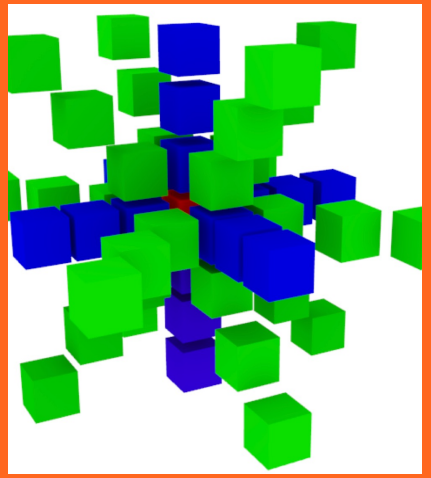
CS-E4690 – Programming parallel supercomputers

Hybrid computing in the CPU paradigm

Maarit Korpi-Lagg

maarit.korpi-lagg@aalto.fi

Aalto University
School of Science



Recap

The two trajectories resulting from the power wall

Multicore processors (core==CPU)

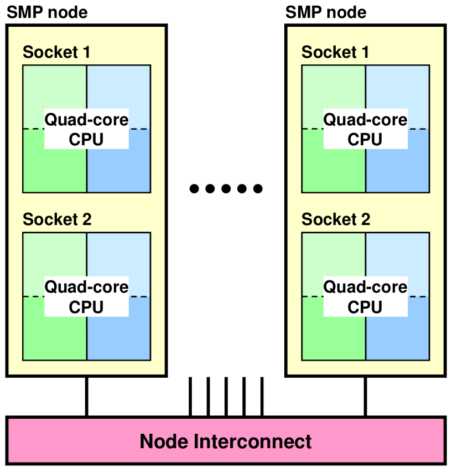
Lecture 5 (this material)

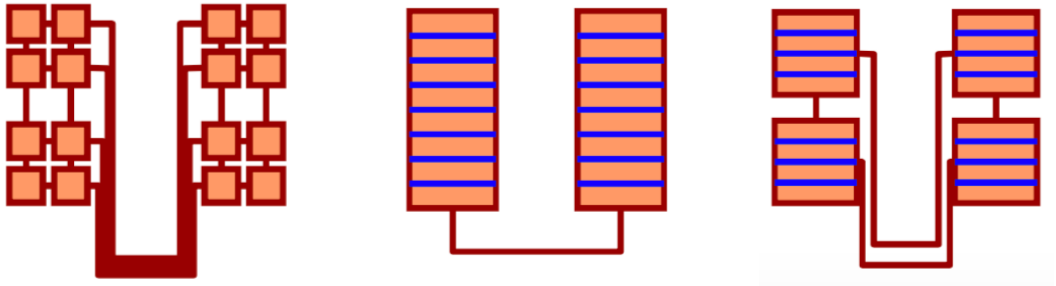
Multi-thread processors (e.g. processors with GPUs)

Lecture 6 (next week's material)

How to combine MPI distributed memory programming models with shared-memory ones?

One of the ultimate questions to answer to create efficient programs for the hybrid HPC platforms





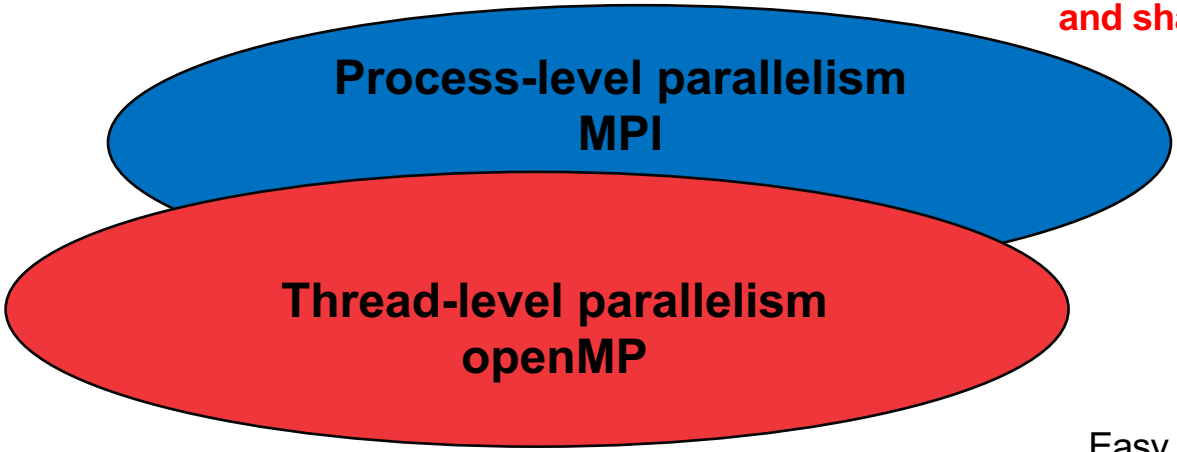
Programmer implements the communication patterns explicitely

Separate address space

Library calls

Complicated

Observation we made: Works both in distributed and shared memory



But is this the optimum?

