Placement of Parallel Applications According to the Topology and the Affinity

PhD Defense

Francois Tessier
Emmanuel Jeannot - Guillaume Mercier

Inria - LaBRI - University of Bordeaux

January 26, 2015

Simulations

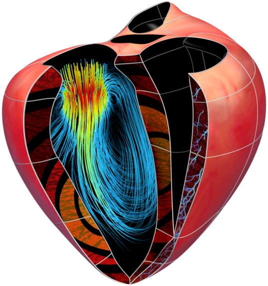


Figure: Heart modelling

- Computer simulation: one of the pillar of science and industry
  - Climate simulation, heart modelling, cosmology, etc.
- Large needs of performance
  - The Human Brain Project goes after at least 1 ExaFLOPS (10^{18} FLOPS) to simulate the brain's neurons
- **Main challenge**: scale these applications
- More parallelization is the only way to meet these requirements
- Implies massively parallel supercomputers

Supercomputers

- Growth of supercomputers to meet the performance needs

	#Nodes	Cores/Node
2011	$\sim 20K$	12
2012	$\sim 10K - 100K$	16 - 32
2015	$\sim 5K - 50K$	100 - 1 000
2018	$\sim 100K - 1000K$	1 000 - 10 000

Source: European Exascale Software Intiative.

- The Blue Waters example
  - More than 400 000 cores
  - Spread to 27 000 nodes
  - Peak performance: 13.34 PetaFLOPS

Figure: The Blue Waters platform

Complex architectures

- Price to pay for the users: topologies are more and more complex
- Complexity of interconnection networks
- Memory hierarchy leading to more NUMA effects
  - Thermal issues for memory banks
  - Gap increasing between the processor and memory performance

	Performance rate
Processor	+60%/y
Memory	+10%/y

  - Not only a power issue but also a bandwidth issue

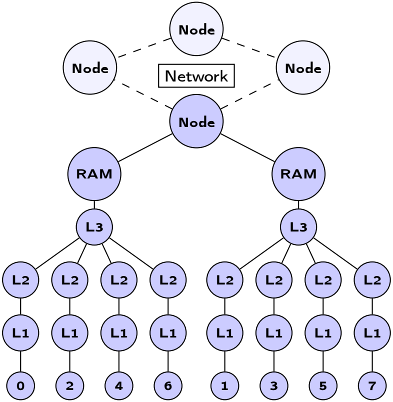


Figure: Typical architecture in HPC

Outline

Outline

Data Locality

- More interesting to access the nearest level in the hierarchy
- **Definition**: distance in hops between a processing entity and the data to which it needs to access

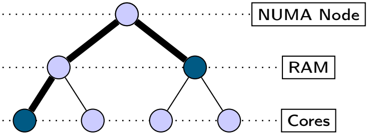


Figure: Data Locality

How to control data locality?

- Locality-aware applications
- Data structures
- Locality-aware languages and compilers
- **Data locality in runtime systems**
  - Runtime improvement
  - **Execution of applications**

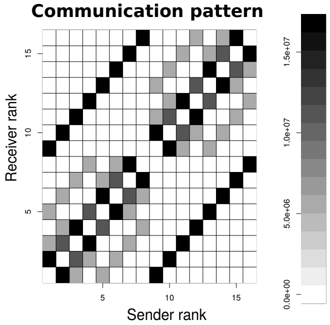
Execution of applications: the placement issue

- One of the levers to optimize the execution of applications: **Application placement**

Two assessments

- Amount of data exchanged between application entities not homogeneous
- Hardware: several levels of hierarchy with various performance
  - Cache hierarchy, memory bus, high-performance network, etc.

→ **Placement policy has an impact on performance**



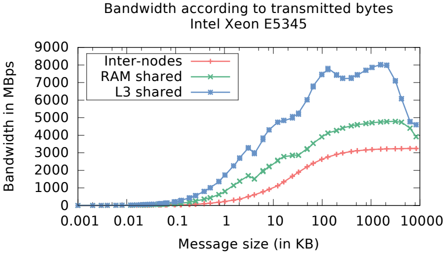
Execution of applications: the placement issue

- One of the levers to optimize the execution of applications: **Application placement**

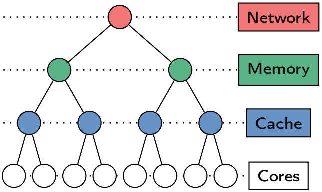
Two assessments

- Amount of data exchanged between application entities not homogeneous
- Hardware: several levels of hierarchy with various performance
  - Cache hierarchy, memory bus, high-performance network, etc.

→ Placement policy has an impact on performance



(a) Higher is better



(b) Topology tree

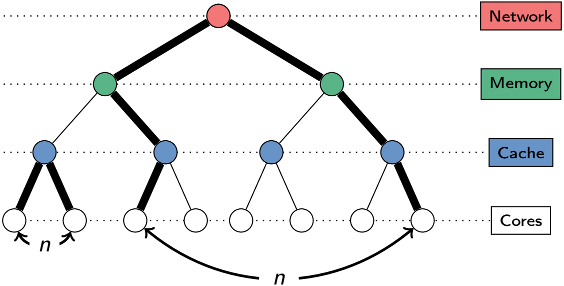
Execution of applications: the placement issue

- One of the levers to optimize the execution of applications: **Application placement**

Two assessments

- Amount of data exchanged between application entities not homogeneous
- Hardware: several levels of hierarchy with various performance
  - Cache hierarchy, memory bus, high-performance network, etc.

