Process Mapping on any Topologies with TopoMatch

Emmanuel Jeannot

March 4, 2021

01

Process Placement Background

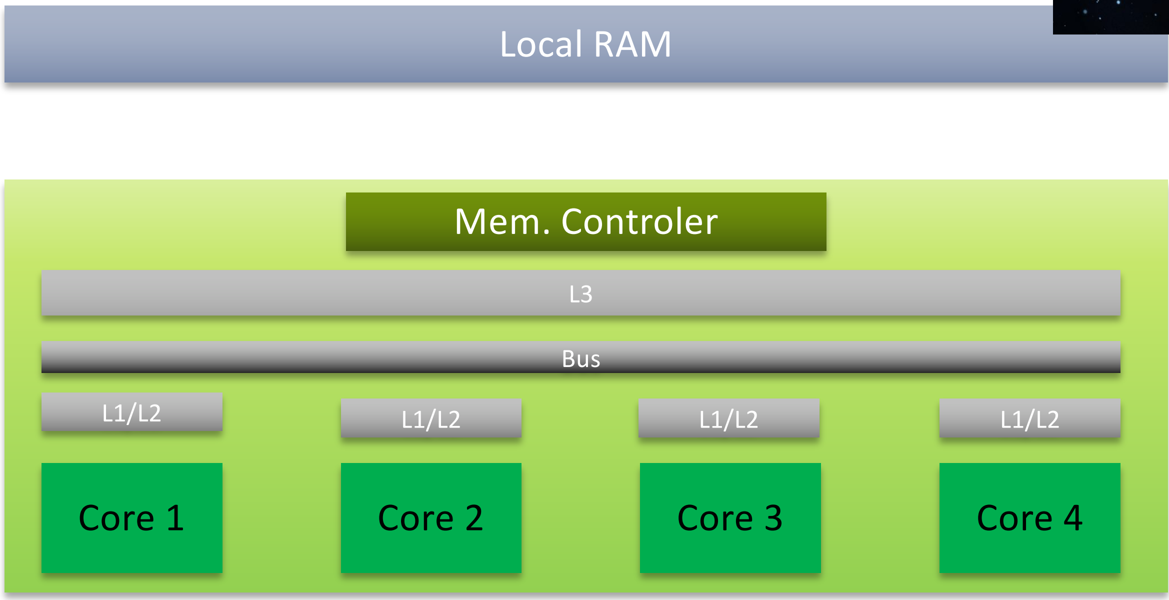
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange

Inria

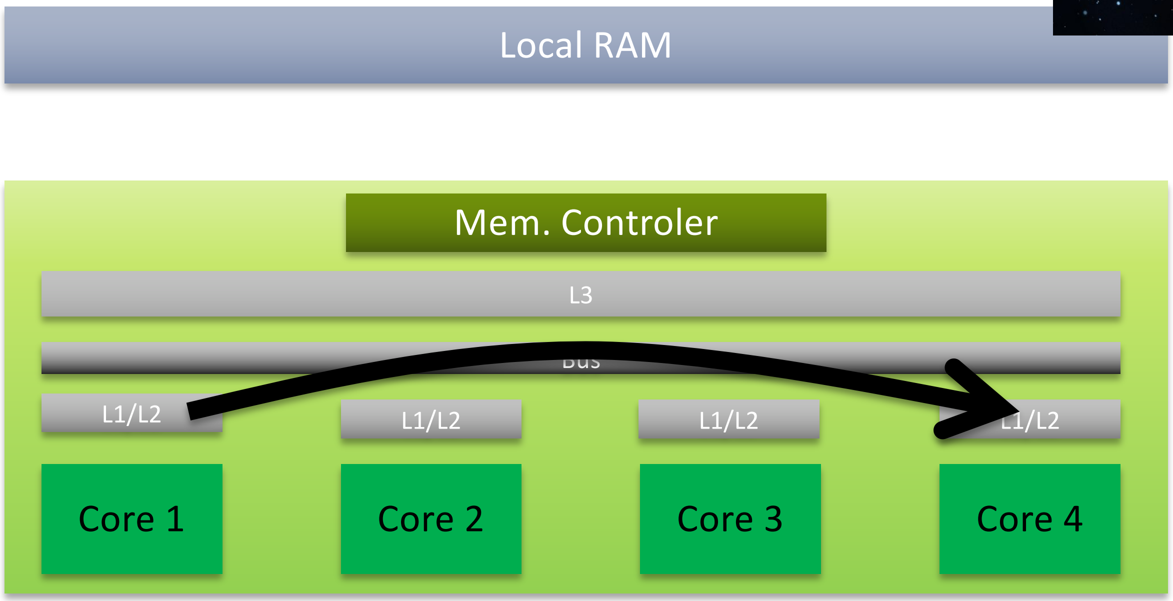
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange



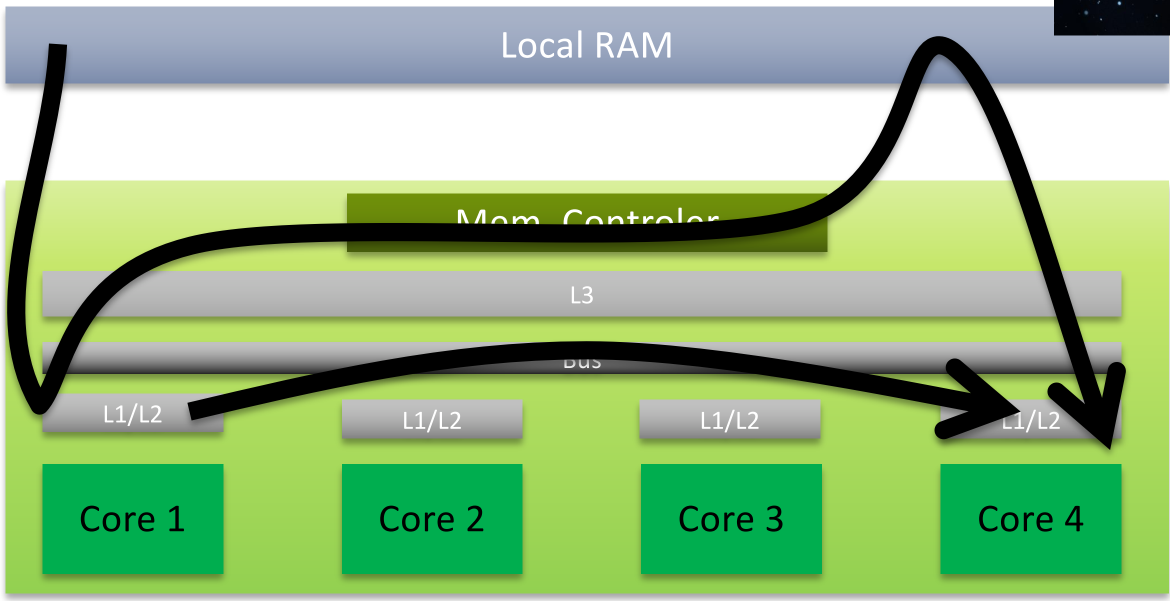
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange



