Optimal Distributed Load Balancing Algorithm for Homogeneous Work Units

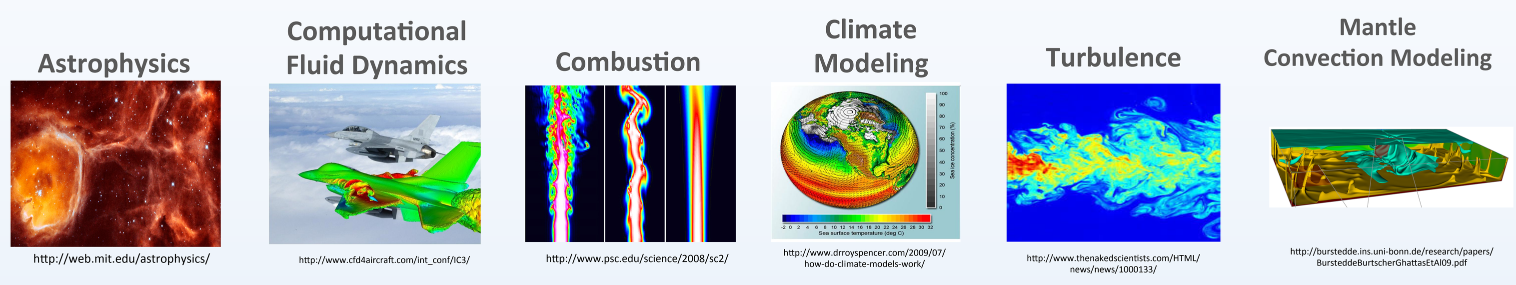
Akhil Langer (4th year PhD Student) :: Laxmikant (Sanjay) Kale (Advisor)
Department of Computer Science, University of Illinois at Urbana-Champaign

PARALLEL PROGRAMMING LABORATORY

Many parallel applications need dynamic load balancing during the course of their execution because of dynamic variation in the computational load. An ideal load balancer is one that can achieve perfect load balance, while doing minimal data migration (i.e. is communication minimizing), is highly scalable to large number of processors and does not have large memory footprint. None of the existing load balancing algorithms - centralized, hybrid, and distributed algorithms – satisfy the criterion of an ideal load balancer. We propose a novel tree-based fully distributed algorithm that is an ideal load balancing algorithm when the work units are homogeneous. We evaluate its performance on Mira and Blue Waters and compare it with several existing load balancing strategies.

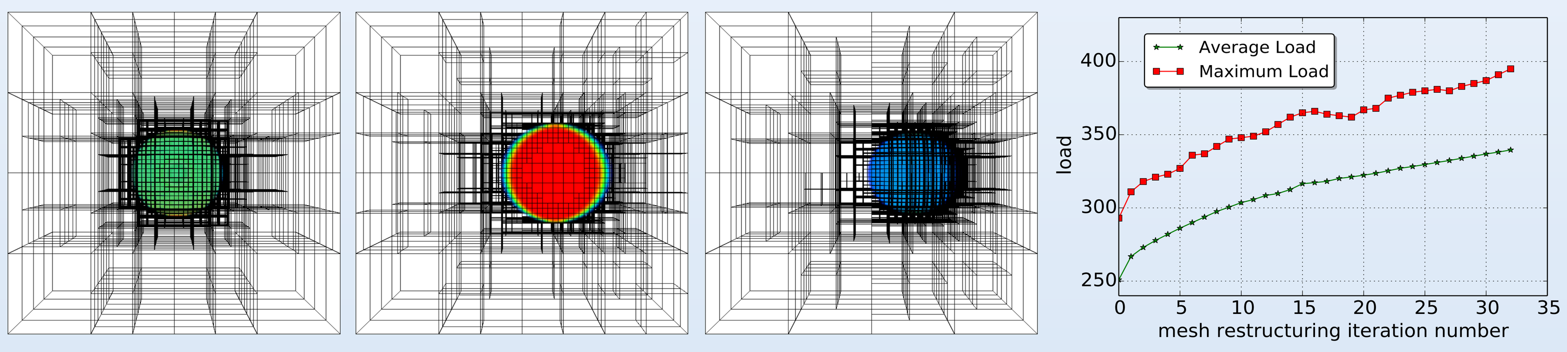
MOTIVATION

Adaptive Mesh Refinement (AMR) simulations constitute significant portion of many of the world's supercomputers usage



