Vector Load Balancing For High-Performance Parallel Applications

Ronak Buch

Preliminary Exam
Department of Computer Science

University of Illinois at Urbana-Champaign

November 11, 2021

Committee

- Luke Olson (UIUC)
- David Padua (UIUC)
- Antonio Peña (BSC)
- **Laxmikant Kale** (UIUC)

Outline

Motivation & Background

Preliminary Work

Planned Work

Conclusion

High Performance Computing

- By definition, HPC requires *high performance*
- Massive, powerful, systems, but hard to use
- *Load balancing* distributes work across system to optimize performance
- Key for all but very regular, static applications

Dynamic Load Balancing

- Adapt to varying loads at runtime, no *a priori* knowledge needed
- Introspectively examine and modify system state
- Enabled by migratable objects, runtime load measurement

Dynamic Load Balancing

- Adapt to varying loads at runtime, no *a priori* knowledge needed
- Introspectively examine and modify system state
- Enabled by migratable objects, runtime load measurement

Measure-Migrate Cycle

What do we measure?

What is Load?

- *Load* is a proxy value standing in for *performance*
  - Metric measuring utilization of a resource over a period

What is Load?

- *Load* is a proxy value standing in for *performance*
  - Metric measuring utilization of a resource over a period
- Real goal is not to "balance load", but minimize execution time

What is Load?

- *Load* is a proxy value standing in for *performance*
    - Metric measuring utilization of a resource over a period
- Real goal is not to “balance load”, but minimize execution time
- Traditionally, balancing CPU time has been sufficient for high performance

What is Load?

- *Load* is a proxy value standing in for *performance*
  - Metric measuring utilization of a resource over a period
- Real goal is not to "balance load", but minimize execution time
- Traditionally, balancing CPU time has been sufficient for high performance

Hypothesis

Can we do better by considering a richer set of metrics?

Diets

How do we design a healthy diet?

Diets

How do we design a healthy diet?

- Calories?

Diets

How do we design a healthy diet?

- Calories? X

Diets

How do we design a healthy diet?

- Calories? X
- Carbohydrates?

Diets

How do we design a healthy diet?

- Calories? X
- Carbohydrates? X

Diets

How do we design a healthy diet?

- Calories? X
- Carbohydrates? X
- Protein?

Diets

How do we design a healthy diet?

- Calories? X
- Carbohydrates? X
- Protein? X

Diets

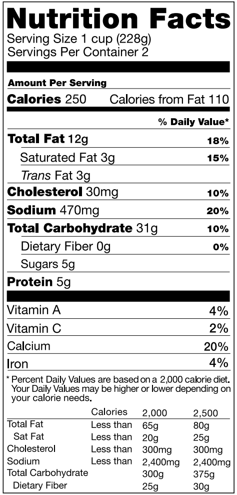
How do we design a healthy diet?

- Calories? X
- Carbohydrates? X
- Protein? X

Solution

All of the above and more. Need a *balanced* diet.

Balanced Diets



Nutrition Facts

Serving Size 1 cup (228g)
Servings Per Container 2

Amount Per Serving

Calories 250 Calories from Fat 110

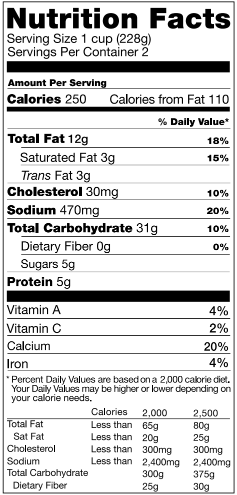
	% Daily Value*
Total Fat 12g	**18%**
Saturated Fat 3g	**15%**
Trans Fat 3g	
Cholesterol 30mg	**10%**
Sodium 470mg	**20%**
Total Carbohydrate 31g	**10%**
Dietary Fiber 0g	**0%**
Sugars 5g	
Protein 5g	

Vitamin A	4%
Vitamin C	2%
Calcium	20%
Iron	4%

* Percent Daily Values are based on a 2,000 calorie diet. Your Daily Values may be higher or lower depending on your calorie needs.

