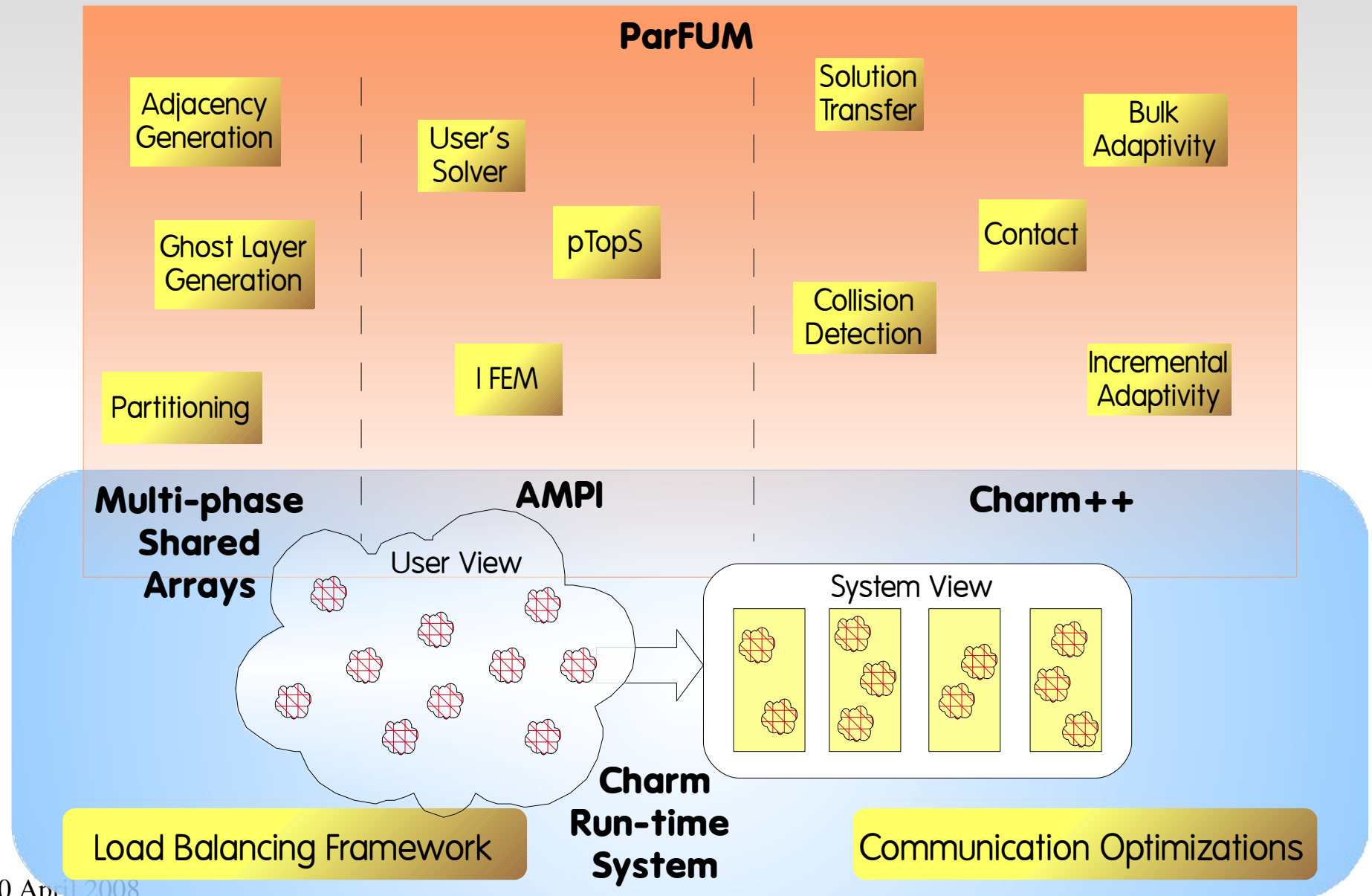


Scalable Dynamic Adaptive Simulations with ParFUM

Terry L. Wilmarth

Center for Simulation of Advanced Rockets
and Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

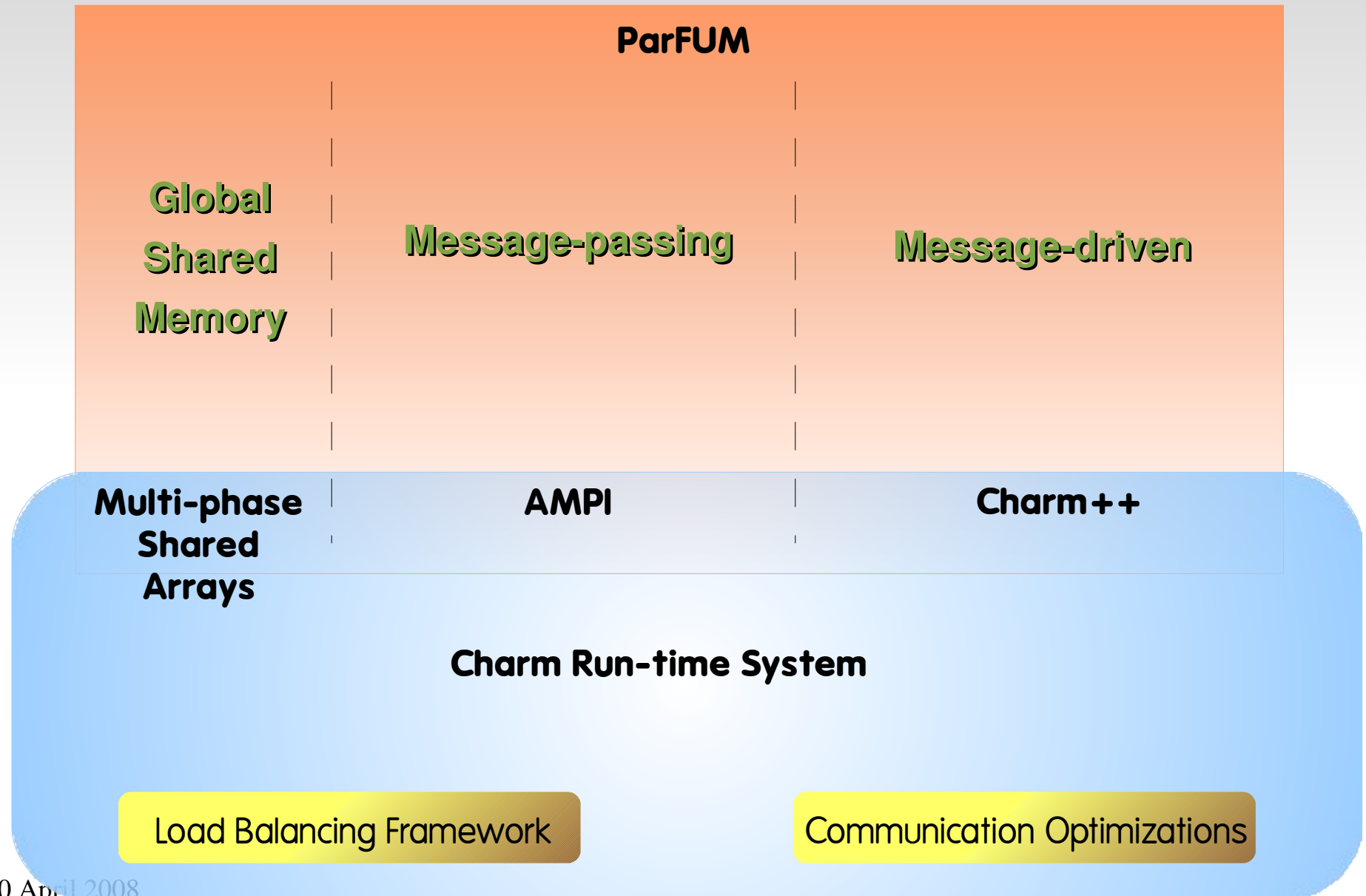
The Big Picture



A Brief Introduction to ParFUM

- Parallel Framework for Unstructured Meshes
- Inherited features from Charm run-time system:
 - object-based virtualization (AMPI threads/partitions)
 - automated dynamic load balancing
 - communication/computation overlap
 - multi-paradigm implementation
 - communication optimizations
 - portability

Multi-paradigm Implementation



ParFUM Features

- Flexible communication (“ghost”) layers
- Parallel partitioning
- MPI-style communication for shared and ghost entities
- C/C++ and FORTRAN bindings
- Supports multiple element types and mixed elements
- Support for topological adjacencies

ParFUM Features (cont'd)

- Mesh adaptivity
- Cohesive elements
- Solution transfer
- Contact detection

Why use ParFUM?

