



Improving (MPI) Applications Performance on Multicore Clusters with Optimized Process Placement

Emmanuel Jeannot, Guillaume Mercier,
François Tessier

LaBRI/INRIA Bordeaux Sud-Ouest/ENSEIRB

Runtime Team

Emmanuel.Jeannot@inria.fr

Introduction



- MPI is the main standard for programming parallel applications
- It provides portable code across platforms
- What about performance portability?



Performance of MPI programs

Depend on many factors:

- Implementation of the standard (e.g. collective com.)
- Parallel algorithm(s)
- Implementation of the algorithm
- Underlying libraries (e.g. BLAS)
- Hardware (processors, cache, network)
- etc.
- and ...



Process placement

The MPI model makes little (no?) assumption on the way MPI processes are mapped to resources

It is often assume that the network topology is flat and hence the process mapping has little impact on the performance

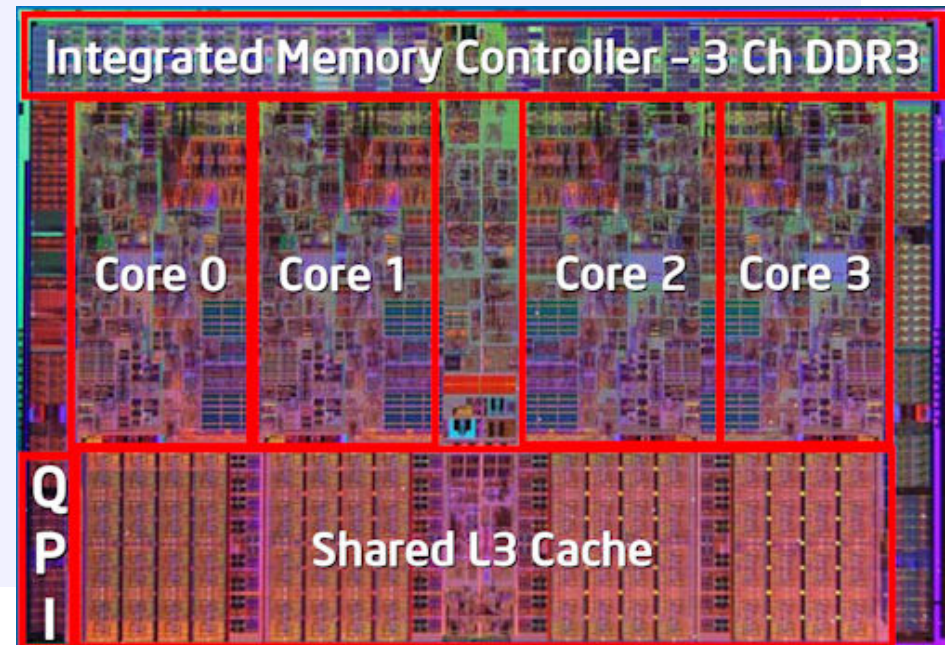


The network topology is not flat

Due to multicore processors current and future parallel machines are hierarchical

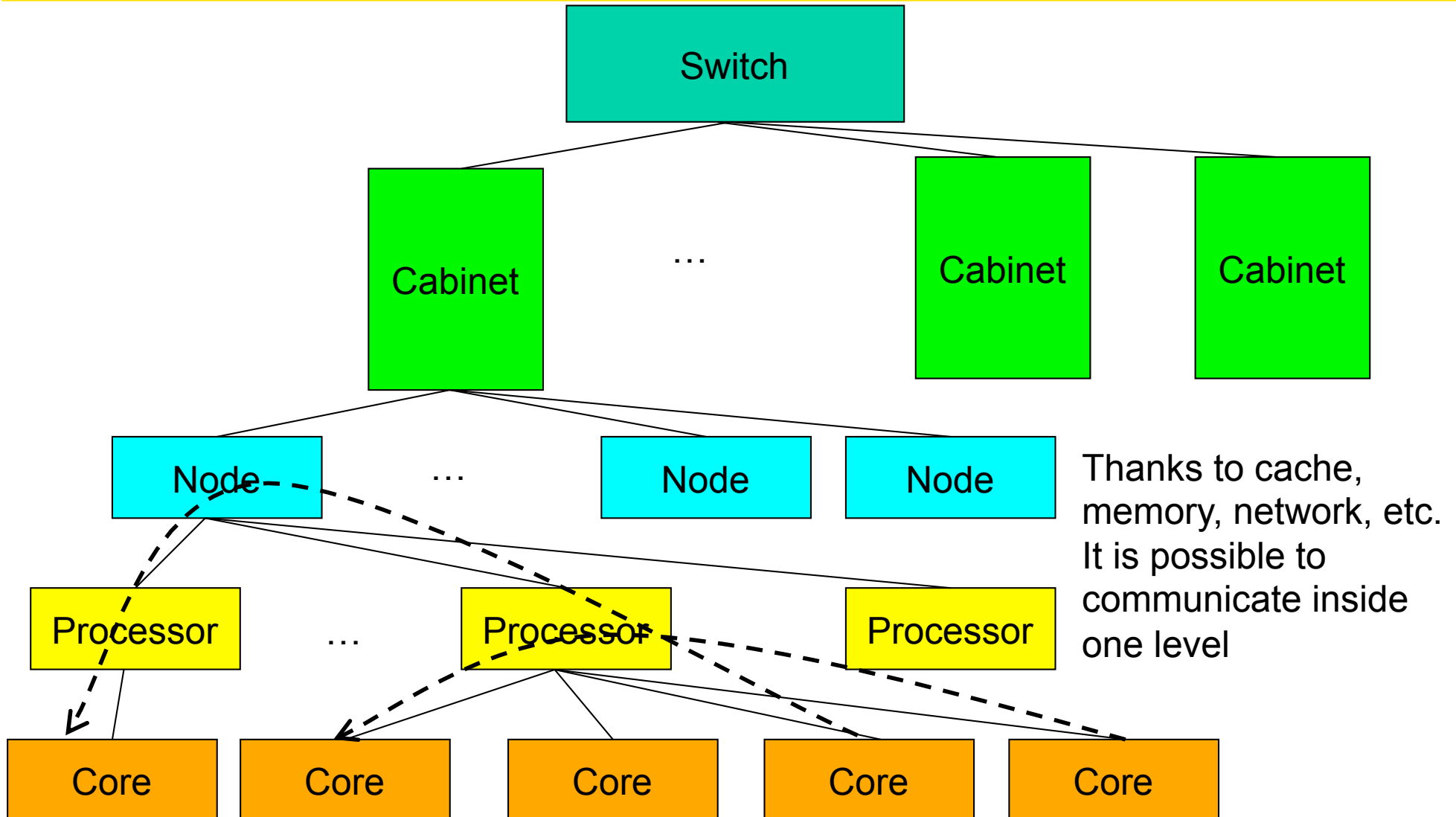
Communication speed depend on:

- receptor and emitter
- Cache hierarchy
- Memory bus
- Interconnection network
- etc.





