ICHEC
Irish Centre for High-end Computing

PRACE

OpenMP: More Features

SoHPC Oisín Robinson

ICHEC
Irish Centre for High-end Computing

Outline

PRACE

- Tasking in OpenMP
- Vectorisation Support
- Cancellation
- User-defined Reduction

More… OpenMP Specifications:
(http://www.openmp.org/specifications/)

Tasking in OpenMP …(1)

- Tasks are independent units of work.
- A task can be executed by any thread in the team, in parallel with others
- Tasks:

  - execution can be immediate or deferred until later

  - execution might be suspended and continued later by the same or different thread

- For parallelizing irregular problems, unbounded loops, recursive algorithms, multi-block grids and trees, …
- OpenMP has always had tasks, never called them explicitly that.

Tasking in OpenMP …(2)

- Tasks are composed of code to execute, data environment, internal control variables
- Parallel tasks can be nested within parallel loops or sections. They may run in parallel to the parent or suspend and run synchronously.

C/C++:

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

Fortran:

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

Scheduling Points:

- taskwait: Waits for all child tasks to be completed by the encountering thread.

  C/C++: #pragma omp taskwait

  Fortran: !$omp taskwait

- omp barrier (implicit or explicit): All tasks created by any thread of the current team are guaranteed to be completed.

Tasking in OpenMP …(3)

```
void process_list(dList *myListptr){
 #pragma omp parallel
 {
  #pragma omp single
  {
   while(myList != NULL){
    #pragma omp task firstprivate(myListptr)
    {
     process(myListptr);
    }
    myListptr = myListptr ->next;
   }
  }
 }
}
```

- Inside parallel region
- One thread adds all tasks to the task queue
- Available thread picks task from the queue

Tasking in OpenMP …(4)

- Clauses: untied, shared, private, firstprivate, default, if
  - tied by default: The same thread from beginning to end will execute the code. It can only be suspended at task scheduling points.
  - untied: Tasks are never tied, different threads execute different parts of the code
  - When if is false, the encountering task is suspended and anew task is executed immediatlely.
- mergeable: a task can have the same data region as the generating task region
- final: a task that makes all its child tasks to be executed sequentially in the same region

Tasking in OpenMP …(5)

- taskyield: task scheduling point at which the current task may be suspended and resumed later.

  C/C++: #pragma omp taskyield

  Fortran: !$omp taskyield

- taskgroup: marks a region such that all tasks started in it belong to a group, does not create a task region; only for group and synchronization purposes.

  C/C++: #pragma omp taskgroup

  Fortran: !$omp taskgroup

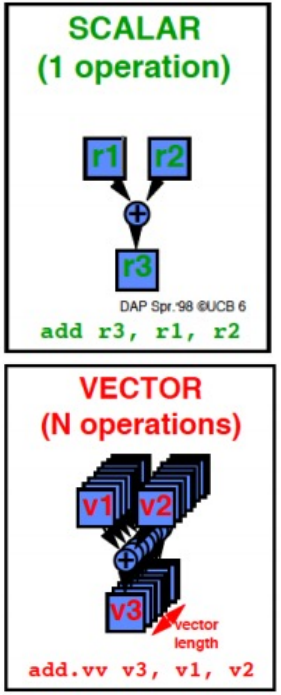
- task depend (dependence-type: list): clause marks that a task depends on other task; type: in, out, inout; list items: can be array sections and must be identical storage or disjoint storage

Vectorisation Support

- Vectorisation: execute a single instruction on multiple data objects in parallel within a single CPU core
- More architectures support longer vector length
- Many compilers exploit vector parallelism which is limited due to dependencies, inner loops, function calls etc.
- Simd construct in OpenMP to support vectorisation

Single Instructions operating on multiple data

- Higher performance and Energy Efficiency



Cancellation …(1)

- Clean way to signal early termination of an OpenMP construct. (break)

  one thread signals

  other threads jump to the end of the construct

- cancel construct: Cancellation activated.

C/C++:
#pragma omp cancel *[construct]*

Fortran:
!$omp cancel *[construct]*

- Clauses: parallel, sections, for/do, taskgroup, if

```
#pragma omp parallel for private(a)
for(i=0; i<N; i++){
  a=testing(i, ...)
  #pragma cancel parallel if(a)
}
```

Cancellation …(2)

- cancellation point construct: allow users to explicitly define a cancellation point at which a check is done if cancellation has been requested, then, cancellation is performed.

C/C++:
#pragma omp cancelltionpoint *[construct]*

Fortran:
!$omp cancellationpoint *[construct]*

- Implicit cancellation point at: cancel, implicit/explicit barriers
- Cancellation is performed if OMP_CANCELLATION is set to true.
- omp_get_cancellation(void): returns if cancellation is activated.

Looking further ahead

- Multiple cores/CPUs dominate the future computer architectures; OpenMP would be the major parallel programming language in these architectures.
- OpenMP is a mature environment.
- From early loop parallelism to a accelerators support, OpenMP has evolved to address requirements for new generation of hardware.
- OpenMP is changing to meet the future needs of Exascale systems.