

Vector Load Balancing for High-Performance Parallel Applications

Ronak Buch

January 13, 2023

Parallel Programming Laboratory
University of Illinois at Urbana-Champaign
Thesis Defense

Committee

- Professor **Laxmikant Kale**
- Professor Luke Olson
- Professor David Padua
- Dr. Antonio Peña (Barcelona Supercomputing Center)

Outline

Background and Motivation

Vector Load Balancing

Balancing Phase-Based Applications

Balancing Heterogeneous Applications

Balancing With Constraints

Scaling Vector Load Balancing

Future Directions & Conclusions

Background and Motivation

“Many hands make light work”

“Many hands make light work”

Caveats

“Many hands make light work”

Caveats

- Work needs to be *divisible*

“Many hands make light work”

Caveats

- Work needs to be *divisible* → Parallelism

“Many hands make light work”

Caveats

- Work needs to be *divisible* → Parallelism

“Many hands make light work”

Caveats

- Work needs to be *divisible* → Parallelism
- Division needs to be *equitable*

“Many hands make light work”

Caveats

- Work needs to be *divisible* → Parallelism
- Division needs to be *equitable* → Load balancing

“Many hands make light work”

Caveats

- Work needs to be *divisible* → Parallelism
- Division needs to be *equitable* → Load balancing

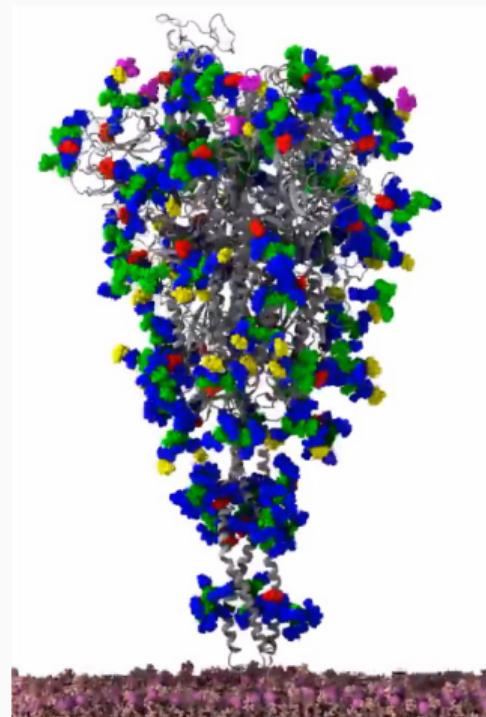
High Performance Computing



Supercomputers: many cores, many nodes, fast network,
large memory, large storage

Why High Performance Computing?

- Advancing science
 - No other way for astrophysics, molecular dynamics, ...
 - Safety for storm surge, epidemic simulation, ...
 - Cost for iterative design, material analysis, ...



Using High Performance Computing

- By definition, HPC requires *high performance*
- Massive, powerful, systems, but hard to use
 - How to keep all the resources occupied?

Using High Performance Computing

- By definition, HPC requires *high performance*
- Massive, powerful, systems, but hard to use
 - How to keep all the resources occupied?
- *Load balancing* distributes work across system to optimize performance

Using High Performance Computing

- By definition, HPC requires *high performance*
- Massive, powerful, systems, but hard to use
 - How to keep all the resources occupied?
- *Load balancing* distributes work across system to optimize performance

Aim for *equitable* division of labor

Dynamic Load Balancing

- Load may change as application runs

Dynamic Load Balancing

- Load may change as application runs
- Adaptively rearrange work amongst PEs to maximize performance as computation evolves
- Most loaded PE usually determines iteration time
- Necessary for scaling all but very regular, static applications

LB Prerequisites

Need three things to use LB:

LB Prerequisites

Need three things to use LB:

Load measurement Track execution of objects

LB Prerequisites

Need three things to use LB:

Load measurement Track execution of objects

Migrability Move objects to different PEs

LB Prerequisites

Need three things to use LB:

Load measurement Track execution of objects

Migrability Move objects to different PEs

Overdecomposition More objects than PEs

LB Prerequisites

Need three things to use LB:

Load measurement Track execution of objects

Migrability Move objects to different PEs

Overdecomposition More objects than PEs

Charm++ provides all three!

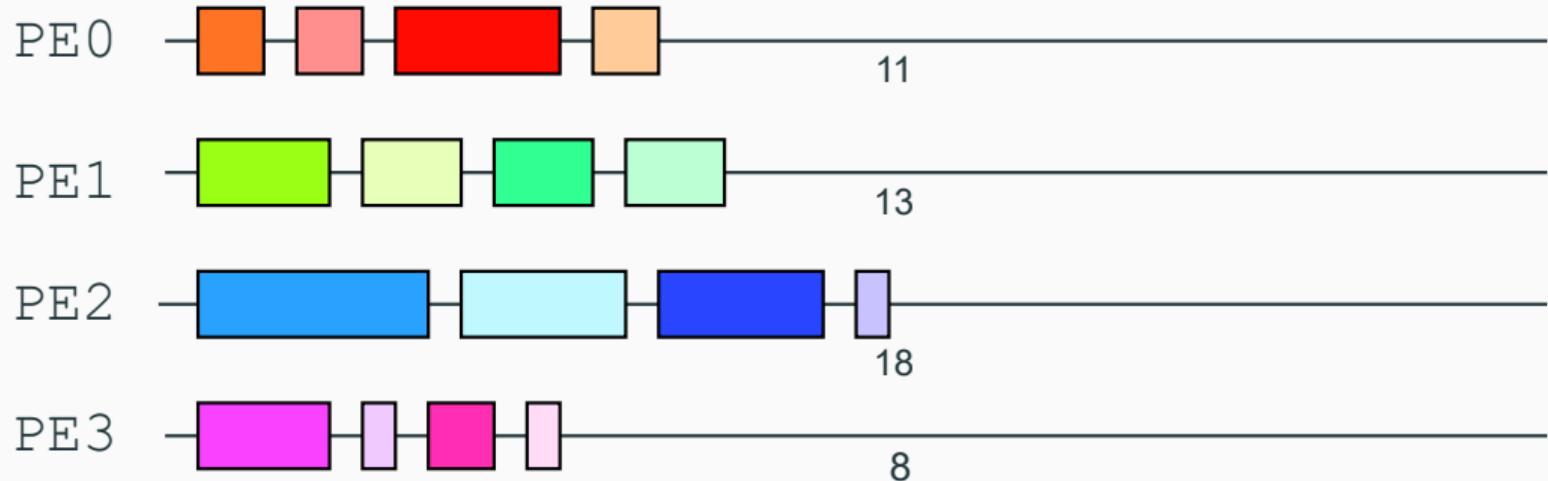
Charm++

- Charm++ is a task-based parallel programming framework for HPC
- Provides all requirements for LB:
 - Active runtime measures object loads
 - Serializable, migratable objects allows work to be moved
 - Supports many objects per PE for overdecomposition

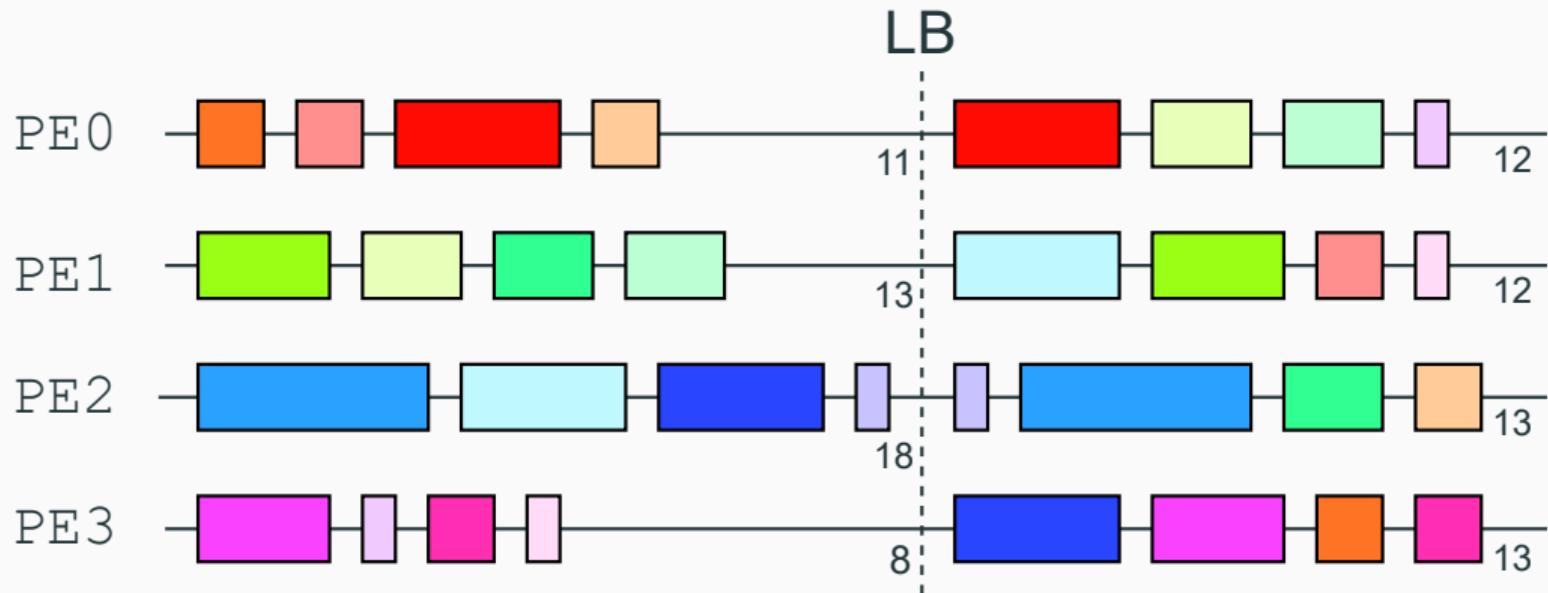
Charm++

- Charm++ is a task-based parallel programming framework for HPC
- Provides all requirements for LB:
 - Active runtime measures object loads
 - Serializable, migratable objects allows work to be moved
 - Supports many objects per PE for overdecomposition
- Used throughout, but work **not Charm++ specific**

Before LB



After LB



Scalar Load

- To measure load traditionally:

Scalar Load

- To measure load traditionally:
 - Start timer when object begins execution

Scalar Load

- To measure load traditionally:
 - Start timer when object begins execution
 - End timer when control returns to RTS

Scalar Load

- To measure load traditionally:
 - Start timer when object begins execution
 - End timer when control returns to RTS
 - Add elapsed time to object's load

Scalar Load

- To measure load traditionally:
 - Start timer when object begins execution
 - End timer when control returns to RTS
 - Add elapsed time to object's load
 - PE load = \sum resident objects' loads

Scalar Load

- To measure load traditionally:
 - Start timer when object begins execution
 - End timer when control returns to RTS
 - Add elapsed time to object's load
 - PE load = \sum resident objects' loads
- Only captures **how long** object spends active

Scalar Load

- To measure load traditionally:
 - Start timer when object begins execution
 - End timer when control returns to RTS
 - Add elapsed time to object's load
 - PE load = \sum resident objects' loads
- Only captures **how long** object spends active
 - CPU time alone does not determine performance

Scalar Load

- To measure load traditionally:
 - Start timer when object begins execution
 - End timer when control returns to RTS
 - Add elapsed time to object's load
 - PE load = \sum resident objects' loads
- Only captures **how long** object spends active
 - CPU time alone does not determine performance

Deficient for many classes of applications!

Scalar Load Deficiencies - Phase

- Phase-based applications

Scalar Load Deficiencies - Phase

- Phase-based applications
 - Iteration divided into several orthogonal phases

Scalar Load Deficiencies - Phase

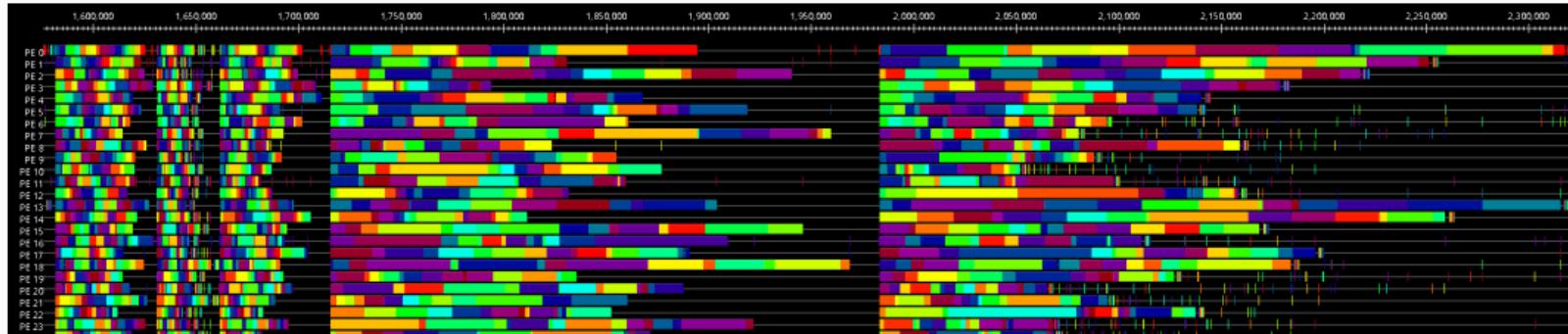
- Phase-based applications
 - Iteration divided into several orthogonal phases
 - Time spent in one phase not fungible with another

Scalar Load Deficiencies - Phase

- Phase-based applications
 - Iteration divided into several orthogonal phases
 - Time spent in one phase not fungible with another
 - **Scalar loses *execution phase* information**

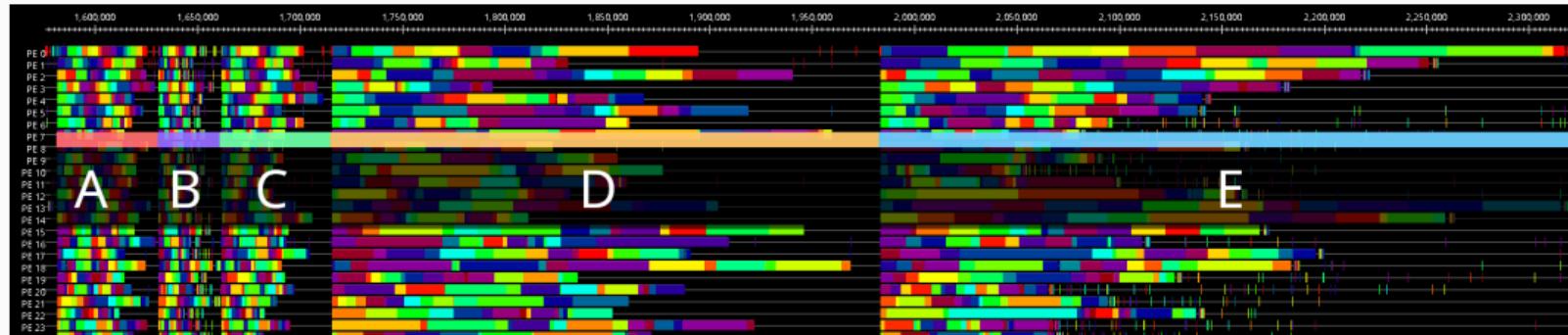
Scalar Load Deficiencies - Phase

- Phase-based applications
 - Iteration divided into several orthogonal phases
 - Time spent in one phase not fungible with another
 - **Scalar loses *execution phase* information**

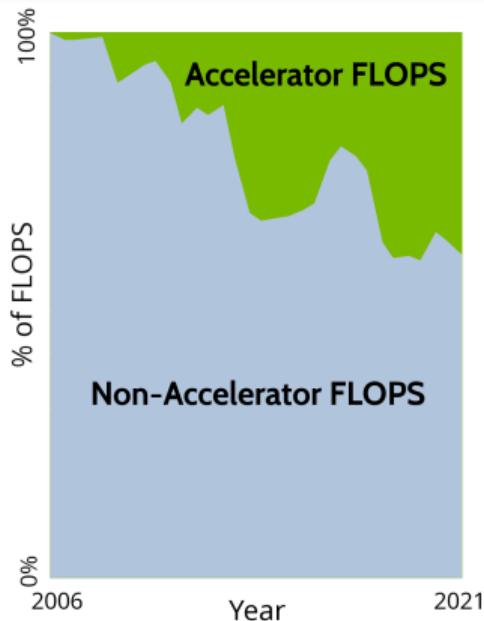


Scalar Load Deficiencies - Phase

- Phase-based applications
 - Iteration divided into several orthogonal phases
 - Time spent in one phase not fungible with another
 - *Scalar loses execution phase information*



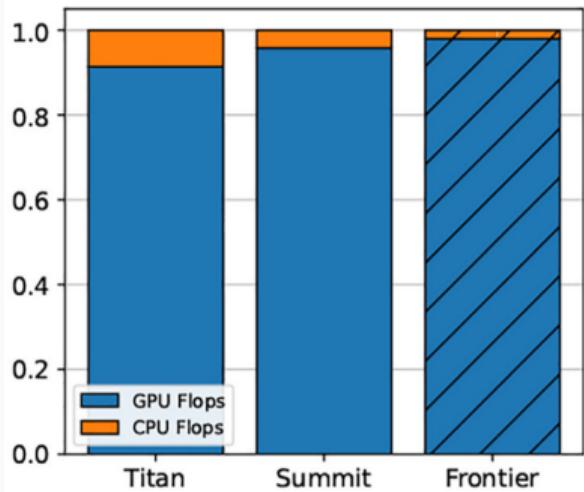
Scalar Load Deficiencies - Heterogeneity



- Accelerators are large source of FLOPS in HPC

Share of Top500 FLOPS
over Time

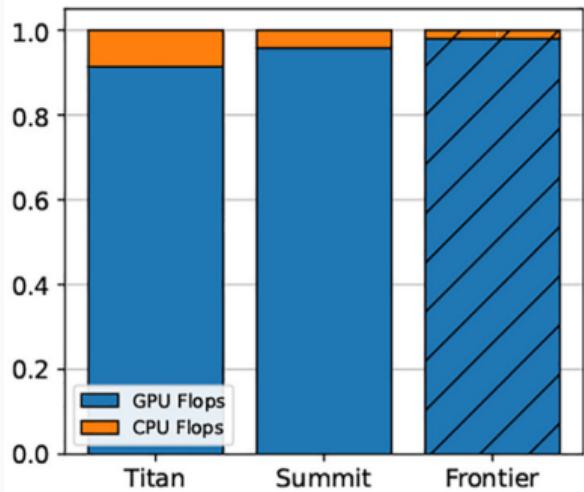
Scalar Load Deficiencies - Heterogeneity



- Accelerators are large source of FLOPS in HPC

Source of OLCF FLOPS

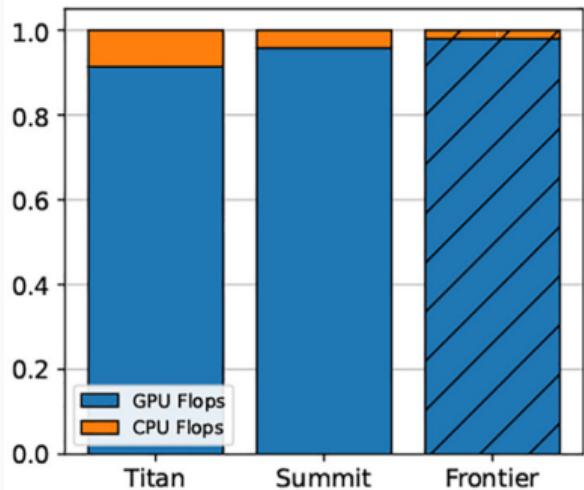
Scalar Load Deficiencies - Heterogeneity



- Accelerators are large source of FLOPS in HPC
- How to store load for GPU tasks?

Source of OLCF FLOPS

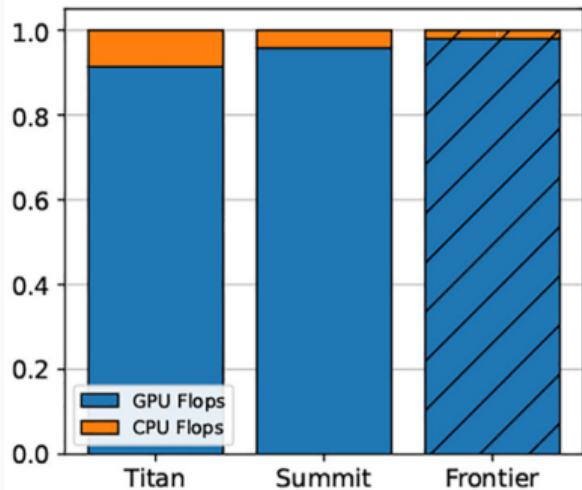
Scalar Load Deficiencies - Heterogeneity



- Accelerators are large source of FLOPS in HPC
- How to store load for GPU tasks?
 - CPU load \neq GPU load

Source of OLCF FLOPS

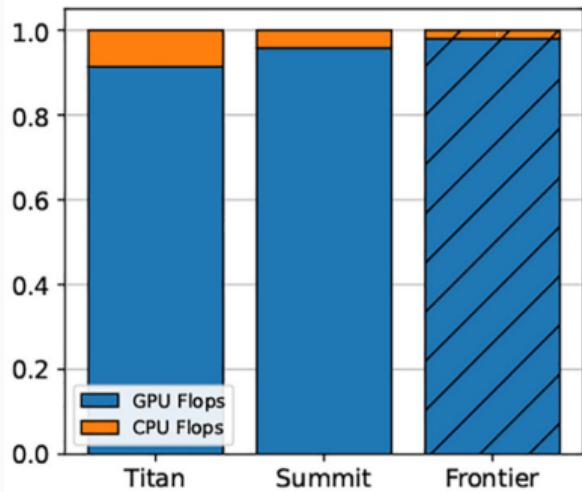
Scalar Load Deficiencies - Heterogeneity



Source of OLCF FLOPS

- Accelerators are large source of FLOPS in HPC
- How to store load for GPU tasks?
 - CPU load \neq GPU load
- How to balance tasks that can run on CPU and GPU?

Scalar Load Deficiencies - Heterogeneity



Source of OLCF FLOPS

- Accelerators are large source of FLOPS in HPC
- How to store load for GPU tasks?
 - CPU load \neq GPU load
- How to balance tasks that can run on CPU and GPU?
- Scalar cannot retain *processor type* of load

Scalar Load Deficiencies - Constraints

- Object mapping may be subject to some constraints
 - Required: Memory footprint, File descriptor count, ...
 - Preferred: Number of messages sent, Cache working set size, ...
- Want to balance load, but stay within constraints

Scalar Load Deficiencies - Constraints

- Object mapping may be subject to some constraints
 - Required: Memory footprint, File descriptor count, ...
 - Preferred: Number of messages sent, Cache working set size, ...
- Want to balance load, but stay within constraints
- Cannot encode *constraints* and load into scalar

What is Load?

- Load is a quantification of the characteristics of entities in an optimization problem

What is Load?

- Load is a quantification of the characteristics of entities in an optimization problem
- Load often consists of more than one metric:

What is Load?

- Load is a quantification of the characteristics of entities in an optimization problem
- Load often consists of more than one metric:
 - In shipping: volume, weight, shape, hazards, . . .

What is Load?

- Load is a quantification of the characteristics of entities in an optimization problem
- Load often consists of more than one metric:
 - In shipping: volume, weight, shape, hazards, ...
 - In nutrition: fat, protein, carbohydrates, vitamins, ...

What is Load?

- Load is a quantification of the characteristics of entities in an optimization problem
- Load often consists of more than one metric:
 - In shipping: volume, weight, shape, hazards, ...
 - In nutrition: fat, protein, carbohydrates, vitamins, ...
 - In sports: strength, speed, agility, experience, ...

What is Load?

- Load is a quantification of the characteristics of entities in an optimization problem
- Load often consists of more than one metric:
 - In shipping: volume, weight, shape, hazards, ...
 - In nutrition: fat, protein, carbohydrates, vitamins, ...
 - In sports: strength, speed, agility, experience, ...
- The same applies to computer programs!

