# Scalable Algorithms for Distributed Memory Adaptive Mesh Refinement

Parallel Programming Laboratory UIUC

*Akhil Langer, Jonathan Lifflander, Phil Miller, Harshitha Menon, Laxmikant V. Kale*
*Parallel Programming Laboratory, Department of Computer Science*

**Kuo-Chuan Pan, *Paul Ricker*
**Department of Astronomy*

*University of Illinois at Urbana-Champaign*

## Introduction

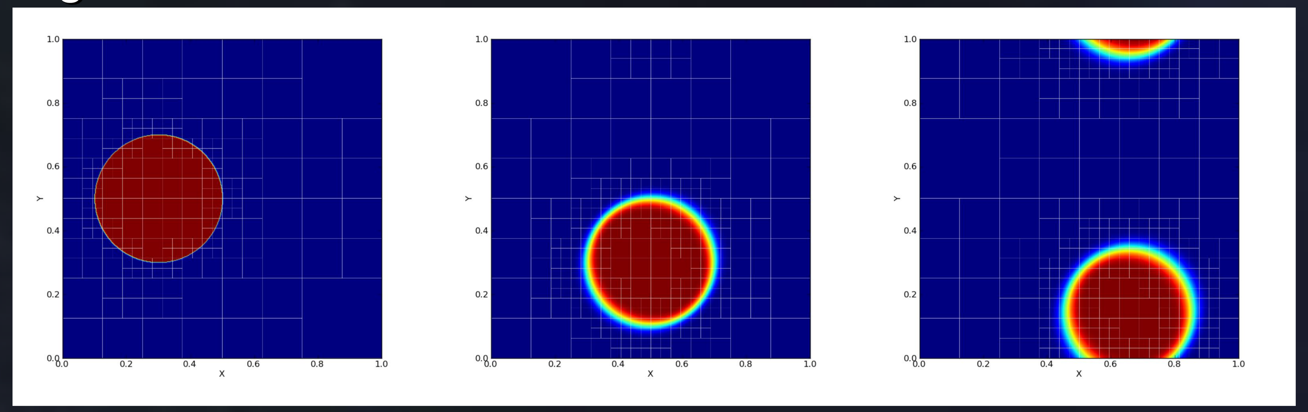
### Motivation

- Eulerian methods widely used in numerical cosmology, global atmospheric modeling, mantle convection modeling, etc.
- Requires simulation of large meshes ( e.g. size $10^{15}$)
- Intractable even on modern supercomputers

### Solution – Adaptive Mesh Refinement (AMR)

Every few iterations of the Euler method

- Refine zones that need finer precision
- Keep others at coarse granularity level or coarsen them
- Neighboring blocks remain within ±1 refinement level of their neighbors

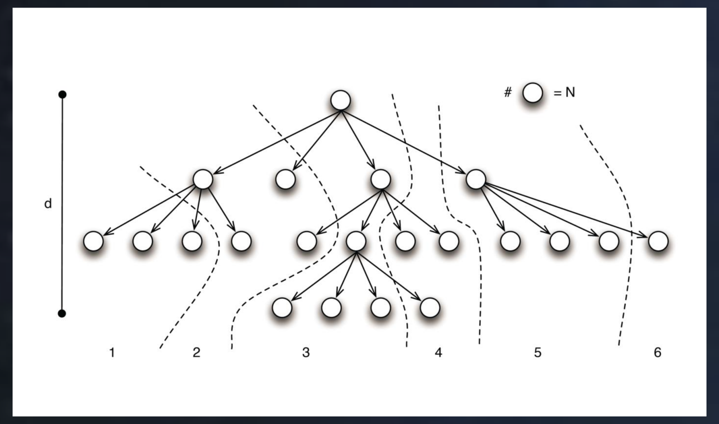


AMR mesh evolving over time. An example simulation of a circular fluid advected by a constant velocity field

## Need for Scalable Algorithms

### Traditional Algorithms

- Each process manages a set of neighboring blocks assigned to it through a space filling curve (e.g. Hilbert curve)



Tree partitioning for assignment to processes

### Limitations of Traditional Algorithms

- $O(\#blocks)$ memory per process to store the tree information
- $O(\log P)$ time to locate neighboring blocks
- $O(d)$ rounds of collective communication during mesh restructuring
- Centralized load balancing – takes O(#blocks) time and memory
- Does not allow coarsening of sibling blocks residing on different processors