Easy

Compiler translates the directives of the programmer into a communication pattern

Directives: the programmer has to correctly identify the parallel parts and dependencies

Common address space

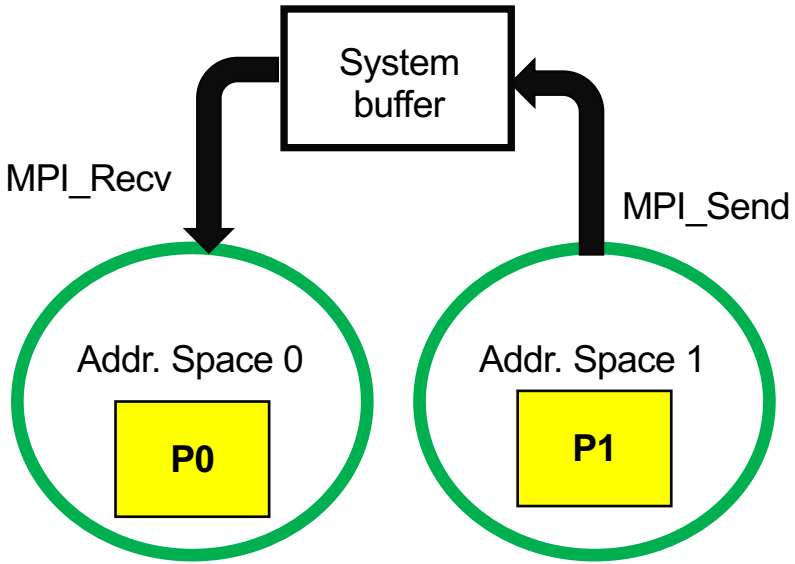
To get yourself started with/reminded about openMP, recommended reading includes

