Scalable and Asynchronous Algorithms for Block Structured Adaptive Mesh Refinement

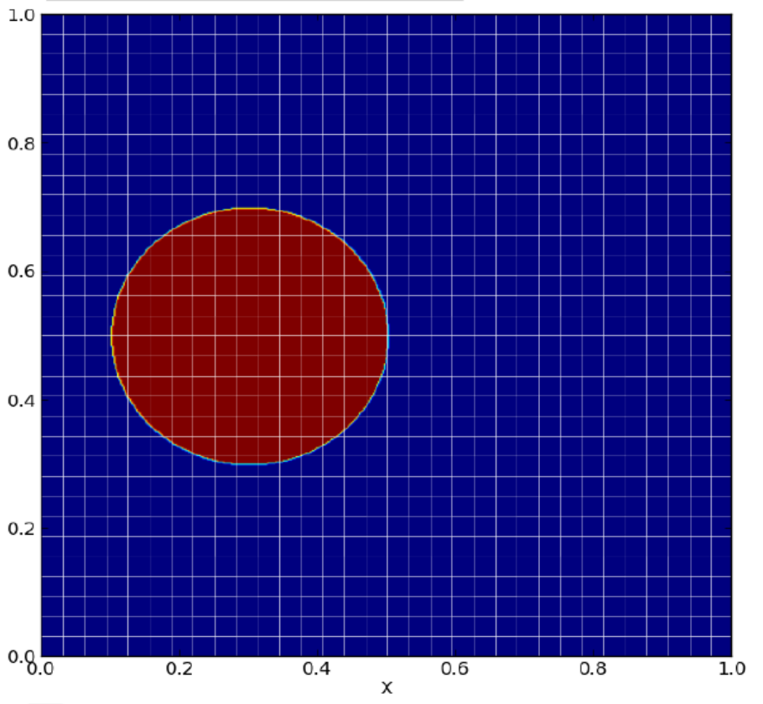
Akhil Langer and Laxmikant Kale
Parallel Programming Laboratory, Department of Computer Science
University of Illinois at Urbana-Champaign

Introduction to Adaptive Mesh Refinement (AMR)

Solving Partial Differential Equations (PDEs)

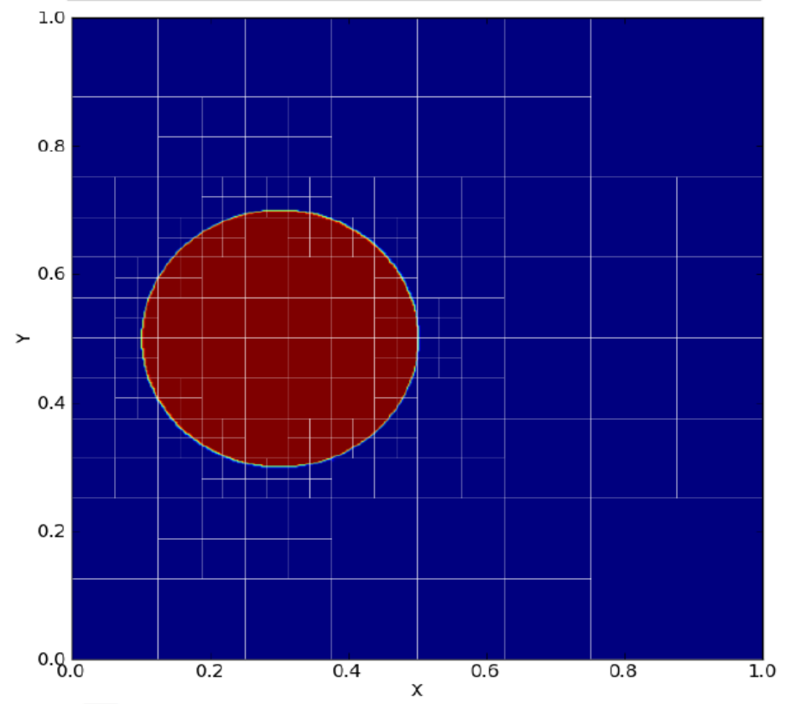
- ☐ PDEs solved using discrete domain
- ☐ Algebraic equations estimate values of unknowns at mesh points
- ☐ Resolution of mesh points determines error