Vector Load Balancing

Vector Load

- Instead of a single scalar l for load, use a vector $\vec{l} = \langle l_1, l_2, \dots, l_d \rangle$ of dimension d

Vector Load

- Instead of a single scalar l for load, use a vector $\vec{l} = \langle l_1, l_2, \dots, l_d \rangle$ of dimension d
- Remedies earlier deficiencies:

Vector Load

- Instead of a using single scalar l for load, use a vector $\vec{l} = \langle l_1, l_2, \dots, l_d \rangle$ of dimension d
- Remedies earlier deficiencies:
 - Phase-based applications
 - Time spent in each phase: $\langle t_A, t_B, \dots, t_n \rangle$

Vector Load

- Instead of using a single scalar l for load, use a vector $\vec{l} = \langle l_1, l_2, \dots, l_d \rangle$ of dimension d
- Remedies earlier deficiencies:
 - Phase-based applications
 - Time spent in each phase: $\langle t_A, t_B, \dots, t_n \rangle$
 - Heterogeneous computation
 - GPU time alongside CPU time: $\langle cpu, gpu \rangle$

Vector Load

- Instead of a using single scalar l for load, use a vector $\vec{l} = \langle l_1, l_2, \dots, l_d \rangle$ of dimension d
- Remedies earlier deficiencies:
 - Phase-based applications
 - Time spent in each phase: $\langle t_A, t_B, \dots, t_n \rangle$
 - Heterogeneous computation
 - GPU time alongside CPU time: $\langle cpu, gpu \rangle$
 - Resource constrained applications
 - Resource usage alongside CPU time: $\langle cpu, mem \rangle$

Measuring Vector Loads

- Features and APIs to measure vector loads:
 - Applications can call function to indicate phase boundary, RTS automatically measures per-phase load
 - Runtime flags to automatically add communication load (# messages, bytes sent)
 - Memory footprint via PUP
 - GPU load via accel or CUDA timers
 - Users may manually specify load vector

Vector Balancing

- Existing LB strategies cannot use a load vector
 - Still compatible, vector converted to scalar via sum, max, etc.
- Multiple dimensions makes vector load balancing more complex
 - Objects can no longer be totally ordered
 - Want to minimize over all dimensions simultaneously
 - Single variable optimization is now multivariate
- New LB strategies are needed

Vector Strategies - Greedy

- Extension of scalar greedy strategy to vector loads
 - Scalar version goes through objects from heaviest to lightest and assigns to the current least loaded processor

Vector Strategies - Greedy

- Extension of scalar greedy strategy to vector loads
 - Scalar version goes through objects from heaviest to lightest and assigns to the current least loaded processor
- Vector version:
 - Create d PE minheaps, each keyed on a different dimension of the vector, add all PEs to each heap
 - Go through objects in descending $\max(\vec{l})$, assign to minimum PE in dimension of $\max(\vec{l})$ and update heaps
- Simple, but focuses on single dimension at a time

Vector Strategies - METIS

- METIS supports giving vertices vector weights
- Reframe LB problem as graph, objects map to vertices and output partitions map to PEs
 - No edges, but could use with comm graph
- Implemented via bipartitioning objects based on $\max(\vec{l})$ like Greedy, but adds extra refinement phase
 - Graph coarsening/refinement for sake of performance
- Generally works pretty well, but gives poor results for some configurations

Vector Strategies - Norm

- Go through objects in descending order and place on the PE such that the post-placement PE load vector norm is globally minimized
 - Works well, but computationally expensive
 - Norm inequalities are not preserved under vector addition:
 - $\|(2, 0)\|_2 < \|(0, 3)\|_2$
 - $\|(2, 0) + (3, 0)\|_2 > \|(0, 3) + (3, 0)\|_2$
 - Makes it non-trivial to reduce search space
- Choice of underlying norm ($2, 4, \infty$, etc.)

Vector Strategies - Norm

- Implemented several different versions of norm-based assignment
 - Exhaustive search
 - k -d tree-based search
 - rk -d tree-based search
- All give same result, but different performance

Vector Strategies - Norm

- Implemented several different versions of norm-based assignment
 - Exhaustive search
 - k -d tree-based search
 - rk -d tree-based search
- All give same result, but different performance
- The rk -d variant generally performs best

NormLB - k -d

- Represent PEs as points in a k -d tree
 - Arbitrary dimensional space partitioning tree
 - Can prune search space as candidates are found
- k -d works well for searching in static point set, but here, tree updated after every assignment
 - Costly update operations
 - Structured pattern of updates 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:
 - Random** Discriminant is uniformly randomly chosen and each insertion has some probability of becoming the root, or root of subtree, ...
 - Relaxed** Instead of cycling through discriminants, $1, 2, \dots, k, 1, \dots$, each node stores arbitrary discriminant $j \in \{1, 2, \dots, k\}$
- Stochasticity keeps tree balanced, makes updates fast

Random Relaxed k -d

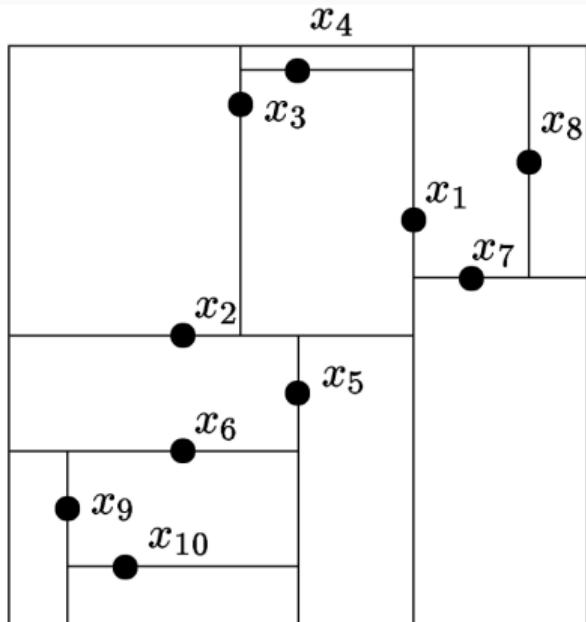


Figure 1: k -d

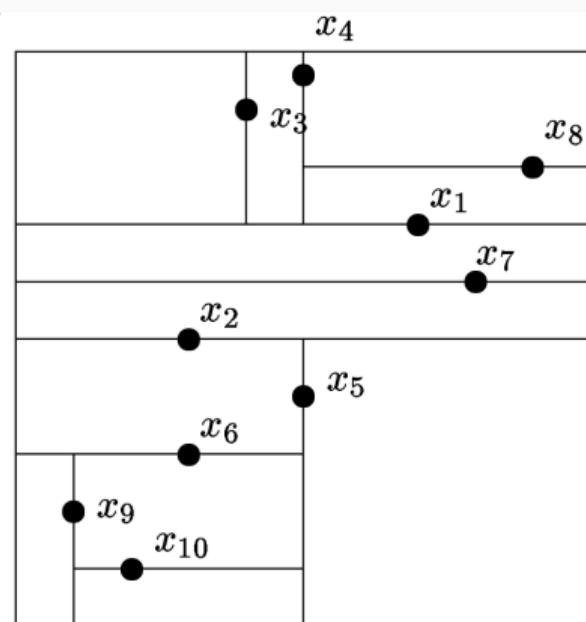
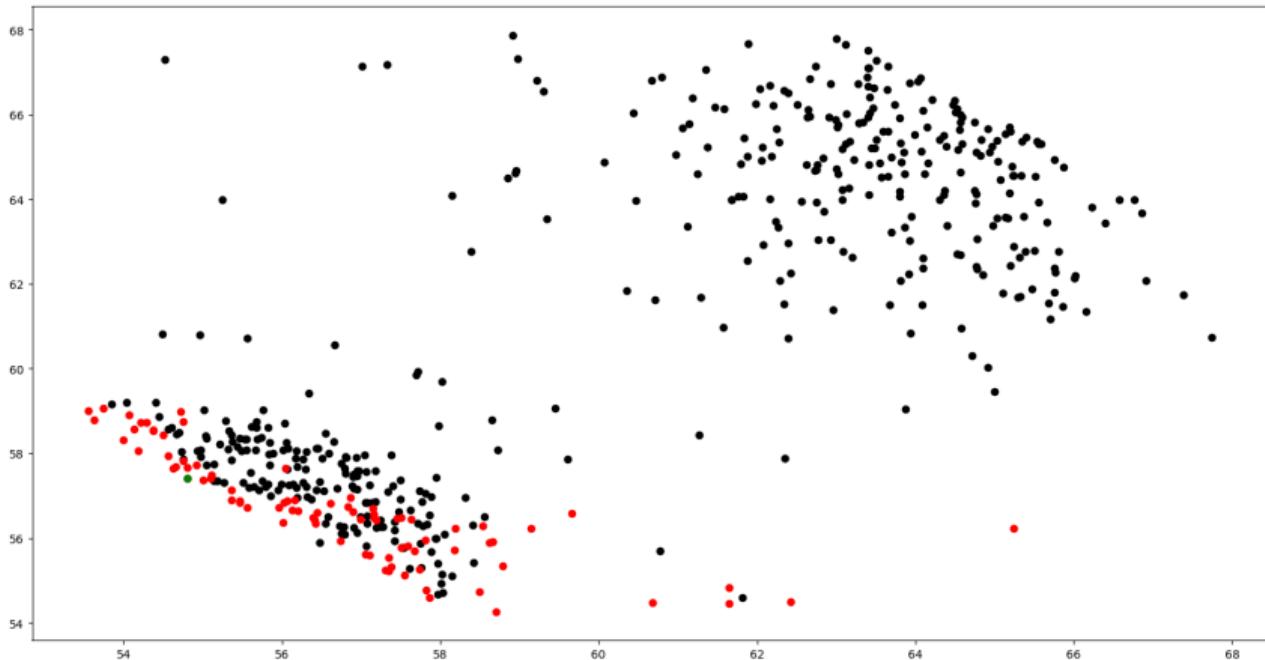


Figure 2: rk-d

k -d - Search Pruning



Objective Function

- How to tell which mapping is better?

Objective Function

- How to tell which mapping is better?
- Easy in scalar world, totally ordered, so smaller max load is better

Objective Function

- How to tell which mapping is better?
- Easy in scalar world, totally ordered, so smaller max load is better
- In vector world, only partially ordered

Objective Function

- How to tell which mapping is better?
- Easy in scalar world, totally ordered, so smaller max load is better
- In vector world, only partially ordered
 - Is $\langle 4, 4 \rangle$ or $\langle 5, 2 \rangle$ better?

Objective Function

- How to tell which mapping is better?
- Easy in scalar world, totally ordered, so smaller max load is better
- In vector world, only partially ordered
 - Is $\langle 4, 4 \rangle$ or $\langle 5, 2 \rangle$ better?
- Have to choose correct objective function for scenario!

Phase Objective Function

M mapping, O set of objects, P set of PEs, d dimensions

$$\arg \min_M \sum_{0 \leq i < d} \left(\max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_i \right) \right) \quad (1)$$

Phase Objective Function

M mapping, O set of objects, P set of PEs, d dimensions

$$\arg \min_M \sum_{0 \leq i < d} \left(\max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_i \right) \right) \quad (1)$$

Load on PE p in dimension i

Phase Objective Function

M mapping, O set of objects, P set of PEs, d dimensions

$$\arg \min_M \sum_{0 \leq i < d} \left(\max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_i \right) \right) \quad (1)$$

Load on PE p in dimension i

↑
Maximum load in dimension i on a single PE

Phase Objective Function

M mapping, O set of objects, P set of PEs, d dimensions

Sum across dimensions

$$\arg \min_M \sum_{0 \leq i < d} \left(\max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_i \right) \right) \quad (1)$$

Load on PE p in dimension i

Maximum load in dimension i on a single PE

Overlapped Objective Function

M mapping, O set of objects, P set of PEs, d dimensions

$$\arg \min_M \max_{0 \leq i < d} \left(\max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_i \right) \right) \quad (2)$$

Load on PE p in dimension i

↑
Maximum load in dimension i on a single PE

Overlapped Objective Function

M mapping, O set of objects, P set of PEs, d dimensions

$$\arg \min_M \max_{0 \leq i < d} \left(\max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_i \right) \right) \quad (2)$$

Max across dimensions

Load on PE p in dimension i

Maximum load in dimension i on a single PE

Generalized Objective Function

Take \vec{l}_M to be the maximum load vector:

$$\vec{l}_M = \left\langle \max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_1 \right), \dots, \max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_d \right) \right\rangle \quad (3)$$

Generalized Objective Function

Take \vec{l}_M to be the maximum load vector:

$$\vec{l}_M = \left\langle \max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_1 \right), \dots, \max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_d \right) \right\rangle \quad (3)$$

Then the general objective function is:

$$\arg \min_M \left\| \vec{l}_M \right\|_k \quad (4)$$

Generalized Objective Function

Take \vec{l}_M to be the maximum load vector:

$$\vec{l}_M = \left\langle \max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_1 \right), \dots, \max_{p \in P} \left(\sum_{\forall o \in O: M(o)=p} (\vec{l}_o)_d \right) \right\rangle \quad (3)$$

Then the general objective function is:

$$\arg \min_M \left\| \vec{l}_M \right\|_k \quad (4)$$

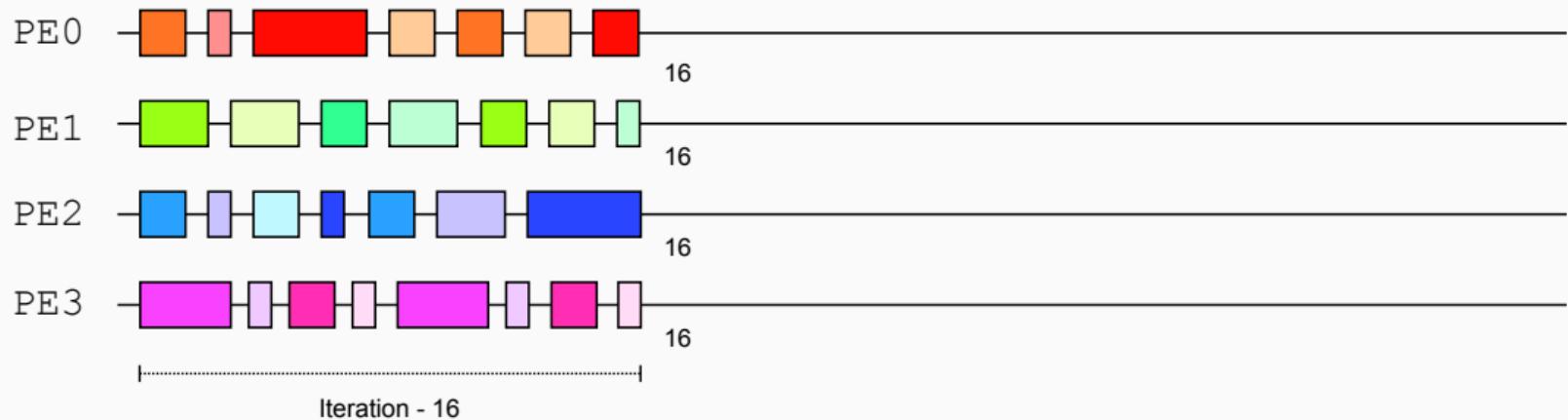
The phase case is $k = 1$, overlap is $k = \infty$

Balancing Phase-Based Applications

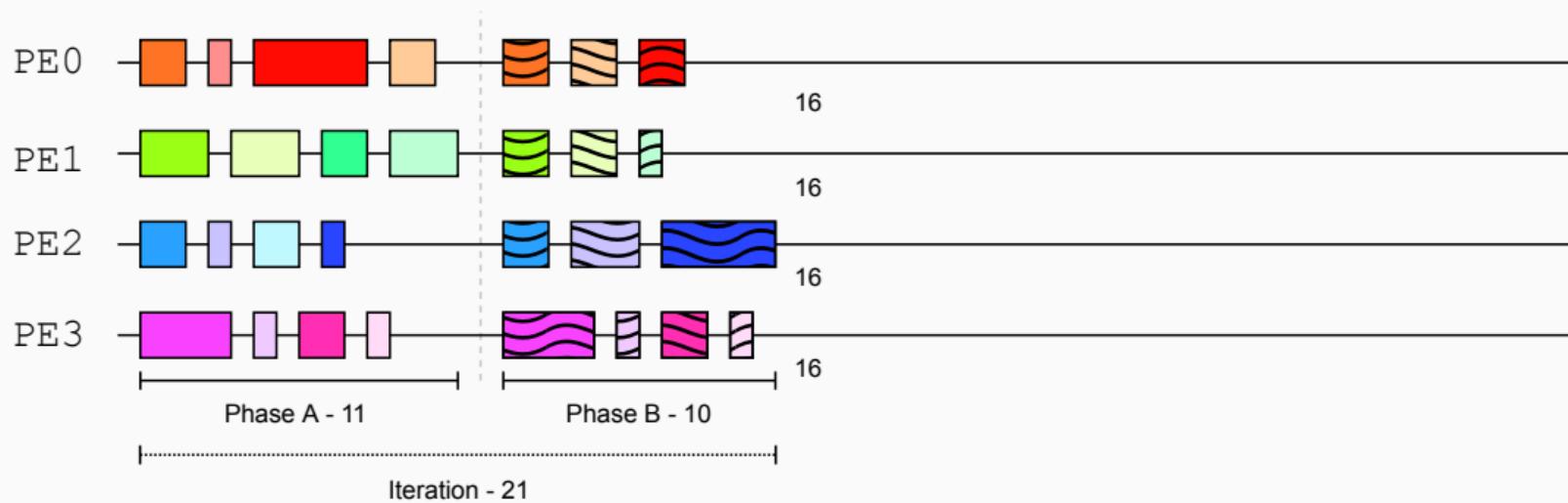
Why Phase-Based Applications?

- Multiphysics
 - Interaction of n different physical processes
- Multiscale
 - Update parts of domain at different frequencies
- Specialization/Data Dependent Computation
 - Update everything, but different techniques for different distances, e.g. EM vs PME
- Optimized Library Usage
 - Transmute data to use optimized library, e.g. graph data -> matrix for BLAS -> graph data

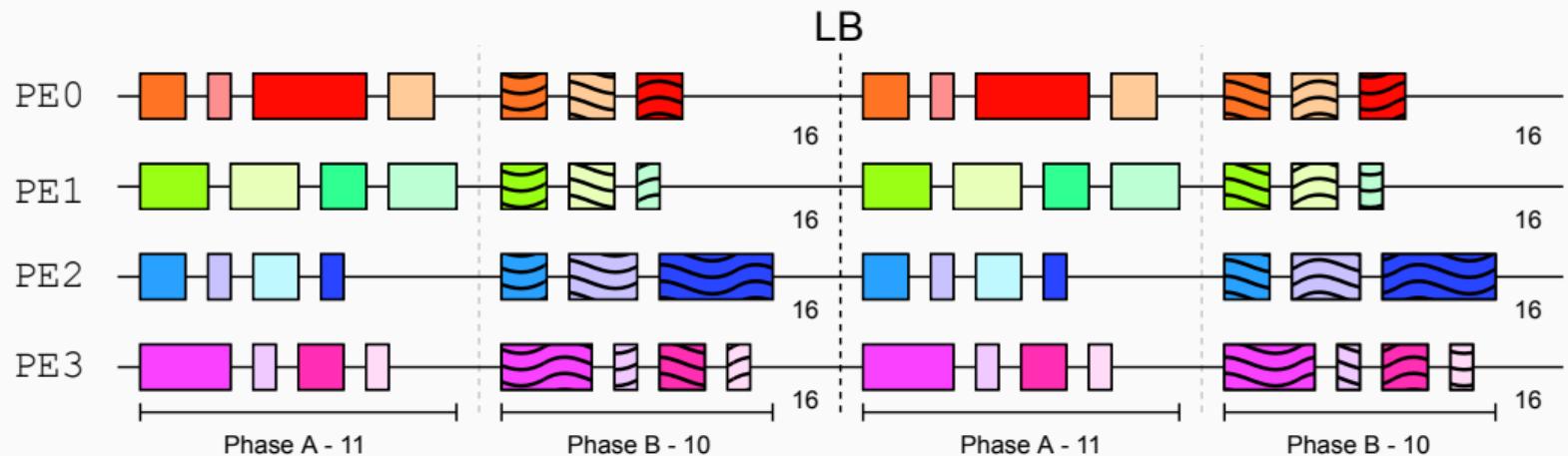
Non-Phase-Based Application



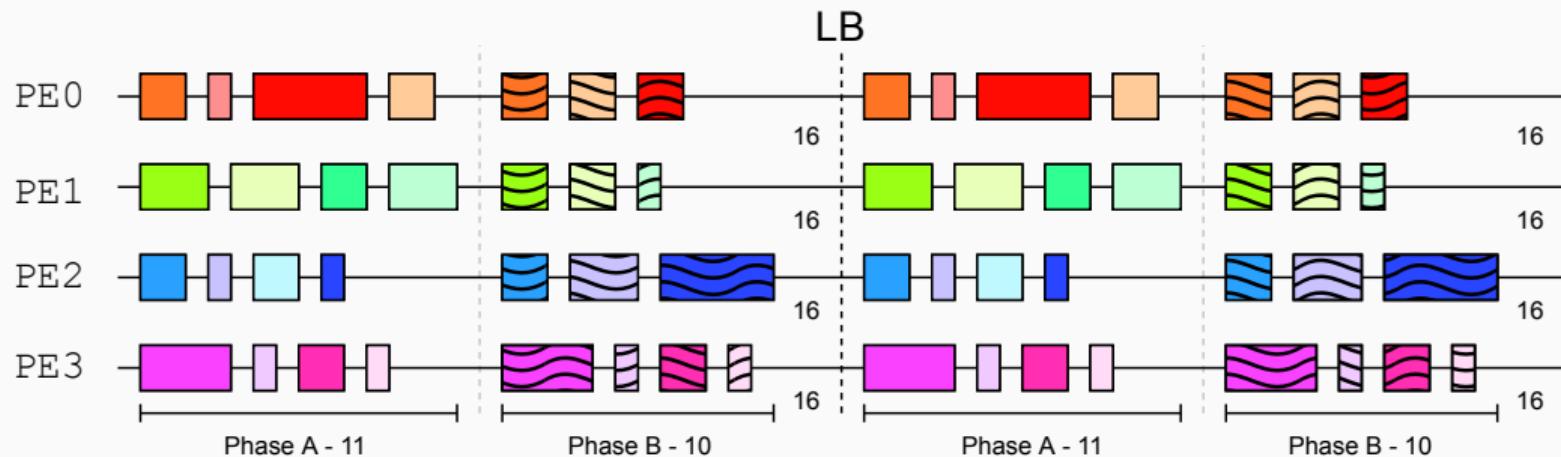
Phase-Based Application



Scalar LB

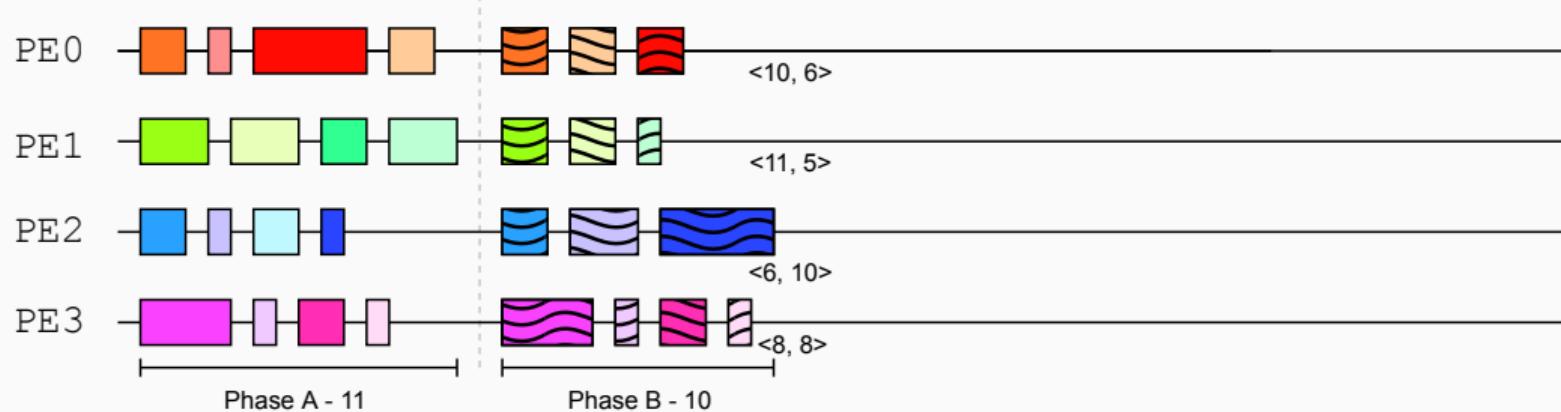


Scalar LB

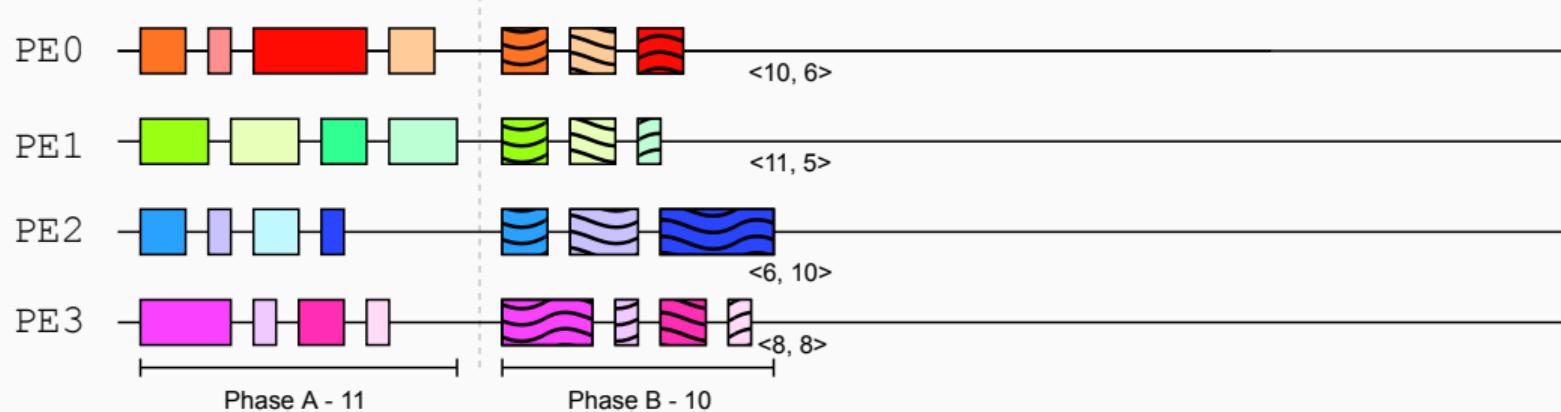


Scalar loads cannot identify imbalance!

