

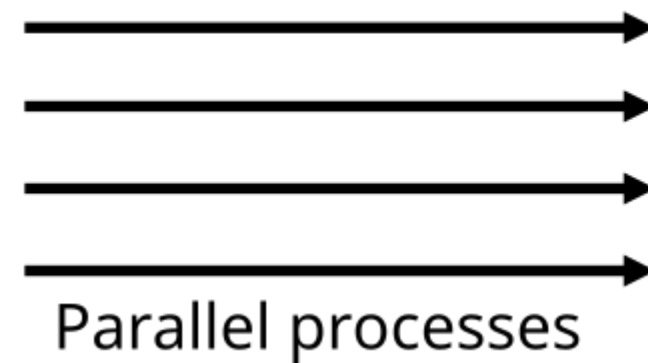
Hybrid MPI+OpenMP programming

CSC Training, 2021-03



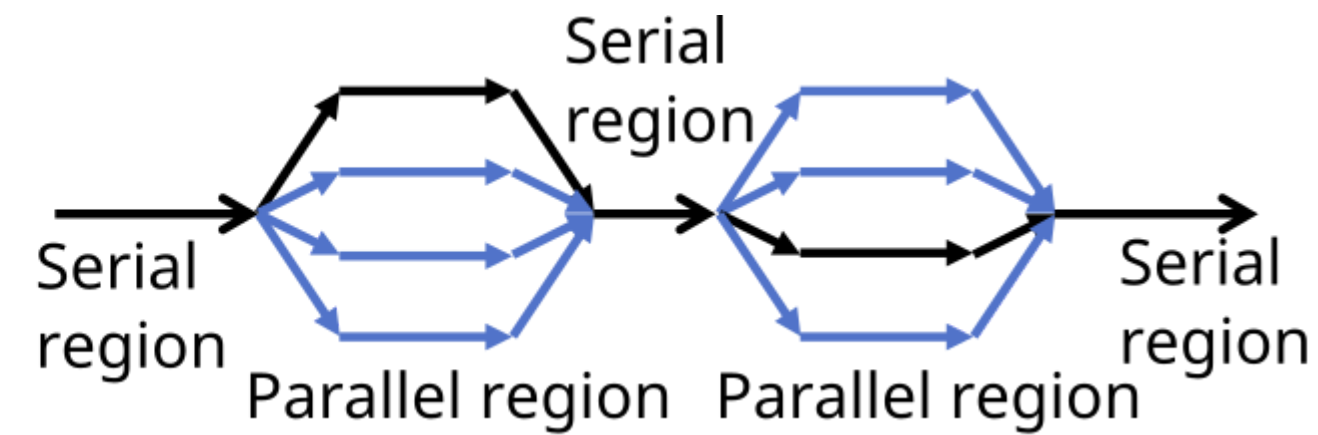
CSC – Finnish expertise in ICT for research, education and public administration

Processes and threads



Process

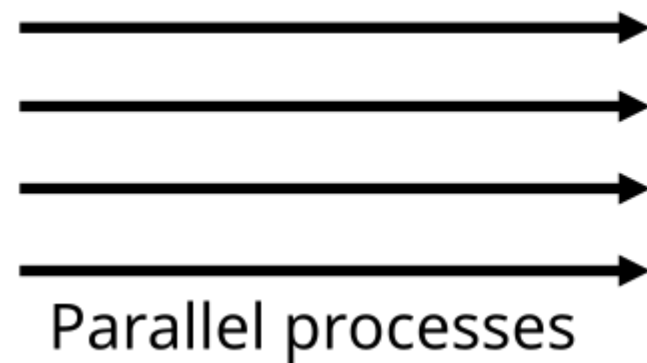
- Independent execution units
- Have their own state information and *own memory* address space