https://ppc.cs.aalto.fi/ch3/

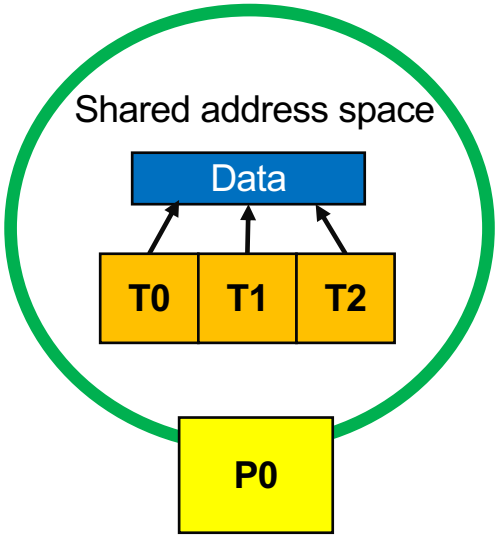
More and docs https://www.openmp.org

Memory models

MPI



openMP

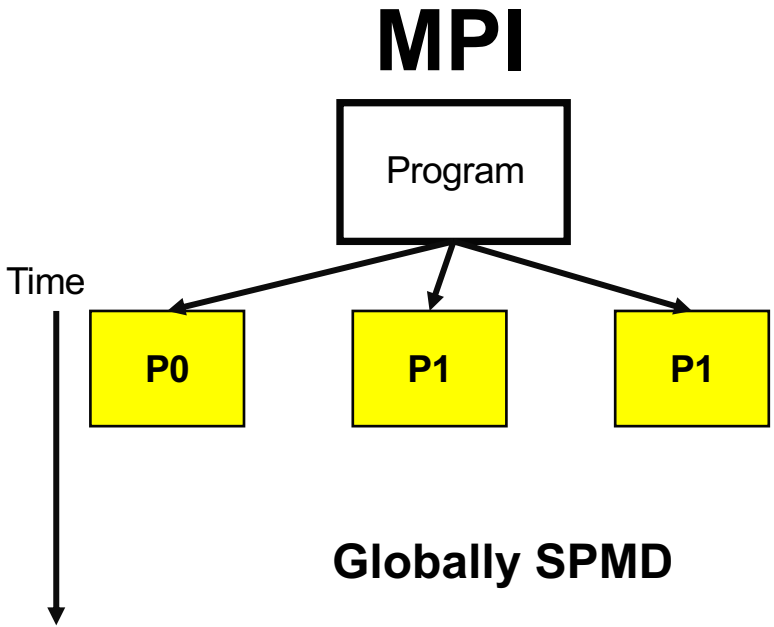


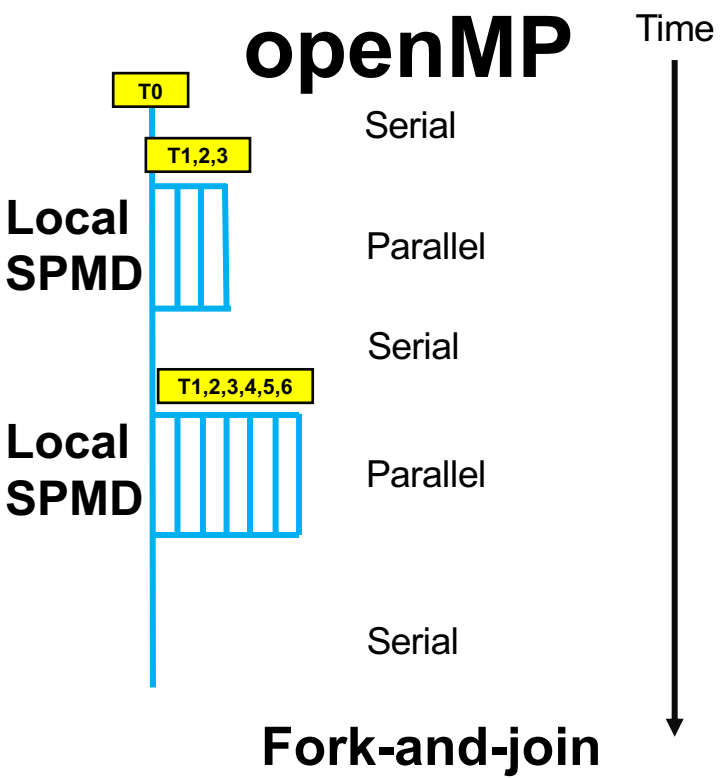
Dominating issue

Passing messages

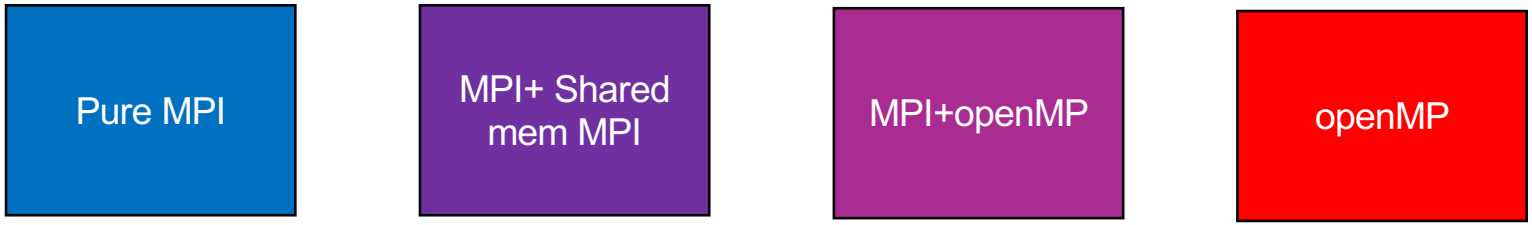
Thread synchronization

Execution models





What are the (lower level) hybrid comp. options?



The mode that has been used so far….

Now provides reference cases

Modes that are discussed today, and tried out in Sheet 5

Use shared mem. MPI within a node and MPI across nodes

Use OpenMP within a node and MPI across nodes

Lecture 4: one-sided p2p comms material

No capacity to investigate here, but **please read [1],** if you are interested in trying out.

Current consensus: not the way to go for distributed memory comp.

What benefits are we expecting?

Two types of improvements can be envisaged

1. **Reducing memory usage, both in the application and by the MPI library (e.g. decreased usage of communication buffers)**

2. **Improved performance and extended scale-up to higher number of CPU cores.**

Memory consumption issues with MPI

Strong scaling scenario: if only shared memory, total consumption remains constant; with MPI there can be an increase due the replication (application) and buffering (system) of data.

Why is this a problem? Core issue: some applications are limited by the amount of memory per core (1-2GB nowadays); this is not going to increase dramatically in the future; better to try to optimize the memory consumption.