	Calories	2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

Balanced Diets



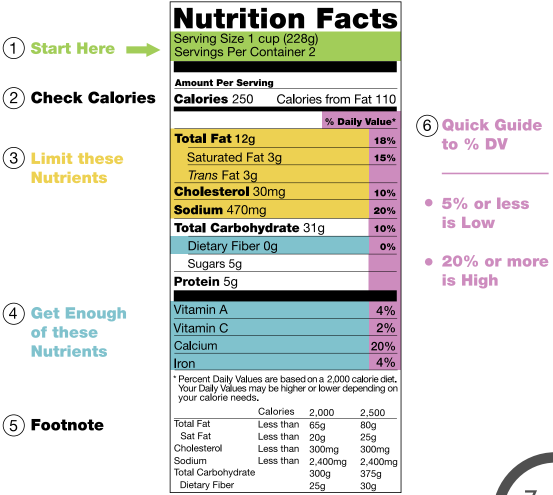
Nutrition Facts
Serving Size 1 cup (228g)
Servings Per Container 2

Amount Per Serving

Calories 250	Calories from Fat 110
	% Daily Value*
Total Fat 12g	**18%**
Saturated Fat 3g	**15%**
Trans Fat 3g	
Cholesterol 30mg	**10%**
Sodium 470mg	**20%**
Total Carbohydrate 31g	**10%**
Dietary Fiber 0g	**0%**
Sugars 5g	
Protein 5g	
Vitamin A	4%
Vitamin C	2%
Calcium	20%
Iron	4%

* Percent Daily Values are based on a 2,000 calorie diet. Your Daily Values may be higher or lower depending on your calorie needs.

	Calories	2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g



1 Start Here

2 Check Calories

3 Limit these Nutrients

4 Get Enough of these Nutrients

5 Footnote

6 Quick Guide to % DV

- 5% or less is Low
- 20% or more is High

Nutrition Facts
Serving Size 1 cup (228g)
Servings Per Container 2

Amount Per Serving

Calories 250	Calories from Fat 110
	% Daily Value*
Total Fat 12g	**18%**
Saturated Fat 3g	**15%**
Trans Fat 3g	
Cholesterol 30mg	**10%**
Sodium 470mg	**20%**
Total Carbohydrate 31g	**10%**
Dietary Fiber 0g	**0%**
Sugars 5g	
Protein 5g	
Vitamin A	4%
Vitamin C	2%
Calcium	20%
Iron	4%

* Percent Daily Values are based on a 2,000 calorie diet. Your Daily Values may be higher or lower depending on your calorie needs.

	Calories	2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

Balanced Diets

Human nutrition is *complex*!

- Must consider several different factors simultaneously
- All targets should be hit daily

Balanced Diets

Human nutrition is *complex*!

- Must consider several different factors simultaneously
- All targets should be hit daily

Parallel

Software performance is similarly *complex*!

Software Performance

- Must consider several different factors simultaneously
  - Local: compute, memory, cache, IO
  - Distributed: network bandwidth, latency, hotspots
  - Schedule: data dependencies, barriers
- Bottleneck may vary depending on system, application, dataset, configuration, state
- Further complicated by hardware trends, advanced simulation techniques
  - Heterogeneous computing, network vs. compute perf
  - Multi-*x* methods, *in-situ* analysis, etc.

Accelerators - Top500

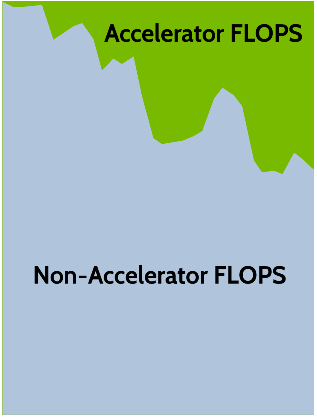


Figure: % of Top500 FLOPS by Source

- First appeared in 2006, now provide 41% of FLOPS of Top500
- Applications becoming increasingly heterogeneous

Multi-*x* Methods

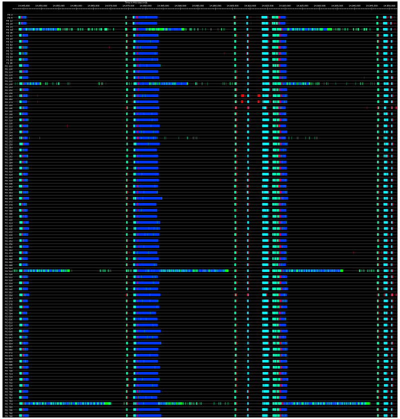


Figure: Timeline of ChaNGa Exection

- Applications with multi-*x* methods generally have distinct phases
- Load varies between phases, chunks are not always active, etc.

