Communication-aware load balancing with TreeMatch in Charm++

The 9th workshop of the Joint Laboratory for Petascale Computing, Lyon

Emmanuel Jeannot Guillaume Mercier François Tessier
In collaboration with the Charm++ Team from the PPL :
Esteban Meneses-Rojas, Gengbin Zheng, Sanjay Kale

June 14, 2013

Introduction

State of the Art

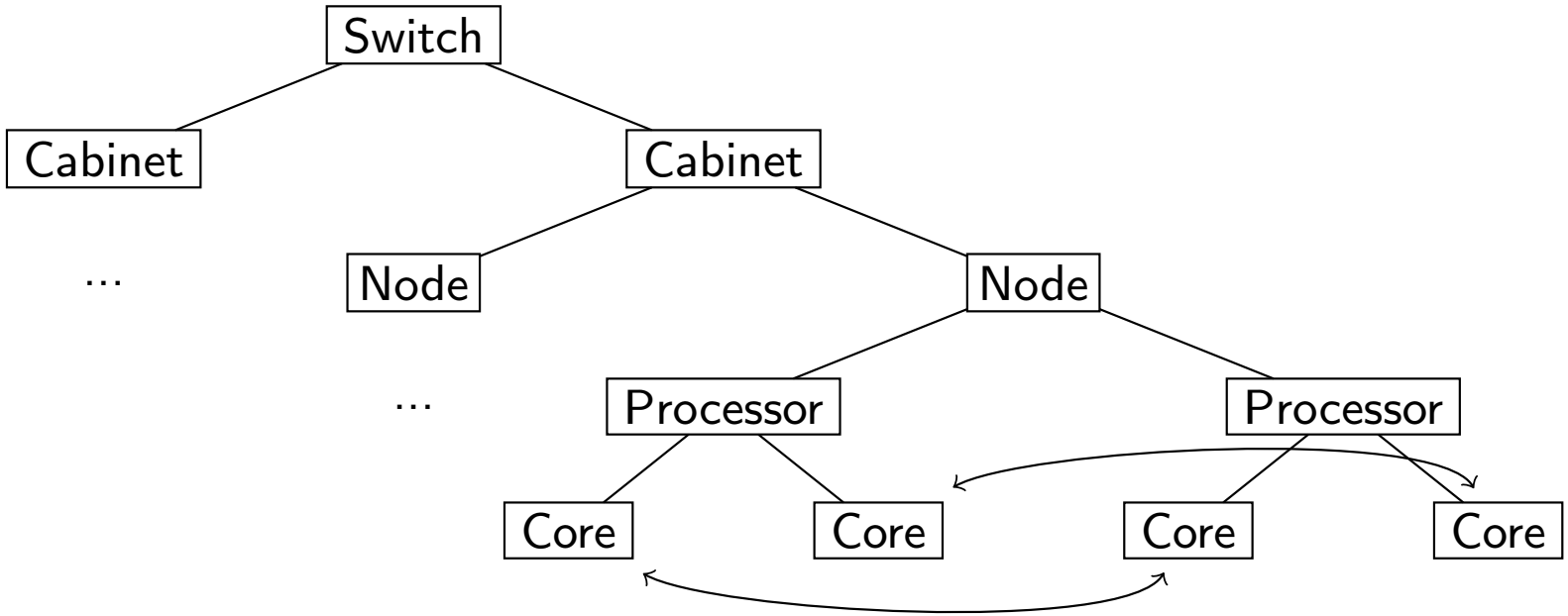
- Multi-node and multi-core architectures : Message passing paradigm
- Fine-grained implementation like Charm++ (independent computing elements called "chares")
- Dynamic load balancing according to a flat topology

Problems

- Topology is not flat!
- Add the notion of processes affinity
- Take into account the communication between processes
- Consider the underlying topology

Why we should consider it

- Many current and future parallel platforms have several levels of hierarchy
- Application processes don't exchange the same amount of data (affinity)
- The process placement policy may have impact on performance
  - Cache hierarchy, memory bus, high-performance network...



Problems

Given

- The parallel machine topology
- The application communication pattern

- Map application processes 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 recovery on iterative applications with Charm++
  - A representation 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

What about Charm++?

Not so easy...

