CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Message Passing Interface (MPI)

Summer School 2016 – Effective High Performance Computing

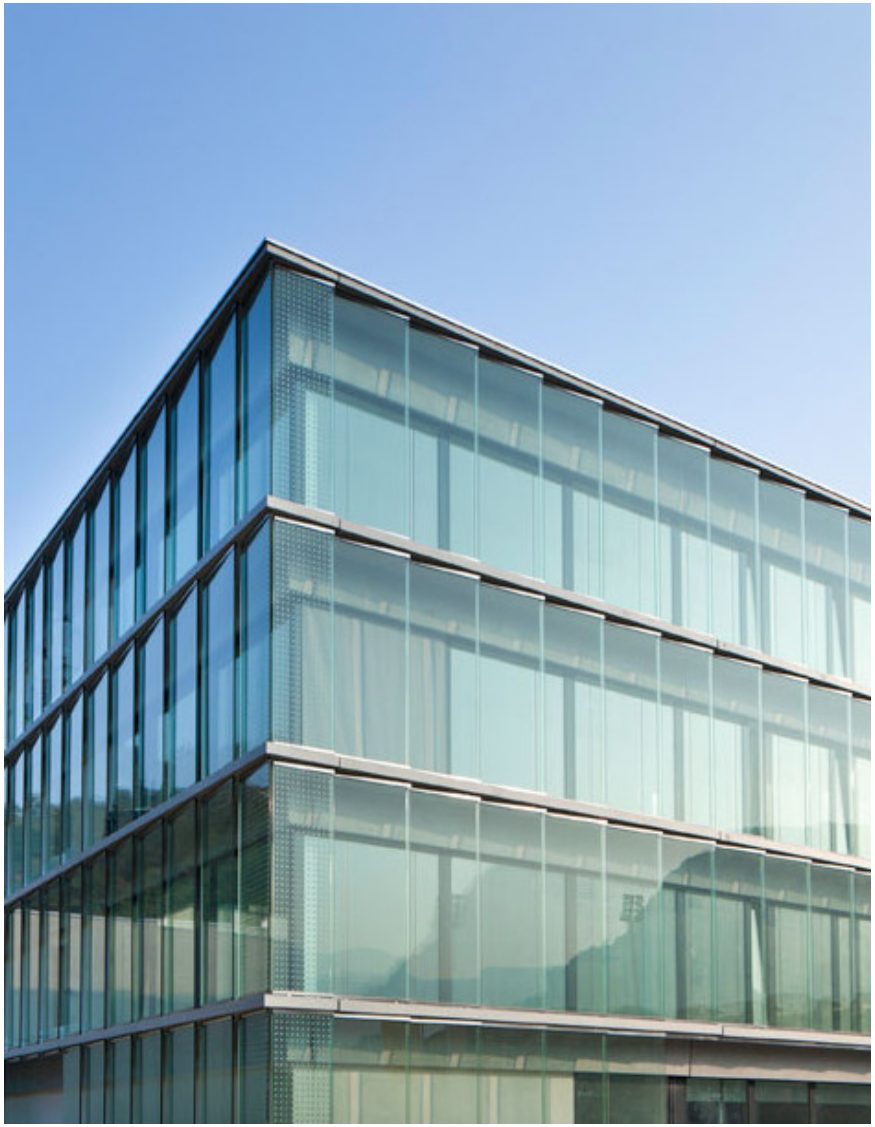
Maxime Martinasso, CSCS

July 20 – 21, 2016

Course Objectives

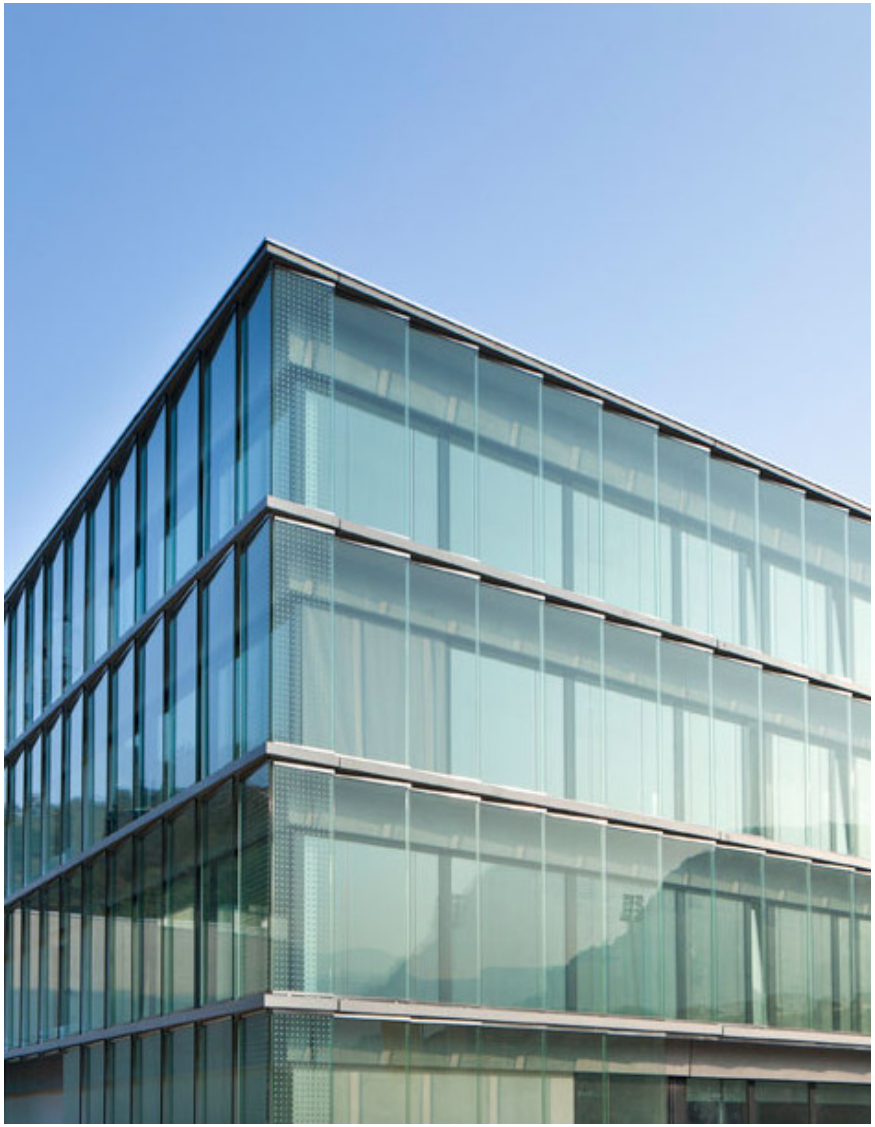
- Hybrid OpenMP/MPI, preparing MPI for OpenMP
- One sided MPI communication

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
  - One sided communication

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Configure MPI to enable OpenMP

Why OpenMP with MPI

- Code not fully parallel with MPI can benefit of extra threads
- Memory limit the number of MPI ranks per node
- To increase scalability (but you may well be not faster than with MPI alone)

Preparing MPI for OpenMP

MPI requires to be setup with threads enabled:

- `MPI_Init` should be replaced by `MPI_Init_thread`

C/C++
```
int MPI_Init_thread(int *argc, char ***argv, int required,
    int *provided)
```

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

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

One sided communication

No more receives?

- A memory region is exposed by all ranks (window)
- Each rank can read or write in a region
- Transfers and synchronisation are decoupled

Terminology

- Origin process: Process with the source buffer, initiates the operation
- Target process: Process with the destination buffer
- Epoch: Virtual time where operations are not completed. Data is consistent after new epoch is started.
- Ordering: order of message between two processes (can be relaxed compared to a strict order)

One sided communication usage

- Expose memory collectively
- Allocate exposed memory
- Dynamic memory exposure
- Communication (put, get, rput, rget)
- Accumulate (acc, racc, get_acc, rget_acc, fetch&op, cas)
- Synchronization

Expose memory

C/C++

```
MPI_Win_create(void *base, MPI_Aint size, int disp_unit,
    MPI_Info info, MPI_Comm comm, MPI_Win *win)
MPI_Win_allocate(MPI_Aint size, int disp_unit, MPI_Info info
    , MPI_Comm comm, void *baseptr, MPI_Win *win)
MPI_Win_free(MPI_Win *win)
```

Exposes (and allocate) the memory buffer to other ranks.