Example of typical parallel machine

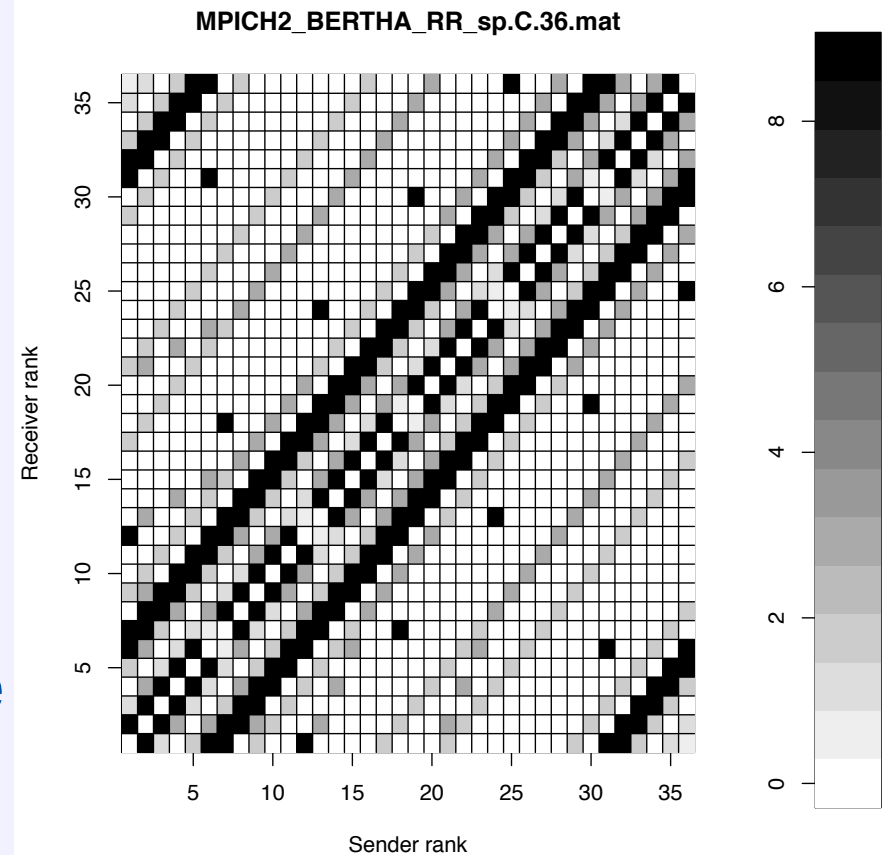




Rationale

Not all the processes exchange the same amount of data

The speed of the communications, and hence performance of the application depend on the way processes are mapped and bind to resources.





Binding or not binding?

Binding is done through syscall (numactl, etc.)

Why not letting the system binding process, he should know what happen?

The scheduler does not take affinity into account:

- Process mapping to core changes during the execution
- Runtime varies from one execution to another
- Pb: cache trashes, I/O



Process placement problem

Given:

- The parallel machine topology
- The processes communication pattern

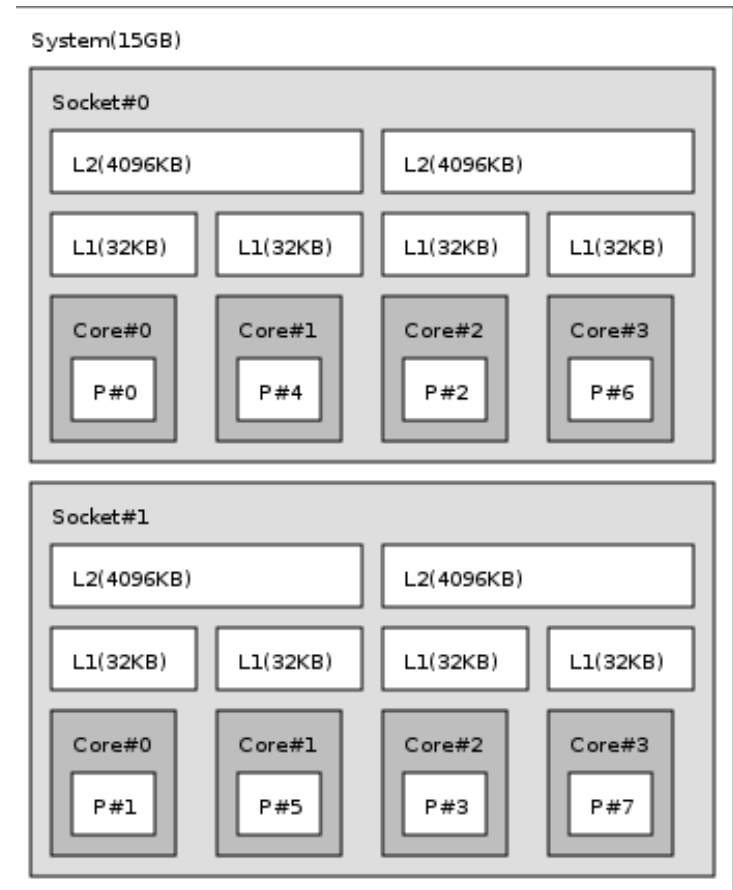
Map processes to resources (cores) to reduce the communication cost.



Obtaining the topology

HWLOC (portable hardware locality)

- Runtime and OpenMPI team
- portable abstraction (across OS, versions, architectures, ...)
- Hierarchical topology
- Modern architecture (NUMA, cores, caches, etc.)
- ID of the cores
- C library to play with
- etc.





Obtaining the communication pattern

No automatic way so far

For now done through application monitoring

Several metrics:

- Messages size
- Number of messages
- Average size

What is the good metric?



State of the art

Process placement fairly well studied problem:

- Graph Partitioning (Scotch/Metis): do not take hierarchy into account.
- [Träff 2002]: placement through graph embedding and graph partitioning
- MPIPP [Chen et al. 2006]: placement through local exchange of processes until no gain is achievable
- [Clet-Ortega & Mercier 09] : placement through graph renumbering



Example

T: topology matrix

0	100	100	10	1000	100	100	10
100	0	10	100	100	1000	10	100
100	10	0	100	100	10	1000	100
10	100	100	0	10	100	100	1000
1000	100	100	10	0	100	100	10
100	1000	10	100	100	0	10	100
100	10	1000	100	100	10	0	100
10	100	100	1000	10	100	100	0

Communication speed between processor 2 and processor 3

Formal problem

Input: T and C two n by n matrices

Output: σ a permutation of size n

Constraint: minimize

$$\sum_{i \neq j} C(i, j) / T(\sigma_i, \sigma_j)$$

C: communication matrix

0	1000	10	1	100	1	1	1
1000	0	1000	1	1	100	1	1
10	1000	0	1000	1	1	100	1
1	1	1000	0	1	1	1	100
100	1	1	1	0	1000	10	1
1	100	1	1	1000	0	1000	1
1	1	100	1	10	1000	0	1000
1	1	1	100	1	1	1000	0

Amount of data exchanged between process 1 and process 3

Example of solutions:

Round-robin: 0 1 2 3 4 5 6 7 241.3

Graph embedding: 3 7 4 0 6 2 5 1 210.52

Optimal (B&B): 0 4 1 5 2 6 3 7 29.08



Complexity of the problem

Finding the optimal permutation is NP-Hard

However, posed this way the problem does not take into account the hierarchy of the topology

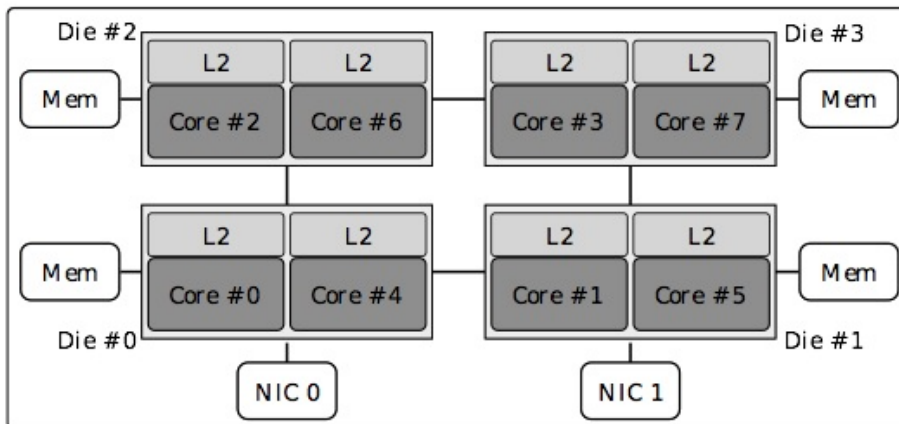
Question: does taking the hierarchy into consideration help?



