# A Brief Introduction to OpenMP

Will Knottenbelt

Imperial College London

*wjk@doc.ic.ac.uk*

February 2015



## Recommended Reading

- OpenMP FAQ
  `http://openmp.org/openmp-faq.html`
- OpenMP on Wikipedia
  `http://en.wikipedia.org/wiki/OpenMP`
- OpenMP Tutorial
  `http://openmp.org/mp-documents/omp-hands-on-SC08.pdf`



## Outline





## Supercomputer Evolution

- Mainstream supercomputers of the 1990s tended to feature single core, single processor nodes with specialised interconnects.
- Imperial took delivery of a Fujitsu AP3000 supercomputer in 1997, now already a museum piece:
  `http://museum.ipsj.or.jp/en/computer/super/0013.html`
- Modern supercomputers feature multi-core, multi-processor nodes with specialised interconnects, see: `http://www.top500.org`
- Clear need for parallelisation mechanism directly targetting multicore shared-memory environments.

## What is OpenMP

- OpenMP is a specification for a set of compiler directives, library routines, and environment variables for specifying shared-memory parallelism
- A primary design goal was to take away the pain of programming multithreaded applications and increase their portability
- C/C++ and Fortran supported
- Evolution directed by the OpenMP Architecture Review Board



## Using OpenMP

- Supports incremental parallelisation of sequential code via addition of compiler directives. So

```
int main() {
  cout << "hello world" << endl;
  return 0;
}
```

becomes:

```
#include <omp.h>
int main() {
  #pragma omp parallel
  { cout << "hello world" << endl; }
  return 0;
}
```



## Using OpenMP (cont.)

- Support built into gcc/g++:

```
g++ omp_basic_hello.cpp -o omp_basic_hello -fopenmp
```

- Default number of threads controlled by environment variable `OMP_NUM_THREADS` (use `setenv` or `export` to set depending on your shell)
- Execute as normal:

```
./omp_basic_hello
```



## Using OpenMP (cont.)

- In addition to parallel constructs there are various useful runtime routines e.g.:

```
void omp_set_num_threads(int num_threads);
int omp_get_num_threads();
int omp_get_thread_num();
int omp_in_parallel();
double omp_get_wtime();
```

## Using OpenMP (cont.)

```
int main(int argc, char *argv[])
{
  int th_id, nthreads;
  #pragma omp parallel private(th_id) shared(nthreads)
  {
    th_id = omp_get_thread_num();
    #pragma omp critical
    { cout << "Hello World from thread " << th_id << '\n'; }
    #pragma omp barrier
    #pragma omp master
    {
      nthreads = omp_get_num_threads();
      cout << "There are " << nthreads << " threads" << '\n';
    }
  }
  return 0;
}
```



## Using OpenMP (cont.)

- For loops can be scheduled in parallel, in a dynamic or static fashion:

```
#pragma omp for schedule(dynamic,chunk)
for (i=0; i<N; i++) {
  c[i] = a[i] + b[i];
}
return 0;
}
```

- Reductions are possible:

```
double ave=0.0, A[MAX]; int i;
#pragma omp parallel for reduction (+:ave)
for (i=0;i< MAX; i++) {
  ave + = A[i];
}
ave = ave/MAX;
```



## OpenMP vs MPI

- OpenMP is a predominantly implemented as a compiler extension; MPI is implemented as a library of functions.
- OpenMP uses threads, MPI processes.
- OpenMP is restricted to shared-memory multiprocessor platforms, the architecture of which can limit its scalability; MPI works on both shared-memory and distributed-memory platforms.
- OpenMP requires less expertise than MPI, allows concise incremental parallelism and yields unified code for sequential and parallel applications. MPI requires more knowledge and more programming to go from serial to parallel code.
- Performance comparable.



## OpenMP + MPI (cont.)

- Increasingly popular as a complementary combination
- Could it really be as simple as:

```
mpic++ program.cpp -o program -fopenmp
```

- Let's try!

# OpenMP + MPI

```
#include <iostream>
#include <omp.h>
#include "mpi.h"

int main(int argc, char **argv) {
  int rank, tid;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  #pragma omp parallel private(tid) num_threads(4)
  {
    tid = omp_get_thread_num();
    #pragma omp critical
    std::cout << "[" << rank << "] Started thread " << tid << std::endl;
  }

  MPI_Finalize();
  return 0;
}
```