Hybrid Programming Models for Emerging High Performance Computing Systems

Ekpe Okorafor

School of Parallel Programming & Parallel Architecture for HPC
ICTP
October, 2014

What you already know

- Parallelizing programs on shared memory computers using OpenMP
- Parallelizing programs on distributed memory computers using MPI

Emerging HPC Systems: Architecture

- General purpose processors are not getting (very much) faster
- The optimal (price/performance) HPC hardware goes to massively parallel computers (MPPs):
  - many compute nodes coupled by high-speed interconnects
  - each compute node is a multi-socket shared memory node
  - each socket holds a multi-core processor
- Hybrid architectures:
  - Coupling of “standard” compute nodes with highly specialized computes nodes (cell processors, general purpose GPUs)

Emerging HPC Systems: Performance Features

We the people hold these to be true!

- **Computation** (Flops) are cheap
- **Communication** (data transfer) is costly
- Data **locality** is key

- **What is the best model for emerging HPC systems?**

Emerging HPC Systems: Best Model

- Should be
  - Easy to use
  - Robust against errors
- Best use of hardware
  - low level, allowing for controlling details
  - compatible with old legacy code
- Energy consumption is main issue
  - Power for transfer >> Power for computation
- Avoid data transfers if possible
- Transfers must be controllable at every level