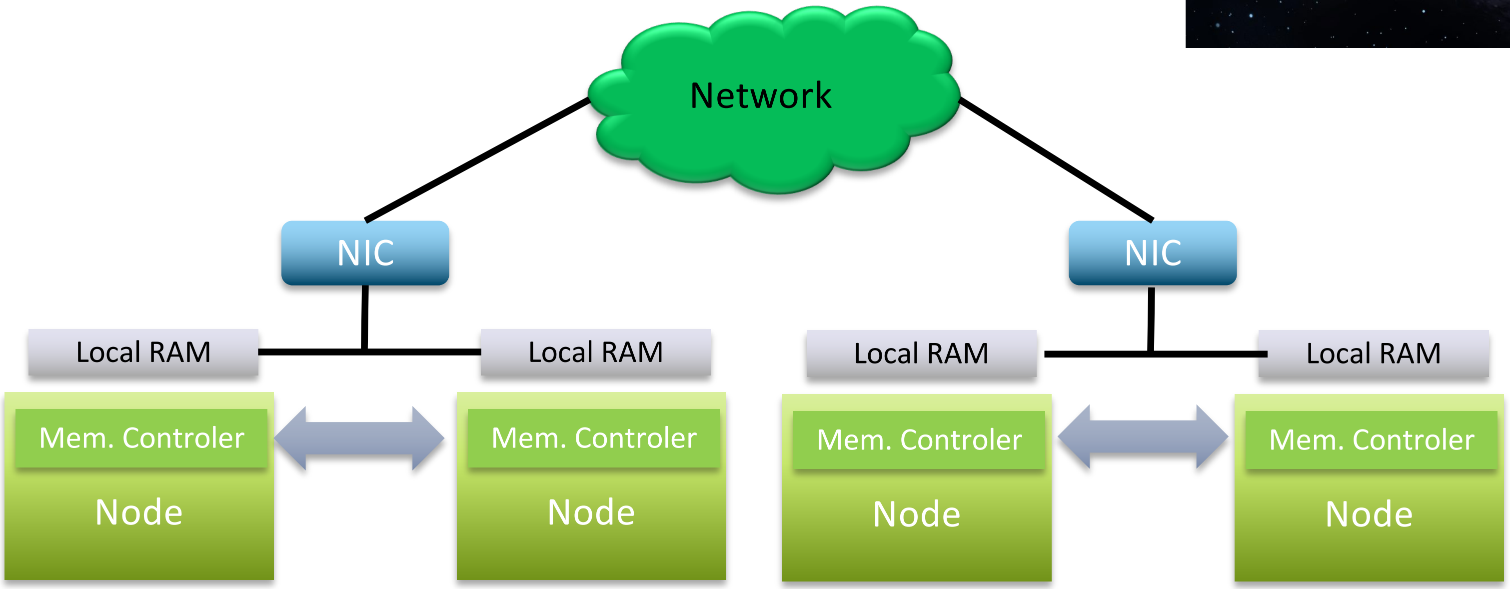
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange



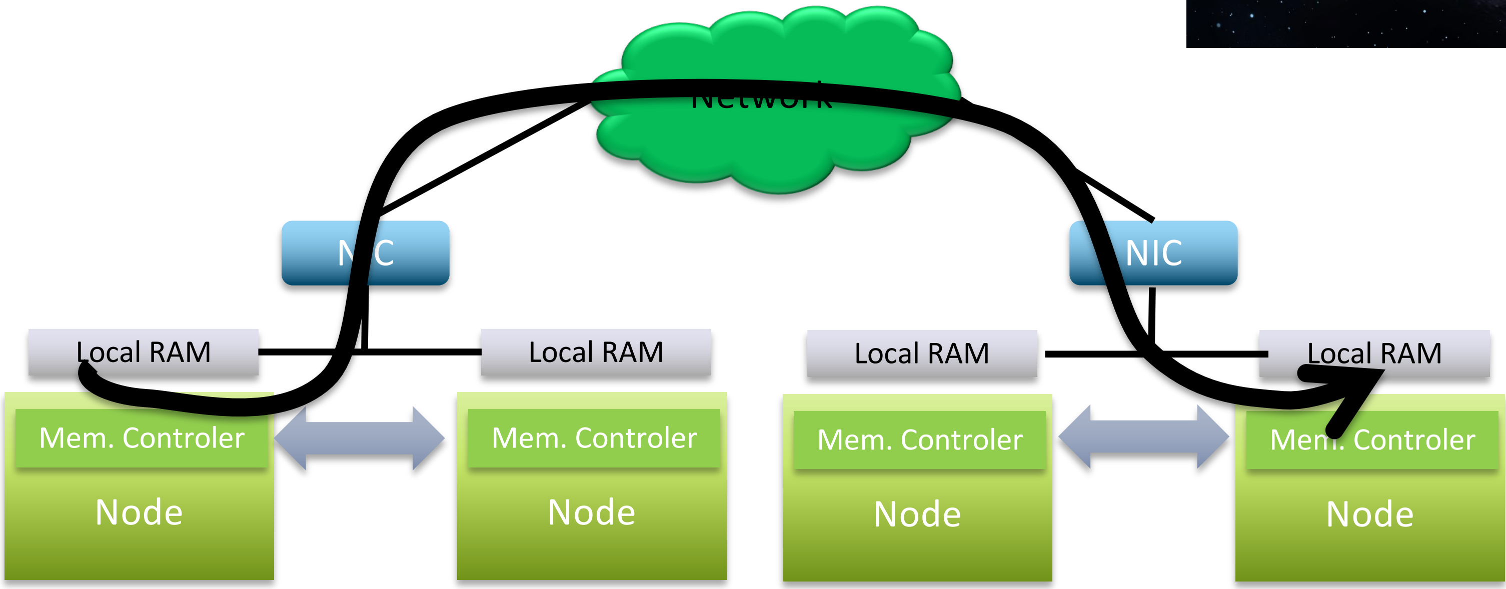
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange



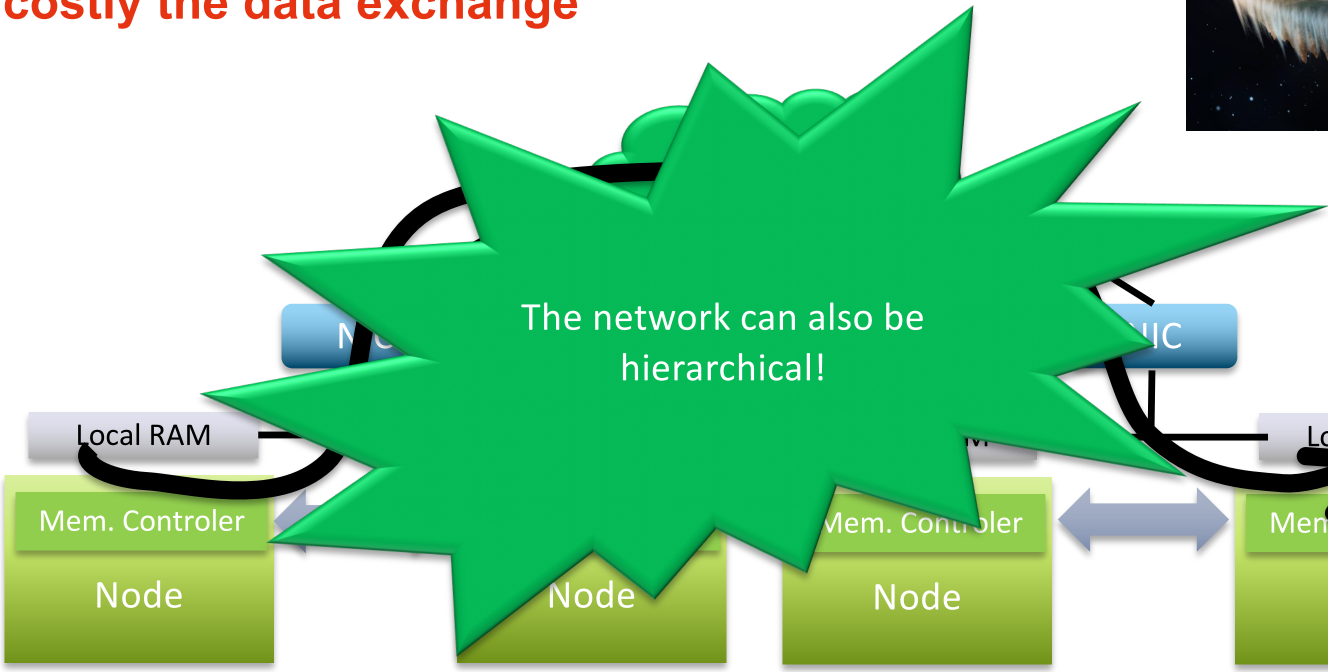
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange



The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange



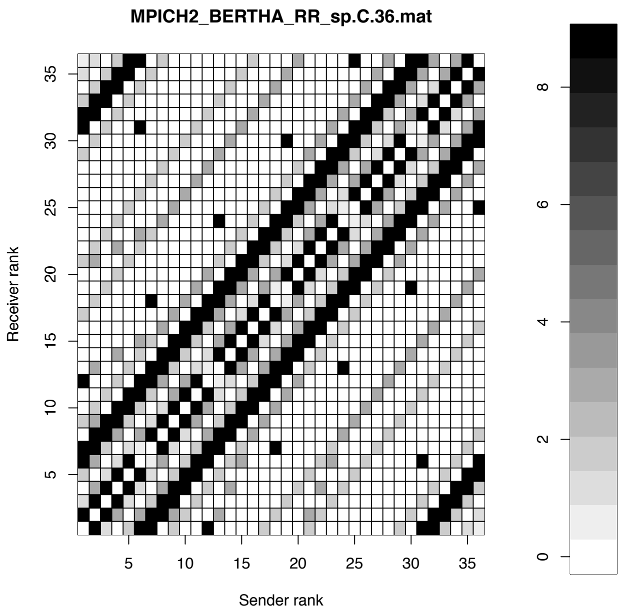
Communication Pattern

Shared memory system:

- **The amount of data shared by threads vary**

Distributed memory system:

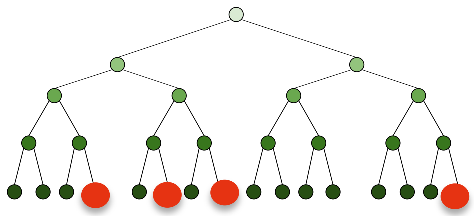
- **The amount of data exchanged between processes vary**



The time spent to exchange data depends on the thread/process mapping

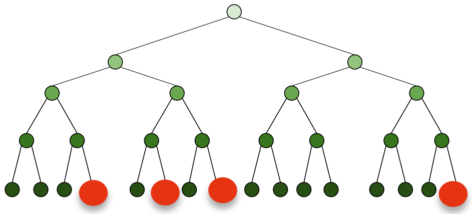
Process Placement Problem

Process Placement Problem

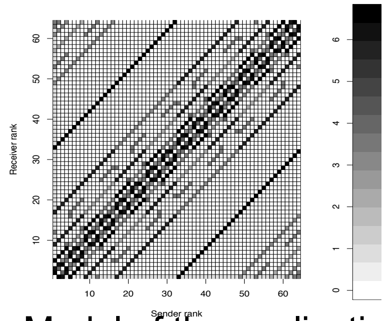


Model of the machine

Process Placement Problem

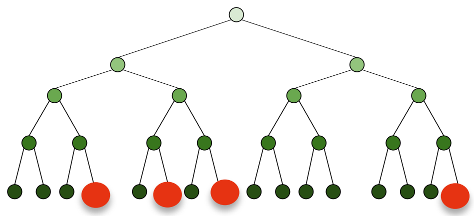


Model of the machine

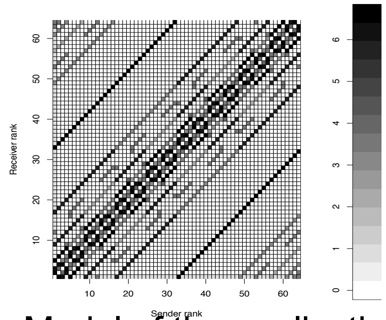


Model of the application

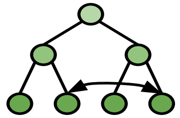
Process Placement Problem



Model of the machine

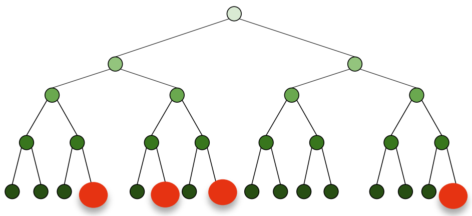


Model of the application

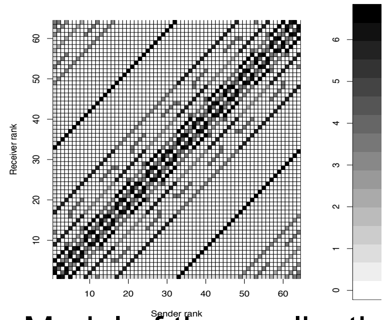


Mapping algorithm

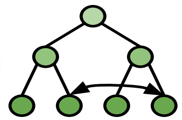
Process Placement Problem



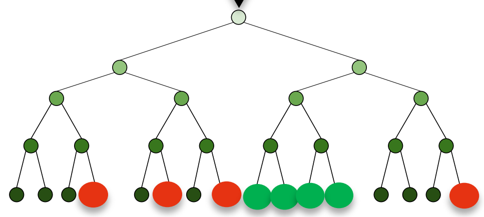
Model of the machine



Model of the application



Mapping algorithm



02

TreeMatch

Project started in 2009

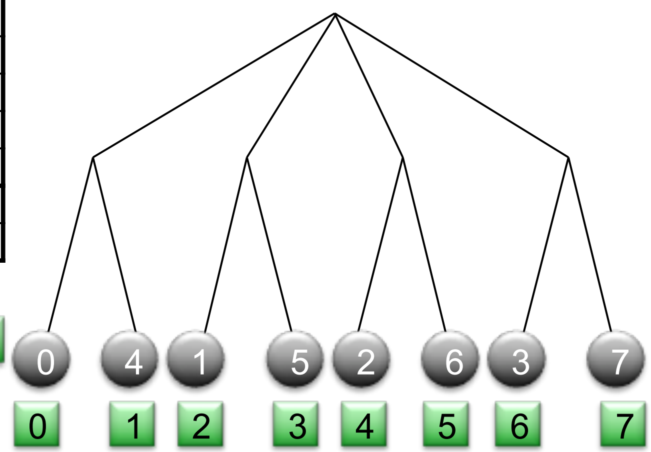
Many contributors:

- **Guillaume Mercier (intégration dans Open MPI)**
- **François Tessier (LB, constraints)**
- **Adèle Viliermet (Batch scheduler)**
- **Pierre Celor (Partitionning algorithm)**
- **Fatima El-Akkary (SW eng., noise analysis)**
- **Thibaut Lausecker (Scotch Interface)**
- **Laurent Dutertre (Preliminary XP)**

