Parallel, Adaptive Scientific Computation in Heterogeneous, Hierarchical, and Non-Dedicated Computing Environments

Jim Teresco

Williams College

OMNINO UNIS NIHILISQUE FIT

Department of Computer Science

Department of Computer Science
Williams College
Williamstown, Massachusetts

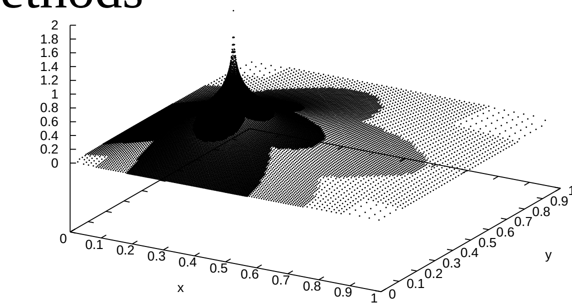
National Institute of Standards and Technology
Mathematical & Computational Sciences Division Seminar Series

June 15, 2006

Yet another Powerpoint-free presentation!

Overview

- Why parallel computing?
  - solve larger problems in less time: clusters, supercomputers
  - recent trends: clock speed increases slowing, more processors per node
- Target computational paradigm: parallel adaptive methods
  - distributed data structures and partitioning
  - dynamic load balancing algorithms
  - load balancing software: Zoltan Toolkit



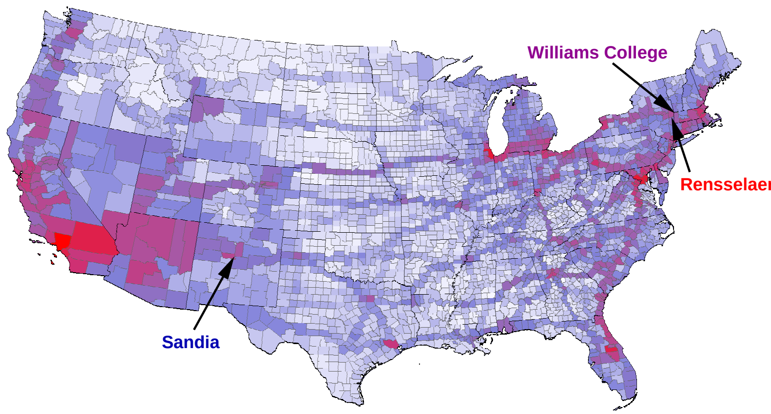
- Heterogeneous, hierarchical and non-dedicated computing environments
  - target environments, including Bullpen cluster
  - what can be adjusted? who can make the adjustments?
  - what can we do at just the load balancing step?

Rensselaer
Williams
D R U M

- Resource-aware parallel computation
  - Dynamic Resource Utilization Model (DRUM)
  - other approaches: hierarchical partitions, process migration, operating system migration

Participants

- Rensselaer Polytechnic Institute
  - Ph.D. students: Jamal Faik (now at Oracle), Luis Gervasio
  - Faculty: Joseph Flaherty
  - Undergraduates: Jin Chang
  - Various SCOREC students/postdocs/staff



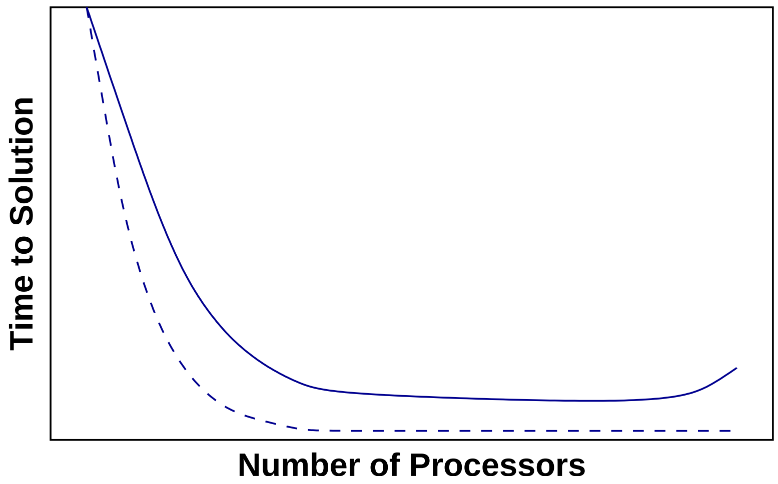
- Sandia National Laboratories
  - Karen Devine and the Zoltan group
- Williams College undergraduates
  - Most recent summers: Laura Effinger-Dean '06, Arjun Sharma '07, Bartley Tablante '07
  - Previous: Kai Chen '04, Lida Ungar '02, Diane Bennett '03
  - 2006 honors thesis student: Travis Vachon '06

Why Parallel Computation?

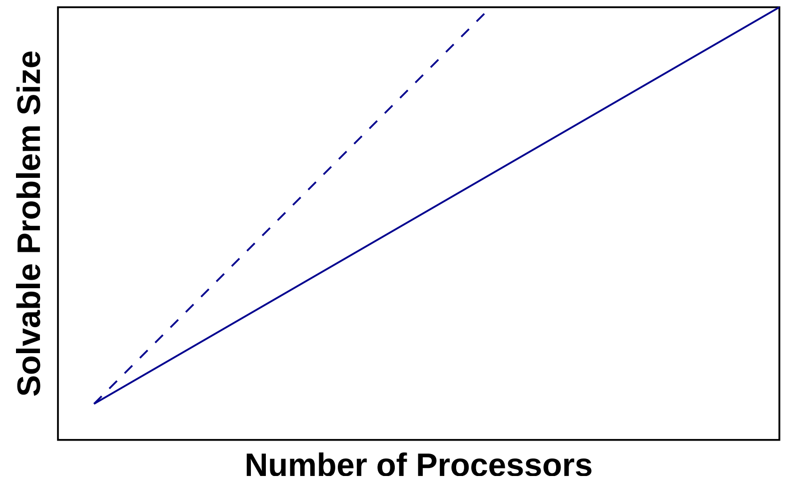
Parallelism adds complexity, so why bother?

Traditionally, there are two major motivations.

Computational speedup



Computational scaling

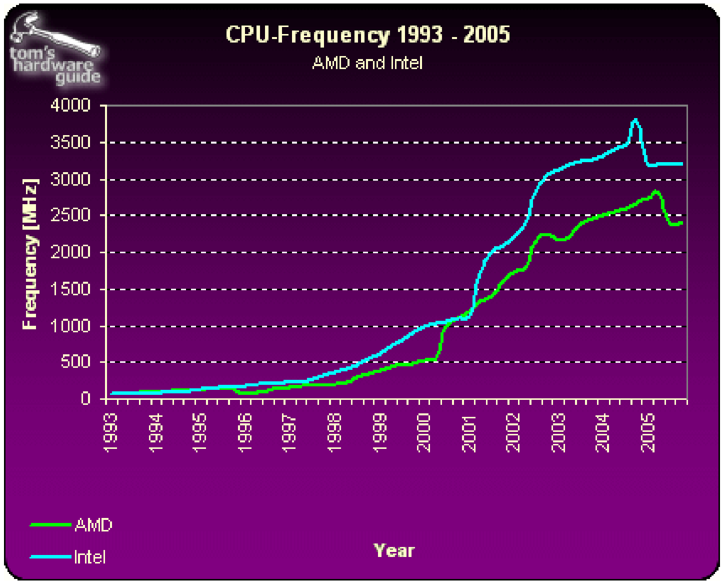


solve the same problem but in less time than on a single processor

solve larger problems than could be solved *at all* on a single processor within time or space constraints

Recent Trends

- Until recently, computational scientists could assume that faster processors were always on the way

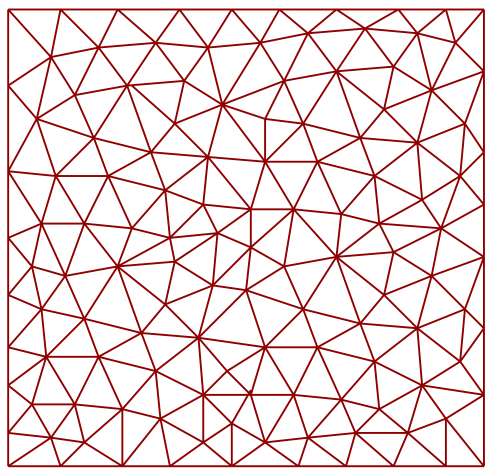


- Manufacturers are hitting the limits of current technology
- Focus now: multiple processors, hyperthreading, multi-core processors
- Today: dual core is common – Soon: 4, 8 or more cores per chip
- Parallel computing is needed to use such systems effectively!

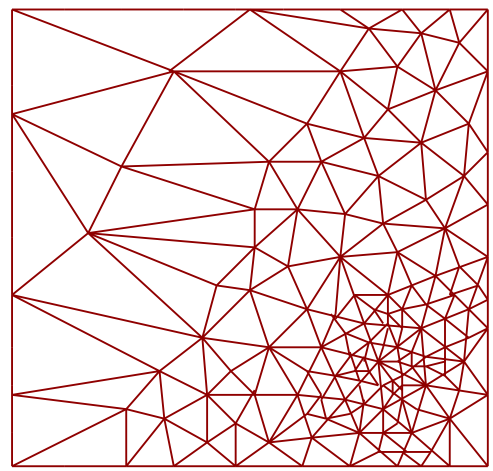
Figure used with permission from article *The Mother of All CPU Charts 2005/2006*, Bert Töpelt, Daniel Schuhmann, Frank Völkel, Tom's Hardware Guide, Nov. 2005, `http://www.tomshardware.com/2005/11/21/the_mother_of_all_cpu_charts_2005/`

Target Applications: Finite Element and Related Methods

- More elements $\Longrightarrow$ better accuracy, but higher cost
- Adaptivity concentrates computational effort where it is needed
- Guided by error estimates or error indicators
- h-adaptivity: mesh enrichment