Halo sizes in strong scaling case with 2nd order Moore stencil in 3D periodic case

Local domain size	Size of halos	Fraction of halos/domain size
$64^3 = 262{,}144$	$66^3 - 64^3 = 25{,}352$	10%
$32^3 = 32{,}768$	$34^3 - 32^3 = 6{,}536$	20%
$16^3 = 4{,}096$	$18^3 - 16^3 = 1{,}736$	42%

Goals?

- **To reduce the total memory requirement;** larger problem sizes can then be computed with the same amount of cores

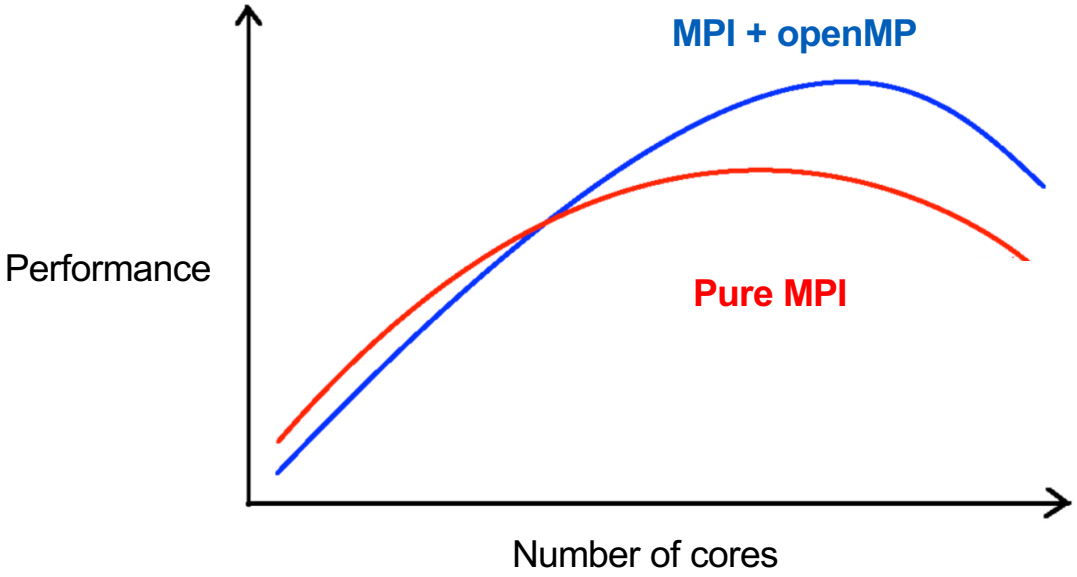
- **Reduced memory footprint per core may also give performance benefit, as data locality is improved:** Data can fit into cache, reducing the demand on memory bandwidth.

How could this be done?

Investigate whether the following strategy is possible:

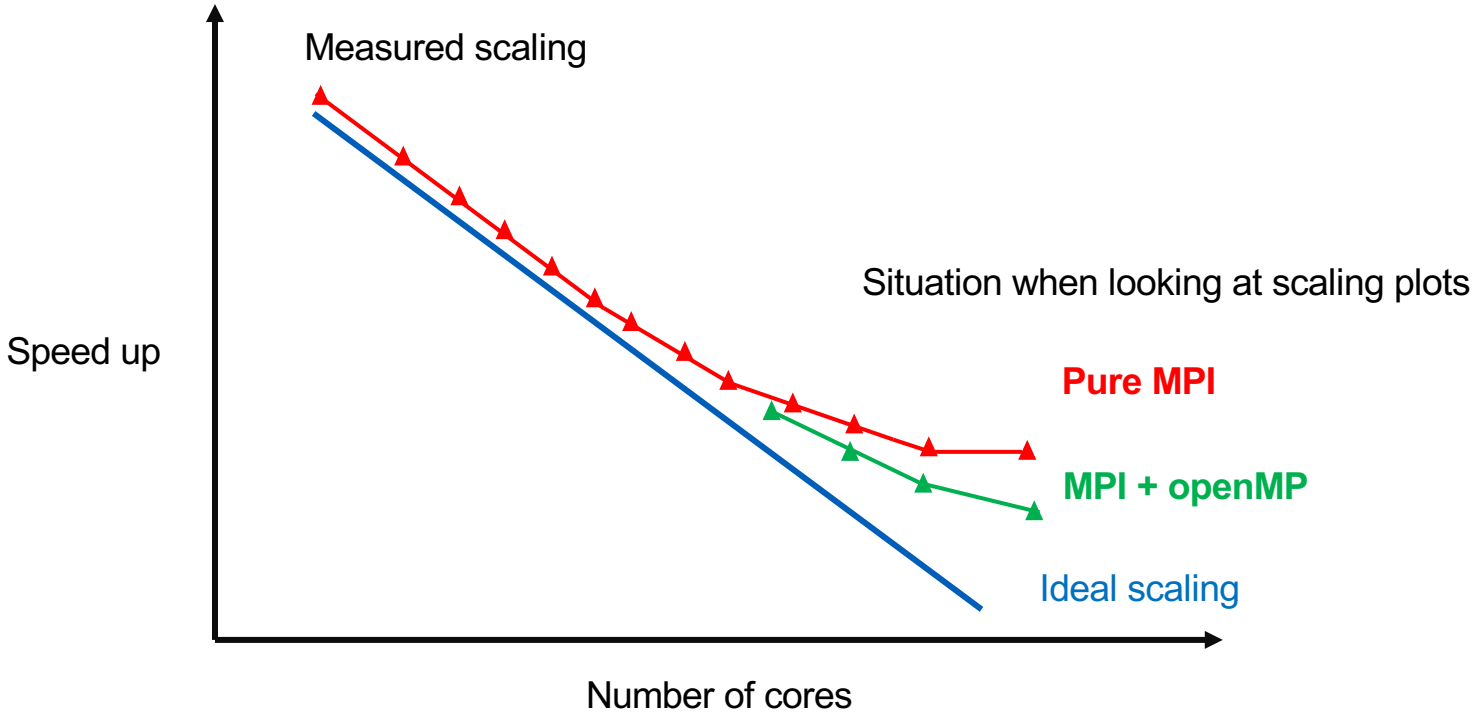
- **Request less MPI processes on each node than there are cores.**
- **This results in some cores being idle**
- **Use openMP threads to make the idle cores work (non-trivial, but possible)**

Performance



At low and intermediate core counts the performance of pure MPI is typically better than hybrid. At high core counts, parallelization overheads with pure MPI kill performance, but hybrid performance can overtake and the code may continue to perform to higher number of cores.

Scale-up


Measured scaling
Situation when looking at scaling plots
Speed up
Pure MPI
MPI + openMP
Ideal scaling
Number of cores

How can these benefits be achieved?

Investigate if there is a possibility to add lower-level parallelism into the application

Typical example: ISLs using MPI

Loop-level parallelism to be added

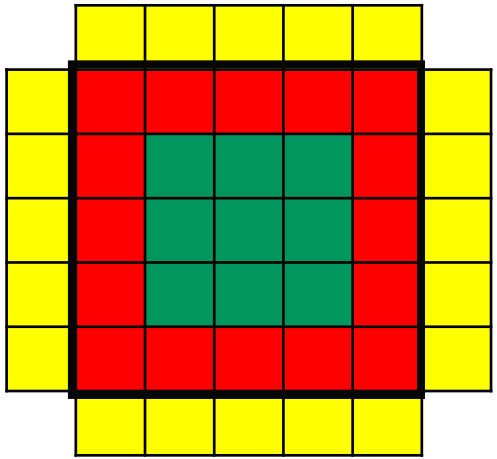
Repeat:

Initiate communication of yellow halos;

Do update of the green zones;

Wait for communications to finalize;

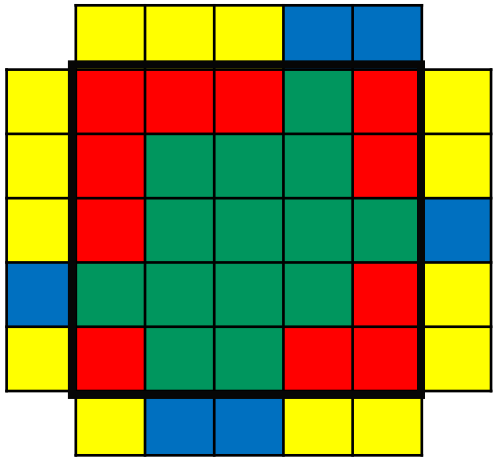
Update the red zones;



How can these benefits be achieved?

Investigate whether you can reduce communication overhead by removing redundant comm. operations

- **Typical example: dynamical ISLs; conditional communication of part of the halos required, but often implemented as full**
- **Do conditional communication in shared memory programming model**
- **Decrease number of MPI processes, give that work to openMPI threads**



OK, makes sense to implement

What to do in practise?