TreeMatch Basic Algorithm

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

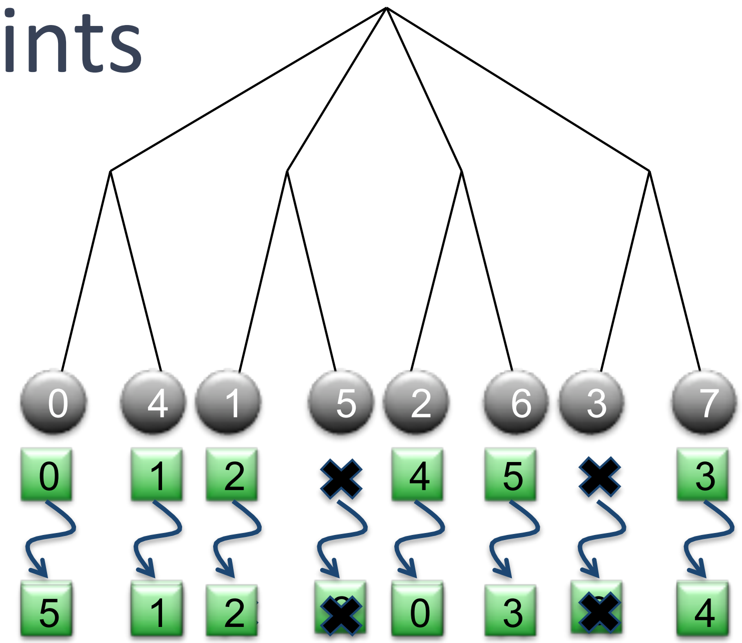
0	1012	202	4
1012	0	4	202
202	4	0	1012
4	202	1012	0

Communication matrix + Tree Topology
= Process permutation

Dealing with Constraints

Problem:

- Given a hierarchichal topology
- An already mapped application onto a subset of the nodes
- Reorder process while ensuring only this subset is used



Solution:

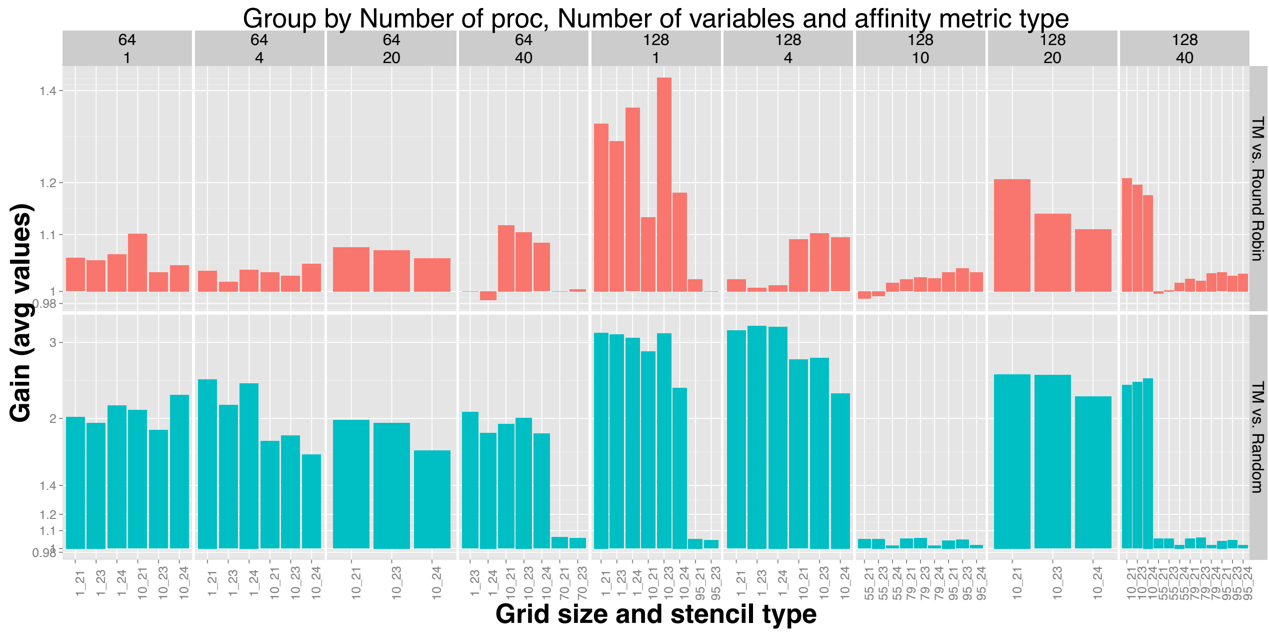
- Extend the communication matrix with dummy nodes
- Process the tree backward by doing k-partitionning
- Force each partition to have the right number of dummy nodes
- Process recursively

03

Use-Cases

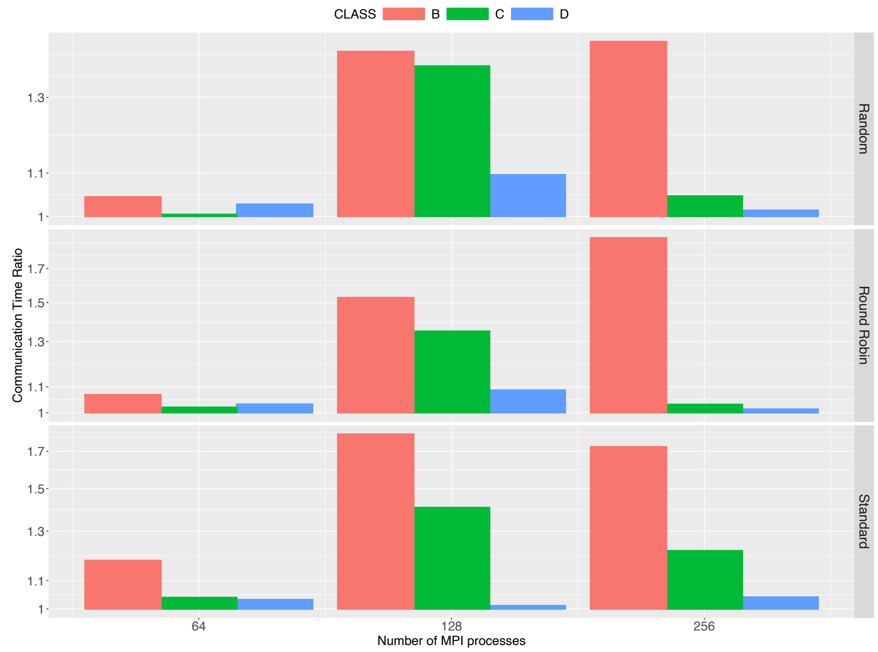
Use-Case 1: Process Mapping

MiniGhost Application (Stencil)