Uniform mesh

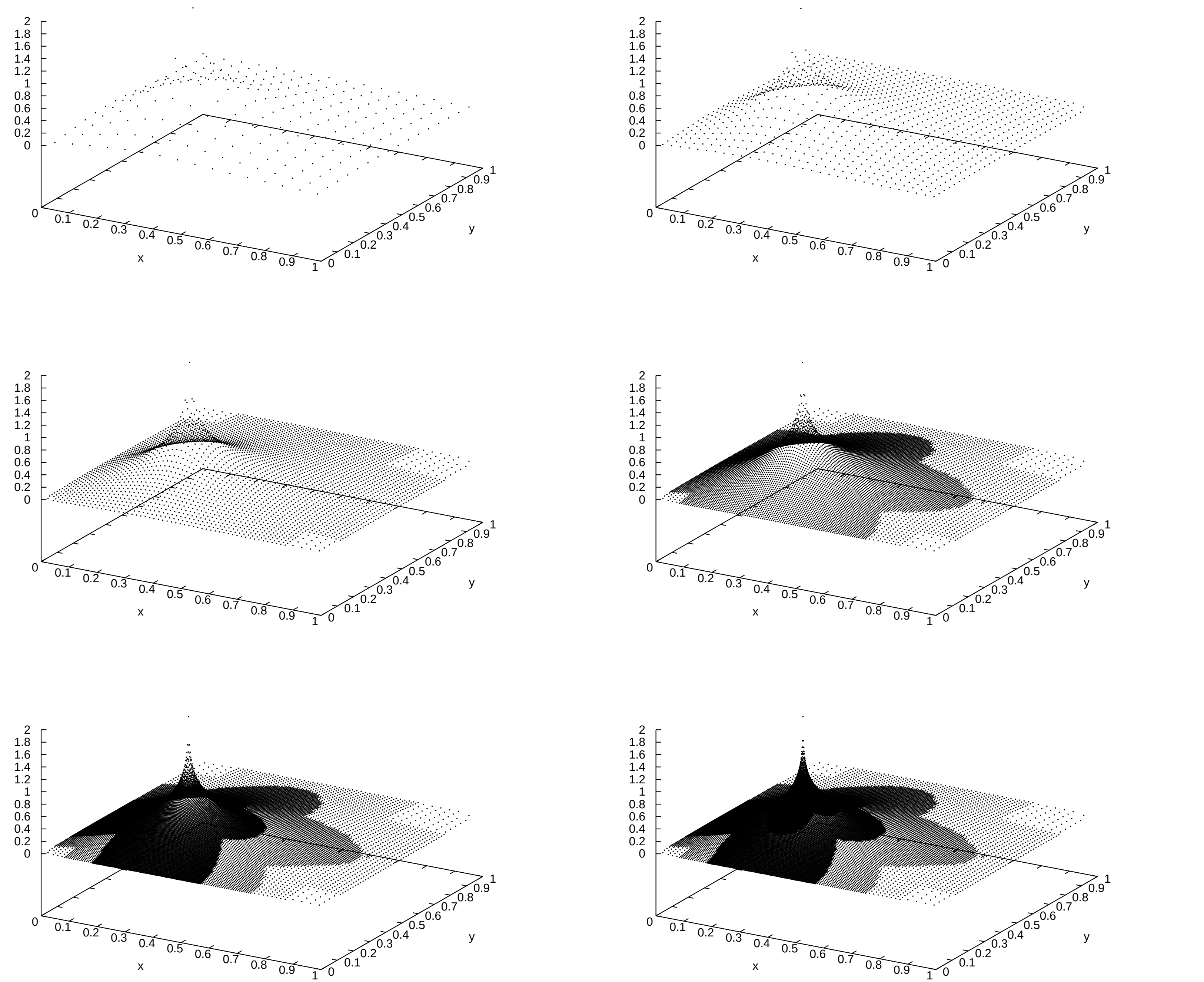


Adapted mesh

- p-adaptivity: method order variation; r-adaptivity: mesh motion
- Local refinement method: time step adaptivity
- Adaptivity is essential

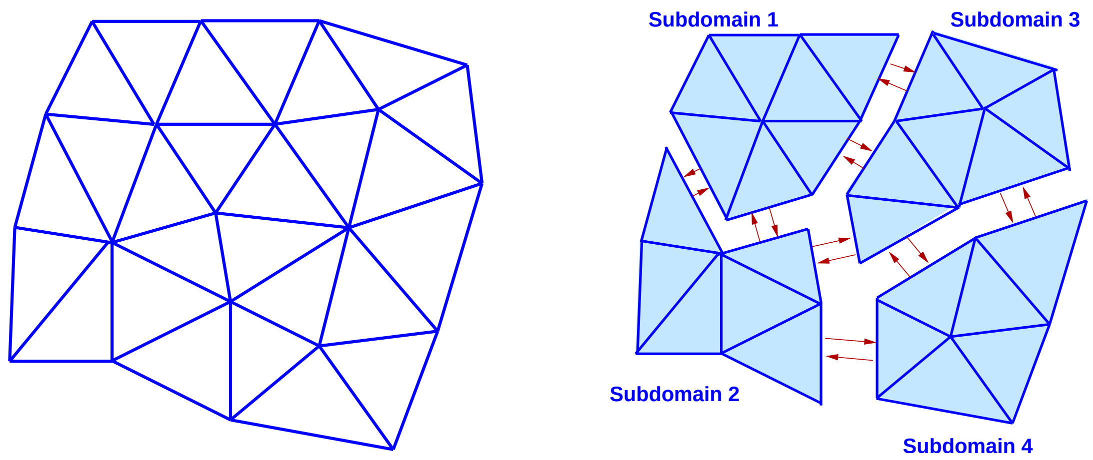
A Simple Adaptive Computation

Refine the underlying mesh to achieve desired accuracy



Parallel Strategy

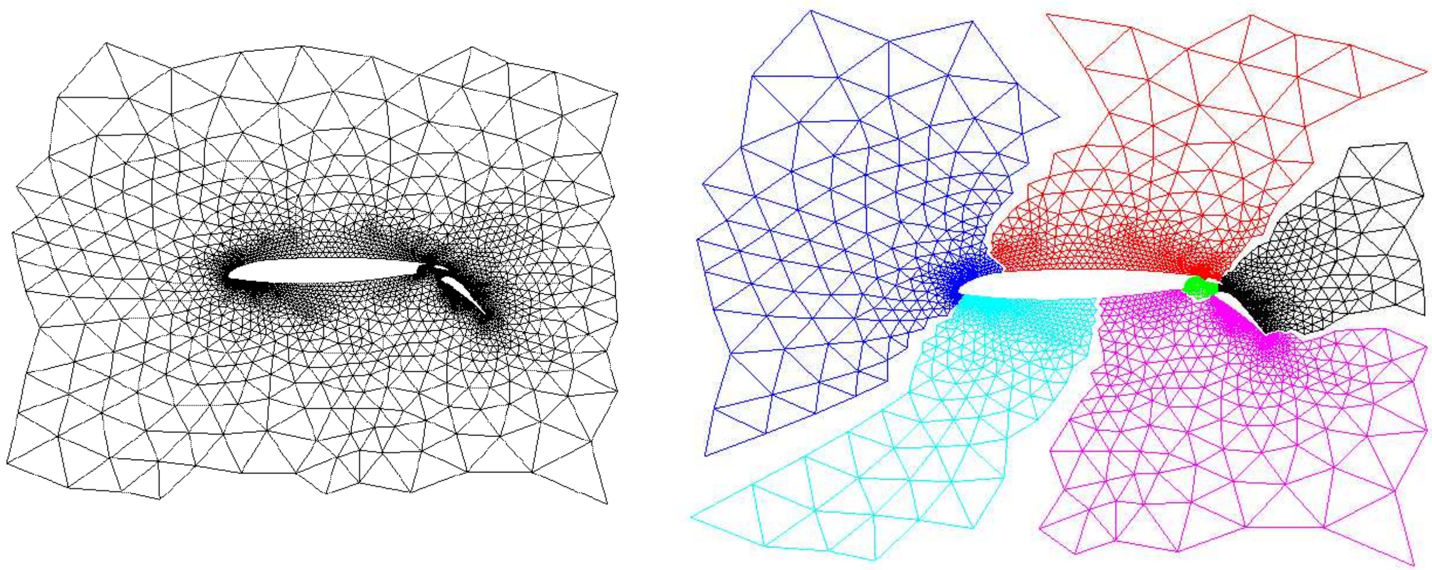
- Dominant paradigm: Single Program Multiple Data (SPMD)
  - distributed memory; communication via message passing (usually MPI)
- Can run the same software on shared and distributed memory systems
- Adaptive methods lend themselves to linked structures
  - automatic parallelization is difficult
- Must explicitly distribute the computation via a domain decomposition



- Distributed structures complicate matters
  - interprocess links, boundary structures, migration support
  - very interesting issues, but not today’s focus

Mesh Partitioning

- Determine and achieve the domain decomposition

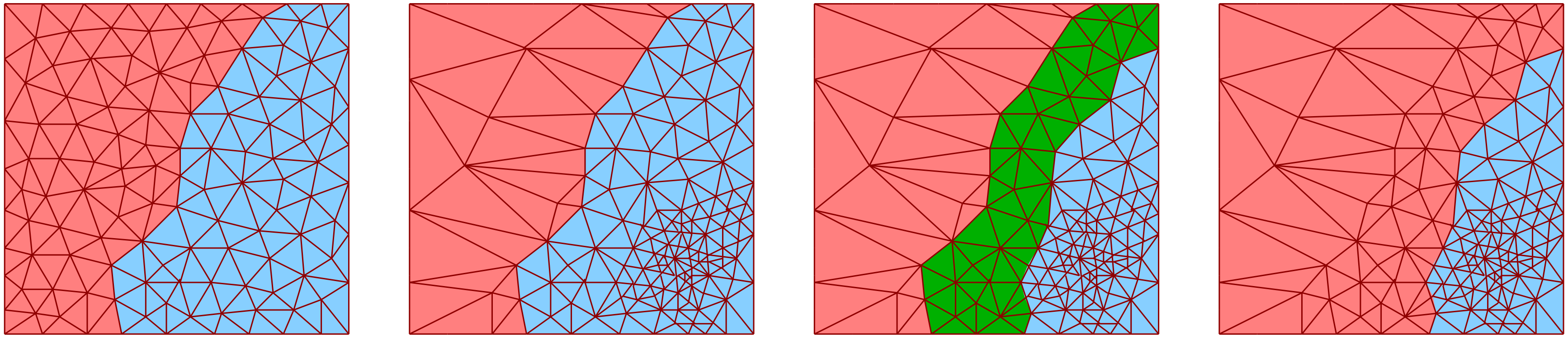


- "Partition quality" is important to solution efficiency
  - evenly distribute mesh elements (computational work)
  - minimize elements on partition boundaries (communication volume)
  - minimize number of "adjacent" processes (number of messages)
- But.. this is essentially graph partitioning: *"Optimal" solution intractable!*

Why dynamic load balancing?

