Vector Load Balancing in Charm++

Ronak Buch

October 19, 2022

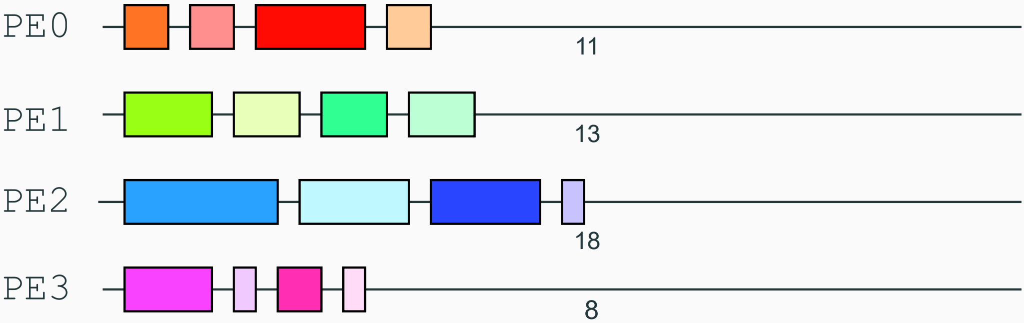
Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

Background and Motivation

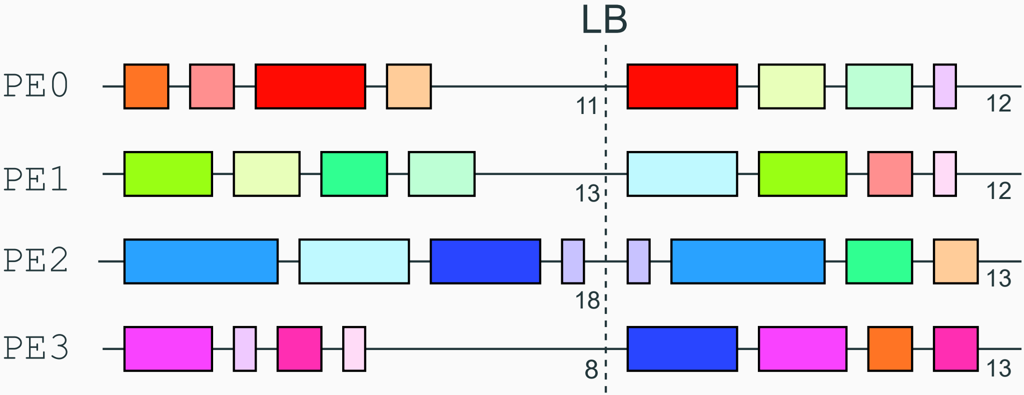
Dynamic Load Balancing

- Adaptively rearrange work amongst PEs to maximize performance as computation evolves
- Most loaded PE usually determines iteration time
- Enabled by migratable objects, runtime instrumentation, overdecomposition
- Necessary for scaling all but very regular, static applications

Before LB



After LB



Scalar Load

- To measure load traditionally:
  - Start timer when object begins execution
  - End timer when control returns to RTS
  - Elapsed time added to object's load
  - PE's load is sum of resident objects' loads

Scalar Load

- To measure load traditionally:
  - Start timer when object begins execution
  - End timer when control returns to RTS
  - Elapsed time added to object's load
  - PE's load is sum of resident objects' loads
- Only captures time object spends active
  - CPU time alone does not determine performance
  - Deficient for many classes of applications

Scalar Load Deficiencies

- Phase-based applications
  - Iteration divided into several orthogonal phases
  - Time spent in different phases not fungible
  - Scalar loses temporal execution constraints

Scalar Load Deficiencies

- Phase-based applications
  - Iteration divided into several orthogonal phases
  - Time spent in different phases not fungible
  - Scalar loses temporal execution constraints
- Resource constrained applications
  - Memory footprint, FD limit, cache working set size, ...