Thread

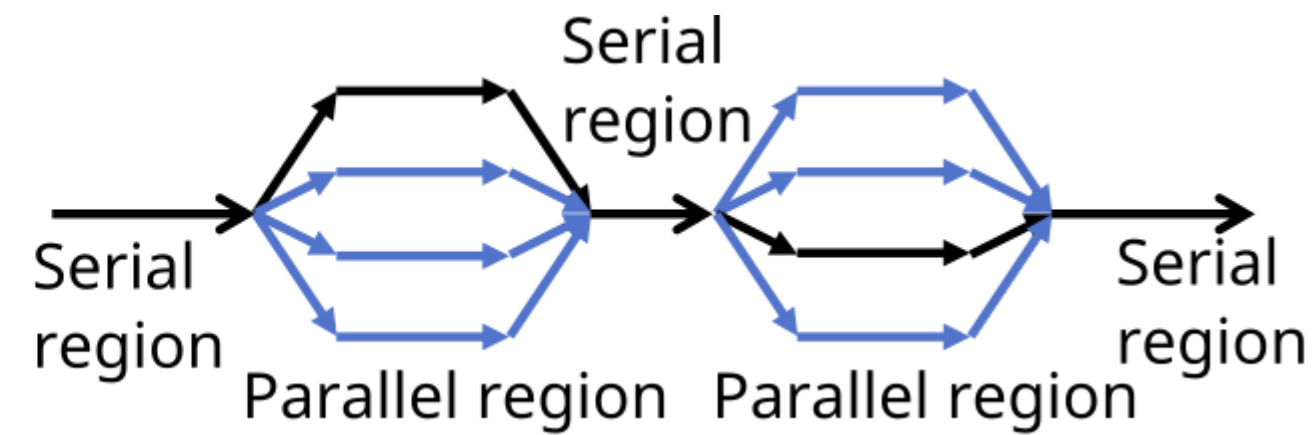
- A single process may contain multiple threads
- Have their own state information, but *share the same memory* address space

Processes and threads



Process

- Long-lived: spawned when parallel program started, killed when program is finished
- Explicit communication between processes



Thread

- Short-lived: created when entering a parallel region, destroyed (joined) when region ends
- Communication through shared memory

Hybrid programming

Hybrid programming: Launch threads (OpenMP) *within* processes (MPI)

Process

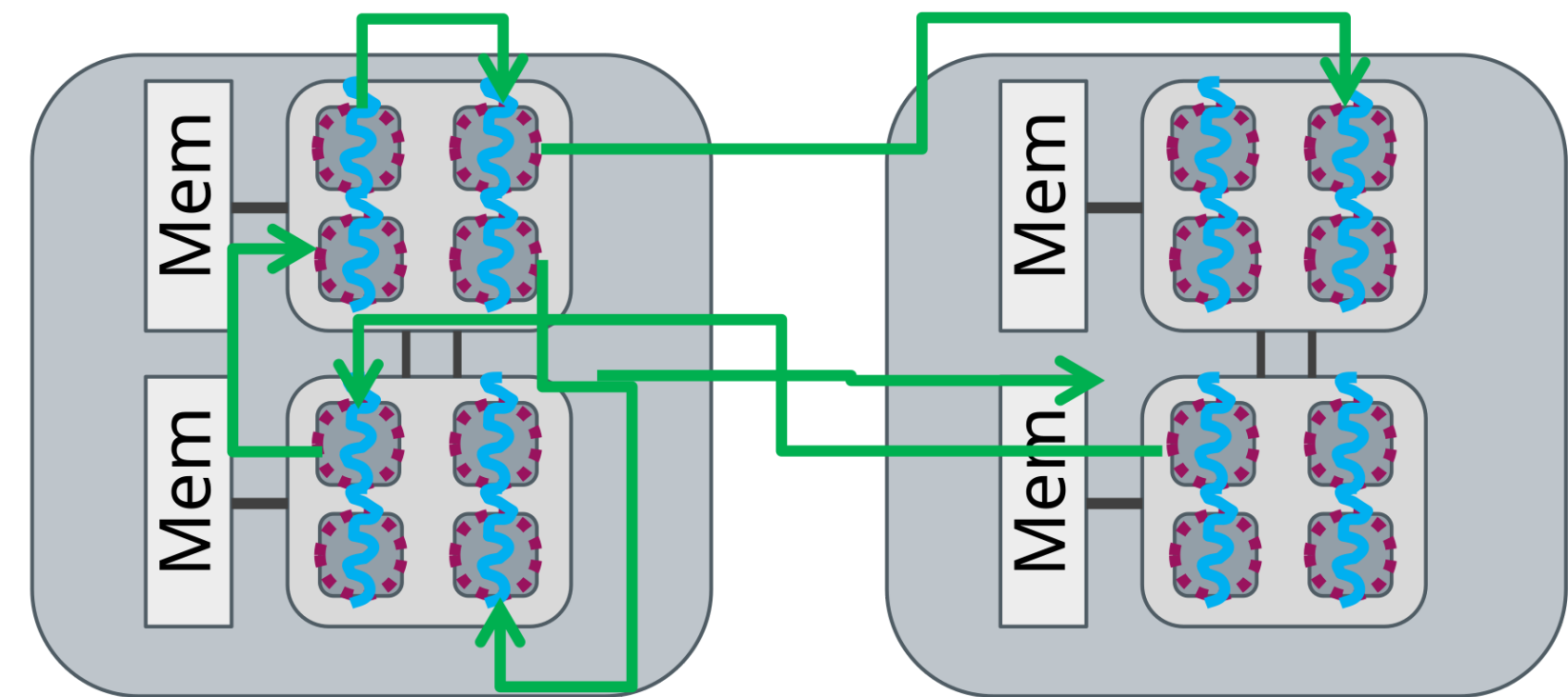
- Independent execution units
- MPI launches N processes at application startup