Need a rebalancing capability in the presence of:

- Unpredictable computational costs
  - Multiphysics
  - Adaptive methods



Initial balanced partition | Adaptivity introduces imbalance | Migrate as needed | Rebalanced partition

- Non-dedicated computational resources
- Heterogeneous computational resources of unknown relative powers

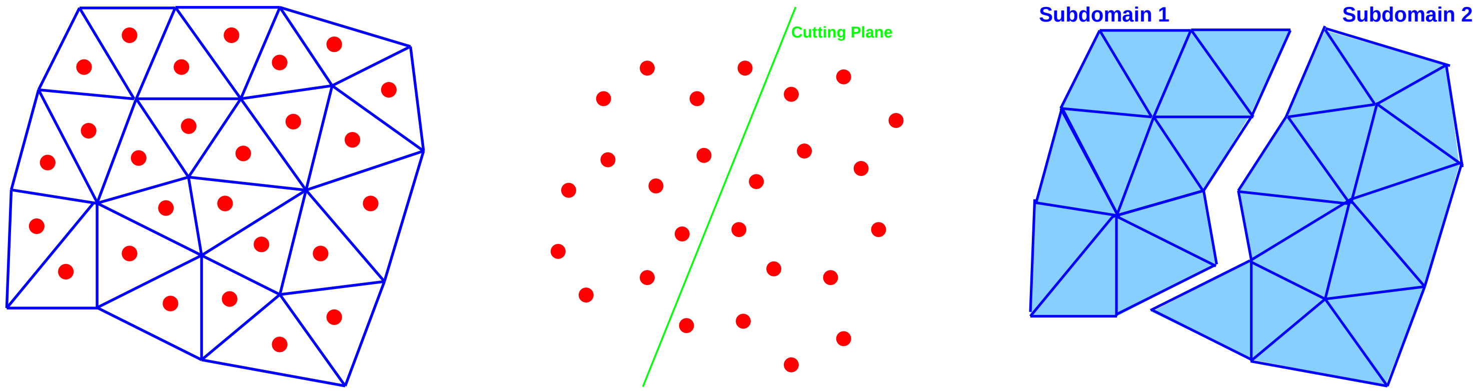
Load Balancing Considerations

- Like a partitioner, a load balancer seeks
  - computational balance
  - minimization of communication and number of messages
- But also must consider
  - cost of computing the new partition
    - may tolerate imbalance to avoid a repartition step
  - cost of moving the data to realize it
    - may prefer incrementality over resulting quality
- Must be able to operate in parallel on distributed input
  - scalability
- It is *not* just graph partitioning – no single algorithm is best for all situations
- Several approaches have been used successfully

Geometric Mesh Partitioning/Load Balancing

Use only coordinate information

- Most commonly use "cutting planes" to divide the mesh



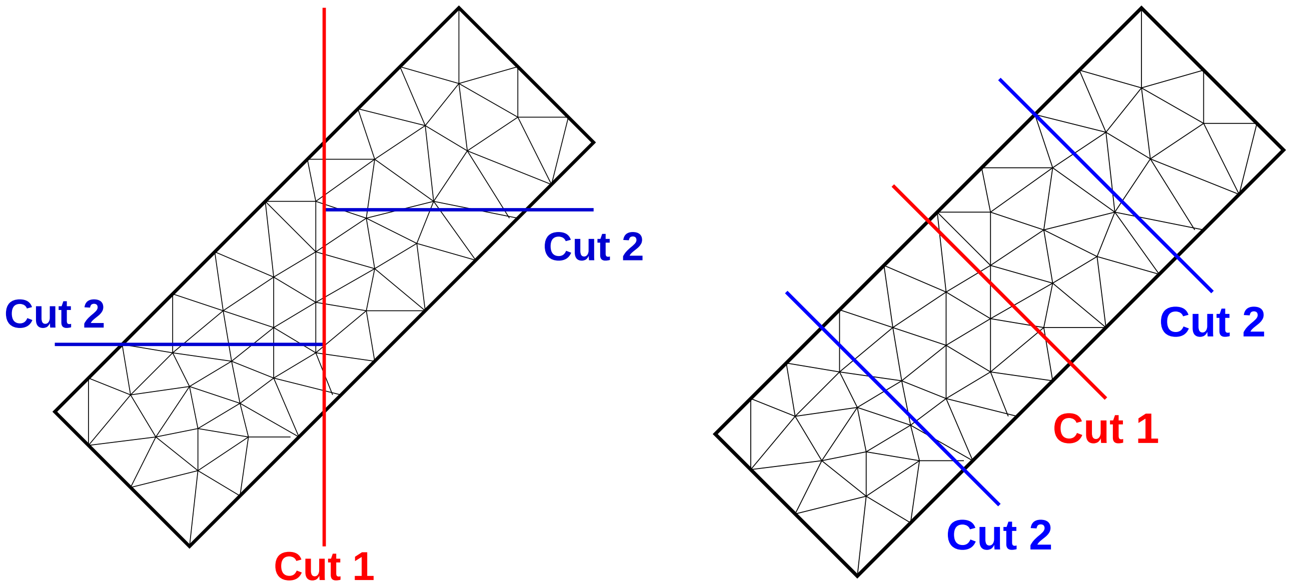
- Tend to be fast, and can achieve strict load balance
- "Unfortunate" cuts may lead to larger partition boundaries
  - cut through a highly refined region
- May be the only option when only coordinates are available
- May be especially beneficial when spatial searches are needed
  - contact problems in crash simulations

Recursive Bisection Mesh Partitioning/Load Balancing

Simple geometric methods

- Recursive methods, recursive cuts determined by

Coordinate Bisection (RCB) Inertial Bisection (RIB)

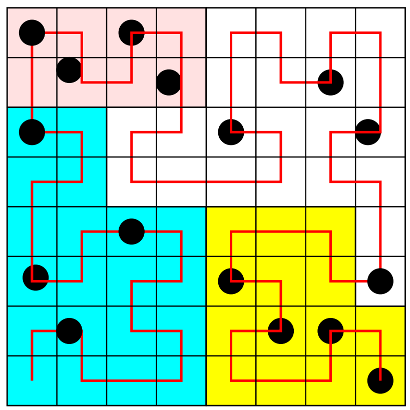


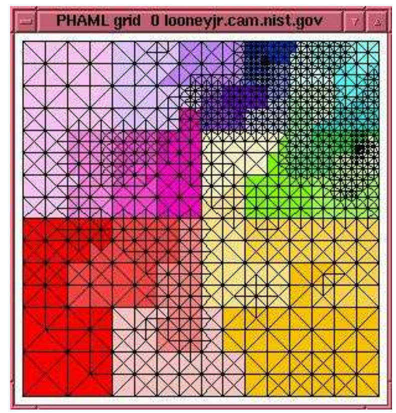
- Simple and fast
- RCB is incremental
- Partition quality may be poor
- Boundary size may be reduced by a post-processing “smoothing” step

SFC Mesh Partitioning/Load Balancing

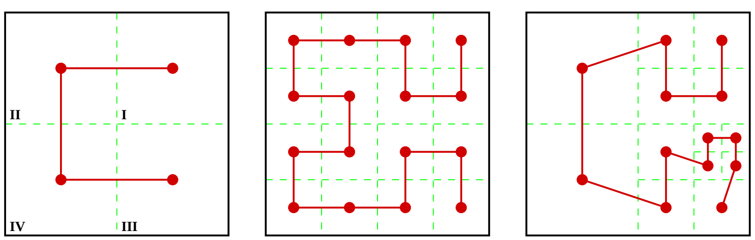
Another geometric method

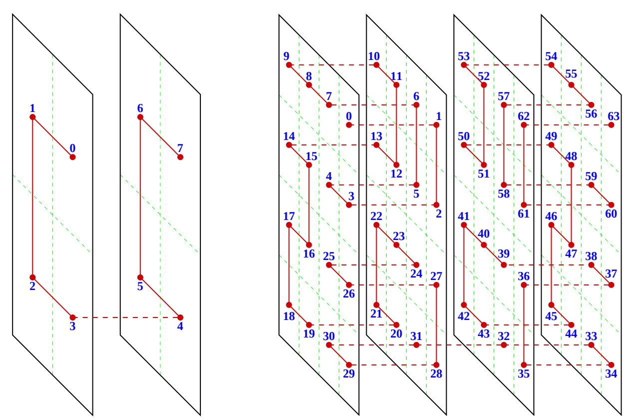
- Use the locality-preserving properties of space-filling curves (SFCs)
- Each element is assigned a coordinate along an SFC
  - a linearization of the objects in two- or three-dimensional space