→ **Placement policy has an impact on performance**



n: amount of communication

Expected issues

- Placement of parallel applications is a well-known problem
- But encountering scaling issues
  - Amount of memory per core decreasing
  - Accumulation of NUMA effects
  - More and more memory/communication-bound applications

	2011	2012	2015	2018
#Nodes	$\sim 20K$	$\sim 10K - 100K$	$\sim 5K - 50K$	$\sim 100K - 1000K$
Cores/Node	12	16 - 32	100 - 1 000	1 000 - 10 000
Memory (PB)	0.3	0.3 - 0.5	5	32 - 64
GB RAM/Core	0.5 - 4	0.5 - 2	0.2 - 1	0.1 - 0.5

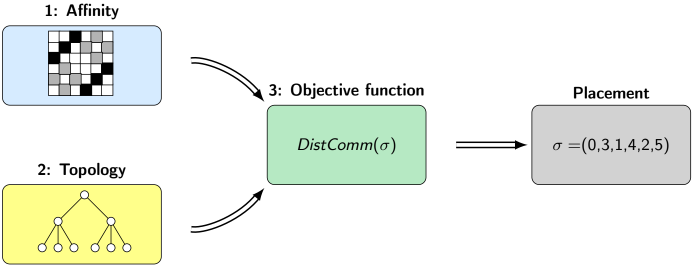
Table: *Source: European Exascale Software Intiative.*

General problems tackled in the PhD thesis
How to take into account data locality for large-scale platforms?

- Too much parallelism to apply application placement by hand
- Development of architectures, all very different
- Need an algorithmic solution considering the hardware characteristics

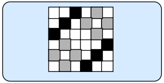
Placement of parallel applications

- How does the application behave?
  - **1: Affinity**
- What is the underlying architecture?
  - **2: Topology (tree)**
- What is the goal?
  - **3: Objective function**



1: Affinity

1: Affinity



- **Definition:** relation between two processing entities according to one or more criteria.
- Possible metrics:
  - **Amount of communication** (e.g. number of messages exchanged)
  - I/O (e.g. amount of data to write on disk)
  - Memory access (e.g. data locality in physical memory banks)
  - etc.

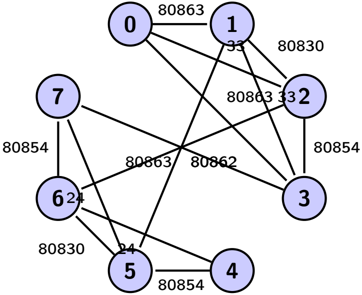
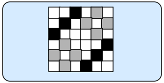


Figure: Affinity pattern (graph) between processing entities

1: Affinity

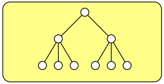
1: Affinity



- **Definition:** relation between two processing entities according to one or more criteria.
- Possible metrics:
  - **Amount of communication** (e.g. number of messages exchanged)
  - I/O (e.g. amount of data to write on disk)
  - Memory access (e.g. data locality in physical memory banks)
  - etc.
- Solutions to gather this affinity pattern
  - **Instrumented versions of runtime implementations (e.g. Open MPI)**
  - **Natively in runtimes (e.g. Charm++)**
  - Trace tools (e.g. Eztrace)
  - Simulation (e.g. SimGrid)
  - Static analysis of the application
  - Data partitioning
  - Skeleton of the application

2: Topology

2: Topology



- Gather topology information
  - No standard means to retrieve this
- hwloc is a solution
  - Abstracts the architecture's characteristics
  - Shows the structure but what about the costs?
- Topology modelling: quantitative or qualitative approach?
  - Qualitative: structural information (provided by hwloc)
  - Quantitative: weighted topology

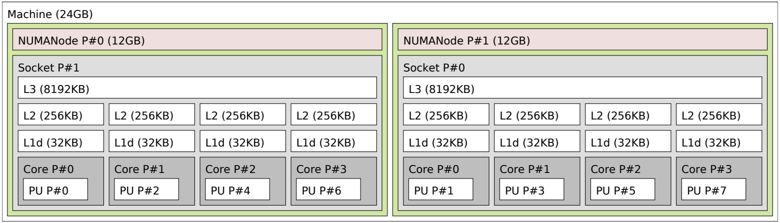
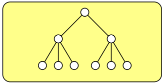


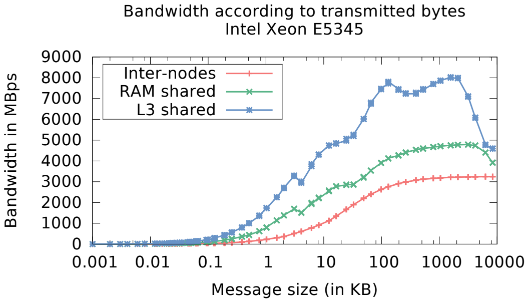
Figure: Hardware topology of an Intel Xeon X5550 architecture

2: Topology

2: Topology



- Gather topology information
  - No standard means to retrieve this
- hwloc is a solution
  - Abstracts the architecture's characteristics
  - Shows the structure but what about the costs?
- Topology modelling: quantitative or qualitative approach?
  - Qualitative: structural information (provided by hwloc)
  - Quantitative: weighted topology



3: Objective function

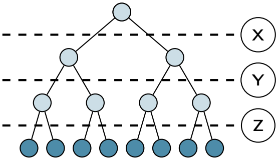
- What do we would like to optimize?
- As input:
  - $A = (V_A, \omega_A)$ the affinity graph with
    - V_A: the processing entities
    - $\omega_A(u, v)$: an affinity metric
  - $H = (V_H, \omega_H)$ the topology tree
    - V_H: the topology nodes
    - $\omega_H(u, v)$ the weight of the topology's edges

