

   #pragma omp taskwait
   return i+j;
}}
```

Irish Centre for High-End Computing

Version 3.0 - Nested parallelism

- ***Recall:*** *Parallel regions can be nested, but support for this is implementation dependent*
- Better support for nested parallelism in v3.0
- New library routines, environment variables, multiple internal control variables



- Control maximum number of active parallel regions:

  OMP_MAX_ACTIVE_LEVELS
  omp_set_max_active_levels()
  omp_get_max_active_levels()

- Control maximum number of OpenMP threads:

  OMP_THREAD_LIMIT

  omp_get_thread_limit()

- To obtain information about nested parallelism:

  omp_get_level(): How many nested parallel regions at this point?

ICHEC
Irish Centre for High-End Computing

omp_get_active_level(): How many active (with 2 or more threads) regions?

omp_get_ancestor_thread_num(level): Which thread-id was my ancestor?

omp_get_team_size(level): How many threads there are at a previous regions?

- Multiple ICVs:

  Allows omp_set_num_threads() inside a parallel region

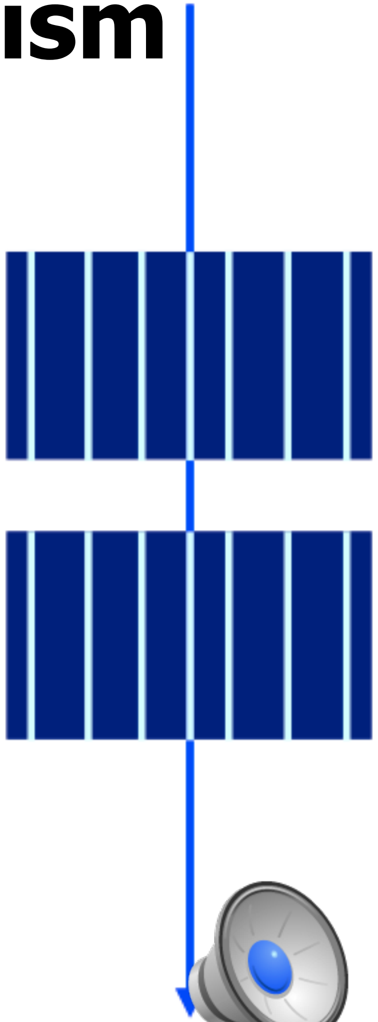
```
#pragma omp parallel num_threads(3)
omp_set_num_threads(omp_get_thread_num()+2);
#pragma omp parallel
  foo();
```

Version 3.0 - Loop parallelism

- ***Recall:*** *The iterations are distributed over team threads.*

- schedule(static):

```
#pragma omp for nowait schedule(static)
for(i=0;i<n;i++)
   a(i)=…
#pragma omp for schedule(static)
for(i=0;i<n;i++)
   a(i)=…
```



- schedule(runtime): omp_set_schedule()
  omp_get_schedule()

- AUTO schedule:

```
#pragma omp parallel for schedule(auto)
for(i=0;i<n;i++)
   …
```

- COLLAPSE clause:

```
#pragma omp parallel for collapse(2)
for(i=0;i<n;i++)
   for(j=0;j<m;j++)
      …
```

ICHEC
Irish Centre for High-End Computing

Version 3.0 - Other Features

- Additional Environment Variables:

  OMP_STACK_SIZE *size [B|K|M|G]*: control of children therad's stack size

  OMP_WAIT_POLICY *[ACTIVE|PASSIVE]*: control of thread's idle behaviour

- Minor fixes and clarifications to Version 2.5

- See OpenMP Specifications for Version 3.0 in openmp.org

Version 3.1 - New Features

- New atomics support capture and write functionality
- Modifications to data environment: intent(in), const-qualified
- Initial support for thread binding: OMP_PROC_BIND
- Extensions to OpenMP tasking model
  - taskyield Construct:

C/C++:

```
#pragma omp taskyield
```

Fortran:

```
!$omp taskyield
```

– final and mergeable Clauses:

- Undeferred task: a task for which execution is not deferred with respect to its generating task region
- Included task: an undeferred task that is sequentially included in generating task region
- Merged task: a task that has the same data environment as that of its generating task region. (mergeable)
- Final task: a task that makes all its child tasks become final and included tasks. (final(*expression*))

– omp_in_final()