Scalar Load Deficiencies

- Phase-based applications
  - Iteration divided into several orthogonal phases
  - Time spent in different phases not fungible
  - Scalar loses temporal execution constraints
- Resource constrained applications
  - Memory footprint, FD limit, cache working set size, ...
- Asynchronous computation
  - Object invokes and waits for GPU kernel, async I/O, ...

Scalar Load Deficiencies

- Phase-based applications
  - Iteration divided into several orthogonal phases
  - Time spent in different phases not fungible
  - Scalar loses temporal execution constraints
- Resource constrained applications
  - Memory footprint, FD limit, cache working set size, ...
- Asynchronous computation
  - Object invokes and waits for GPU kernel, async I/O, ...
- Cannot be captured in single scalar value!

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:
    - In shipping: consider cargo volume and weight
    - In nutrition: consider fat, protein, carbohydrates, ...

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: consider cargo volume and weight
    - In nutrition: consider fat, protein, carbohydrates, . . .
- The same applies to computer programs!

Vector Load Balancing

Vector Load

- Instead of a using single scalar l for load, use a vector $\vec{l} = < l_1, l_2, \ldots, l_d >$ of dimension d

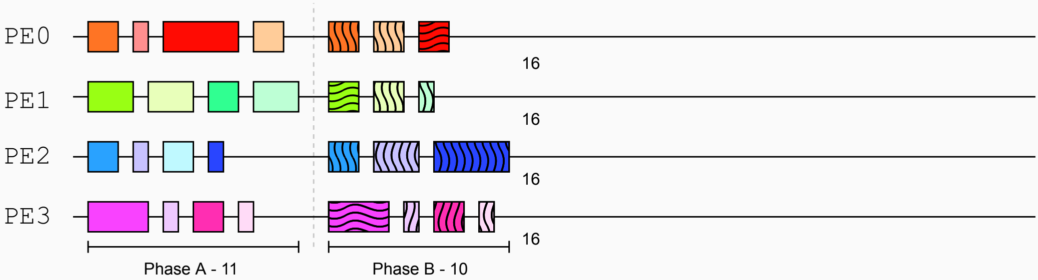
Vector Load

- Instead of a using single scalar l for load, use a vector $\vec{l} =< l_1, l_2, \ldots, l_d >$ of dimension d
- Remedies earlier deficiencies:
    - Phase-based applications
        - Time spent in each phase: $< t_A, t_B, \ldots, t_n >$
    - Resource constrained applications
        - Resource usage alongside CPU time: $< cpu, mem >$
    - Asynchronous computation
        - GPU time alongside CPU time: $< cpu, gpu >$
- Rich, flexible, and extensible packaging of load

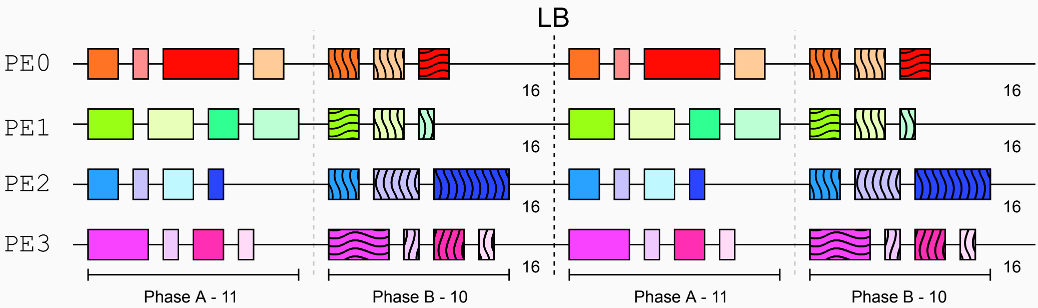
Measuring Vector Loads

- 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 (msgs, bytes sent)
  - Memory footprint via `PUP`
  - GPU load via `accel` or CUDA timers
  - Users may manually specify load vector

