Communication and Topology-aware Load Balancing in Charm++ with TreeMatch

IEEE Cluster 2013, Indianapolis, IN

Emmanuel Jeannot Esteban Meneses-Rojas Guillaume Mercier
François Tessier Gengbin Zheng

September 24, 2013

Scalable execution of parallel applications

- Number of cores is increasing
- But **memory per core** is decreasing
- Application will need to communicate even more than now

Our solution

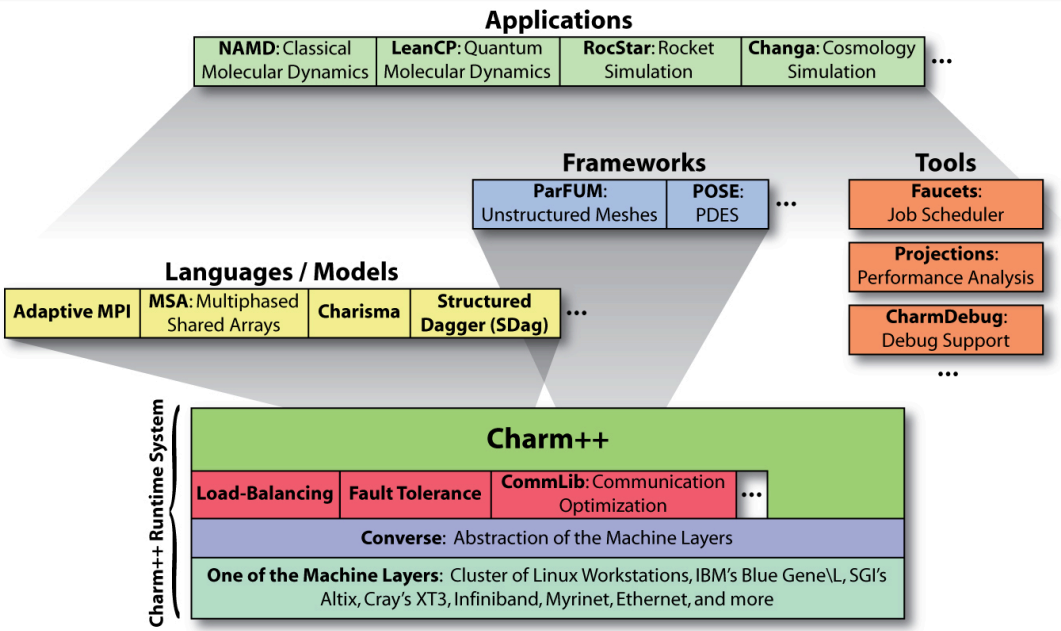
- Process placement should take into account **process affinity**
- Here: **load balancing in Charm++** taking into account:
  - load
  - affinity
  - topology
  - migration cost (transfer time)

Outline

Outline

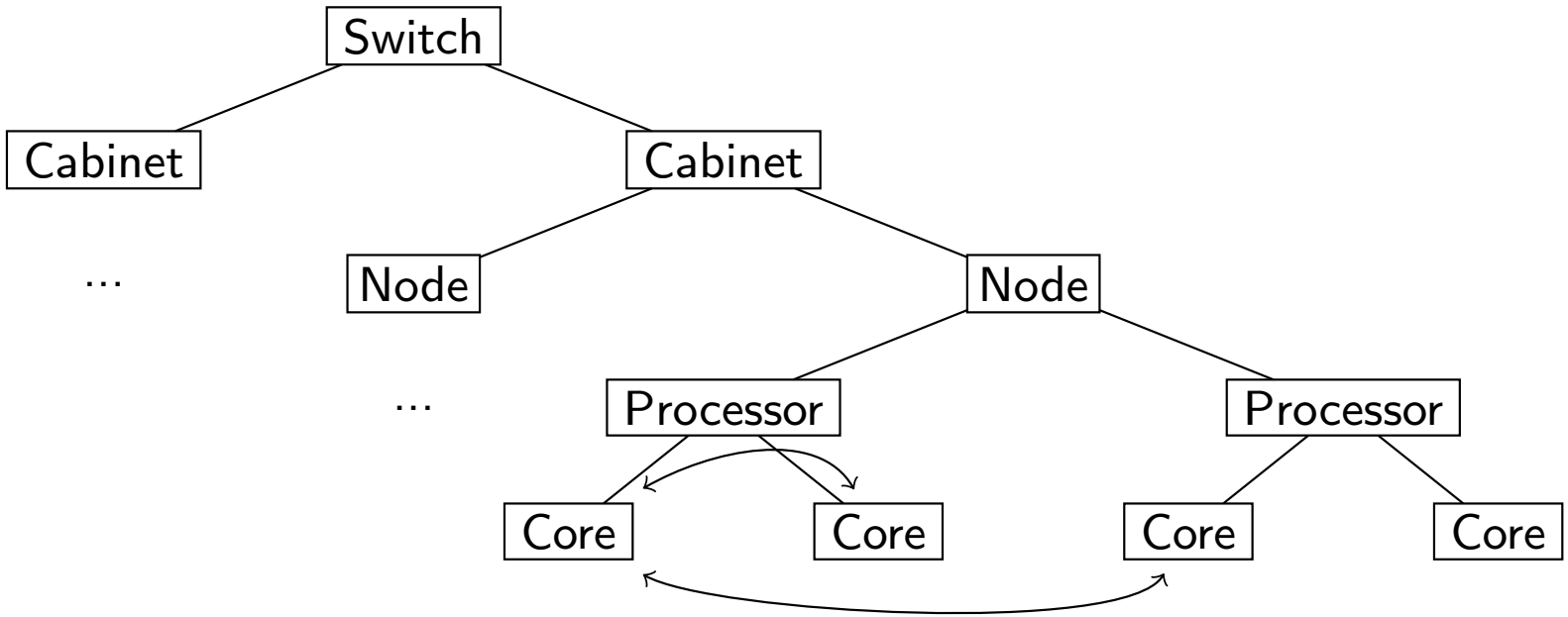
Features

- Parallel object-oriented programming language based on C++
- Programs are decomposed into a number of cooperating message-driven objects called **chares**.
- In general we have more chares than processing units
- Chares are mapped to physical processors by an adaptive runtime system
- Load balancers can be called to **migrate** chares
- Chares placement and load balancing is transparent for the programmer



Why we should consider it

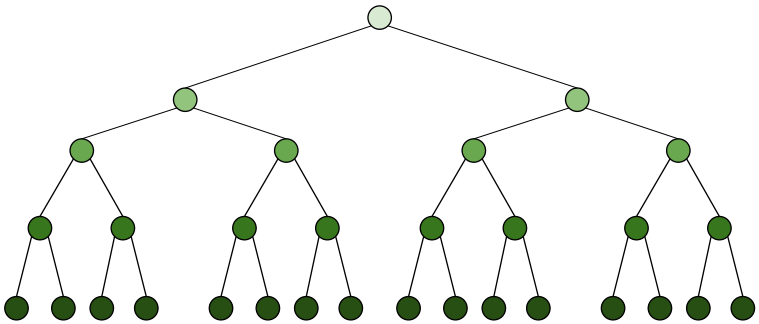
- Many current and future parallel platforms have several levels of hierarchy
- Application Chares/processes do not exchange the same amount of data (affinity)
- The process placement policy may have impact on performance
  - Cache hierarchy, memory bus, high-performance network...
- In this work we deal with **tree topologies** only