- Better performance for irregular problems
- Ease of use:
 - Fast conversion of serial codes to parallel
 - Even faster conversion of MPI codes to benefit from load balancing and other features
 - You can still use FORTRAN if you really want to
- Extremely portable, even to latest greatest supercomputers
- Development is collaboration-driven

User Responsibilities

- Specifying mesh data and attributes: two modes
 - ParFUM manages data
 - User writes solver to use ParFUM data format
 - User manages data
 - User writes packing and resizing code for their data
- Solver (implementation, porting, etc.)
- Use of simple collective calls to maintain consistency of data on shared/ghost entities

Virtualization of Partitions

- Create N virtual processors (mesh partitions), where $N \gg P$, the number of processors
- How to choose N :
 - minimize ratio of remote data to local data $\rightarrow$ larger partitions
 - minimize communication $\rightarrow$ larger partitions
 - maximize adaptive overlap $\rightarrow$ more VPs
 - maximize agility of load balancing $\rightarrow$ sufficient VPs
 - optimize cache performance $\rightarrow$ smaller partitions
- Start with ~ 2000 elements per partition

Virtualization of Dynamic Fracture

- Uses localized mesh adaptivity for solution accuracy; 50,000 elements initially
- Virtualization overhead on one processor

VPs	1	4	8	10	16	24	32
Time (10 ³ s)	7.9	8.4	9.2	9.7	10.7	11.7	12.0
% Increase	-	6.3	16.5	22.8	35.4	48.1	51.8

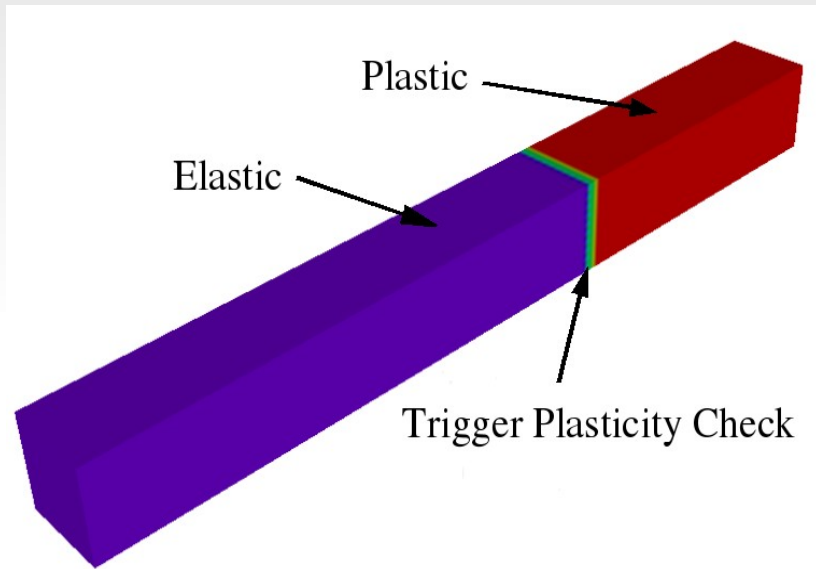
- Virtualization benefits on 16 processors

VPs/Proc	1	4	8	10	16	24	32
Time (s)	1328	934	835	857	807	769	770
% Decrease	-	29.7	37.1	35.5	39.2	42.1	42.0

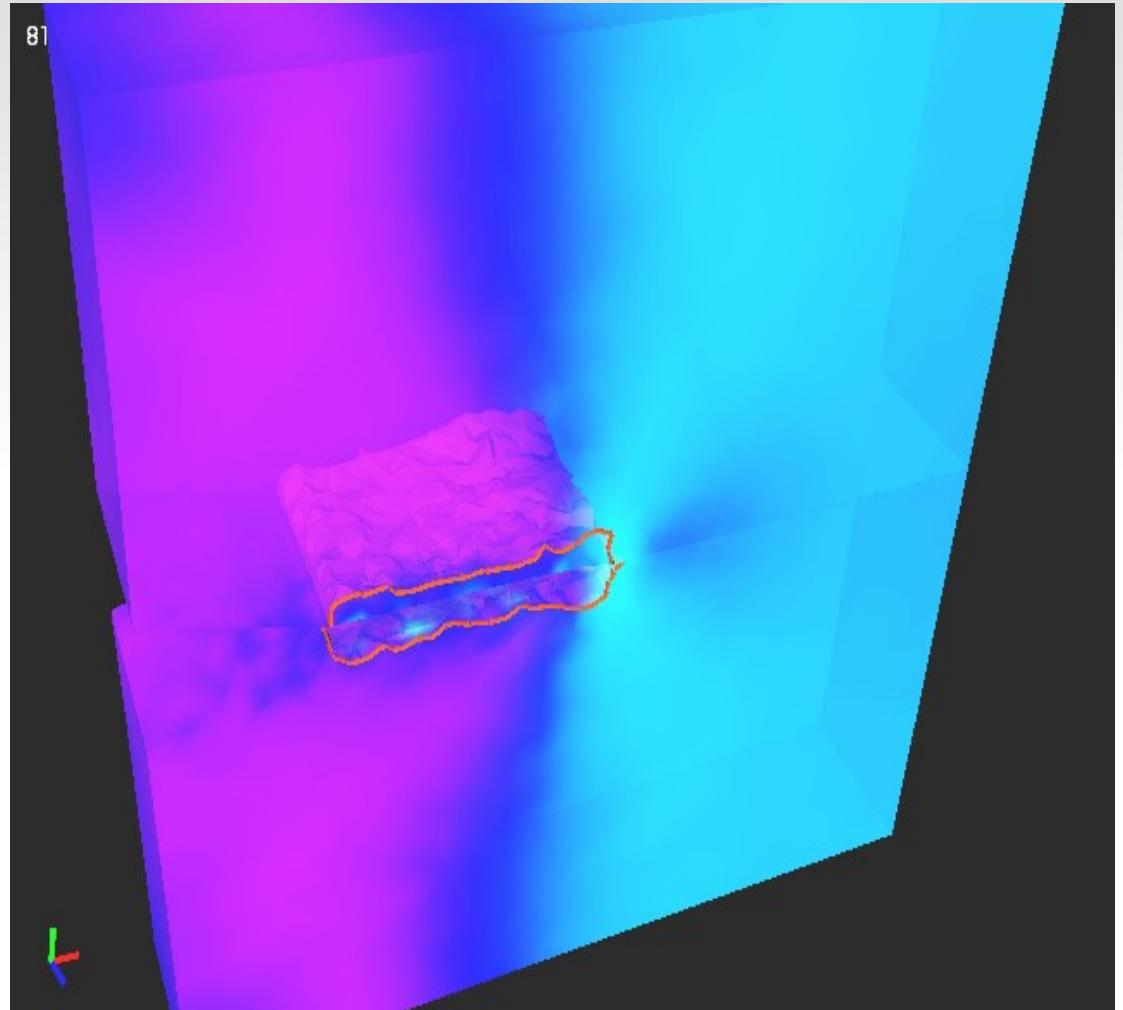
Performance Challenges

- Computational load change with physical state change
- Mesh adaptivity
- Cohesive finite elements
- Contact
- Multi-scale simulations
- ➔ Irregular problems -> Load Balancing

Dynamic Changes in Computational Load

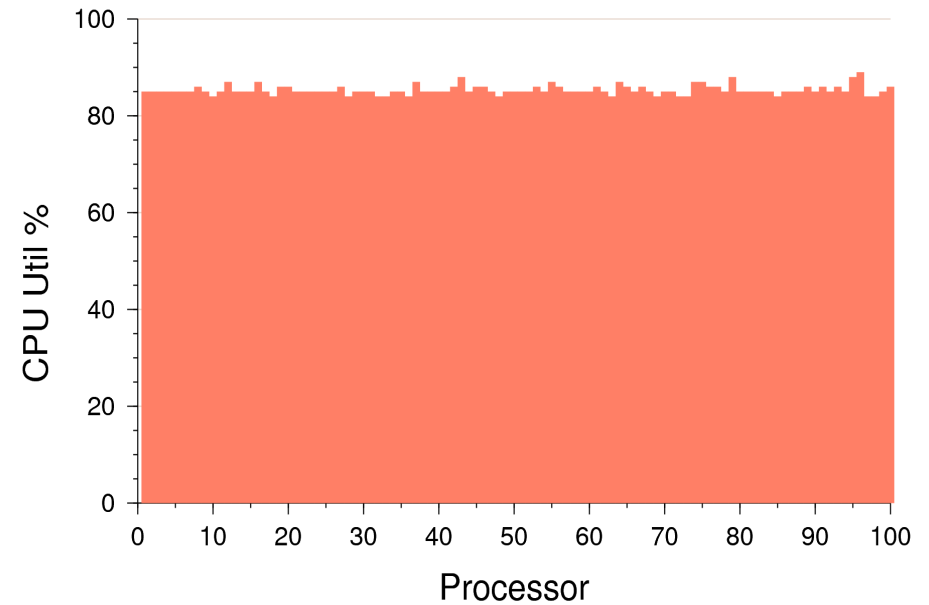
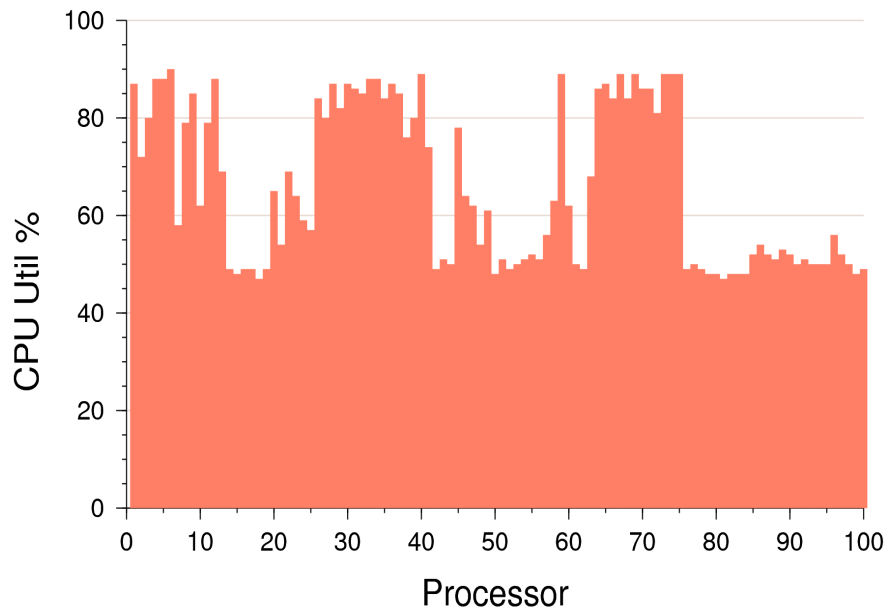


[G. Zheng, M. Breitenfeld, H. Govind,
P. Geubelle, L. Kale]