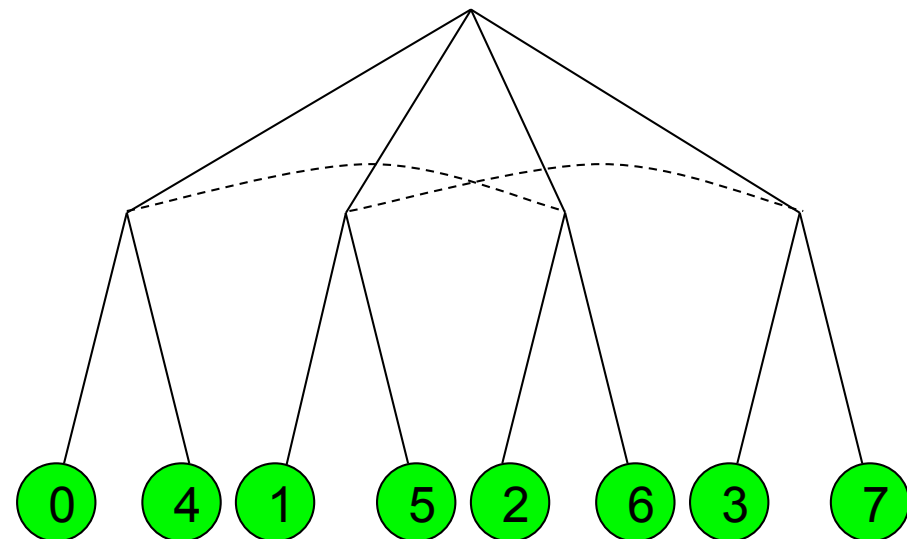
Taking into account the hierarchy

Topology matrix

0	100	100	10	1000	100	100	10
100	0	10	100	100	1000	10	100
100	10	0	100	100	10	1000	100
10	100	100	0	10	100	100	1000
1000	100	100	10	0	100	100	10
100	1000	10	100	100	0	10	100
100	10	1000	100	100	10	0	100
10	100	100	1000	10	100	100	0



HWLOC output





Mapping the communication matrix to the topology tree: the TreeMatch algorithm

Idea: for each level of the tree, group nodes to minimize remaining communication.

Group size should be equal to the arity of the considered level



Example

C: communication matrix

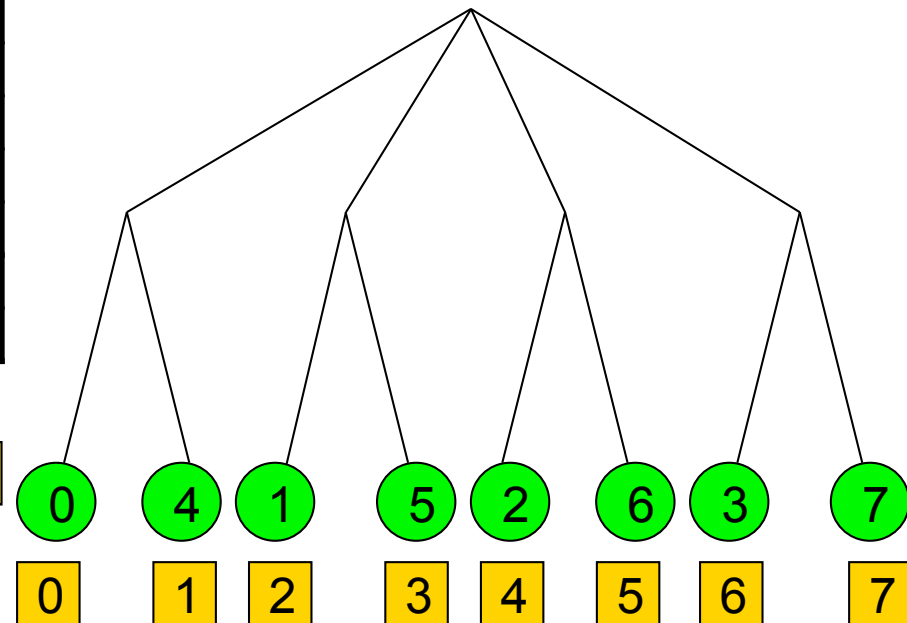
0	1000	10	1	100	1	1	1
1000	0	1000	1	1	100	1	1
10	1000	0	1000	1	1	100	1
1	1	1000	0	1	1	1	100
100	1	1	1	0	1000	10	1
1	100	1	1	1000	0	1000	1
1	1	100	1	10	1000	0	1000
1	1	1	100	1	1	1000	0



+

Grouped matrix

0	1012	202	4
1012	0	4	202
202	4	0	1012
4	202	1012	0





A more complex example

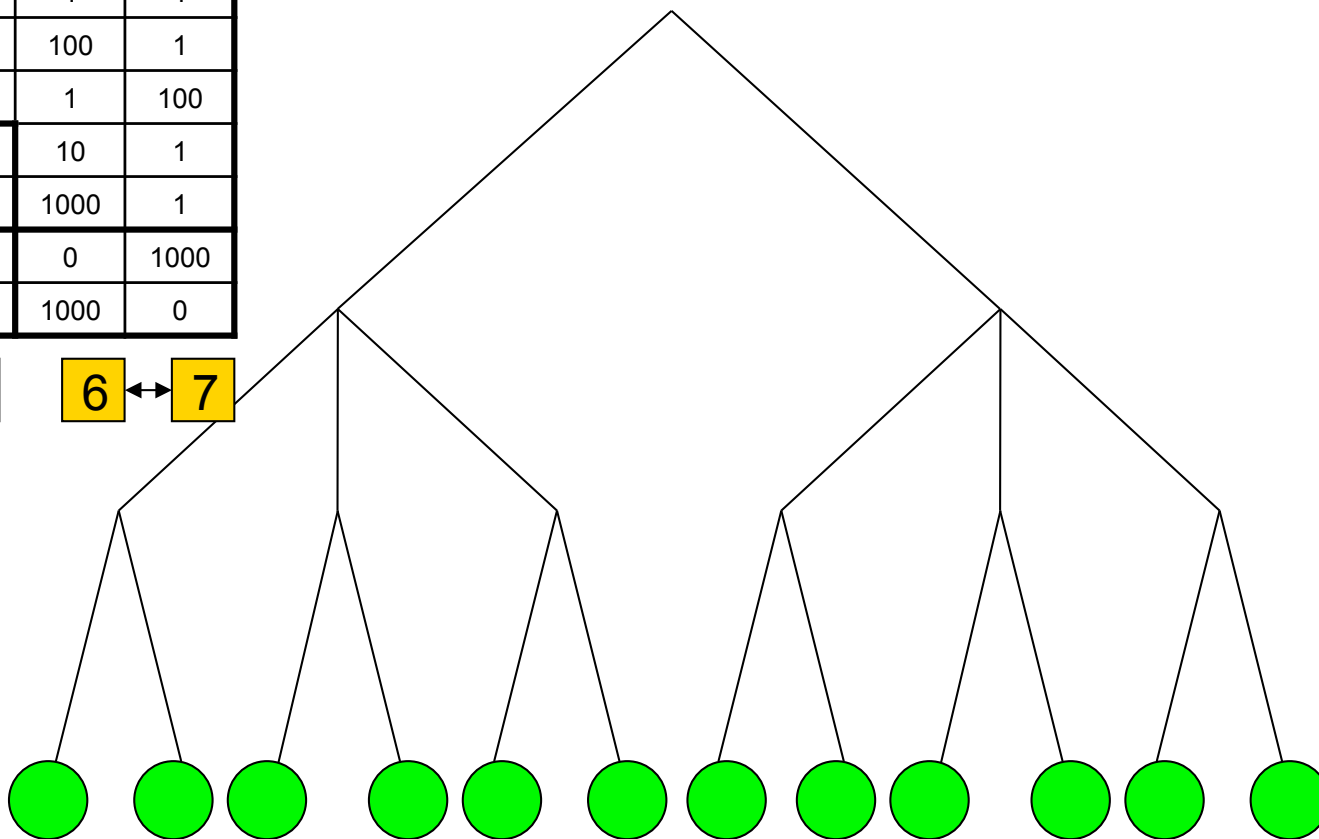
0	1000	10	1	100	1	1	1
1000	0	1000	1	1	100	1	1
10	1000	0	1000	1	1	100	1
1	1	1000	0	1	1	1	100
100	1	1	1	0	1000	10	1
1	100	1	1	1000	0	1000	1
1	1	100	1	10	1000	0	1000
1	1	1	100	1	1	1000	0