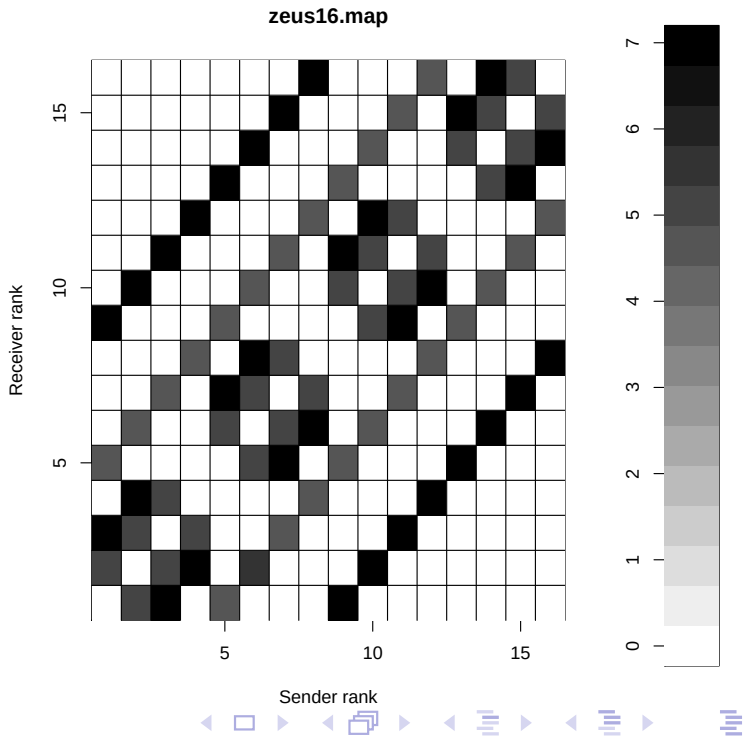


Given

- The parallel machine topology
- The application communication pattern

- Map application processes/chares to physical resources (cores) to reduce the communication costs

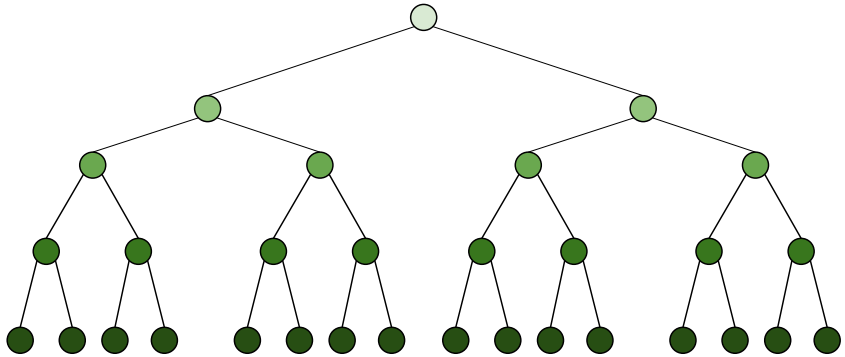


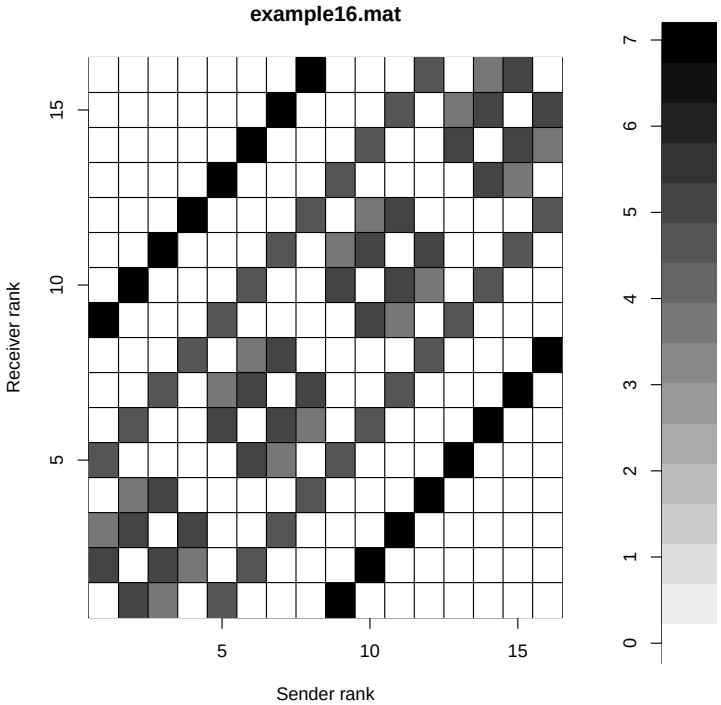


The TreeMatch Algorithm

- Algorithm and environment to compute processes placement based on processes affinities and NUMA topology
- Input :
  - The communication pattern of the application
    - Preliminary execution with a monitored MPI implementation for static placement
    - Dynamic recording on iterative applications with Charm++
  - A model (tree) of the underlying architecture : Hwloc can provide us this.
- Output :
  - A processes permutation σ such that σ_i is the core number on which we have to bind the process i

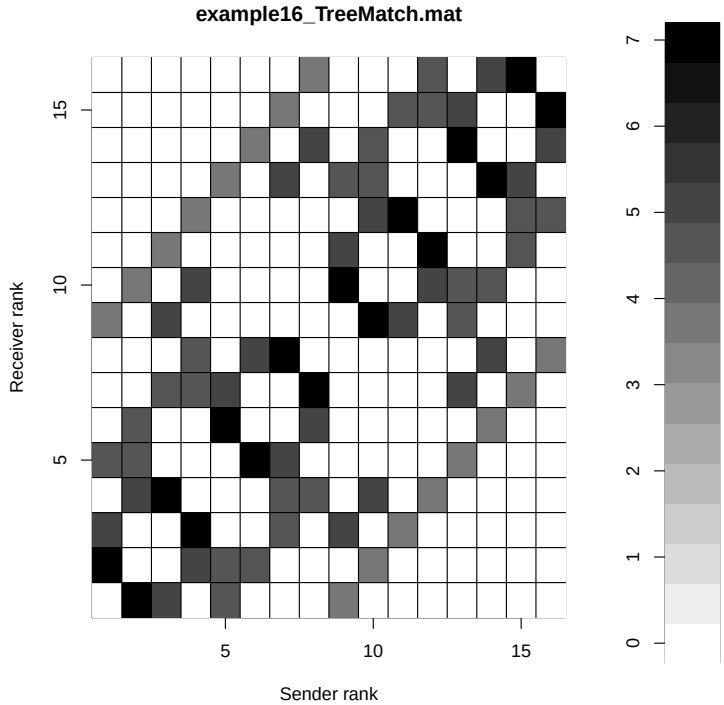
Example





$$\sigma = (0,2,8,10,4,6,12,14,1,3,9,11,5,7,13,15)$$

$$\Longrightarrow$$



TreeMatch Vs. existing solution

Graph partitionners

- Parallel Scotch
- (Par)Metis

Other static algorithms

- [Träff 02]: placement through graph embedding and graph partitioning
- MPIPP [Chen et al. 2006]: placement through local exchange of processes
- LibTopoMap [Hoefler & Snir 11]: placement through network model + graph partitioning (ParMetis)

Other topology-aware load-balacing algorithms

- [L. L. Pilla, et al. 2012] NUCOLB, shared memory machines
- [L. L. Pilla, et al. 2012] HwTopoLB

All these solution requires quantitative information about the network and the communication duration.
TreeMatch: only qualitative information about the topology (the structure) is required.

Load balancing

Principle

- Iterative applications
- load balancer called at regular interval
- Migrate chares in order to optimize several criteria
- Charm++ runtime system provides:
  - chares load
  - chares affinity
  - etc. . .

Constraints

- Dealing with complex modern architectures
- Taking into account communications between elements
- Cost of migrations

Several issues raised

Not so easy...

- Scalability of TreeMatch
- Need to find a relevant compromise between processes affinities and load balancing
  - Compute-bound applications
  - Communication-bound applications
- Impact of chares migrations? What about load balancing time?