3: Objective function

$DistComm(\sigma)$

- Application placement: $\sigma : V_A \rightarrow V_H$
- $DistComm(\sigma)$: **Amount of data weighted by the crossed distance in tree**
  - Qualitative approach
  - Tree topology only

- $DistComm(\sigma) = Z \times 2 + Y \times 4 + X \times 6$
  - Z : Amount of data going through the level 2
  - Y : Amount of data going through the level 1
  - X : Amount of data going through the root
  - We have proved that min $DistComm(\sigma)$ is NP-Hard



3: Objective function

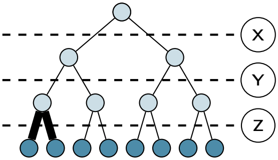
- What do we would like to optimize?
- As input:
  - $A = (V_A, \omega_A)$ the affinity graph with
    - V_A: the processing entities
    - $\omega_A(u, v)$: an affinity metric
  - $H = (V_H, \omega_H)$ the topology tree
    - V_H: the topology nodes
    - $\omega_H(u, v)$ the weight of the topology's edges

3: Objective function

$DistComm(\sigma)$

- Application placement: $\sigma : V_A \rightarrow V_H$
- $DistComm(\sigma)$: **Amount of data weighted by the crossed distance in tree**
  - Qualitative approach
  - Tree topology only

- $DistComm(\sigma) = Z \times 2 + Y \times 4 + X \times 6$
  - Z : Amount of data going through the level 2
  - Y : Amount of data going through the level 1
  - X : Amount of data going through the root
  - We have proved that min $DistComm(\sigma)$ is NP-Hard



3: Objective function

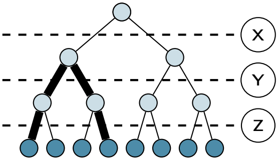
- What do we would like to optimize?
- As input:
  - $A = (V_A, \omega_A)$ the affinity graph with
    - V_A: the processing entities
    - $\omega_A(u, v)$: an affinity metric
  - $H = (V_H, \omega_H)$ the topology tree
    - V_H: the topology nodes
    - $\omega_H(u, v)$ the weight of the topology's edges

3: Objective function

$DistComm(\sigma)$

- Application placement: $\sigma : V_A \rightarrow V_H$
- $DistComm(\sigma)$ **: Amount of data weighted by the crossed distance in tree**
  - Qualitative approach
  - Tree topology only

- $DistComm(\sigma) = Z \times 2 + Y \times 4 + X \times 6$
  - Z : Amount of data going through the level 2
  - Y : Amount of data going through the level 1
  - X : Amount of data going through the root
  - We have proved that min $DistComm(\sigma)$ is NP-Hard



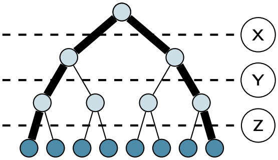
3: Objective function

- What do we would like to optimize?
- As input:
  - $A = (V_A, \omega_A)$ the affinity graph with
    - V_A: the processing entities
    - $\omega_A(u, v)$: an affinity metric
  - $H = (V_H, \omega_H)$ the topology tree
    - V_H: the topology nodes
    - $\omega_H(u, v)$ the weight of the topology's edges
- Application placement: $\sigma : V_A \rightarrow V_H$
- $DistComm(\sigma)$: **Amount of data weighted by the crossed distance in tree**
  - Qualitative approach
  - Tree topology only

3: Objective function

$DistComm(\sigma)$

- $DistComm(\sigma) = Z \times 2 + Y \times 4 + X \times 6$
  - Z : Amount of data going through the level 2
  - Y : Amount of data going through the level 1
  - X : Amount of data going through the root
  - We have proved that min $DistComm(\sigma)$ is NP-Hard



The TreeMatch algorithm

- Algorithm[1,2] and environment to compute processing entities placement based on their affinities and NUMA topology
- Requires tree topology, based on a qualitative approach
- **Input:**
  - The affinity pattern of the application
  - A model (tree) of the underlying architecture (qualitative approach)
- **Output:**
  - A processes permutation σ such that σ_i is the core number on which we have to bind the process i
- **Goal:**
  - $\min DistComm(\sigma)$
- Combinatorial complexity with optimality to 128 processing entities then heuristic for larger input

[1] E. Jeannot and G. Mercier. "Near-optimal placement of MPI processes on hierarchical NUMA architectures". In: *Euro-Par 2010-Parallel Processing* (2010), pp. 199–210.

[2] Emmanuel Jeannot, Guillaume Mercier, and François Tessier. "Process Placement in Multicore Clusters: Algorithmic Issues and Practical Techniques". In: *IEEE Transactions on Parallel and Distributed Systems* (2014).

Issues in application placement for large-scale platforms

Assessments

- Simulation applications need to scale on large and complex platforms
- Hierarchical hardware topologies
- Placement policy has an impact on performance

Problems: How to efficiently place parallel applications according to the affinity and the topology?