Mesh is restructured very frequently, typically every two steps

Frequent refining and coarsening creates load imbalance across processes



Problem Statement

Develop a scalable distributed load balancer that can distribute the work-units equally amongst processors while minimizing data communication

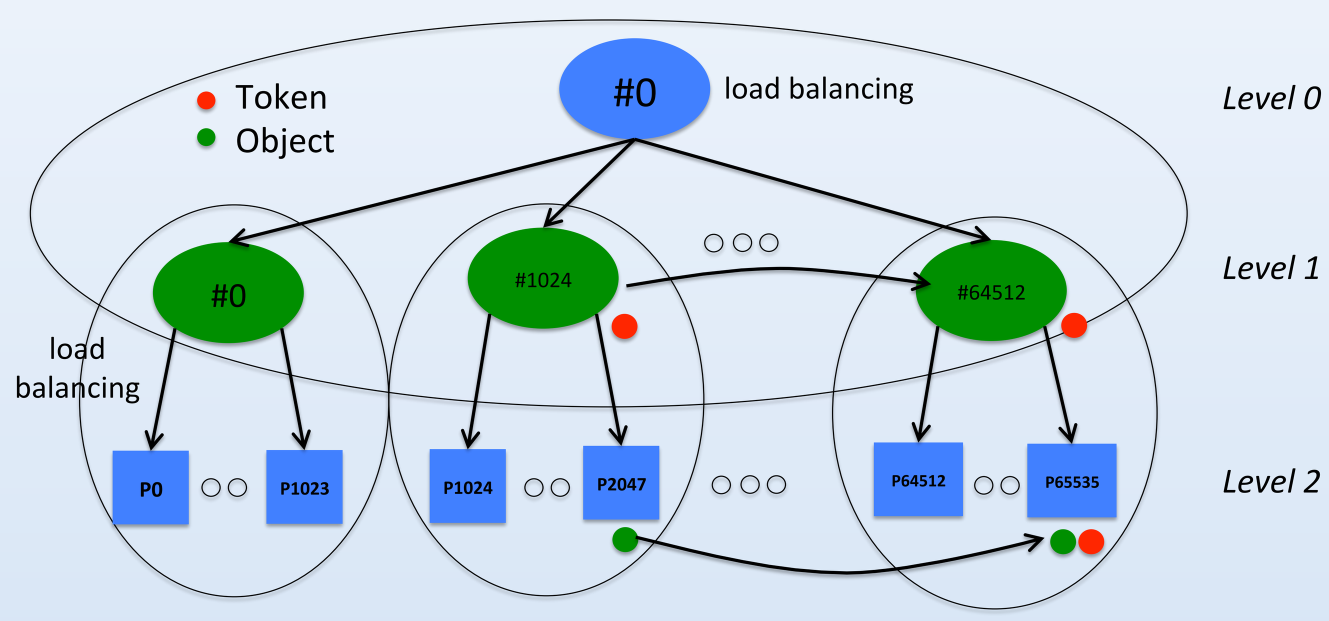
CENTRALIZED AND HIERARCHICAL ALGORITHMS

Centralized

Approach: Assign load from overloaded processors to under-loaded processors

Disadvantages: Significant overhead, O(p) data structures, O(p) time

Hierarchical

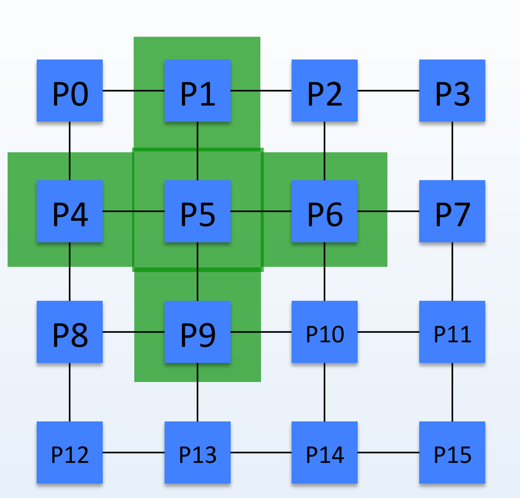


Approach
- ❑ Create subgroup of processes
- ❑ Collect load information at root of each subgroup
- ❑ Higher levels receive aggregate info
- ❑ Higher levels deliver decision at aggregate level