Vector LB

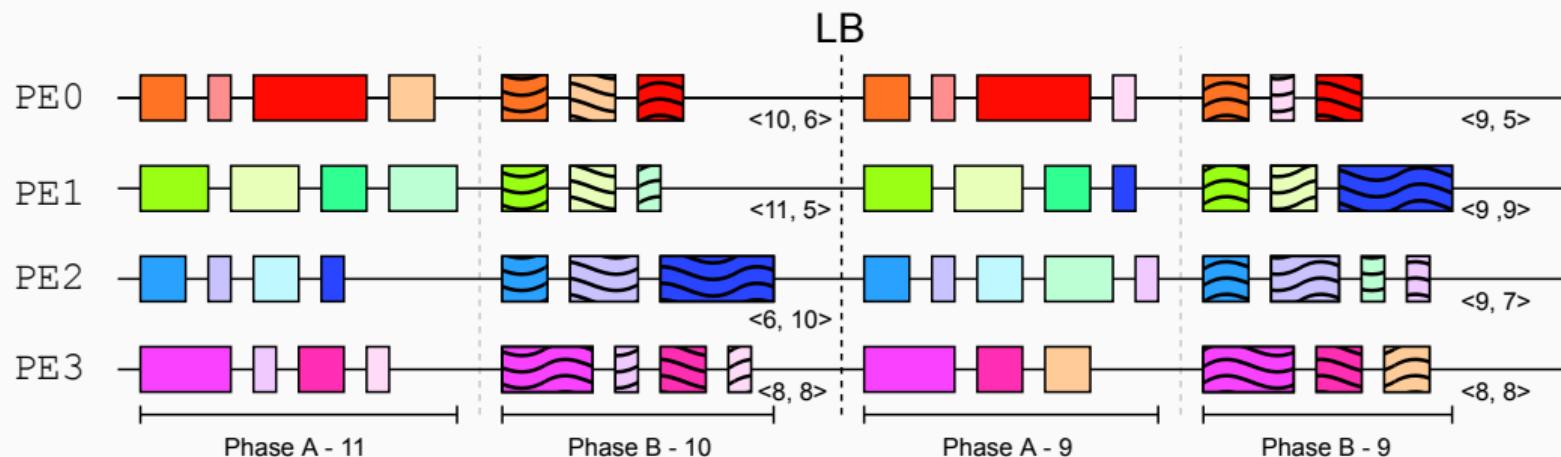


Vector LB

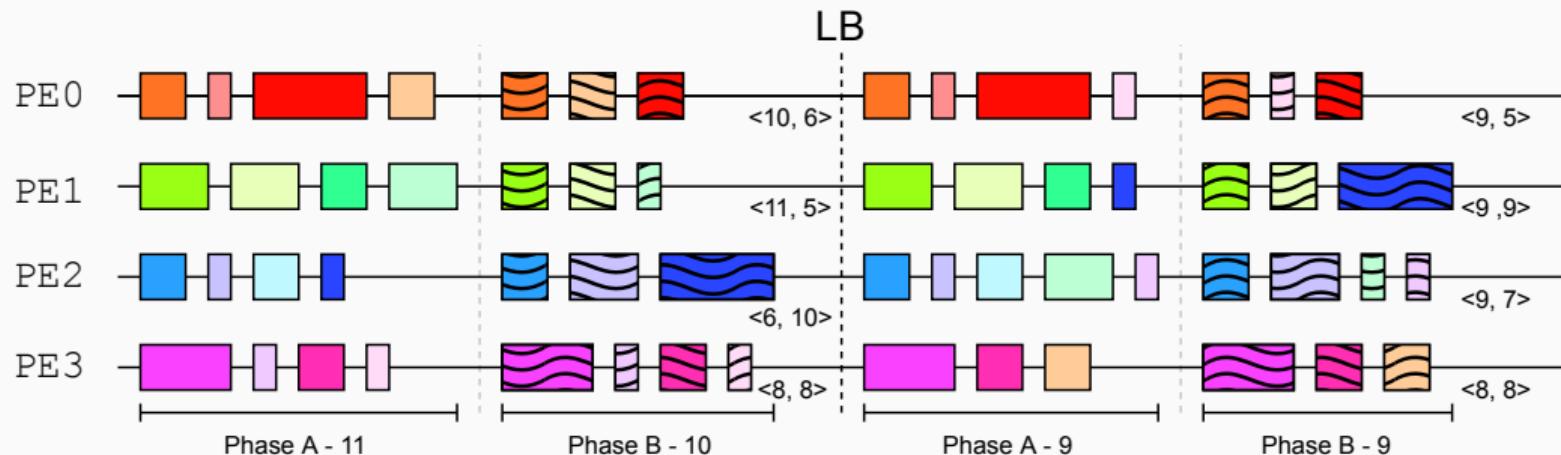


Identifies imbalance!

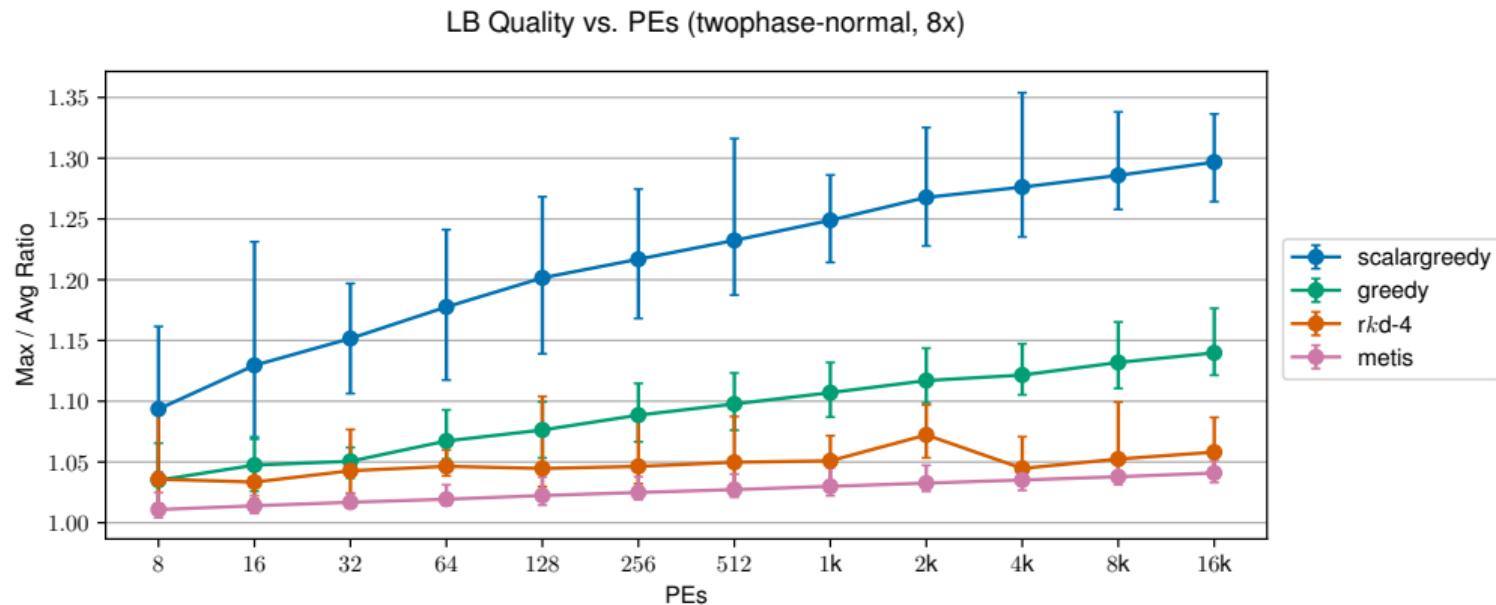
Vector LB



Vector LB

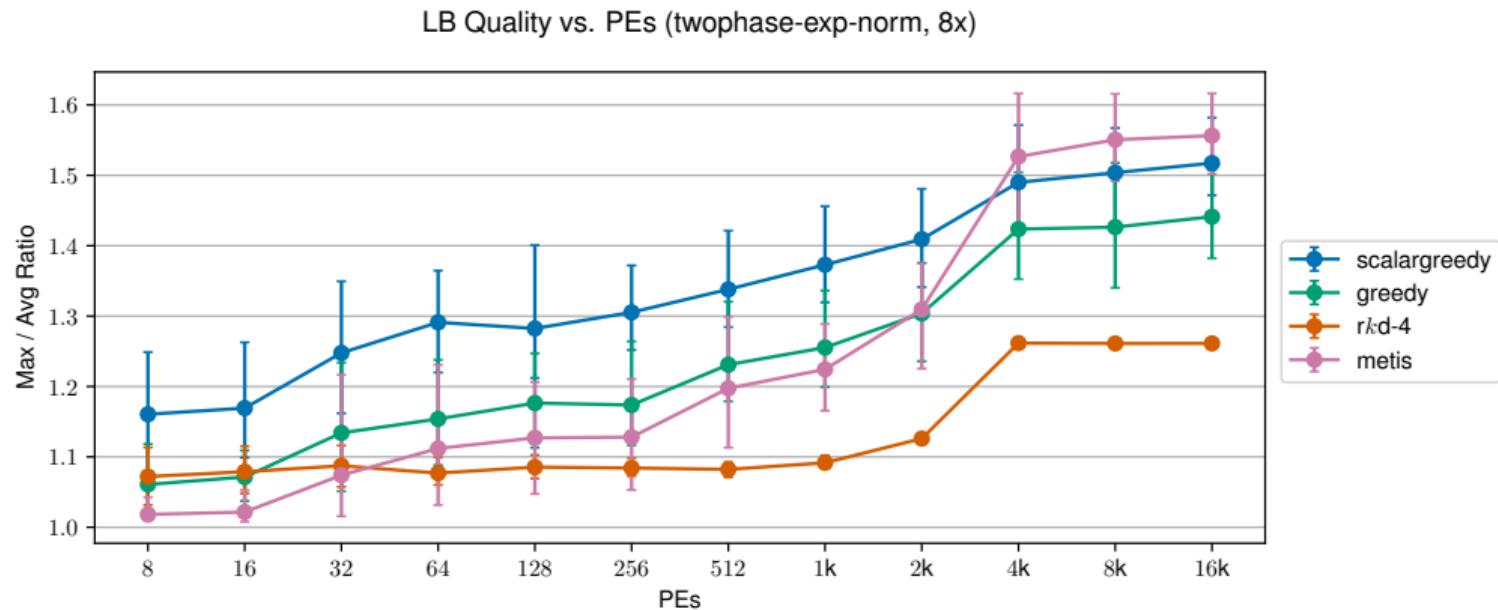


Vector LB Simulations - Phase



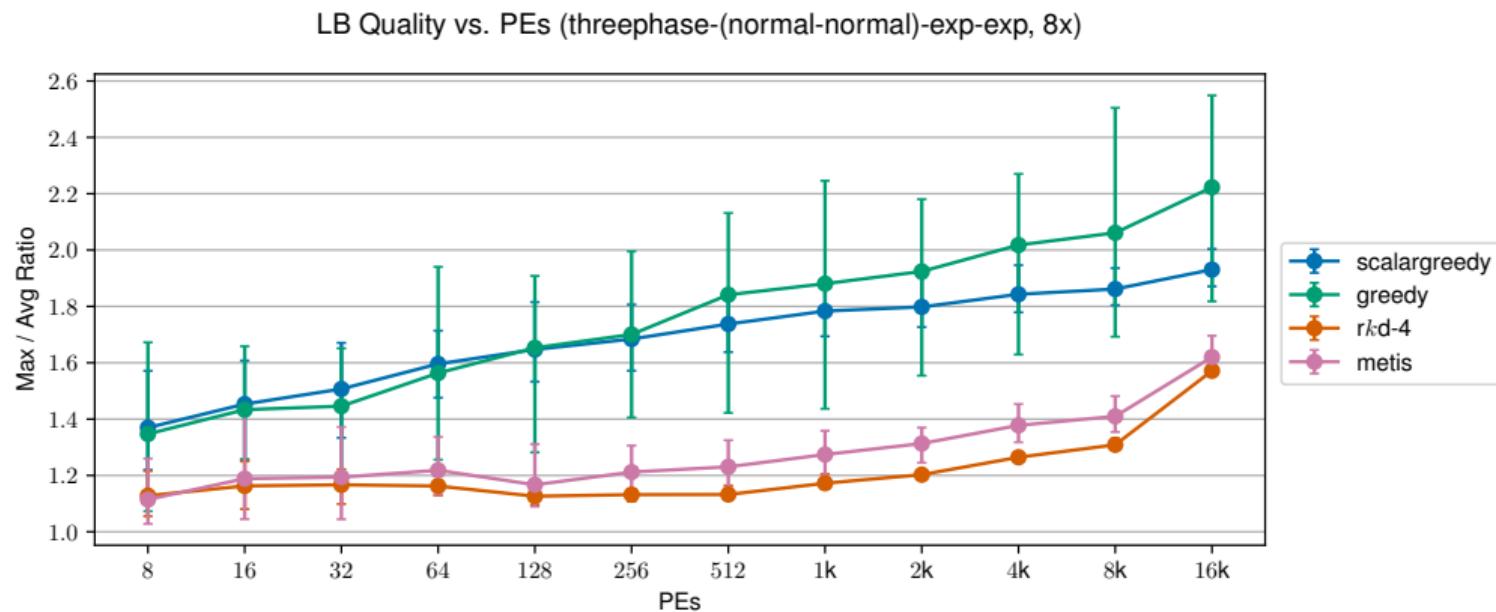
100 trials, Synthetic data: 2 phase (normal $\mu = 10, \sigma = 3$, normal $\mu = 10, \sigma = 3$) 38

Vector LB Simulations - Phase



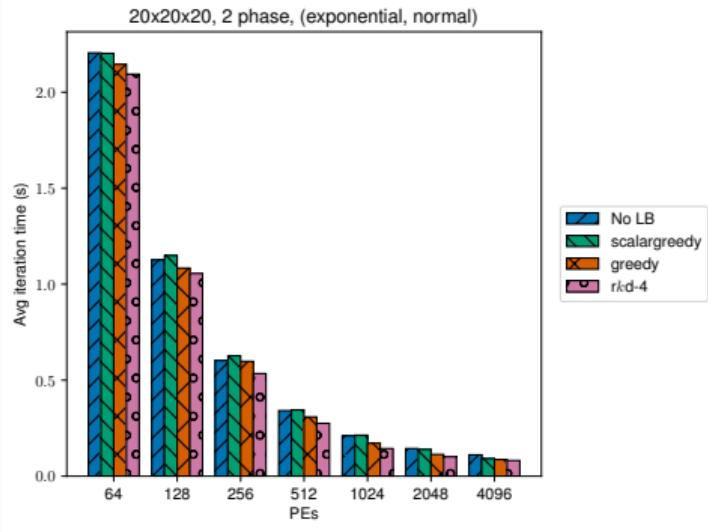
100 trials, Synthetic data: 2 phase ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Vector LB Simulations - Phase

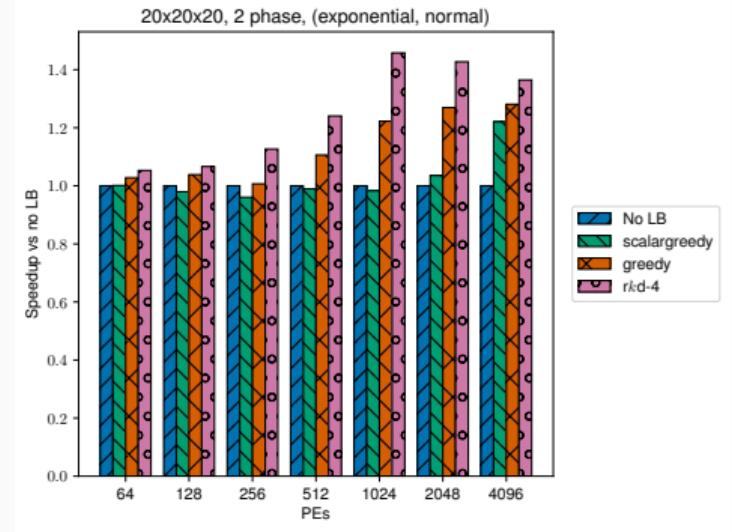


100 trials, Synthetic data: 3 phase (normal $\mu = 1_{(0.8)}, 5_{(0.2)}$, $\sigma = 0.1$, exp $\lambda = 0.15$, exp $\lambda = 0.15$)

2 Phase Stencil



Runtime



Speedup

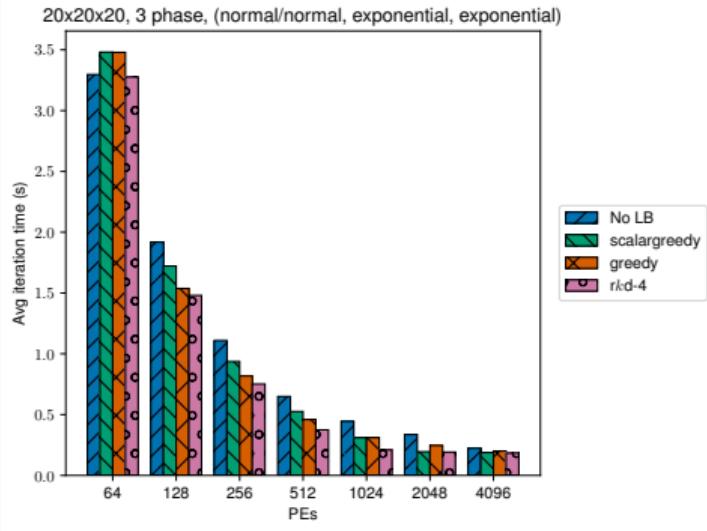
KNL partition of Stampede2, 2 phase ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

2 Phase Stencil Speedup

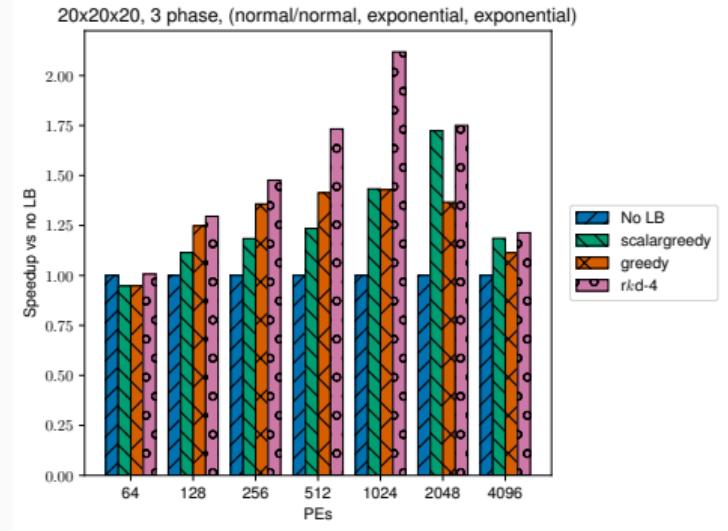
Size	Strategy	Speedup		
		Minimum	Average	Maximum
10x10x10	<i>Scalar Greedy</i>	1.03	1.14	1.28
	<i>Greedy</i>	1.09	1.23	1.29
	<i>rk-d</i>	1.12	1.28	1.42
20x20x20	<i>Scalar Greedy</i>	0.96	1.02	1.22
	<i>Greedy</i>	1.01	1.14	1.28
	<i>rk-d</i>	1.05	1.25	1.46

Table 1: Speedup Over Baseline with Load Balancing for Two Phase (Exponentially, Normally) Distributed Problem

3 Phase Stencil



Runtime



Speedup

KNL partition of Stampede2, 3 phase (normal $\mu = 1_{(0.8)}, 5_{(0.2)}$, $\sigma = 0.1$, exp $\lambda = 0.15$, exp $\lambda = 0.15$)

3 Phase Stencil Speedup

Size	Strategy	Speedup		
		Minimum	Average	Maximum
10x10x10	<i>Scalar Greedy</i>	1.23	1.29	1.37
	<i>Greedy</i>	1.08	1.20	1.50
	<i>rk-d</i>	1.33	1.56	1.76
20x20x20	<i>Scalar Greedy</i>	0.95	1.26	1.72
	<i>Greedy</i>	0.95	1.27	1.43
	<i>rk-d</i>	1.01	1.51	2.12

Table 2: Speedup with Load Balancing for Three Phase
(Normally/Normally, Exponentially, Exponentially) Distributed
Problem

MPI Support

- Adaptive MPI is MPI on top of Charm++
 - Adds automatic load balancing to MPI programs
 - Measurement: RTS measures per-rank execution time
 - Overdecomposition: Use more ranks than PEs
 - Migratability: Ranks are ULTs that can migrate