### At extreme scale

- As available memory per process decreases, traditional algorithms pose memory bottleneck

## Scalable Algorithms

### Design

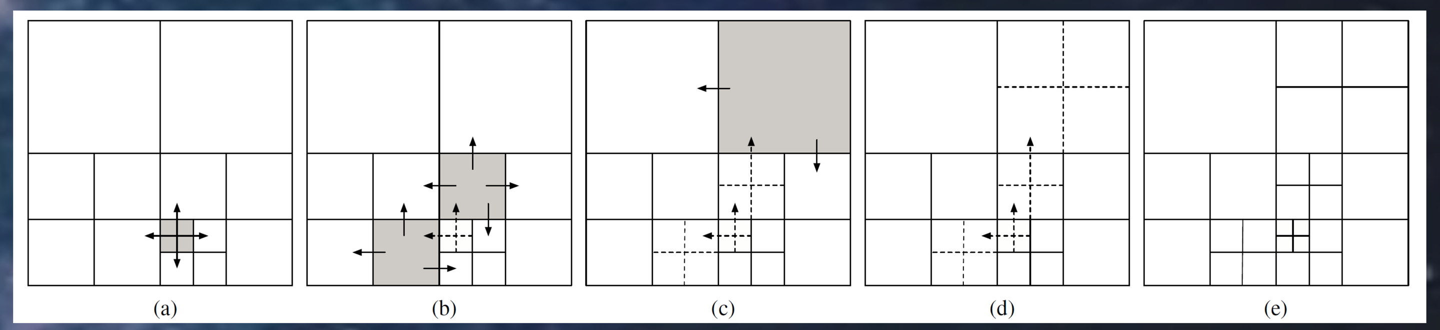
Each block acts as a first class entity – Charm++ object:

- Acts as a virtual processor – allowing overlap of computation with communication of other blocks on the same physical process
- Uniquely identified by its location in the refinement tree
- Dynamically placed on any physical process – facilitating dynamic load balancing
- Unit of algorithm expression – reduces implementation complexity
- End-point of communication – run-time system handles communication between arbitrary blocks
- $O(\frac{\#blocks}{P})$ memory per process to store the tree information

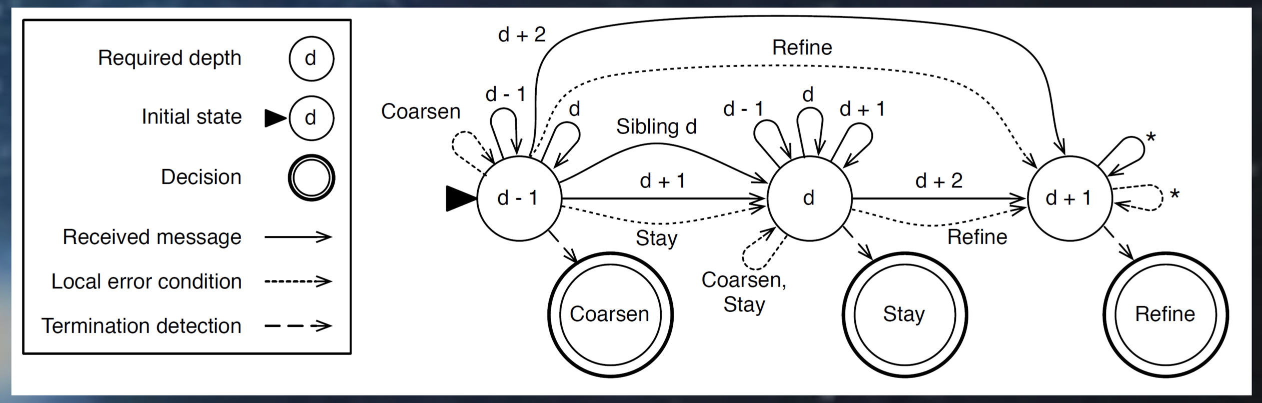
### The Mesh Restructuring Algorithm

Executes in two phases separated by a system quiescence state:

- Phase 1
  - Based on local error estimate, make one of the following decision: *refine, stay* or *coarsen*
  - Communicate *refine* and *stay* decision to neighboring blocks
  - Update decision based on the DFA below and communicate change in decision
  - Wait for system quiescence state – takes $O(\log P)$ time
- Phase 2
  - Create new blocks or destroy existing ones based on the refinement decision
  - Wait for system quiescence state