Thread




- Threads share memory space
- Threads are created and destroyed (parallel regions)

Hybrid programming

- Shared memory programming inside a node, message passing between nodes
- Matches well modern supercomputer hardware
- Often one MPI task / socket, but one should experiment with the ratio



Supercomputer node

-  MPI task
-  OpenMP thread
-  MPI message

Example: Hybrid hello

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

int main(int argc, char *argv[]) {
    int my_id, omp_rank;
    int provided, required=MPI_THREAD_FUNNELED;

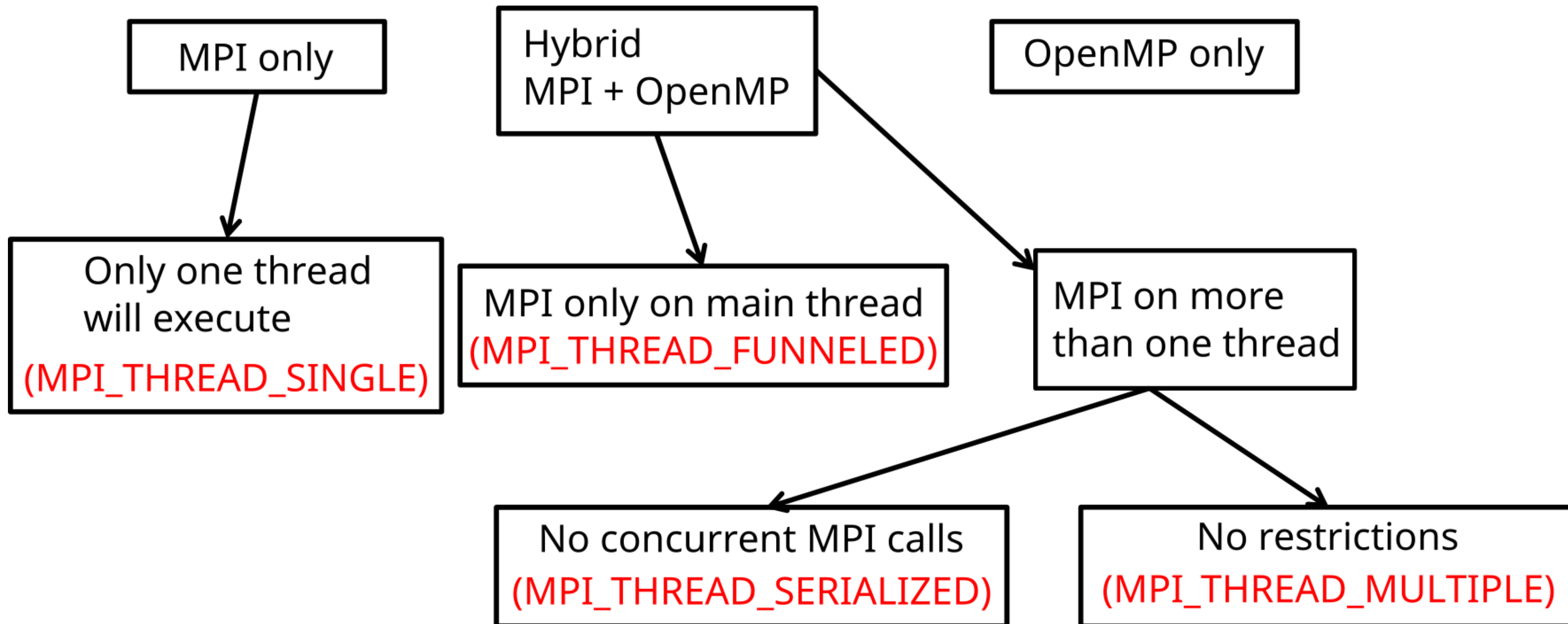
    MPI_Init_thread(&argc, &argv, required,
                    &provided);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
    #pragma omp parallel private(omp_rank)
    {
        omp_rank = omp_get_thread_num();
        printf("I'm thread %d in process %d\n",
               omp_rank, my_id);
    }