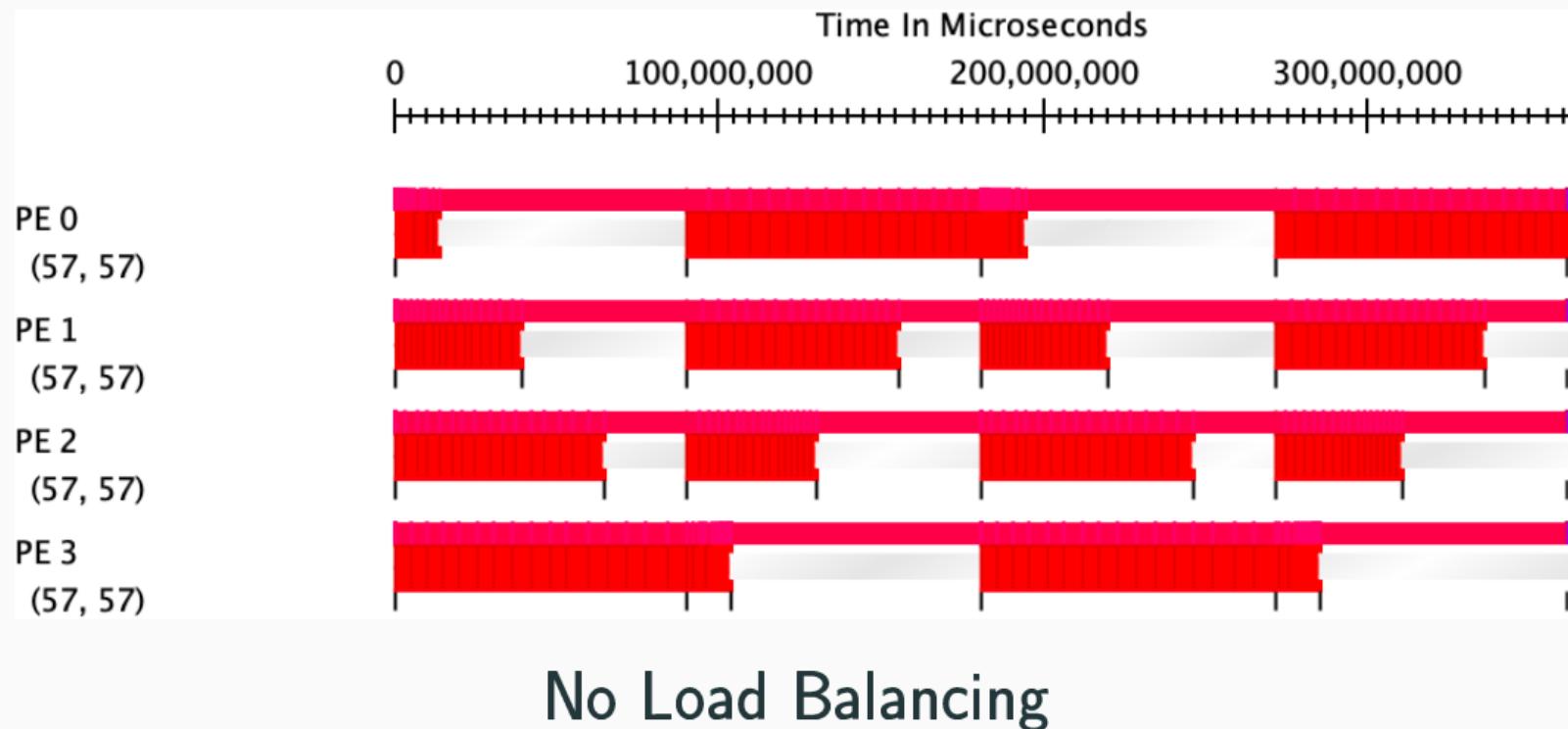
MPI Support

- Adaptive MPI is MPI on top of Charm++
 - Adds automatic load balancing to MPI programs
 - Measurement: RTS measures per-rank execution time
 - Overdecomposition: Use more ranks than PEs
 - Migratability: Ranks are ULTs that can migrate
- Added support for phase measurements
 - Call `void AMPI_Load_set_phase(int phase)` to mark boundary

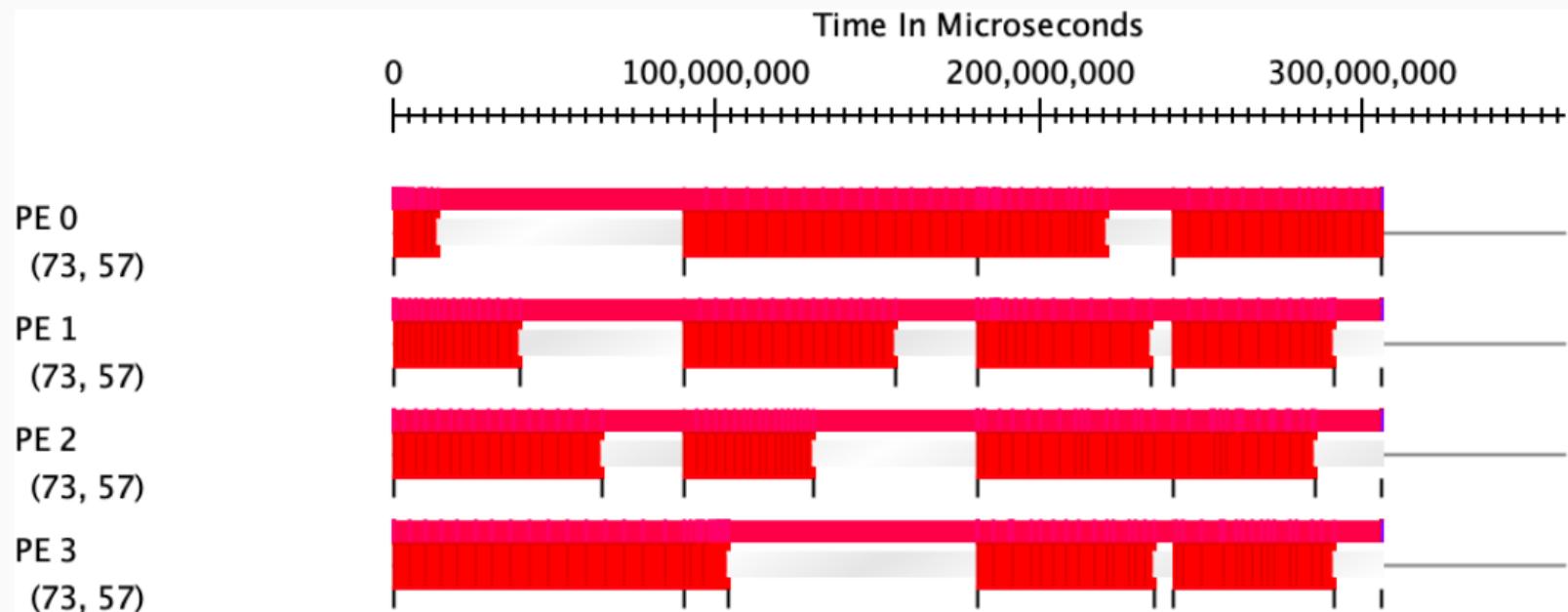
AMPI Vector LB

- Phase-based MPI mini-app
- Two phases:
 - First grows linearly with index
 - Second shrinks linearly with index
- Evaluated on local test machine, proof of concept

AMPI No LB

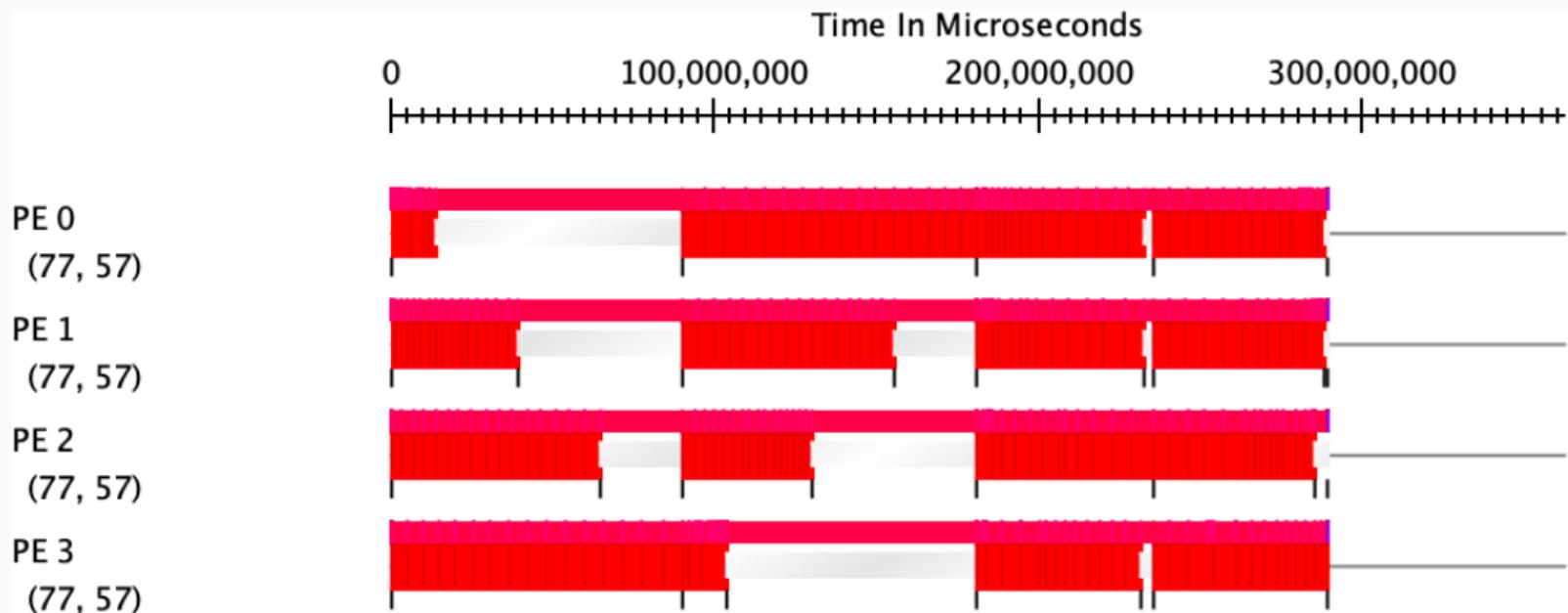


AMPI Scalar Greedy LB



Scalar Load Balancing

AMPI k -d LB



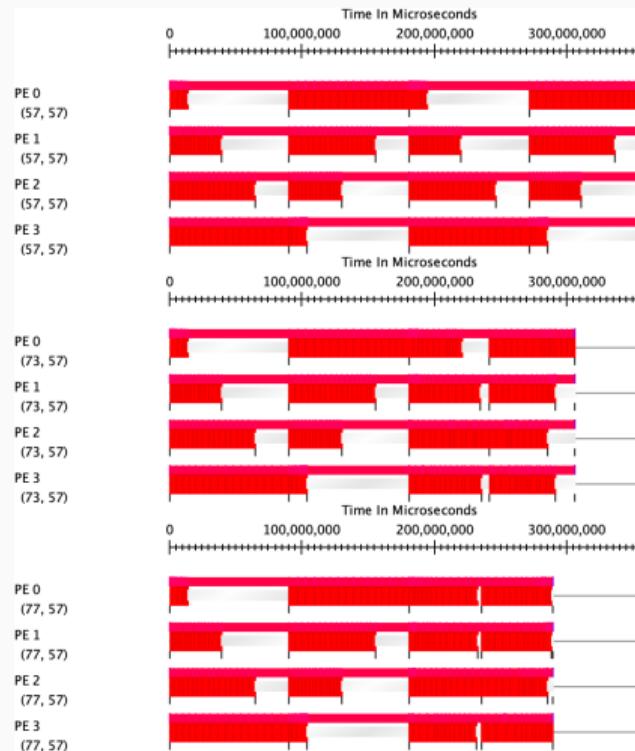
Vector Load Balancing

AMPI LB Performance

LB Off

Scalar LB
(1.44x speedup)

Vector LB
(1.67x speedup)



VT

- Task-based programming model from Sandia
- Designed for modular use with MPI programs
- Supports phase-level load measurement, but no vector LBs
- Collaborated with team to create two adapters:
 1. To allow Charm++ LBs to be used in VT
 2. To ingest VT logs into Charm++ LB simulator

EMPIRE

- Unstructured finite element PIC code from Sandia
 - Used for electric fields, magnetic fields, high-energy plasma simulation
- Ported to VT in attempt to deal with imbalance
- Complicated phase-based execution structure
 - 14 dimensions

EMPIRE

- Unstructured finite element PIC code from Sandia
 - Used for electric fields, magnetic fields, high-energy plasma simulation
- Ported to VT in attempt to deal with imbalance
- Complicated phase-based execution structure
 - 14 dimensions
- *kdLB* gives approximately 12% performance improvement over previous best LB strategy

EMPIRE - Previous Best LB

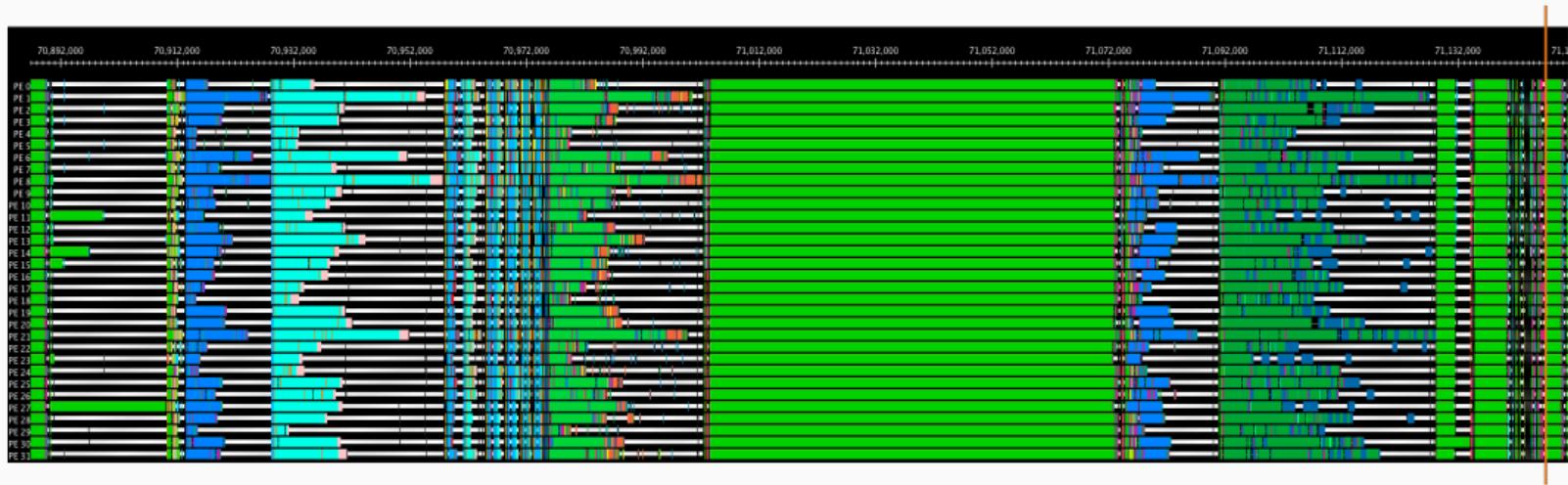


Figure 3: With previous best LB

EMPIRE - k -d LB

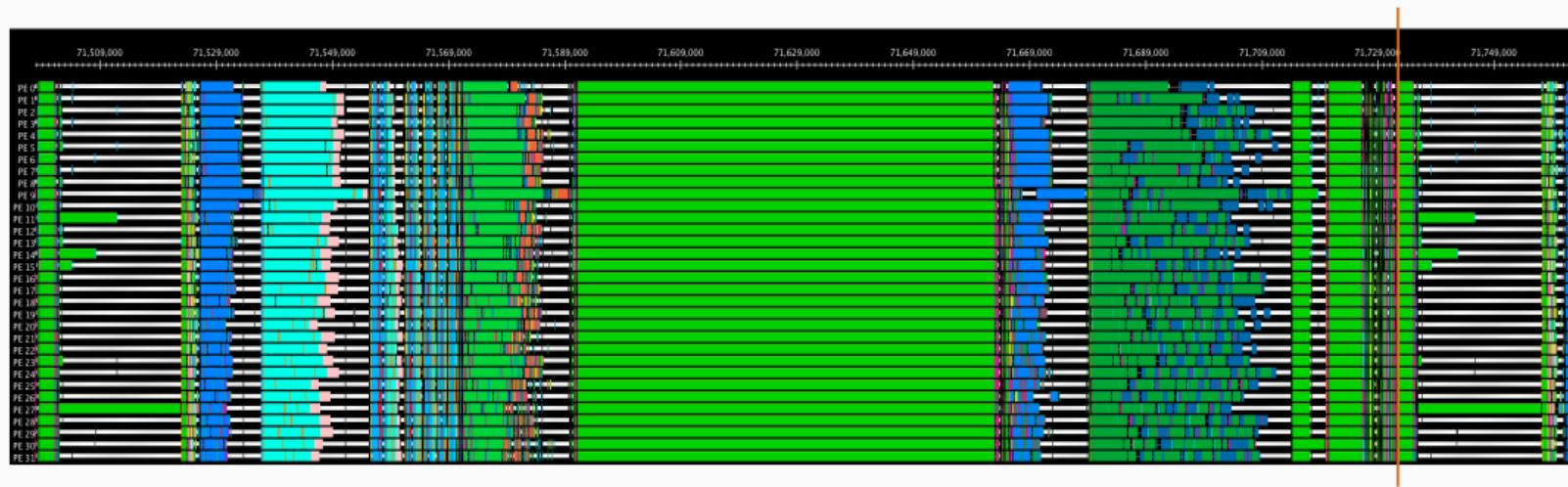


Figure 4: With $kdLB$

Phase-Based Summary

- Load vectors fixes fundamental load representation issue with scalar
- Demonstrated utility with multiple programming models: Charm++, MPI, VT
- Speedups up to 2.12x compared to no LB, scalar best 1.72x

Balancing Heterogeneous Applications

Heterogeneous Execution

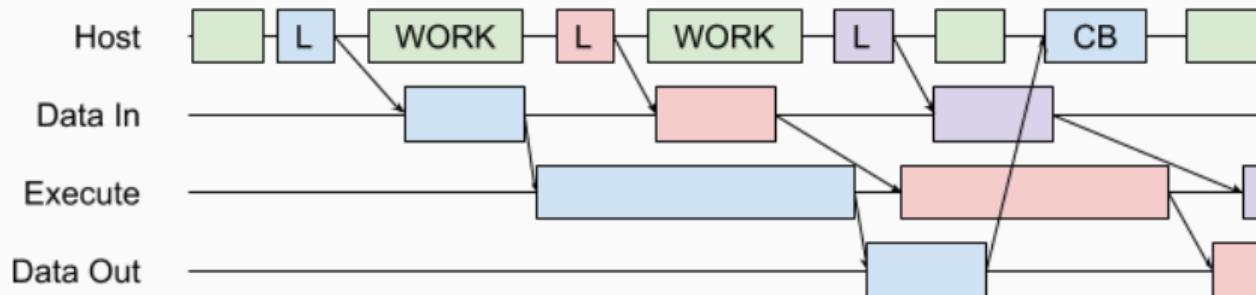
- Two types of heterogeneous execution:
 - *Retargetable* - Malleable tasks can execute on CPU or accelerator, intranode
 - *Non-Retargetable* - Tasks can execute on only one target architecture, but not all tasks have same target, intranode and/or internode
- We study these two cases separately

Intra-Node Balancing for Retargetable Work

- Use runtime information to manage execution of retargetable tasks
- Cede control of kernels, data, etc. to RTS
- Balance between resources to minimize idle time
- *Modifies* object's load vector
 - Load shifts from one dimension to another

GPU Manager

- Kernels registered with Charm++
- Runtime manages dataflow, execution, callbacks when kernels complete
- Asynchronously overlaps data movement and execution



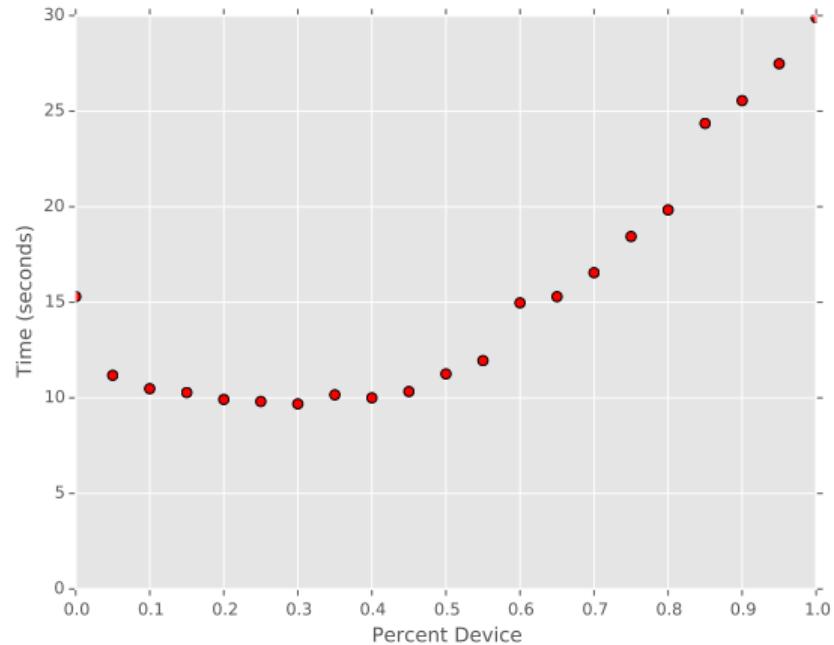
Accel Framework

- Generates code for specially written and tagged functions
 - Host CPU and CUDA variants
 - Code written with `splitIndex`, `numSplits`, analogous to `threadIdx`, `blockDim` in CUDA
 - Annotations for data flow direction, lifetime
 - Can batch kernel launches, other optimizations
- Execution platform decided at runtime
 - Various scheduling strategies available

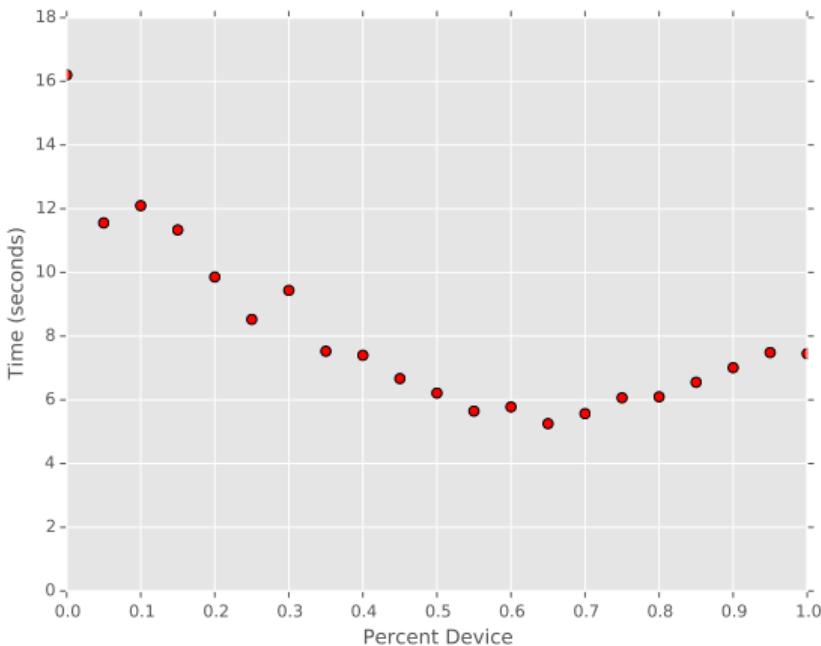
Benchmark Description

- 2D Stencil
 - 5-point stencil
 - 6144x6096 grid
- Molecular Dynamics
 - 5x5x5 grid
 - 32k molecules
- Messaging goes through CPU, so requires copy back
 - MD has higher FLOP/byte ratio, better for GPU
- One GPU node of Stampede (2x Intel Xeon E5-2680, NVIDIA K20)

2D Stencil Results



Molecular Dynamics Results



Timing for md

Intranode Results

	Best Split	Host Only	Device Only
stencil2d	30% device	1.58x	3.09x
md	65% device	3.02x	1.46x

Speedup of Best Configuration Relative to Host/Device Only

Intranode Results

	Best Split	Host Only	Device Only
stencil2d	30% device	1.58x	3.09x
md	65% device	3.02x	1.46x

Speedup of Best Configuration Relative to Host/Device Only

Intranode balancing of retargetable work helps
performance

Vector LB for Non-Retargetable Work

- Have some GPU-only tasks, some CPU-only tasks
 - Both sets executing simultaneously

Vector LB for Non-Retargetable Work

- Have some GPU-only tasks, some CPU-only tasks
 - Both sets executing simultaneously
- Each hardware resource gets own dimension in vector

Vector LB for Non-Retargetable Work

- Have some GPU-only tasks, some CPU-only tasks
 - Both sets executing simultaneously
- Each hardware resource gets own dimension in vector
- Performance determined by which resource finishes last
 - Use maximum-based objective function

Why Non-Retargetable?

- Less scope for balancing than retargetable case

Why Non-Retargetable?

- Less scope for balancing than retargetable case
- Better control over memory locations
 - RDMA, GPU collectives
 - Avoid thrashing

Why Non-Retargetable?