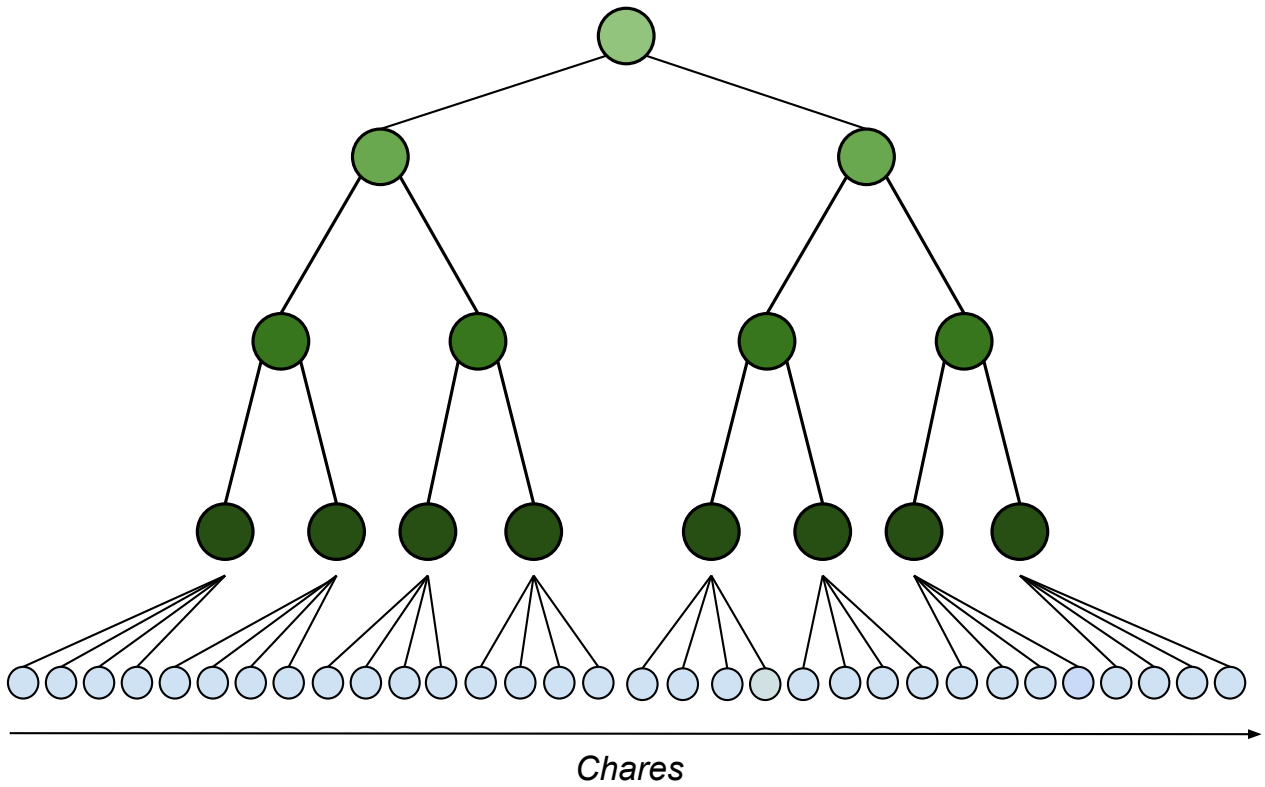
- Several issues raised!
- Scalability of TreeMatch
- Need to find a relevant compromise between processes affinities and load balancing
  - Compute-bound applications
  - Communication-bound applications
- Impact of chares migrations? What about load balancing time?

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.