It is a collective operations among all ranks.

`MPI_Info` contains user configuration.

Dynamic memory

It is possible to create a window with no memory attached.

```
C/C++
MPI_Win_create_dynamic(MPI_Info info, MPI_Comm comm, MPI_Win
    *win)
```

It is possible to attach/detach memory to/from a window.

```
C/C++
MPI_Win_attach(MPI_Win win, void *base, MPI_Aint size)
MPI_Win_detach(MPI_Win win, const void *base)
```

Communication

Two functions: Write data into the window (put)

C/C++

```
MPI_Put(const void *origin_addr, int origin_count,
    MPI_Datatype origin_datatype, int target_rank, MPI_Aint
    target_disp, int target_count, MPI_Datatype
    target_datatype, MPI_Win win)
```

Read data from the window (get)

C/C++

```
MPI_Get(void *origin_addr, int origin_count, MPI_Datatype
    origin_datatype, int target_rank, MPI_Aint target_disp,
    int target_count, MPI_Datatype target_datatype, MPI_Win
    win)
```

- Non blocking communication
- Concurrent accesses give undefined behaviour (not an error).

Accumulate

```
C/C++
MPI_Accumulate(const void *origin_addr, int origin_count,
    MPI_Datatype origin_datatype, int target_rank, MPI_Aint
    target_disp, int target_count, MPI_Datatype
    target_datatype, MPI_Op op, MPI_Win win)
```

Remote accumulations, replace value in target buffer with accumulated.

- Only predefined operations
- Conflicting accesses are managed by the selected ordering

Other operations: `MPI_Fetch_and_op`, `MPI_Compare_and_swap`

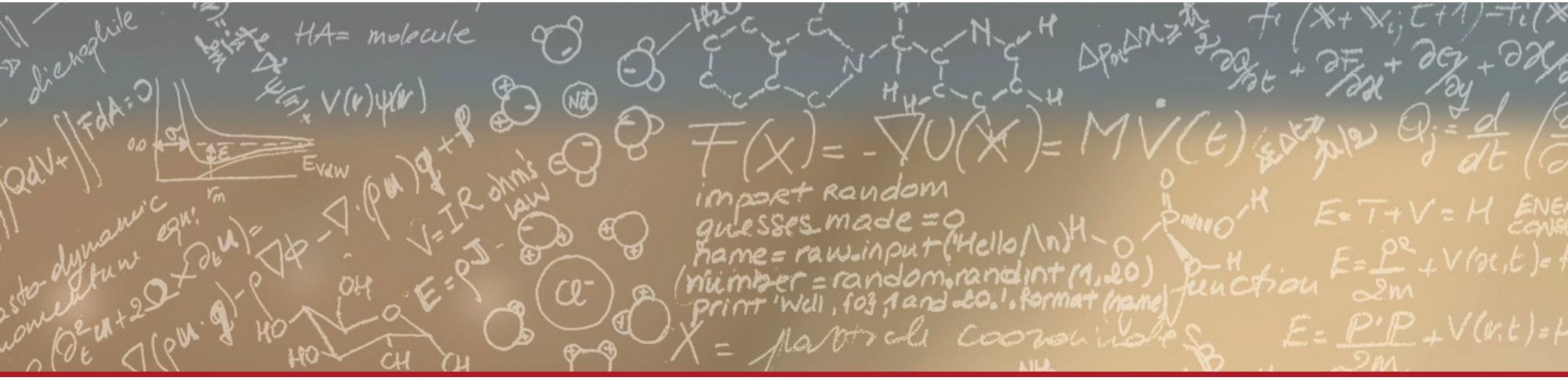
Synchronisation

```
C/C++
MPI_Win_fence(int assert, MPI_Win win)
```

Collectively synchronizes all RMA calls started before fence on the window.

Other synchronisation:

- Asynchronous with PSCW: Post, Start, Complete, Wait
- Lock / Unlock, Lock All
- Flush outstanding operations
- Synchronise a window



Thank you for your attention.