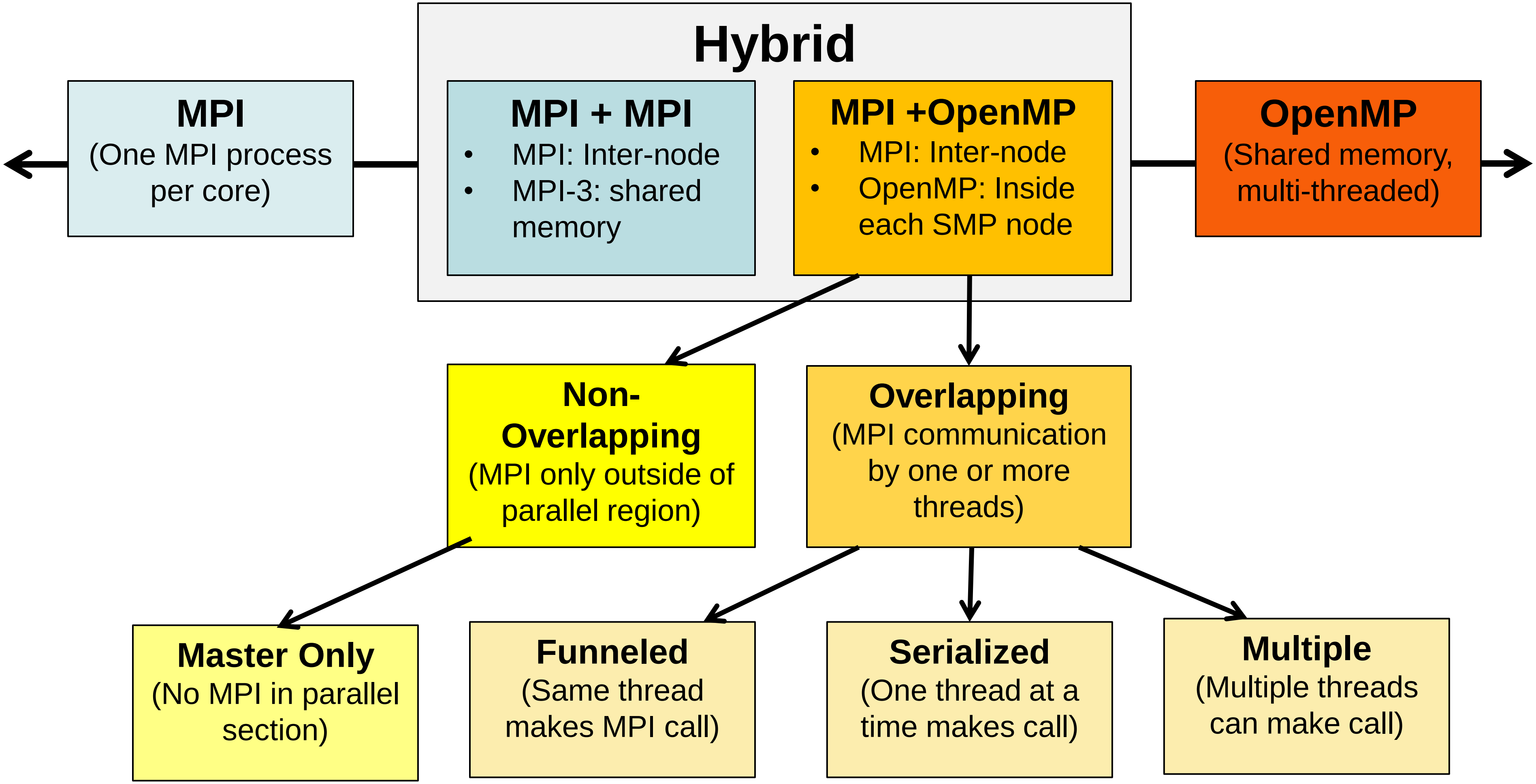
Emerging HPC Systems: Best Model

- Could use MPI across whole system
- Cannot (in general) use OpenMP/threads across whole system
  - requires support for single address space
  - this is possible in software, but very inefficient
  - also possible in hardware, but expensive
- **Could use OpenMP/threads within a node and MPI between nodes**
- **Combine shared and distributed memory**

Topics

- Hybrid Parallel Programming Models
- Hybrid Modes (MPI + OpenMP)
- NAS Parallel Benchmarks
- Other Hybrid Modes (MPI + MPI-3)
- Summary

Parallel Programming Models



Reasons to combine MPI & OpenMP

1. This software approach matches the hardware trend.
2. Some applications expose two levels of parallelism: coarse-grained (suitable for MPI), and fine-grained (best suited for OpenMP)
3. Application requirements or system restrictions may limit the number of MPI processes. Thus, OpenMP can offer an additional amount of parallelism.
4. Some application show unbalanced workload at the MPI level. OpenMP can be used to address this issue by assigning a different number of threads to each MPI process.

Basic Hybrid Program Template

Start with MPI initialization

(Serial regions are executed by the master thread of the MPI process)

Create OMP parallel regions within each MPI process

- MPI calls may be allowed here too
- MPI rank is known to all threads

Call MPI in single-threaded regions

Finalize MPI

```
MPI_Init
....
MPI_Call
....
   OMP parallel
   ....
   MPI_Call
   ....
   end parallel
....
MPI_Call
....
MPI_Finalize
```

Hybrid Mode: Master Only

Fortran:

```
!$OMP parallel
work…
!$OMP end parallel

call MPI_Send(…)

!$OMP parallel
work…
!$OMP end parallel
```

C:

```
#pragma omp parallel
{
work…
}
ierror=MPI_Send(…);
#pragma omp parallel
{
work…
}
```

Hybrid Mode: Funneled

Fortran:

```
!$OMP parallel
… work
!$OMP barrier
!$OMP master
call MPI_Send(…)
!$OMP end master
!$OMP barrier
.. work
!$OMP end parallel
```

C:

```
#pragma omp parallel
{
… work
#pragma omp barrier
#pragma omp master
{
ierror=MPI_Send(…);
}
#pragma omp barrier
… work
}
```

- Use `OMP Barrier` since there is no implicit barrier in master work-share construct (`OMP Master`).
- All other threads will be sleeping.

Hybrid Mode: Serialized

Fortran:

```
!$OMP parallel
… work
!$OMP barrier
!$OMP single
 call MPI_Send(…)
!$OMP end single
!Don't need OMP barrier
… work
!$OMP end parallel
```

C:

```
#pragma omp parallel
{
… work
#pragma omp barrier
#pragma omp single
{
ierror=MPI_Send(…);
}
//Don't need omp barrier
… work
}
```

- Best to use `OMP Barrier` only at beginning, since there is an implicit barrier in the SINGLE workshare construct, `OMP SINGLE`
- All other threads will be sleeping.

Hybrid Mode: Multiple

Fortran:

```
!$OMP parallel
… work
call MPI_Send(…)
… work
!$OMP end parallel
```

C:

```
#pragma omp parallel
{
… work
ierror=MPI_Send(…);
… work
}
```

- No restrictions

Hybrid Mode

Support Level	Description
MPI_THREAD_SINGLE	Only one thread will execute.
MPI_THREAD_FUNNELED	Process may be multithreaded but only main thread can make MPI calls. **Default mode**.
MPI_THREAD_SERIALIZE	Process may be multithreaded, any thread can make MPI calls but threads cannot execute MPI calls concurrently.
MPI_THREAD_MULTIPLE	Multiple threads may call MPI. No restrictions.