- Less scope for balancing than retargetable case
- Better control over memory locations
 - RDMA, GPU collectives
 - Avoid thrashing
- Low-level primitives, proprietary features

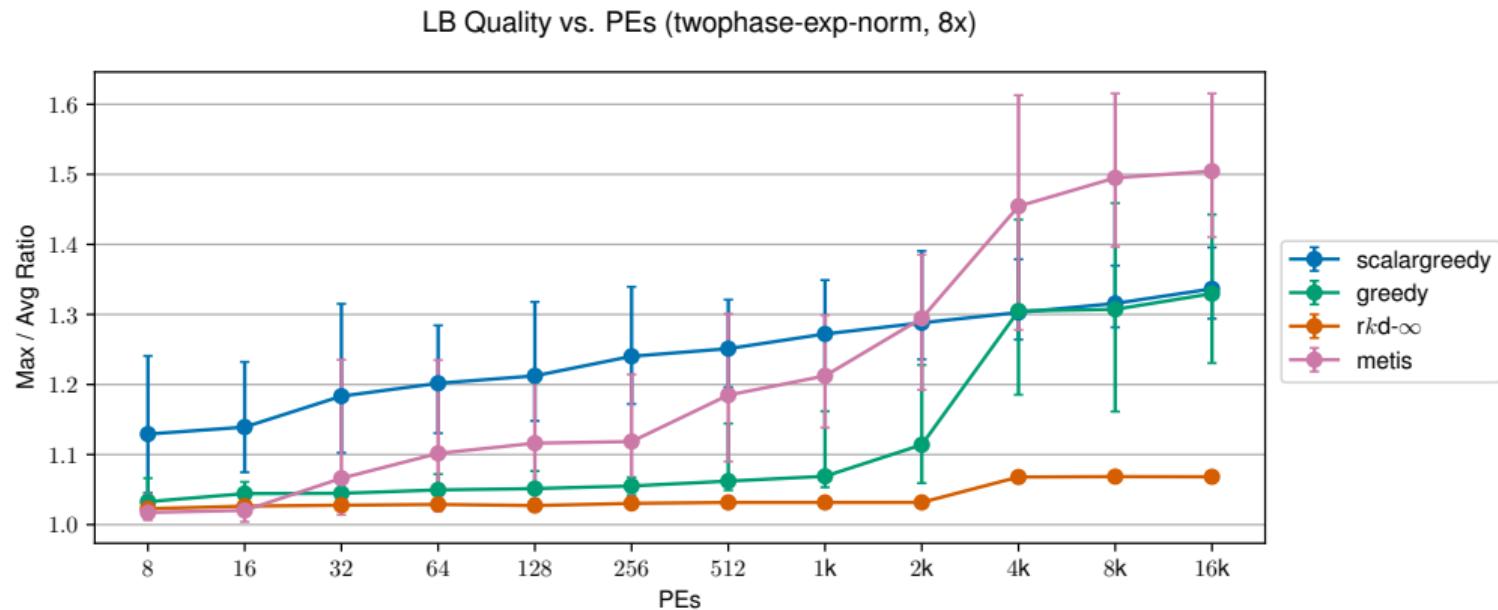
Why Non-Retargetable?

- Less scope for balancing than retargetable case
- Better control over memory locations
 - RDMA, GPU collectives
 - Avoid thrashing
- Low-level primitives, proprietary features
- Easier to integrate into existing applications

Why Non-Retargetable?

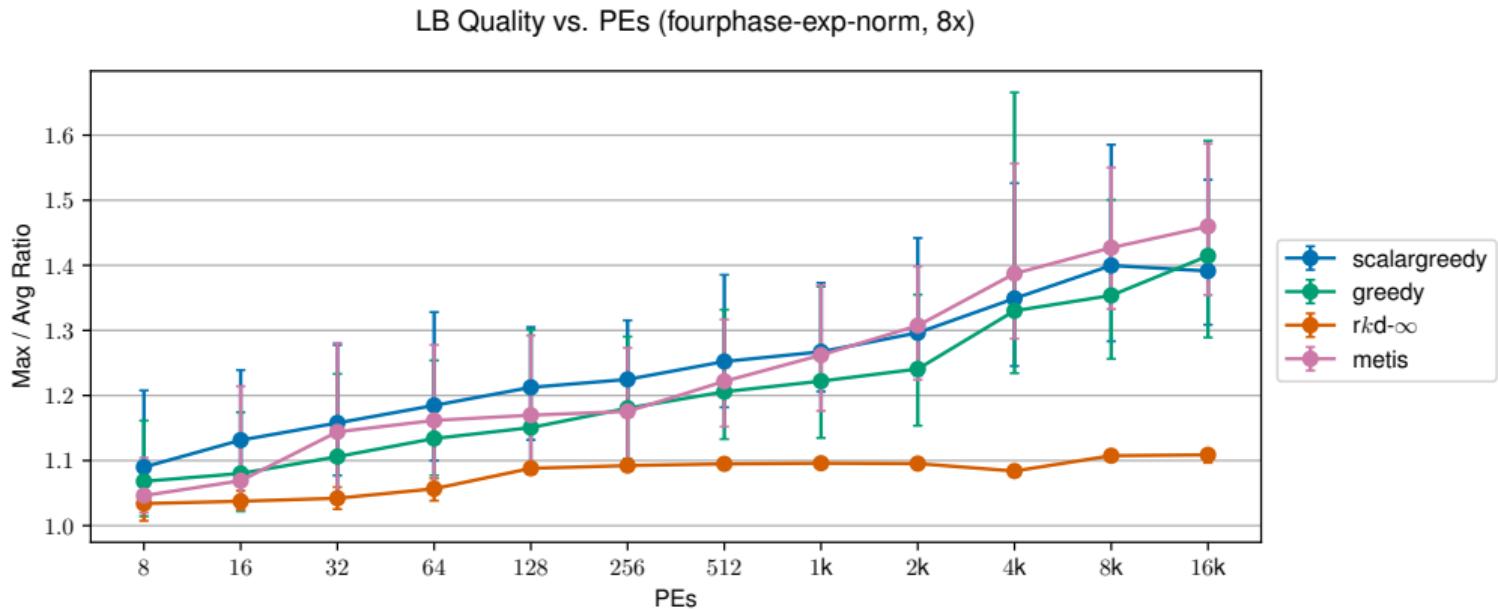
- Less scope for balancing than retargetable case
- Better control over memory locations
 - RDMA, GPU collectives
 - Avoid thrashing
- Low-level primitives, proprietary features
- Easier to integrate into existing applications
- Used by NAMD, ChaNGa, etc.

Vector LB Quality - 2D



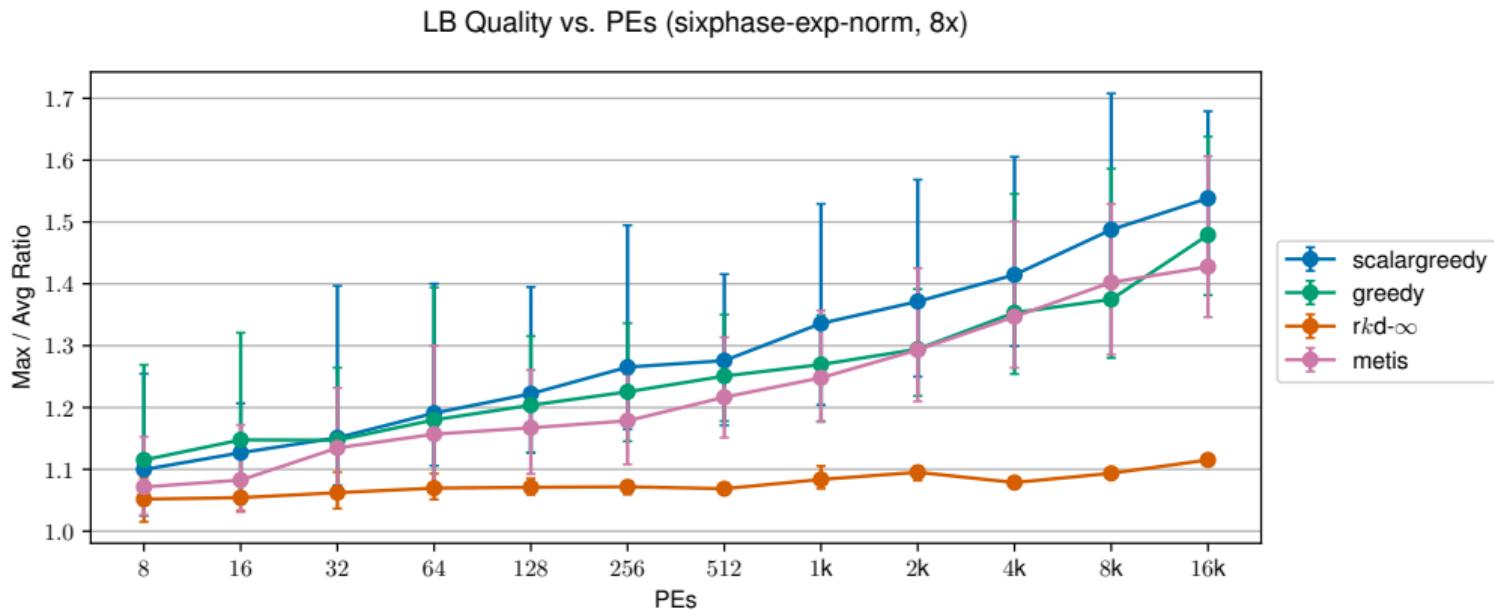
100 trials, Synthetic data: 2 phase ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Overlap LB Quality - 4D



100 trials, Synthetic data: 4 phase ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Overlap LB Quality - 6D



100 trials, Synthetic data: 6 phase ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Heterogeneous LB Summary

- Runtime intranode task retargeting improves balance and performance
- Vector allows for accurate representation of heterogeneous load
- Vector LB can balance tasks targeted to different hardware

Balancing With Constraints

Constraint LB Motivation

- System constraints may restrict load balancing remappings
 - Moving objects with large memory footprints may exceed node's capacity
 - Router buffer sizes may constrain number of messages sent from node

Constraint LB Motivation

- System constraints may restrict load balancing remappings
 - Moving objects with large memory footprints may exceed node's capacity
 - Router buffer sizes may constrain number of messages sent from node
- Constraints likely different units, different scale compared to load timings

Constraint LB Motivation

- System constraints may restrict load balancing remappings
 - Moving objects with large memory footprints may exceed node's capacity
 - Router buffer sizes may constrain number of messages sent from node
- Constraints likely different units, different scale compared to load timings
- Can add constraints to vector, but need specialized balancer

k -d Constraint LB

- New variant of k -d based LB
- Accepts list of constraints corresponding to dimensions of load vector

k -d Constraint LB

- New variant of k -d based LB
- Accepts list of constraints corresponding to dimensions of load vector
- Splits load vector into two parts, one to minimize, one to constrain

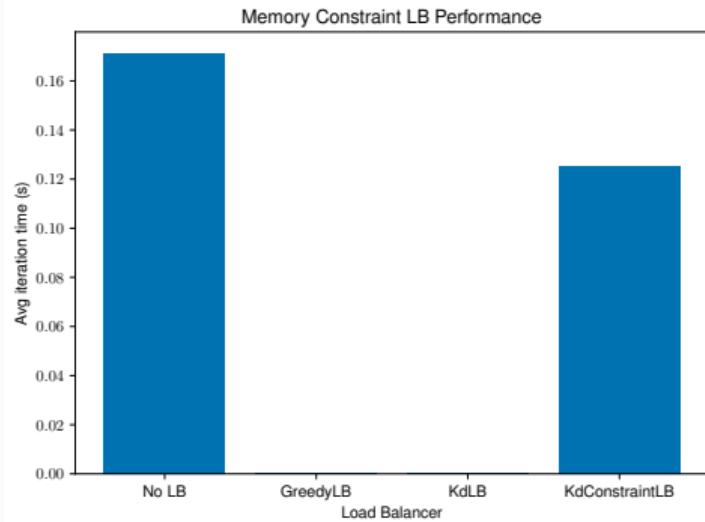
k -d Constraint LB

- New variant of k -d based LB
- Accepts list of constraints corresponding to dimensions of load vector
- Splits load vector into two parts, one to minimize, one to constrain
- Culls PE search space to non-violating hypervolume
 - Minimizes as normal for other dimensions within this

k -d Constraint LB

- New variant of k -d based LB
- Accepts list of constraints corresponding to dimensions of load vector
- Splits load vector into two parts, one to minimize, one to constrain
- Culls PE search space to non-violating hypervolume
 - Minimizes as normal for other dimensions within this
- Supports arbitrary of dimensions/constraints, norm selection

Memory Constrained LB Results



- No LB case works, but slow
- Greedy and regular k -d exceed memory and crash
- k -d-constraint works and give 1.37x speedup

Results for Memory Constrained Scenario

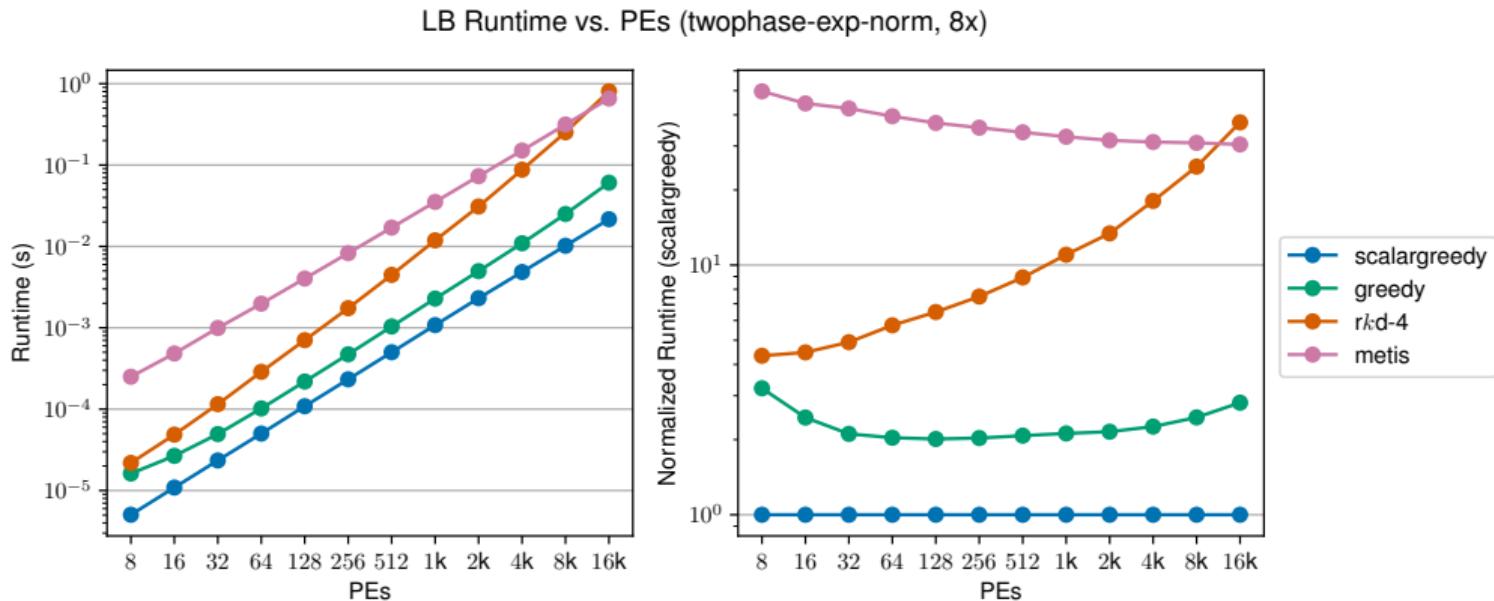
4 nodes on KNL partition of Stampede2, 2D vector (load, memory footprint)

Scaling Vector Load Balancing

Vector LB Performance Overview

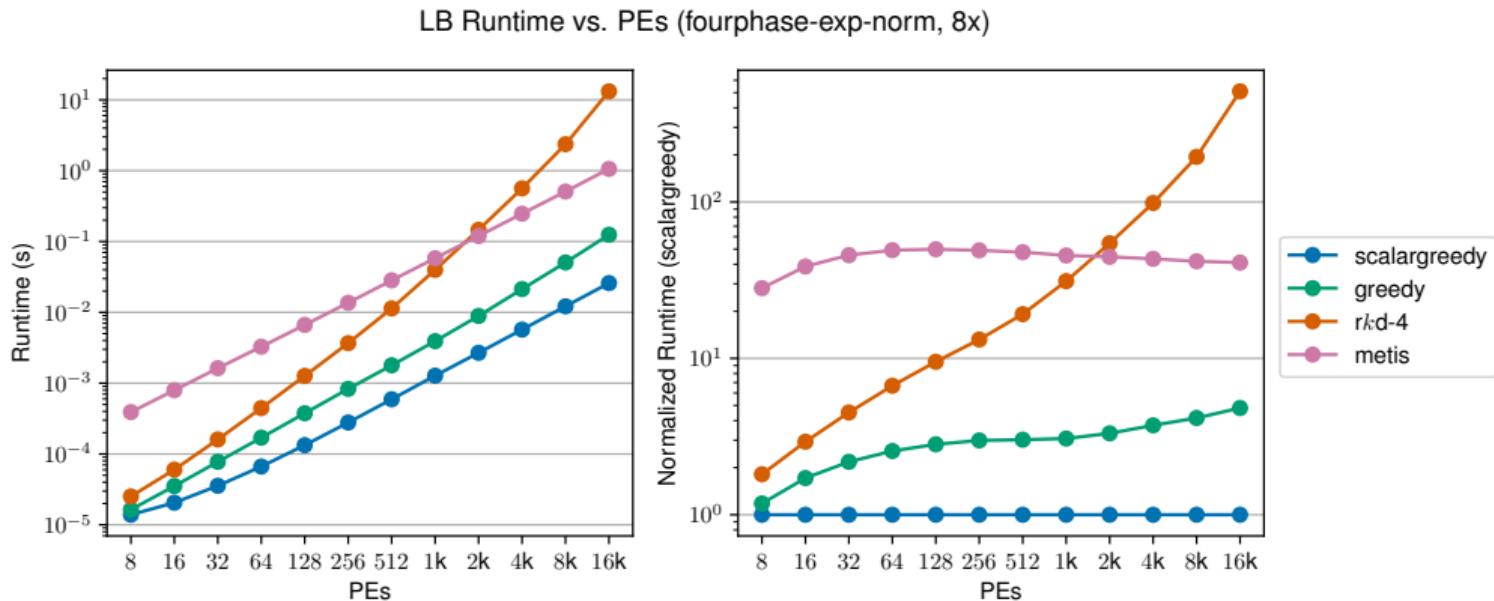
- Vector LB improves quality, but runtime can be long
 - Lack of total order → inefficient data structures
 - Curse of dimensionality
- At scale, large PE counts and even more objects can make performance problematic
 - Can be multiple orders of magnitude slower than scalar

Vector LB - Baseline 2D



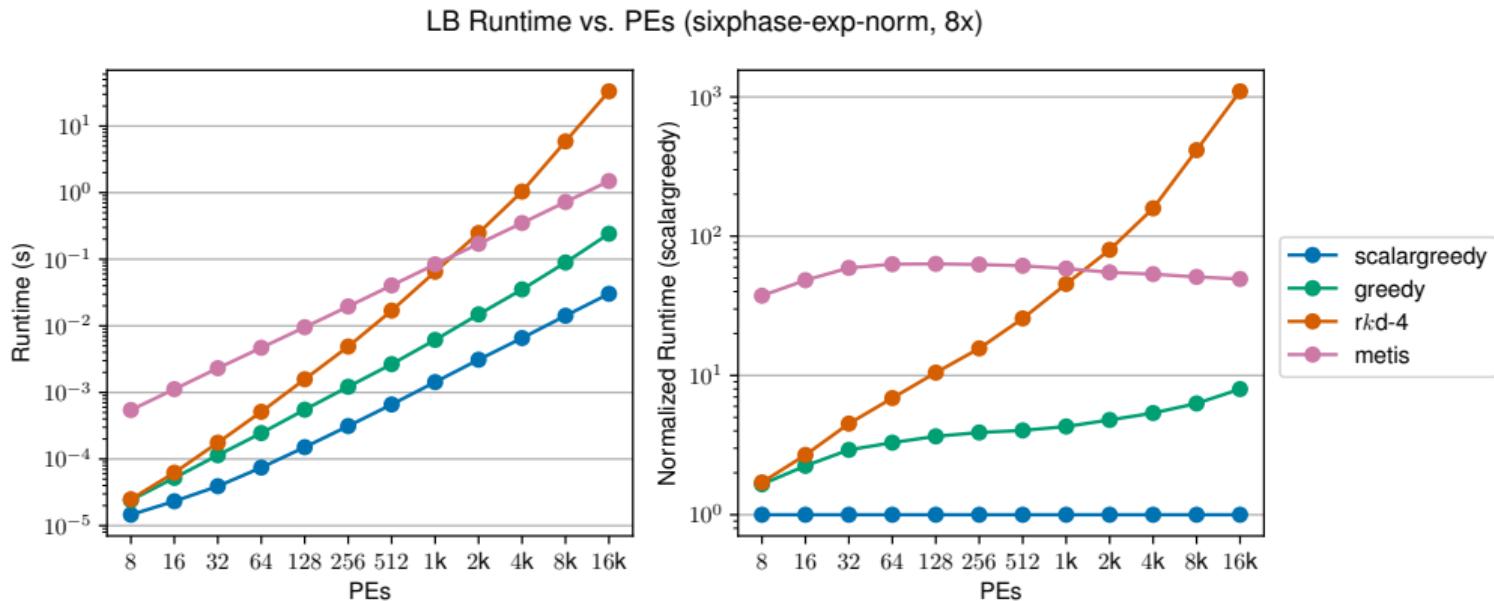
100 trials, Synthetic data: 2D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Vector LB - Baseline 4D



100 trials, Synthetic data: 4D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Vector LB - Baseline 6D



100 trials, Synthetic data: 6D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Vector LB Performance Problems

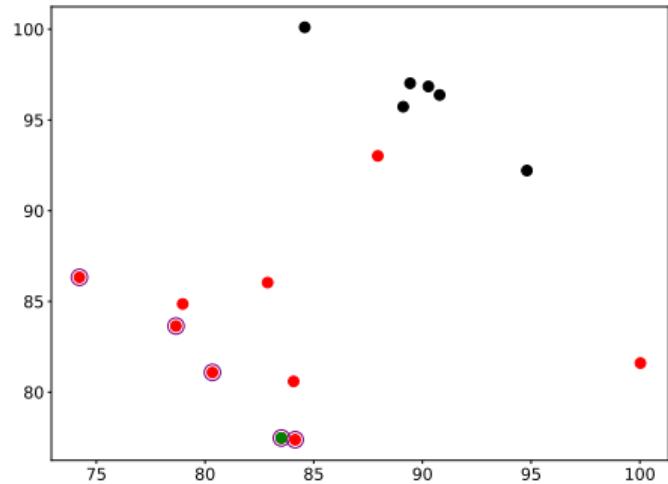
- Previously, we saw robust quality offered by k -d strategy
- However, for 6D case, k -d over 1000x slower than scalar greedy, over 10x slower than METIS

Vector LB Performance Problems

- Previously, we saw robust quality offered by k -d strategy
- However, for 6D case, k -d over 1000x slower than scalar greedy, over 10x slower than METIS
- Study three optimizations:
 - Pareto search
 - Early exit
 - Hierarchical execution

Pareto Search

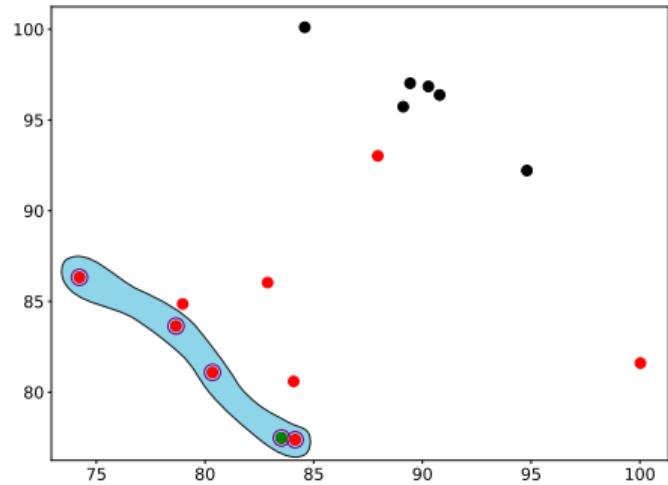
- Norm-minimizing PEs will always be in Pareto frontier of PEs
- Reduce search space by restricting to Pareto frontier
 - No change in results



Iteration of rk-d Search

Pareto Search

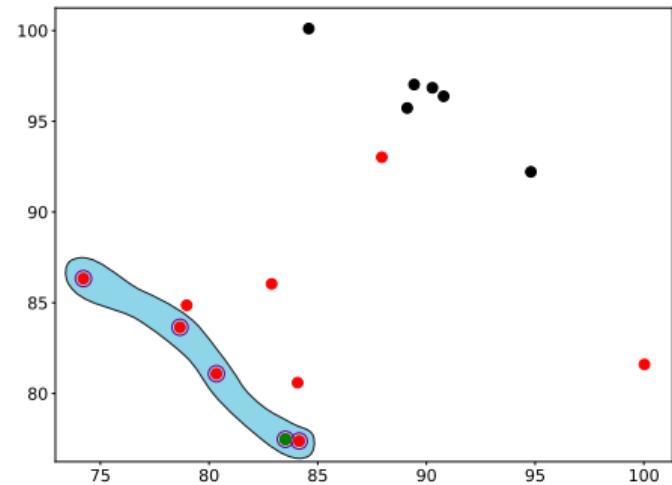
- Norm-minimizing PEs will always be in Pareto frontier of PEs
- Reduce search space by restricting to Pareto frontier
 - No change in results