Uniform meshes



- ☐ High resolution required for handling difficult regions (discontinuities, steep gradients, shocks, etc.)
- ☐ Computationally extremely costly

Adaptively Refined Meshes



- ☐ Start with a coarse grid
- ☐ Identify regions that need finer resolution
- ☐ Superimpose finer subgrids only on those regions

Applications

- CFD
- Astrophysics
- Climate Modeling
- Turbulence
- Mantle Convection Modeling
- Combustion
- Biophysics
- and many more

Existing Frameworks

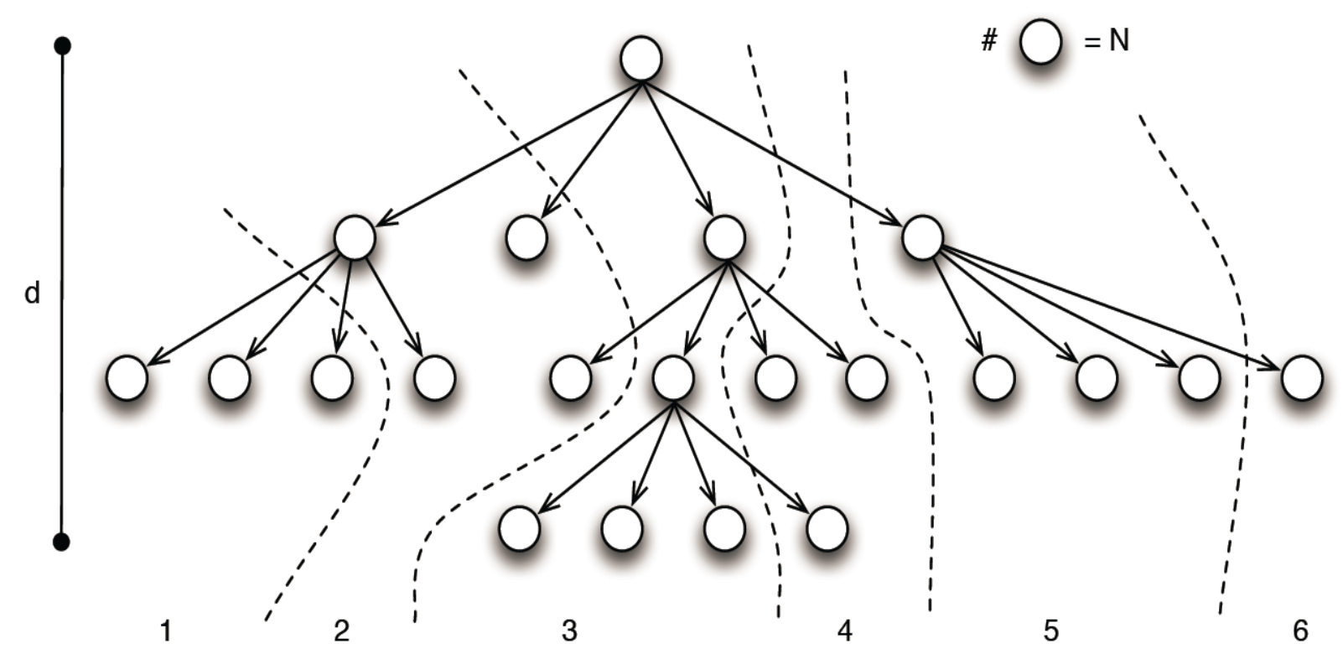
- Enzo-P
- Chombo
- PARAMESH
- SAMRAI
- FLASH
- p4est
- deal.II
- and many more

AMR makes it feasible to solve problems that are intractable on uniform grid

Typical Traditional Approach

Background on AMR

- ☐ Refinement levels of neighboring blocks differ by ±1
- ☐ Refinement structure can be represented using a quad-tree (2D)/ oct-tree (3D)