Use shared mem. MPI within a node and MPI across nodes

Use OpenMP within a node and MPI across nodes

First, check if your MPI library supports threading

ompi_info | grep "Thread support"

Triton:

Thread support: posix (MPI_THREAD_MULTIPLE: yes, OPAL support: yes, OMPI progress: no, ORTE progress: yes, Event lib: yes)

Hybrid/hello_class.c
scripts/job_hybrid_example.sh

How to make MPI to co-operate with threads?

Instead of MPI_Init() one should call

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

MPI_THREAD_SINGLE (0) Only one thread will execute (Equiv. of MPI_Init(). No openMP parallel regions in the code expected.

MPI_THREAD_FUNNELED (1) If the process is multithreaded, only the thread that called MPI_Init_thread will make MPI calls.

MPI_THREAD_SERIALIZED (2) If the process is multithreaded, only one thread will make MPI library calls at one time.

MPI_THREAD_MULTIPLE (3) If the process is multithreaded, multiple threads may call MPI at once with no restrictions.

All threads share all MPI objects (communicators, requests)

Case MPI_THREAD_FUNNELED

All MPI calls are made by the openMP master thread OUTSIDE parallel regions, or inside openMP master regions.

```
int main(int argc, char ** argv) {
int data[100], provided;
MPI_Init_thread(&argc, &argv, MPI_THREAD_FUNNELED, &provided);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
#pragma omp parallel for
for (i = 0; i < 100; i++)
compute(data[i]);
/* Do MPI stuff */
MPI_Finalize();
return 0; }
```

Master-only style, if calls only outside parallel regions

Master-only type programming

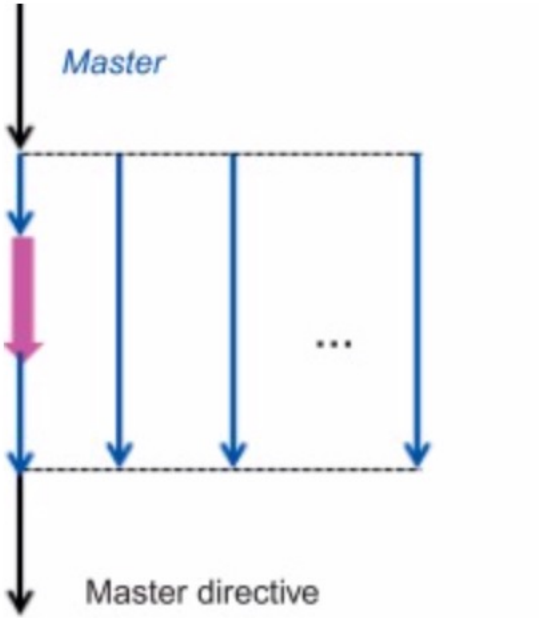
- All MPI calls outside openMP parallel regions
- Straightforward fork-and-join parallelism typical for openMP
- Easy and safe: Each parallel region imposes a synchronization, hence programmer does not have to worry about it. High overhead.
- During the MPI calls by master, all other threads are idling; using derived data types can be especially devastating, as the packing/unpacking of data is serialized
- Poor data locality; all data passes through the cache of the master thread

More cons than pros

Funneled type programming

- **MPI calls are made by OpenMP master thread, but take place inside OpenMP parallel “master” regions.**

```
#pragma omp parallel {
... work
#pragma omp barrier
#pragma omp master {
   MPI_Send(...);
}
#pragma omp barrier ...
work
}
```



Funneled type programming

- **Two restrictions are relaxed in comparison to master-only programming:**
  - there are now cheaper ways available to synchronise threads than opening and closing parallel regions
  - It possible for other threads to do useful computation while the master thread is executing MPI calls.

Serialized mode of programming

- Any thread inside an OpenMP parallel region may make calls to the MPI library, but the threads must be synchronised in such a way that only one thread at a time may be in an MPI call.