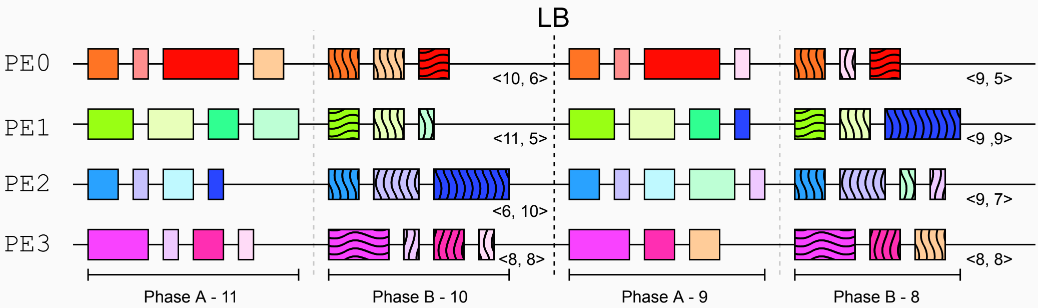
Phase-Based Application



Phase Unaware LB



Phase Aware LB



Vector Load Balancing Strategies

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 $\max(\vec{l})$ 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-norm, ∞-norm, etc.)

Vector Strategies - Norm

- Implemented several different versions of norm-based assignment
  - Exhaustive search
  - k-d tree-based search
  - rk-d tree-based search
  - Pareto frontier-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
  - Pareto frontier-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, $1, 2, \ldots, k, 1, \ldots$, each node stores arbitrary discriminant $j \in \{1, 2, \ldots, k\}$

- Stochasticity keeps tree balanced, makes updates fast

Random Relaxed k-d

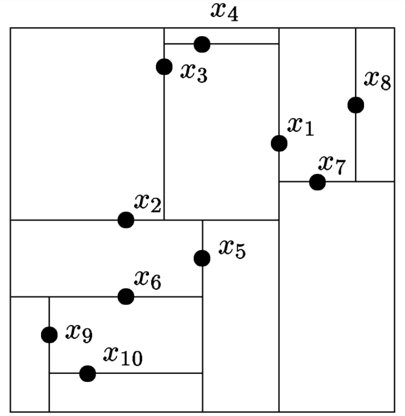


Figure 1: k-d

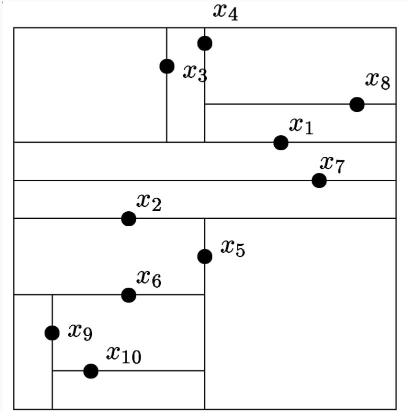
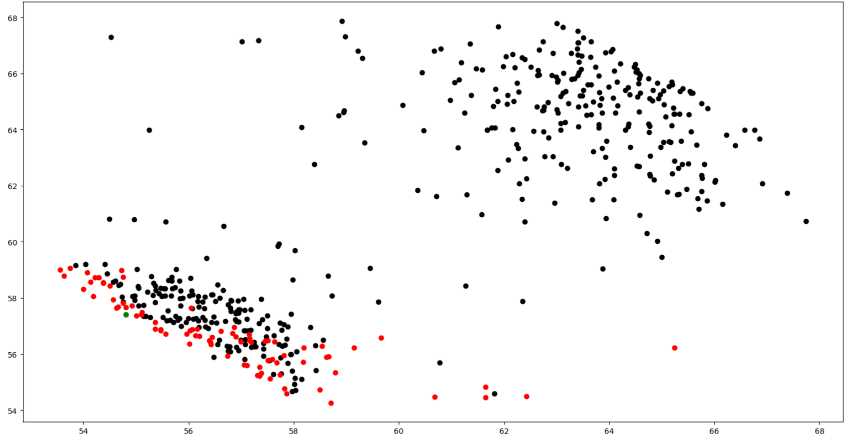


Figure 2: rk-d

k-d - Pruning



NormLB - Performance

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 1: Performance of Norm-Based Strategies