+

Grouped matrix

0	1012	202	4
1012	0	4	202
202	4	0	1012
4	202	1012	0





A more complex example

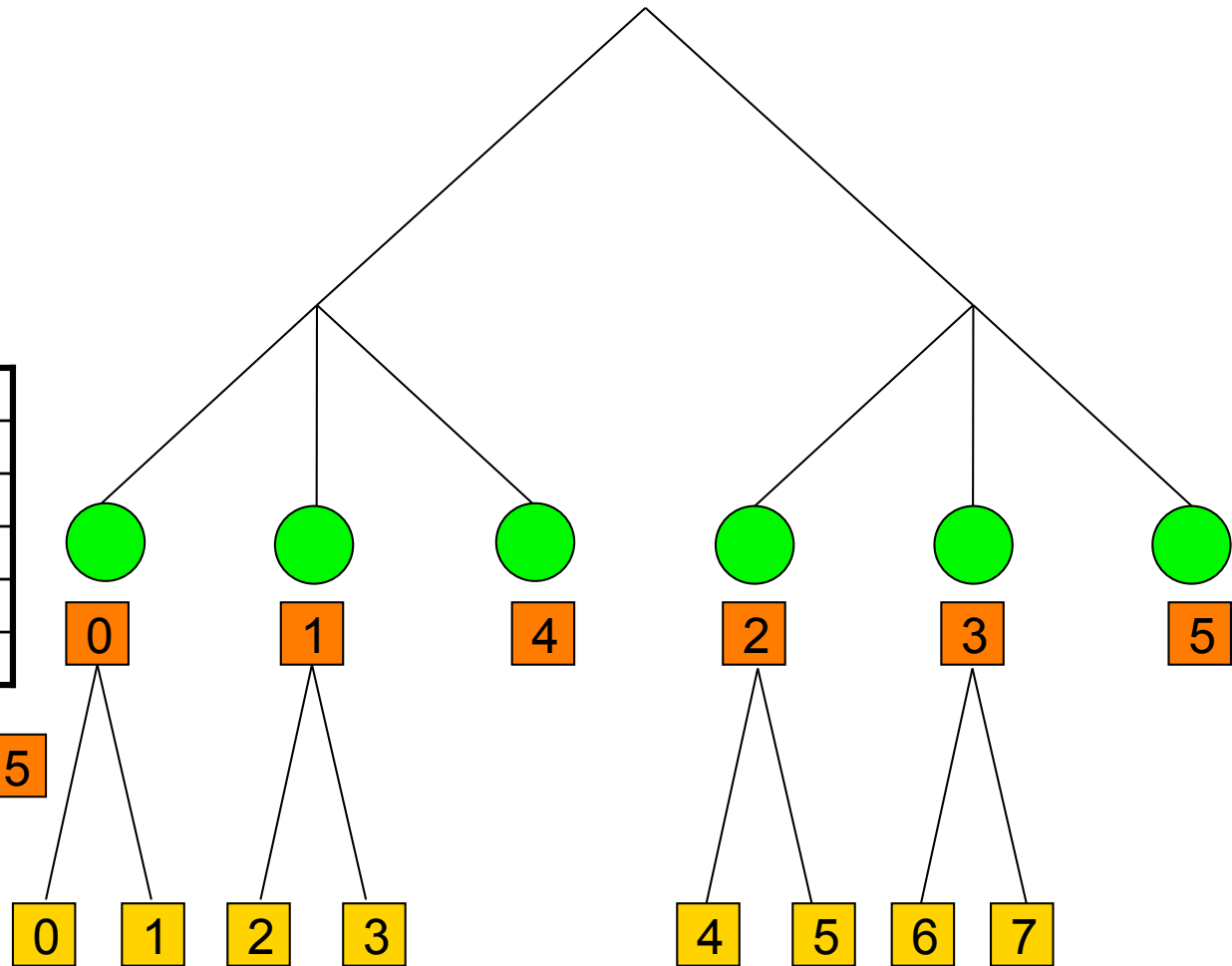


Grouped matrix

0	1012	202	4
1012	0	4	202
202	4	0	1012
4	202	1012	0

Extended grouped matrix

0	1012	202	4	0	0
1012	0	4	202	0	0
202	4	0	1012	0	0
4	202	1012	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0