Several tracks

- Static placement
  - Proof of concept of TreeMatch on parallel platforms
  - Understand the impact of placement of parallel applications (metrics, etc.)
- Dynamic placement
  - Dynamically improve the data locality during the execution
  - Temporality notion for affinity
  - Combine the topology-aware placement with CPU load balancing

Outline

Static placement

Static placement

- Processing entities mapped at launch time on computing units
- One mapping for the application lifetime

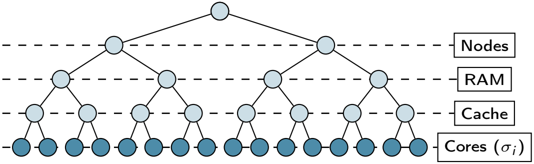
Objectives

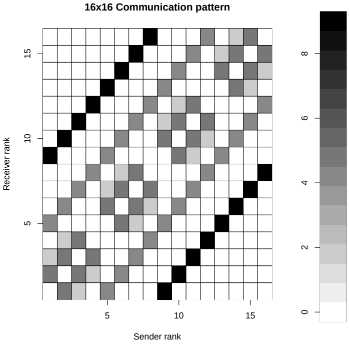
- Minimize execution time but: how?
- Evaluate the relevance of minimizing the communication costs

Contributions

- Proof of concept of TreeMatch for parallel platform
  - Case study to evaluate the static application placement and the impact of affinity metrics
- TreeMatch improvements
  - Taking into consideration constraints
  - Proof of the NP-Completeness of min $DistComm(\sigma)$

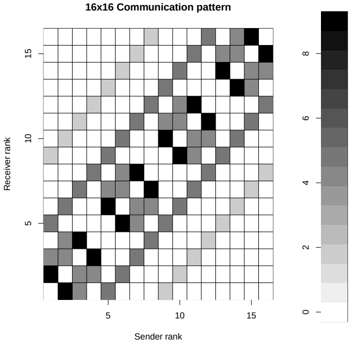
Toy example



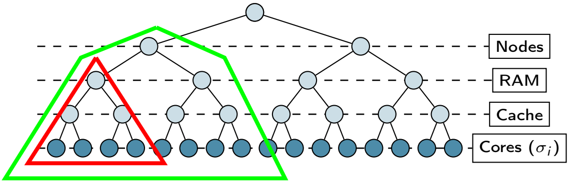


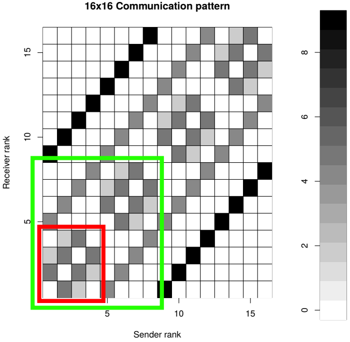
σ =(0,2,8,10,4, 6,12,14,1,3,9, 11,5,7,13,15)

$\Longrightarrow$



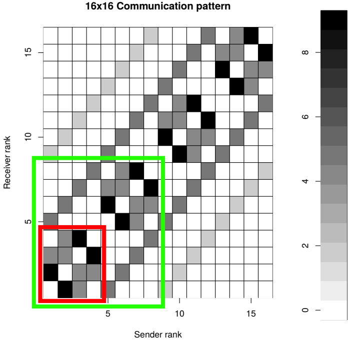
Toy example





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

$\Longrightarrow$



State of the Art

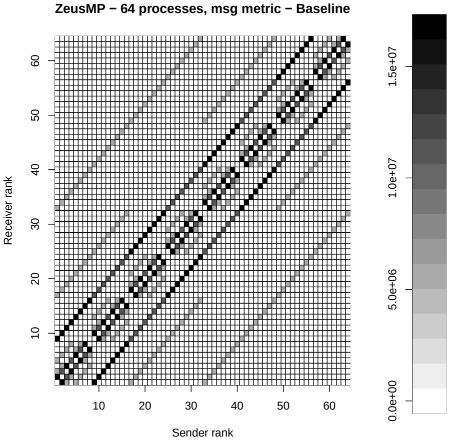
- Graph partitionners are able to give a solution to the placement problem
- Scotch (6.0.0), Chaco (2.2) and ParMETIS (3.1.1): graph partitionners
- MPIPP: randomized algorithm

Methods	Hardware independent	Paradigm independent	NUMA effects	Network	Qualitative approach	Dynamic topology
MPIPP	√	√		√		
Scotch	√	√		Tree		
Chaco	√	√	√			
ParMETIS	√	√	√			√
LibTopoMap	√			√		
Träff		√		√		
TreeMatch	√	√	√	Tree	√	√

- Comparison to hardware and paradigm independent methods
- Case study of a real application

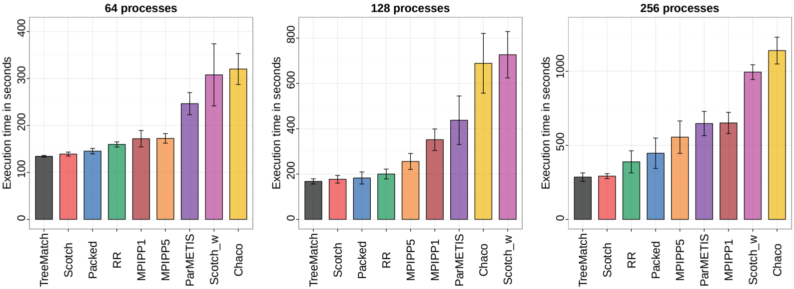
Case study

- PlaFRIM cluster
  - Nodes: Intel Xeon 5550 - 8 cores - 12 GB RAM
- ZeusMP/2
  - CFD Application
  - Irregular communication pattern



Case study - Results

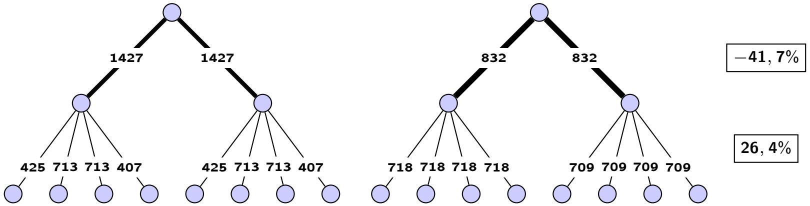
- *Packed* and *Round Robin*: standard strategies (Packed is the default mapping in Open MPI)
- **TreeMatch outperforms *Packed* and *RR* up to 25%**
- Two versions of Scotch
  - Scotch_w : weighting of the topology after benchmarking
  - Scotch: Normalized weights
  - TreeMatch slightly better or comparable