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

Objectives and Results

Phase Objective Function

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

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

Phase Objective Function

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

$$\underset{M}{\arg\min} \sum_{0 \le i < d} \left(\max_{p \in P} \left(\overbrace{\sum_{\forall o \in O : M(o) = p} (\vec{l_o})_i}^{\text{Load on PE } p \text{ in dimension } i} \right) \right) \tag{1}$$

Phase Objective Function

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

Load on PE p in dimension i

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

Maximum load in dimension i on a single PE

Phase Objective Function

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

Sum across dimensions

Load on PE p in dimension i

$$\arg\min_{M} \sum_{0 \le i < d} \left(\max_{p \in P} \left(\sum_{\forall o \in O : M(o) = p} (\vec{l_o})_i \right) \right) \tag{1}$$

Maximum load in dimension i on a single PE

Overlapped Objective Function

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

Load on PE p in dimension i

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

Maximum load in dimension i on a single PE

Overlapped Objective Function

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

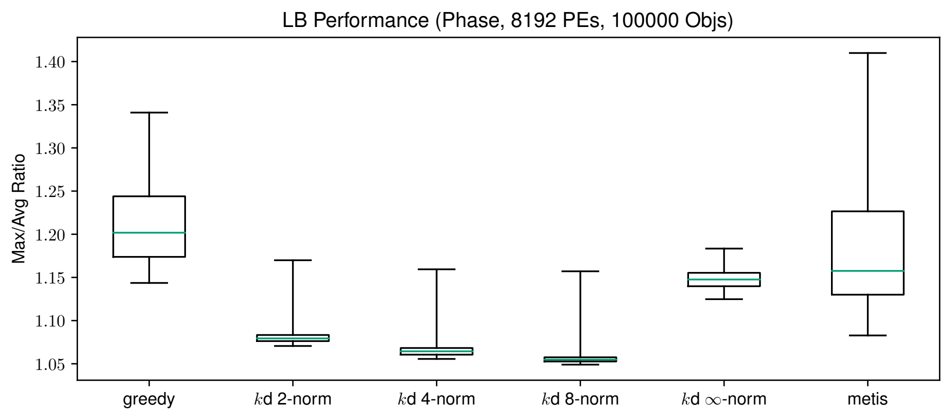
Max across dimensions

Load on PE p in dimension i

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

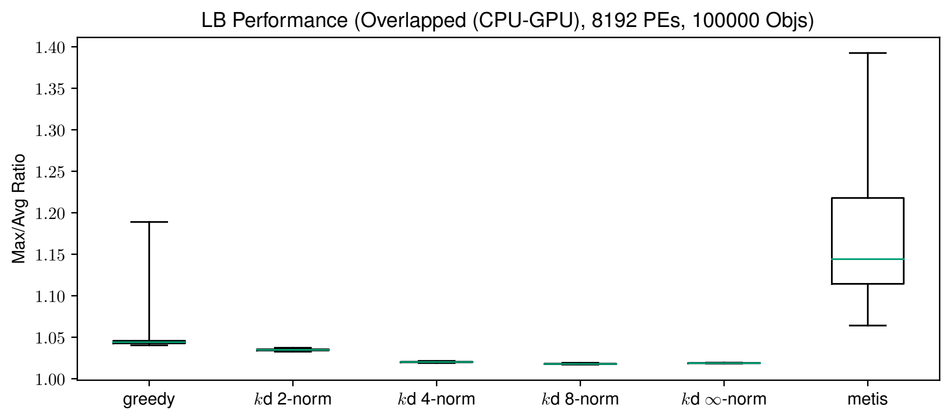
Maximum load in dimension i on a single PE

Vector LB Simulations - Phase



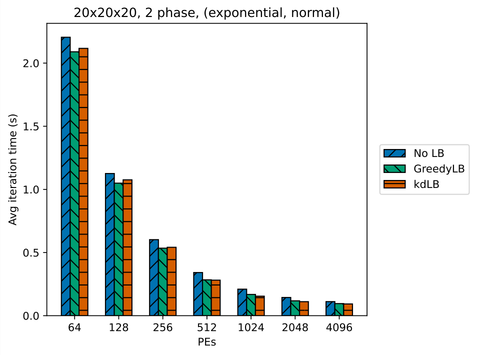
100 trials, Synthetic data: 2 phase (exp $\lambda = 0.15$, normal $\mu = 10, \sigma^2 = 3$)