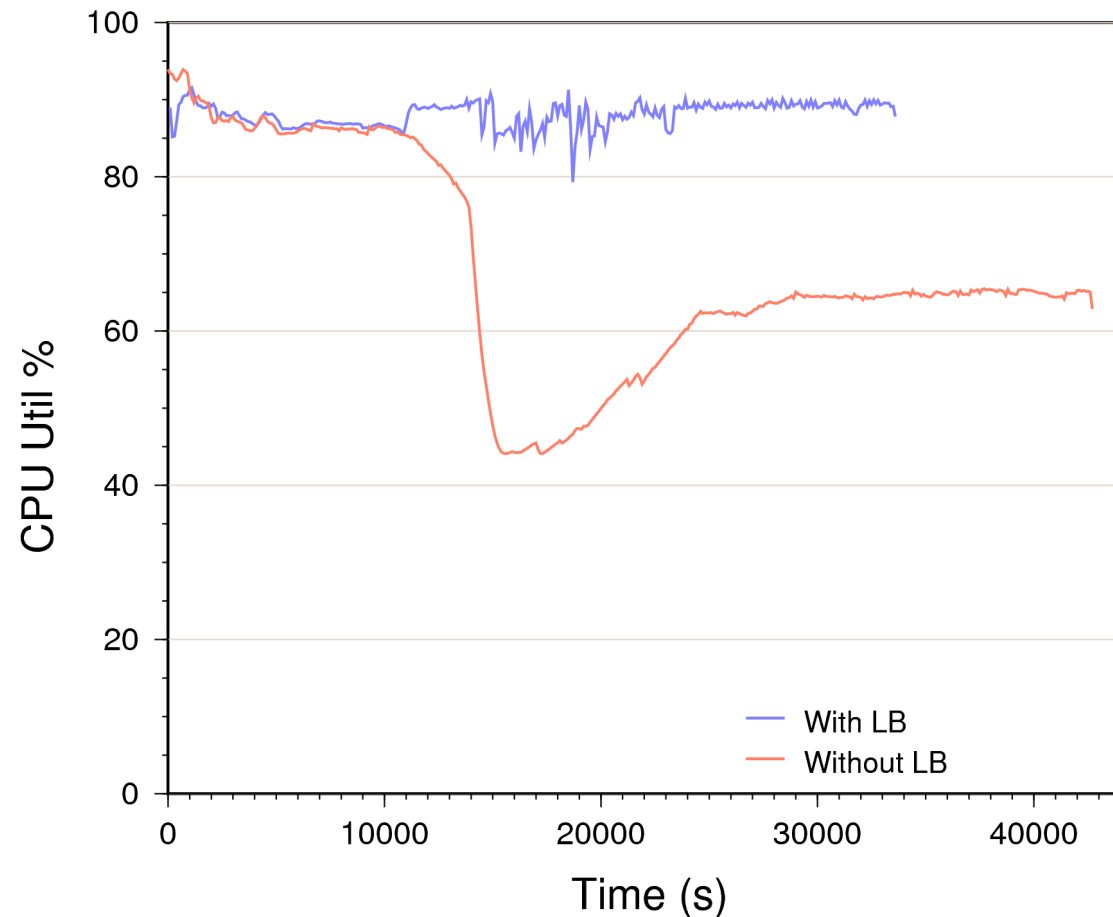
Dynamic Fracture

- Periodic load balancing



Dynamic Fracture

- Periodic load balancing



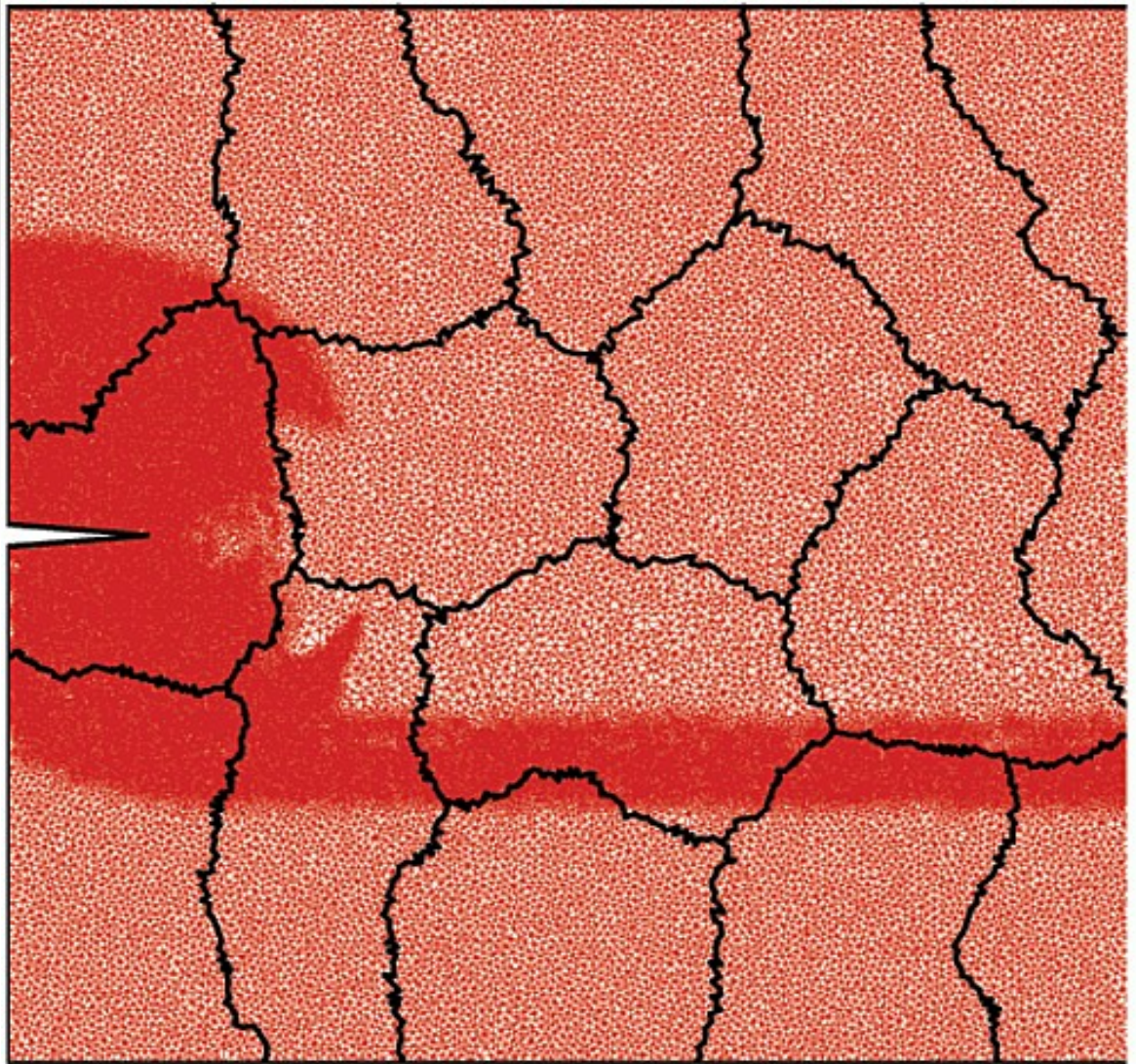
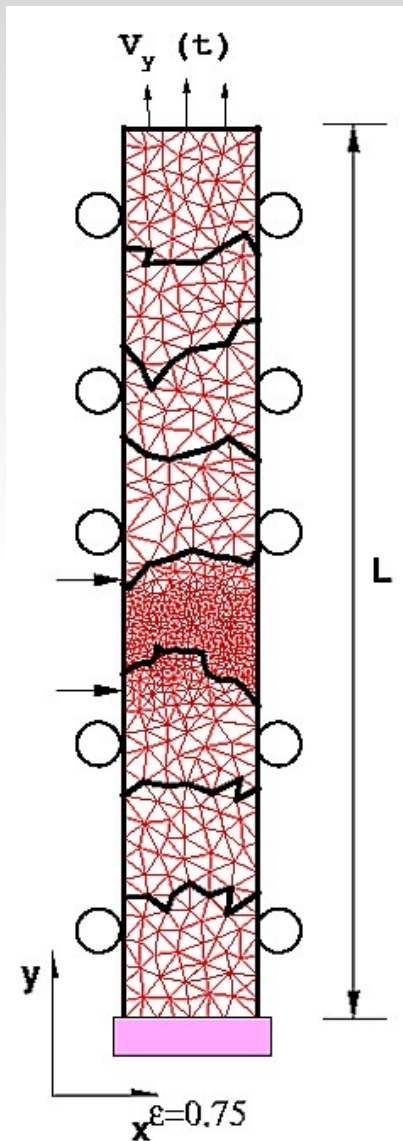
Mesh Adaptivity

- Two approaches in ParFUM
 - Incremental adaptivity (2D triangle meshes)
 - edge bisection, edge contraction, edge flip
 - supported in meshes with 1 layer of edge-neighbor ghosts
 - each individual operation leaves mesh consistent
 - used in SDG code [A. Becker, R. Haber, et al]
 - Bulk adaptivity (2D triangle, 3D tetrahedral meshes)
 - edge bisection, edge contraction, edge flips
 - supported in meshes with any or no ghost layers
 - operations performed in bulk; ghosts and adjacencies updated at end

Mesh Adaptivity

- Higher level operations [T. Wilmarth, A. Becker]
 - Refinement: longest edge bisection
 - Coarsening: shortest edge contraction
 - Smoothing
 - Optimization
 - Mesh gradation
 - Scaling
- User sets sizing on mesh entities as desired