Disadvantages
- ❑ Excessive data collection at lower level
- ❑ Work done at multiple levels
- ❑ Requires manual tuning for subgroup size or number of levels

DISTRIBUTED ALGORITHMS

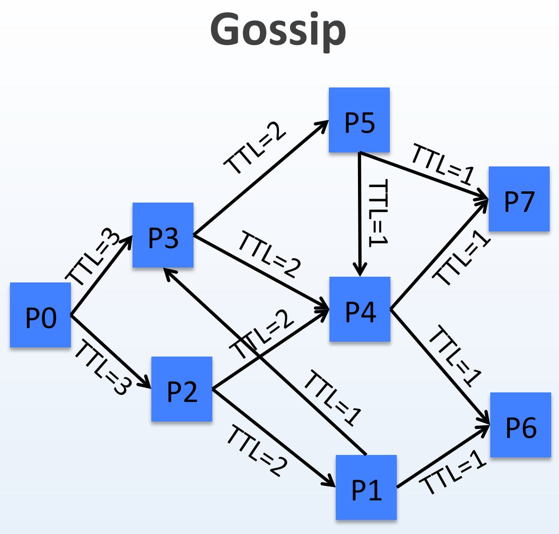
Diffusion



Decisions based on localized load information e.g. near neighbor

Disadvantages:
- ❑ Improve load balance rather than obtain global balance
- ❑ Multiple iterations to converge
- ❑ Slow convergence

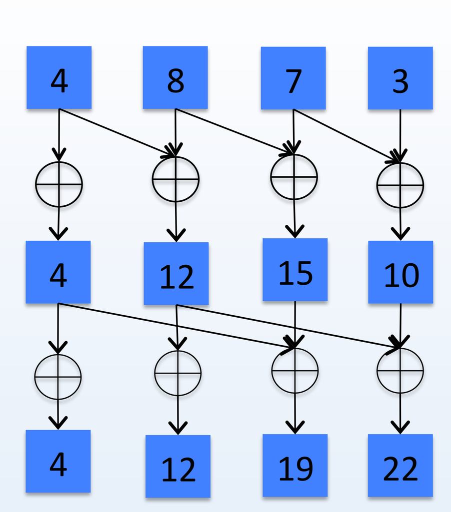
Gossip



Gossip protocol to propagate load information followed by probabilistic transfer of work units

Disadvantages:
- ❑ Multiple attempts to obtain target processor of a load
- ❑ User specified balance %age
- ❑ No guarantees on load balance

Parallel Prefix



Obtain task's global id using parallel prefix, assign to process

Optimization: Subtract global minimum load

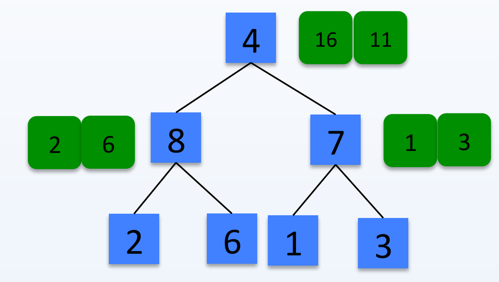
Advantages:
- ❑ Perfect load balance

Disadvantages:
- ❑ Excessive data migration
- ❑ Excessive network power consumption, congestion

PROPOSED TREE BASED ALGORITHM

1. Initialization (one-time)
- ❑ Construct spanning tree of processors with any branching factor

2. Upward Pass
- ❑ Wait for subtree load messages from child processors
- ❑ subtree load = my_load + Σ(child subtree load)
- ❑ Send subtree load to parent processor



3. Prepare for Downward Pass (first called at root and then called recursively on children)
- ❑ Compute child subtree loads after lb
- ❑ Call "Prepare for Downward Pass" on children
- ❑ If subtree load before lb > subtree load after lb
  - ➢ I am a work supplier subtree
  - ➢ Wait for receivers information
- ❑ If subtree load before lb < subtree load after lb
  - ➢ I am a work receiver subtree
  - ➢ Wait for suppliers information
- ❑ Do Downward Pass

Supplier Information
- ❑ Process#
- ❑ #work-units to be supplied
- ❑ Is process itself the supplier?

Receiver Information
- ❑ Process#
- ❑ #work-units to be received
- ❑ Is process itself the receiver?