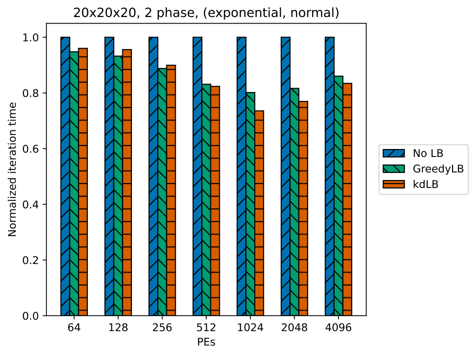
Vector LB Simulations - Overlapped



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

Vector LB Runs





Runtime

Normalized

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

VT

- Task-based programming model from Sandia
- 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
- Phase-based application called EMPIRE
    - 14 dimensions
- kdLB gives approximately 12% performance improvement over previous best LB strategy

VT

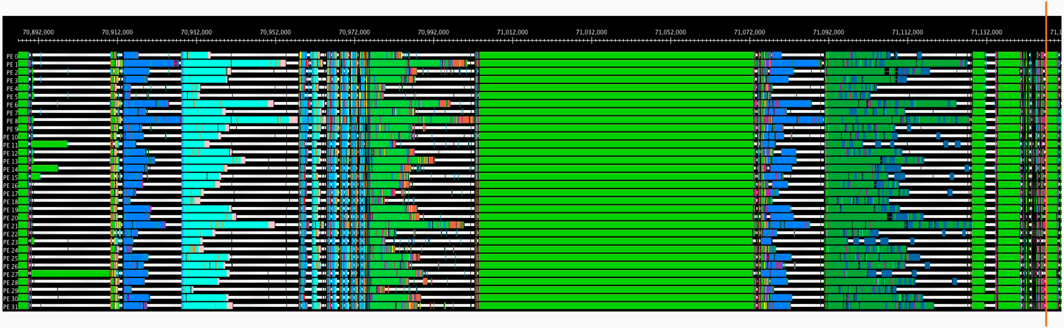


Figure 3: With previous best LB

VT

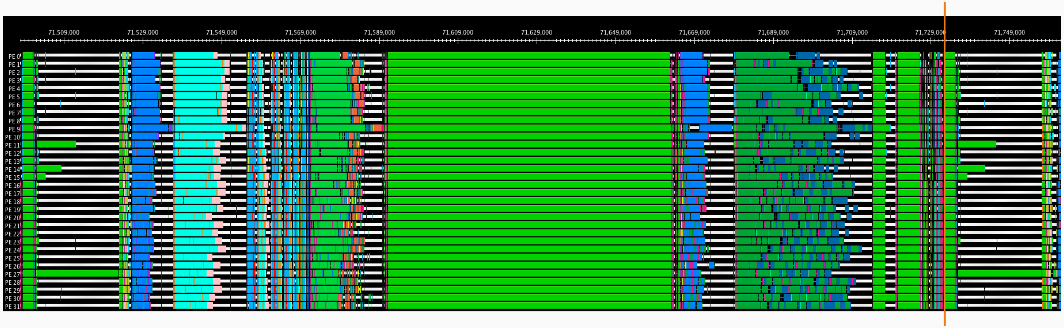


Figure 4: With kdLB

Additional Topics

- kdConstrainLB allows subset of dimensions to be constrained while rest are minimized
    - Prevented `malloc` failures in memory constrained run
    - Other LB strategies resulted in crashes
- Additional class of approximate norm-based strategies that tradeoff quality for performance

Conclusions

- Load characteristics of complex, modern applications cannot be captured in a single scalar
- New LB strategies can tractably utilize the additional detail provided by a load vector
- Vector LB has shown improvement over scalar LB across synthetic and production applications

Questions?