A set of blocks assigned to a process

Disadvantages

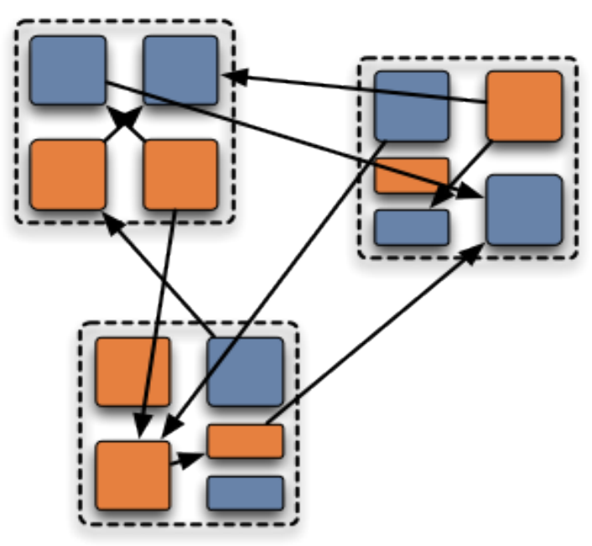
Tree meta-data replicated on each process	Level-by-level restructuring	Does not allow coarsening of sibling blocks residing on different processors
☐ $O(\#blocks)$ memory per-process ☐ High memory footprint Memory Bottleneck	☐ ripple propagation ☐ $O(d)$ reductions Synchronization overheads	

Scalable Approach – Basic Design

Basic Design: Promote individual block as first-class entities, instead of a process

Block acts as a virtual process

- ☐ overlap of computation with communication of other blocks on same physical process
- ☐ Run time handles communication between arbitrary blocks

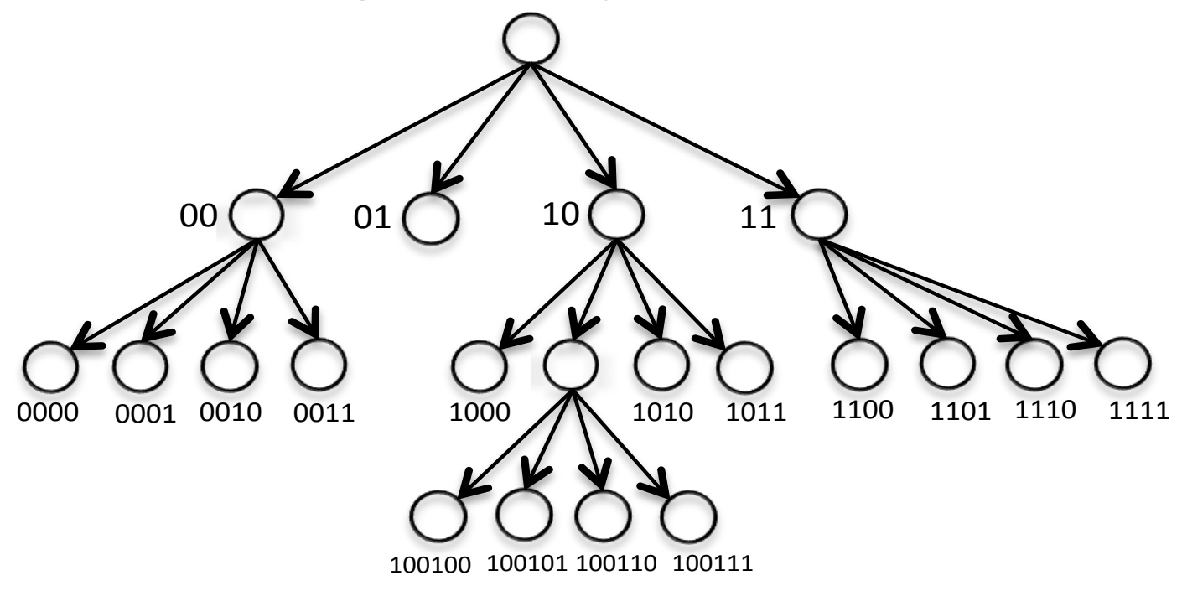


Dynamic placement of blocks on physical processes

- ☐ Facilitates dynamic load balancing

Block Naming

- ☐ Bitvector describing path from root to block's node
- ☐ One bit per dimension at each level
- ☐ Easy to compute parent, children, siblings
  - using bit manipulation