- Hilbert SFC is most effective

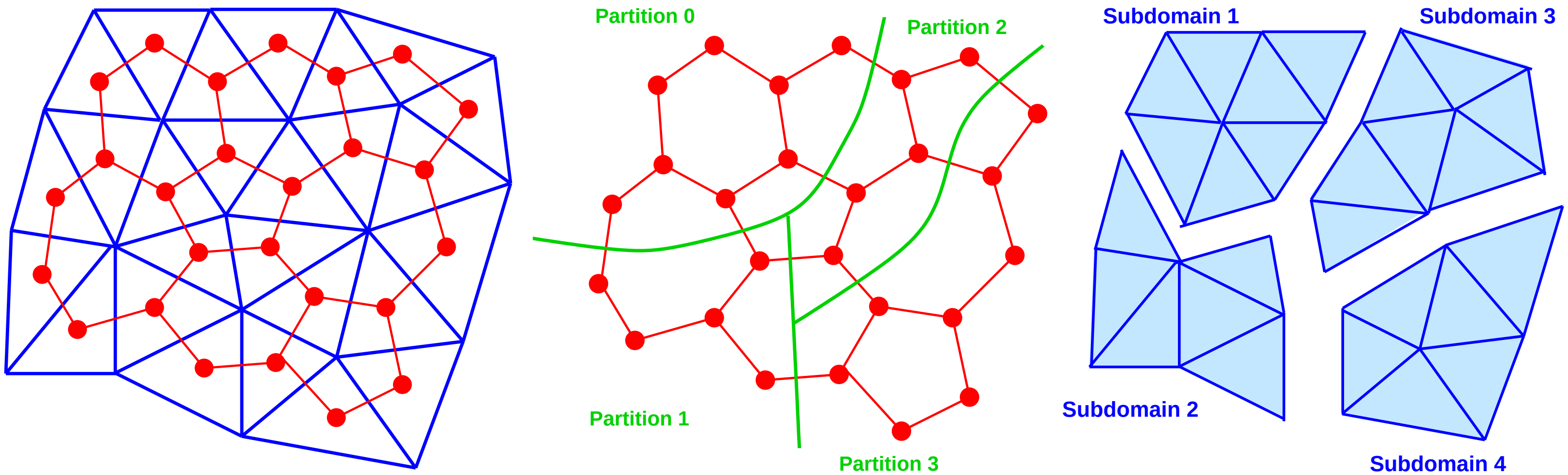




- Related methods: octree partitioning, refinement tree partitioning

Graph-Based Mesh Partitioning/Load Balancing

Use connectivity information



- Spectral methods (Chaco)
  - – prohibitively expensive and difficult to parallelize
  - – produces excellent partitions
- Multilevel partitioning (Parmetis, Jostle)
  - – much faster than spectral, but still more expensive than geometric
  - – quality of partitions approaches that of spectral methods
- May introduce some load imbalance to improve boundary sizes

Load Balancing Algorithm Implementations

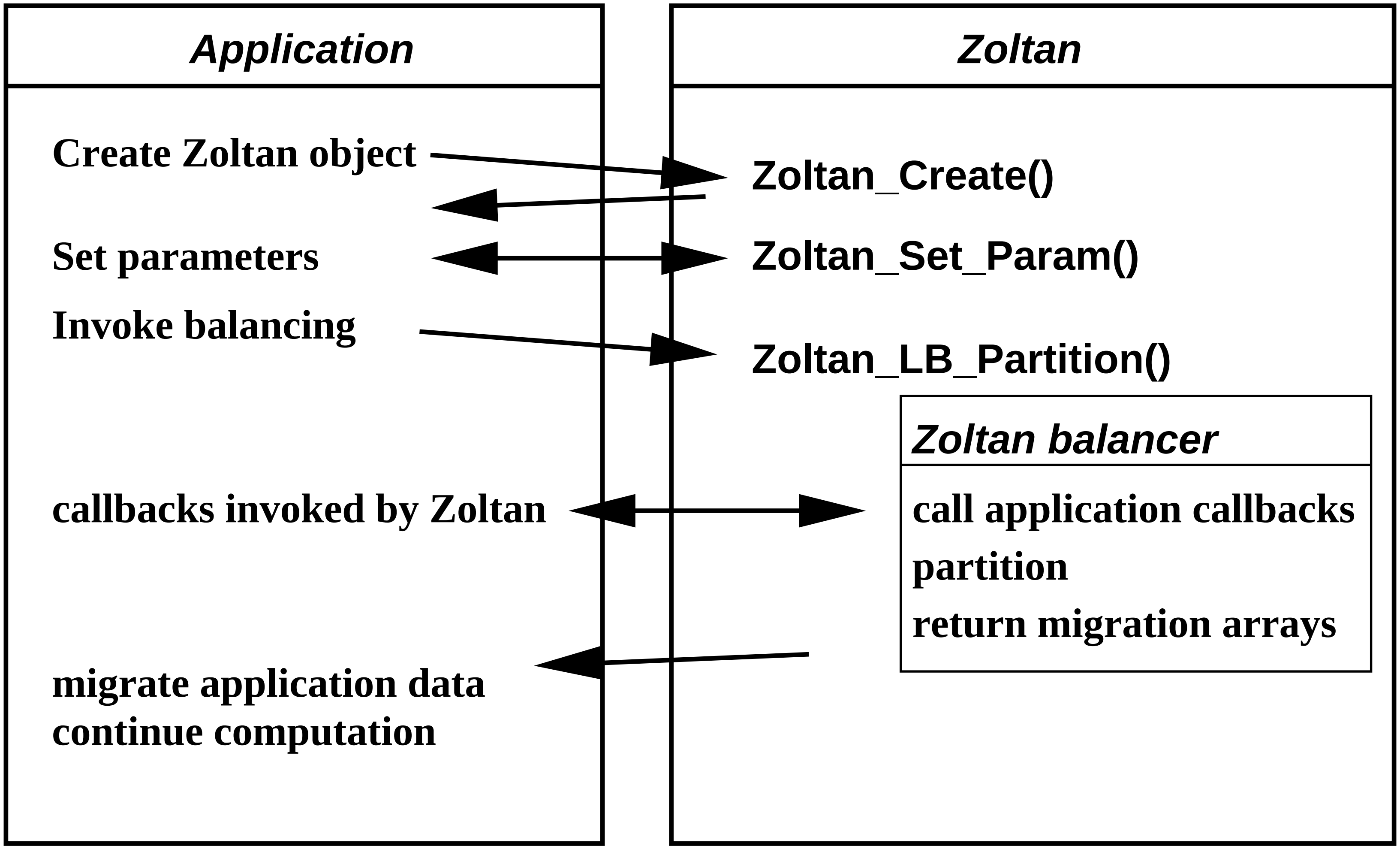
- Again, no single algorithm is best in all situations
- Some are difficult to implement
- Bad: implementation within an application or framework
  - likely usable only by a single application
  - at best, usable by a few applications that share common data structures
  - unlikely that an expert in load balancing is the developer
- Better: implementation within reusable libraries
  - load balancing experts can develop and optimize implementations
  - application programmers can make use without worrying about details
  - but...how to deal with the variety of applications and data structures?
    - ∗ require specific input and output structures
      – applications must construct them
    - ∗ data-structure neutral design
      – applications only need to provide a small set of callback functions

Zoltan Toolkit

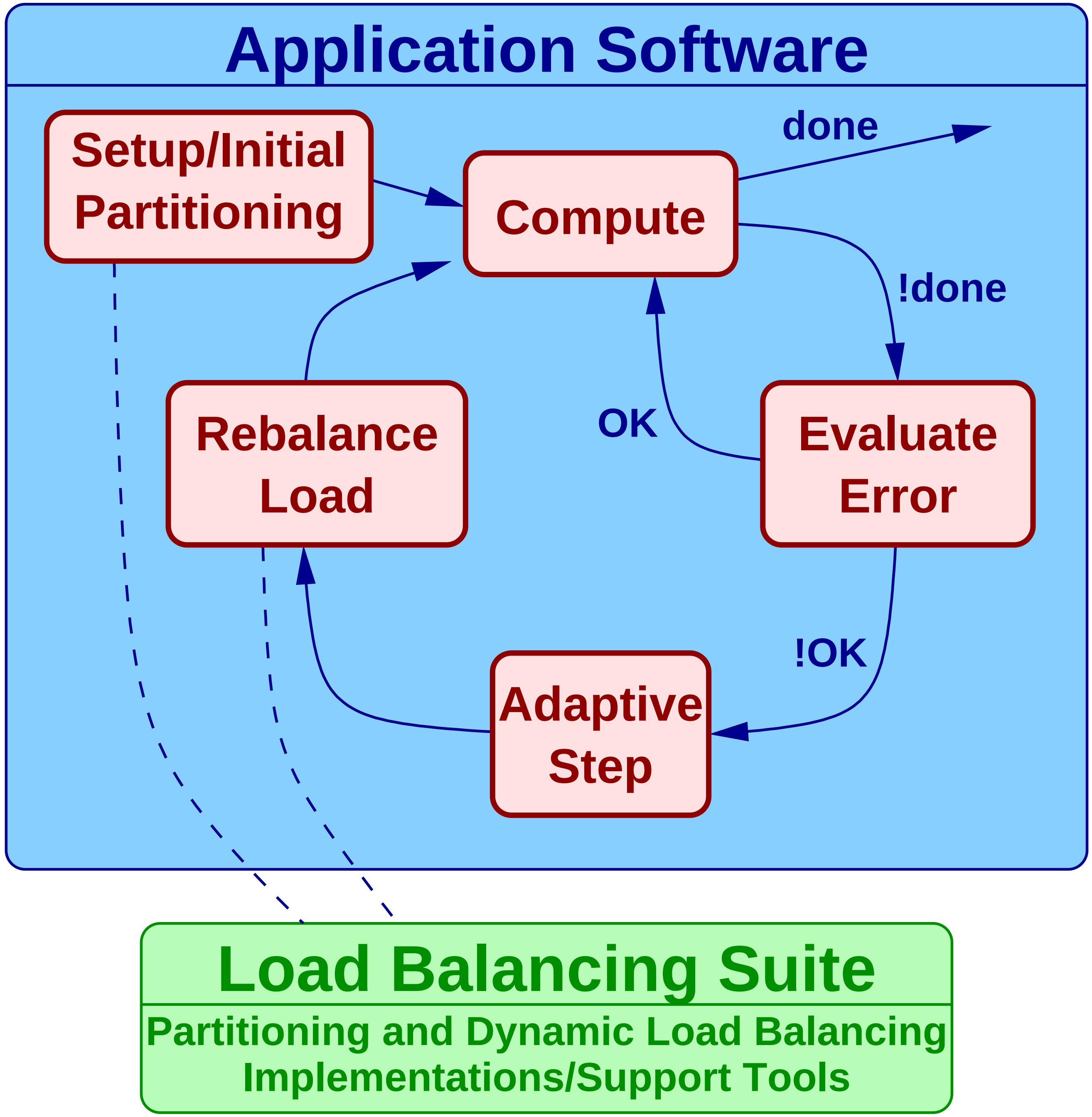
Includes suite of partitioning algorithms, developed at Sandia National Laboratories