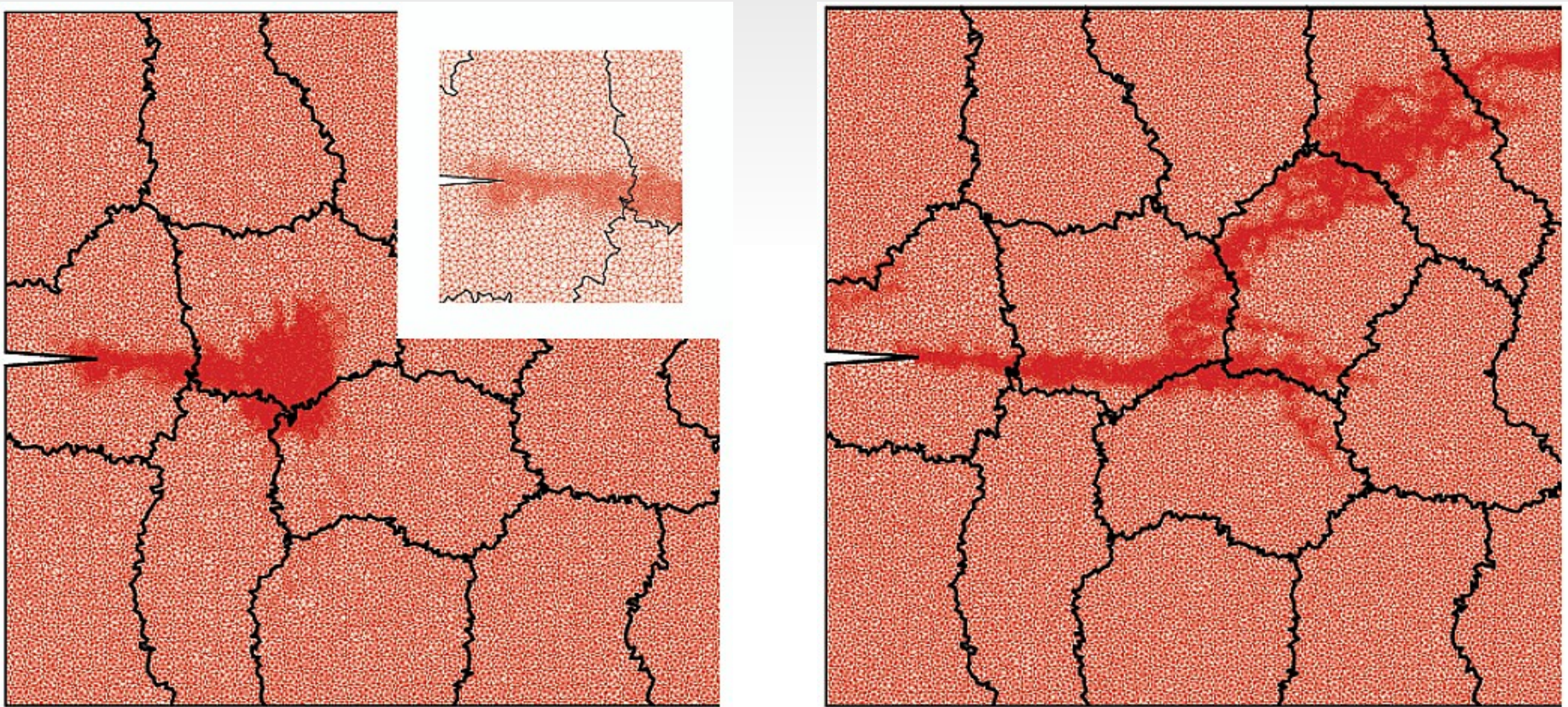
Mesh Adaptivity



[S. Mangala, T. Wilmarth, S. Chakravorty, N. Choudhury, L. Kale, P. Geubelle]

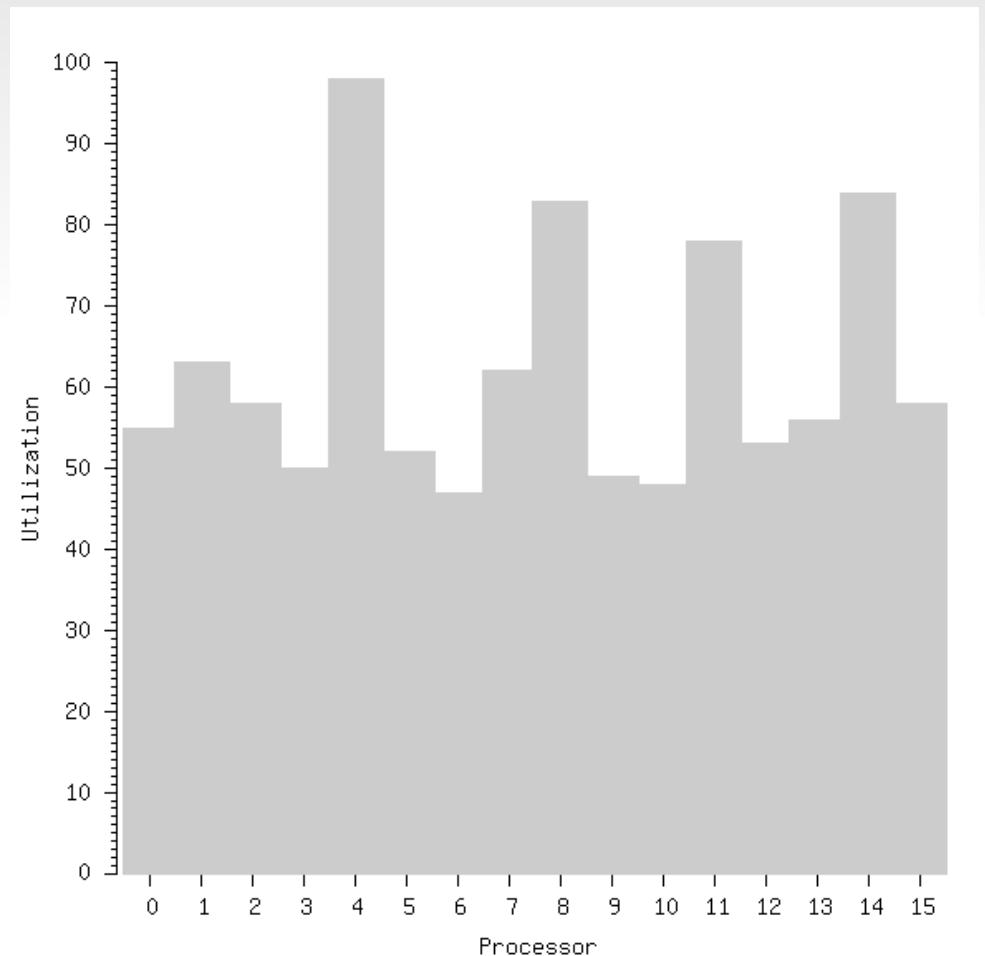
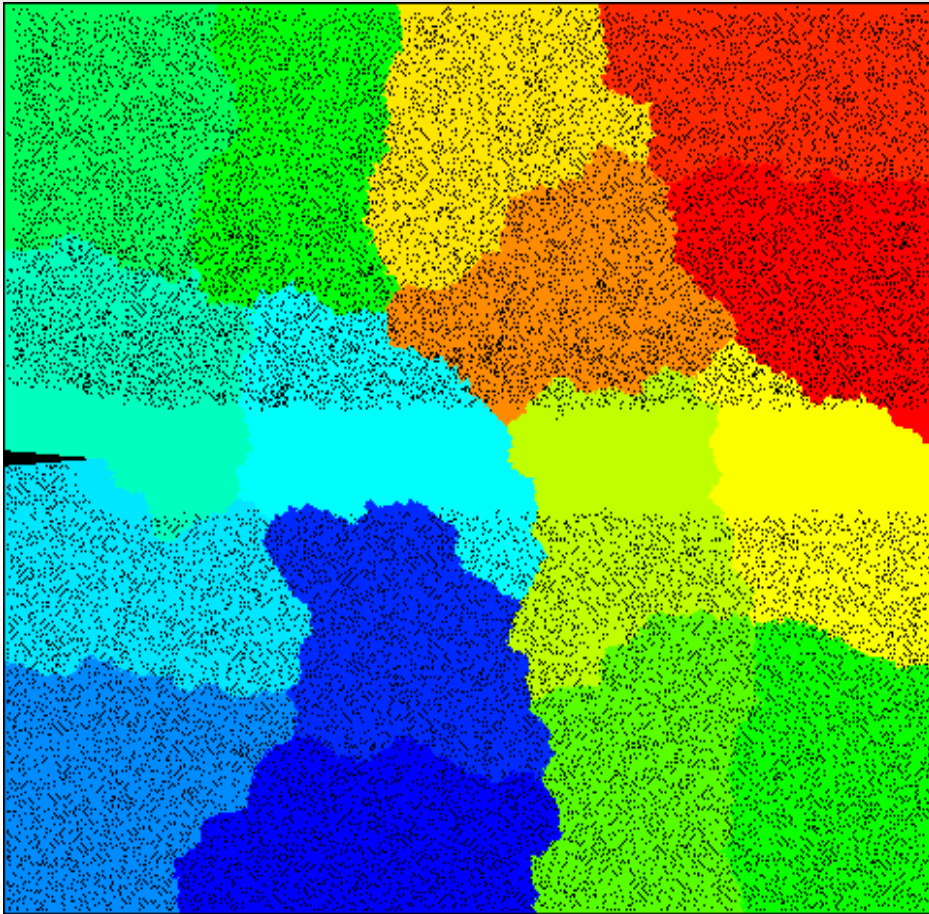
Dynamic Fracture