Iteration of rk-d Search

Pareto Search

- Norm-minimizing PEs will always be in Pareto frontier of PEs
- Reduce search space by restricting to Pareto frontier
 - No change in results



Iteration of $rk\text{-}d$ Search

Reduce from 10 to ≤ 5 PEs searched

Pareto Search

Algorithm 1: rk-d Pareto Algorithm

Input: Set of objects O , set of PEs P

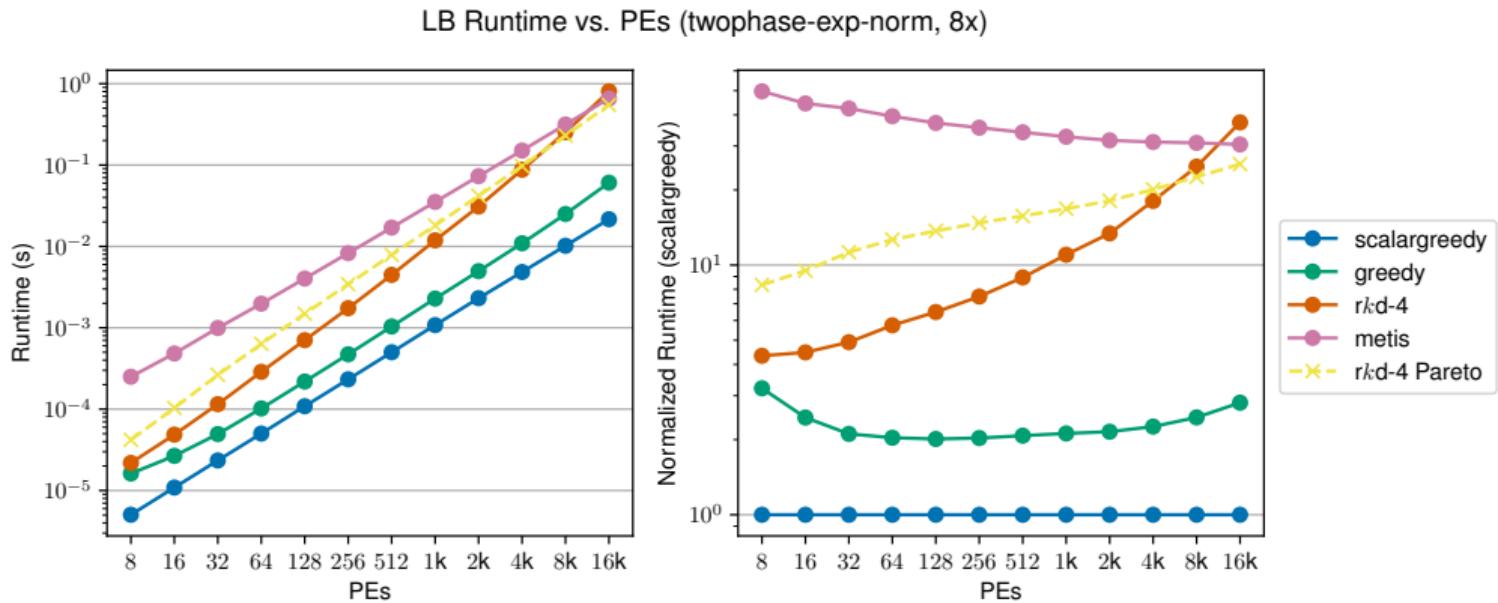
Output: New mapping in sol

```
1   $T \leftarrow MakeTree(P);$ 
2   $F \leftarrow MakeFrontier(T);$ 
3  forall  $o \in sorted(O)$  do
4       $p_{min} \leftarrow FindMinNormPE(F, o);$                                 /* Search only through frontier F */
5       $T \leftarrow Remove(T, p_{min});$ 
6       $F \leftarrow Remove(F, p_{min});$ 
7       $nn \leftarrow NearestNeighbor(F, p_{min});$ 
8       $sol.Assign(o, p_{min});$ 
9       $T \leftarrow Add(T, p_{min});$ 
10      $F \leftarrow UpdateFrontier(T, nn);$ 
11 end
```

Test Details

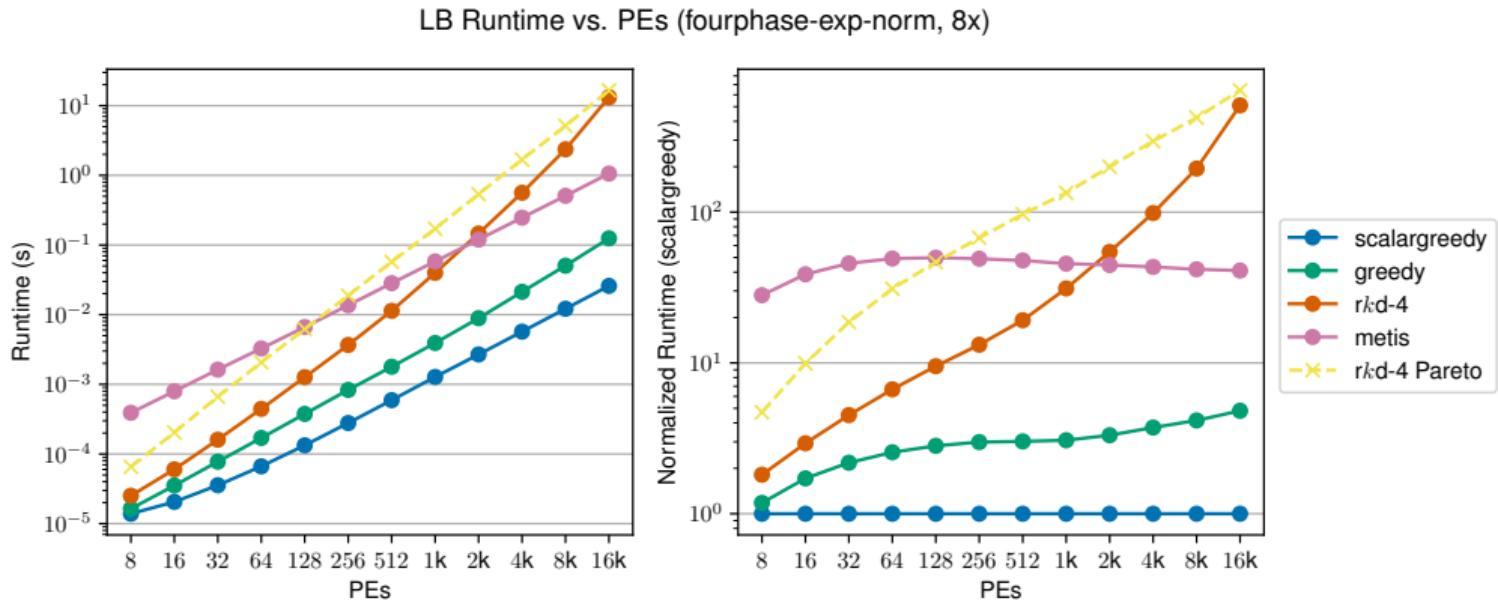
- All tests conducted on dedicated 4-core node with ARM Ampere A1 at 3 GHz
- Compiled with gcc 11.3.0 with -O3 and the -ffast-math
 - Floating-point precision loss is acceptable
- Quality evaluations use the sum-based objective function (used for phase-based LB)

Pareto Search Runtime - 2D



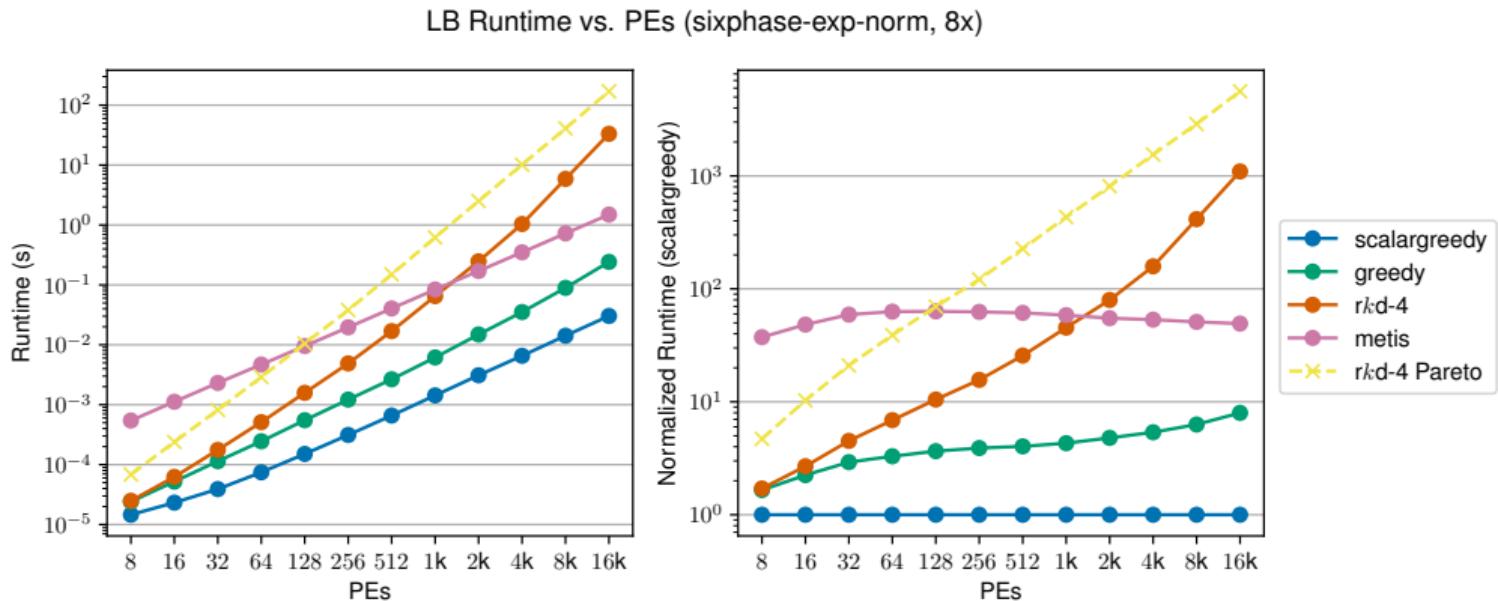
100 trials, Synthetic data: 2D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Pareto Search Runtime - 4D



100 trials, Synthetic data: 4D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Pareto Search Runtime - 6D



100 trials, Synthetic data: 6D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Pareto Search Performance

- Only beneficial at large scale and low dimensionality (e.g. 2D at 16k PEs)
- Progressively slower with increasing dimensionality
- Improves as scaling continues, but absolute time still large

Strategy	Runtime (s)		
	2D	4D	6D
<i>rk-d</i>	0.81	13.19	33.22
<i>rk-d Pareto</i>	0.55	16.60	170.14

rk-d/Pareto Runtime, 16k PEs, (Exp., Norm.) Load Vectors

Pareto Search Performance

- Only beneficial at large scale and low dimensionality (e.g. 2D at 16k PEs)
- Progressively slower with increasing dimensionality
- Improves as scaling continues, but absolute time still large

Strategy	Runtime (s)		
	2D	4D	6D
rk-d	0.81	13.19	33.22
rk-d Pareto	0.55	16.60	170.14

rk-d/Pareto Runtime, 16k PEs, (Exp., Norm.) Load Vectors

Search time reduces, but cost of maintaining frontier too high

Early Exit Motivation

- In regular k -d strategy, we search whole space to find norm-minimizing PE
 - Prune where possible, but still have to search many PEs
 - Worse as dimension increases

Early Exit Motivation

- In regular k -d strategy, we search whole space to find norm-minimizing PE
 - Prune where possible, but still have to search many PEs
 - Worse as dimension increases
- Quality determined not by any one PE, but maximum load vector across all

Early Exit Motivation

- In regular k -d strategy, we search whole space to find norm-minimizing PE
 - Prune where possible, but still have to search many PEs
 - Worse as dimension increases
- Quality determined not by any one PE, but maximum load vector across all
- If we find a PE that does not increase the maximum load vector, stop searching and return!

Early Exit Design

- Continue to search for norm-minimizing PE, adopt new candidate when it beats current best

Early Exit Design

- Continue to search for norm-minimizing PE, adopt new candidate when it beats current best
- Add new *limit* parameter

Early Exit Design

- Continue to search for norm-minimizing PE, adopt new candidate when it beats current best
- Add new *limit* parameter
 - Determines how thoroughly to search the space
 - If adopted candidate does not increase max. load vector, decrement; do early exit when $limit = 0$
 - Tradeoff between time and quality

Early Exit Design

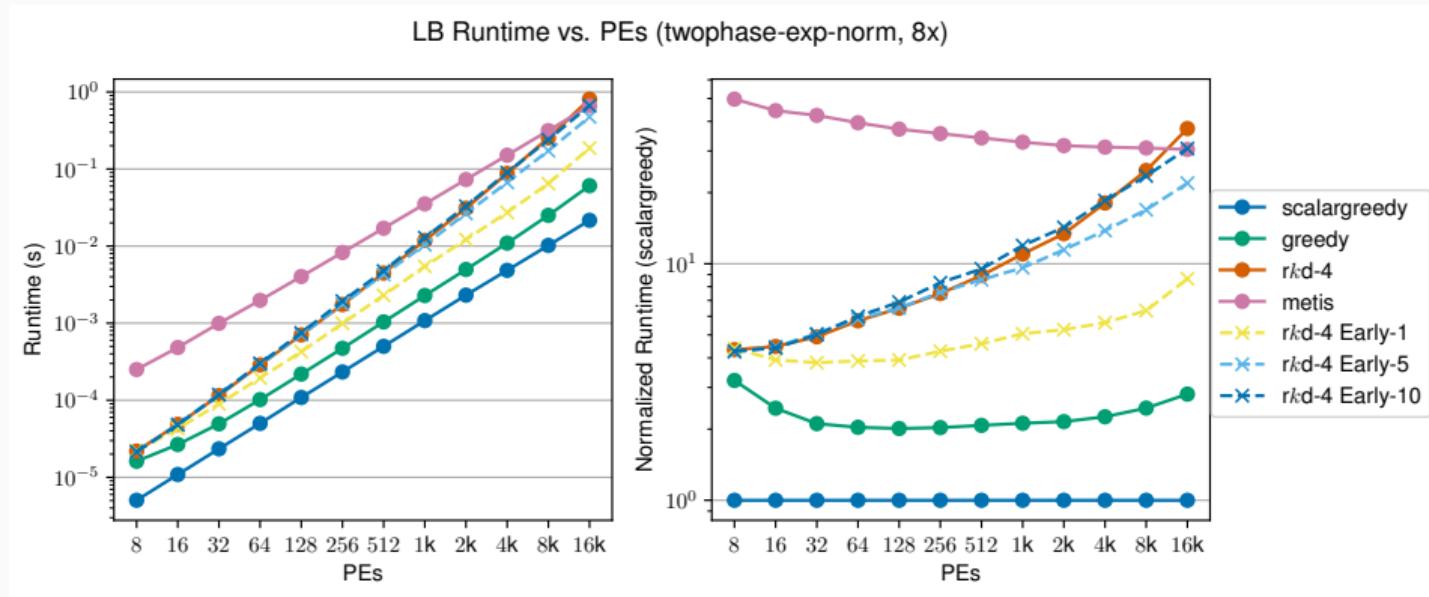
- Continue to search for norm-minimizing PE, adopt new candidate when it beats current best
- Add new *limit* parameter
 - Determines how thoroughly to search the space
 - If adopted candidate does not increase max. load vector, decrement; do early exit when $limit = 0$
 - Tradeoff between time and quality
- Test with $limit = 1, 5, 10$

Early Exit Algorithm

Algorithm 2: rk-d Early Exit Algorithm

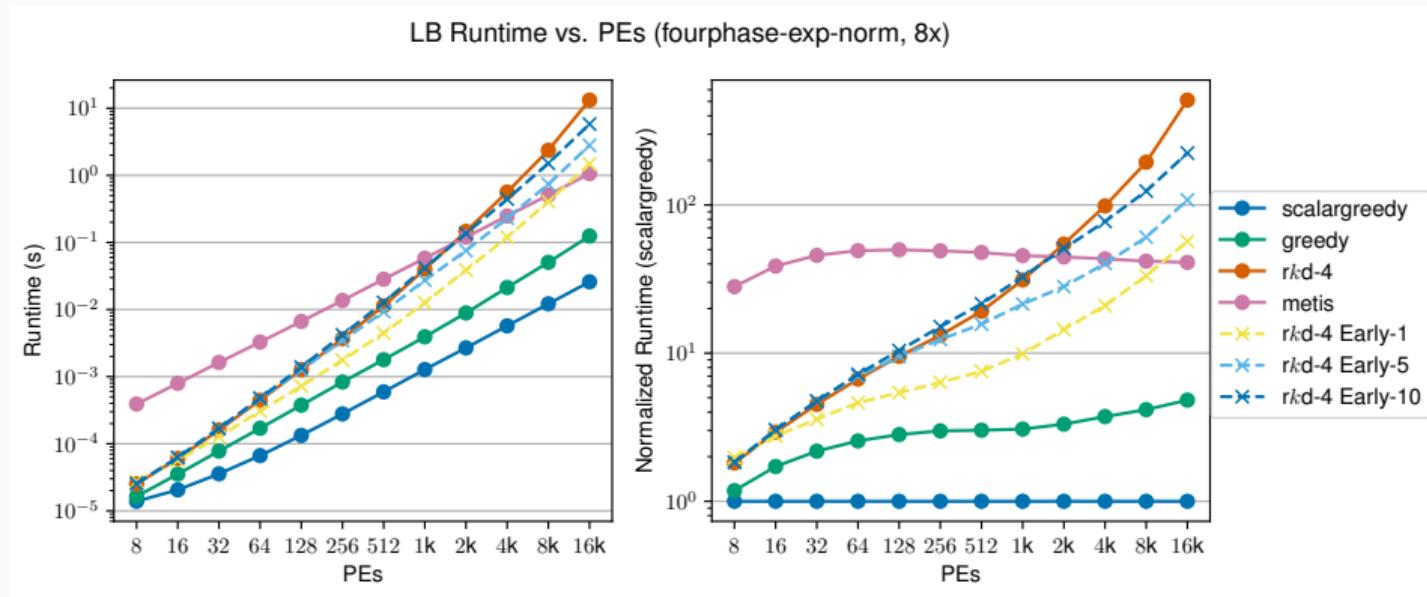
```
1  Function FindMinNormPEEEarly(tree, o, limit, bounds = ⟨0, ..., 0⟩):
2      if tree.left ≠ NULL then
3          |   pbest, limit ← FindMinNormPEEEarly(tree.left, o, limit, bounds);
4      end
5      if limit > 0 ∧ ‖tree.data + o‖ < normbest then
6          |   normbest ← ‖tree.data + o‖;
7          |   pbest ← tree.data;
8          |   if (tree.data + o)[i] ≤ loadmax[i] ∀i ∈ 1, ..., d then
9              |       |   limit ← limit - 1;                                /* Candidate found, so reduce limit */
10             end
11         end
12         if limit > 0 ∧ tree.right ≠ NULL then
13             |   oldBound ← bounds[tree.dim];
14             |   bounds[tree.dim] ← tree.data[tree.dim];
15             |   if ‖bounds + o‖ < normbest then
16                 |       |   pbest, limit ← FindMinNormPEEEarly(tree.right, o, limit, bounds);
17             end
18             |   bounds[tree.dim] ← oldBound;
19         end
20     return pbest, limit
```

Early Exit Runtime - 2D



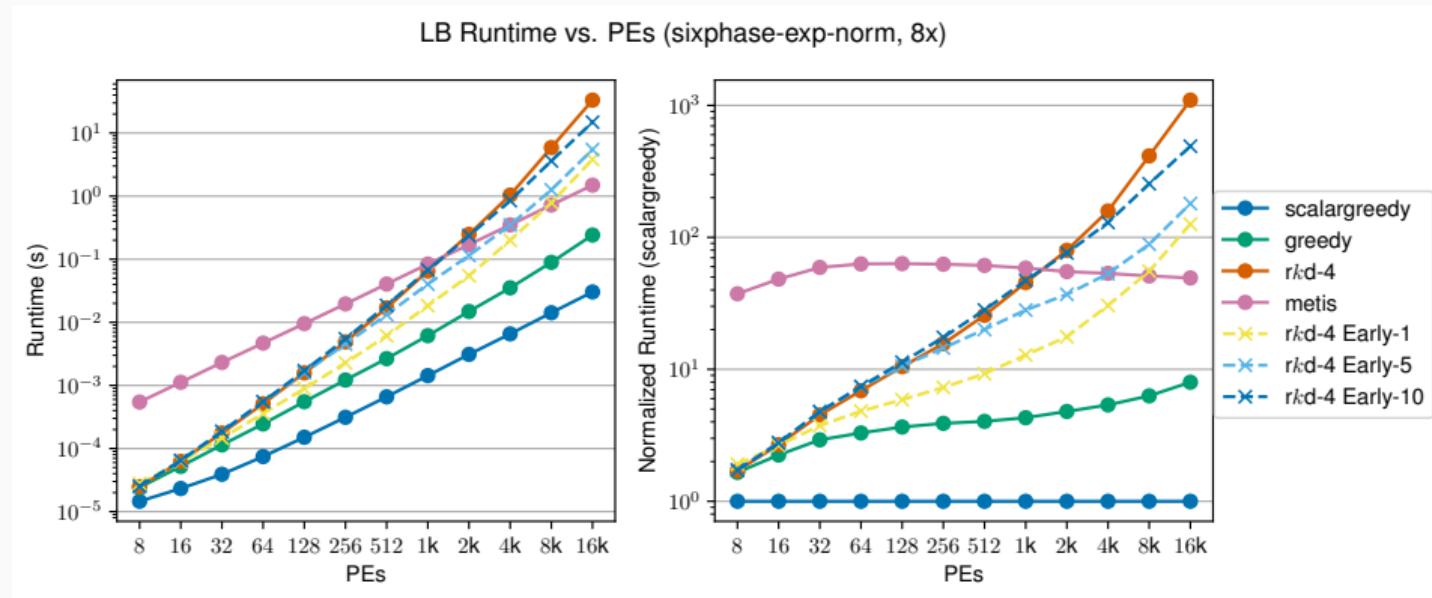
100 trials, Synthetic data: 2D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Early Exit Runtime - 4D



100 trials, Synthetic data: 4D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Early Exit Runtime - 6D



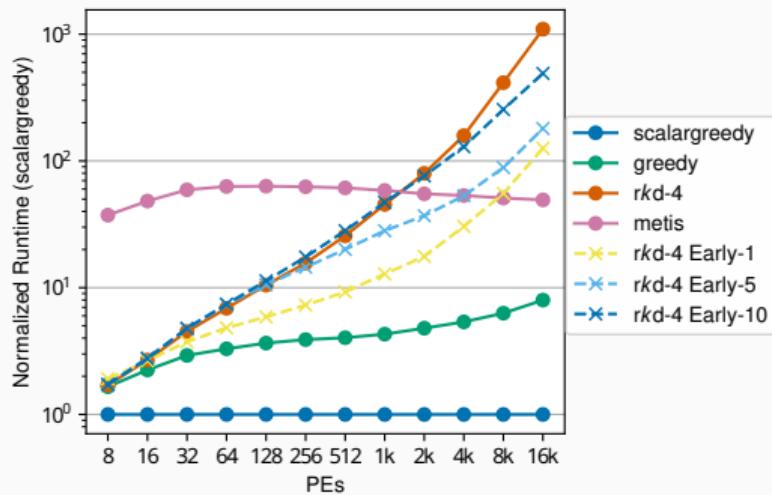
100 trials, Synthetic data: 6D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Early Exit Performance

- Improves runtime significantly
 - Benefit increases as dimension increases
 - Almost order of magnitude faster for 6D
- Performance improvement depends on load distribution