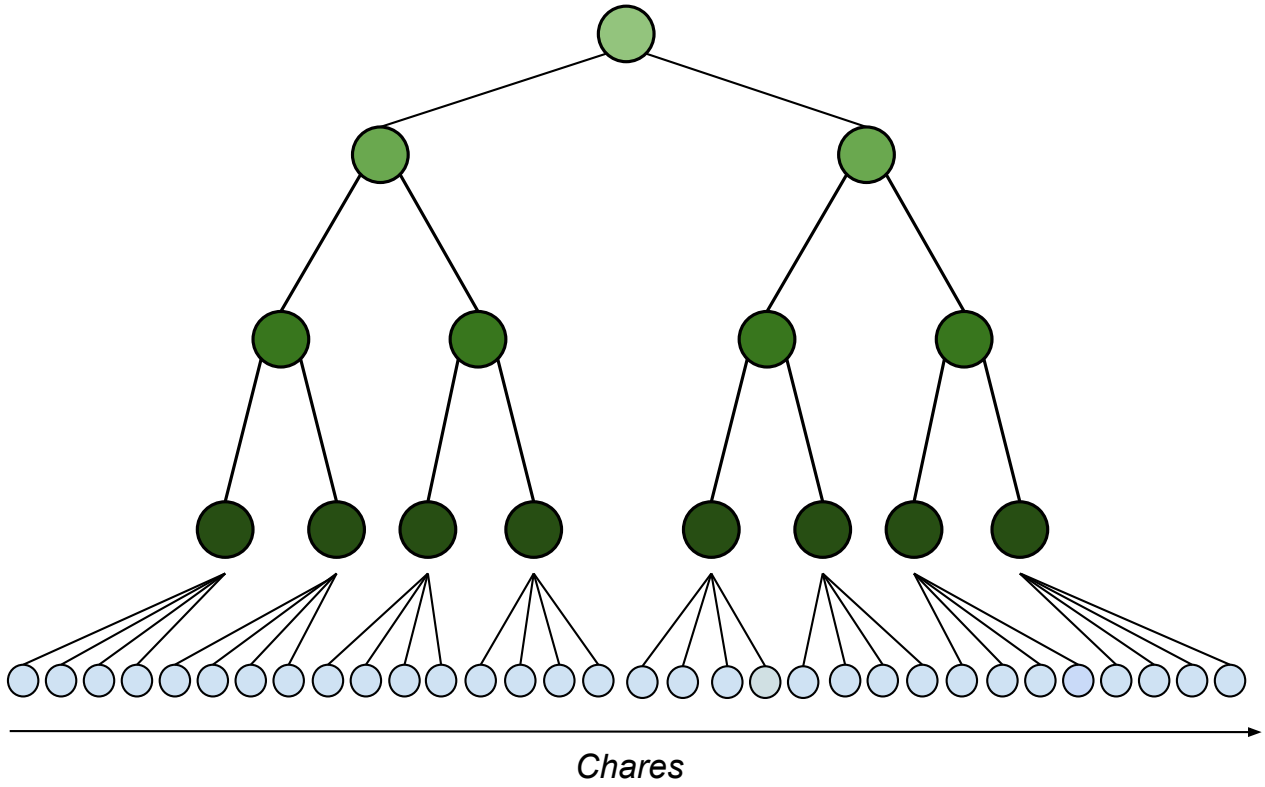
The next slides will present two load balancers relying on TreeMatch

- TMLB_Min_Weight which applies a communication-aware load balancing by favoring the CPU load levelling and minimizing migrations
- TMLB_TreeBased which performs a parallel communication-aware load balancing by giving advantage to the minimization of communication cost.

Outline

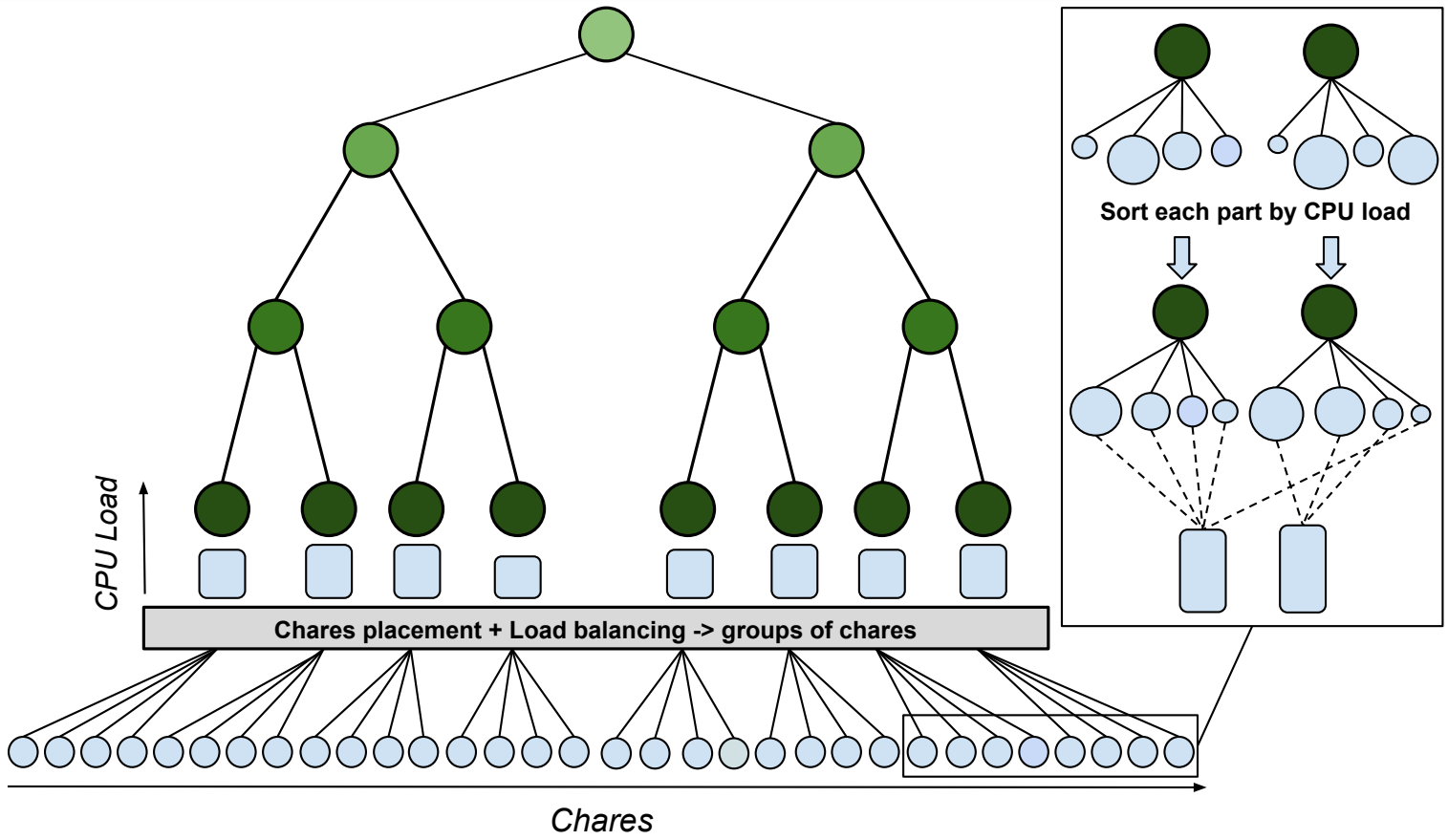
TMLB_Min_Weight

- Applies TreeMatch on all chares (fake topology : #leaves = #chares)
- Binds chares according to their load, leveling on less loaded chares (see example below)
- Hungarian algorithm to minimize the migrations (min. weight matching)