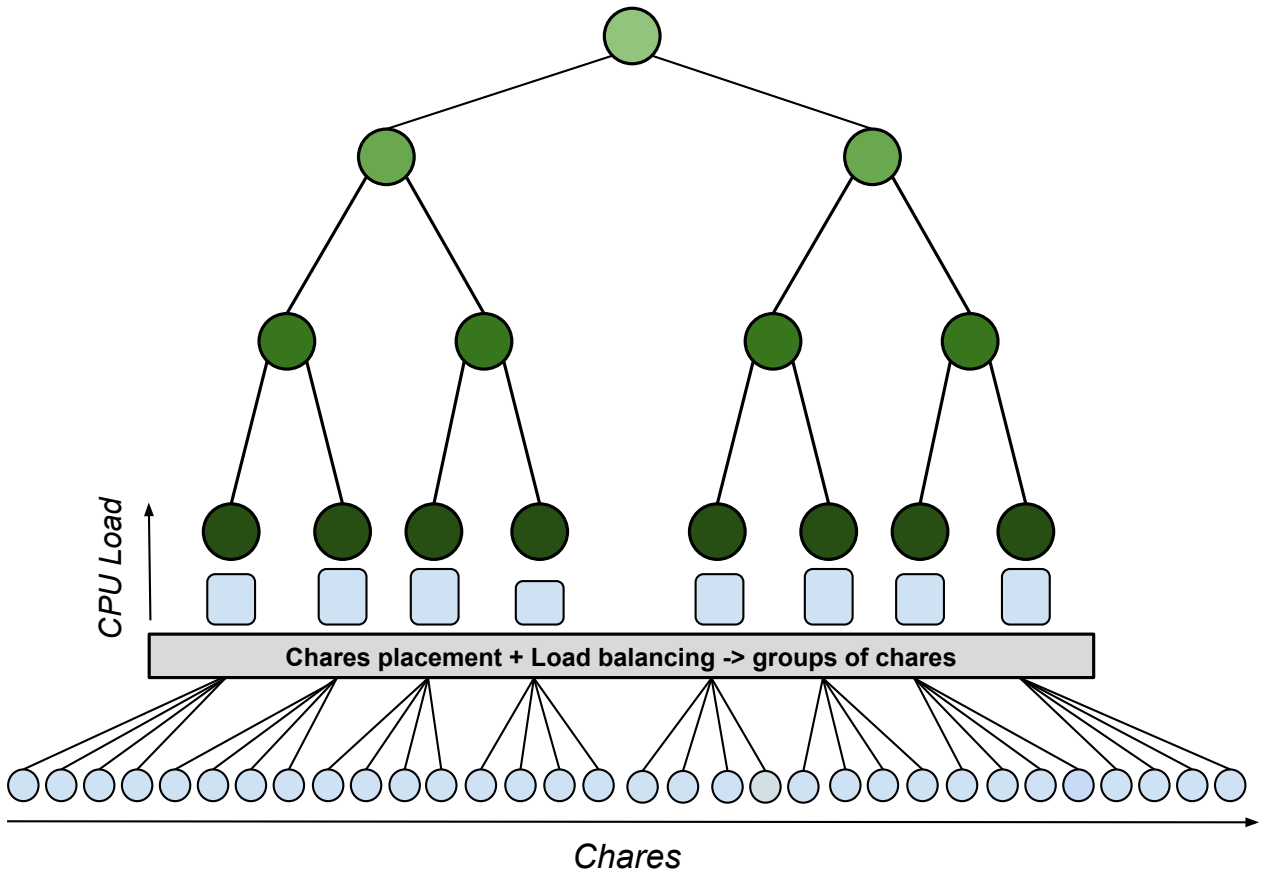
TMLB_Min_Weight

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



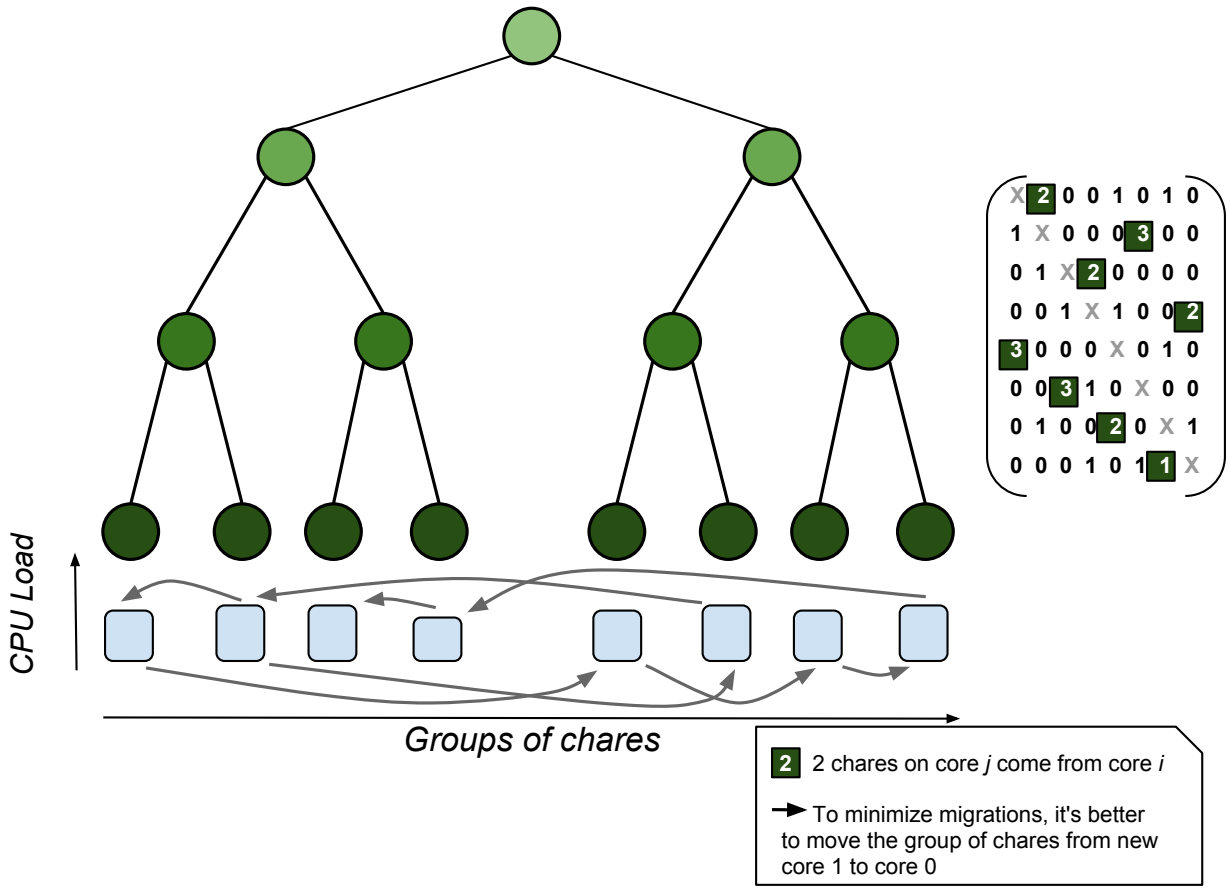
TMLB_Min_Weight

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