Takeaways

- A single metric cannot capture the intricacies of performance
- Modern systems and algorithms exacerbate the issue
- Load balancing loses efficacy when load is not representative of performance

Takeaways

- A single metric cannot capture the intricacies of performance
- Modern systems and algorithms exacerbate the issue
- Load balancing loses efficacy when load is not representative of performance

Solution

Provide a richer set of data to load balancers!

Outline

Motivation & Background

Preliminary Work

Planned Work

Conclusion

Vector Load Balancing

- Rather than being a single scalar value, *load* is a vector of multiple values
- Composed of things like:
  - Various resource measurements, e.g. CPU/GPU/network/memory/IO
  - Timings of separate phases of an iteration
  - Application specific parameters, e.g. number of particles

Charm++

- Charm++ is a task-based parallel programming framework
- Supports load balancing via three features:
  - Active Runtime - Allows dynamic arbitrary measurements
  - Migratable Objects - Allows work units to be moved
  - Overdecomposition - Provides granularity for balancing
- Rewrote LB infrastructure to support vector loads, more flexible LB paradigms
- Used for all thesis work, but not Charm++ specific (i.e. vector LB support in AMPI)

Measuring Vector Loads

- Features and APIs to add vector load measurement have been added to Charm++
  - Application can add calls to indicate phase boundaries, RTS will automatically track per-phase load
  - Runtime flags to automatically add communication load (msgs, bytes sent)
  - User can specify load vector explicitly

Traditional Objective

In traditional LB, goal is to find mapping that minimizes the maximum load on a processor.

Traditional Objective

In traditional LB, goal is to find mapping that minimizes the maximum load on a processor.

$$\arg\min_{M} \max_{p \in P} \left(\sum_{\forall o \in O : M(o) = p} l_o \right)$$

O - set of objects

P - set of processors

l_o - load of object o

$M : O \rightarrow P$ - function mapping objects to processors

Vector Objective

In vector LB, goal is to find mapping that minimizes the sum of the maximum load on a processor across dimensions.

Vector Objective

In vector LB, goal is to find mapping that minimizes the sum of the maximum load on a processor across dimensions. *(Other formulations are possible!)*

Vector Objective

In vector LB, goal is to find mapping that minimizes the sum of the maximum load on a processor across dimensions. *(Other formulations are possible!)*

$$\arg\min_{M} \boxed{\sum_{0\leq i<d}} \left(\max_{p\in P}\left(\sum_{\forall o\in O:M(o)=p} \boxed{(\overrightarrow{l_o})_i}\right)\right)$$

d - number of dimensions

$\overrightarrow{l_o}$ - load vector of object o

$(\overrightarrow{l_o})_i$ - ith component of o's load vector

Vector Balancing

- Dimensionality makes vector load balancing computationally difficult
- Objects can no longer be totally ordered
- Want to minimize the “maximum” over all dimensions simultaneously
    - Single variable optimization is now multivariable
- New LB strategies are needed

Vector Strategies

Requirements:

- Ingest and use multidimensional loads
- Some means of assessing and comparing quality of mappings
- Result correlated with objective function

Naïve Vector Greedy

- Simple extension of standard scalar Greedy strategy
- Use maximum of object load and processor load as representative
- Converts vector problem into scalar; can be used with any scalar strategy
- Very fast and surprisingly effective for some applications, but poor overall, oblivious to dimensionality

MultiGreedy

- Extends Greedy to consider each dimension separately
  - Stores procs in d min-heaps
  - Stores objects in one max-heap
- Finds the object with maximum load across all dimensions and places it on PE with minimum load in that dimension
- Only works well when object has load in only one dimension, e.g. $(0, 0, 0, l, 0)$

Strategy Issues

- Solve multidimensional problem via reducing to one dimension
- Missing a notion of simultaneity
- For more realistic cases, have to consider vector holistically

Holistic Vector Strategies

- Find object with maximum norm and place on PE with minimum norm after placement
  - Works well, but computationally expensive
  - PE "weight" varies with object, i.e. $\|(2,0)\|_2 < \|(0,3)\|_2$, but when adding object with $(3,0)$, $\|(5,0)\|_2 > \|(3,3)\|_2$
- Place objects based on largest load in vector as before, then refine partitions to improve balance (used by METIS)

NormLB - Exhaustive

- Initial implementation orders objects by norm and then does exhaustive search across all PEs for placement
  - Quality is exactly as desired
  - Performance is very poor ($\Theta(p \cdot o)$)

Method	Makespan	Strategy Time (s)
Greedy	1965.83	0.32
Norm	1674.86	22.72

Table: Greedy vs Norm (*1e4 PEs, 1e6 objs*) (2d)

NormLB - k-d

- To improve performance, we use a k-d tree to guide PE selection
    - Arbitrary dimension space partitioning tree
    - Allows PE search to be bounded as candidates are found
- k-d works well for searching in static point set, but here, tree updated after every assignment
    - Costly update operations
    - Pattern of updates often results in unbalanced tree
- Can be worse than the naïve exhaustive version!