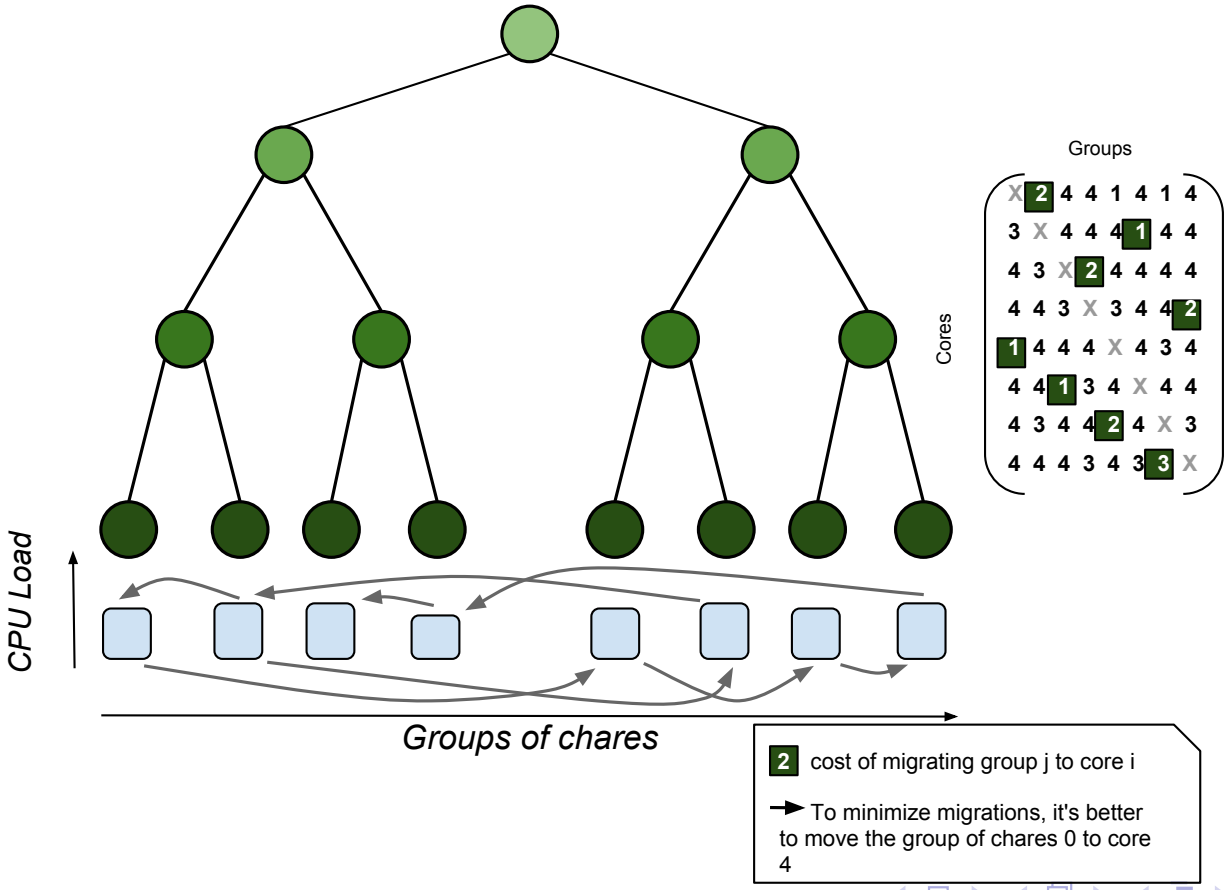
TMLB_Min_Weight

- Applies TreeMatch on all chares (fake topology : #leaves = #chares)
- Binds chares according to their load, leveling on less loaded chares (see example below)
- Hungarian algorithm to minimize the migrations (min. weight matching)



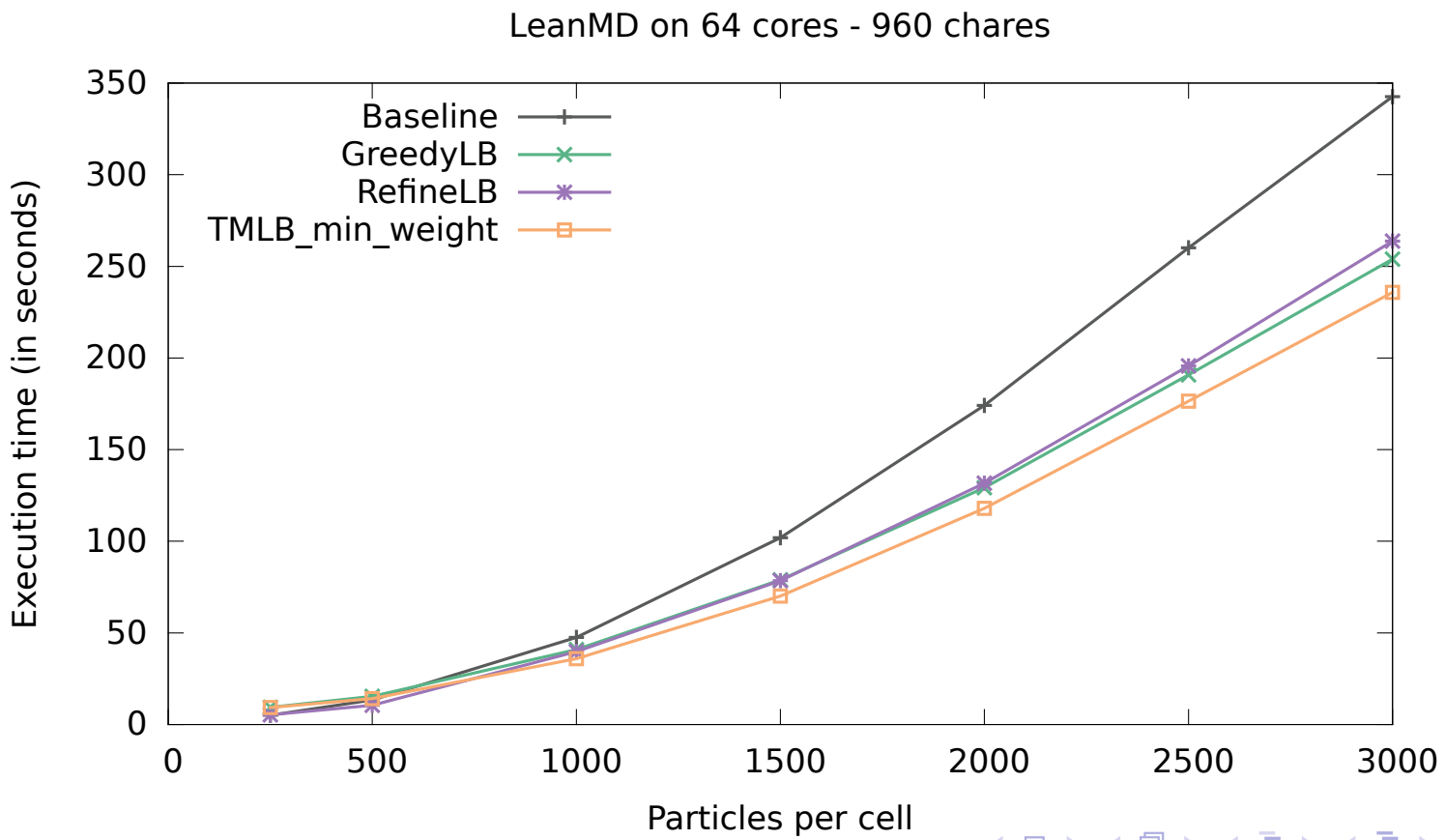
TMLB_Min_Weight

- Applies TreeMatch on all chares (fake topology : #leaves = #chares)
- Binds chares according to their load, leveling on less loaded chares (see example below)
- Hungarian algorithm to minimize the migrations (min. weight matching)