(a) 64 processes (b) 128 processes (c) 256 processes

Case study - Communication distribution

- Impact of static placement on the amount of communication going through the topology links
- ZeusMP/2 on 16 cores (2 nodes), one node depicted
  - In thousand of messages exchanged
- A large amount of communication transferred to the subtrees

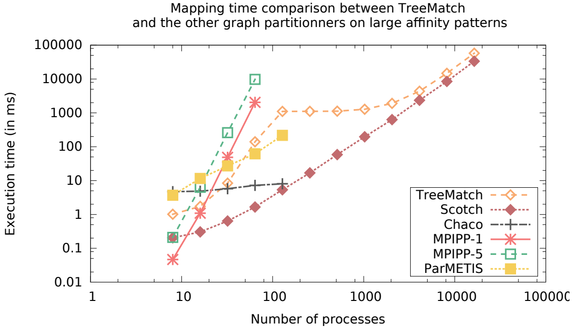


(a) With default placement

(b) With TreeMatch placement

Mapping time for benchmarks (Sparse matrices)

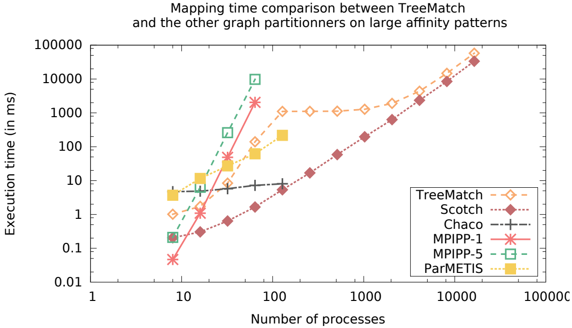
- New heuristic for TreeMatch makes it to improve scalability beyond 128 processes
- Follow a linear curve on large cases
- Around 1 second to 128 processes then comparable to Scotch



The mapping time is a scalability constraint for dynamic placement

Mapping time for benchmarks (Sparse matrices)

- New heuristic for TreeMatch makes it to improve scalability beyond 128 processes
- Follow a linear curve on large cases
- Around 1 second to 128 processes then comparable to Scotch



The mapping time is a scalability constraint for dynamic placement

Outline

Goals and programming model

Objectives

- Improve data locality dynamically
- Take advantage of load balancing systems to add a topology-aware component
- Consider affinity temporality

Charm++

- Fine-grained paradigm: cooperating objects called **chares**
- Plugable load balancing algorithms at launch time
  - Native Charm++ load balancers
  - Quantitative topology-aware load balancers: NucoLB, HwTopoLB[3]
- Load balancers able to natively **migrate** chares
- Adaptive runtime system supplying chares and cores statistics (load, affinity, etc.)

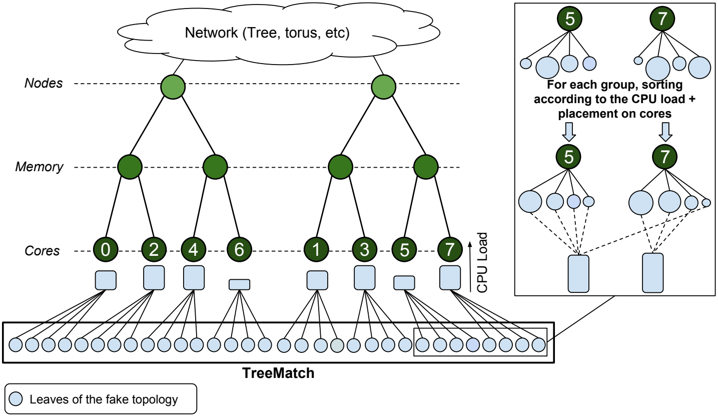
Contributions

- Two load balancers respectively for:
  - Compute-bound applications
  - Communication-bound applications
- Work in collaboration with the JLPC and the PPL

[3] Laércio L Pilla, Christiane Pousa Ribeiro, Daniel Cordeiro, Chao Mei, Abhinav Bhatele, Philippe OA Navaux, François Broquedis, Jean-François Méhaut, and Laxmikant V Kale. "A Hierarchical Approach for Load Balancing on Parallel Multi-core Systems". In: *Parallel Processing (ICPP), 2012 41st International Conference on*. IEEE. 2012, pp. 118–127.

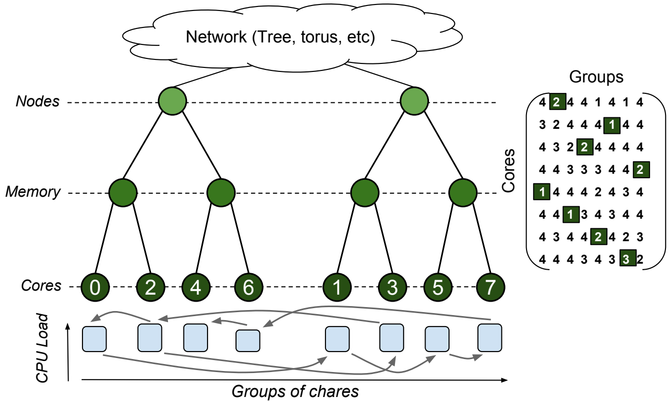
TMLB_Min_Weight for compute-bound applications