Random Relaxed k-d

- Random Relaxed k-d trees help solve these problems; two key differences from standard k-d:

Relaxed Instead of cycling through discriminants, $1, 2, \ldots, k, 1, \ldots$, each node stores arbitrary discriminant $j \in \{1, 2, \ldots, k\}$

Random Discriminant is uniformly randomly chosen and each insertion has some probability of becoming the root, or root of subtree, . . .

Random Relaxed k-d

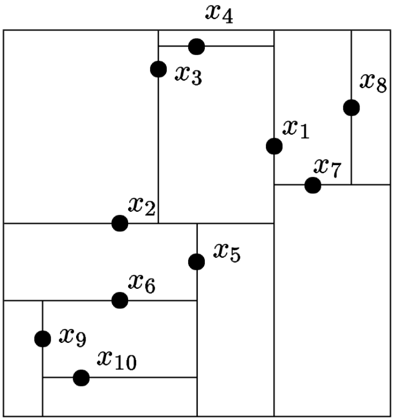


Figure: k-d

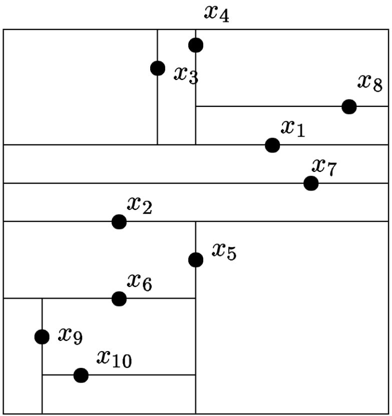


Figure: rk-d

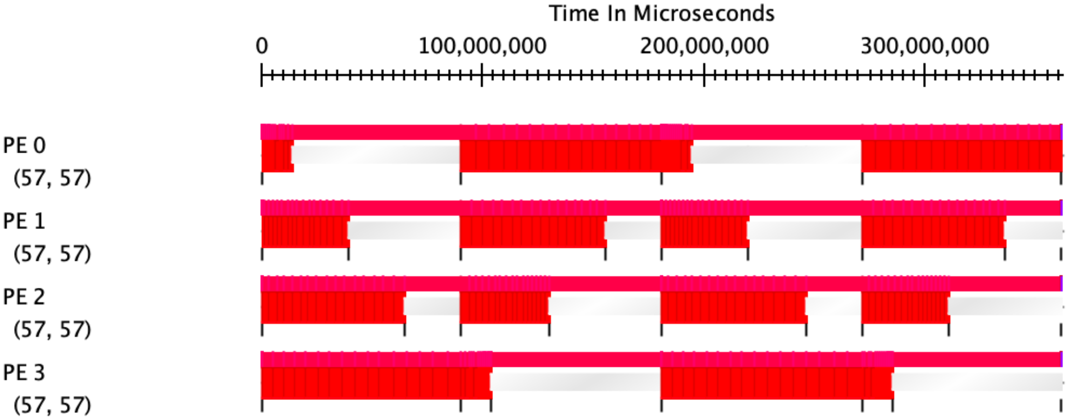
NormLB - rk-d

- These low-cost arbitrary updates and stochastic balancing improve LB (all provide same results)

Method	Strategy Time (s)	
	1e4 PEs, 1e5 objs	*1e4 PEs, 1e6 objs*
Exhaustive	2.18	21.54
Standard k-d	0.93	27.55
Relaxed k-d	0.57	7.96