MPI Initialization

- MPI_Init_thread works in a similar way to MPI_Init by initializing MPI on the main thread.
- It has two integer arguments:
  - Required ([in] Level of desired thread support)
  - Provided ([out] Level of provided thread support)

- C syntax

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

- Fortran syntax

```
MPI_INIT_THREAD(REQUIRED, PROVIDED, IERROR)
INTEGER REQUIRED, PROVIDED, IERROR
```

MPI Hybrid: Hello World

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

int main(int argc, char *argv[]) {
  int numprocs, rank, namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int iam = 0, np = 1;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name(processor_name, &namelen);

  #pragma omp parallel default(shared) private(iam, np)
  {
    np = omp_get_num_threads();
    iam = omp_get_thread_num();
    printf("Hello from thread %d out of %d from process %d out of %d on %s\n",
           iam, np, rank, numprocs, processor_name);
  }

  MPI_Finalize();
```

MPI Hybrid: Hello World

- **Compiling and Linking Mixed MPI and OpenMP Programs**

Once you have your example program, you can compile and link it with

```
$mpicc -openmp hello.c -o hello
```

- **Running Mixed Programs**

```
$export OMP_NUM_THREADS=4
$mpirun -np 2 -x OMP_NUM_THREADS ./hello
```

```
Hello from thread 0 out of 4 from process 0 out of 2 on hpcnode004
Hello from thread 1 out of 4 from process 0 out of 2 on hpcnode004
Hello from thread 2 out of 4 from process 0 out of 2 on hpcnode004
Hello from thread 3 out of 4 from process 0 out of 2 on hpcnode004
Hello from thread 0 out of 4 from process 1 out of 2 on hpcnode001
Hello from thread 3 out of 4 from process 1 out of 2 on hpcnode001
Hello from thread 1 out of 4 from process 1 out of 2 on hpcnode001
Hello from thread 2 out of 4 from process 1 out of 2 on hpcnode001
```

- **Note that you have to tell OpenMPI to set the OMP_NUM_THREADS environment variable for OpenMP for each process it starts using the -x OMP_NUM_THREADS command line**

Things To Watch Out For!

- Introducing OpenMP into an existing MPI code also means introducing the drawbacks of OpenMP, such as the following:
  - Limitations when it comes to control of work distribution and synchronization
  - Overhead introduced by thread creation and synchronization
  - Dependence on quality of compiler and runtime support for OpenMP
  - Shared memory issues (ccNUMA architectures)
- The interaction of MPI and OpenMP runtime libraries may have negative side effects on the program's performance.
- Some appliations naturally expose only one level of parallelism, and there may be no benefit in introducing a hierarchical parallelism.

NAS Parallel Benchmarks