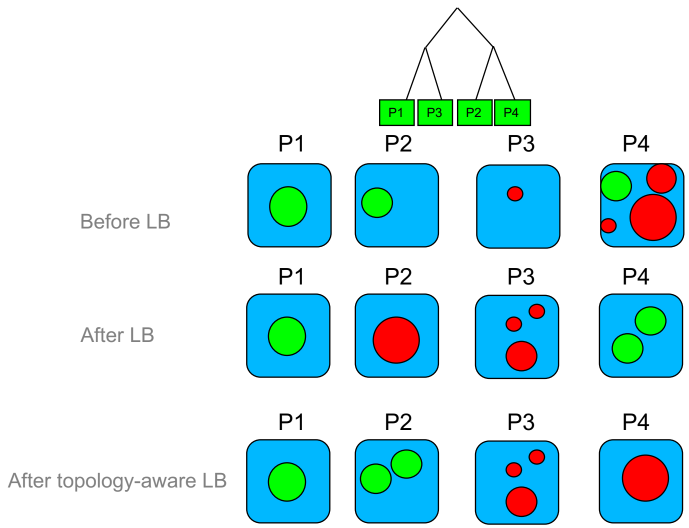
Use-Case 2: Rank reordering

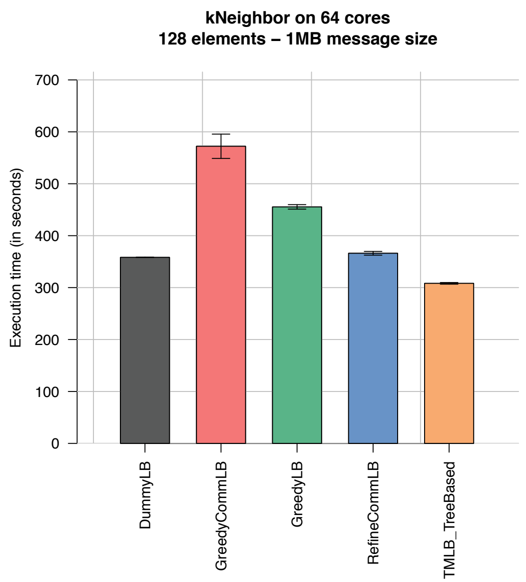
1. **Gather communication pattern**
2. **Compute new mapping**
3. **Change communicator**
4. **Exchange data**
5. **Continue computation with new communicator**

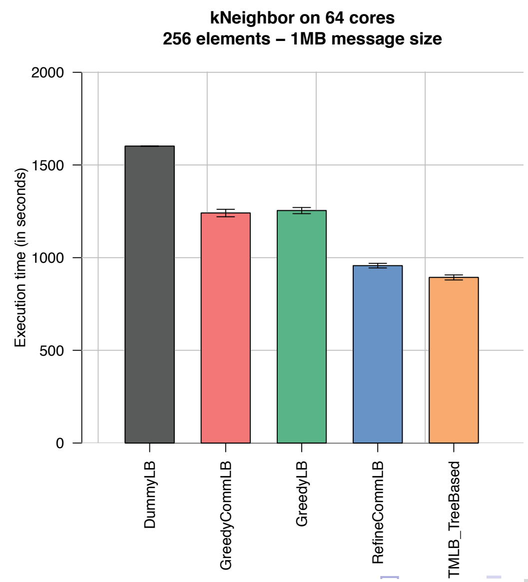


Case of Conjugate Gradient (CG –NAS).

Topology-Aware Load Balancing



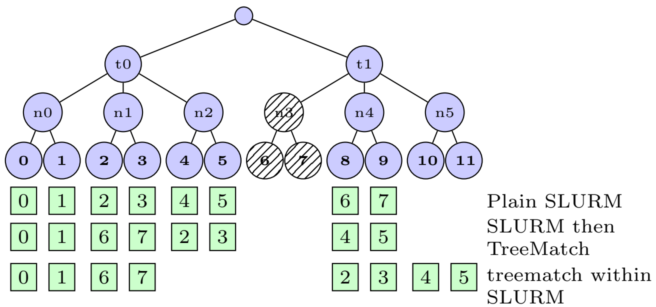


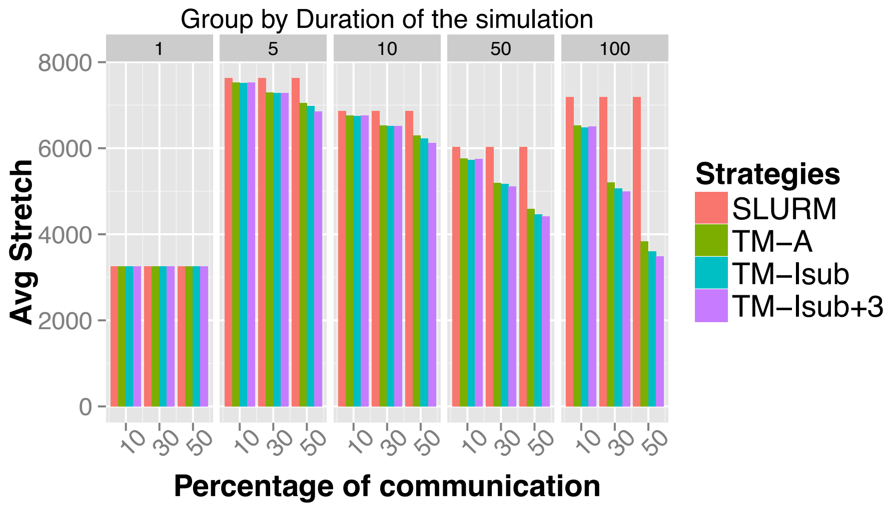


Implemented within Charm++

Batch Scheduling

1. **Gather pattern before submitting job**
2. **Use TreeMatch to allocate resources to the job**





04

TopoMatch

TreeMatch is limted to Tree Topology

Scotch (https://gitlab.inria.fr/scotch/scotch): a software package for

- **graph and mesh/hypergraph partitioning,**
- **graph clustering**
- **sparse matrix ordering**

Lift this limitation:

- **Scotch already used in TreeMatch: core graph partitionning**
- **Scotch manage different type of architectures**
  - Decomposition-defined (deco)
  - Specific (Mesh, hypercube, tleaf), etc.

Topomatch: Managing Scotch in TreeMatch

Same interface and same set of features:

- **If standard tree topology : use TreeMatch**
- **If other topologies : use Scotch**

Important features:

- **Any kind of topology (including Hwloc)**
- **Manage constraints**
- **Manage oversubscribing**
- **Different evaluation metric (Hope-Byte, Sum-Com, Max-Com)**
- **Optional exhaustive search**
- **Fast mapping (multithreaded)**
- **Fast I/O**
- **Nice verbosity management**

Using Scotch

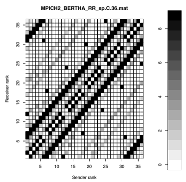
With constraints

```
C : constaint
T : The Scotch topology target Input:
m : The communication matrix

SCOTCH_archInit(sub_arch);
SCOTCH_archSub(sub_arch T, |C|, C);
local_sol ← scotch_partitioning(sub_arch, m);

// Renumber solution to change frame of
reference;
foreach i in 0..|C| − 1 do
   global_sol[i] ← C[local_sol[i]];
```

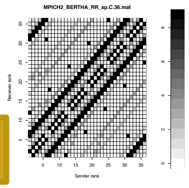
Without constaints

```
T : The Scotch topology target
m :  The communication matrix

graph ← com_mat_to_scotch_graph(m, |T|×
sparse_factor);

strat ← set_scotch_strategy(SCOTCH_STRATBALANCE);
partition ← SCOTCH_ComputeMapping (graph, T, strat);
```

Bucket grouping (group of size 2)



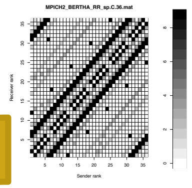
Bucket grouping (group of size 2)