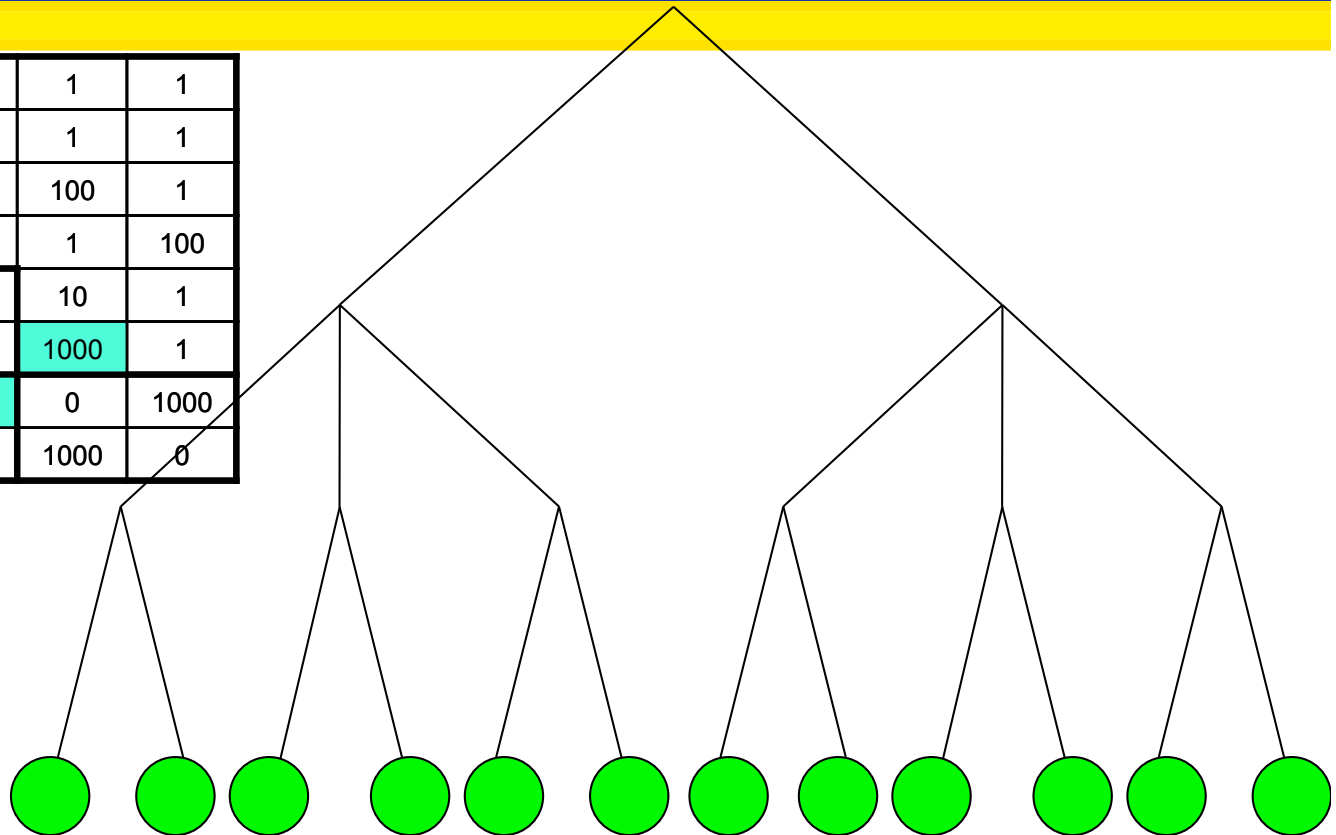
Grouped matrix

0	412
412	0



A more complex example

0	1000	10	1	100	1	1	1
1000	0	1000	1	1	100	1	1
10	1000	0	1000	1	1	100	1
1	1	1000	0	1	1	1	100
100	1	1	1	0	1000	10	1
1	100	1	1	1000	0	1000	1
1	1	100	1	10	1000	0	1000
1	1	1	100	1	1	1000	0



TreeMatch:

0	1	2	3			4	5	6	7		
---	---	---	---	--	--	---	---	---	---	--	--

Packed:

0	1	2	3	4	5	6	7				
---	---	---	---	---	---	---	---	--	--	--	--

Packed solution worst than the TreeMatch one because there is a large communication between processes 5 and 6



TreeMatch properties

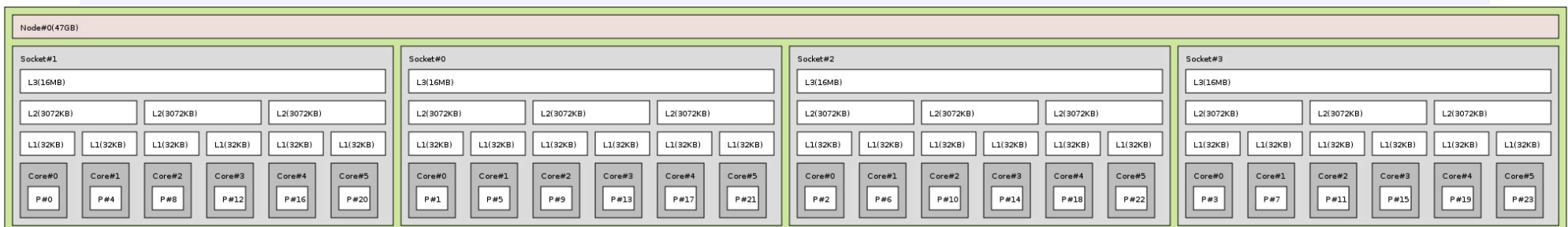
- Work if there is more cores than processes (by adding virtual processes that do not communicate)
- Work whatever the arity of a given level (do not need to be binary)
- **Optimal** if the communication pattern is hierarchic and symmetric and the topology tree is balanced



Experiments

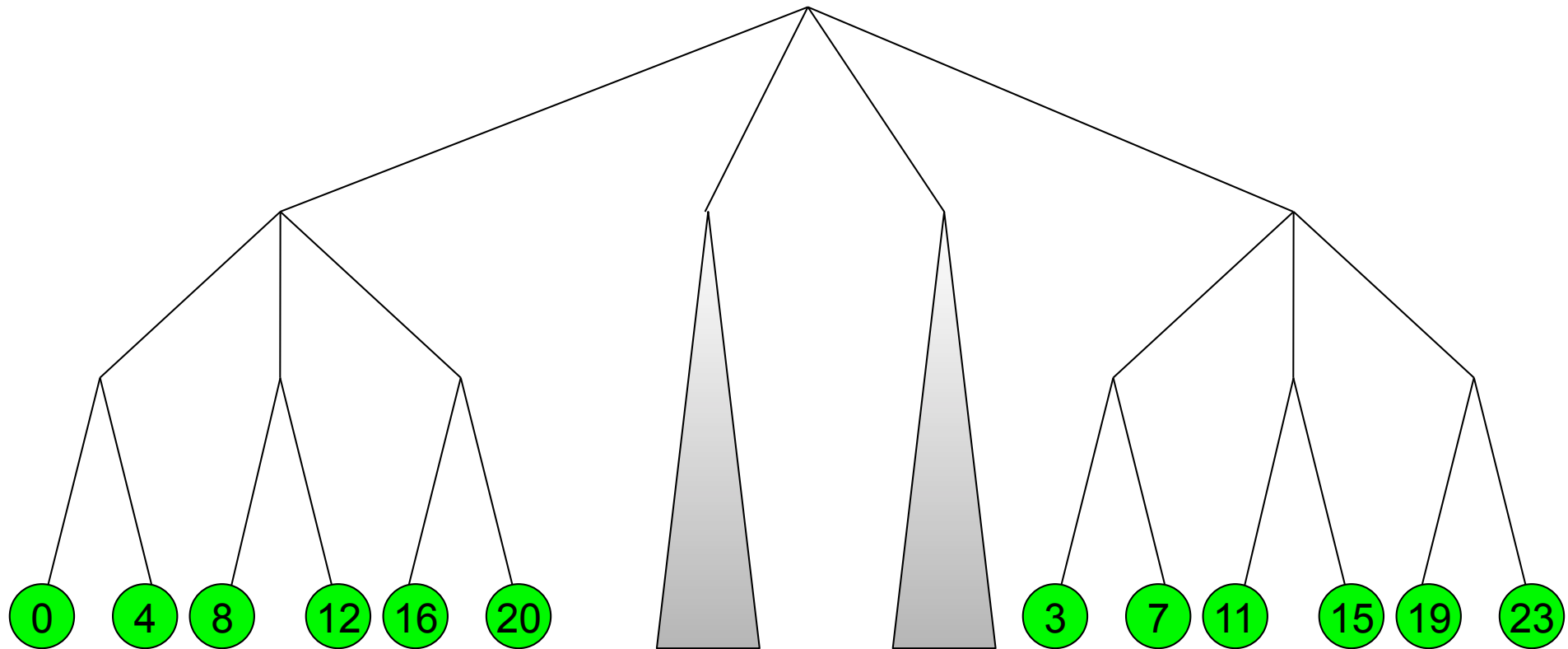
We use the NAS benchmarks:

- All the kernels
- Class: A,B,C,D
- Size: 16, 32/36, 64
- On highly NUMA machine (4 nodes of 4 Xeon quad-core Dunnington)
- Comparison with : MPIPP [Chen et al. 2006] (two versions), Packed (by sub-tree), Round-Robin (process i to core i).