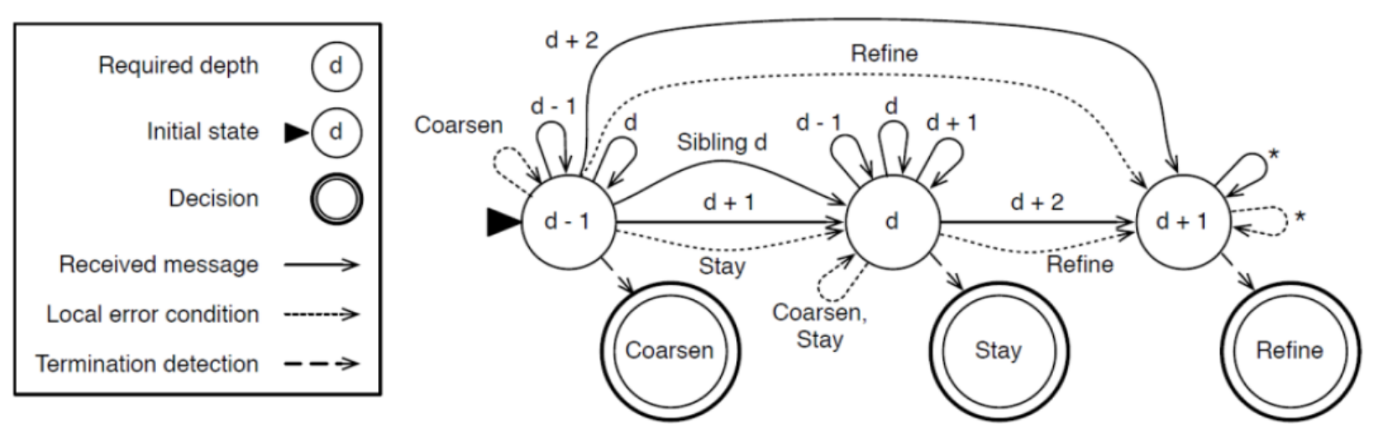
Block is a unit of algorithm expression

- ☐ Simplifies implementation complexity

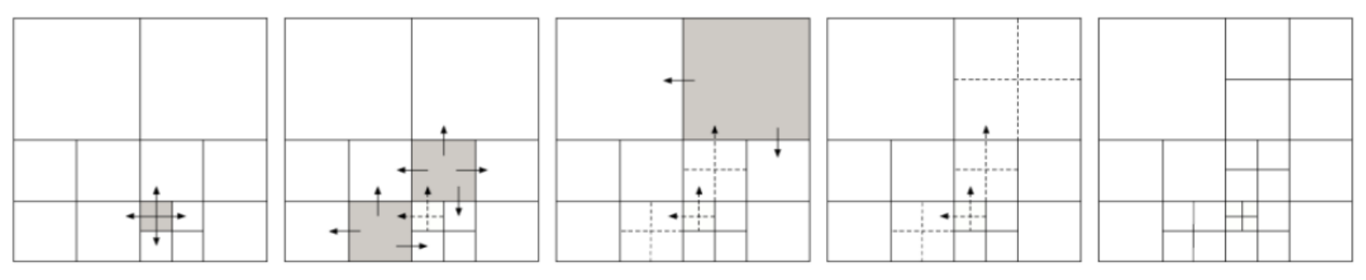
Highly Asynchronous Scalable Approach

The Highly Asynchronous Mesh Restructuring Algorithm

- ☐ Based on local error estimate, blocks make *refine, coarse or stay* decisions
- ☐ *refine* and *stay* decisions communicated to neighbors
- ☐ Decisions updated based on DFA, changes in decision are communicated



- ☐ *refine, stay* propagate along the mesh irrespective of blocks refinement levels



- ☐ *When to stop?*
  - System quiescence indicates global consensus on refinement decisions
  - Blocks proceed to next iterations when quiescence detected, no need to wait for blocks to be created
  - Messages directed to not yet created blocks are buffered

Time complexity of
Quiescence detection: $O(\log P)$

Algorithmic Benefits

	Typical Traditional Approach	Charm++ Approach
Memory	$O(\#blocks)$ per process	$O(\#blocks/P)$ per process
Mesh Restructuring	$O(d)$ reductions ≈ $O(d\log P)$ time	1 Quiescence detection ≈ $O(\log P)$ time
	Synchronized	Highly asynchronous
Neighbor Lookup	$O(P)$ data structure	Hash table
	$O(\log P)$ time	$O(1)$ time
Implementation	Complex	Simple, sloc: 1300 for 2D, 1600 for 3D Advection

