Topology-aware and load balancing techniques using Charm++, AMPI and related technologies

Juan Galvez, Laxmikant Kale

Blue Waters Symposium - June 6, 2018

PPL UIUC Parallel Programming Laboratory

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Charm++ and AMPI

- Charm++ / Charmpy
  - Object-oriented parallel programming model w/ *migratable* objects
  - Adaptive runtime system
    - asynchronous message driven execution (communication and computation overlap)
    - dynamic load balancing and communication optimization
    - fault tolerance
- AMPI (Adaptive MPI)
  - Run MPI applications on top of Charm++
  - Can have multiple migratable MPI ranks per process
  - Obtain benefits of Charm++ runtime system

PAID project goals

- Improve *resource utilization of applications* on Blue Waters
  1) Topology-aware placement of tasks, to improve communication performance
  2) Load balancing, to minimize CPU idle time
- Applications include:
  - HARM3D (Scott Noble) - Load balancing using AMPI
  - PSDNS-CCD3D (PK Yeung) - Topology-aware mapping for MPI
  - MILC (Steven Gottlieb) – Topology-aware mapping for MPI
  - NAMD, OpenAtom, ChaNGa (topology and load balancing)

Outline of talk

- Topology-aware mapping
- Load balancing
  - Charm++
  - AMPI
  - Recent developments in Charm++ load balancing

Topology-aware mapping

- Communicating tasks (e.g. MPI ranks) should be placed in same or nearby nodes
  - Reduce *latency* and *network load*
- **Goal**: place tasks such that load of network links is minimized → NP-hard
- Match the application communication graph to the machine topology
  - Effect is highly dependent on machine topology, application and job size
  - Blue Waters has 3D torus topology
  - On Blue Waters performance can improve by > 2x

Automatic task mapping tool

- Maps MPI ranks to cores in a given job allocation, during application launch
- Available as system module on Blue Waters
  - /sw/bw/topomapping/README.txt
- How it works
  - Do profiling run (obtain communication graph)
  - In production runs. Inside batch job (example):

```
$ module load topomapping
$ aprun -n 8192 mapper -t 30 -f comm_graph.txt  # comm graph from profiling run
$ export MPICH_RANK_REORDER_METHOD=3
$ aprun -n 8192 ./application app_parameters
```

Automatic task mapping tool

- Application observed speedup
  - MILC: up to **2.2x**
  - Qbox: up to **70%**
  - CCDKernel: up to **22.6%**
  - PSDNS-CCD3D: **3.5%**
- Summary
  - Good for applications with heavy point-to-point communication
  - Harder to map applications with heavy use of collectives

Load balancing

- Many science applications (e.g. biology, cosmology, fluid) can exhibit load imbalance
  - Cores do different amount of work
  - All cores must wait for slowest core (many idle cores at very large scales)
- **Load balancing (LB) problem**:
  - Given set of tasks, each task t has load l_t
  - Assign tasks to processors such that
  - Minimize maximum load M (aka makespan) of processors
  - <u>Performance can improve by up to</u> $\frac{M^{nolb}}{M^{opt}}$