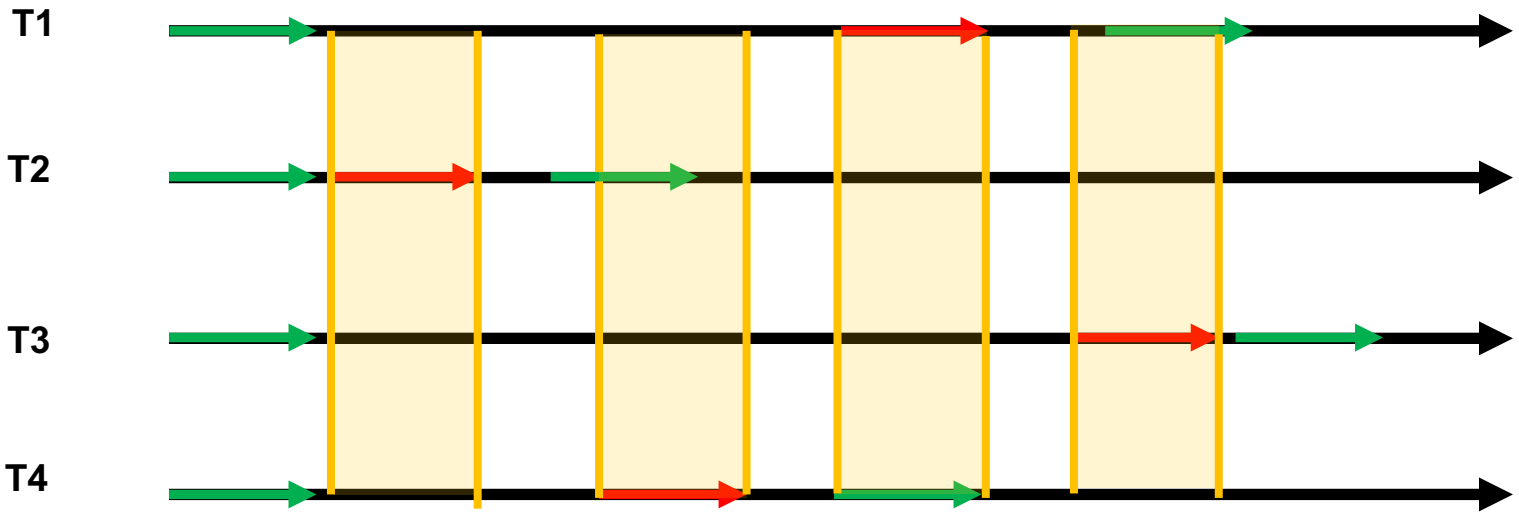
```
#pragma omp parallel {
... work
#pragma omp critical {
MPI_Send(...); }
... work }
```

Serialized mode of programming

- Threads can communicate their own data to other threads in other processes. This improves locality, since the message data is not all being cycled through one cache.
- It is now often necessary to use tags or communicators to distinguish between messages from (or to) different threads in the same MPI process. This is because the ordering of the sends and receives posted by different threads is non-deterministic.
- Ensuring threads do not enter MPI calls at the same time, by enclosing the MPI calls into openMP critical regions, may result in idle threads.

Serialized mode of programming

On a certain MPI rank of processes:



MPI calls are embedded in omp critical sections

Black: time
Green: computation
Red: communication
Orange: critical

Multiple style programming

- Any thread inside (or outside) an openMP parallel region may call MPI, and there are no restrictions on how many threads may be executing MPI calls at the same time.

```
#pragma omp parallel {
... work
MPI_Send(...);
... work
}
```

- MPI assumes that it should take care of thread safety internally.
- Application code can become very inefficient. Efficient usage of this model requires advanced knowledge on openMP; skip but if interested, read [2].

OK, makes sense to implement

What to do in practise?

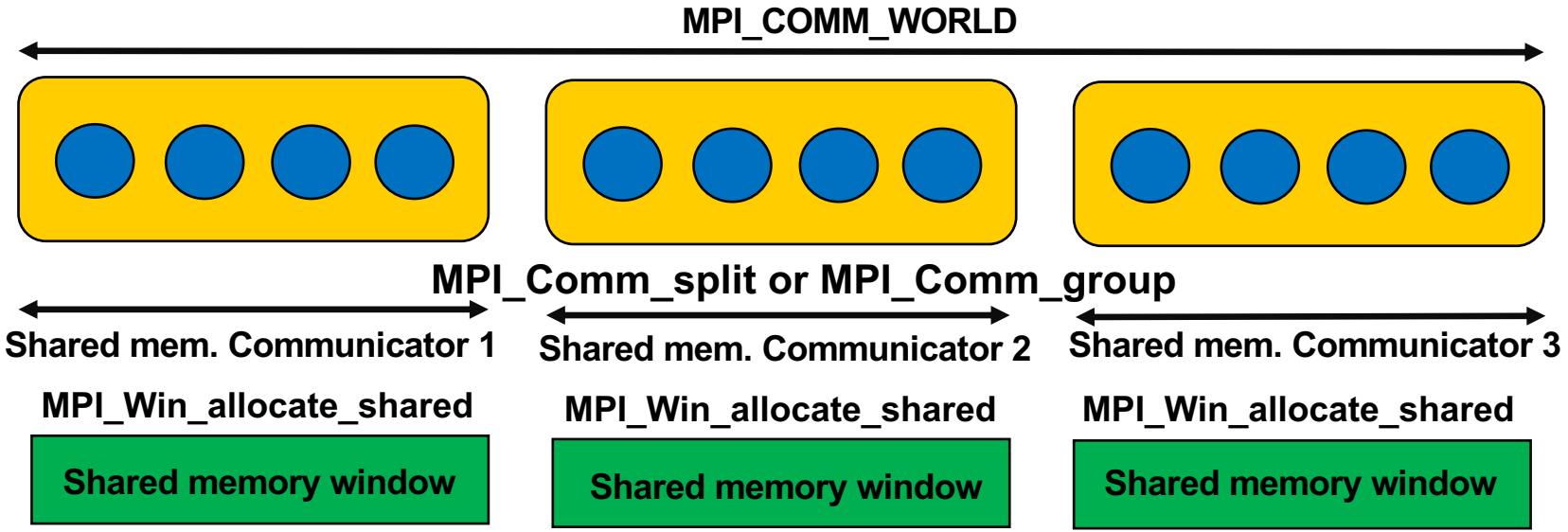
Use shared mem. MPI within a node and MPI across nodes