Sample communication
matrix values

Inria

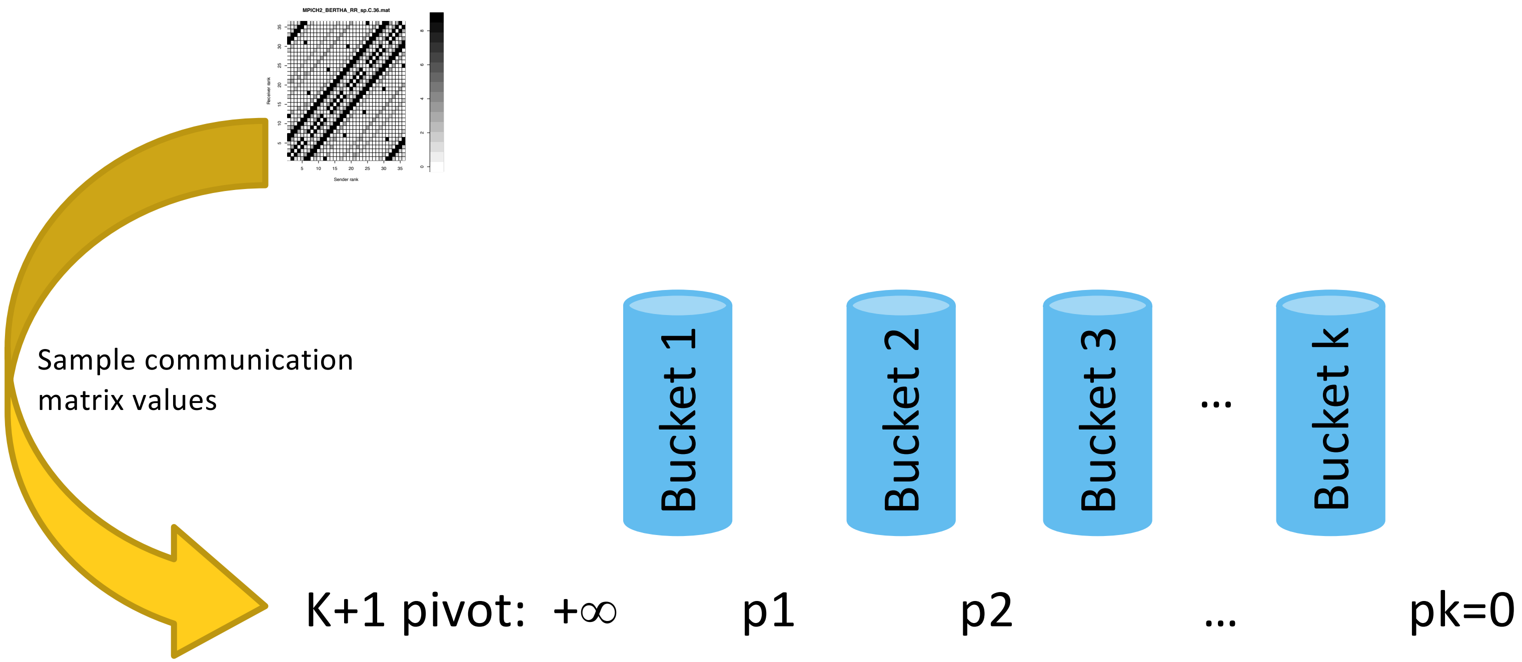
Bucket grouping (group of size 2)



