Topology Aware Mapping and Load Balancing

Ronak Buch, Laxmikant Kale

rabuch2@illinois.edu

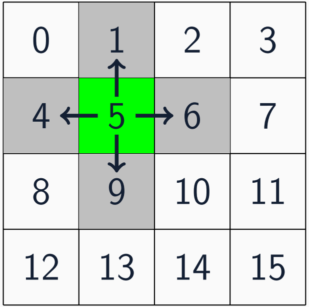
Blue Waters Symposium - May 19, 2017

Topology Aware Mapping

Topology Aware Task Placement

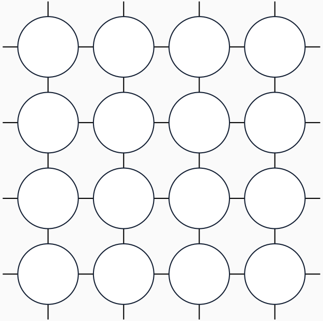
- What is topology aware task placement?
  - Place tasks so that communicating tasks are closer in the machine topology: minimize off-node traffic and link contention
  - Not the same as simple rank reordering
  - Takes into account the actual nodes in the job and distance in the network between tasks
- Why topology aware task placement?
  - Execution time can improve significantly ($> 2x$) for large jobs in some applications compared to default BW mapping

Mapping Example - 2D Application on 2D Torus



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Application Domain

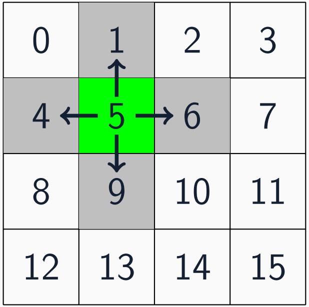


Machine Topology

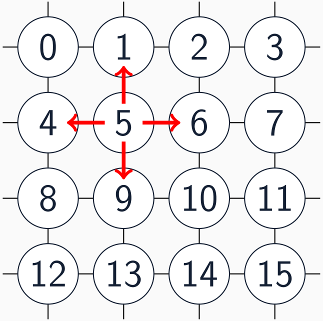
2D application grid, ranks ordered by row

Neighbor communication, each cell an MPI task

Mapping Example - 2D Application on 2D Torus



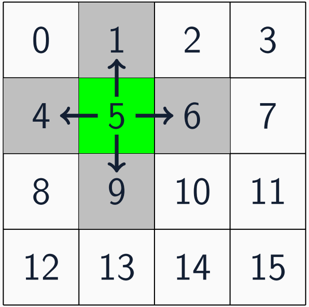
Application Domain



Machine Topology

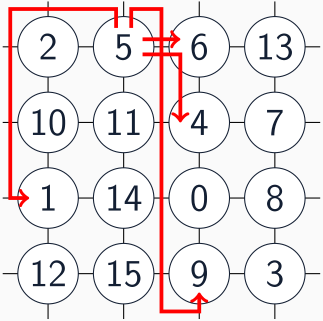
If same shape, preserve structure

Mapping Example - 2D Application on 2D Torus



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Application Domain



2	5	6	13
10	11	4	7
1	14	0	8
12	15	9	3

Machine Topology

Bad placement → Longer paths → Higher link utilization → Link contention → Longer communication time

2D 10x10 Virtual Grid → 2D 10x10 Torus

- Near-neighbor communication, each task sends unit load to each neighbor
- Random shortest path for each pair of communicating ranks

Layout	Avg Hops	Max Hops	Avg Link Load	Max Link Load
Row Order	1	1	1	1
Random	5.15	9	5.15	14

2D 10x10 Virtual Grid → 2D 5x20 Torus

Layout	Avg Hops	Max Hops	Avg Link Load	Max Link Load
Row Order	6.16	12	6.16	17
Random	6.15	11	6.15	25
Optimal[1]	2.15	11	2.15	7

[1] Really, this is “near optimal”

How Important is Task Placement for my Application?

- Application communication patterns (near-neighbor, collectives)
- Volume of communication
- Communication to computation ratio
- Allocation shape
- Job size: larger jobs have more communication, and potentially larger distances between tasks
- Some applications (e.g. MILC) can get more than 2*x*-3*x* speedup in execution time on large jobs

How to do Mapping?