- Accurately capture failure process



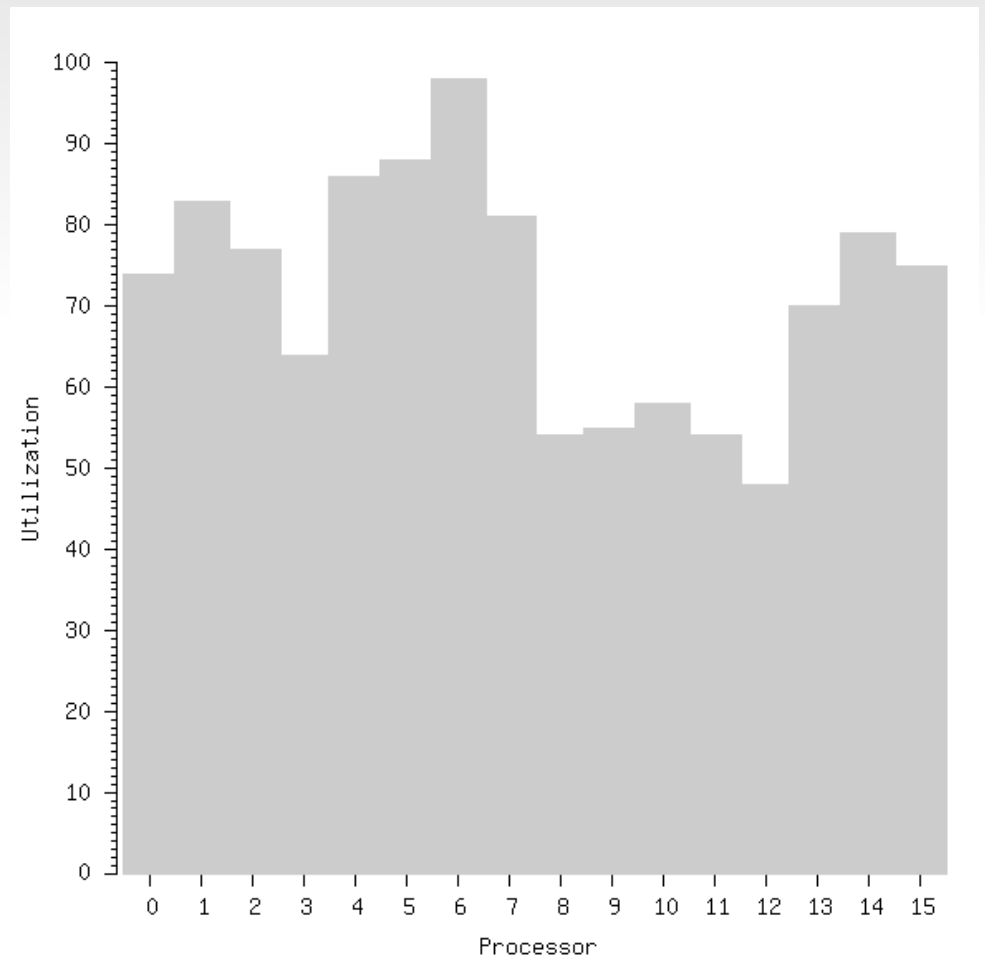
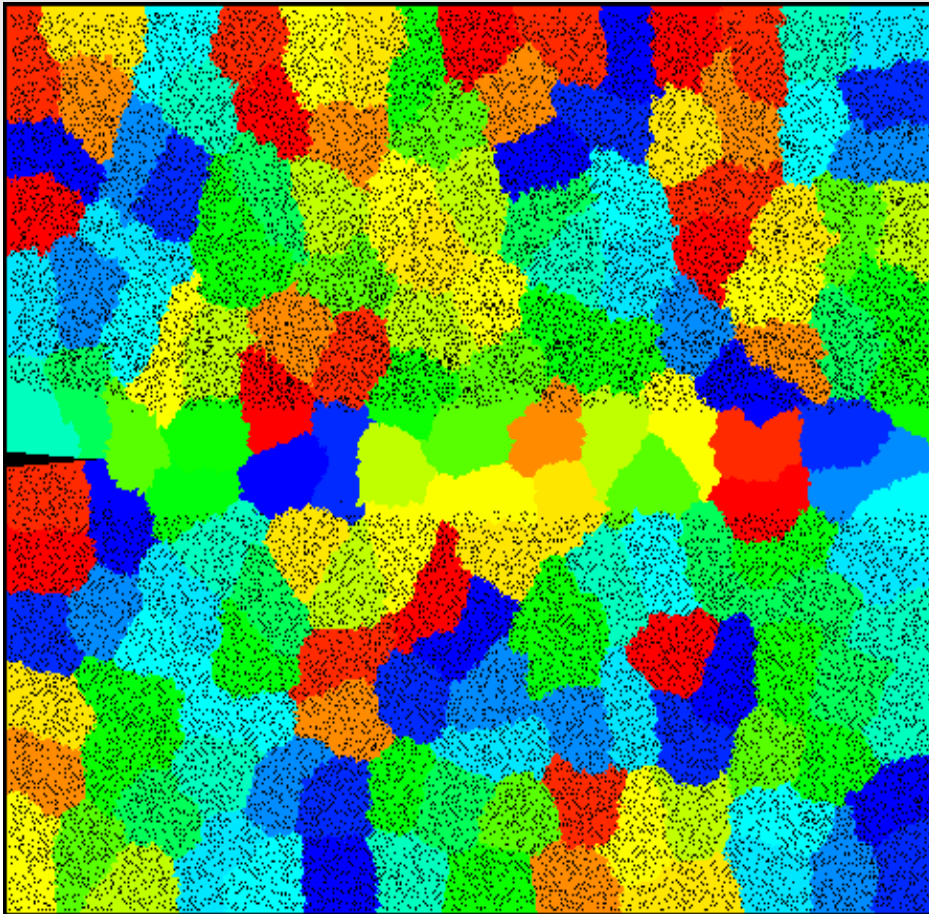
Dynamic Fracture

- Severe load imbalance



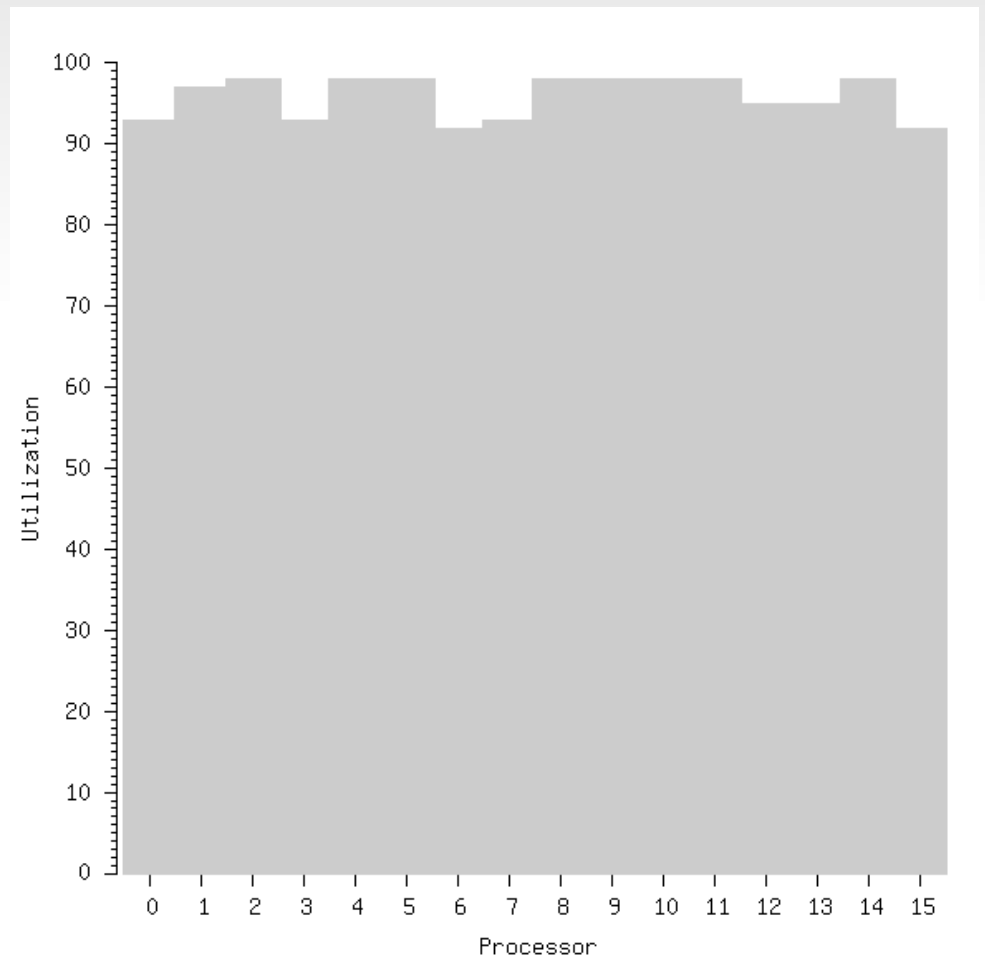
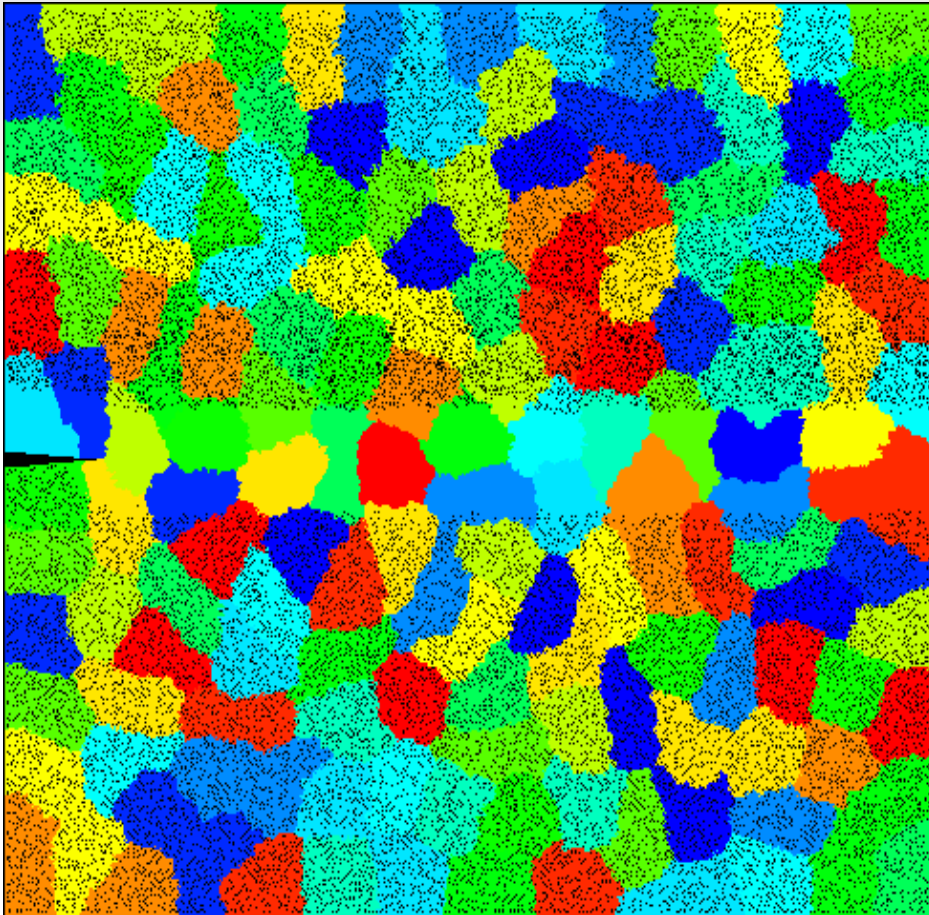
Dynamic Fracture