<u>Implementation</u>

Charm++ run-time system

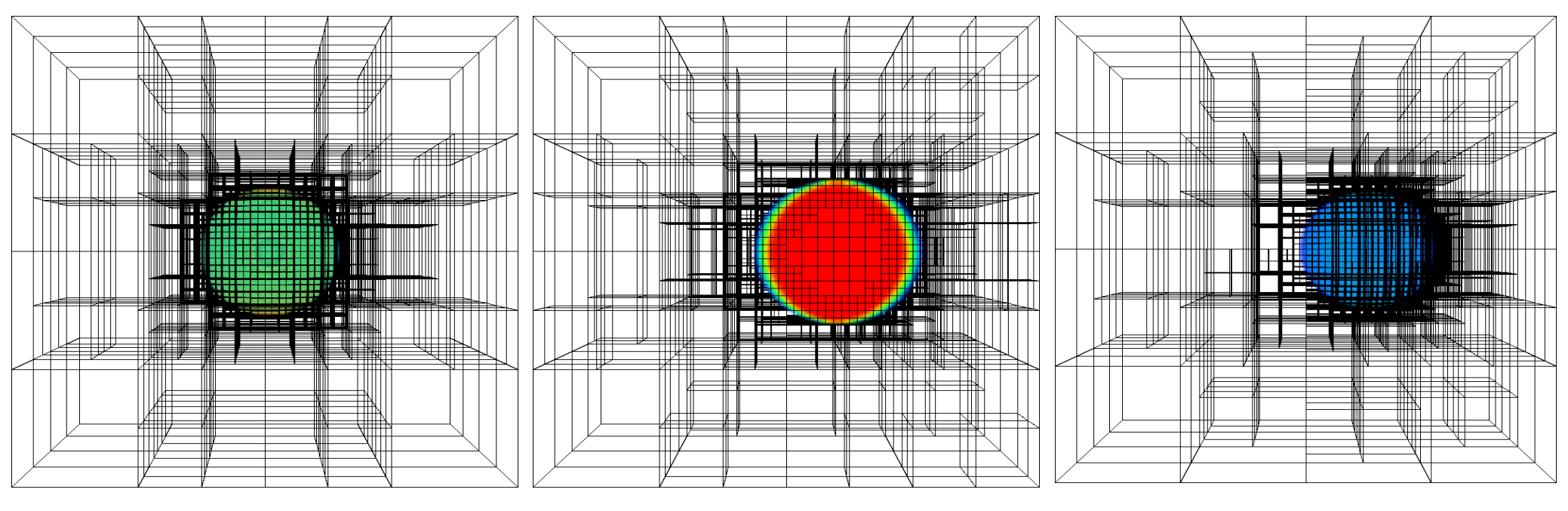
- ☐ Custom chare arrays

<u>Dynamic Load Balancing</u>

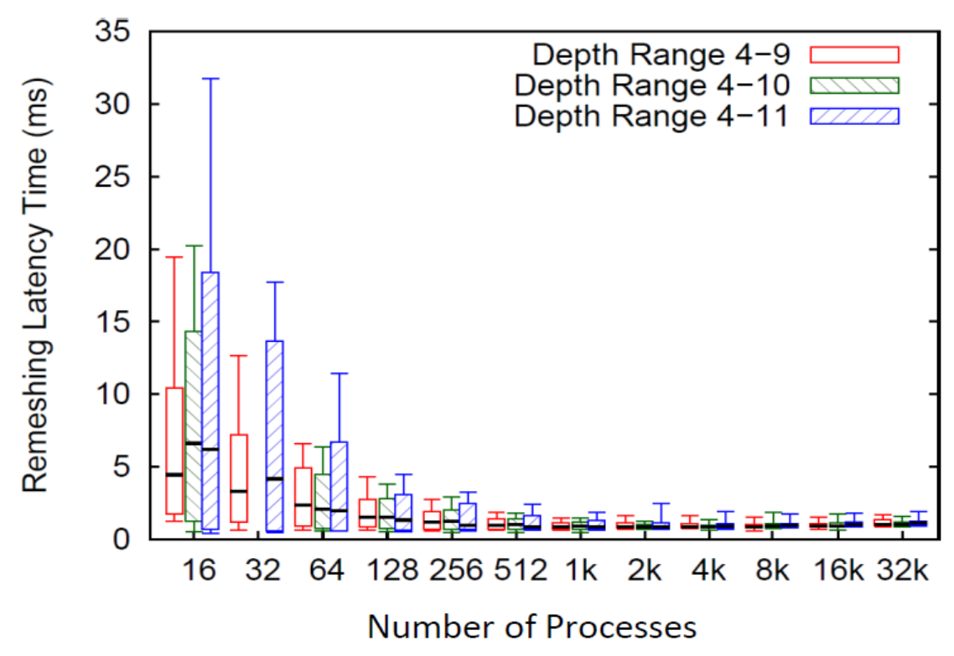
- ☐ Blocks created and destroyed as simulation progresses, creating load imbalance
- ☐ Distributed load balancer
  - ☐ Gossip protocol to spread load information + probabilistic transfer of load
  - ☐ O(log P) space, O(log P + #blocks/P) time