    MPI_Finalize();
}
```

```
$ mpicc -fopenmp hybrid-hello.c -o hybrid-hello
$ srun --account=yourproject --ntasks=2
    --cpus-per-task=4 --partition=test
    ./hybrid-hello
```

```
I'm thread 0 in process 0
I'm thread 0 in process 1
I'm thread 2 in process 1
I'm thread 3 in process 1
I'm thread 1 in process 1
I'm thread 3 in process 0
I'm thread 1 in process 0
I'm thread 2 in process 0
```

Thread support in MPI



Thread safe initialization

`MPI_Init_thread(required, provided)`

argc, argv

Command line arguments in C

required

Required thread safety level

provided

Supported thread safety level

error

Error value; in C/C++ it's the return value of the function, and in Fortran an additional output parameter

- Pre-defined integer constants: `MPI_THREAD_SINGLE` < `MPI_THREAD_FUNNELED`
< `MPI_THREAD_SERIALIZED` < `MPI_THREAD_MULTIPLE`

Hybrid programming styles: fine/coarse grained

- Fine-grained
 - Use **omp parallel do/for** on the most intensive loops
 - Possible to hybridize an existing MPI code with little effort and in parts
- Coarse-grained
 - Use OpenMP threads to replace MPI tasks
 - Whole (or most of) program within the same parallel region
 - More likely to scale over the whole node, enables all cores to communicate (if supported by MPI implementation)

Potential advantages of the hybrid approach

- Fewer MPI processes for a given amount of cores
 - Improved load balance
 - All-to-all communication bottlenecks alleviated
 - Decreased memory consumption if an implementation uses replicated data
- Additional parallelization levels may be available
- Possibility for dedicating threads for different tasks
 - e.g. dedicated communication thread or parallel I/O
 - Note that OpenMP worksharing constructs (e.g. OMP DO) will not be applicable when dedicating threads

Disadvantages of hybridization

- Increased overhead from thread creation/destruction
- More complicated programming
 - Code readability and maintainability issues
- Thread support in MPI and other libraries needs to be considered

Real-world hybrid applications

- Vlasiator – space weather simulations
 - Fluid in 6D spatial/velocity space: spatial grid with MPI, velocity grid with OpenMP
 - Lower memory footprint and better load balancing
- CP2K – electronic structure simulations
 - Especially so called hybrid functionals benefit from threading
 - Lower memory footprint allows more efficient computations

Real-world hybrid applications

- Vlasiator, 200 nodes (4800 cores) on Sisu

Threads per process	Hyperthreads per core	MPI ranks per node	Performance (108 cells/s)	Memory use per node (GB)
1	1	24	1.06	28.4
2	1	12	1.06	24.6
4	1	6	1.04	22.8
6	1	4	1.02	22.2
2	2	24	1.35	28.5
4	2	12	1.33	24.6
6	2	8	1.32	23.4
12	2	4	1.25	22.2

Summary

- Hybrid programming maps well to modern hardware
- In theory, hybrid programming offers several advantages
- In practice, all the advantages can be difficult to realize
- As number of cores inside a node increases, advantages of hybrid approach are likely to become more and more relevant
- MPI provides different levels of thread support