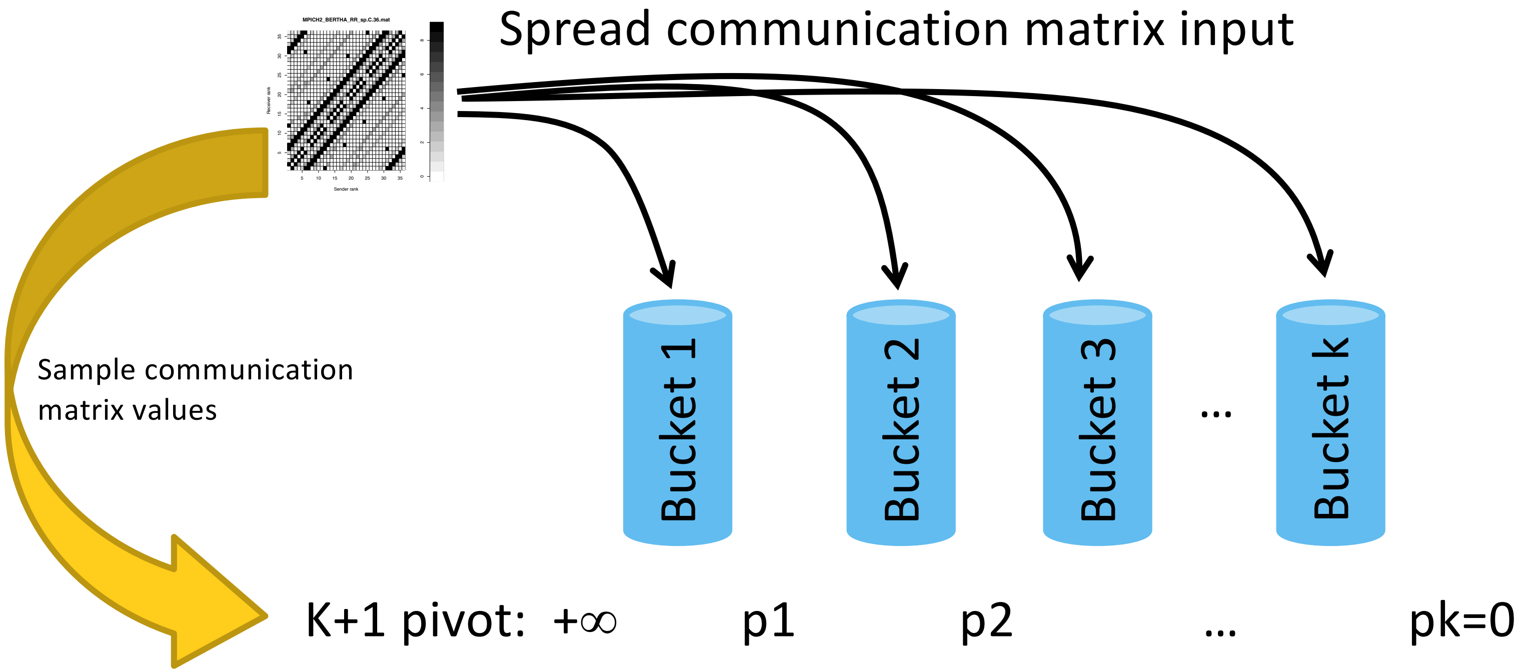
Sample communication matrix values

K+1 pivot: $+\infty$ p1 p2 … pk=0

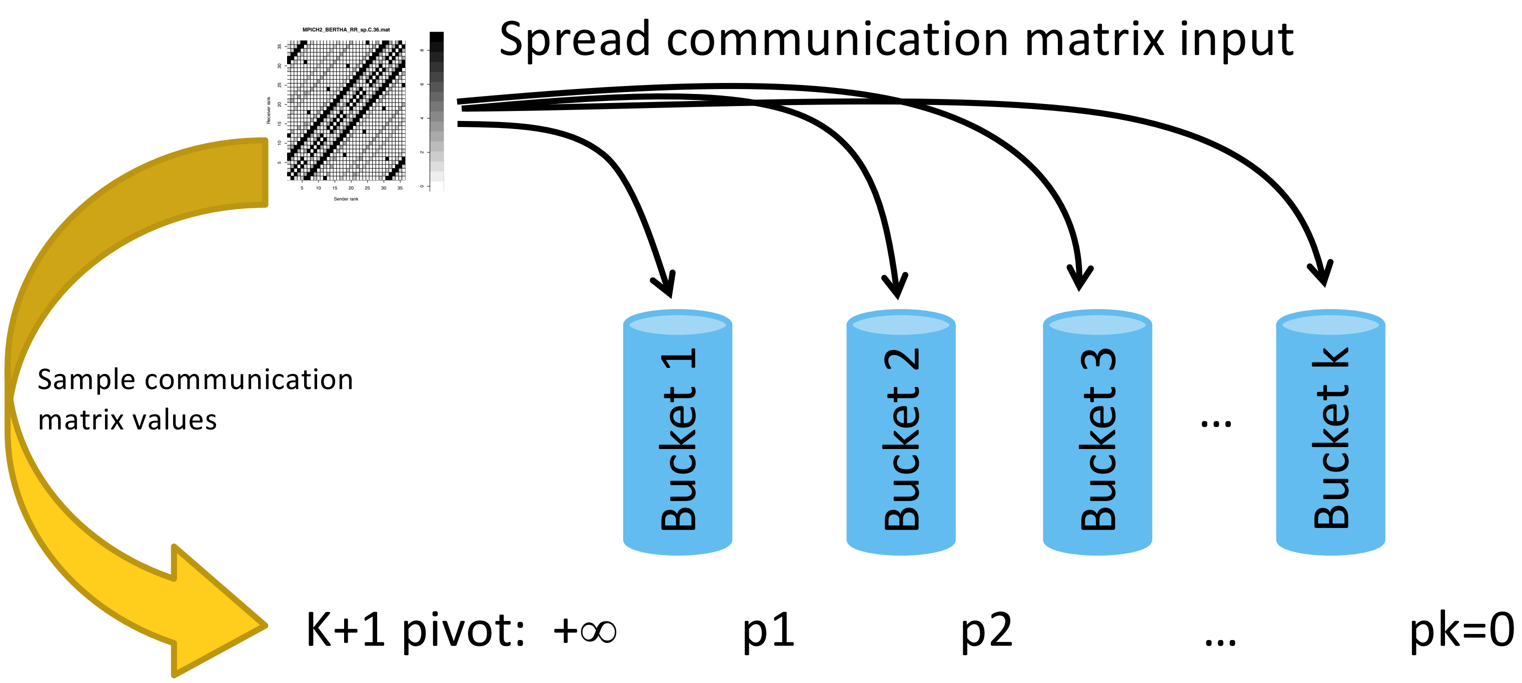
Bucket grouping (group of size 2)



Bucket grouping (group of size 2)



Bucket grouping (group of size 2)



Algorithm

- Sort bucket 1
- Find independent group stating from largest values of bucket 1
- If not enough groups : sort bucket 2
- Etc.

Gain need to sort a few buckets instead of all values

Bucket grouping timings

Number of processes	Partial sorting						Full sorting		
	nb buckets	sorted elements	number of buckets used	init time	sort time	grouping time	init time	sort time	total time
4096	8	10251	1	0.14	0.004	0.16	0.11	7.55	7.68
8182	8	107234	1	0.56	0.04	0.62	0.45	26.08	36.60
16000	8	862567	1	2.41	0.48	2.99	5.32	1144.37	51.94
32768	16	22849	1	45.05	0.17	50.96	57.16	833.57	942.26

TABLE I: Bucket grouping timing. Partial sorting vs. full sorting comparison on an Intel Xeon CPU E5-2680 at 2.50GHz.

05

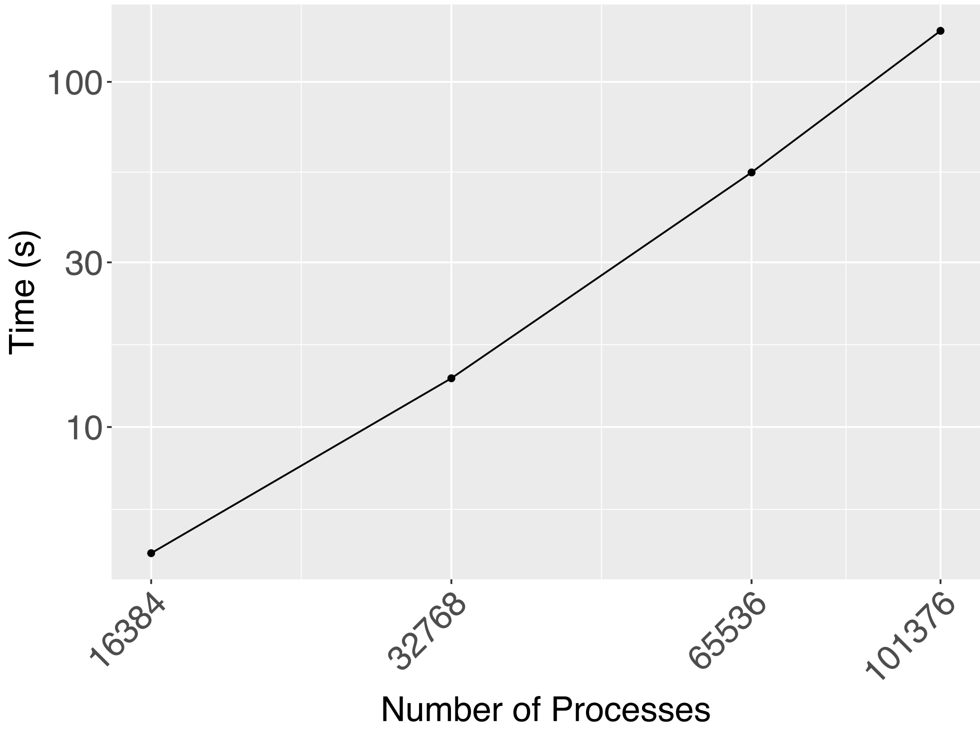
Results

Mapping time

Mapping dense communication matrix on a tree topology.

Xeon 6230, 2*768GB, Optane DCPMM.

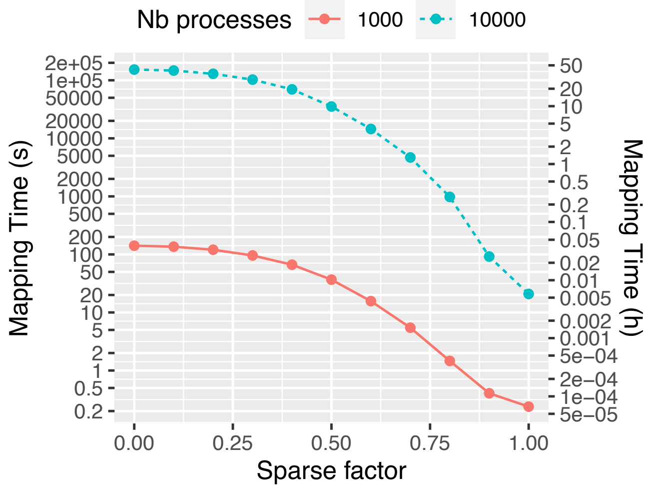
101 736: half the number of cores of Summit.



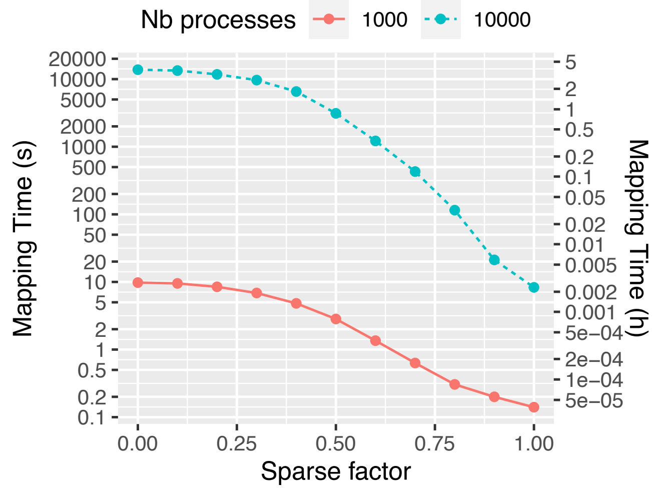
Sparse factor : mapping time

Dense communication matrix : too many information for Scotch?