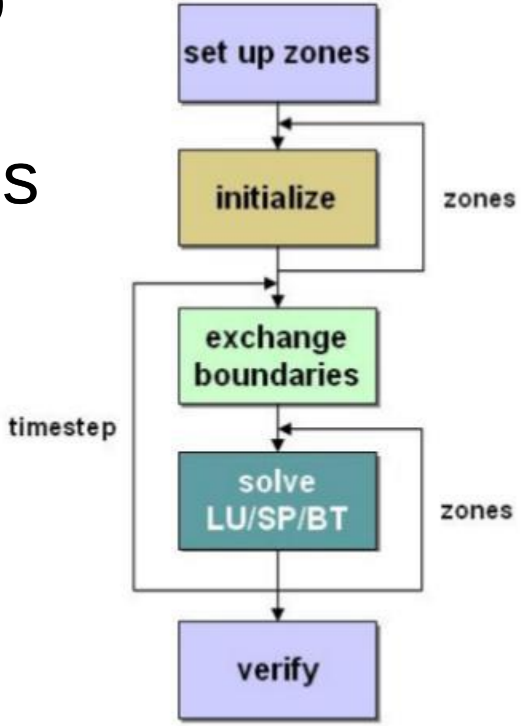
- Simulated 3D CFD simulation
- Uses (Alternating Direction Implicit) ADI in 3D to solve the discrete Navier-Stokes equations
- Parallelization by domain decomposition
- Consider the multi-zone benchmarks
- Zone size varies widely, therefore, a load-balancing algorithm is used

NPB on Ranger at TACC

- DDR infiniband network
- 3936 compute blades
- Each node: 4 2.3GHz AMD “Barcelona” quad core
- Peak performance: 579 TFlops
- MVAPICH, numactl
- PGI F90 7.1
- Class E benchmark: 4224 x 3456 x 92 points in 4096 zones
- Equally-sized zones - Scalar Pentadiagonal (SP)
- Zones of varying size - Block Triagonal (BT)

NPB: Multi-zone Hybrid

- Multi-zone versions of NPB (NPB-MZ) are designed to exploit multiple levels of parallelism in applications and to test the effectiveness of multi-level and hybrid parallelization paradigms and tools.
  - BT-MZ - uneven-size zones within a problem class, increased number of zones as problem class grows
  - SP-MZ - even-size zones within a problem class, increased number of zones as problem class grows



NPB: Multi-zone Hybrid

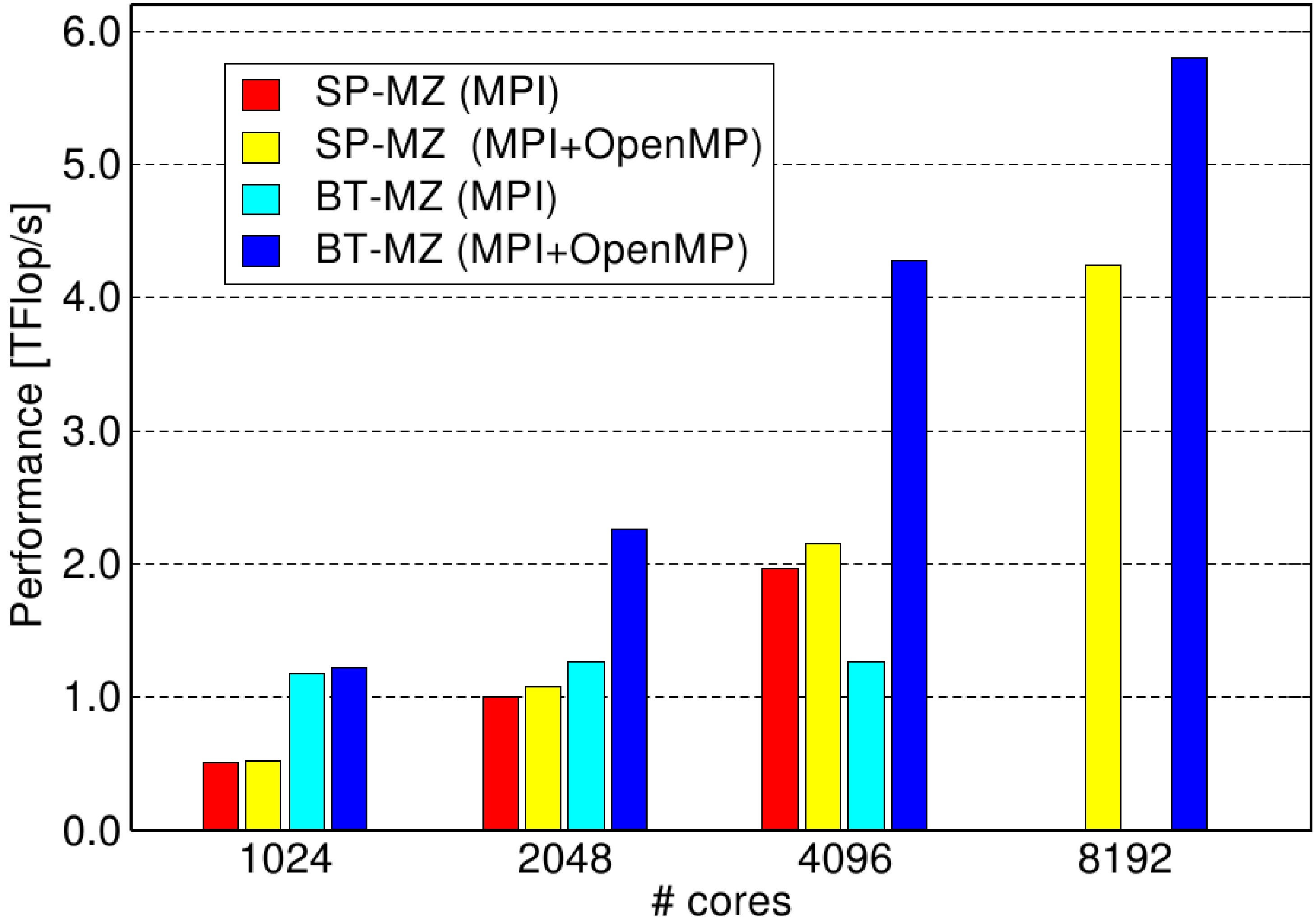


Figure from: Ralf Rabenseifner, Georg Hager, Gabriele Jost: Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes

What Can We Glean From This?

- BT-MZ:
  - Inherent workload imbalance on MPI level
  - #nprocs = #nzones yields poor performance
  - #nprocs < #zones => better workload balance, but decreases parallelism
  - Hybrid MPI/OpenMP yields better load-balance, maintains amount of parallelism
- SP-MZ:
  - No workload imbalance on MPI level, pure MPI should perform best
  - MPI/OpenMP outperforms MPI (on some platforms) due to network contention access within a node

In Simple Terms!

- Pure OpenMP performing better than pure MPI within node is a necessity to have hybrid code better than pure MPI across node.

But not sufficient...

- It is often very hard to make hybrid programs run faster than pure MPI solutions.
- One needs to use hardware specific tuning.
- Hybrid programming includes both programming techniques and system tools.

Hybrid Model: MPI + MPI

Rolf Rabenseifner[1)] Georg Hager[2)] Gabriele Jost[3)]
Rabenseifner@hlrs.de Georg.Hager@rrze.uni-erlangen.de gjost@supersmith.com

[1)] High Performance Computing Center (HLRS), University of Stuttgart, Germany
[2)] Regional Computing Center (RRZE), University of Erlangen, Germany
[3)] Supersmith, Maximum Performance Software, USA

- Hybrid MPI+MPI – (MPI for inter-node communication + MPI-3.0 shared memory programming)
- Advantages:
  - No message passing inside of the SMP nodes
  - Using only one parallel programming standard