- General interface to a variety of partitioners and load balancers
- Application programmer can avoid the details of load balancing
- Interact with application through callback functions and migration arrays
  - "data structure neutral" design
- Switch among load balancers easily; experiment to find what works best
- Provides high quality implementations of:
  - Coordinate bisection, Inertial bisection
  - Octree/SFC partitioning (with Loy, Gervasio, Campbell – RPI)
  - Hilbert SFC partitioning
  - Refinement tree balancing (Mitchell – NIST)
  - Hypergraph partitioning
- Provides easier-to-use interfaces for:
  - Metis/Parmetis (Karypis, Kumar, Schloegel – Minnesota)
  - Jostle (Walshaw – Greenwich)
- Freely available: `http://www.cs.sandia.gov/Zoltan/`

Zoltan Toolkit Interaction with Applications

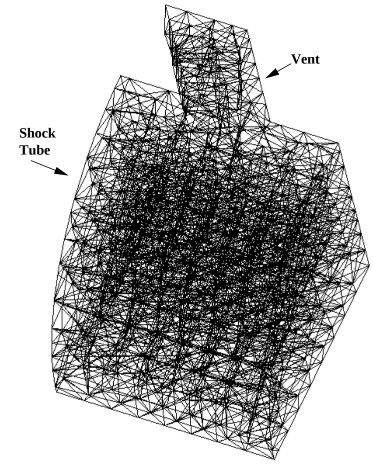


Typical Computation Flow

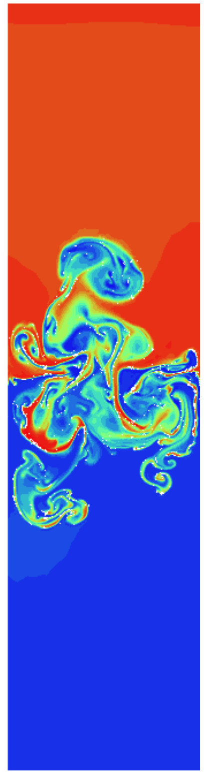


Example Parallel Adaptive Software

We wish to run several applications.



- Rensselaer's "LOCO"
  - parallel adaptive discontinuous Galerkin solution of compressible Euler equations in C.
  - using Parallel Mesh Database
  - "perforated shock tube" problem
- Rensselaer's "DG"
  - also discontinuous Galerkin methods, but in C++
  - using Algorithm-Oriented Mesh Database
  - Rayleigh-Taylor flow instabilities and others



- Mitchell's PHAML
  - Fortran 90, adaptive solutions of various PDEs
- Simmetrix, Inc. MeshSim-based applications
- Real interest for parallel computing is in 3D transient problems

Target Computational Environments

- FreeBSD Lab, Williams CS: 12 dual hyperthreaded 2.4 GHz Intel Xeon processor systems
- *Bullpen Cluster*, Williams CS: 13 node Sun cluster, total of 4 300 MHz and 21 450 MHz UltraSparc II processors
- *Dhanni Cluster*, Williams CS: 14 nodes, 8 with 2 hyperthreaded 2.8 GHz Intel Xeon processors, 2 with 2 dual-core hyperthreaded 2.8 GHz Intel Xeon processors, 1 with dual hyperthreaded 2.4 GHz Intel Xeon processor (compile node), and 3 with 1 1GHz Intel Pentium III
- *Medusa Cluster*, RPI: 32 dual 2.0GHz Intel Xeon processor systems
- ASCI-class supercomputers: large clusters of SMPs
- Grid computers – “clusters of clusters” or “clusters of supercomputers”

This is just a small sample of the wide variety of systems in use.

- long-term “resource-aware computation” goal: software that can run efficiently on any of them
- work described here is one step toward this goal, current focus on systems found at places like Williams

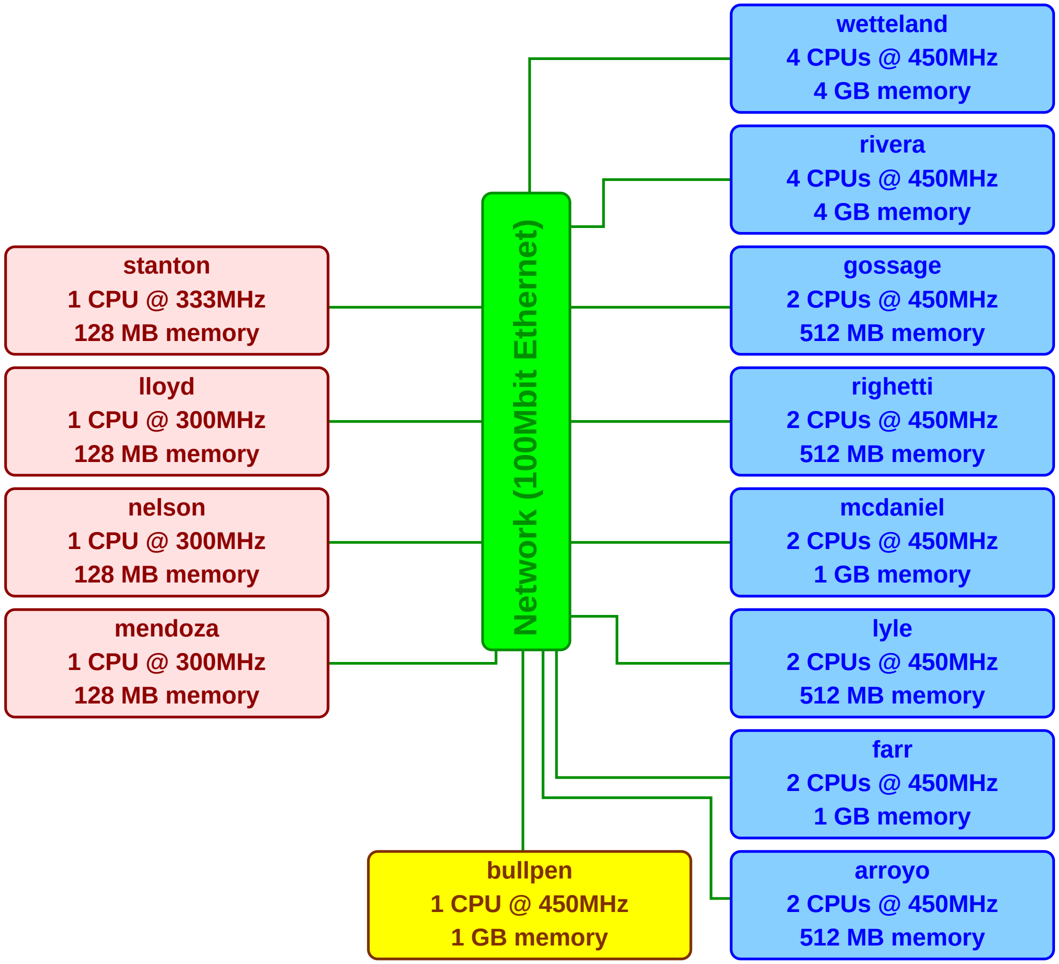
Resource-Aware Computing Motivations

- Heterogeneous processor speeds
  - seem straightforward to deal with
  - does it matter? computation proceeds only as fast as the *slowest* process
- Distributed *vs*. shared memory
  - some algorithms may be a more appropriate choice than others
- Non-dedicated computational resources
  - can be highly dynamic, transient
  - will the situation change by the time we can react?
- Heterogeneous or non-dedicated networks
- Hierarchical network structures
  - message cost depends on the path it must take
- Relative speeds of processors/memory/networks
  - important even when targeting different homogeneous clusters

What Can Be Adjusted?

- Choice of solution methods and algorithms
  - different approaches for multithreading *vs*. distributed memory
- Parallelization paradigm
  - threads *vs*. message passing *vs*. actor/theater model *vs*. hybrid approaches
  - “bag-of-tasks” master/slave *vs*. domain decomposition
- Ordering of computation and/or communication
- Replication of data or computation
- Communication patterns (*e.g.*, message packing)
- Optimal number of processors, processes, or threads
  - not necessarily one process/thread per processor
- Our focus: partitioning and dynamic load balancing
  - tradeoffs for imbalance *vs*. communication volume
  - variable-sized partitions
  - avoid communication across slowest interfaces

Bullpen Cluster at Williams College



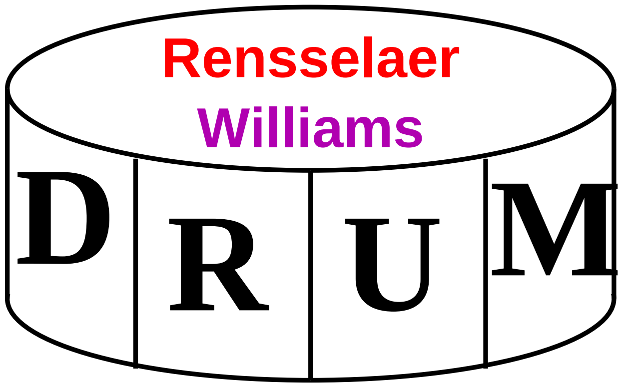
All nodes contain (aging) Sun UltraSparc II processors

`http://bullpen.cs.williams.edu/`

Resource-Aware Load Balancing

- Goal: account for environment characteristics in load balancing
- Idea: build a model of the computing environment and use it to guide load balancing
  - represent heterogeneity and hierarchy
    - processor heterogeneity, SMP
    - network capabilities, load, hierarchy
  - static capability and dynamic monitoring feedback
- Use existing load balancing procedures to produce, as appropriate
  - variable size partitions
  - “hierarchical” partitions
- Longer-term: tailor other parts of the computation to the environment
- Alternate approach: process-level or system-level load balancing