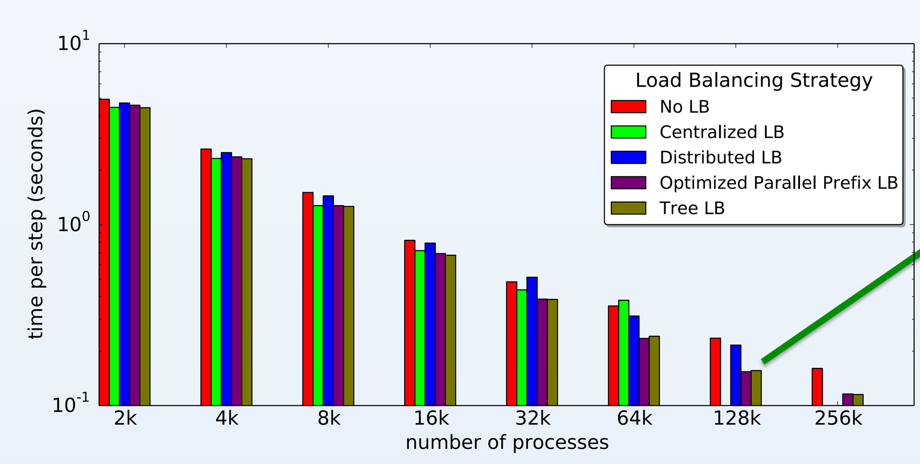
4. Do Downward Pass (Migration decisions and migrations take place in this step)
- ❑ Tag itself and child subtrees as work receiver/supplier
- ❑ "*Matchmaking*" -- assign suppliers to receivers
- ❑ If root process of work supplier subtree is itself the supplier, and root process of work receiver subtree is itself the receiver, then ask supplier to initiate work transfer
- ❑ Randomly send either receivers list to suppliers, or suppliers list to receivers (randomization keeps list sizes and number of messages per process small)

PERFORMANCE EVALUATION

Testbeds

IBM BG/Q	Cray XE6/XK7
PowerPC A2 1.6GHz	AMD 6276 Interlagos 2.45 GHz
16 cores, 4 hardware threads per core	16 FPU cores

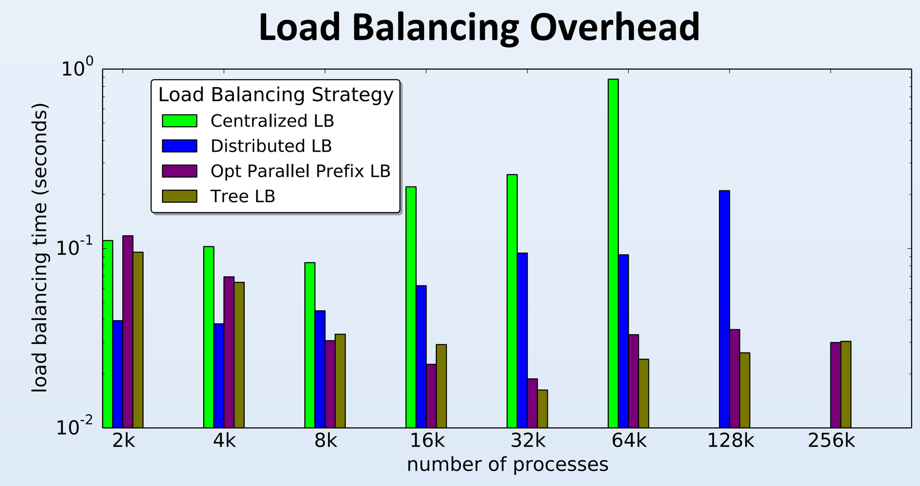
Time per step with various load balancing strategies



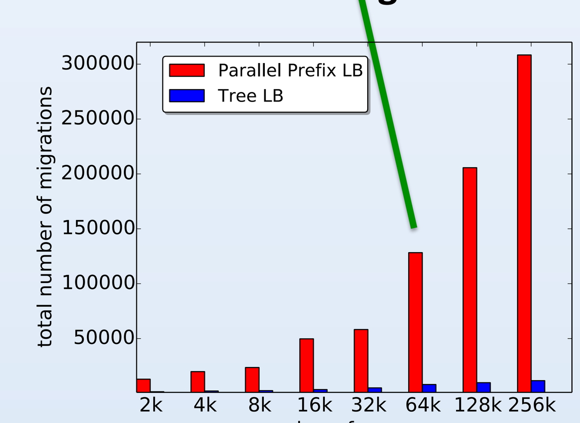
35% improvement over No LB
28% improvement over Gossip LB