- Perhaps easiest solution is to use an automatic tool and test to see if there are actual improvements. Worst case: no improvement
  - Remember that it might only improve in certain job sizes and job shapes
- Analyze the communication behavior of the application
  - Knowledge of application: computation to communication ratio, volume of point to point traffic, collectives pattern
  - If user doesn't have this knowledge → Profiling tools

Profiling Tools

- CrayPat
  - Large suite of performance analysis tools, including tracing
- mpiP
  - Link-time library: no need to recompile application
  - Presents statistical information on MPI calls
- TopoProfiler
  - Link-time library
  - Collects information on MPI calls
  - Includes topological information (hop-bytes, rank placement in 3D torus) from the run
  - Extract the communication graph (messages and bytes communicated between MPI tasks)

Mapping Tools

- In the application: *hard*
  - Can get topological info at runtime, but have to move manually
- CrayPat's `grid_order` tool
  - Can place communicating tasks in same node, but doesn't place communicating nodes nearby, doesn't deal with irregular patterns
- Topaware
  - Needs to know application grid shape, assumes near-neighbor comm pattern in grid, only works in specific geometries
- genM
  - Works with any MPI application, system topology
  - No source code changes
  - Works with any communication pattern, including irregular

genM

- Communication graph as input
- Goal: minimize average (per-rank) hopbytes
    - $hopbytes_a = \sum_n bytes(a, n) \cdot distance(a, n)$
    - Keep communicating tasks close, weighted by bytes
- NP-hard problem → multiple trials of random greedy search
- Runs separate instance of algorithm on every processor

Running genM

Inside batch job (comm_graph.txt from TopoProfiler):

```
aprun -n NUM_PROCESSORS ./mapper comm_graph.txt
export MPICH_RANK_REORDER_METHOD=3
aprun -n NUM_PROCESSORS ./program args
```

`MPICH_RANK_REORDER_METHOD = 3` tells aprun to use a custom rank order from `MPICH_RANK_ORDER` in the current directory.

Example: MILC

- Lattice QCD Application
- 4D lattice (3 spatial dimensions + time)
- Extremely sensitive to placement
- Near-neighbor communication pattern

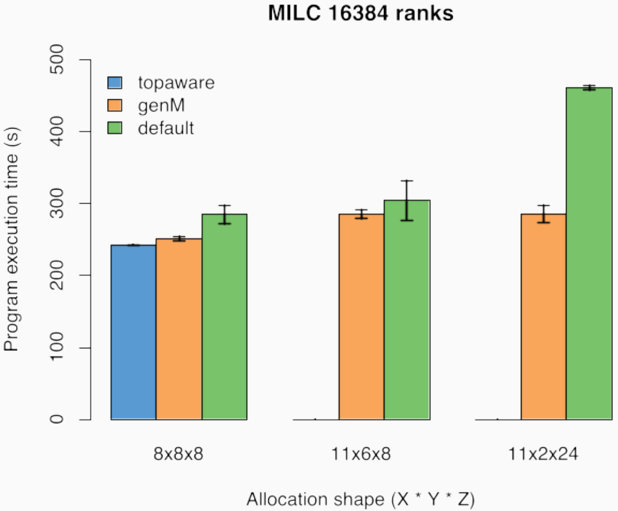
MILC Profiling - Default Mapping, 8x8x8, 16K Tasks

```
Total Execution Time:  342.8184
Average Hopbytes (rank average):  2.6368
Min Hopbytes:  2.2034 on Rank:  6624
Max Hopbytes:  3.7892 on Rank:  15360
Average Idle Time (rank average):  138.7806
Min Idle Time:  113.4179 on Rank:  4596
Max Idle Time:  166.2557 on Rank:  16201
Average P2P Operation Time (rank average):  99.6960
Min P2P Operation Time:  72.5383 on Rank:  3989
Max P2P Operation Time:  127.9738 on Rank:  14696
```

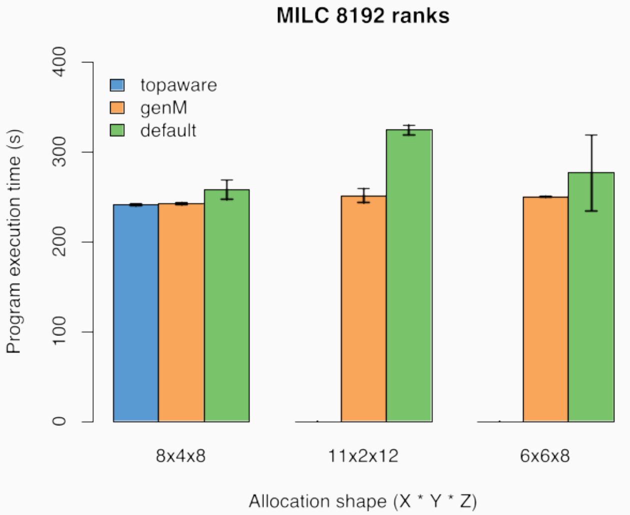
MILC Profiling - Default Mapping, 8x8x8, 16K Tasks

```
Total Execution Time:  100%
Average Hopbytes (rank average):  2.6368
Min Hopbytes:  2.2034 on Rank:  6624
Max Hopbytes:  3.7892 on Rank:  15360
Average Idle Time (rank average):   41%
Min Idle Time:  113.4179 on Rank:  4596
Max Idle Time:  166.2557 on Rank:  16201
Average P2P Operation Time (rank average):   29%
Min P2P Operation Time:  72.5383 on Rank:  3989
Max P2P Operation Time:  127.9738 on Rank:  14696
```