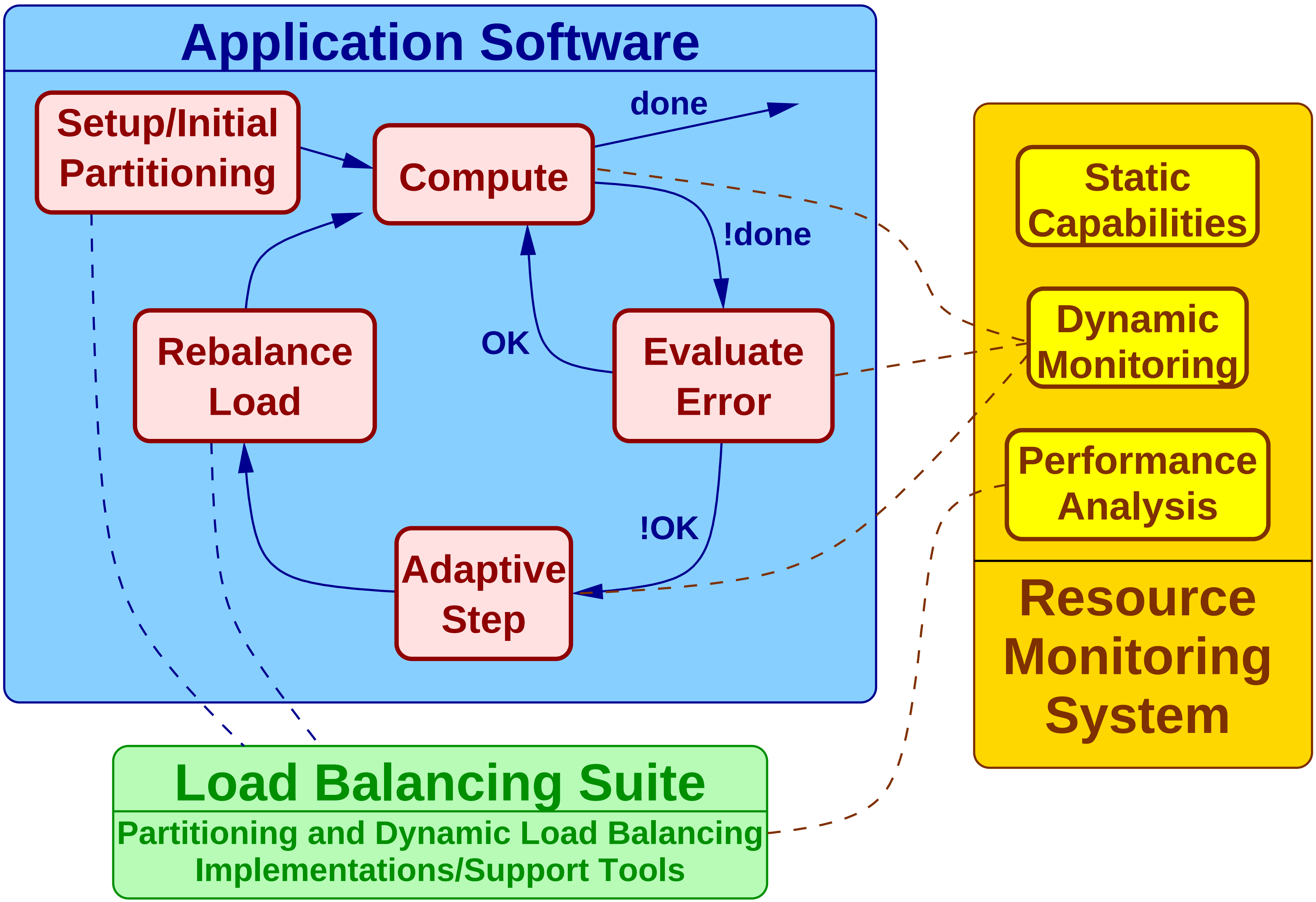
DRUM: Dynamic Resource Utilization Model



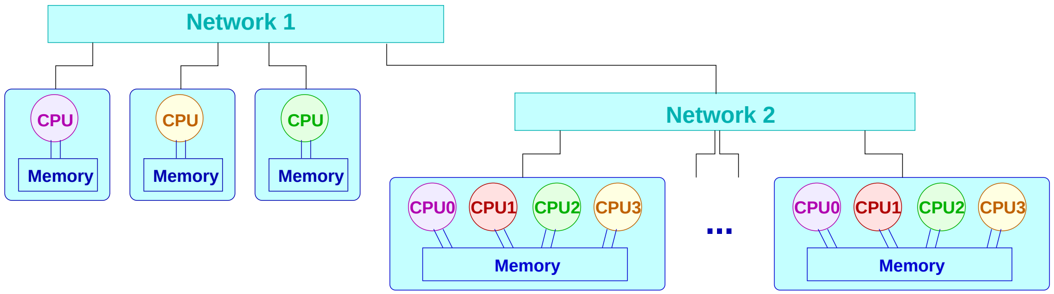
- Run-time model encapsulates the details of the execution environment
- Supports dynamic load balancing for environments with
  - heterogeneous processing capabilities
  - heterogeneous network speeds
  - hierarchical network topology
  - non-dedicated resources
- Not dependent on any specific application, data structure, or partitioner

`http://www.cs.williams.edu/drum/`

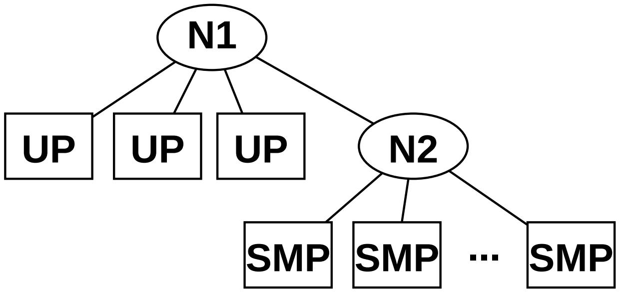
Computation Flow with DRUM monitoring



DRUM: Dynamic Resource Utilization Model



Computing Environment



Machine Model

- Tree structure based on network hierarchy
- Computation nodes, assigned "processing power"
  - UP – uniprocessor node
  - SMP – symmetric multiprocessing node
- Communication nodes
  - network characteristics (bandwidth, latency)
  - assigned a processing power as a function of children

DRUM: Dynamic Resource Utilization Model

- Static capabilities
  - gathered by benchmarks or specified manually *once* per system
  - processor speeds, network capabilities and topology
- Dynamic performance monitoring
  - gathered by "agent" threads managed through a simple API
  - communication interface (NIC) monitors
    - monitor incoming and outgoing packets and/or available bandwidth
  - CPU/Memory monitors
    - monitors CPU and memory usage and availability
- Combine static capability information and dynamic monitoring feedback
- Straightforward to use powers to create weighted partitions with existing procedures
- Optional Zoltan interface allows use by applications with no modifications
- More details and computational results in recent papers:
  Applied Numerical Mathematics, 52(2-3), pp. 133-152, 2005
  Computing in Science & Engineering, 7(2), pp. 40-50, 2005

