Distributed communication-aware load balancing with TreeMatch in Charm++

The 11th workshop of the Joint Laboratory for Petascale Computing, Sophia-Antipolis

Emmanuel Jeannot  Guillaume Mercier  Francois Tessier

*In collaboration with the Charm++ Team from the PPL:* Esteban Meneses-Rojas, Gengbin Zheng, Sanjay Kale

June 9, 2014
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++** considering:
  - CPU load
  - process affinity (or other communicating objects)
  - topology: network and intra-node
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.
- Charm++ is able to use MPI for the processes communications.
Processes Placement

Why we should consider it

- Many current and future parallel platforms have several levels of hierarchy
- Application shares/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...

Diagram:
- Switch
- Cabinet
- Cabinet
- Node
- Node
- Processor
- Processor
- Core
- Core
- Core
- Core

Francois Tessier
TreeMatch in Charm++
Given

- The parallel machine topology
- The application communication pattern

- Map application processes to physical resources (cores) to reduce the communication costs (NP-complete)
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 model (tree) of the underlying architecture: Hwloc can provide us this.
- Output:
  - A processes permutation $\sigma$ such that $\sigma_i$ is the core number on which we have to bind the process $i$
- TreeMatch can only work on tree topologies. How to deal with 3d torus?
**libtopomap**

- Library that enables to map processes on various network topologies
- Used in TreeMatchLB to consider the Blue Waters 3d torus

**Figure:** 3d Torus and a Cray Gemini router
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

**Some other communication-aware load-balancing algorithms**
- Some "built-in" Charm++ load balancers: RefineCommLB, GreedyCommLB...
Several issues raised

- Several issues raised!
- Scalability of TreeMatch
- Need to find a relevant compromise between processes affinities and load balancing
- What about load balancing time?

The next slides will present our load balancer relying on TreeMatch and libtopomap which performs a parallel and distributed communication-aware load balancing.
First step: minimize communication cost on network

- libtopomap reorders processes from a communicator
- How to use it to reorder groups of processes (or chares)? Example: groups of chares on nodes
  - Charm++ uses MPI: full access to the MPI API
  - New MPI communicator with MPI_Comm_split

Network (3d torus, tree, ...)

Nodes

New communicator

Francois Tessier
Strategy for Charm++ - Intra-node placement

TreeMatch load balancer

1\textsuperscript{st} step: Remap groups of chares on nodes according to the communication on the network
- libtopomap (example: part of 3d Torus)

2\textsuperscript{nd} step: Reorder chares inside each node (distributed)
- Apply TreeMatch on the NUMA topology and the chares communication pattern
- Bind chares according to their load (leveling on less loaded chares)
- Each node carries out its own placement in parallel
Strategy for Charm++ - Intra-node placement

TreeMatch load balancer

1\textsuperscript{st} step: Remap groups of chares on nodes according to the communication on the network
- \texttt{libtopomap} (example: part of 3d Torus)

2\textsuperscript{nd} step: Reorder chares inside each node (distributed)
- Apply TreeMatch on the NUMA topology and the chares communication pattern
- Bind chares according to their load (leveling on less loaded chares)
- Each node carries out its own placement in parallel

\textbf{Figure:} Part of a 3d Torus attributed by the resource manager
TreeMatch load balancer

1\textsuperscript{st} step: Remap groups of chares on nodes according to the communication on the network
- libtopomap (example: part of 3d Torus)

2\textsuperscript{nd} step: Reorder chares inside each node (distributed)
- Apply TreeMatch on the NUMA topology and the chares communication pattern
- Bind chares according to their load (leveling on less loaded chares)
- Each node carries out its own placement in parallel
Strategy for Charm++ - Intra-node placement

**TreeMatch load balancer**

- 1\(^{st}\) step: Remap groups of chares on nodes according to the communication on the network
  - libtopomap (example: part of 3d Torus)
- 2\(^{nd}\) step: Reorder chares inside each node (distributed)
  - Apply TreeMatch on the NUMA topology and the chares communication pattern
  - Bind chares according to their load (leveling on less loaded chares)
  - Each node carries out its own placement in parallel
Strategy for Charm++ - Intra-node placement

TreeMatch load balancer

- 1\textsuperscript{st} step: Remap groups of chares on nodes according to the communication on the network
  - libtopomap (example: part of 3d Torus)
- 2\textsuperscript{nd} step: Reorder chares inside each node (distributed)
  - Apply TreeMatch on the NUMA topology and the chares communication pattern
  - Bind chares according to their load (leveling on less loaded chares)
  - Each node carries out its own placement in parallel
TreeMatch load balancer

1\textsuperscript{st} step: Remap groups of chares on nodes according to the communication on the network
   - libtopomap (example: part of 3d Torus)

2\textsuperscript{nd} step: Reorder chares inside each node (distributed)
   - Apply TreeMatch on the NUMA topology and the chares communication pattern
   - Bind chares according to their load (leveling on less loaded chares)
   - Each node carries out its own placement in parallel
Results

**kNeighbor**

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

- Experiments on 16 nodes with 8 cores on each (Intel Xeon 5550) - PlaFRIM Cluster
- 1 MB messages - 100 iterations - 7-Neighbor

![Execution time versus shares by core](image)

- Average time for each 7-kNeighbor iteration (in ms)
- Number of shares by core
- DummyLB
- TreeMatchLB

Francois Tessier

TreeMatch in Charm++
kNeighbor

- Experiments on 16 nodes with 32 cores on each (AMD Interlagos 6276) - Blue Waters Cluster
- 1 MB messages - 100 iterations - 7-Neighbor
- Bad performances...

![Graph showing execution time versus cores by core](image)

Average time for each 7-kNeighbor iteration (in ms)
Number of cores by core

- DummyLB
- TreeMatchLB

Francois Tessier
TreeMatch in Charm++
**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)
Results

What about the load balancing time?

- Comparison between the sequential and the distributed versions of TreeMatchLB

- The master node distributes the data to each node which will compute its own characters placement. This data distribution can be done in parallel (around 20% of improvements)
Results

What about the load balancing time?

- Comparison between the sequential and the distributed versions of TreeMatchLB
- The master node distributes the data to each node which will compute its own chances placement. This data distribution can be done in parallel (around 20% of improvements)
What about the load balancing time?

- Linear trajectory while the number of chares is doubled
- TreeMatchLB is slower than the other Greedy strategies
- RefineCommLB which provides some good results for communication-bound applications is not scalable (fails from 8192 chares)

![Graph showing load balancing time vs. number of chares]

**Figure:** Load balancing time of the different strategies vs. number of chares for the KNeighbor application.
Future work and Conclusion

The end

- Topology is not flat!
- Processes affinities are not homogeneous
- Take into account these information to map chares give us improvement
- Algorithm adapted to large problems (Distributed)

JLPC collaborations

- 10 days during August 2013 at the PPL
- Paper accepted at IEEE Cluster 2013

Future work

- Find a better way to gather the topology (Hwloc?)
- Bad performances on Blue Waters... Need to understand why (Architecture ?)
- Perform more large scale experiments
- Hybrid architecture? Intel MIC?
- Evaluate our solution on other applications
- Compare to other load balancer (NUCOLB, application-specific LBs)
Thanks for your attention!
Any questions?