- Change VP mapping



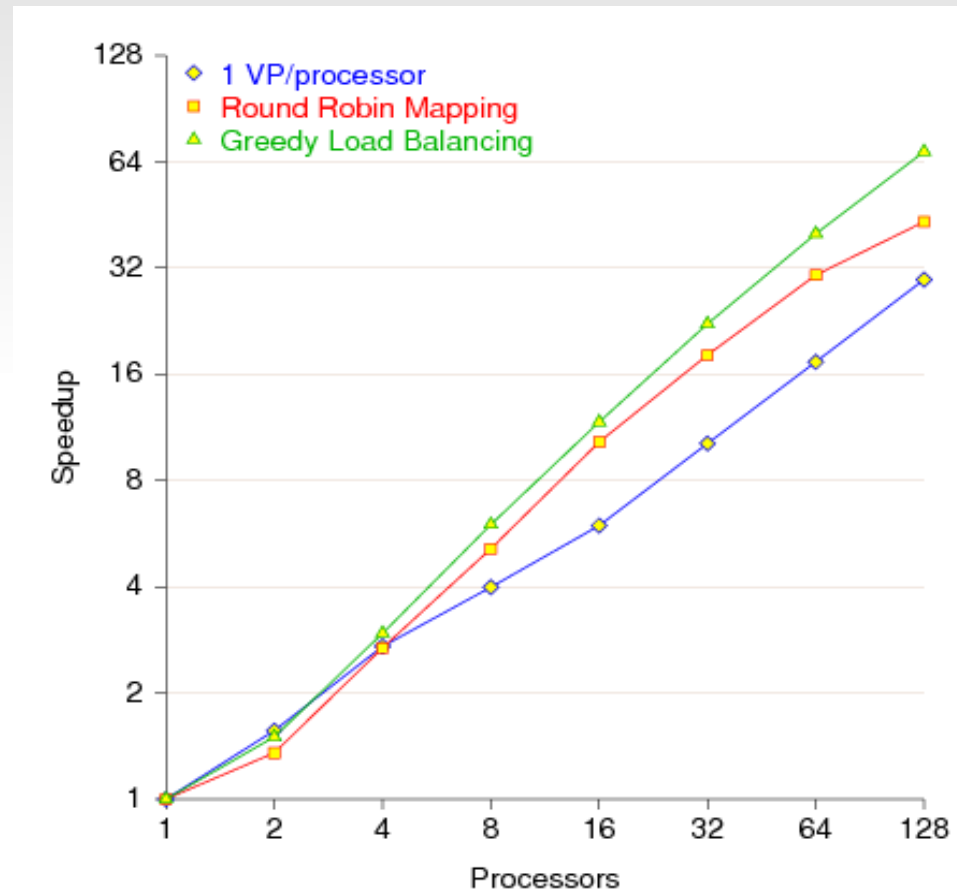
Dynamic Fracture

- Load balancing, greedy strategy, applied after mesh adaptation (every 2000 timesteps)



Dynamic Fracture

- Preliminary performance for adaptive application



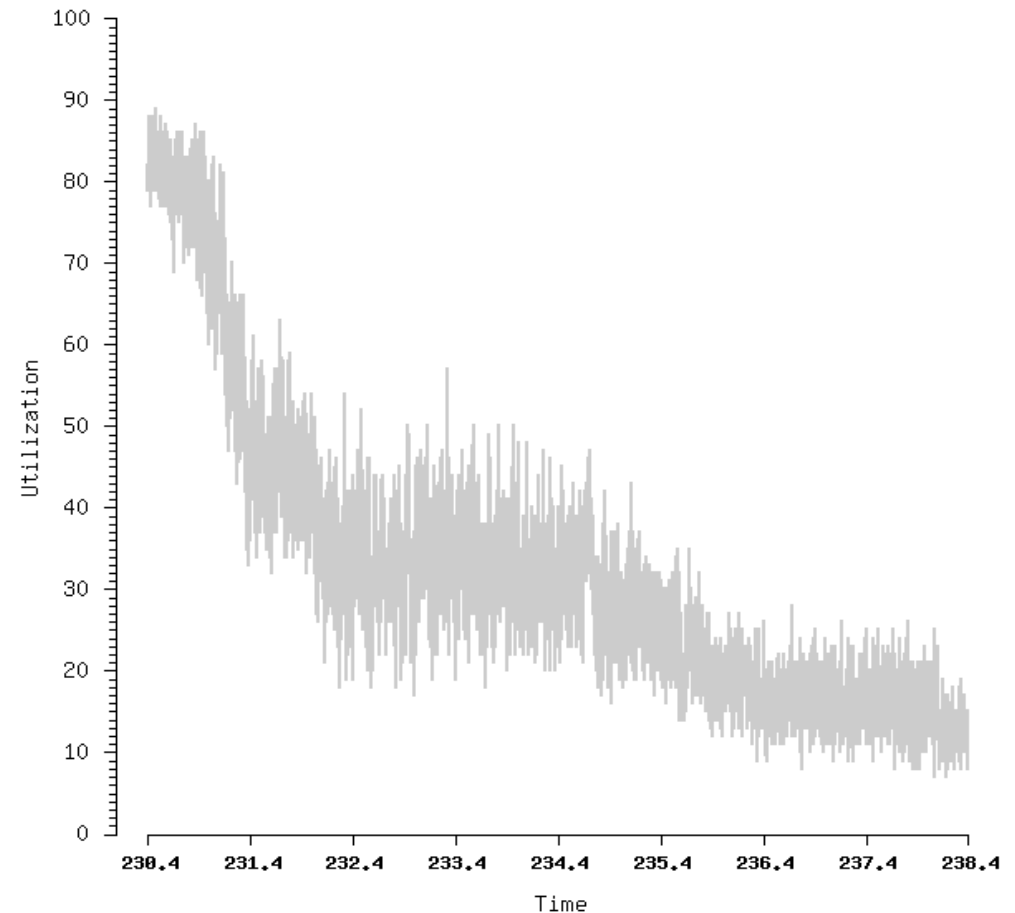
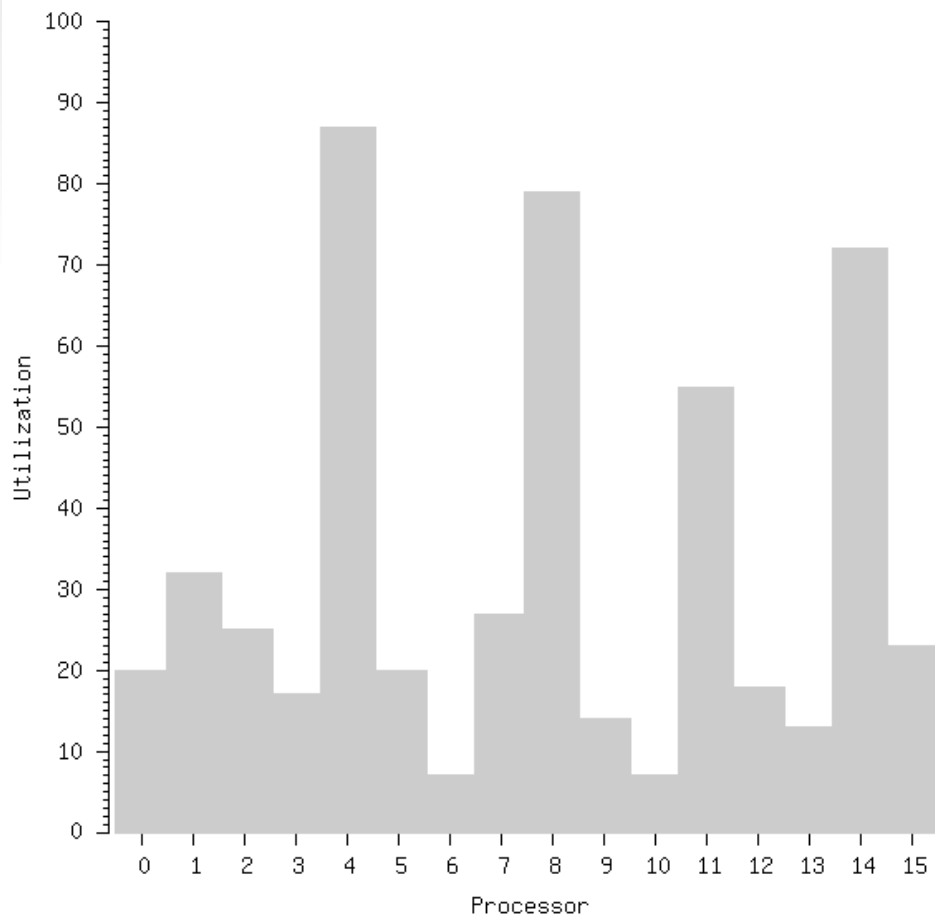
Dynamic Fracture

- Load balancing after mesh adaptivity results in excellent performance during computation phase
- What about adaptivity phase?