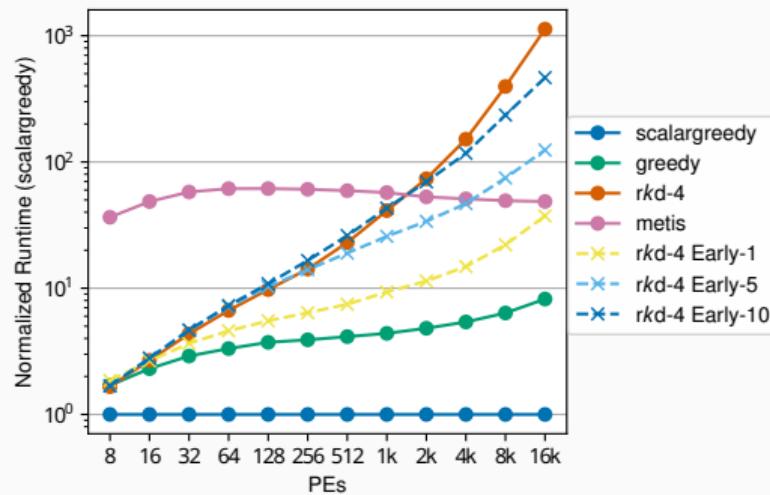
Early Exit Runtime - Load Dependence

LB Runtime vs. PEs (sixphase-exp-norm, 8x)



6D (Exponentially, Normally) Distributed

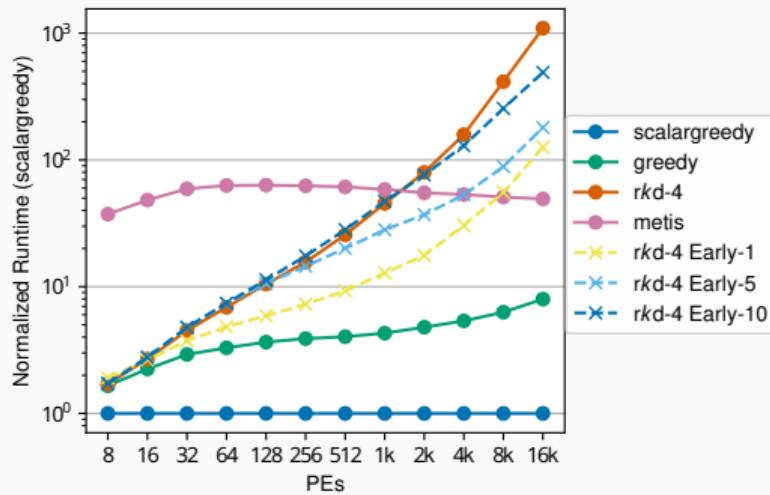
LB Runtime vs. PEs (sixphase-normal, 8x)



6D (Normally) Distributed

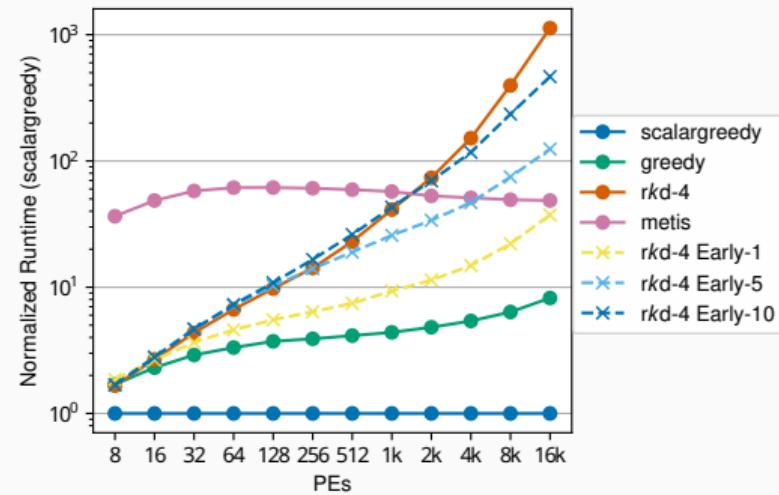
Early Exit Runtime - Load Dependence

LB Runtime vs. PEs (sixphase-exp-norm, 8x)



6D (Exponentially, Normally) Distributed

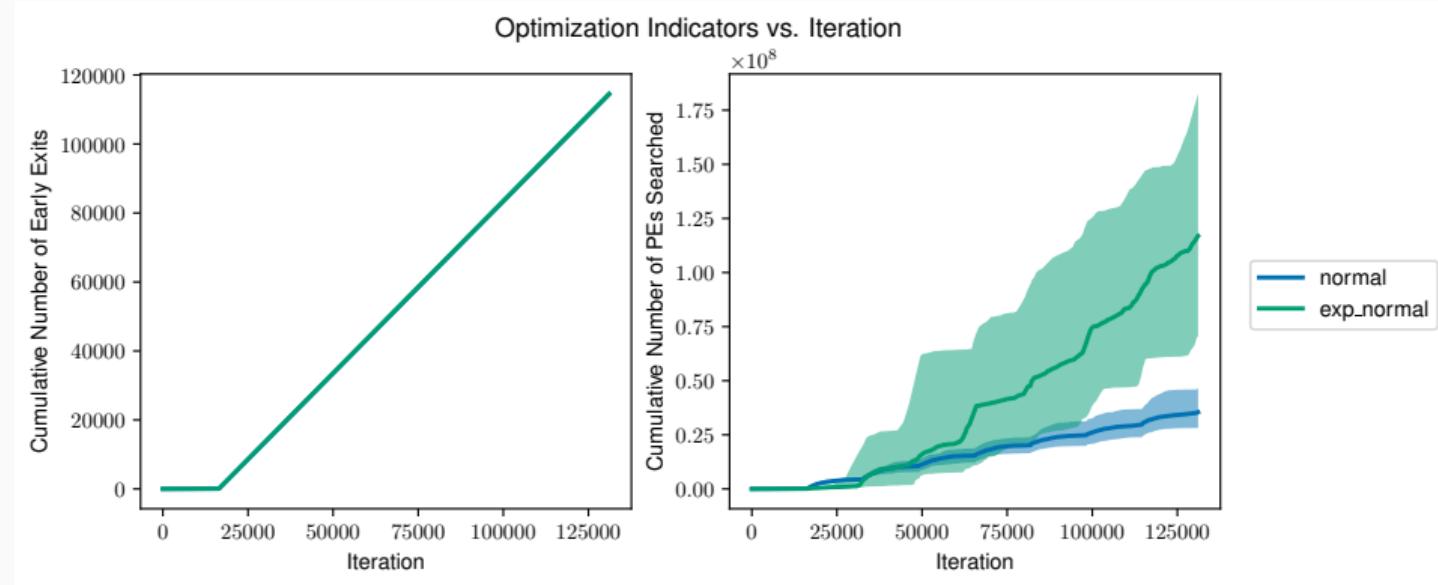
LB Runtime vs. PEs (sixphase-normal, 8x)



6D (Normally) Distributed

At 16k PEs, Early-1 beats METIS for (normal) case, loses for (exponential, normal)

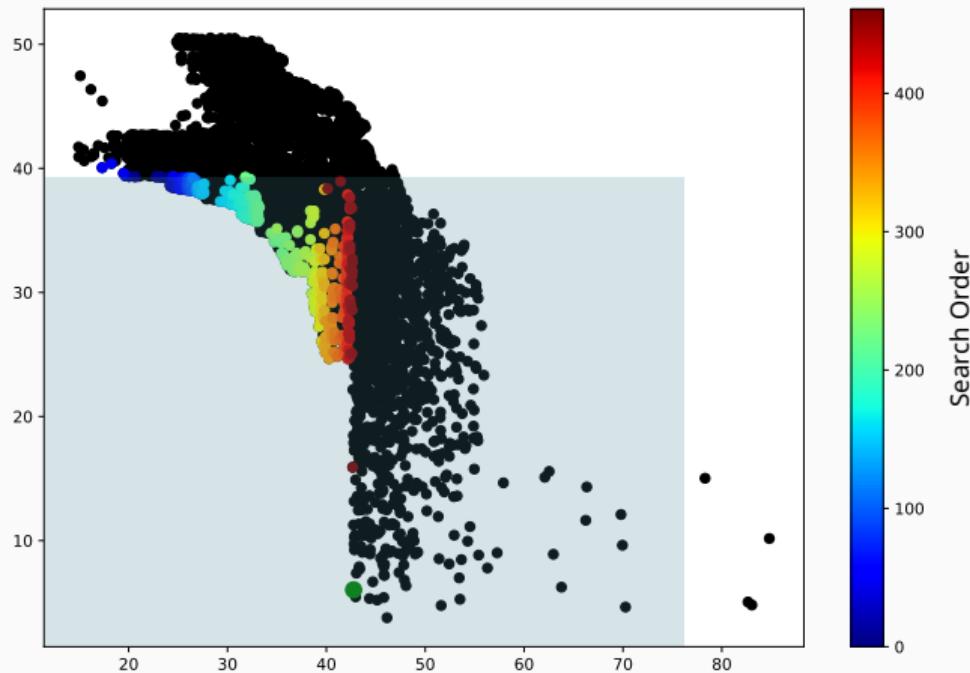
Early Exit Performance Indicators



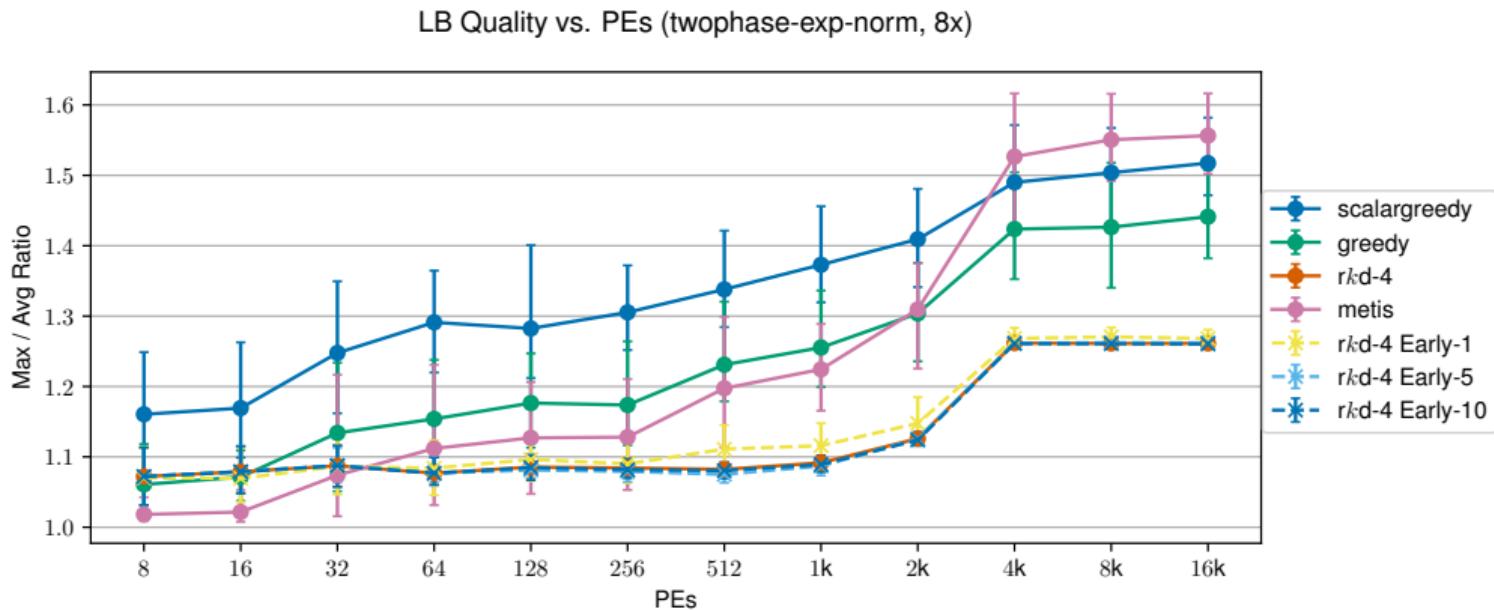
Comparison of Cumulative Early Exit Performance Indicators for 6-Normally and 6-(Exponentially, Normally) Distributed Vectors, 16k PEs

Early Exit Anomalous Search

May thoroughly search
“smaller” or less
variable dimensions
when dimensional
distributions are
different

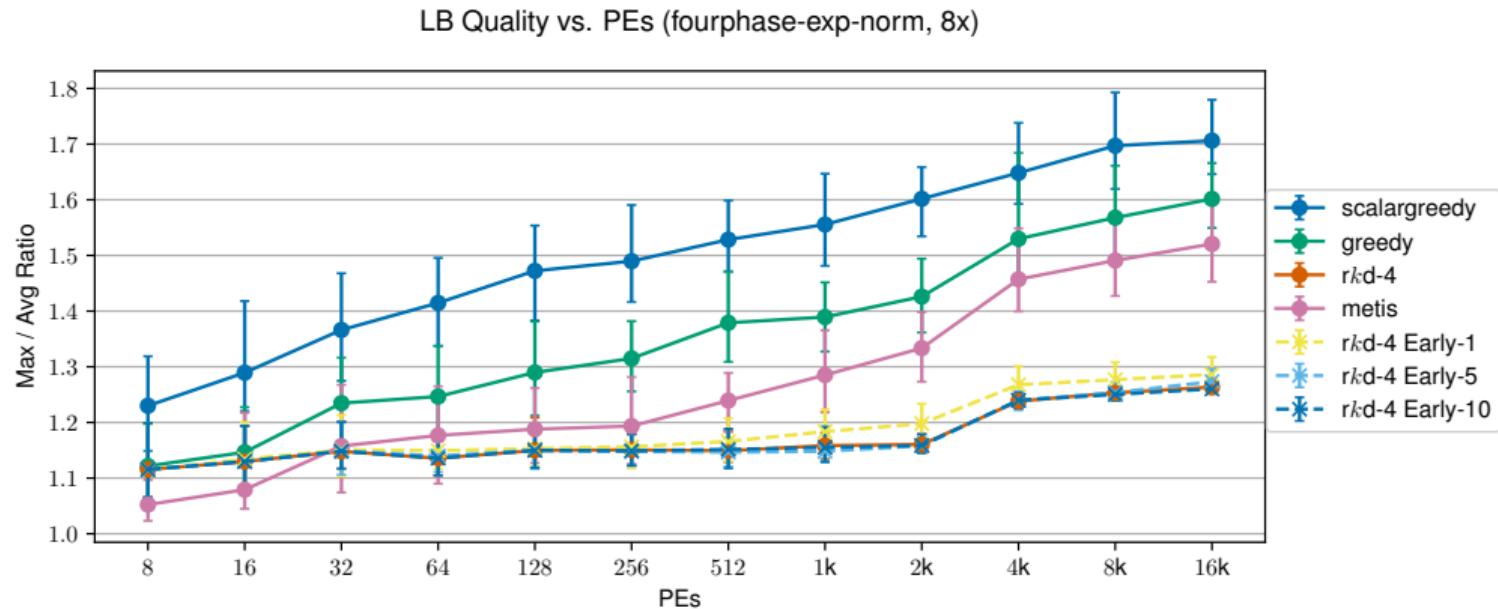


Early Exit Quality - 2D



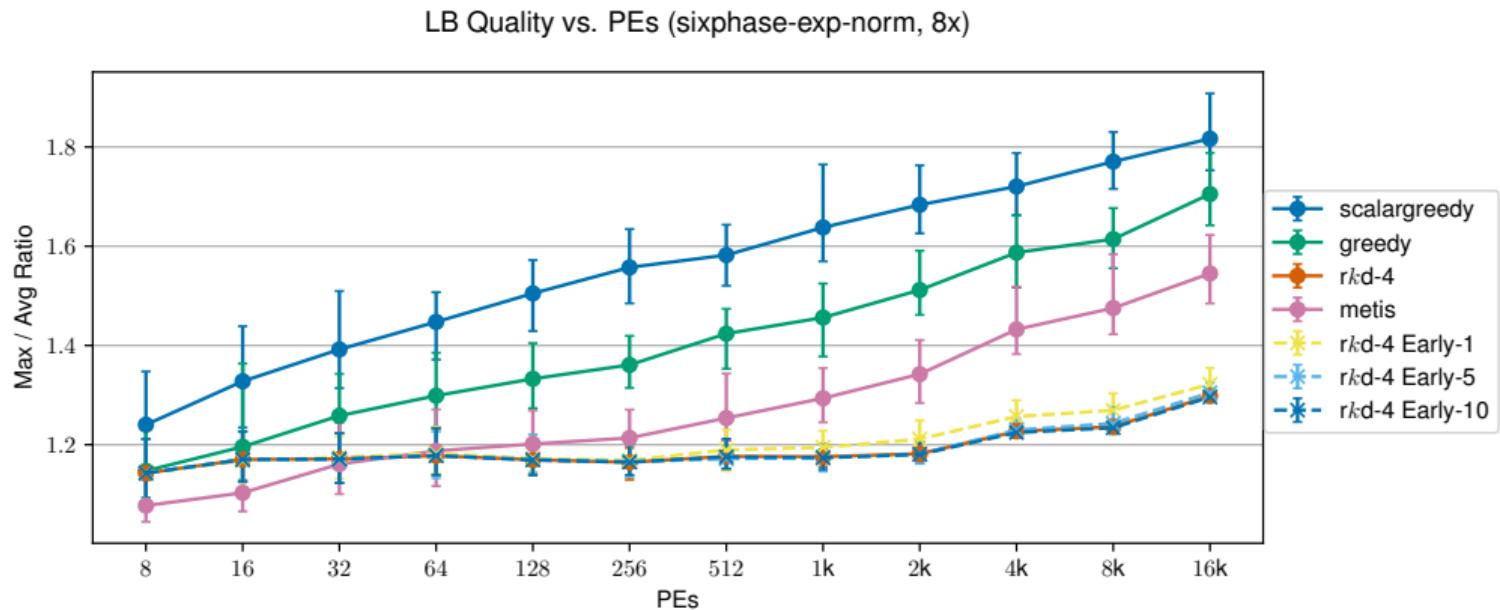
100 trials, Synthetic data: 2D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Early Exit Quality - 4D



100 trials, Synthetic data: 4D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Early Exit Quality - 6D



100 trials, Synthetic data: 6D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Early Exit Quality Table

Strategy	PEs											
	8	16	32	64	128	256	512	1K	2K	4K	8K	16K
rk-d	<i>Min</i>	0.94	0.93	0.94	0.96	0.97	0.97	0.98	0.98	0.98	0.98	0.98
	<i>Median</i>	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	<i>Max</i>	1.10	1.08	1.06	1.07	1.06	1.04	1.04	1.03	1.05	1.05	1.05
Early-1	<i>Min</i>	0.94	0.92	0.91	0.97	0.98	0.96	0.98	0.98	0.97	0.99	0.99
	<i>Median</i>	1.00	1.00	1.01	1.01	1.01	1.01	1.02	1.02	1.02	1.01	1.02
	<i>Max</i>	1.09	1.09	1.11	1.13	1.13	1.13	1.14	1.15	1.10	1.12	1.10
Early-5	<i>Min</i>	0.94	0.93	0.93	0.96	0.97	0.96	0.98	0.98	0.98	0.98	0.98
	<i>Median</i>	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	<i>Max</i>	1.10	1.08	1.05	1.06	1.06	1.06	1.07	1.07	1.07	1.06	1.07
Early-10	<i>Min</i>	0.94	0.93	0.94	0.95	0.97	0.97	0.98	0.97	0.98	0.98	0.98
	<i>Median</i>	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	<i>Max</i>	1.10	1.08	1.06	1.07	1.06	1.04	1.04	1.04	1.05	1.04	1.04

Quality of Early Exit with Varying *limit* Normalized to Median of rk-d (All Tests)

Early Exit Summary

- Large performance improvements, especially at scale

Early Exit Summary

- Large performance improvements, especially at scale
 - Over an order of magnitude at most
 - Can depend on load distribution

Early Exit Summary

- Large performance improvements, especially at scale
 - Over an order of magnitude at most
 - Can depend on load distribution
- Degradation in quality, but minor

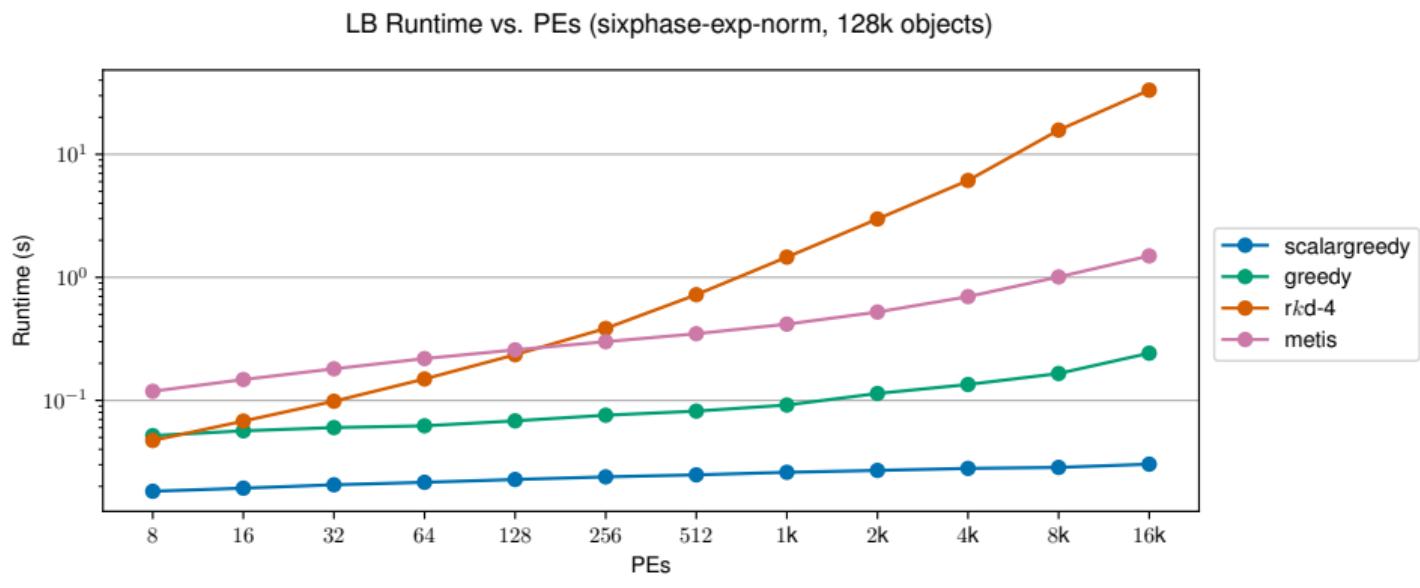
Early Exit Summary

- Large performance improvements, especially at scale
 - Over an order of magnitude at most
 - Can depend on load distribution
- Degradation in quality, but minor
 - Across all tests, bounded by 1.15x at worst, 1.02x on average

Early Exit Summary

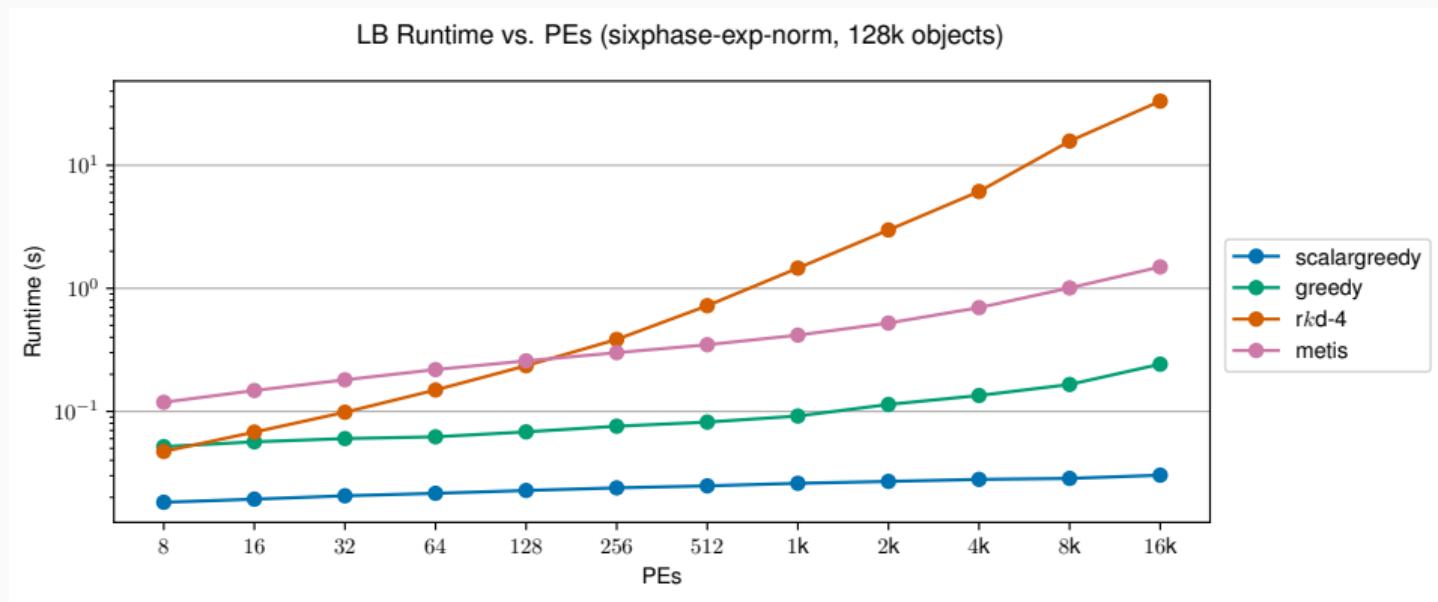
- Large performance improvements, especially at scale
 - Over an order of magnitude at most
 - Can depend on load distribution
- Degradation in quality, but minor
 - Across all tests, bounded by 1.15x at worst, 1.02x on average
- $limit = 1$ best choice for most