- Load balancing for compute-bound applications
- Algorithm steps
  - Reorders chares on cores with TreeMatch (favouring CPU load balancing)
  - Reorders groups of chares on cores to minimize the migrations
- Assignment problem resolved by the Hungarian algorithm
  - Find a independent set of minimal weight
  - Applied on migration cost matrix



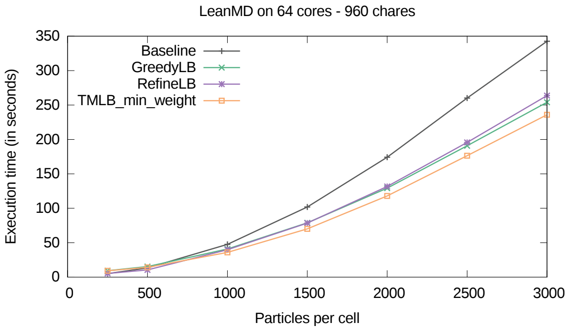
TMLB_Min_Weight - Minimizing migrations

- Load balancing for compute-bound applications
- Algorithm steps
  - Reorders chares on cores with TreeMatch (favouring CPU load balancing)
  - Reorders groups of chares on cores to minimize the migrations
- Assignment problem resolved by the Hungarian algorithm
  - Find a independent set of minimal weight
  - Applied on migration cost matrix



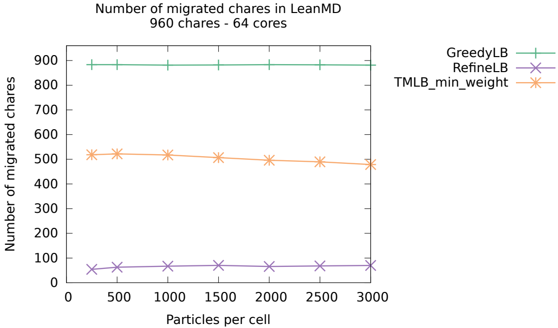
TMLB_Min_Weight - Results

- LeanMD: Charm++-based molecular dynamics application
  - Compute-bound application
  - Very unbalanced
- Compared to natives Charm++ load balancers
  - GreedyLB: highest loaded chare on less loaded core
  - RefineLB: chares from overloaded cores to less loaded ones to reach average
- Up to 30% of gain compared to the baseline and between 5% and 10% compared to the native load balancers
- Amount of migrations
  - Migration time reduced by 5% with the Hungarian algorithm



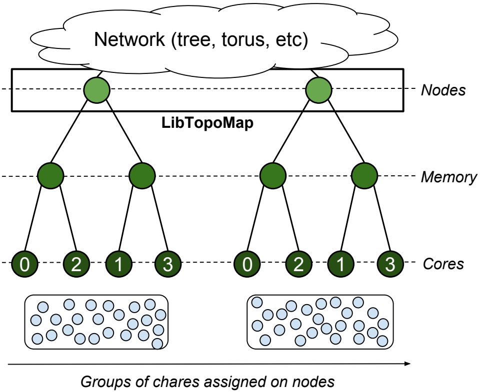
TMLB_Min_Weight - Results

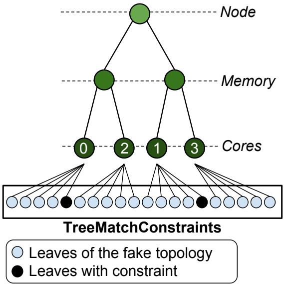
- LeanMD: Charm++-based molecular dynamics application
  - Compute-bound application
  - Very unbalanced
- Compared to natives Charm++ load balancers
  - GreedyLB: highest loaded chare on less loaded core
  - RefineLB: chares from overloaded cores to less loaded ones to reach average
- Up to 30% of gain compared to the baseline and between 5% and 10% compared to the native load balancers
- Amount of migrations
  - Migration time reduced by 5% with the Hungarian algorithm



TMLB_TreeBased for communication-bound applications

- Load balancing for communication-bound applications
- Hierarchical and distributed algorithm
  - Reorders groups of chares on nodes (LibTopoMap)
  - Reorders chares inside each node: TreeMatch with constraints
  - Each node in parallel





TMLB_TreeBased for communication-bound applications

- Load balancing for communication-bound applications
- Hierarchical and distributed algorithm
  - Reorders groups of chares on nodes (LibTopoMap)
  - Reorders chares inside each node: TreeMatch with constraints
  - Each node in parallel
- Algorithm designed for scalability
  - Consider the network to perform a first placement on nodes
  - Parallel and distributed topology-aware load balancing inside nodes
  - No sensitivity to initial placement

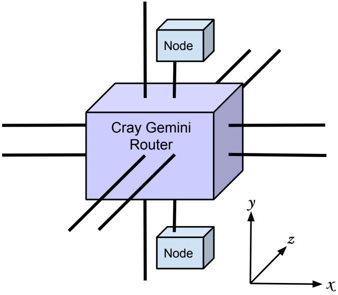
TMLB_TreeBased - Network

How to deal with the network topology?

- TreeMatch works only on tree topologies
- LibTopoMap: library able to place processes on any network topology

Example: 3D torus Cray Gemini network

- Algorithm steps
  - Convert the batch scheduler allocation to a readable format for LibTopoMap
  - Apply network placement (groups of chares on nodes) with LibTopoMap



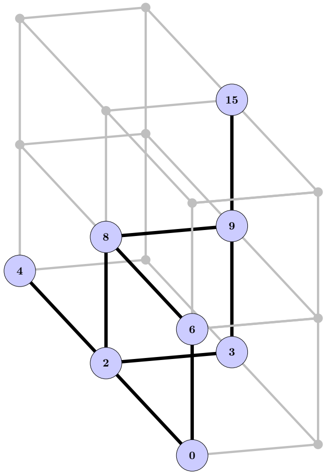
TMLB_TreeBased - Network

How to deal with the network topology?