1 of the 4 nodes of the target machine

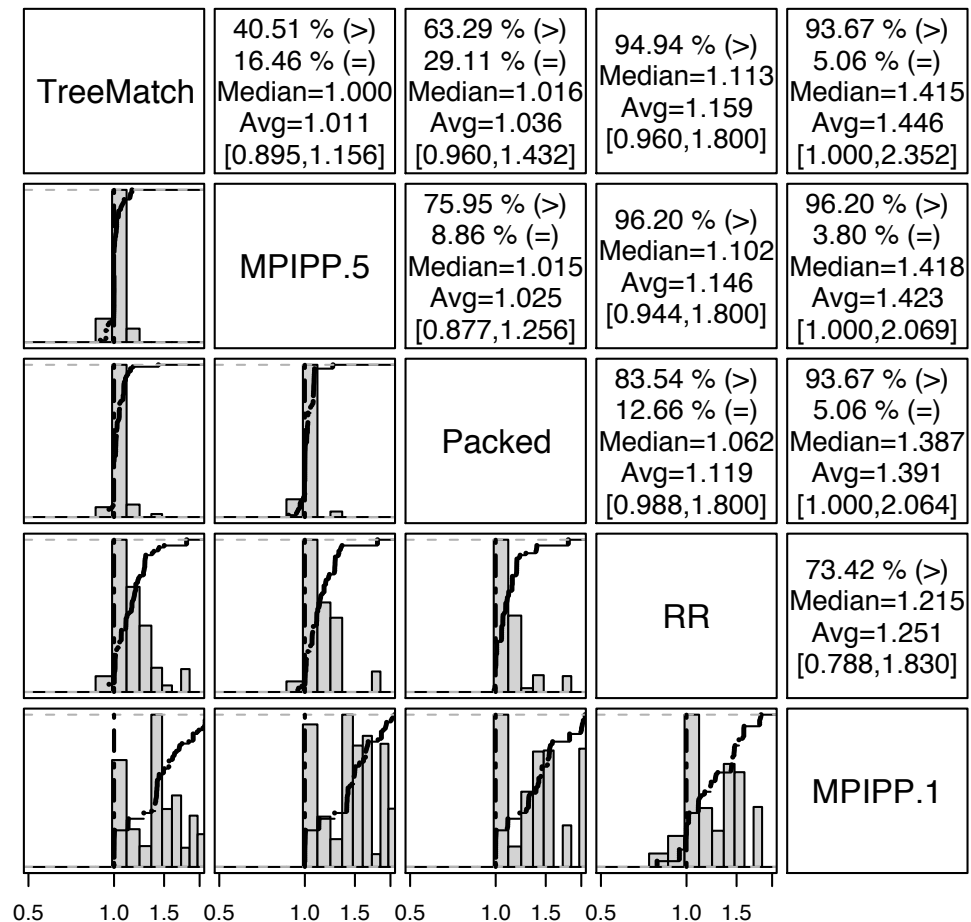




Simulation Results

We simulate the execution time using our model

Sim NAS ALL



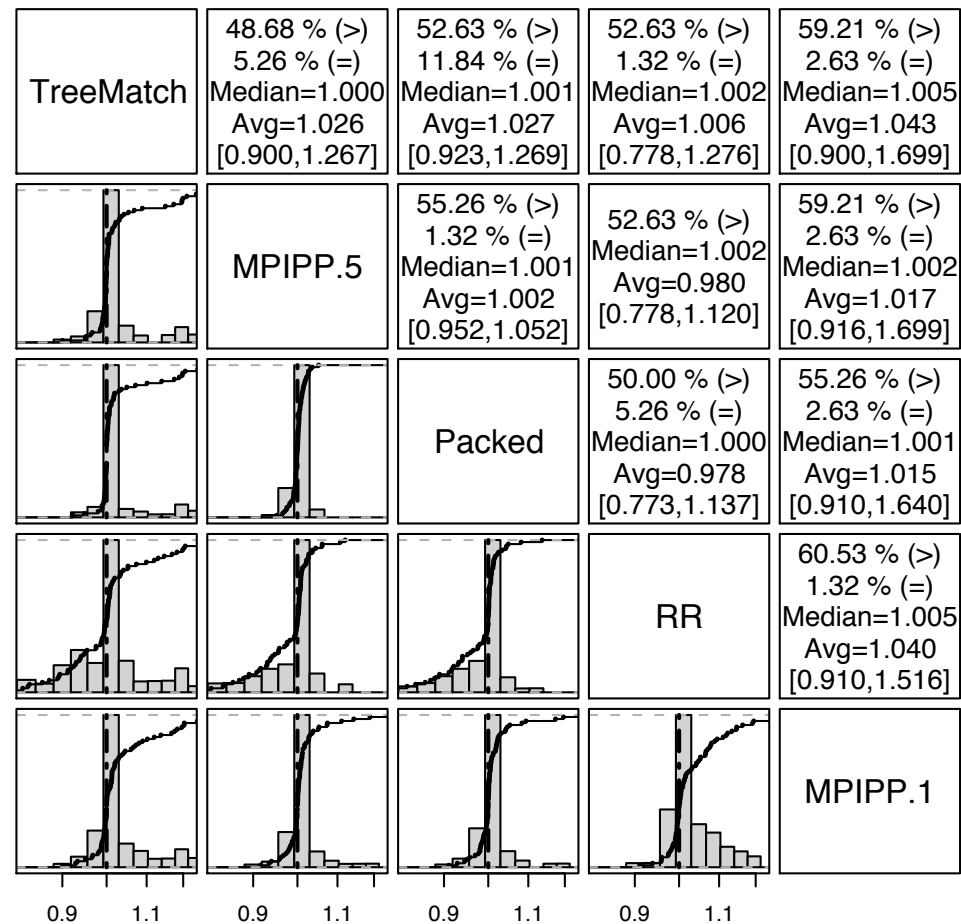


NAS on the real machine

Best strategy:
TreeMatch

Some very bad
results against
round-robin

Bertha ALL





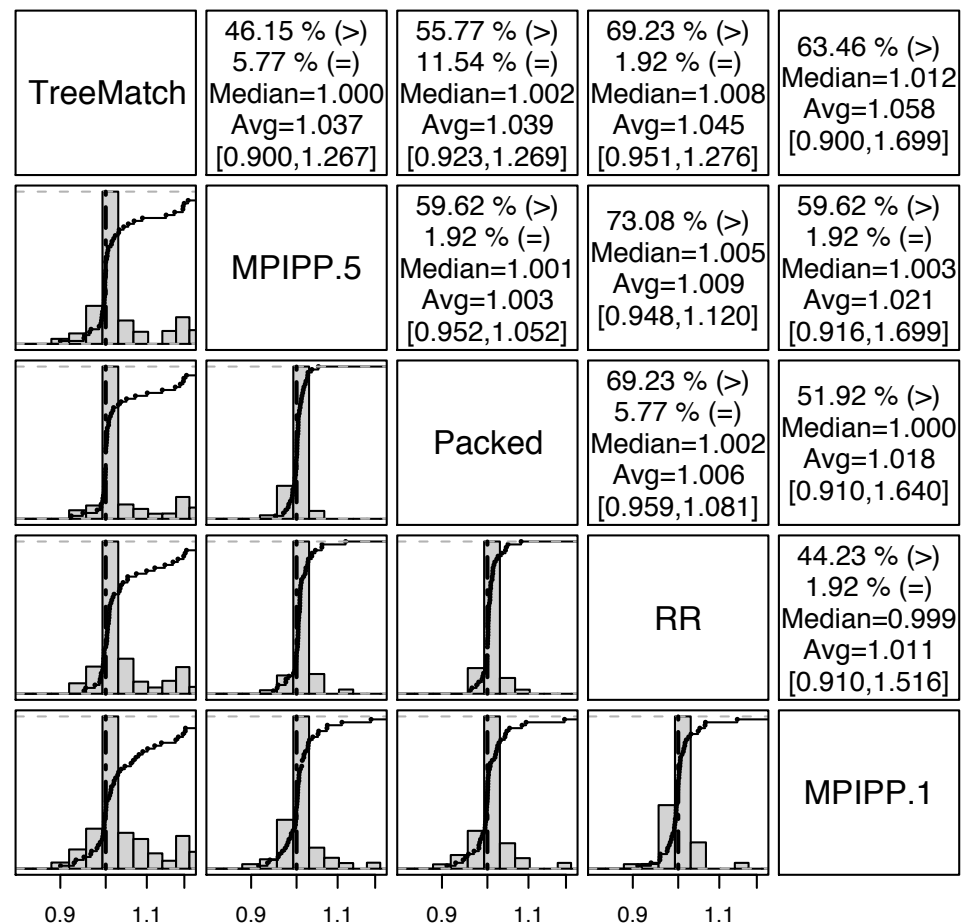
32-64 processes

Several nodes are used

Best: TreeMatch (up to 27% improvement).