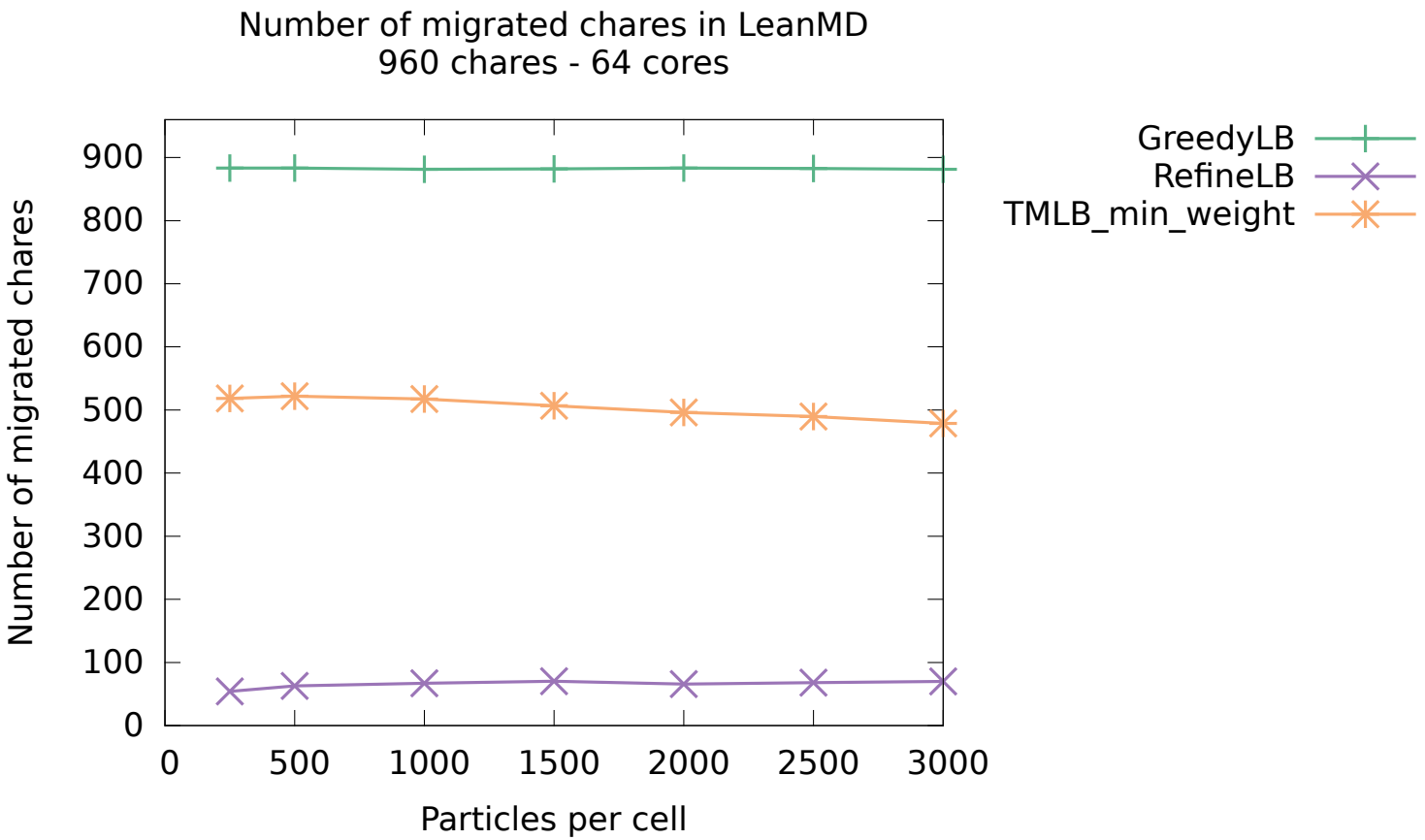
LeanMD

- Molecular Dynamics application
- Massive unbalance, few communications
- Experiments on 8 nodes with 8 cores on each (Intel Xeon 5550)



LeanMD - Migrations

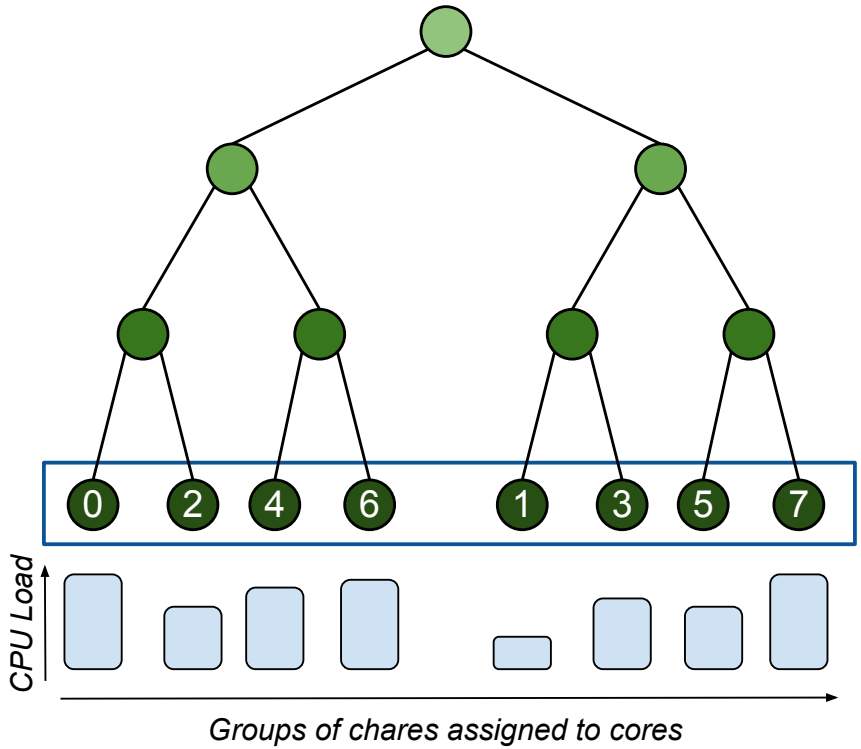
- Comparing to TMLB_Min_Weight without minimizing migrations :
  - Execution time up to 5% better
  - Around 200 migrations less



Outline

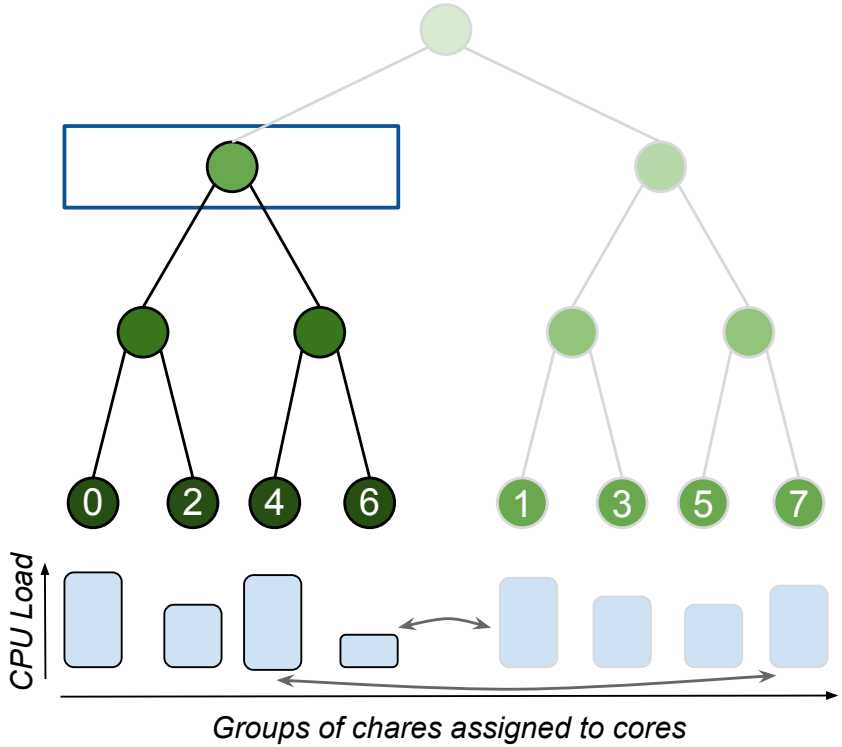
TMLB_TreeBased

- 1^{st} step : Applies TreeMatch while considering groups of chares on cores
- 2^{nd} step : Reorders chares inside each node
  - Defines the subtree
  - Creates a fake topology with as much leaves as the number of chares + something (constraints)
  - Applies TreeMatch on this topology and the chares communication pattern
  - Binds chares according to their load (leveling on less loaded chares)
  - Each node in parallel