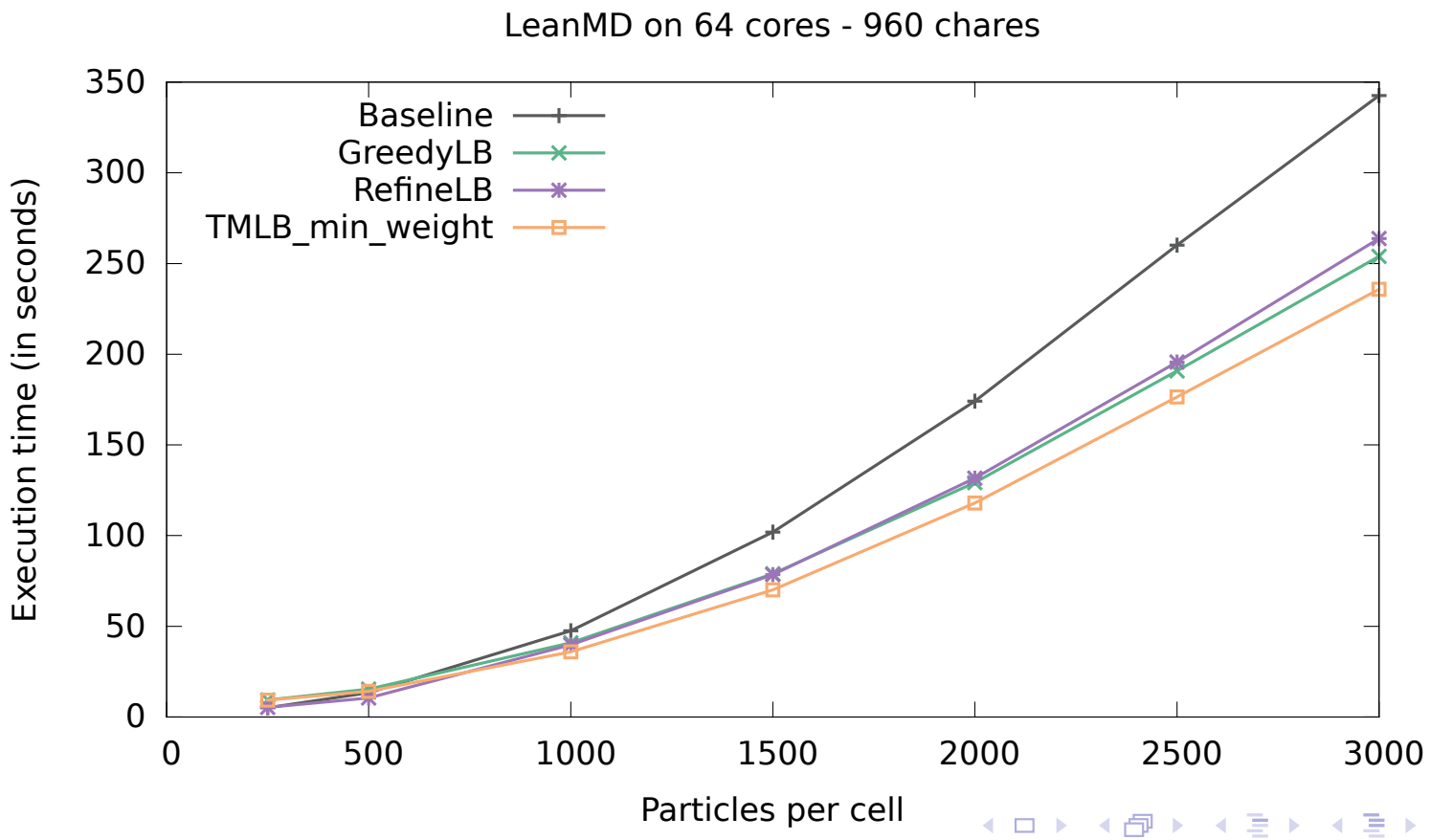
TMLB_Min_Weight

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



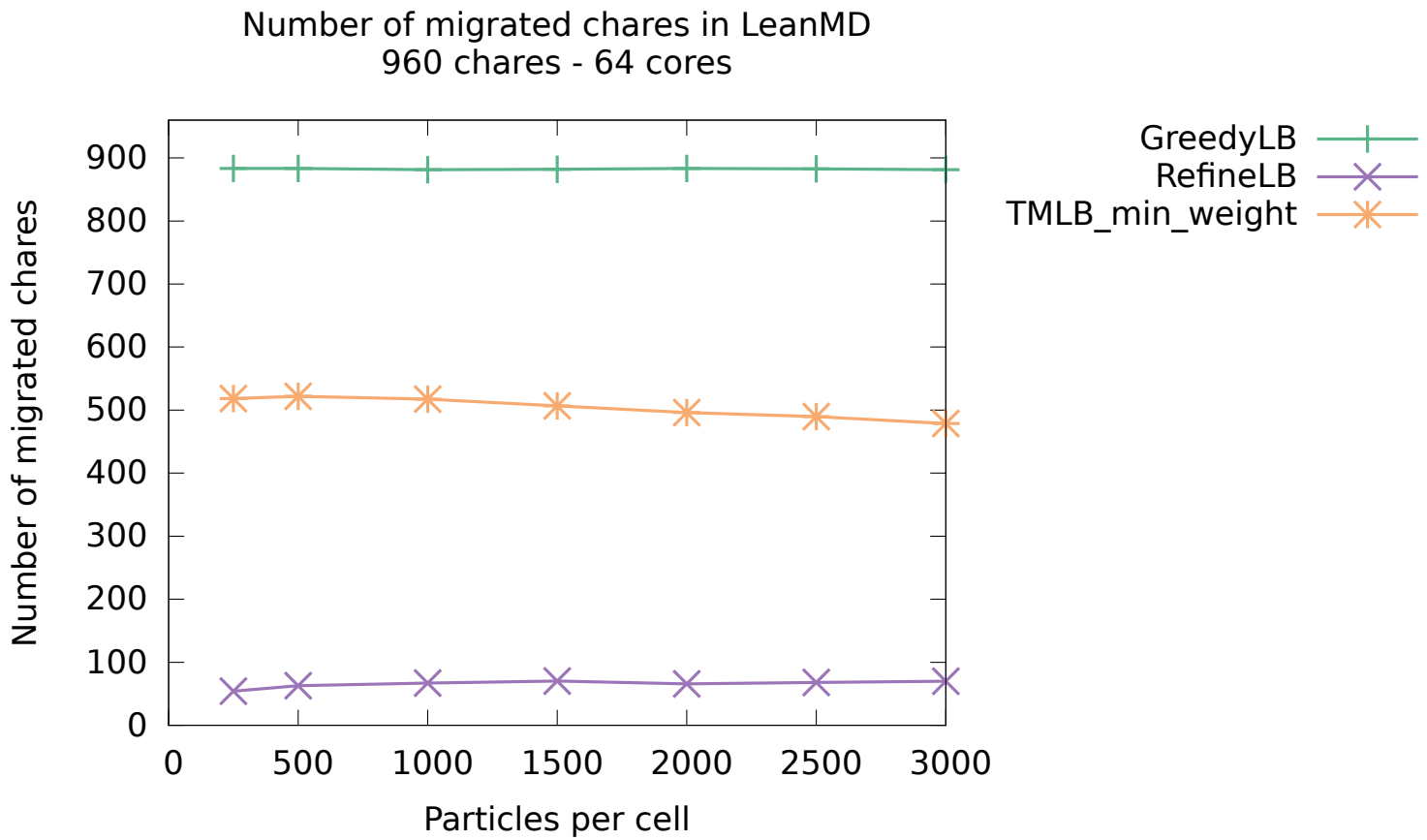
LeanMD

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



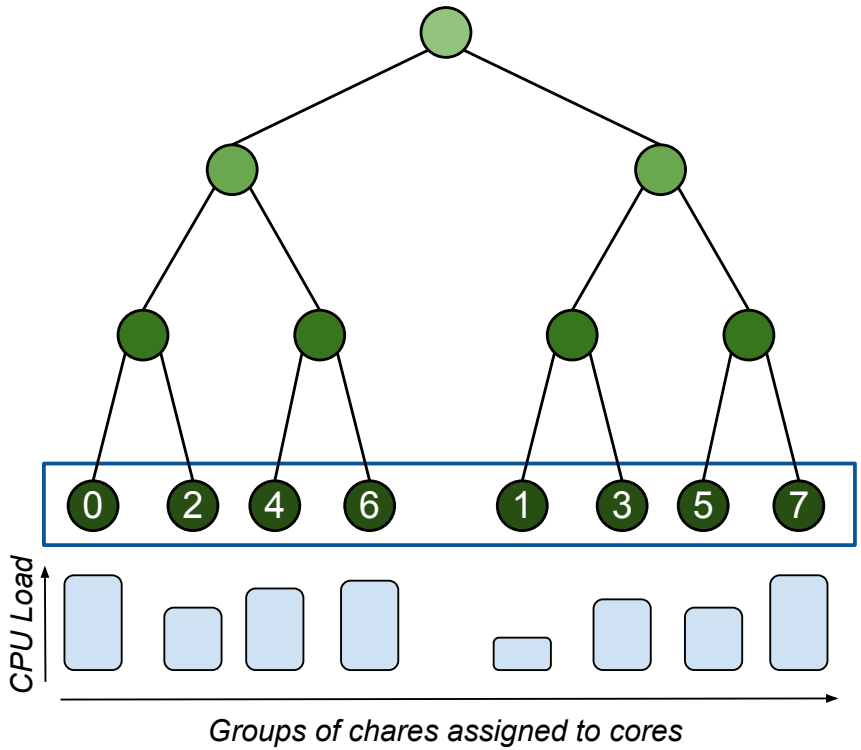