Hierarchical Motivation - Fixed # Objects



Runtime with 6D (Exponentially, Normally) Distributed Vectors, 128k Objects

Hierarchical Motivation - Fixed # Objects

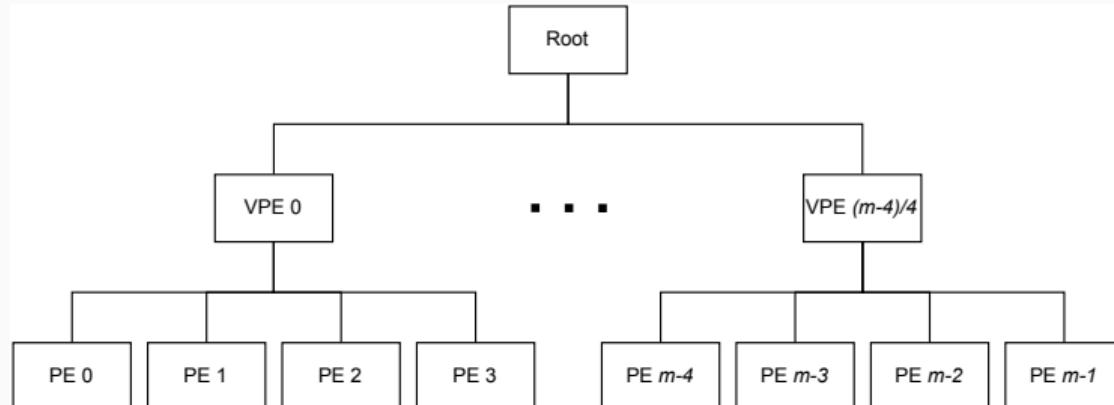


Runtime with 6D (Exponentially, Normally) Distributed Vectors, 128k Objects

Vector LBs most affected by change in PE count (especially rkd-4)

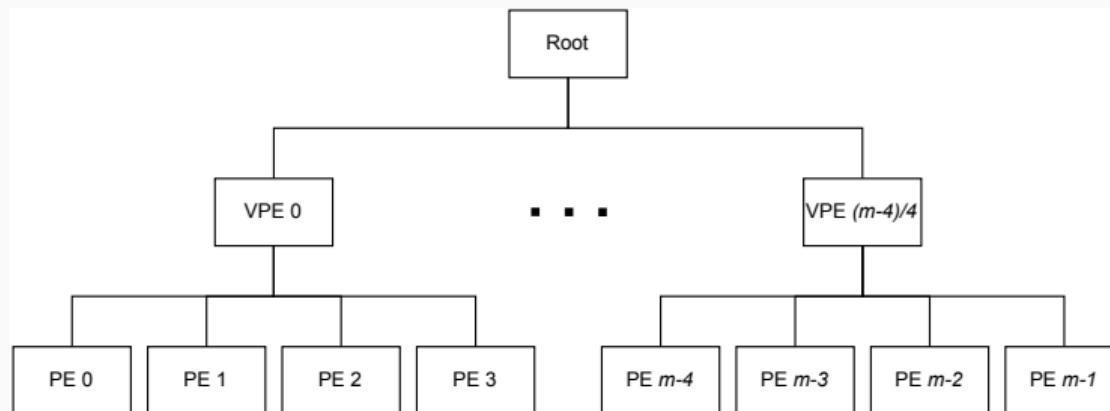
Hierarchical Load Balancing

- Decompose problem hierarchically, run LB at each level
- Coarsen PEs on upward pass, uncoarsen coming down
- Root assigns to small number of coarsened “virtual” PEs, intermediate nodes remap to actual PEs in subtree



Hierarchical Load Balancing

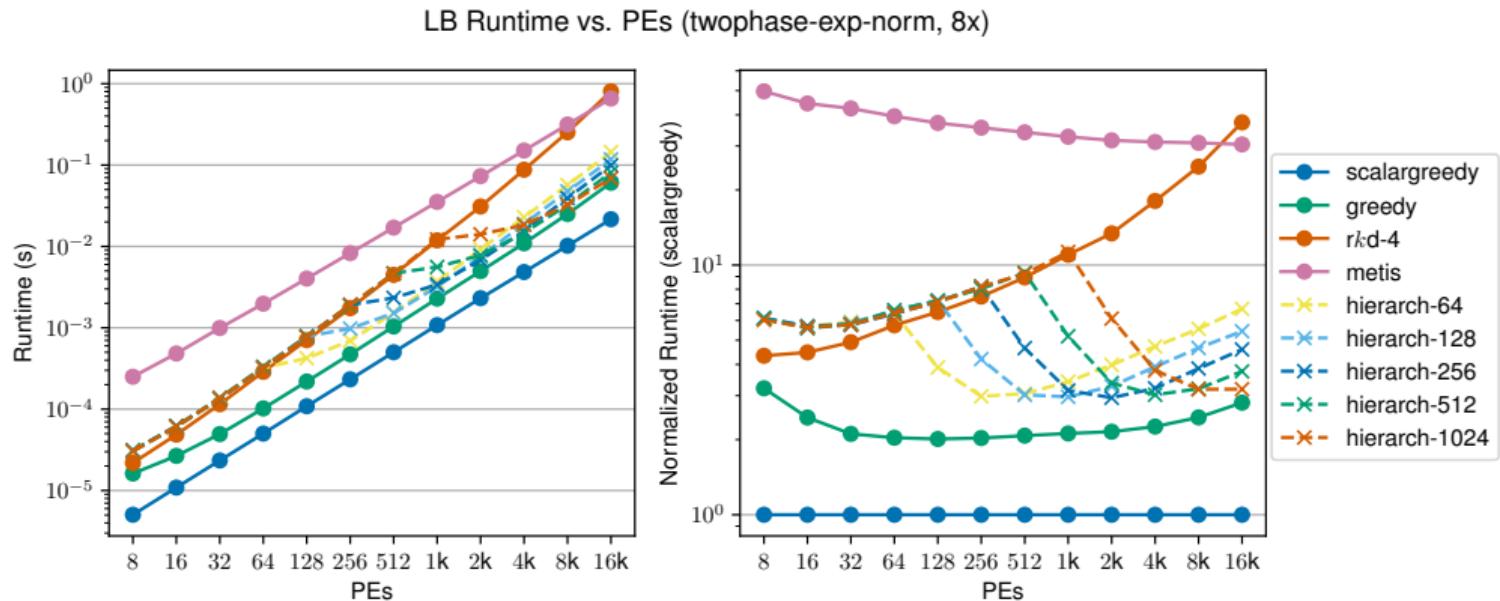
- Decompose problem hierarchically, run LB at each level
- Coarsen PEs on upward pass, uncoarsen coming down
- Root assigns to small number of coarsened “virtual” PEs, intermediate nodes remap to actual PEs in subtree
 - Intermediate LBs can run in parallel



Hierarchical Configuration

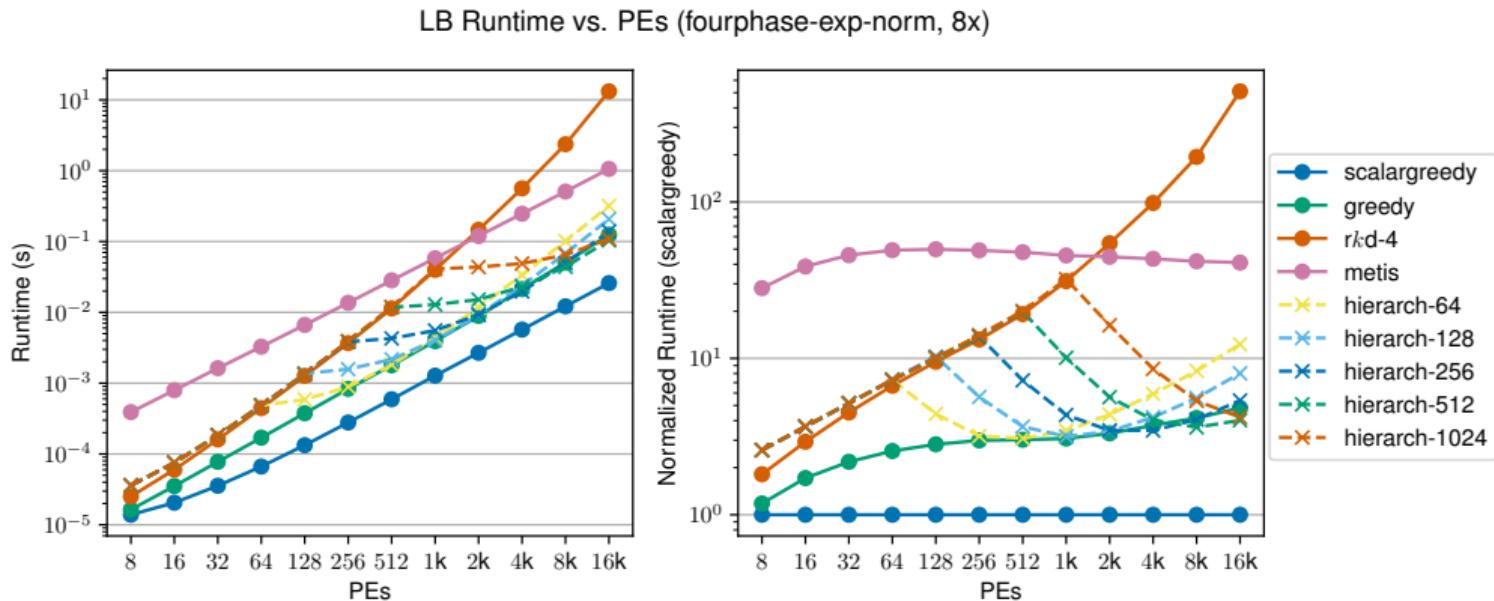
- Three level tree
- $rk-d-4$ at root and intermediate levels of tree
- Branching factor varies from 64-1024

Hierarchical LB Runtime - 2D



100 trials, Synthetic data: 2D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

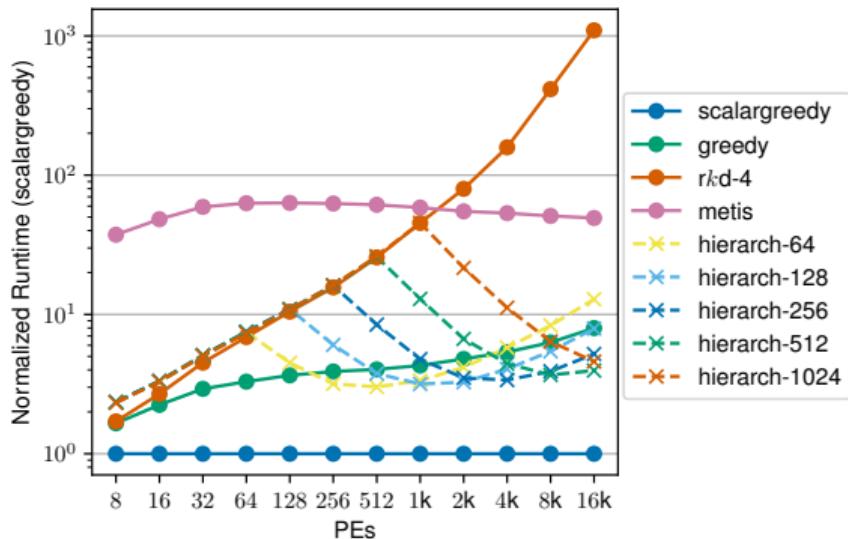
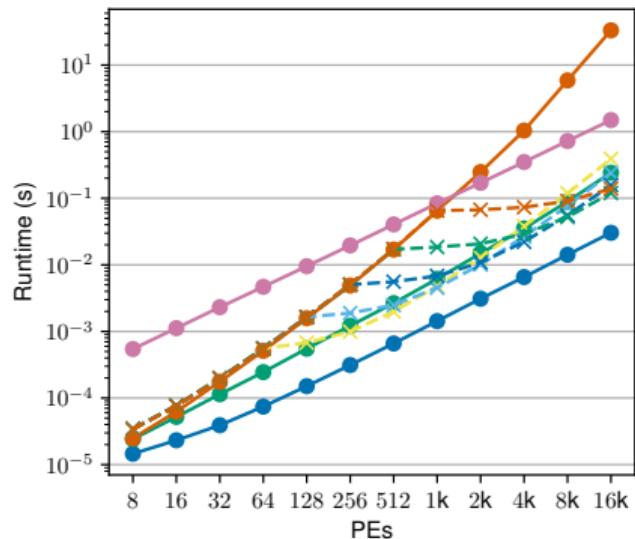
Hierarchical LB Runtime - 4D



100 trials, Synthetic data: 4D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Hierarchical LB Runtime - 6D

LB Runtime vs. PEs (sixphase-exp-norm, 8x)



100 trials, Synthetic data: 6D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

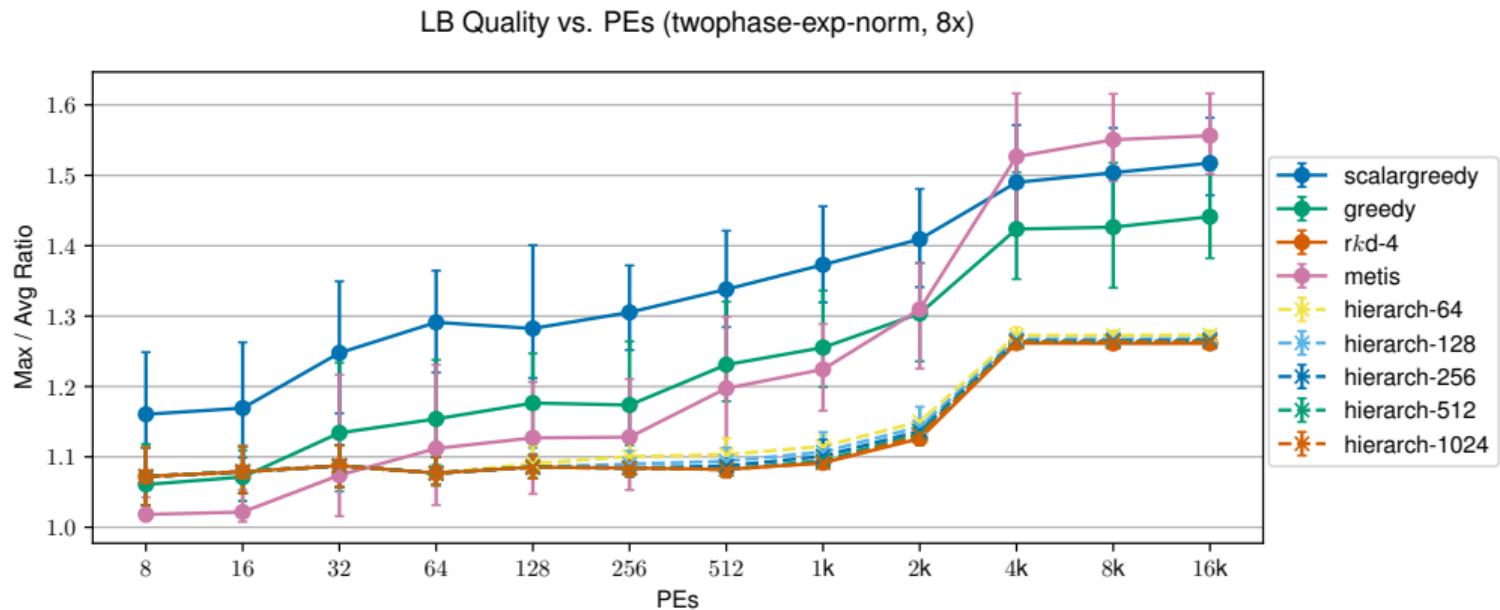
Hierarchical Performance Summary

- Fastest of all tested optimizations by far
 - Within roughly one order of magnitude of scalar greedy
 - Benefit increases with dimensionality
- Branching factor “sweet spot” depends on PE count

Hierarchical Performance Summary

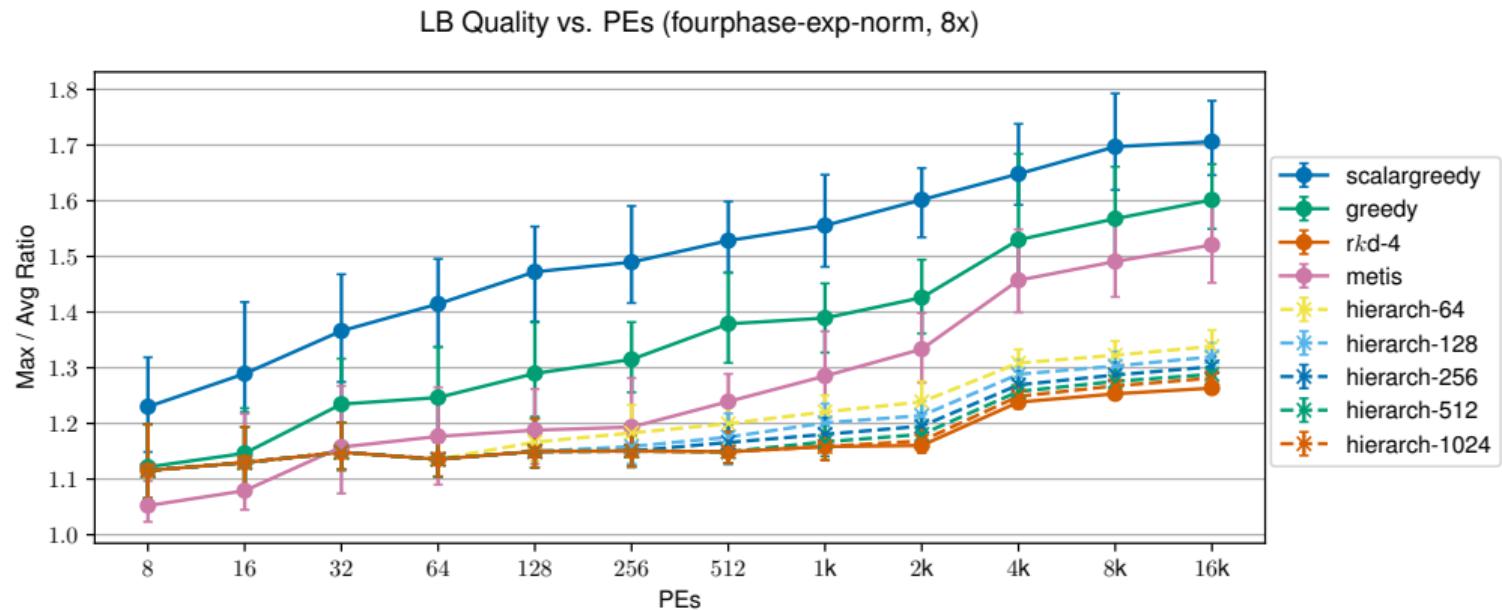
- Fastest of all tested optimizations by far
 - Within roughly one order of magnitude of scalar greedy
 - Benefit increases with dimensionality
- Branching factor “sweet spot” depends on PE count
- Impacts quality, since less “global” information is available for each instance of LB

Hierarchical Quality - 2D



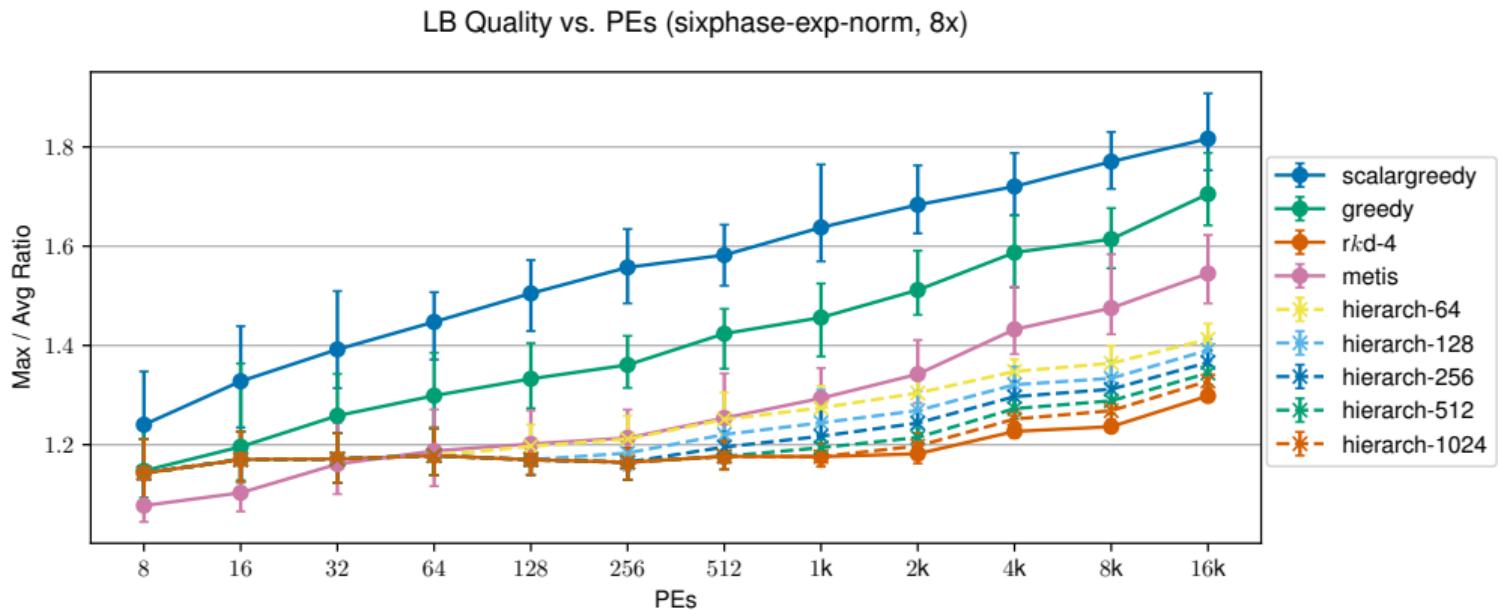
100 trials, Synthetic data: 2D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Hierarchical Quality - 4D



100 trials, Synthetic data: 4D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Hierarchical Quality - 6D



100 trials, Synthetic data: 6D ($\exp \lambda = 0.15$, normal $\mu = 10, \sigma = 3$)

Hierarchical Quality Summary

- Larger branching factors provide better quality
 - Better information for final assignments to PEs at intermediate nodes
- Degradation in quality increases with both dimensionality and scale
 - Can be tuned via branching factor
- In all cases, still better quality than next best LB

Scaling Vector LB Summary

- Varied degree of success with optimizations
 - Pareto too limited to be useful, often worse than original
 - Early Exit gave good benefits without much quality degradation
 - Hierarchical fastest of all, can degrade quality, but degree is tunable via parameters
- Significant performance improvement makes vector LB tenable at large scale
- Optimizations are mutually compatible, can compose in future if needed

Future Directions & Conclusions

Future Directions

- Add vector support to more types of balancers
 - Geometrical, migration-aware, communication-aware, distributed all in progress
- Preprocessing load vectors for dimensionality reduction
- ML to select relevant metrics for inclusion in vector
- Package for wider public use
 - Already usable in Charm++, {A}MPI, VT

Conclusions

- Load characteristics of complex, modern applications cannot be captured in a single scalar, but vector suffices
- New strategies can utilize the additional detail provided by a load vector for different objectives/constraints
- Vector LB improves over scalar LB across several classes of synthetic and production applications
- Runtime can be slow, but optimizations are effective

Questions?

rabuch2@illinois.edu