Timesteps	Compute	Adapt	Total	% Adapt
1-2000	17.047	12.967	30.014	43.203
2001-4000	32.873	9.837	42.710	23.032
4001-6000	36.484	17.073	53.557	31.878
6001-8000	41.186	15.756	56.942	27.670
8001-10000	46.644	14.886	61.530	24.193
10001-12000	50.831	17.302	68.133	25.394
12001-14000	59.155	21.215	80.370	26.396
14001-16000	68.703	25.102	93.805	26.759
16001-18000	76.503	19.897	96.400	20.640
18001-20000	81.763	NA	81.763	NA

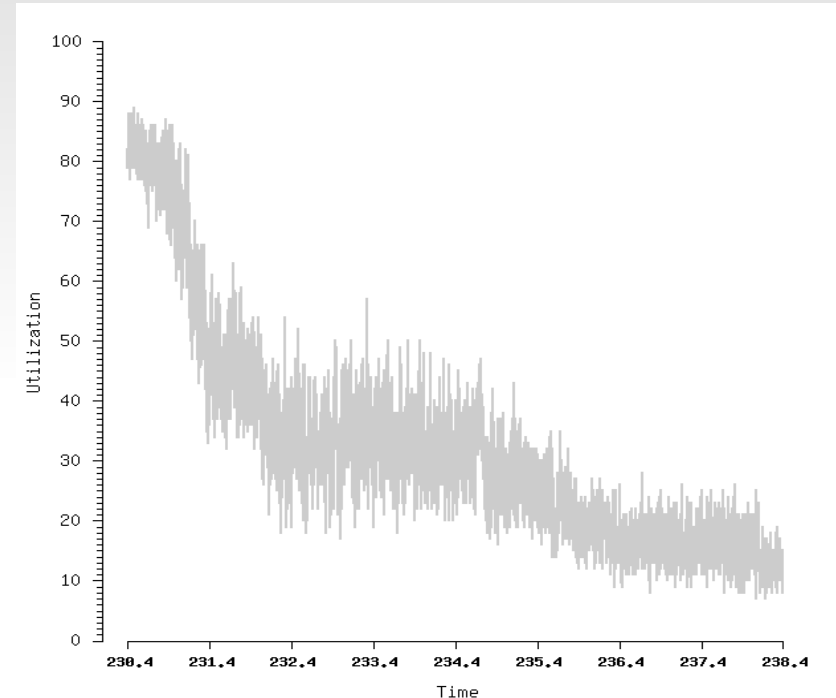
Mesh Refinement Phase

- Extreme load imbalance
- Peak utilization at start of phase



Mesh Refinement Phase

- How to balance load?
 - Principle of persistence does not hold for instrumentation
 - Phase is too short to instrument and to call load balancer repeatedly
 - We have domain-specific knowledge of what will be refined
 - We can estimate the load on a partition prior to mesh modification

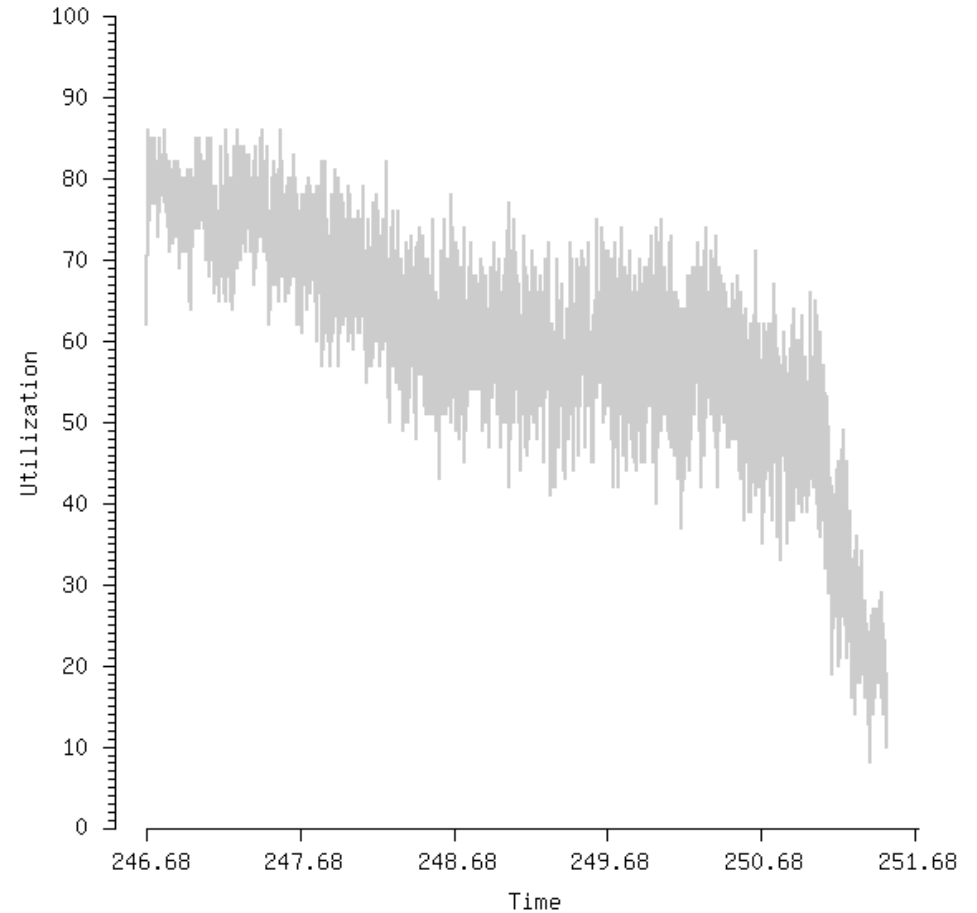
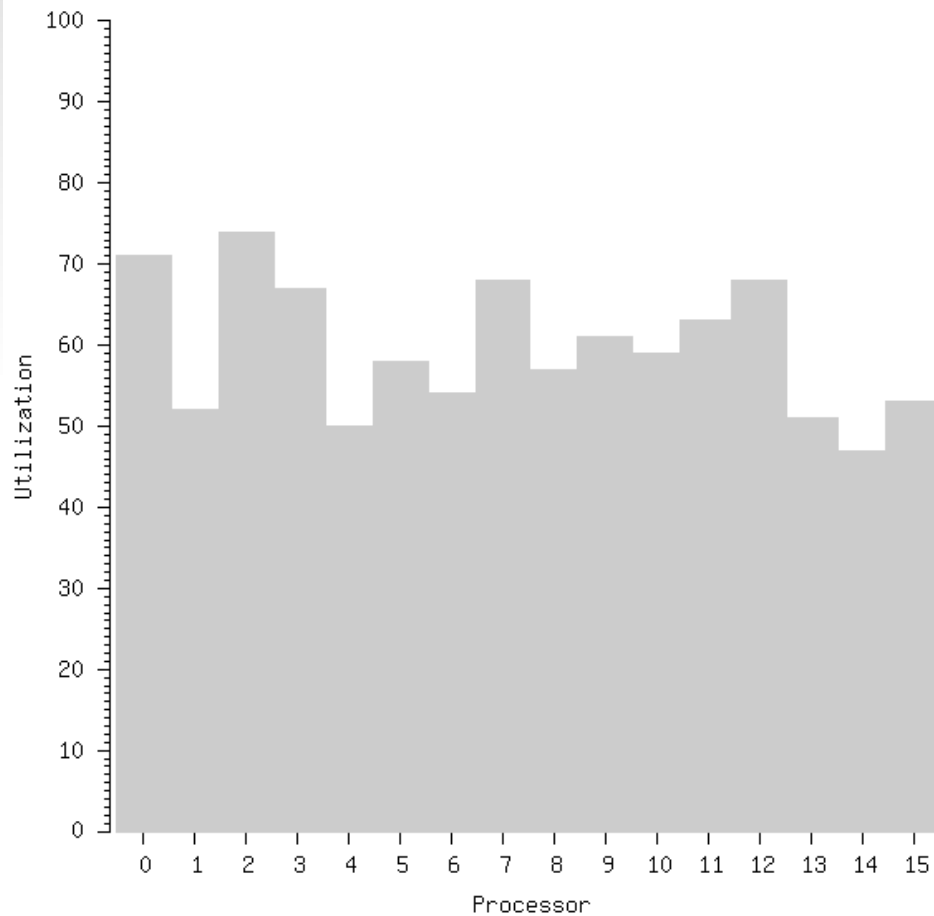


Mesh Refinement Phase

- Pre-balancing: model-based load balancing [S. Chakravorty, T. Wilmarth]
 - ParFUM uses user-specified mesh adaptation parameters to measure potential load during adaptivity (no other instrumentation)
 - Passes load information to Charm++ Run-time System, which then migrates VPs appropriately
 - When migration is finished, adaptivity phase commences
 - Still essentially automatic (no user input required)

Mesh Refinement Phase

- Mesh refinement phase performance improves



Mesh Refinement Phase

- Coarsening component of adaptivity phase is equally costly
- Pre-balancing by refinement criteria insufficient
- Cost of pre-balancing is low
- Incremental adaptivity is not appropriate for this degree of mesh modification