LeanMD - Migrations

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



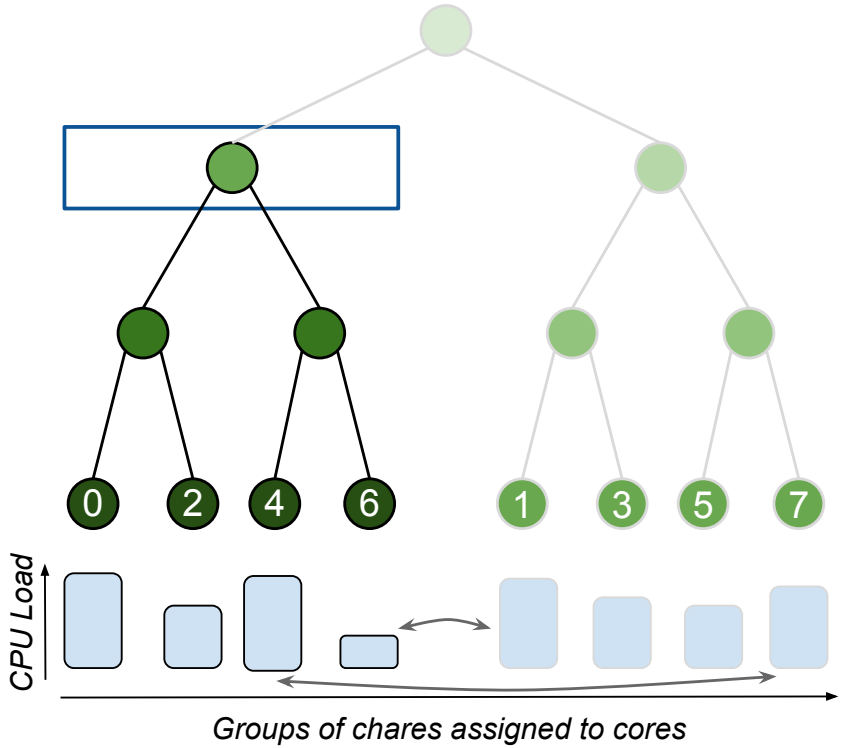
TMLB_TreeBased

- 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 (levelling on less loaded chares)
  - Each node in parallel