MILC Mapping Results on BW



MILC Mapping Results on BW - con't



Load Balancing

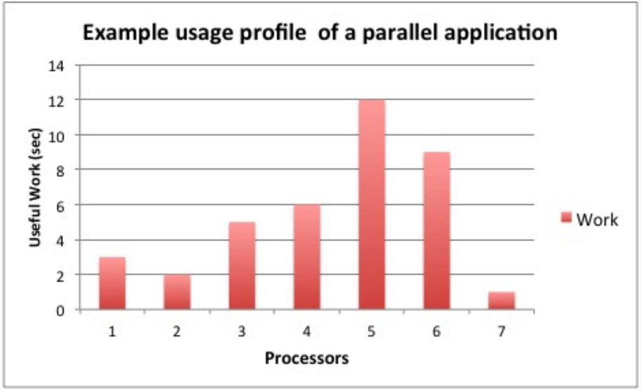
Background

Load balance is integral to achieving scalability and performance.

Hard to do, in general:

- Modern machines too complex to manually manage locations
- Multi-resolution - variation in task granularity
- Multi-module - loosely connected diverse tasks
- Dynamic/adaptive applications have high variation

What is Load Imbalance?



- Application completes only when all processors finish
- Different processors get variable amounts of work
- Resources wasted in waiting

Golden Rule of Load Balancing

Fallacy: objective of load balancing is to minimize variance in load across processors

Example:

- 50,000 tasks of equal size, 500 processors:
    - A: All processors get 99, except last 5 gets $100 + 99 = 199$
    - OR, B: All processors have 101, except last 5 get 1

Identical variance, but situation A is much worse!

Golden Rule: It is okay if a few processors are idle, but avoid having processors that are overloaded with work.

How to Diagnose Load Imbalance

- Often hidden in statements such as:
    - High idle time
    - Very high synchronization overhead
    - Time spent in collectives (processors waiting for a reduction, for example)
- Count total amount of computation (ops/flops) per processor
    - In each phase!
    - Balance may change from phase to phase

How do we Balance Load?

In **Charm**++:

- Objects are migratable
- Programs are overdecomposed, so there are many objects
- Runtime measures PE and object load
- Balancing algorithms compute new mappings
- Runtime moves objects according to mapping

Charm++ Balancing Results

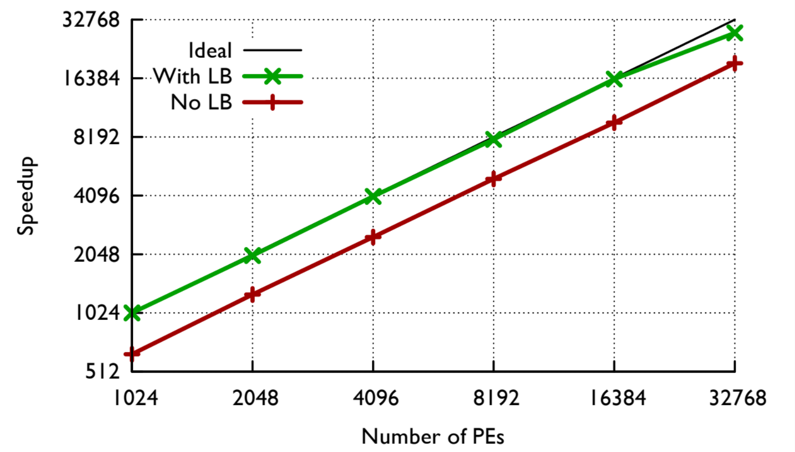


Figure: Charm++ LeanMD Results

What about MPI?

In **MPI**:

- Ranks aren't generally migratable
- Programs aren't overdecomposed, usually ranks $\leq$ cores
- No automatic load measurement

But, if application developers can handle these things, the same algorithms can be used to compute new mappings.