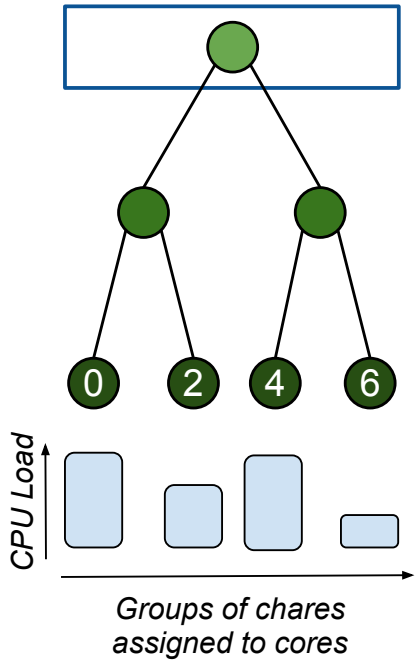
TMLB_TreeBased

- 1^{st} step : Applies TreeMatch while considering groups of chares on cores
- 2^{nd} step : Reorders chares inside each node
  - Defines the subtree
  - Creates a fake topology with as much leaves as the number of chares + something... (constraints)
  - Applies TreeMatch on this topology and the chares communication pattern
  - Binds chares according to their load (leveling on less loaded chares)
  - Each node in parallel



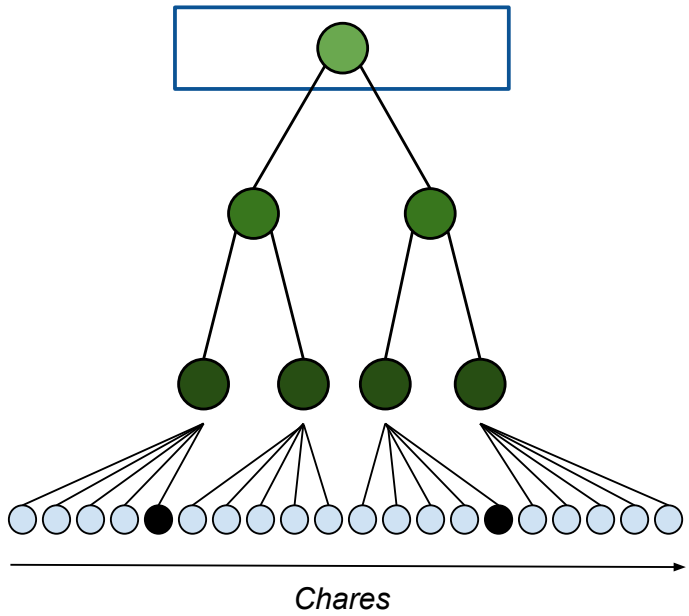
TMLB_TreeBased

- 1^{st} step : Applies TreeMatch while considering groups of chares on cores
- 2^{nd} step : Reorders chares inside each node
  - Defines the subtree
  - Creates a fake topology with as much leaves as the number of chares + something... (constraints)
  - Applies TreeMatch on this topology and the chares communication pattern
  - Binds chares according to their load (leveling on less loaded chares)
  - Each node in parallel



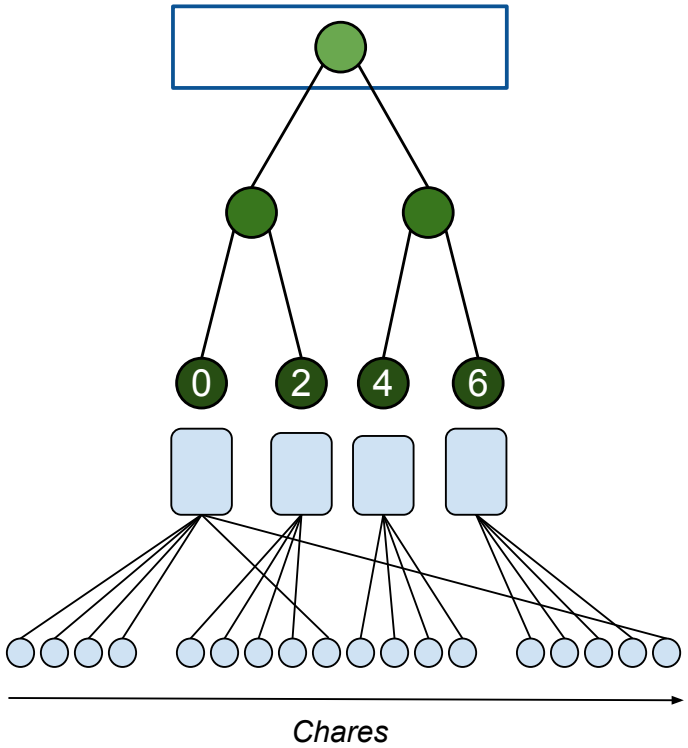
TMLB_TreeBased

- 1^{st} step : Applies TreeMatch while considering groups of chares on cores
- 2^{nd} step : Reorders chares inside each node
  - Defines the subtree
  - Creates a fake topology with as much leaves as the number of chares + something... (constraints)
  - Applies TreeMatch on this topology and the chares communication pattern
  - Binds chares according to their load (leveling on less loaded chares)
  - Each node in parallel



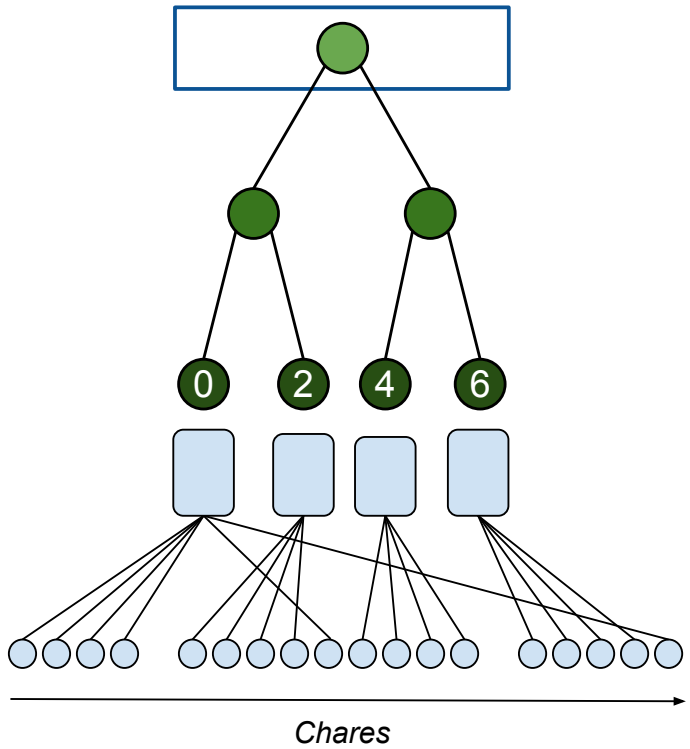
TMLB_TreeBased

- 1^{st} step : Applies TreeMatch while considering groups of chares on cores
- 2^{nd} step : Reorders chares inside each node
  - Defines the subtree
  - Creates a fake topology with as much leaves as the number of chares + something... (constraints)
  - Applies TreeMatch on this topology and the chares communication pattern
  - Binds chares according to their load (leveling on less loaded chares)
  - Each node in parallel