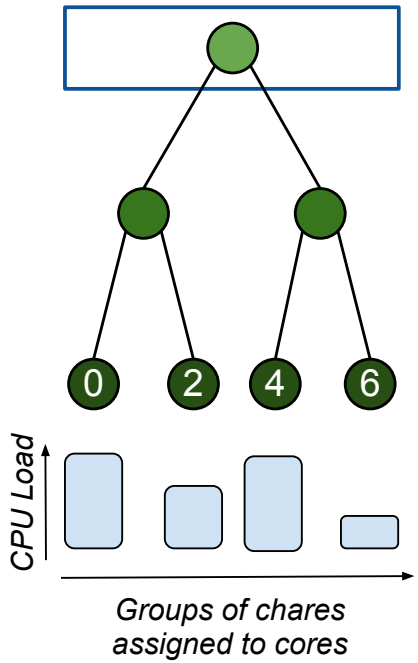
TMLB_TreeBased

- 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



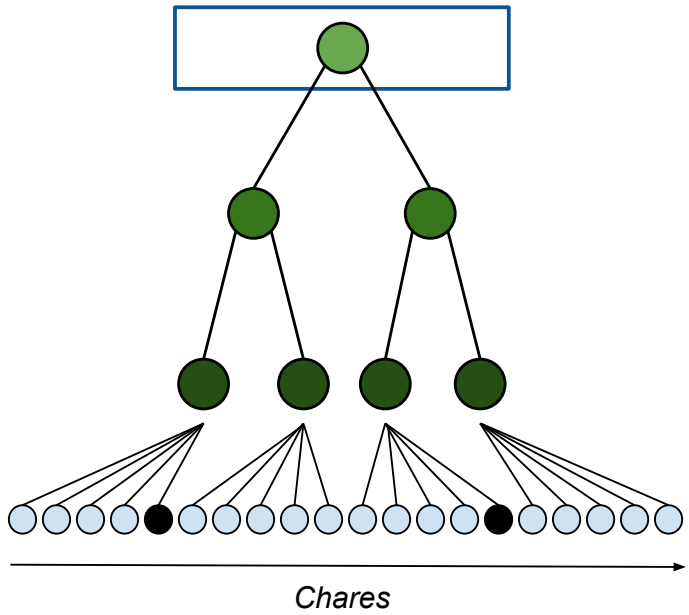
TMLB_TreeBased

- 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



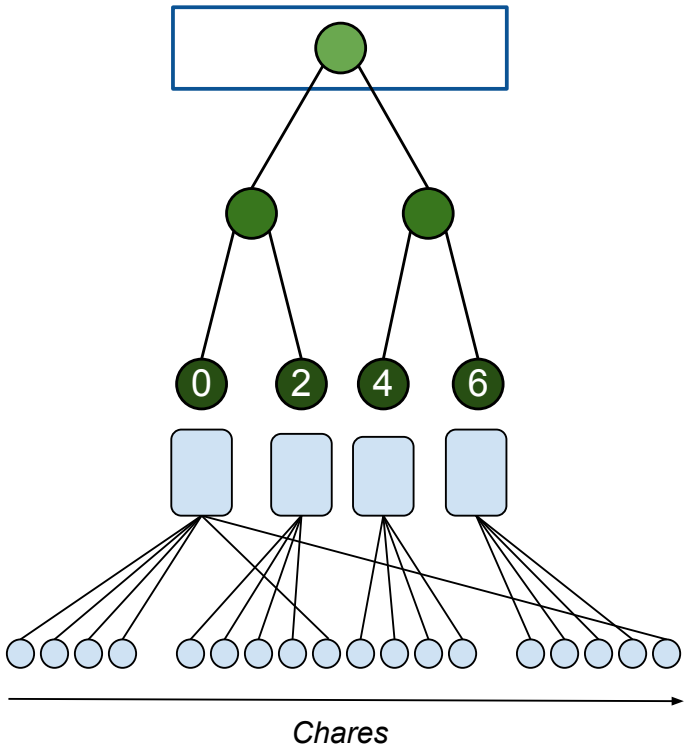
TMLB_TreeBased

- 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



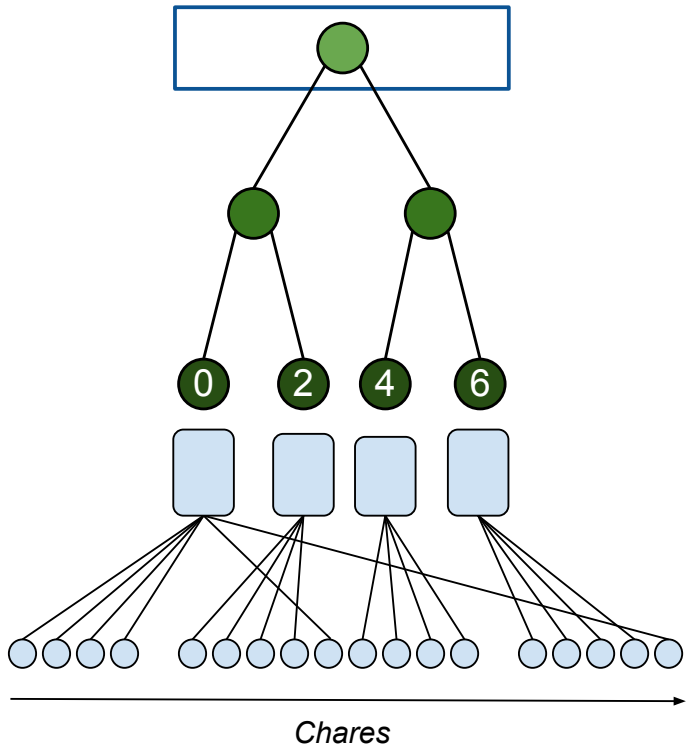
TMLB_TreeBased

- 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



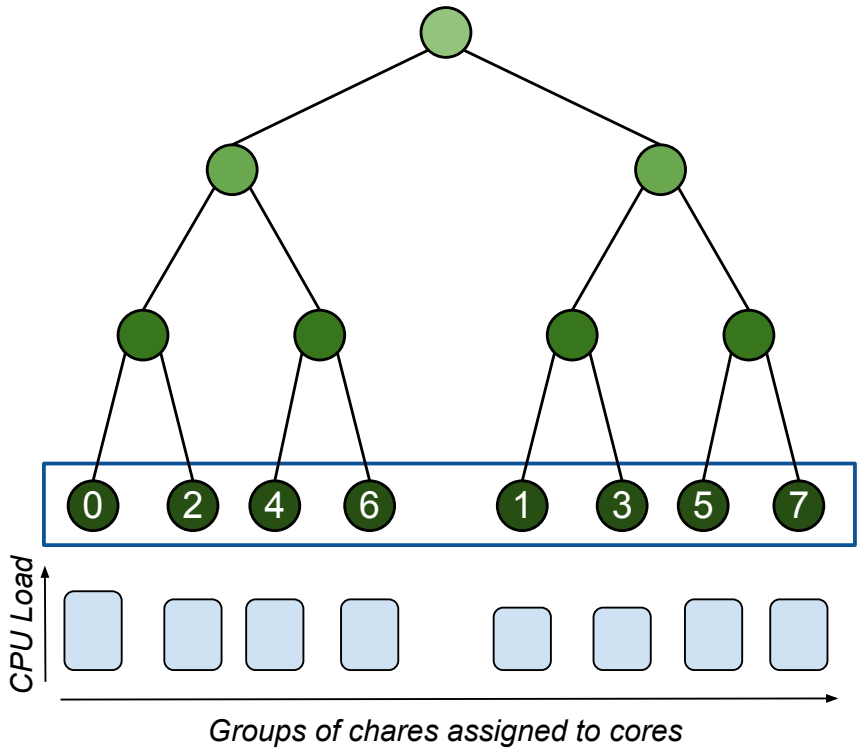
TMLB_TreeBased