Load Balancing Library

We have extracted load balancing strategies from Charm++ and released them as a standalone library.

Can easily be linked to an MPI program and be used to make object placement decisions.

Application calls into library with object load and location data, library returns new mapping given by selected strategy.

No mechanism for migration is included; application is responsible for that.

Using the Load Balancing Library

To use the library, call:

```
void CharmLB_Assign_Objs(
  LBData& interface,
  STRATEGY strategy
)
```

`interface` is a structure that contains the processor and load data, the new mapping is stored inside when this call returns.

What if Manually Migrating is Impossible?

If an application can't be easily decomposed into migratable data segments, it can be hard to apply the library.

However, **Adaptive MPI (AMPI)** may help.

AMPI is an MPI implementation on top of Charm++ which supports load balancing by migrating entire ranks without requiring code modifications.

Some limitations exist (notably global variables), but has been shown to be effective.

AMPI Balancing Results

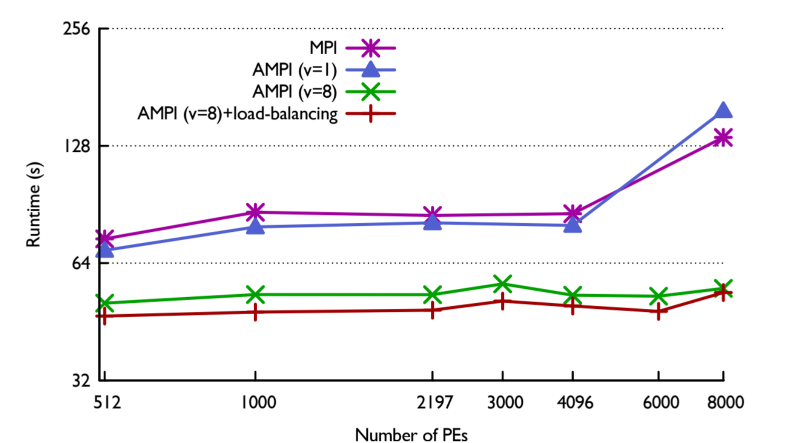


Figure: LULESH AMPI Results

What about Accelerators?

Charm++ load balancers have a long history in balancing work across CPUs.

Balancing work across CPUs and GPUs is more complicated:

- Some tasks are GPU only
- Some tasks have both CPU and GPU versions
- Performance for same task varies based on hardware

`vector` load balancing and `accel` load balancing are results of new research to address these problems.

Vector Load Balancing

Balances programs with CPU only and GPU only work.

The Charm++ runtime measures load data for each hardware target. Thus, an object has a two-dimensional load vector, $< cpuload, gpuload >$.

The work is then distributed to minimize the maximum load in both dimensions as much as possible.

Currently working on adding additional dimensions, such as memory load.

Synthetic Vector Load Balancing Results

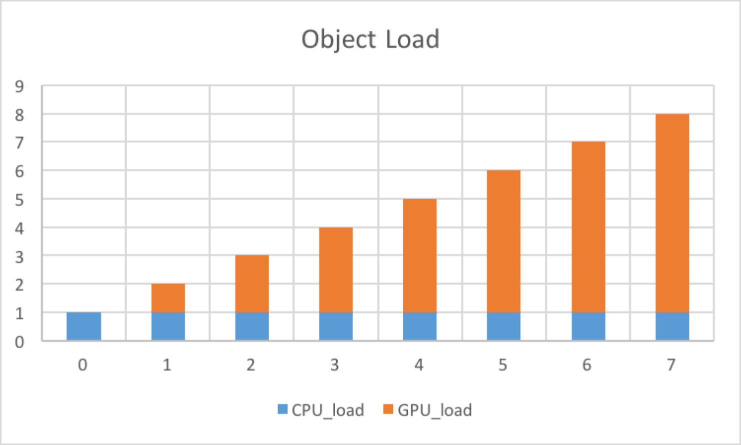


Figure: Object Loads

Synthetic Vector Load Balancing Results (con't)