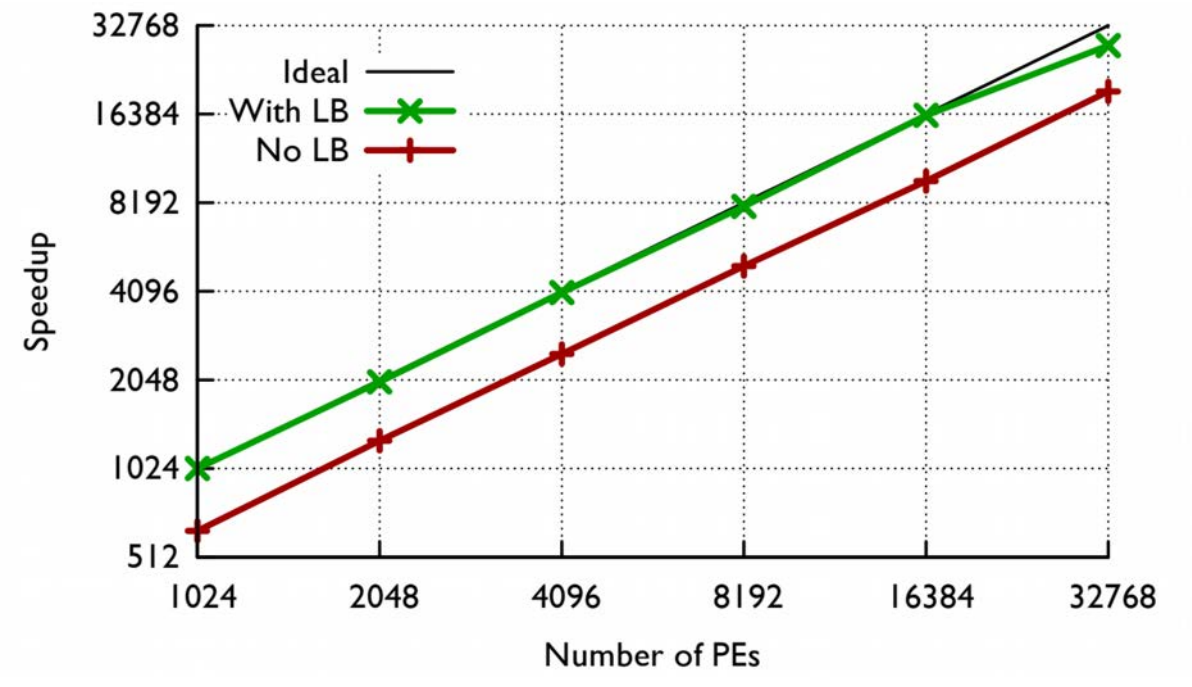
Challenges of load balancing in HPC

- Increasingly larger machines
  - Hundreds of thousands of cores, millions of tasks
  - Strong scaling makes tasks smaller, LB problem larger
- **Costs of LB in large parallel applications**
  - Collecting *statistics*
    - Efficiency of statistics communication
    - Comm-aware LB requires comm graph, but expensive to collect and use
  - Cost of calculating *task-to-core mapping*
  - *Migrating tasks*
    - Move fewer tasks
    - Move tasks within same node or nearby is cheaper

Load balancing with Charm++

- Charm++ provides automatic load balancing
  - Application transparent
  - Balance load within and across nodes
- How it works
  1) Runtime measures load of objects
  2) Calculates new assignment of objects to cores
  3) Migrates objects

Charm++ balancing results



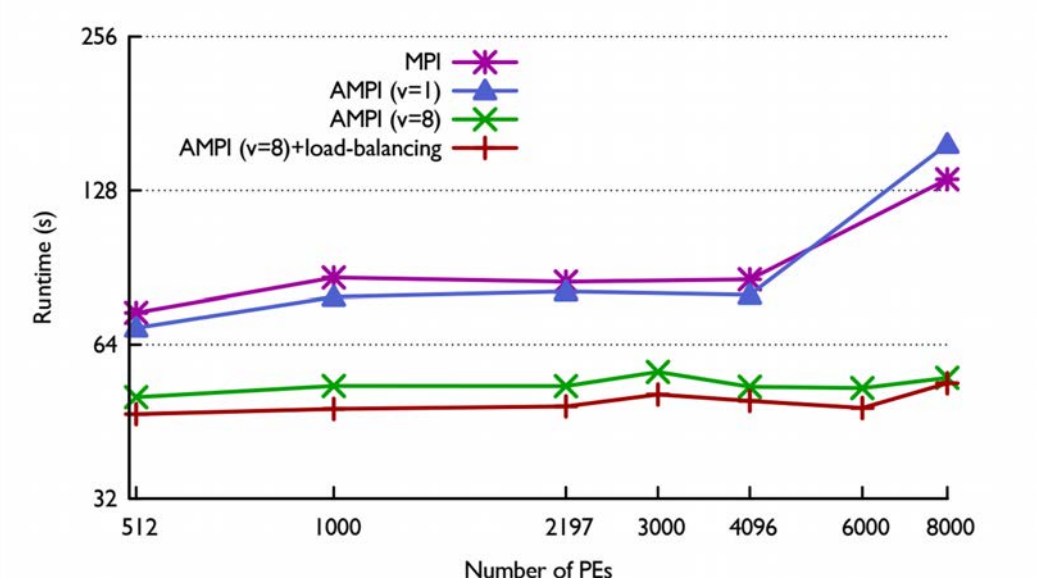
Charm++ LeanMD results

LB for non-Charm++ applications

- Load balancing *library*
  - Subset of load balancing algorithms from Charm++
  - Application still has to measure load and move work
- AMPI
  - Run MPI applications on top of Charm++, without change (if no global variables) or by tagging globals

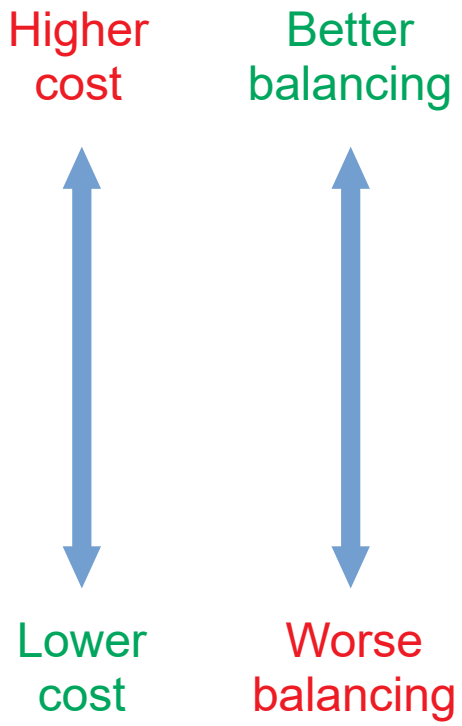
AMPI

- HARM3D: 10% speedup using Charm++ Greedy load balancer (of 20% max possible speedup)
- LULESH AMPI results