- TreeMatch works only on tree topologies
- LibTopoMap: library able to place processes on any network topology

Example: 3D torus Cray Gemini network

- Algorithm steps
  - Convert the batch scheduler allocation to a readable format for LibTopoMap
  - Apply network placement (groups of chares on nodes) with LibTopoMap



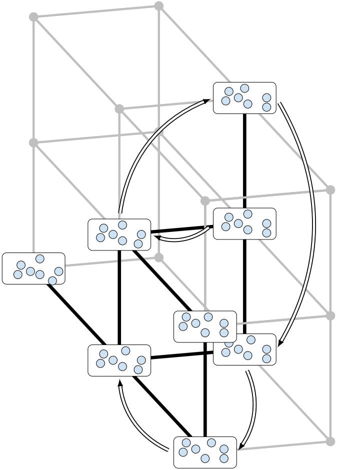
TMLB_TreeBased - Network

How to deal with the network topology?

- TreeMatch works only on tree topologies
- LibTopoMap: library able to place processes on any network topology

Example: 3D torus Cray Gemini network

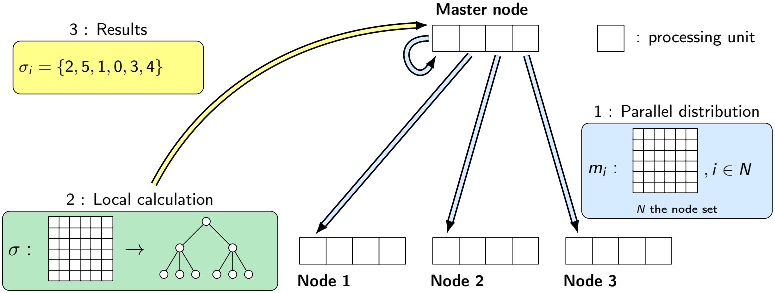
- Algorithm steps
  - Convert the batch scheduler allocation to a readable format for LibTopoMap
  - Apply network placement (groups of chares on nodes) with LibTopoMap



TMLB_TreeBased - Parallelization

How to improve the algorithm scalability?

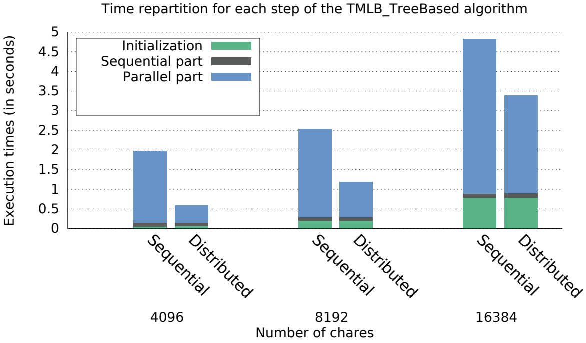
- Parallelized and distributed version of TMLB_TreeBased
  - Two levels of parallelization
    - OpenMP
    - The Charm++ mechanisms for distribution
  - Up to 130% of improvement compared to the fully sequential version
    - Carried out on 16 nodes (32 cores/node)
    - Parallel part: TreeMatch called on each node



TMLB_TreeBased - Parallelization

How to improve the algorithm scalability?

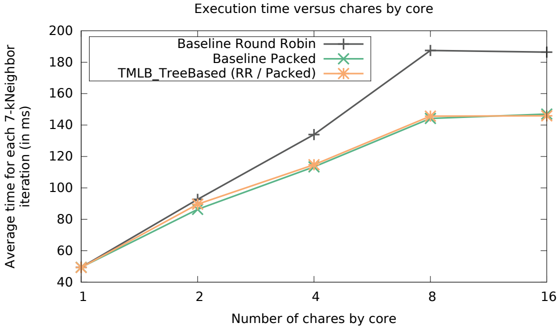
- Parallelized and distributed version of TMLB_TreeBased
  - Two levels of parallelization
    - OpenMP
    - The Charm++ mechanisms for distribution
  - Up to 130% of improvement compared to the fully sequential version
    - Carried out on 16 nodes (32 cores/node)
    - Parallel part: TreeMatch called on each node



TMLB_TreeBased - Behavior faced with the initial placement

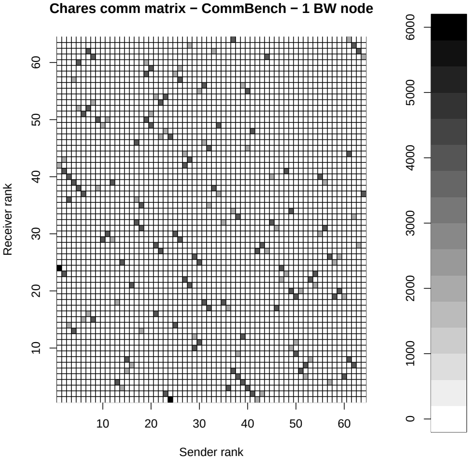
What is the sensitivity of TMLB_TreeBased to initial placement?

