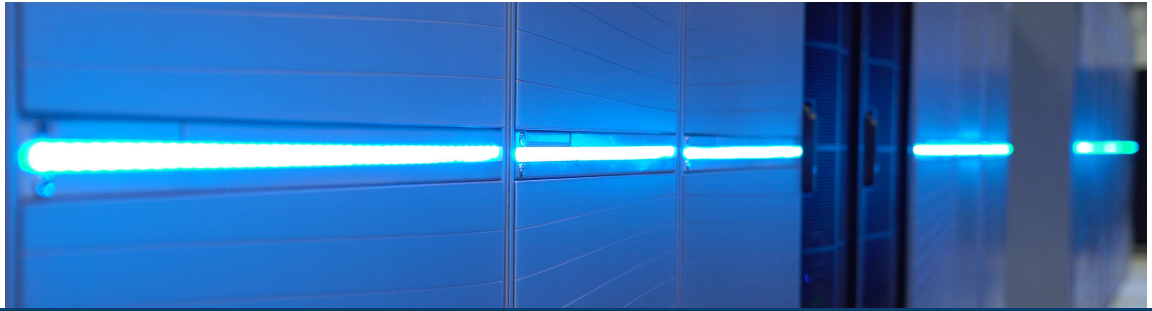

Part XII: Hybrid Programming

COMPILING & LINKING

MPI compiler wrappers combined with OpenMP command line argument, e.g.:

C Generic MPI Compiler Wrappers with OpenMP Command Line Switch

```
$ mpicc -fopenmp example.c -o example.exec
```

Fortran Generic MPI Compiler Wrappers with OpenMP Command Line Switch

```
$ mpifort -fopenmp example.f90 -o example.exec
```

PYTHON: no compilation is needed.

BASIC CODE STRUCTURE IN C

```c
#include <stdio.h>
#include <mpi.h>
#include <omp.h>

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

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // here comes your MPI & OpenMP code

    MPI_Finalize();
    return(0);
}
```

BASIC CODE STRUCTURE IN C

```c
#include <stdio.h>
#include <mpi.h>
#include <omp.h>

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

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // here comes your MPI & OpenMP code

    MPI_Finalize();
    return(0);
}
```

```c
//----------------------------------
MPI_call();

#pragma omp parallel for
for(i = 0; i < x; i++){
  stuff;
  stuff;
}

MPI_call();
//----------------------------------
```

BASIC CODE STRUCTURE IN C

```c
#include <stdio.h>
#include <mpi.h>
#include <omp.h>

int main(int argc, char **argv) {
    int size, rank;
    int prov, req=3;

    MPI_Init_thread(&argc, &argv, req, &prov);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // here comes your MPI & OpenMP code

    MPI_Finalize();
    return(0);
}
```

```c
//---------------------------------
MPI_call();

#pragma omp parallel for
for(i = 0; i < x; i++){
  stuff;
  MPI_call();
}

MPI_call();
//---------------------------------
```

THREAD COMPLIANCE [MPI-4.0, 11.6]

- An MPI library is thread compliant if
  1. Concurrent threads can make use of MPI routines and the result will be as if they were executed in some order.
  2. Blocking routines will only block the executing thread, allowing other threads to make progress.
- MPI libraries are not required to be thread compliant
- Alternative initialization routines to request certain levels of thread compliance
- These functions are always safe to use in a multithreaded setting: `MPI_Initialized`, `MPI_Finalized`, `MPI_Query_thread`, `MPI_Is_thread_main`, `MPI_Get_version`, `MPI_Get_library_version`

THREAD SUPPORT LEVELS [MPI-4.0, 11.2.1]

The following predefined values are used to express all possible levels of thread support:

`MPI_THREAD_SINGLE`	program is single threaded
`MPI_THREAD_FUNNELED`	MPI routines are only used by the main thread
`MPI_THREAD_SERIALIZED`	MPI routines are used by multiple threads, but not concurrently
`MPI_THREAD_MULTIPLE`	MPI is thread compliant, no restrictions

`MPI_THREAD_SINGLE` < `MPI_THREAD_FUNNELED` < `MPI_THREAD_SERIALIZED` < `MPI_THREAD_MULTIPLE`

INITIALIZATION [MPI-4.0, 11.2.1]

C

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

F08

```
MPI_Init_thread(required, provided, ierror)
integer, intent(in) :: required
integer, intent(out) :: provided
integer, optional, intent(out) :: ierror
```

- `required` and `provided` specify thread support levels
- If possible, `provided = required`
- Otherwise, if possible, `provided > required`
- Otherwise, `provided < required`
- `MPI_Init` is equivalent to `required = MPI_THREAD_SINGLE`

Initialization and Finalization

Initialization and finalization of MPI should occur on the same thread, the main thread.

INQUIRY FUNCTIONS [MPI-4.0, 11.2.1]

Query level of thread support:

C
```
int MPI_Query_thread(int *provided)
```

F08
```
MPI_Query_thread(provided, ierror)
integer, intent(out) :: provided
integer, optional, intent(out) :: ierror
```

Check whether the calling thread is the main thread:

C
```
int MPI_Is_thread_main(int* flag)
```

F08
```
MPI_Is_thread_main(flag, ierror)
logical, intent(out) :: flag
integer, optional, intent(out) :: ierror
```