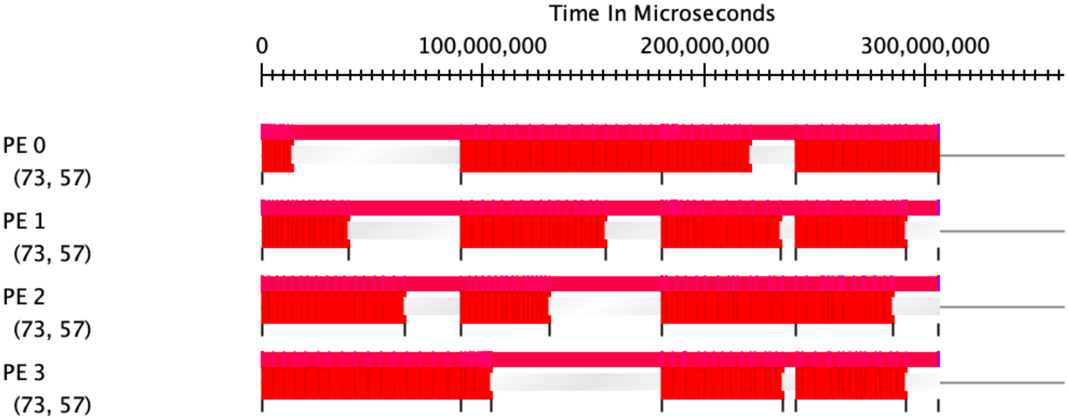
Table: Performance of Norm-Based Strategies (2d)

Vector LB Performance



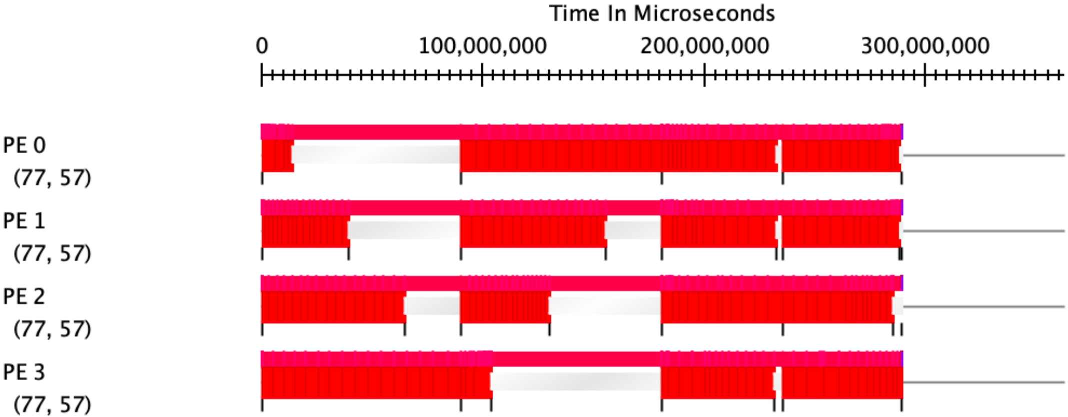
No Load Balancing

Vector LB Performance



Regular Load Balancing

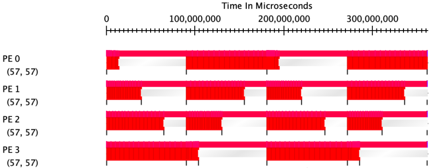
Vector LB Performance



Vector Load Balancing

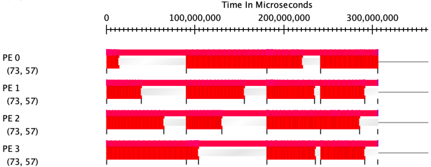
Vector LB Performance

LB Off



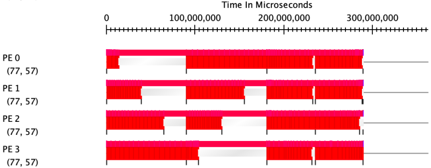
Phase Unaware

(1.44x speedup)



Phase Aware

(1.67x speedup)



METISLB

- METIS is a graph partitioning package
  - Provide graph with weighted edges and vertices, returns partitions with load within some tolerance while minimizing weight of edge cut
  - Supports vertex loads for vectors
- Can use for vector LB by providing fully disconnected graph
- Uses multi-objective bipartitioning and coarsening/uncoarsening for performance

METISLB Performance

- Performance is good, but quality drops off at larger scales for unknown reason

	1e3 PEs, 1e5 objs		*1e4 PEs, 1e6 objs*	
Method	Makespan	Strategy Time (s)	Makespan	Strategy Time (s)
METIS	1674.46	0.1656	2303.32	3.1851
k-d	1678.86	0.2228	1674.99	7.4073

Table: Comparison of METIS and k-d Strategies (2d)

