CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Message Passing Interface (MPI)

Summer School 2017 – Effective High Performance Computing

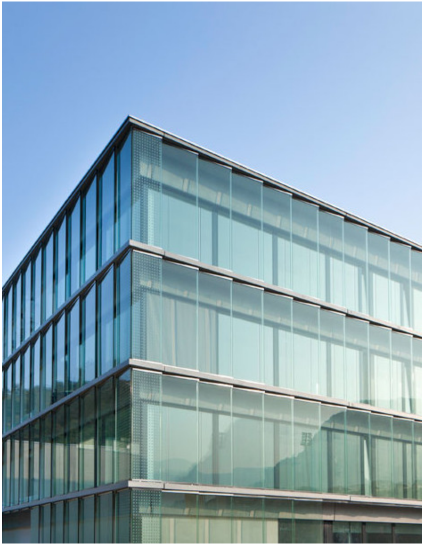
Tim Robinson, CSCS

July 19–20, 2017

Course Objectives

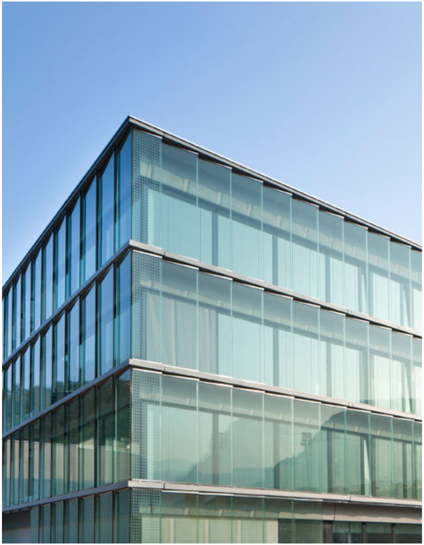
- Hybrid OpenMP/MPI, preparing MPI for OpenMP

General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Topology
- Datatypes
- Other topics

General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Topology
- Datatypes
- Other topics
  - Hybrid OpenMP/MPI

Configure MPI to enable OpenMP

Why OpenMP with MPI

There are two main motivations:

- To reduce the memory footprint (both in application and in communication buffers)
- To increase scalability (but you might not be faster than with MPI alone)

Reducing memory requirements

- Memory per core is generally decreasing
- MPI applications require some data to be replicated between MPI processes
  - Read-only lookup table where every process has a copy
  - Halo regions of neighbours
  - MPI internal data structions, e.g. communicaiton buffers

Local domain size	halos	% data in halos
$100^3 = 1,000,000$	$102^3 - 100^3 = 61,208$	6%
$50^3 = 125,000$	$52^3 - 50^3 = 15,608$	11%
$20^3 = 8,000$	$22^3 - 20^3 = 2648$	25%

Improving performance

- In the regime where MPI is scaling well, OpenMP will introduce overhead
- OpenMP can be used to exploit lower levels of parallelism
  - Might be hard to load balance with MPI
  - Might have irregular communication pattern
- In some cases collective communication overheads might be reduced
- OpenMP has in-built load balancing capabilities (loop schedules, tasks)

Preparing MPI for OpenMP

MPI requires to be setup with threads enabled:

- `MPI_Init` should be replaced by `MPI_Init_thread`

Pseudo-code

```
MPI_Init_thread(required, provided, ierror)
```

`required` specifies the requested level of thread support, and the actual level of support is then returned into `provided`.
You should check the value of `provided` after the call

MPI Thread level

- `MPI_THREAD_SINGLE`: Only one thread will execute (MPI-only application)
- `MPI_THREAD_FUNNELED`: Only master thread will make MPI calls (master = the thread calling `MPI_Init_thread`)
- `MPI_THREAD_SERIALIZED`: Only one thread at a time will make MPI calls (user responsibility)
- `MPI_THREAD_MULTIPLE`: Any thread may call MPI at any time, however that leads to slower performance (lock mechanism in MPI)

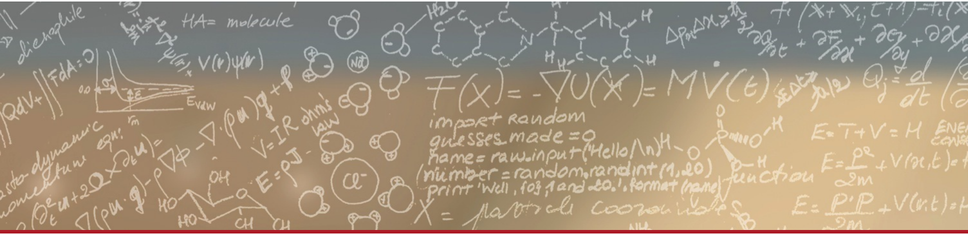
Checking the level of thread support provided

- The four classes of support are guaranteed to be monotonicly increasing `MPI_THREAD_SINGLE` $<$ `MPI_THREAD_FUNNELED` $<$ `MPI_THREAD_SERIALIZED` $<$ `MPI_THREAD_MULTIPLE`

- Therefore you can test to make sure at least the level you require is provided and exit if not:

Pseudo-code

```
if (provided < requested) {
    printf("Not high enough level of thread support.\n");
    MPI_Abort(MPI_COMM_WORLD,1);
}
```


HA= molecule
V=IR ohms law
import random
guesses_made = 0
name = raw_input('Hello!\n')
number = random.randint(1, 20)
print 'Well, {0}, I and 20.'.format(name)
E = T + V = H
function

Thank you for your attention.