- 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



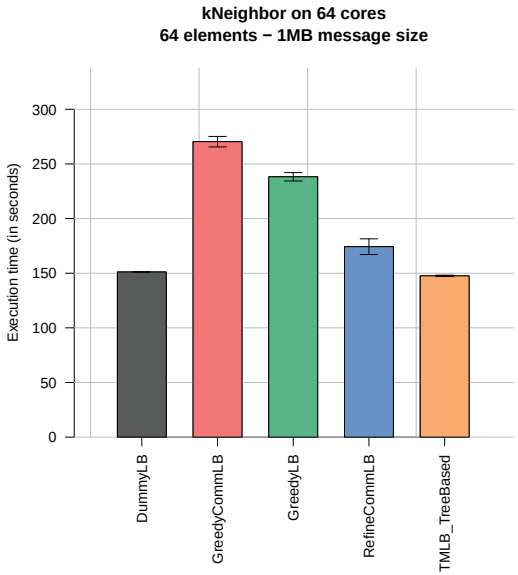
TMLB_TreeBased

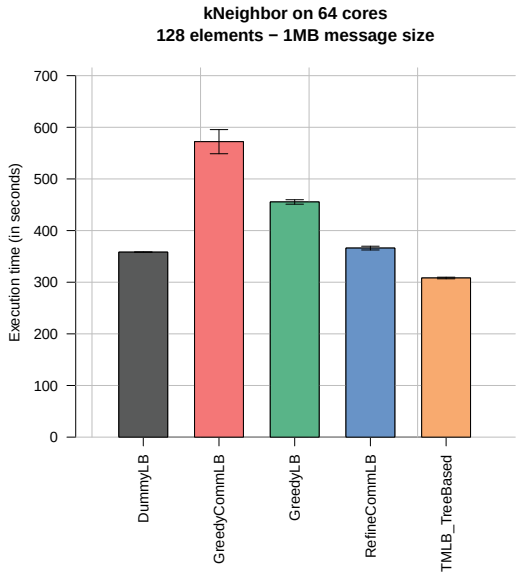
- 1^{st} step : Applies TreeMatch while considering groups of chares on cores
- 2^{nd} step : Reorders chares inside each node
  - Defines the subtree
  - Creates a fake topology with as much leaves as the number of chares + something... (constraints)
  - Applies TreeMatch on this topology and the chares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