- Application (kNeighbor) for which the optimal placement is known
- Testbed: Intel Xeon Nehalem X5550 (8 cores)
- TMLB_TreeBased VS optimal placement VS default placement
  - The initial mapping may vary according to the core numbering
- No sensitivity of TMLB_TreeBased to initial placement
- Converge to the optimal placement



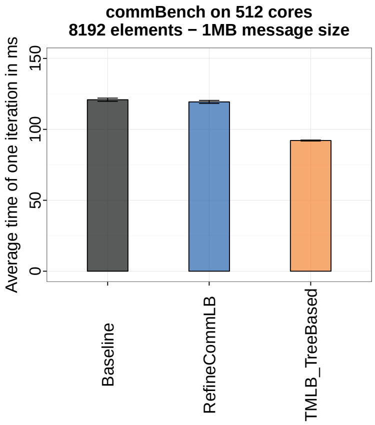
TMLB_TreeBased - commBench

- Benchmark simulating irregular communications
- Execution time improvement up to 25% on 512 cores
- RefineCommLB: locality-aware version of RefineLB



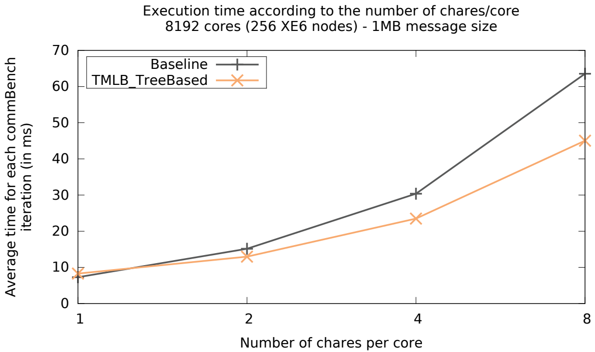
TMLB_TreeBased - commBench

- Benchmark simulating irregular communications
- Execution time improvement up to 25% on 512 cores
- RefineCommLB: locality-aware version of RefineLB



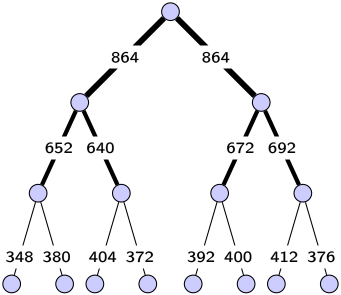
TMLB_TreeBased - commBench

- Scalability: 8192 cores (256 XE6 nodes) on Blue Waters
- Up to 65536 chares, i.e. 8 chares/core
- Native Charm++ load balancers do not work at such scale
- 28% of improvement compared to baseline

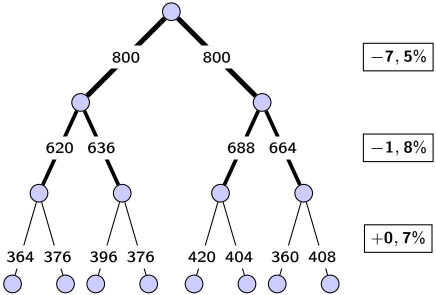


TMLB_TreeBased - Results analysis

- Communication distribution on the topology links before and after the call to TMLB_TreeBased (in thousands of messages)
  - Locality of communication improved



(a) Before load balancing



(b) After the call to TMLB_TreeBased

Dynamic placement - partial conclusion

- Able to apply topology-aware load balancing
  - For compute-bound applications
  - For communication-bound applications
- Joint work with the PPL at Urbana and the JLPC[4]

[4] François Tessier, Emmanuel Jeannot, Esteban Meneses, Guillaume Mercier, and Gengbin Zheng. "Communication and Topology-aware Load Balancing in Charm++ with TreeMatch". Anglais. In: *IEEE Cluster 2013*. Indianapolis, États-Unis: IEEE, Sept. 2013. URL: http://hal.inria.fr/hal-00851148.

Outline

Conclusion

Problems

- Take into account data locality for applications in large-scale platform
- More precisely, efficiently place parallel applications according to the affinity and the topology

Contributions

- Static placement
  - Proof of concept of the TreeMatch algorithm on parallel platforms
  - Proof that min $DistComm(\sigma)$ is NP-Hard
  - Significant amount of experiments
  - Improve real application up to 25% compared to default mappings
- Dynamic placement
  - Application independent Charm++ load balancers for compute-bound and communication-bound applications
  - Up to 30% of gain on a compute-bound application
  - Outperforms by 25% the native load balancers on large-scale experiments
  - Overcomes a limitation of the TreeMatch algorithm: oversubscribing

Perspectives

Short and medium term

- TreeMatch algorithm improvements
  - Include partitioning algorithms from Scotch
  - Network awareness: LibTopoMap, Scotch, hwloc
  - Oversubscribing management implementation
- Better understand the criteria that impact performance when doing placement
  - Hardware counters
  - Skeleton of applications

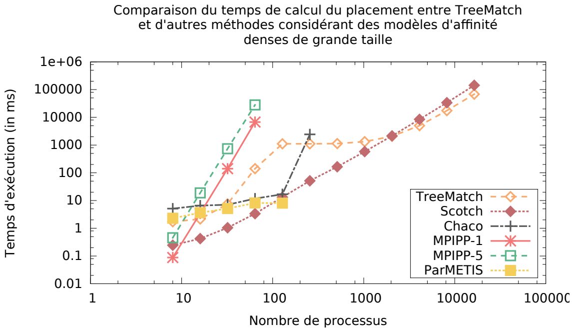
Long term

- How to measure affinity ?
- Other ways to take action in applications execution
  - Affinity-aware job allocations: Adèle Villiermet PhD thesis
- Placement techniques for storage resources: new collaboration with ANL in the context of the JLESC
  - Topology-aware I/O aggregation

Conclusion

Thank you for your attention!

Mapping time on dense affinity matrices



Mapping time on dense affinity matrices

- Load balancing time compared to other strategies
  - TMLB_TreeBased is slower than the native strategies
  - Counterbalanced by the quality of the topology-aware load balancing