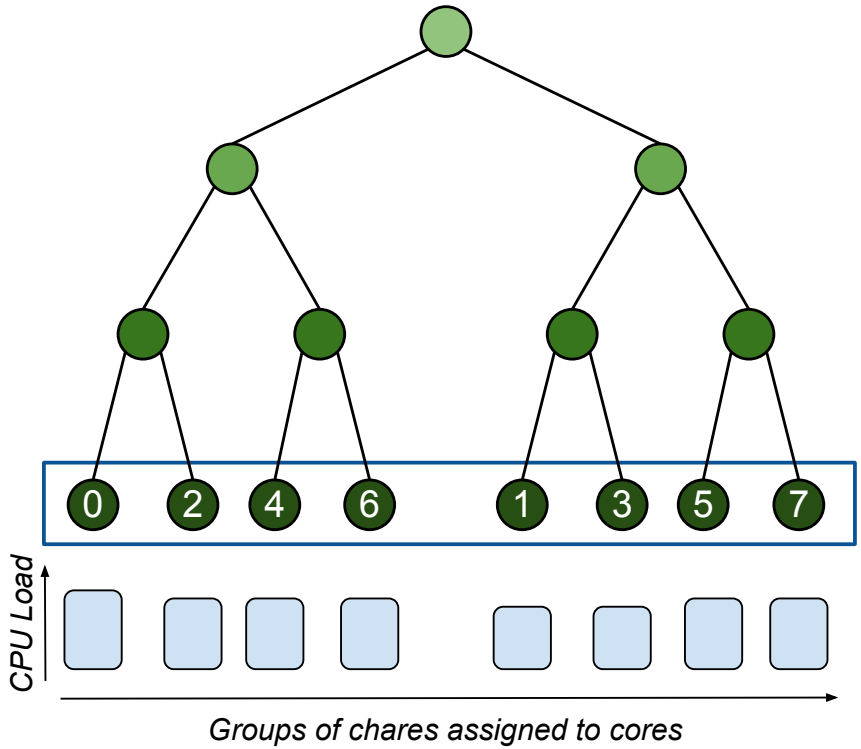
TMLB_TreeBased

- 1^{st} step : Applies TreeMatch while considering groups of chares on cores
- 2^{nd} step : Reorders chares inside each node
  - Defines the subtree
  - Creates a fake topology with as much leaves as the number of chares + something... (constraints)
  - Applies TreeMatch on this topology and the chares communication pattern
  - Binds chares according to their load (leveling on less loaded chares)
  - Each node in parallel



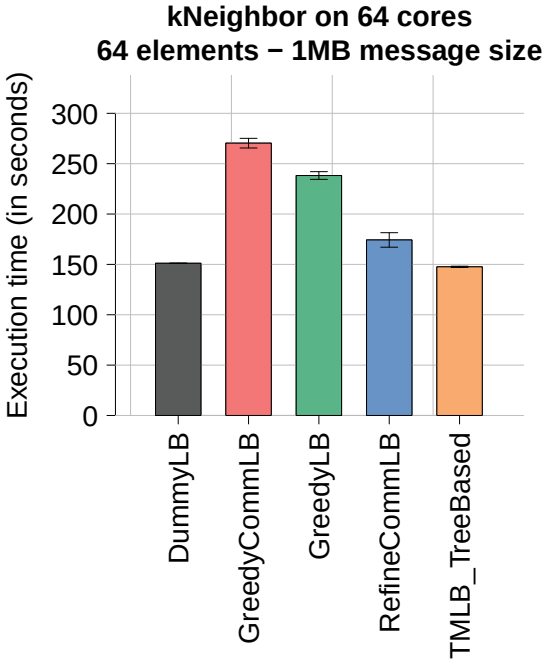
TMLB_TreeBased

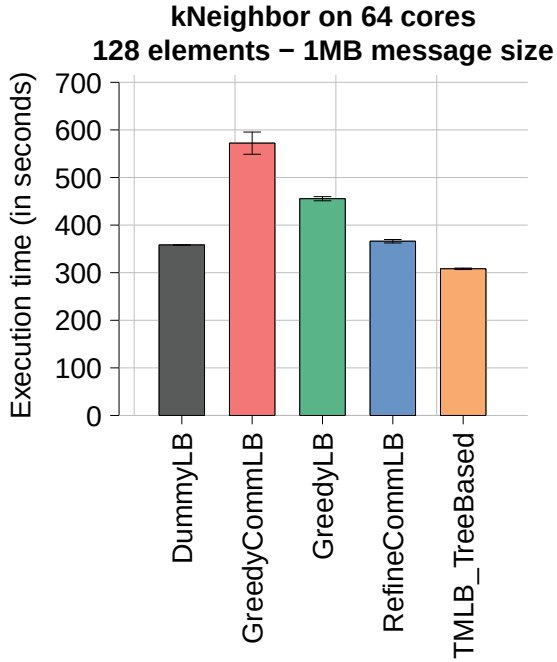
- 1^{st} step : Applies TreeMatch while considering groups of chares on cores
- 2^{nd} step : Reorders chares inside each node
  - Defines the subtree
  - Creates a fake topology with as much leaves as the number of chares + something... (constraints)
  - Applies TreeMatch on this topology and the chares communication pattern
  - Binds chares according to their load (leveling on less loaded chares)
  - Each node in parallel

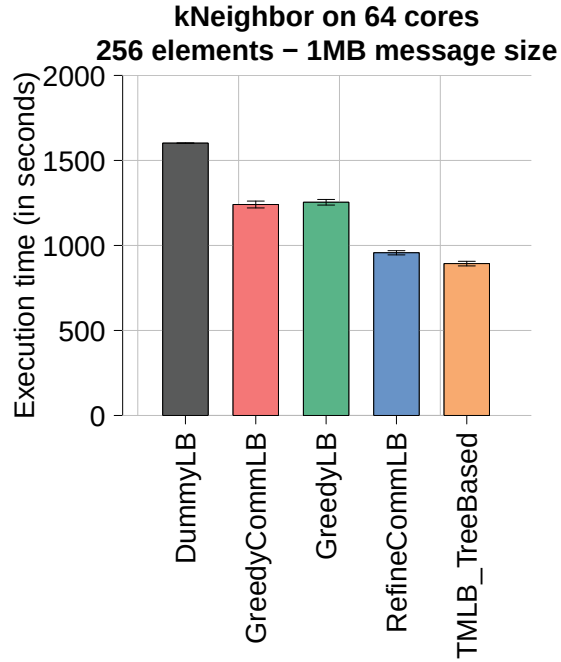


kNeighbor

- Benchmarks application designed to simulate intensive communication between processes
- Experiments on 8 nodes with 8 cores on each (Intel Xeon 5550)
- Particularly compared to RefineCommLB
  - Takes into account load and communication
  - Minimizes migrations