95% reduction in migrations over Parallel Prefix LB

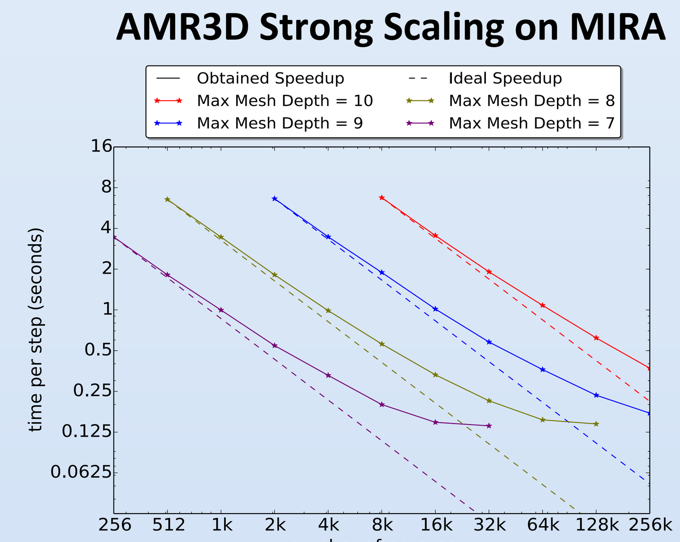
Load Balancing Overhead



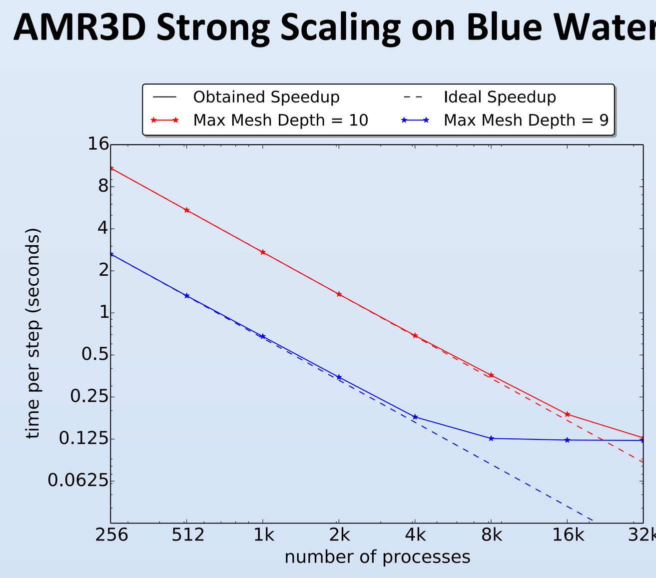
Number of Migrations



AMR3D Strong Scaling on MIRA



AMR3D Strong Scaling on Blue Waters



Uniform grid equivalent size (when max-depth = 10) = 549,755,813,888 zones

57% parallel efficiency at 256k processes **67% parallel efficiency at 32k processes**

CONTRIBUTIONS

- ❑ Communication minimizing optimal distributed load balancing algorithm:
  - ➢ *Minimal migration of work units*, 95% improvement over parallel prefix load balancer
  - ➢ *Very small memory footprint* – maximum list size of 13 at 128k processors
  - ➢ *Fully distributed* - $O(\log p)^2$ time complexity, assuming O(log p) memory footprint
  - ➢ Significant *reduction in network power consumption*
  - ➢ Performance Comparison with various other strategies
- ❑ *High Parallel Efficiencies* achieved for strong scaling of 3D Adaptive Mesh Refinement

References
1. Harshitha, et. Al. A Distributed Dynamic Load Balancer for Iterative Applications. SC 13.
2. G. Zheng, et al. Periodic Hierarchical Load Balancing for Large Supercomputers. IJHPCA, 2011.
3. A. Corradi, et al. Diffusive Load Balancing Policies for Dynamic Applications. IEEE Concurrency, 1999.
4. J. Lifflander et al. Persistence-based Load Balancers for Iterative Overdecomposed Applications. HPDC, 2012.
5. Langer, et al. Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement. SBAC-PAD 2012.