Sparcify (keep only largest value) the input communication matrix.



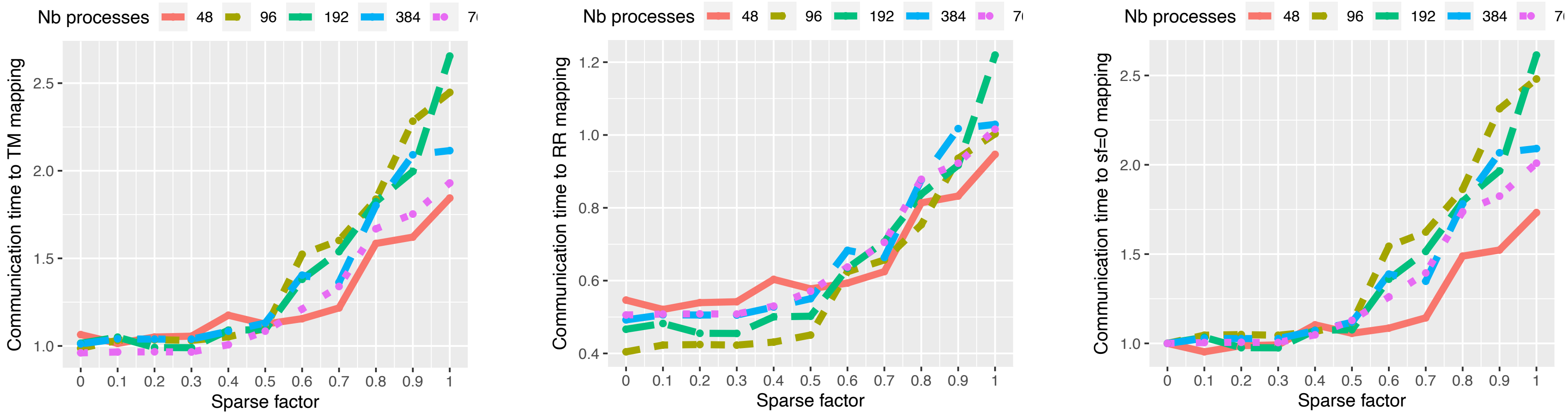
Fat tree Topology (86400 leaves)



2D Mesh Topology (125000 nodes)

Sparse factor : mapping quality

Emulation : MPI_Alltoallv to execute random communication pattern in function of the sparse factor.



TopoMatch + Scotch: Plafrim 2 (Miriel) results

Conclusion : safe to use SF = 0.5 (default TopoMatch Value)

Impact of the noise

Difficult to have the exact value of the communication matrix.

Four types of noise:

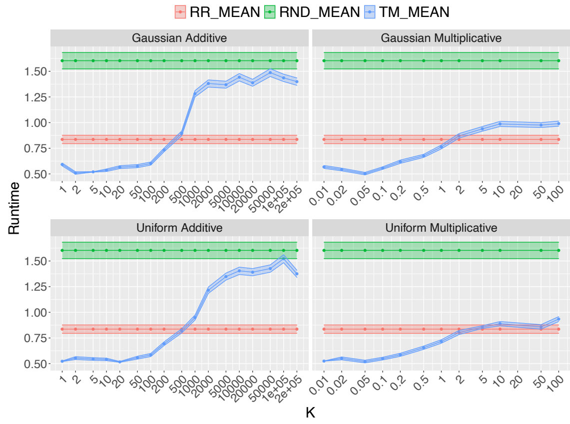
1) $\tilde{M} \leftarrow M + M * \mathcal{N}(0, k)$ (Gaussian Multiplicative),
2) $\tilde{M} \leftarrow M + \mathcal{N}(0, k)$ (Gaussian Additive),
3) $\tilde{M} \leftarrow M + M * \mathcal{U}(-k, k)$ (Uniform Multiplicative),
4) $\tilde{M} \leftarrow M + \mathcal{U}(-k, k)$ (Uniform Additive).

Negative entries are truncated to 0.

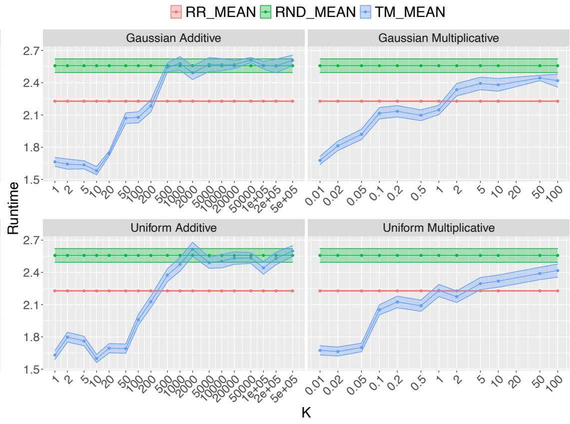
Impact of noise

Noise increase: TM perf degrades -> RR -> Random

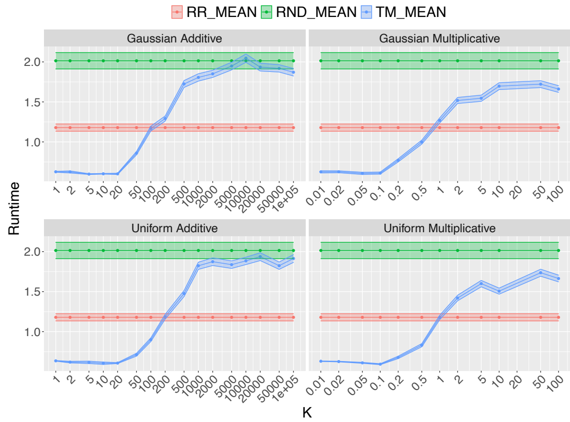
48 node : TM similar to RR for small k.



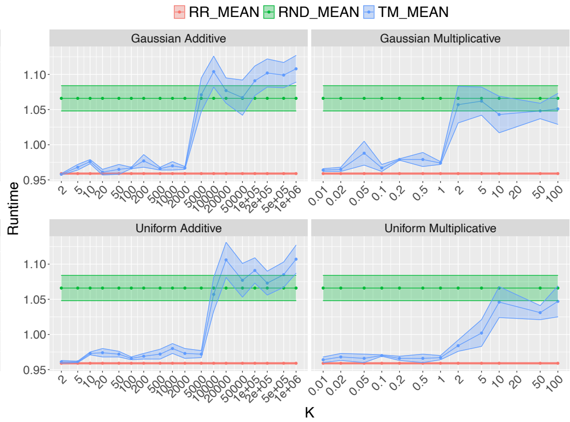
(a) 10 40 120 4 96 23 10



(b) 40 400 10 1 96 21 10



(c) 5 120 120 4 96 21 10



(d) 50 4 40 40 48 21 10 11

MiniGhost: Plafrim 2 (Miriel) results

Conclusion

- **Process placement helps in optimizing communication cost of parallel applications**
- **Useful in many context**
- **Main abstraction: communication matrix**
- **TopoMatch: generic tool for arbitrary topologies**

Thank you!

Follow us on www.inria.fr

inventors for the digital world