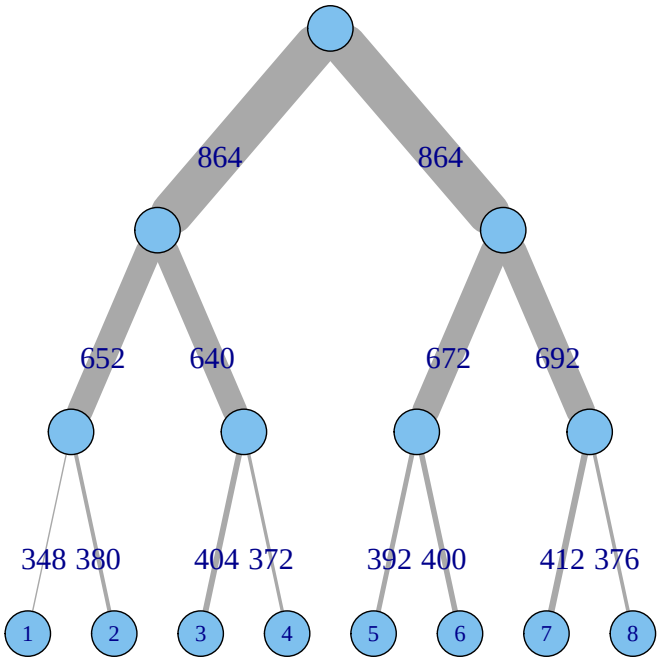




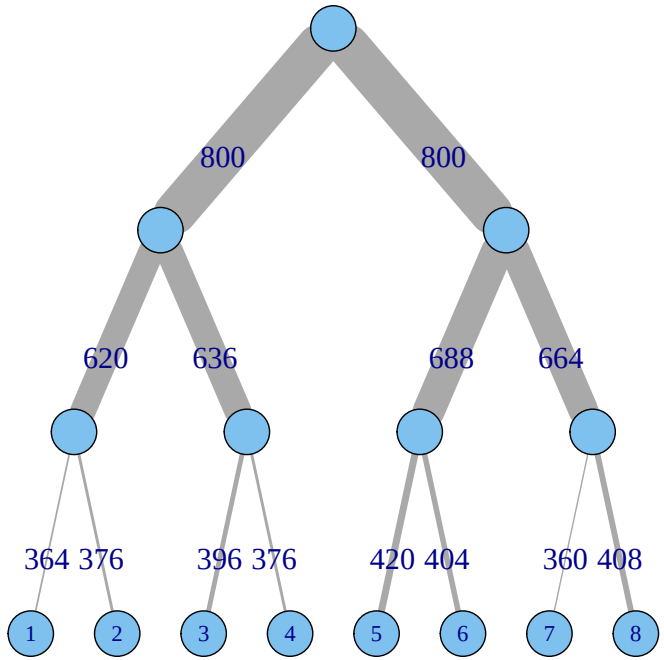
Impact on communication

- Communications evolution between ten iterations

Communication between 10 iterations without any load balancing strategy (in thousands of messages sent)

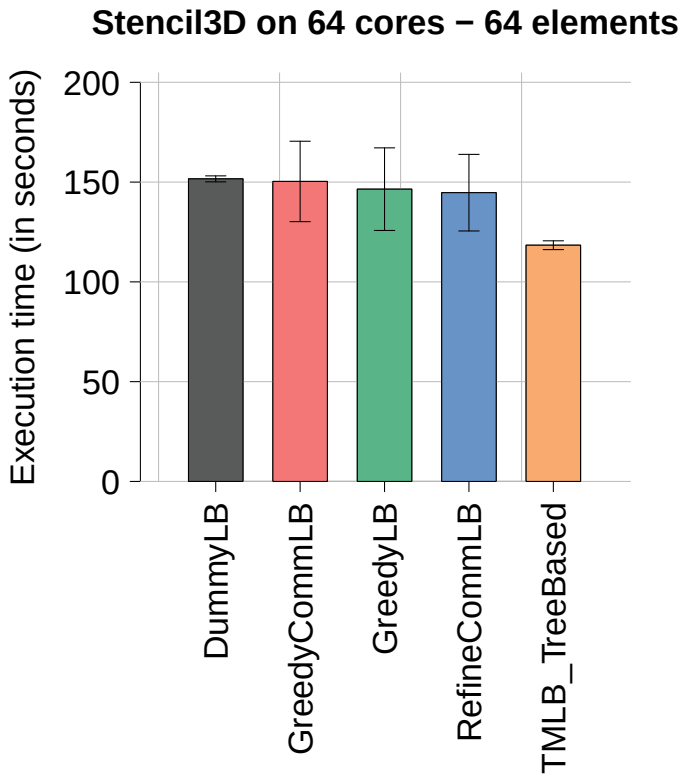


Communication between 10 iterations after the first call of TreeMatchLB (in thousands of messages sent)



Stencil3D

- 3 dimensional stencil with regular communication with fixed neighbors
- One chare per core : balance only considering communications
- Only one load balancing step after 10 iterations
- Experiments on 8 nodes with 8 cores on each (Intel Xeon 5550)



What about the load balancing time?

- Linear trajectory while the number of chares is doubled
- TMLB_TreeBased is clearly slower than the other strategies
- But the parallel version is almost implemented...

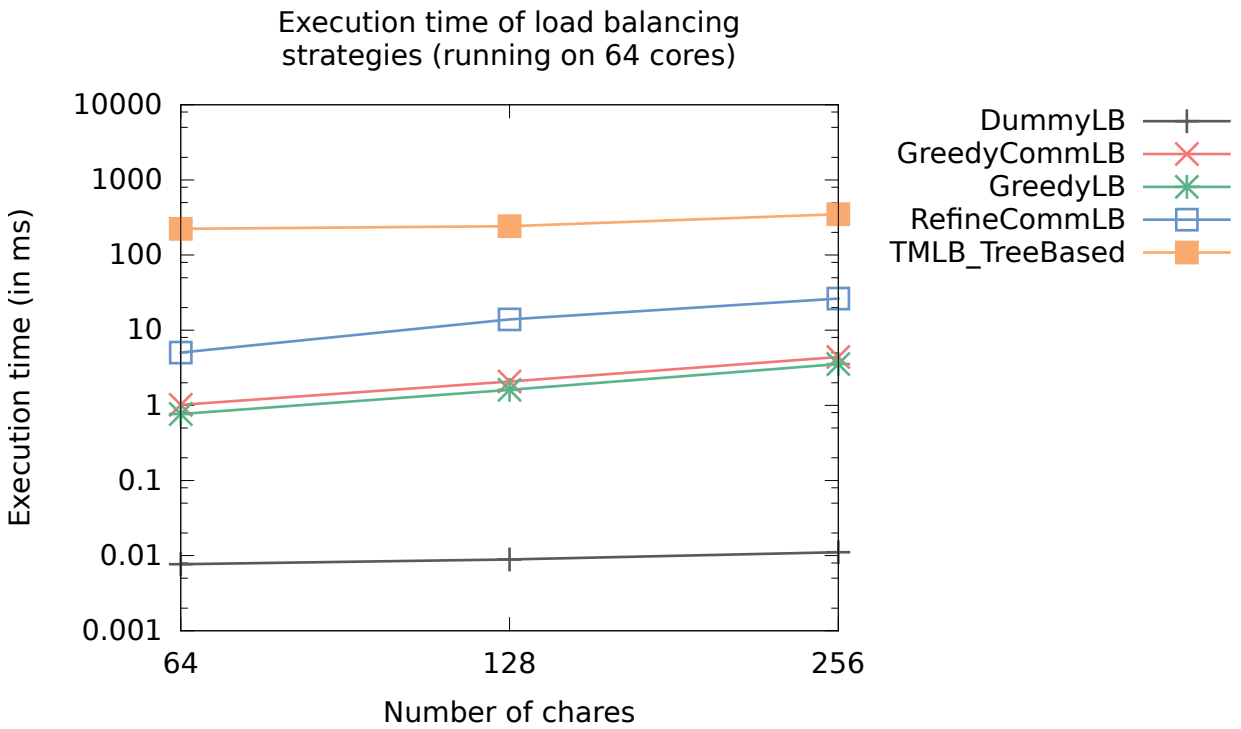


Figure : Load balancing time of the different strategies vs. number of chares for the KNeighbor application.

Outline

Conclusion

- Topology is not flat!
- Processes affinities are not homogeneous
- Take into account these information to map chares give us improvement
- Need to distinguish between compute-bound and communication-bound application
- Several criteria taken into account: affinity, topology, load, migration cost, etc. . .

Future work

- Find a better way to gather the topology (Hwloc?)
- Distribute and parallelize TMLB_TreeBased on the different nodes (work in progess with the PPL)
- Make TMLB_TreeBased more scalable for large scale clusters: allow to chose the level in the hierarchy where the algorithm will be distributed
- Hybrid architecture? Intel MIC?
- Continue collaborations between Inria and PPL

Thanks for your attention !
Any questions?