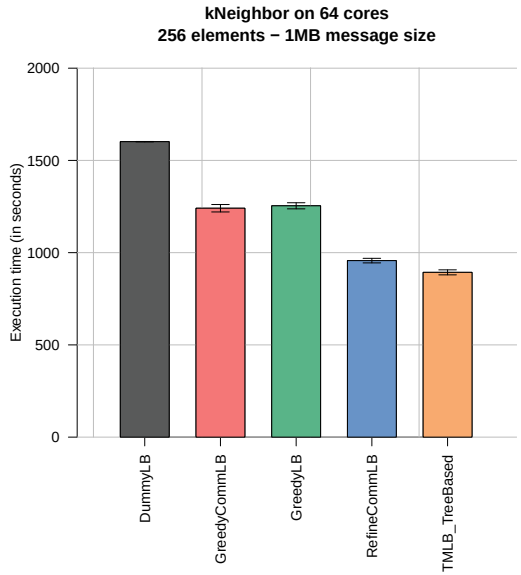


kNeighbor

- 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



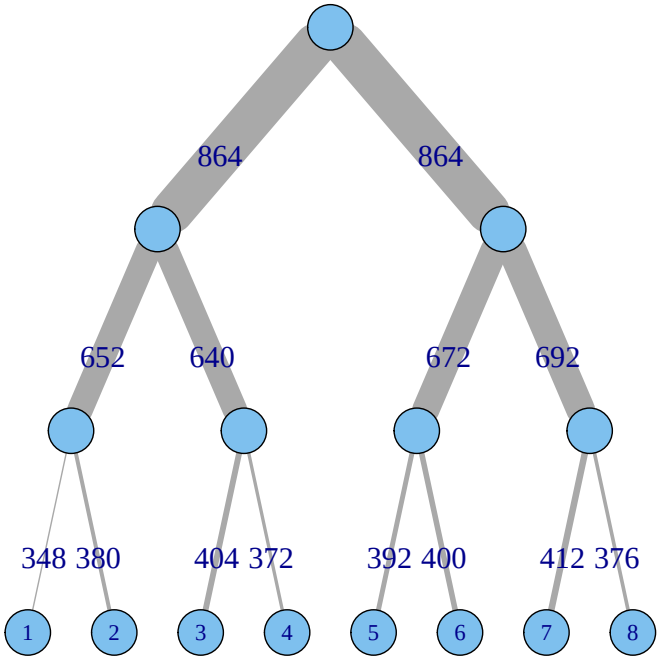




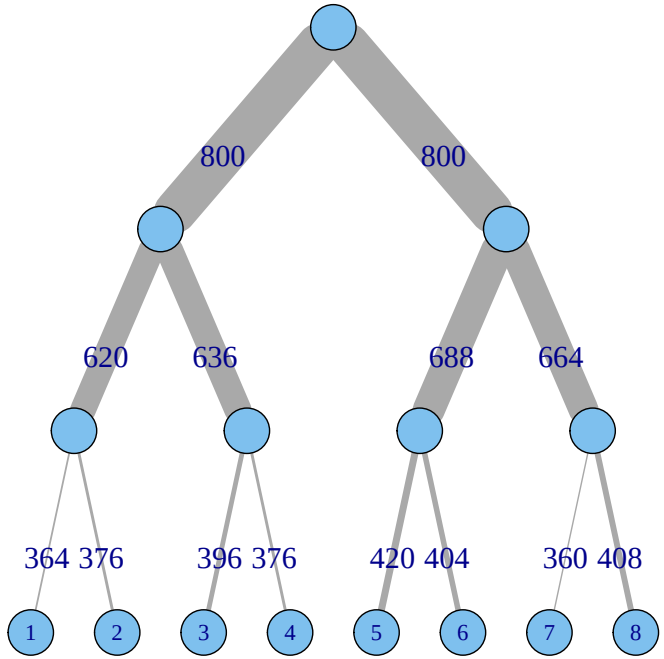
Impact on communication

- Communications evolution between ten iterations

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

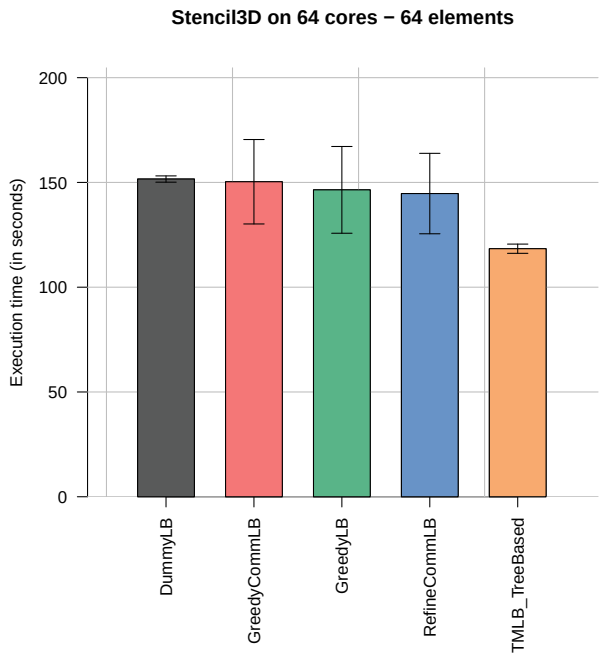


Communication between 10 iterations after the first call of TreeMatchLB (in thousands of messages sent)



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

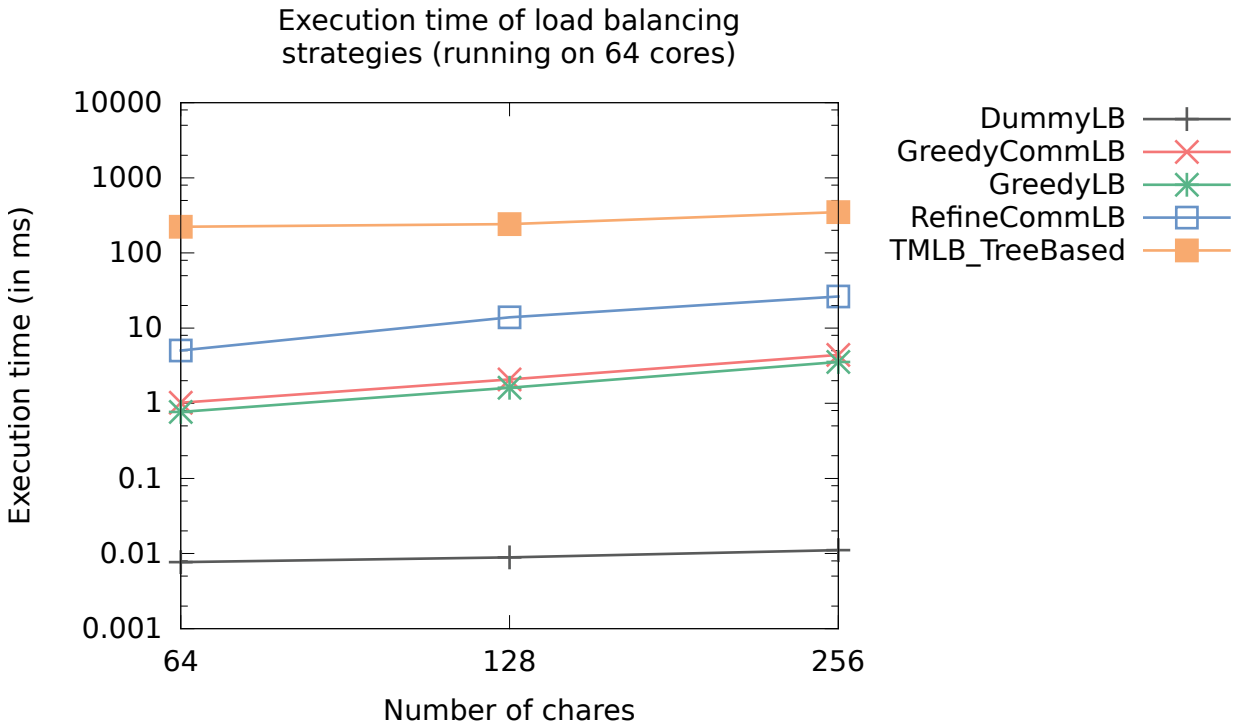


Figure : Load balancing time of the different strategies vs. number of chares for the KNeighbor application.

Future work and Conclusion

Future work

- Find a better way to gather the topology (Hwloc?)
- Distribute the parallel part of TMLB_TreeBased on the different nodes (planned work with the PPL)
- Make TMLB_TreeBased more scalable: allow to chose the level in the hierarchy where the algorithm will be distributed

The end

- Topology is not flat!
- Processes affinities are not homogeneous
- Take into account these information to map chares give us improvement
- Adapt our algorithm to large problems (Distributed)
- Continue collaborations with the PPL
  - Common paper submitted for IEEE Cluster 2013

The End

Thanks for your attention !

Any questions?