Introduction to OpenMP

Lecture 2: OpenMP Fundamentals

epcc

THE UNIVERSITY OF EDINBURGH

Overview

- Basic Concepts in OpenMP
- Compiling and running OpenMP programs

What is OpenMP?

- OpenMP is an API designed for programming shared memory parallel computers.
- OpenMP uses the concepts of *threads* and *tasks*
- OpenMP is a set of extensions to Fortran, C and C++
- The extensions consist of:
  - Compiler directives
  - Runtime library routines
  - Environment variables

Directives and sentinels

- A directive is a special line of source code with meaning only to certain compilers.
- A directive is distinguished by a sentinel at the start of the line.
- OpenMP sentinels are:
  - Fortran: `!$OMP`
  - C/C++: `#pragma omp`
- This means that OpenMP directives are ignored if the code is compiled as regular sequential Fortran/C/C++.

Parallel region

- The *parallel region* is the basic parallel construct in OpenMP.
- A parallel region defines a section of a program.
- Program begins execution on a single thread (the master thread).
- When the first parallel region is encountered, the master thread creates a team of threads (fork/join model).
- Every thread executes the statements which are inside the parallel region
- At the end of the parallel region, the master thread waits for the other threads to finish, and continues executing the next statements

Parallel region

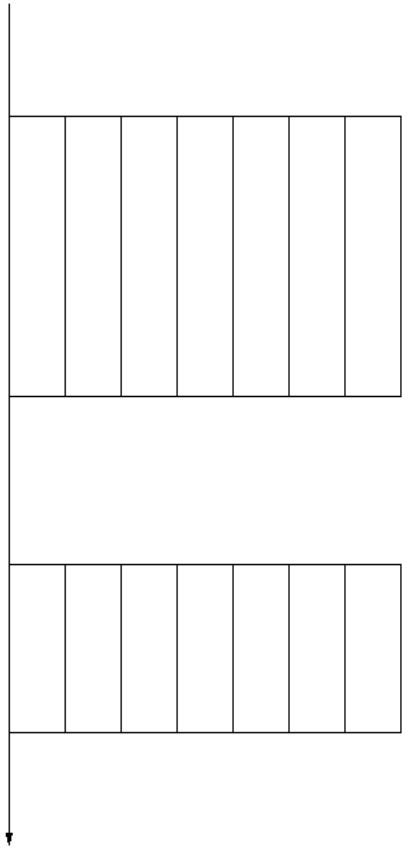
Sequential part

Parallel region

Sequential part

Parallel region

Sequential part



```
      PROGRAM FRED
        .
        .
!$OMP PARALLEL
        .
        .
        .
        .
        .
        .
        .
        .
!$OMP END PARALLEL
        .
        .
        .
        .
!$OMP PARALLEL
        .
        .
        .
        .
!$OMP END PARALLEL
        .
        .
        .
```

```
int main(){
.
.
#pragma omp parallel
{
.
.
.
.
.
.
.
.
}
.
.
.
.
#pragma omp parallel
{
.
.
.
}
.
.
.
```

Shared and private data

- Inside a parallel region, variables can either be *shared* or *private.*
- All threads see the same copy of shared variables.
- All threads can read or write shared variables.
- Each thread has its own copy of private variables: these are invisible to other threads.
- A private variable can only be read or written by its own thread.

Parallel loops

- In a parallel region, all threads execute the same code
- OpenMP also has directives which indicate that work should be divided up between threads, not replicated.
  - this is called worksharing
- Since loops are the main source of parallelism in many applications, OpenMP has extensive support for parallelising loops.
- The are a number of options to control which loop iterations are executed by which threads.
- It is up to the programmer to ensure that the iterations of a parallel loop are *independent*.
- Only loops where the iteration count can be computed before the execution of the loop begins can be parallelised in this way.

Synchronisation

- The main synchronisation concepts used in OpenMP are:
- Barrier
  - all threads must arrive at a barrier before any thread can proceed past it
  - e.g. delimiting phases of computation
- Critical region
  - a section of code which only one thread at a time can enter
  - e.g. modification of shared variables
- Atomic update
  - an update to a variable which can be performed only by one thread at a time
  - e.g. modification of shared variables (special case)

Compiling and running OpenMP programs

- OpenMP is built-in to most of the compilers you are likely to use.
- To compile an OpenMP program you need to add a (compiler-specific) flag to your compile and link commands.
  - `-fopenmp` for gcc/gfortran
  - `-openmp` for Intel compilers
  - on by default in Cray compilers
- The number of threads which will be used is determined at runtime by the `OMP_NUM_THREADS` environment variable
  - set this before you run the program
  - e.g. `export OMP_NUM_THREADS=4`
- Run in the same way you would a sequential program
  - type the name of the executable

Reusing this material