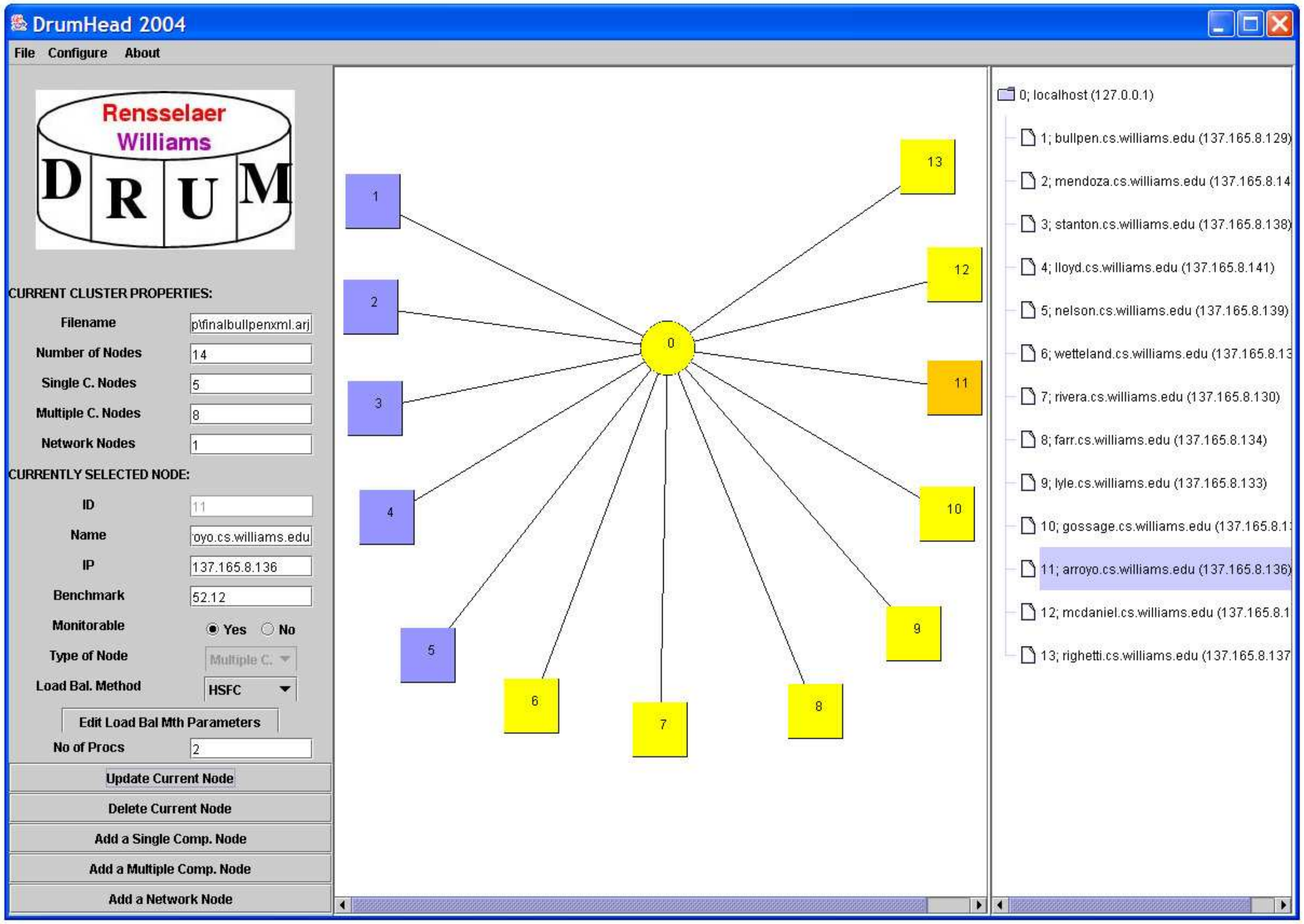
DRUM Setup and Configuration

- DRUM needs some information about the environment to construct its model
- Some may be detected at run time, some must be specified
- DRUM reads a “cluster description file” in XML format

```
<machinemodel>
<node type="NETWORK" nodenum="0" name="" IP="" isMonitorable="false"
parent="-1" imgx="361.0" imgy="52.0">
<lbmethod lbm="HSFC" KEEP_CUTS="1"></lbmethod></node>
<node type="SINGLE_COMPUTING" nodenum="2" name="mendoza.cs.williams.edu"
IP="137.165.8.140"  isMonitorable="true" parent="0"
benchmark="52.43" imgx="50.0" imgy="138.0"></node>
<node type="MULTIPLE_COMPUTING" nodenum="3" name="rivera.cs.williams.edu"
IP="137.165.8.130" isMonitorable="false" parent="0"
imgx="74.64 "imgy="263.0" benchmark="82.55" numprocs="4">
<lbmethod lbm="HSFC" KEEP_CUTS="1">
</lbmethod></node>
...
</machinemodel>
```

- XML file needs only be updated if the cluster changes
- Arjun Sharma project Summer ’04: improve generation of these files

DRUMhead: Model Creation Tool



- Build topology description
- Run benchmark suite to determine static capabilities
- Specify hierarchical balancing parameters
- Write XML file description

DRUM Monitoring Capabilities

- DRUM needs to gather performance statistics at run time
- Some available through kernel statistics
  - specific to Solaris and Linux
- Some available through Simple Network Management Protocol (SNMP)
  - not all environments make SNMP information available
- Similar information is available through the Network Weather Service (NWS)
  - DRUM can use NWS information (Laura Effinger-Dean, Summer ’04)
- DRUM’s modular design allows other monitoring tools to be used in the future

DRUM: Dynamic Resource Utilization Model

- Straightforward to give less work to slower or busy processors
- Idea: also give less work to processes that communicate over slow or busy network interfaces
- Compute the "power" for a process as the weighted sum

$$power = w^{comm}c + w^{cpu}p, \quad w^{comm} + w^{cpu} = 1$$

  of processing power p and communication power c

  - how to choose p and c?
  - how to choose weights?
  - can we do something better than a weighted sum?
- Processing power based on static benchmark plus monitored CPU usage and idle times
- Communication power based on interface packet counts or available bandwidth measures
- Communication and processing weights currently set manually

DRUM: Dynamic Resource Utilization Model

- Processing power, $p_{n,j}$, for process j on node n based on:
  - the node's "MFLOPS" rating obtained from benchmarking, b_n
  - per-process CPU utilizations, $u_{n,j}$
  - usable idle time
    - monitor fraction of idle time of CPU t (of m CPUs in node), i_t
    - compute the overall idle time in node n, $\Sigma_{t=1}^{m} i_t$
    - compute total usable idle time on node n, $\min(k_n - \Sigma_{j=1}^{k_n} u_{n,j}, \Sigma_{t=1}^{m} i_t)$
  - Compute average CPU usage and idle times per process:

$$\overline{u}_n = \frac{1}{k_n} \sum_{j=1}^{k_n} u_{n,j}, \quad \bar{i}_n = \frac{1}{k_n} \min(k_n - \sum_{j=1}^{k_n} u_{n,j}, \sum_{t=1}^{m} i_t)$$

  - processing power estimated as:

$$p_{n,j} = b_n(\overline{u}_n + \bar{i}_n), \; j = 1, 2, \ldots, k_n$$

DRUM: Dynamic Resource Utilization Model

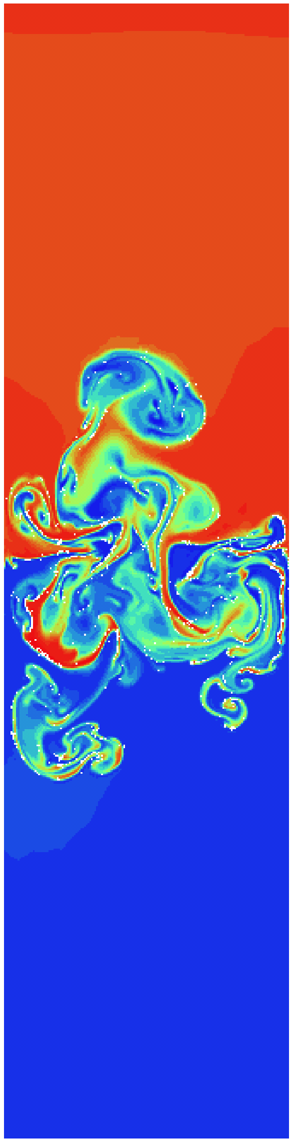
- Communication power, c_n, for interface i (of s interfaces) based on:
  - communication activity factor, $CAF_{n,i}$
  - software loopback interfaces and interfaces with $CAF = 0$ are ignored

$$c_n = \frac{1}{\frac{1}{s}\sum_{i=1}^{s} CAF_{n,i}}$$

- or, NWS-measured “available bandwidth” (Effinger-Dean, Summer ’04)
- Effectively just need a way to specify *relative* communication powers
- Considering other ways to determine a communication power and weight

DRUM: Early Computational Example

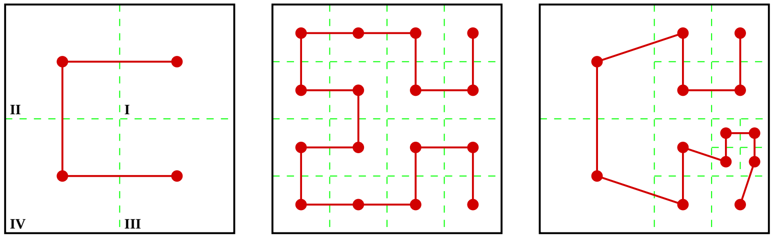
- Execution environment
  - eight processors of Williams College “Bullpen” cluster
  - five “fast” (450MHz), three “slow” (300MHz) processors
- Computation
  - 2D discontinuous Galerkin solution of R-T instability
  - 200 adaptive mesh refinement/rebalancing steps
  - maximum of 6320 elements
- DRUM resource-aware load balancing results
  - assigning 50% more work to fast nodes: theoretical 24% improvement
  - straightforward Octree/SFC partitioning: 9507 seconds
  - dynamic monitoring and partition-weighted Octree/SFC: 7351 seconds
  - 22.6% improvement, 94.1% of theoretical



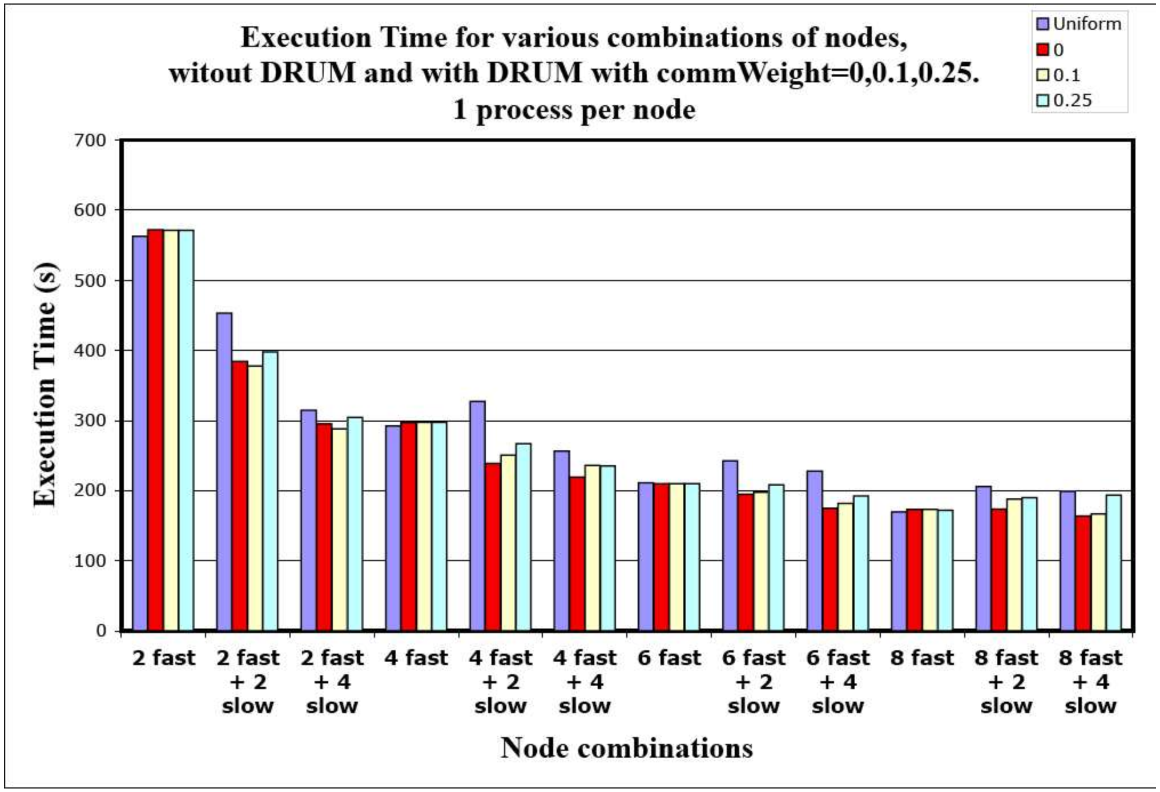
DRUM: Dynamic Resource Utilization Model