Use OpenMP within a node and MPI across nodes

Example programs:
Hybrid/OMP_MPI_X.c

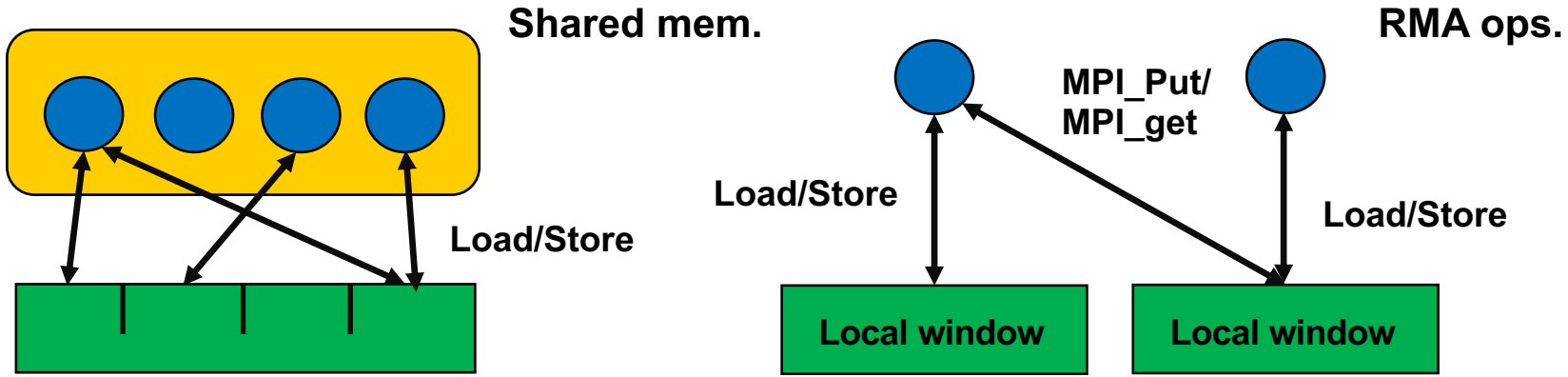
What is shared memory computing using MPI?

- **"Standard" MPI mode for internode comms, shared memory mode for the intranode comms; altogether only one programming standard**



Each blue circle= MPI process; orange boxes SMP nodes

Similarities and differences between RMA ops.



- No MPI_Put/MPI_get used in shared memory MPI mode; only loads/stores to the correct address of each core
- All RMA ops. are available, e.g. the atomic MPI_Accumulate and MPI_Get_accumulate
- Synchronization as in the RMA ops, e.g., fenching

OK, makes sense to implement

What to do in practise?

Use shared mem. MPI within a node and MPI across nodes

Example programs:
Hybrid/MPIs_MPI_X.c

Use OpenMP within a node and MPI across nodes

Example programs:
Hybrid/OMP_MPI_X.c

Useful reading

[1] A. Basumallik, S. Min and R. Eigenmann, "Programming Distributed Memory Sytems Using OpenMP," 2007 IEEE International Parallel and Distributed Processing Symposium, 2007, pp. 1-8, doi: 10.1109/IPDPS.2007.370397.

[2] http://www.intertwine-project.eu/sites/default/files/images/INTERTWinE_Best_Practice_Guide_MPI%2BOmpSs_1.0.pdf

Basics of openMP: https://ppc.cs.aalto.fi/ch3/

More and docs: https://www.openmp.org