Comparable to MPIPIP.5 (but faster runtime)

Bertha 64-32



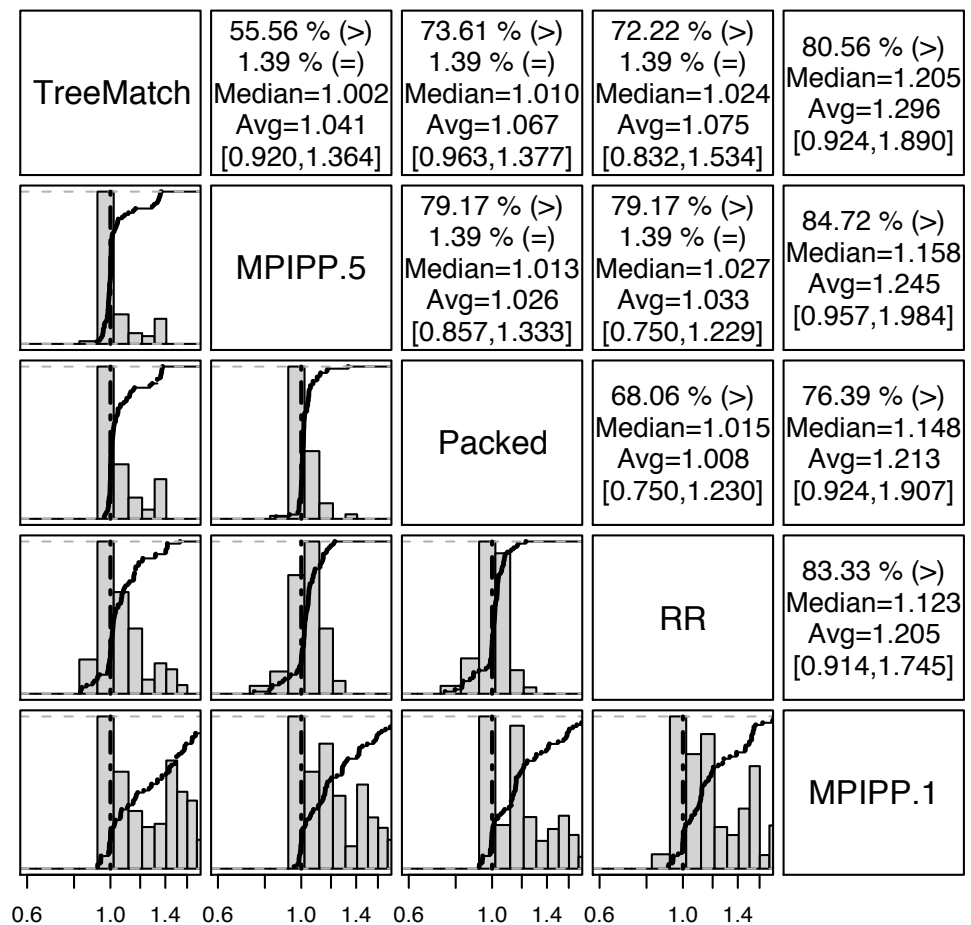


Communication Only Application

We extract the communication pattern of the NAS

Up to 37% improvement

Model ALL



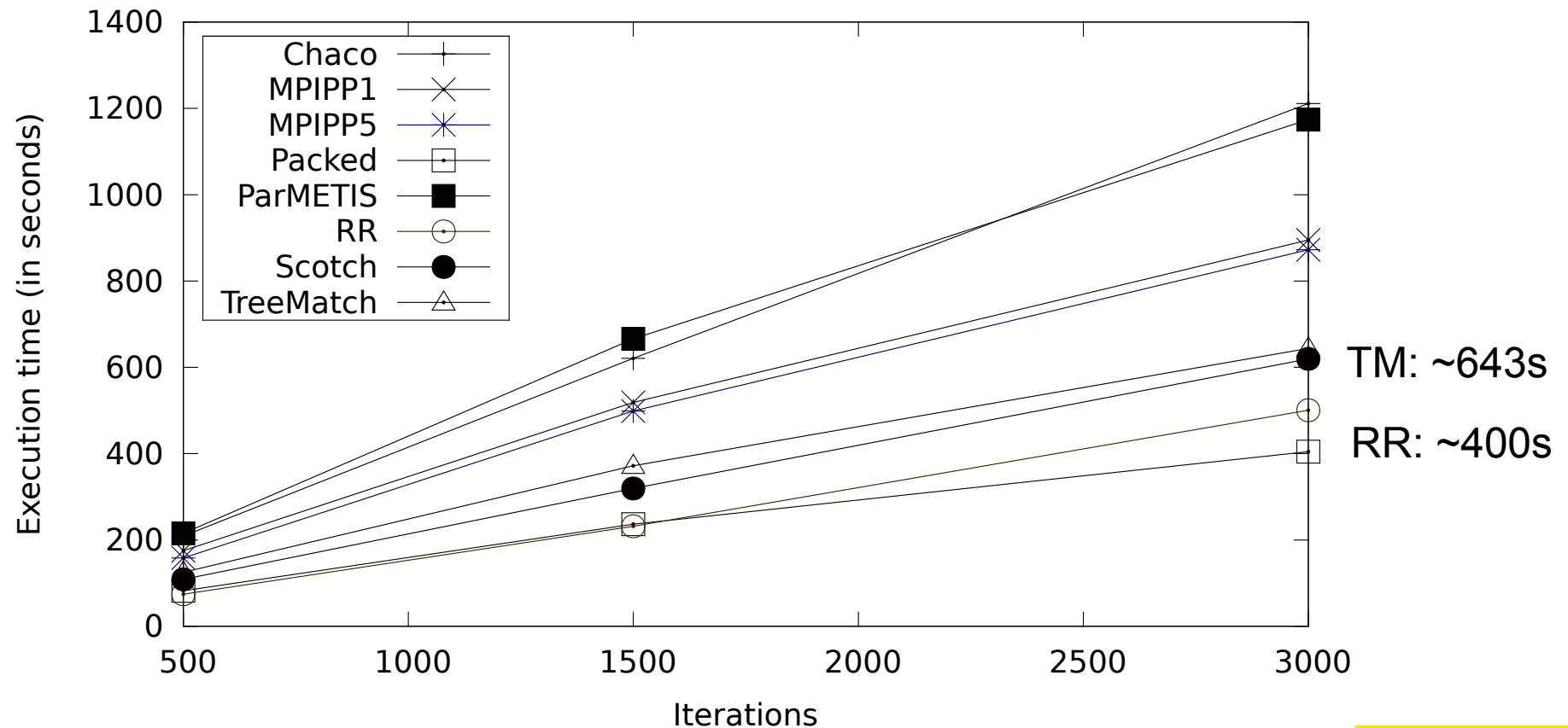


Zeus Application case of different metrics

Zeus: Radiating and Magnetized Flows Simulator

Infiniband cluster Intel Nehalem. Open MPI: 256 processes

Processes binding on ZeusMP CFD application - metric : avg - processus : 256



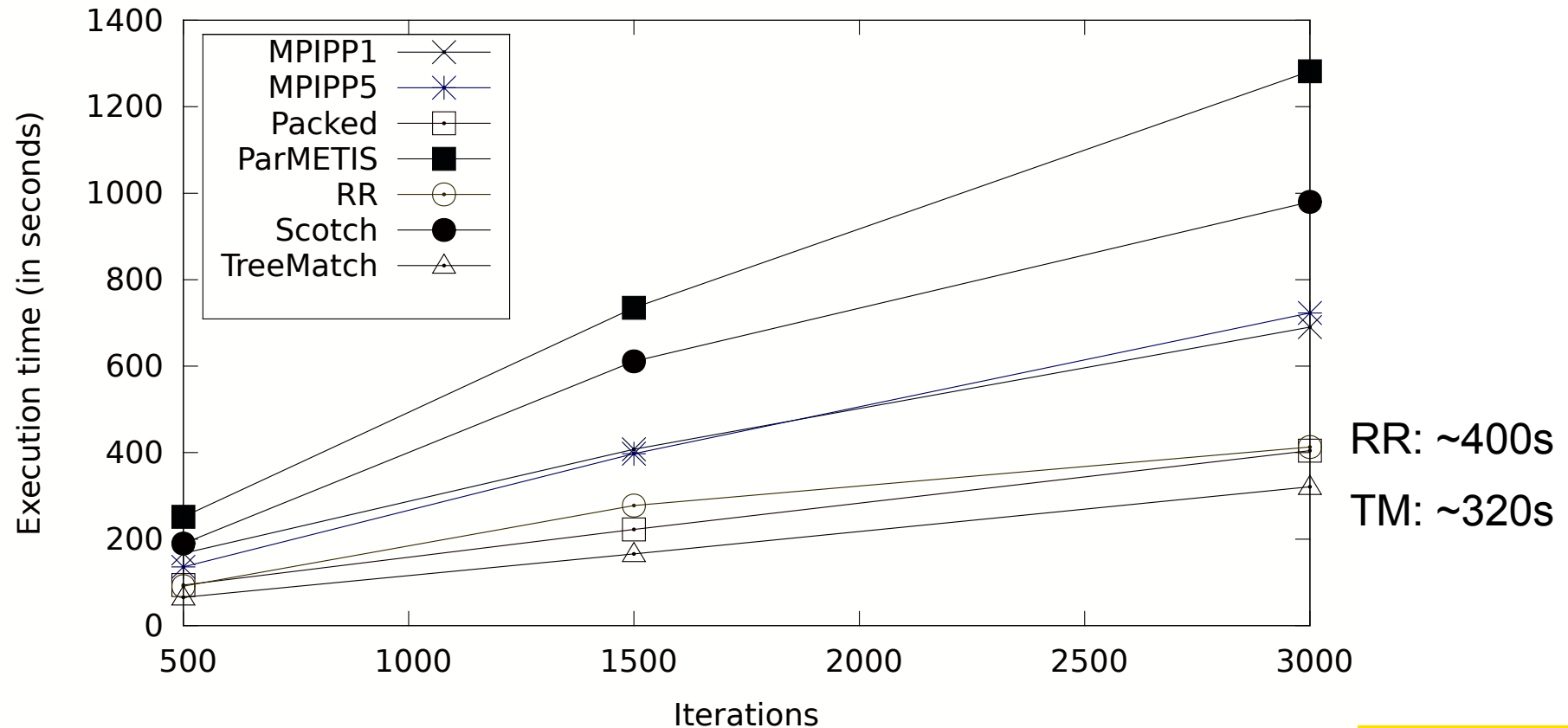


Zeus Application case of different metrics

Zeus: Radiating and Magnetized Flows Simulator

Infiniband cluster Intel Nehalem. Open MPI: 256 processes

Processes binding on ZeusMP CFD application - metric : size - processus : 256



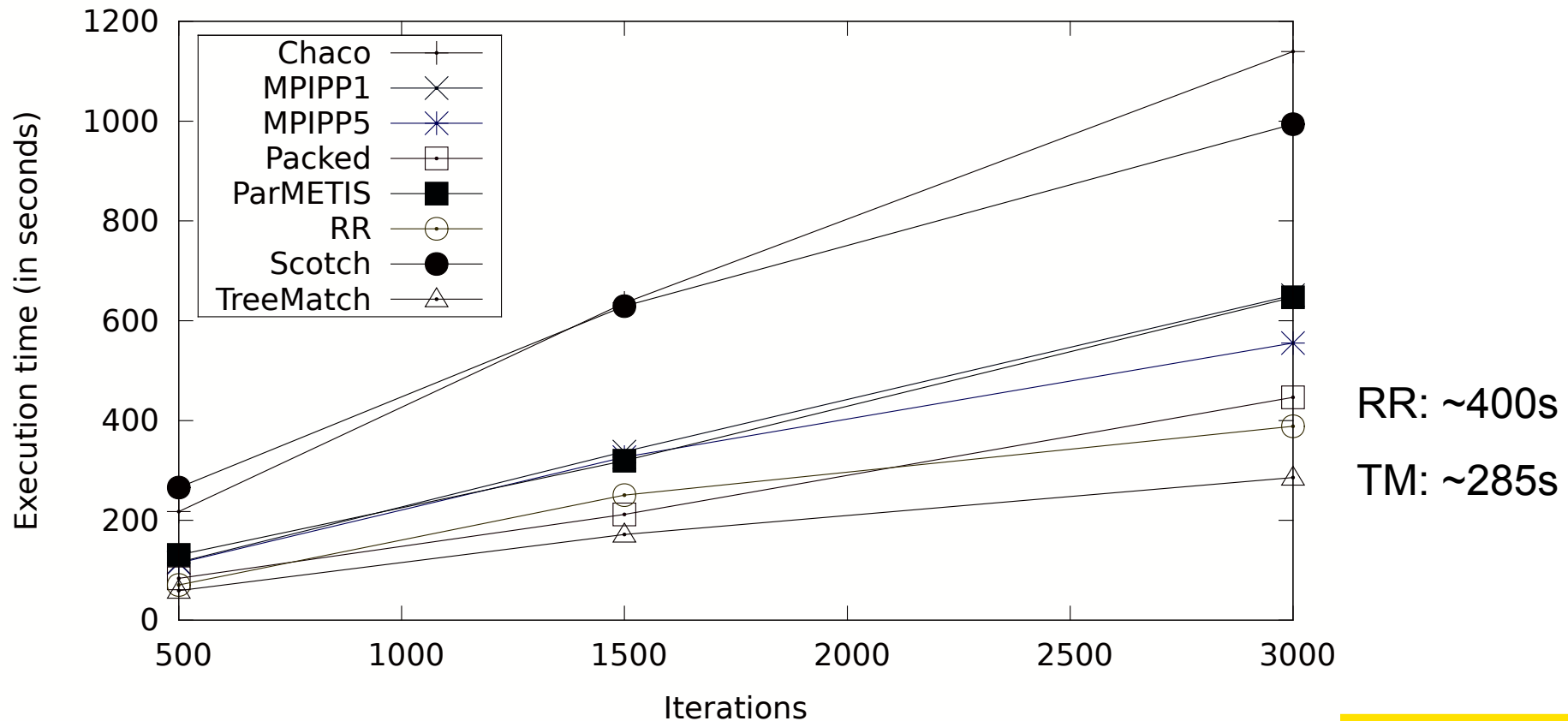


Zeus Application case of different metrics

Zeus: Radiating and Magnetized Flows Simulator

Infiniband cluster Intel Nehalem. Open MPI: 256 processes

Processes binding on ZeusMP CFD application - metric : msg - processus : 256





Conclusion

Mapping processes can help to reduce the communication cost

TreeMatch: an algorithm to perform such mapping

Implemented as a standalone tool

Implemented in Dev branches of OpenMPI and MPIVH2

Results:

- TreeMatch: best method on average (outperform Metis and Scotch)
- Some apps are more sensitive to binding than others (CFD : good candidate)
- The way we model application behavior has a huge impact.



Future work

On going work

Future work:

- Test real applications
- Top Down?
- Improve model (NUMA effect)
- Hybrid case
- Dynamic adaptation
- Automation
- Process topology interface of MPI 2.2 (With J. L. Träff).