Propagation of refinement decision messages based on local-error criteria and near-neighbor communication

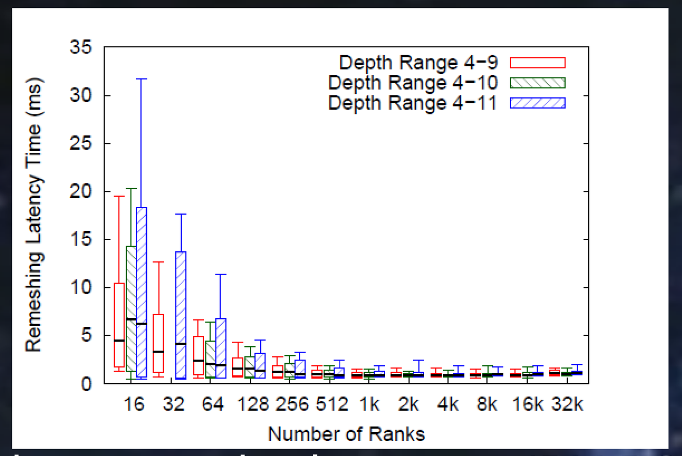



The finite state machine describing each block's decision process during the mesh restructuring algorithm

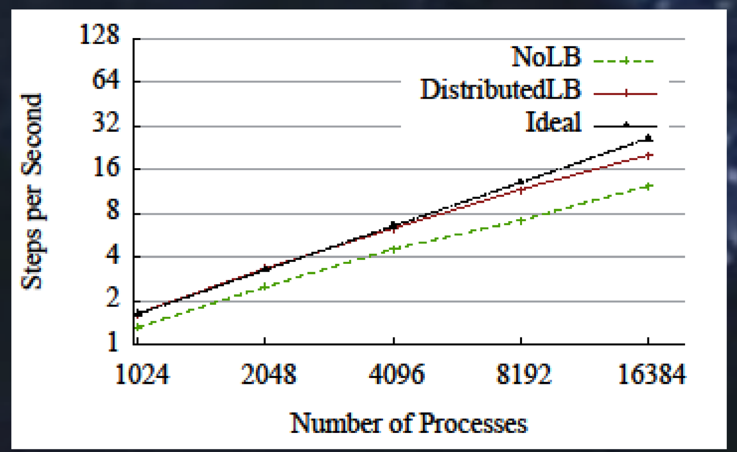
### Dynamic Distributed Load Balancing

- Load balance blocks across processors every few iterations
- Charm++ provided distributed load balancer – *Grapevine*
- Competitive with the centralized load balancers while incurring negligible overhead

## Experimental Results




Mesh restructuring latency on IBM BG/Q




Time steps per second strong scaling (max mesh depth: 15) on IBM BG/Q

## Conclusion

- Elevate blocks to first-class entities- Charm++ objects identified with bit-vector ids
  - No $O(P)$ data structures, constant time neighbor lookup
  - Enables asynchronous progress in computation
- Adapt mesh using near-neighbor point-to-point messages and quiescence detection
  - Only 2 quiescence states vs $O(d)$ reductions
  - Eliminate memory hungry collectives taking $O(d \log P)$ time
- Distributed dynamic load balancing of blocks

Enables high performance for much more deeply refined computations than are currently practiced

## References

1. AMR algorithm and benchmark source code," 2012. Source code and scripts available at git://charm.cs.illinois.edu/benchmarks/amr.git.
2. Langer, A., Lifflander, J., Miller, P., Pan, K. C., Kale, L. V., & Ricker, P. (2012, October). Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on* (pp. 100-107). IEEE.
3. Kale, L., Arya, A., Jain, N., Langer, A., Lifflander, J., Menon, H., Ni, X., Sun, Y., Totoni, E., Venkataraman, R.,, Wesolowski, L. Migratable Objects + Active Messages + Adaptive Runtime = Productivity + Performance: A Submission to the 2012 HPC Class II Challenge [SC 2012]. PPL Technical Report: 12-47

**Acknowledgements**

The authors were supported by grants MITRE Research Agreement No. 81990, NSF ITR-HECURA-0833188, NSF OCI-0725070. This research used resources of the Oak Ridge Leadership Computing Facility located in the Oak Ridge National Laboratory and the Argonne Leadership Computing Facility at Argonne National Laboratory which are supported by the Office of Science of the Department of Energy under Contract DEAC05-00OR22725 and DE-AC02-06CH11357, respectively.