  - No OpenMP problems (e.g., thread-safety isn't an issue)
- Issues:
  - Communicator must be split into shared memory islands
  - To minimize shared memory communication overhead:
    - Halos (or the data accessed by the neighbors)must be stored in MPI shared memory windows
  - Same work-sharing as with pure MPI

EXTRA SLIDES!

MPI-3: Shared Memory

- Split main communicator into shared memory islands
  - MPI_Comm_split_type
- Define a shared memory window on each island
  - MPI_Win_allocate_shared
  - Result (by default):
    - contiguous array, directly accessible by all processes of the island
- Accesses and synchronization
  - Normal assignments and expressions
  - No MPI_PUT/GET !
  - Normal MPI one-sided synchronization, e.g., MPI_WIN_FENCE

Summary: Pure MPI + OpenMP Only

MPI + OpenMP

- Seen with NPB-MZ examples
  - BT-MZ strong improvement (as expected)
  - SP-MZ small improvement
  - Usability on higher number of cores
- •Advantages
  - Memory consumption (This is probably the most important advantage)
  - Load balancing
  - Two levels of parallelism
    - Outer distributed memory halo data transfer (MPI)
    - Inner shared memory ease of SMP parallelization (OpenMP)
- Quite doable
- Does have a huge amount of pitfalls
- Optimum: Somewhere in the area of 1 MPI process per NUMA domain

Summary: Hybrid MPI + OpenMP

- **Pure MPI**
  + Ease of use
  – Topology and mapping problems may need to be solved (depends on loss of efficiency with these problems)
  – Number of cores may be more limited than with MPI+OpenMP
  + Good candidate for perfectly load-balanced applications
- **OpenMP only**
  + Ease of use
  – Limited to problems with tiny communication footprint
  – Source code modifications are necessary (Variables that are used with “shared” data scope must be allocated as “sharable”)
  ± (Only) for the appropriate application a suitable tool

Summary: Hybrid MPI + OpenMP

MPI+OpenMP:

  - Pitfalls of both MPI & OpenMPI

- Pitfalls through combination of MPI & OpenMP
  - E.g., topology and mapping problems
  - Many mismatch problems
- Tools are available
  - It is not easier than analyzing pure MPI programs
  - Most hybrid programs are Masteronly style
- Overlapping communication and computation with several threads
  - Requires thread-safety quality of MPI library
  - Loss of OpenMP worksharing support means using OpenMP tasks as workaround

Summary: Hybrid MPI + MPI-3

MPI+MPI Shared Memory:

- Two levels of parallelism
  - Outer -> distributed memory - > halo data transfer (MPI)
  - Inner ->shared memory -> halo transfer or direct access (MPI-3)
- New promising hybrid parallelization model
- No real experience up to now
- No OpenMP and thread-safety problems