Outline

Motivation & Background

Preliminary Work

Planned Work

Conclusion

Locality in LB

- Vector loads give performance insight with increased nuance and detail
- However, performance may also vary based on the location of objects
  - The distance between communicating objects changes latency, load on links, routers
  - Balanced via graph partitioners, geometric strategies
- Currently captured via RTS communication graph or application provided positions
  - For vector: first application positions, then comm graph

LB Position API

- Geometric strategies use *ad hoc* data passing
  - e.g. ChaNGa uses `LBRegisterObjUserData` to pass in `void*`
  - Each application needs its own custom LB strategies
- Adding standardized LB position API to Charm++
  - `setObjPosition(const vector<LBRealType>& pos)`
  - Allows positions of arbitrary dimension
- Allows for generic, application agnostic strategies
- Fully implemented, currently testing with ChaNGa and other applications

Geometric Vector Strategies

- Currently using orthogonal recursive bisection with position API
- In scalar world, find split coordinate that minimizes differences in load between both halves
- In vector world, things are more complicated
  - Each dimension may have a different split coordinate
  - Select by taking average, minimizing square difference, etc.
  - Rather than splitting at a single coordinate, allow objects in some neighborhood to go to either half
  - Topic of active experimentation

Communication Vector Strategies

- Already have METIS vector strategy, but no edges
- Charm++ automatically measures comm graph, uses METIS for scalar comm-aware LB
- Will combine these two into one, providing comm-aware vector LB
- Appropriate for same applications that use geometric strategies
  - Heavy communication
  - Clustered datasets

Refinement Vector Strategies

- Current strategies don't take existing mapping into account
  - Migration cost can be high
- Most applications do not need global partitioning at every step
  - Principle of persistance
- Will develop refinement based strategies to address this case
  - Find overloaded processors, move some of their objects away, e.g. diffusion

Distributed Vector Strategies

- All current vector strategies are hierarchical
- Performance bottleneck for large jobs
- Charm++ has distributed strategies for scalability
    - Load statistics are not gathered at one location
    - Processors negotiate load interchange with neighbors
- Will add distributed vector diffusion strategy
    - Diffuse load vector with neighbors
    - Approximate subset-sum to find objects to transfer

Performance

- Performance can still be an issue
  - Only worthwhile if (time saved by LB) < (time to perform LB)
- Use dimensionality reduction to simplify for large d
- Current bounded versions of Norm LBs tradeoff quality and performance
  - More intelligent sampling, better heuristics
- New strategies that further compress the search space
  - e.g. search only within Pareto frontier
- Distributed strategies, parallelize existing

Vector vs Sequence of Scalar

- Can good results be achieved by running scalar LBs over different dimensions in sequence?
  - Either during one LB invocation or at every phase boundary
  - ChaNGa does this now across time, ORB at big step, refine at small step
- Missing holistic awareness of proper vector strategy
- May be sufficient for some applications

New Metrics

- Runtime currently measures phases, network load
  - User can also add values manually
- Measurement of new metrics will improve usability:
  - GPU - Experimental version uses CUPTI to get kernel times, how to attribute remains a question
  - Cache/Memory - PAPI counters to measure accesses

Constraint Based Balancing

- Objective function for all LBs so far has been minimizing the maximum of everything
- Sometimes constraints are sufficient
  - Keep memory footprint below size of LLC or memory capacity of node
  - Ensure all processors within 10% of average number of bytes sent per processor
- Add way for user to specify dimensional objectives and constraint aware strategy

Applications

Astrophysics Multistepping causes distinct phases, individual steps also have phases

PDES CPU load is not a good measure of useful work because of rollbacks

MD Reliant on GPU performance

Object Cache Computationally light, data/comm heavy objects can overload cache/injection FIFO

Outline

Motivation & Background

Preliminary Work

Planned Work

Conclusion

Differentation

- Most work in vector balancing is theoretical
  - Proving bounds, sketching abstract algorithms
- Extant applied work is specific and isolated
  - Internal to specific application, only for graph partitioning

Differentation

- Most work in vector balancing is theoretical
  - Proving bounds, sketching abstract algorithms
- Extant applied work is specific and isolated
  - Internal to specific application, only for graph partitioning

Thesis Work

- Very applied, considers strategy performance
- Integrated, includes measurement and migration
- Offers many different strategies

Conclusions

- Complex, modern applications need sophisticated performance measurement
- Combining different metrics into a vector has been shown to improve the quality of load balancing
- New techniques must maintain communication locality to be useful for certain class of applications