Types of load balancing

- *Centralized*
  - Global view of objects and cores
- *Hierarchical*
  - Cores/processes organized in tree structure
  - Information is aggregated in upper levels
- *Distributed*
  - Cores make decisions based on “local” info



Centralized load balancing

- *Greedy* heuristic very good for general load balancing problem
  - Sort objects by load, assign current object to current least loaded processor $O(n \log n)$
  - 4/3 (worst case) approximation ratio, cases where optimal or very close to optimal
- **But**: it can move all the objects, move them anywhere

GreedyRefine

- Greedy variant: in algorithm, assign current object to previous core if load doesn't exceed a threshold Δ, otherwise to least loaded
- Δ can be greedy maxload, or $\alpha \times \text{avgload}$ (avgload is lower bound of optimum maxload)
- <u>Solution similar to greedy, while limiting object migration</u>
- Load balancing time has been reduced by 4-7x compared to greedy
  - Less migrations
  - Faster implementation of algorithm
  - Performance optimizations in lb framework
- *TopoGreedyRefine*: constrain object migration to least loaded core *in vicinity*

LB benchmarks

Load balancing algorithm	Time per lb step (s)	Migration time per lb step (s)	Total program time (s)
Greedy	2.22		64.9
GreedyRefine	0.354	0.115	23.9
TopoGreedyRefine	0.316	0.064	19.93

- 3D stencil code with synthetic imbalance
- 1024 nodes on Blue Waters, 29k cores and 229k objects
- TopoGreedyRefine preserves initial mapping
  - Better communication performance

Hierarchical load balancing

- Added topology-aware trees
  - Tree is based on network topology
  - Cores are grouped based on their proximity
- NAMD
  - Uses hierarchical load balancing
  - Uses custom topology-aware load balancer
  - Topology-aware trees help preserve mapping: avg 10% speedup

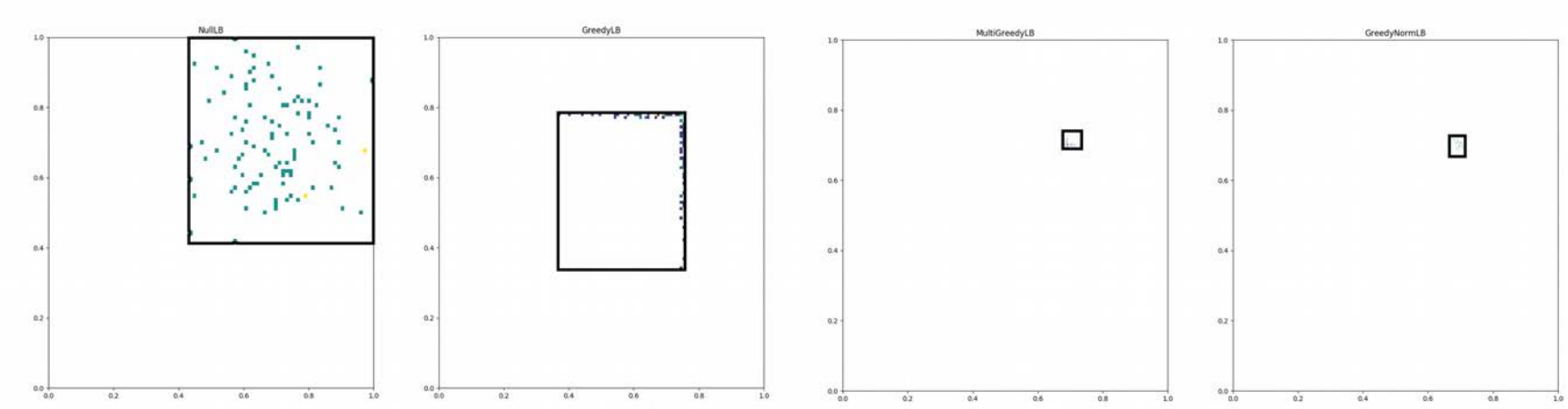
Distributed graph refinement

- When load balancing, node sends load statistics to neighbor nodes
- Load tokens are exchanged between nodes to agree on load balancing
- Load is transferred after token exchange
- <u>Can balance load and preserve topology-aware mapping</u>

Summary

- Topology-aware mapping, load balancing important for efficient resource utilization
- Load balancing that preserves mapping efficient way to have best of both worlds
- Charm++ load balancing framework
  - Fast, scalable load balancing
  - Upcoming support for heterogeneous load balancing
  - AMPI applications automatically benefit

Vector Load Balancing



NullLB GreedyLB MultiGreedyLB GreedyNormLB