	W/o Pre-balancing		With Pre-balancing		
P	Refine	Coarsen	Refine	Coarsen	Cost
8	118.237	128.561	110.876	129.387	7.342
16	76.773	83.670	58.096	89.225	5.226
32	43.623	45.084	30.586	49.840	3.815
64	32.123	27.126	22.730	52.507	3.385

Bulk Mesh Adaptivity

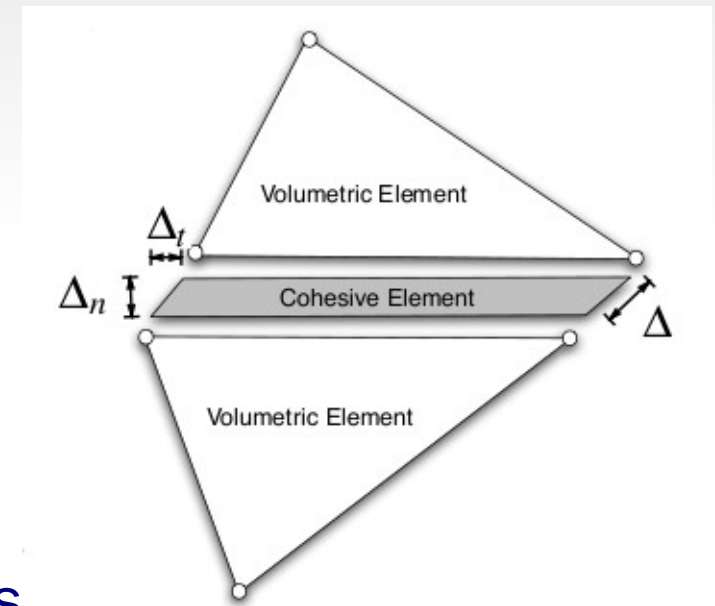
- Fast parallel algorithm for edge bisect in 3D when edge is on partition boundary: requires four asynchronous multicasts in average case
- Allows parallel operations on disjoint sets of neighboring partitions
- Uses element adjacency information based on globally unique element IDs
- Maintains consistent shared entities
- Ghost layers and user adjacencies updated at end of bulk mesh modification

Bulk Mesh Adaptivity: Ongoing

- Fast parallel algorithm for edge contract in 3D (will be much like edge bisection)
- Fast parallel edge flipping operations (for mesh optimizations)
- Re-implement existing refinement, coarsening and optimization algorithms (currently using incremental)
- Add domain boundary preservation
- [T. Wilmarth, A. Becker, S. Chakravorty]

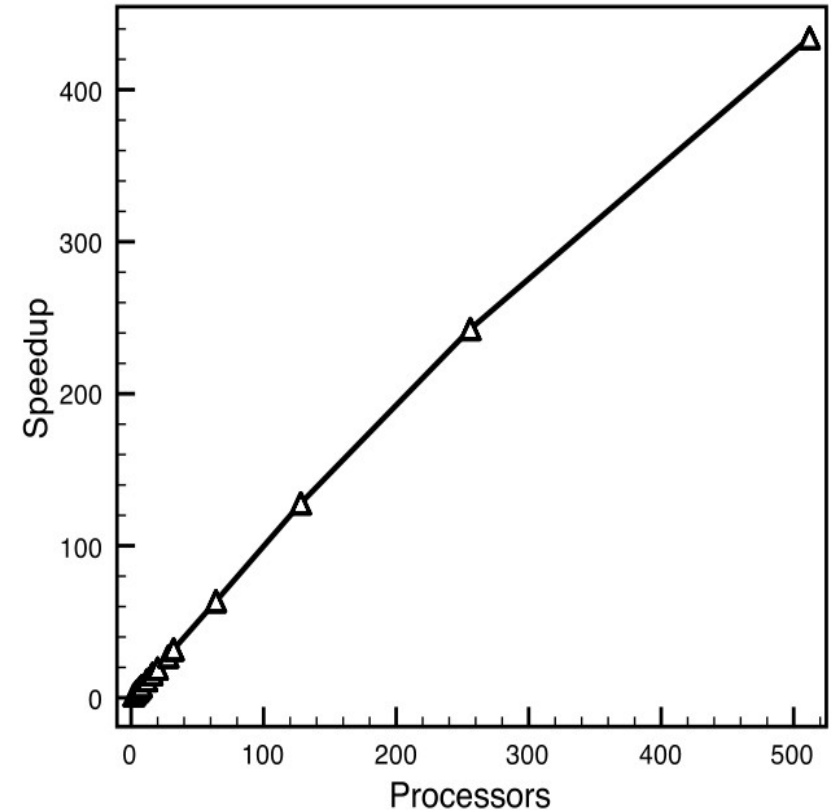
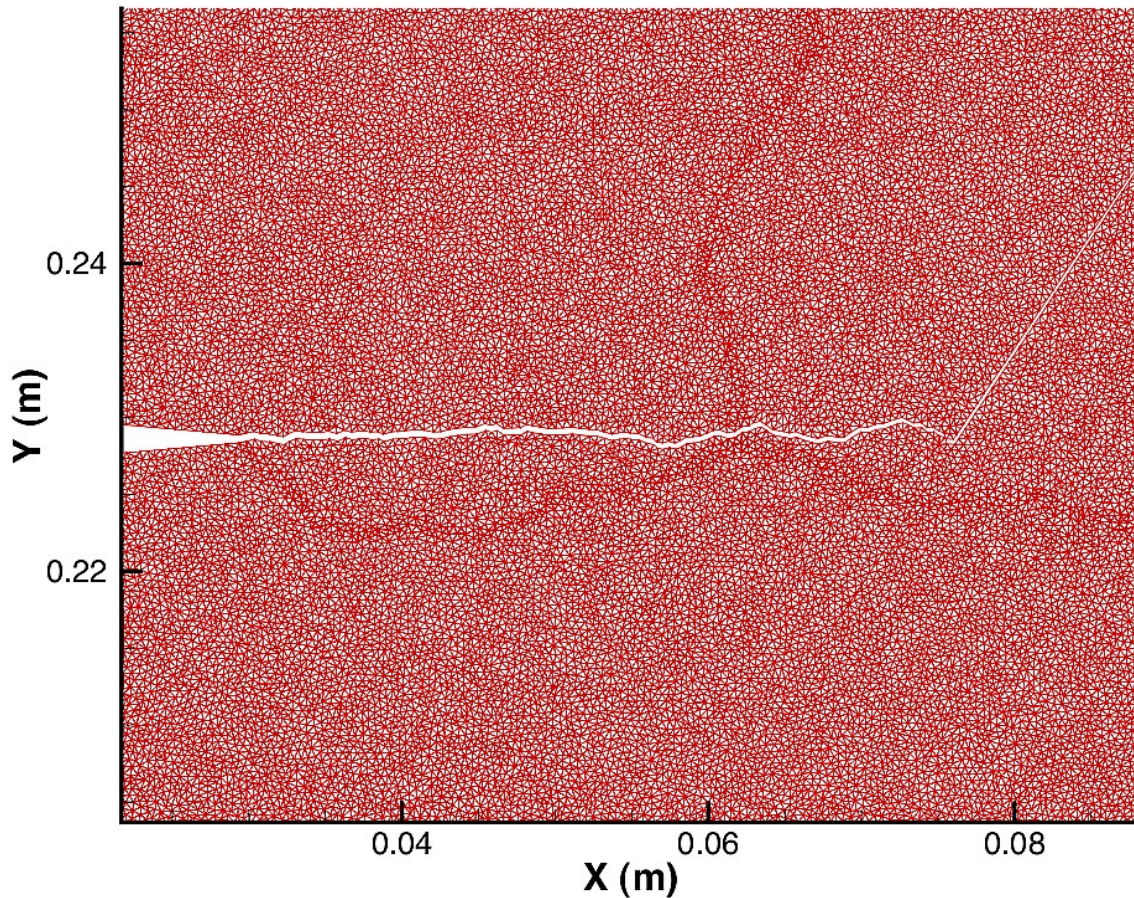
Cohesive Finite Elements

- CFEs model progressive material failure and propagation of cracks through domain
- Located at interfaces between volumetric elements
- Two schemes:
 - Intrinsic: everpresent contributors to deformation
 - Extrinsic: introduced based on external traction-based criterion
 - Activated extrinsic: everpresent CFEs do not contribute until “activated” [S. Mangala, P. Geubelle, I. Dooley, L. Kale]



Cohesive Finite Elements

- Initial performance with activated CFEs: no load imbalance



Cohesive Finite Elements: Ongoing

- Insertion of extrinsic CFEs as needed [I. Dooley, A. Becker, T. Wilmarth, G. Paulino, K. Park]
- Will result in load imbalance as crack passes through partitions
- Dynamic fracture simulation needs:
 - fine mesh near failure zone to capture stress concentrations accurately
 - large domain to accurately capture loading and avoid wave reflections from boundary
 - Dynamic mesh adaptation in mesh with mix of volumetric and cohesive elements

Contact: Ongoing

- Detect when domain fragments come into contact
- Uses Charm++ Collision Detection [O. Lawlor]
- Potential for load imbalance:
 - Only partitions with domain boundary participate
 - Only domain boundary elements can collide
 - Fragment movement problem (bounding box too large); may require repartitioning
 - Element collisions between pairs of partitions can be distributed to idle processors

Future Directions

- Load balancing enhancements:
 - model-based LB with bulk adaptivity
- Dynamic repartitioning:
 - A full repartitioning to same number of partitions can balance load, but...
 - Maintain ideal VP size: partition VPs that grow too large (less expensive than full repartitioning); increases the number of partitions!
- Multi-scale simulation: many interesting load balancing problems

Closing Remarks

- ParFUM software available:
 - <http://charm.cs.uiuc.edu/download>
- Charm++ Workshop, May 1st - 3rd
 - <http://charm.cs.uiuc.edu/charmWorkshop>
 - ParFUM tutorial: 3rd May, 9:00am