<u>Advection Benchmark</u>

- ☐ First order up-wind method in 3D
- ☐ Advection of a tracer along a fluid

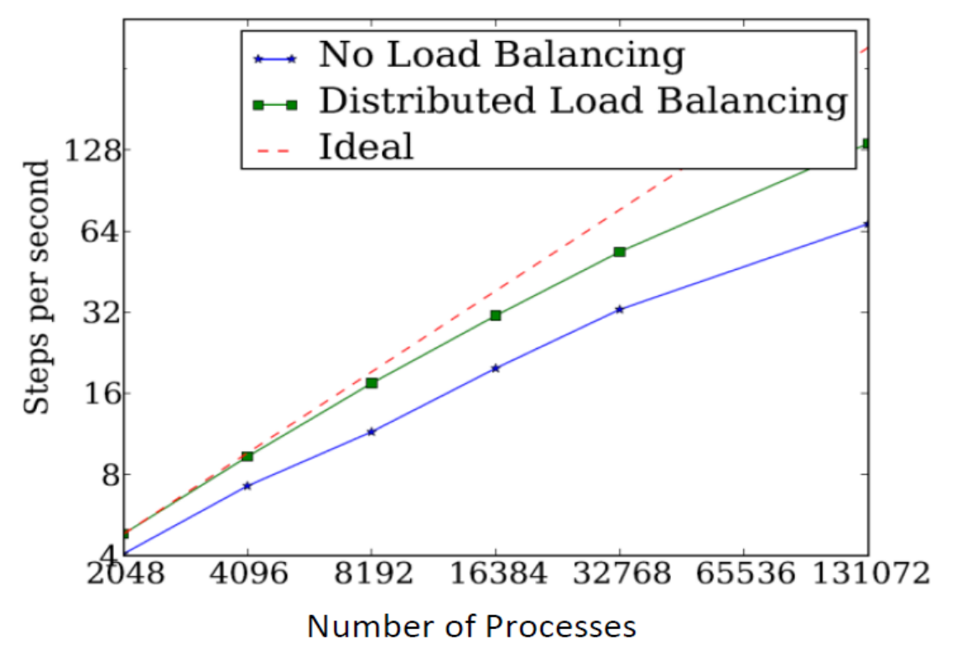


Performance Results



Mesh restructuring time on IBM BG/Q

- ☐ Remeshing decisions and communication: scalable
- ☐ Termination detection time: logarithmic in #processes



2D Advection on IBM BG/Q with max-depth of 15
Strong scaling (highly stressed) efficiency of 80% at 16,384 cores

<u>Conclusion</u>

- ☐ Fully distributed (No $O(P)$ or $O(\#blocks)$ data structures)
- ☐ Scalable and asynchronous mesh restructuring algorithm
- ☐ Asynchronous progress in computation
- ☐ New approach is scalable and promises high performance for much more deeply refined computations than are currently practiced

<u>Future Work</u>

- ☐ Use work stealing in between the distributed load balancing steps
- ☐ Benchmark at higher scales and for more applications

Student Research Symposium
IEEE High Performance Computing Conference (HiPC) 2013