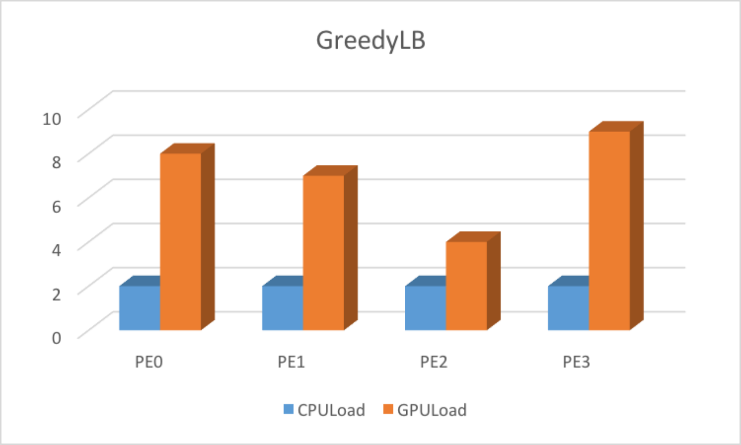


Figure: Regular `GreedyLB` Results

Synthetic Vector Load Balancing Results (con't)

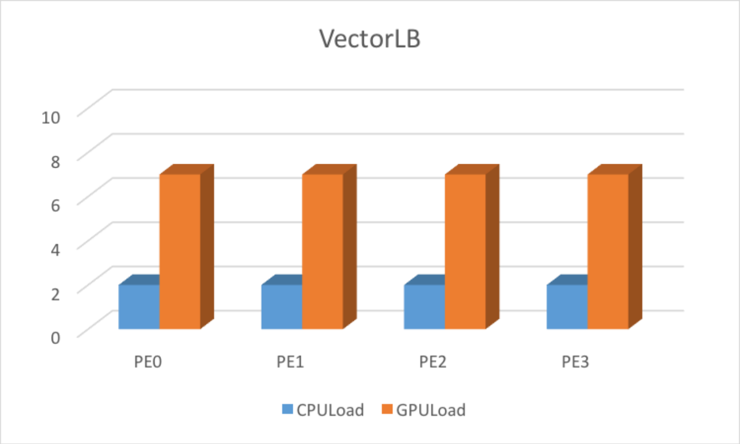


Figure: `VectorLB` Results

Accel Load Balancing

Some programs have fungible work that can be retargeted to different hardware devices.

Different from `vector` since static division of labor no longer exists.

Now, we must find the optimal "split" point to equally divide the tasks between the CPU and GPU.

To do this, the runtime dynamically assigns work to different hardware targets based on some strategy. Users can specify static division percentages or dynamic strategies where the runtime searches for the optimal split.

Accel Load Balancing Results

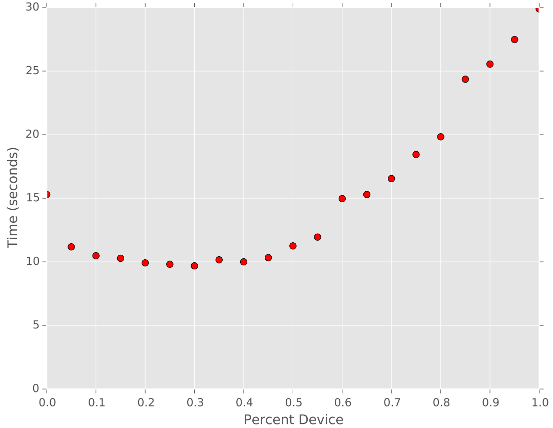


Figure: Accel Stencil3d Load Balancing Results

Accel Load Balancing Results (con't)

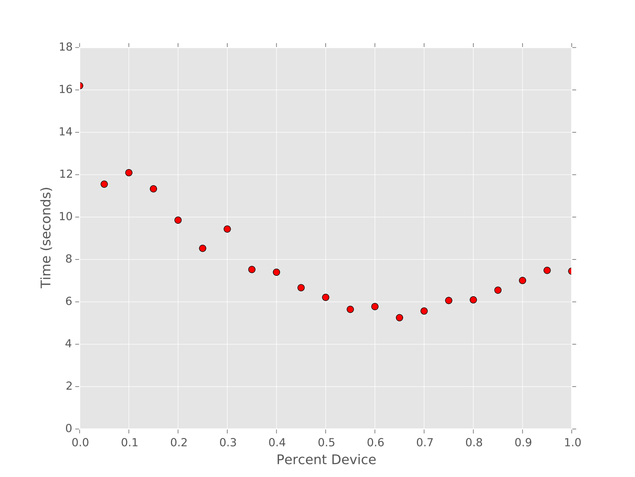


Figure: Accel MD Load Balancing Results

Beyond the Scope of This Talk

- Multi-strategy load balancing
- Custom load metrics
- Details of heterogeneous load balancing
- Automatic load balance frequency and strategy selection
- Writing your own load balancing strategy

If interested: talk to us!

Questions?

rabuch2@illinois.edu

charm@cs.illinois.edu

https://charm.cs.illinois.edu