Example results on a heterogeneous cluster

- Using Mitchell's PHAML software to solve 2D Laplace equation
- Combination of "fast" (450 MHz) and "slow" (300 or 333 MHz) processors
  - 2, 4, 6, 8 fast processors
  - 0, 2, 4 slow processors
- Adaptive simulation for 17 refinement steps, resulting in 524,500 nodes
- Straightforward DRUM-aware HSFC partitioning used, vary w^{comm}

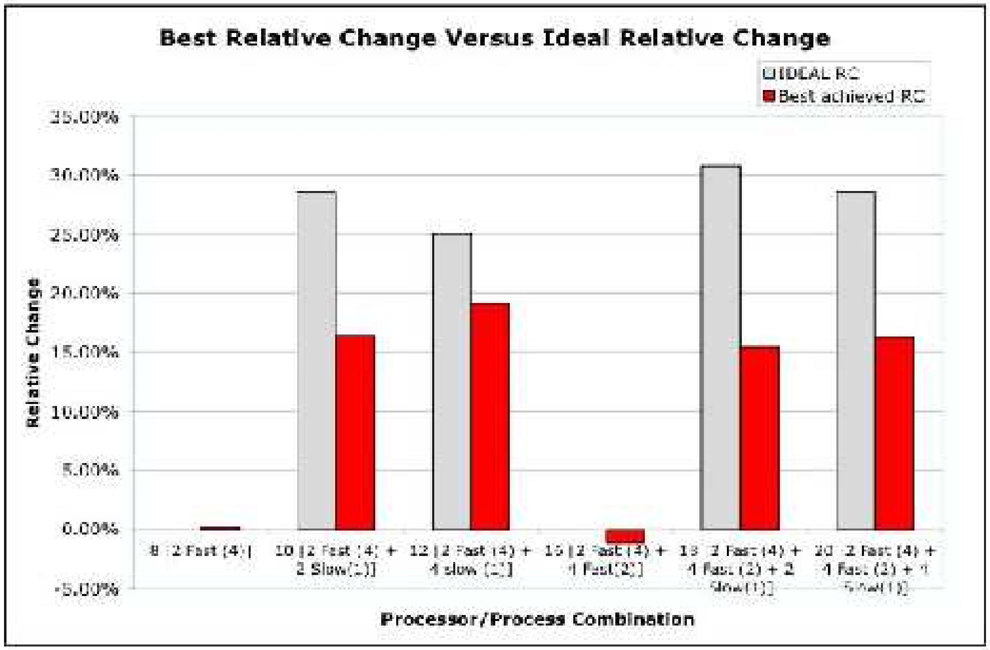


DRUM on a Heterogeneous Cluster



- Runs on a homogeneous subset of nodes indicate low overhead
- Adding 2 or 4 “slow” nodes to a run on “fast” nodes slows the computation without DRUM, but with DRUM these processors can be used effectively
- Here, accounting for communication ($w^{comm} > 0$) is usually not helpful
  - the computation is latency-dominated

DRUM on a Heterogeneous Cluster

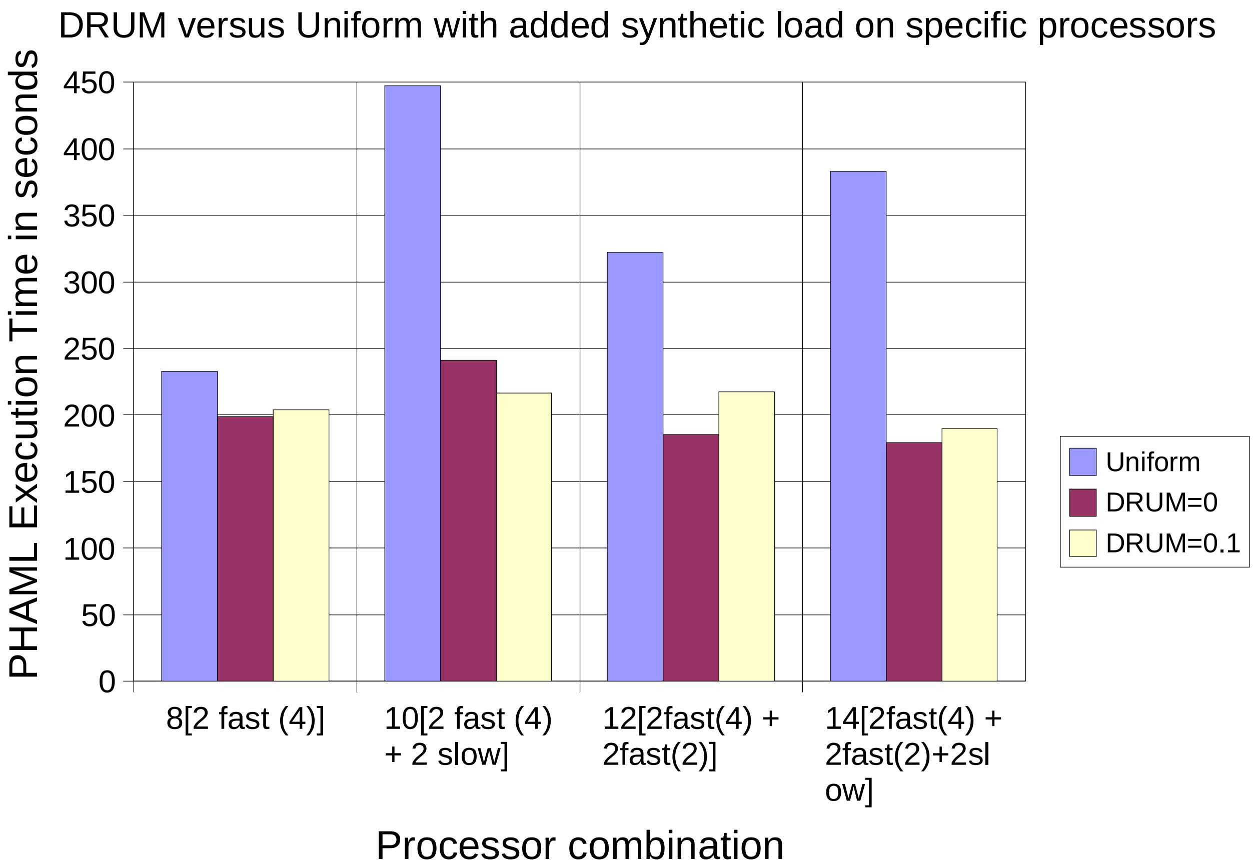


- Grey bars: “ideal” relative change – speedup we could get under perfect conditions by accounting only for processor speeds
- Red bars show best achieved on each combination of nodes across all attempted w^{comm} values
- A significant part of the ideal change is achieved

DRUM: Dynamic Resource Utilization Model

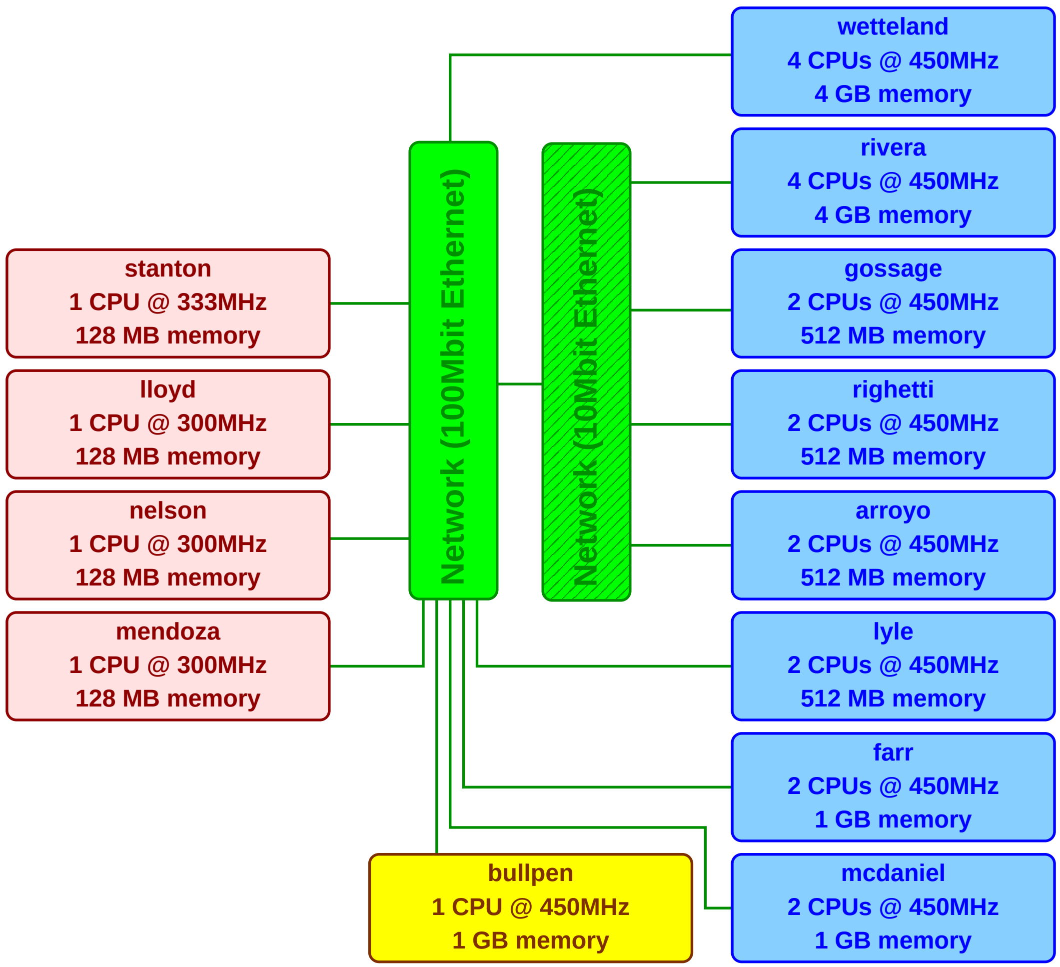
Example computation in a dynamic system

- Same cluster and application parameters, *but*
  - external computational load on nodes running two of the processes
- Demonstrates DRUM's dynamic capabilities
- DRUM dynamically adjusts the powers appropriately



Modified Bullpen Cluster

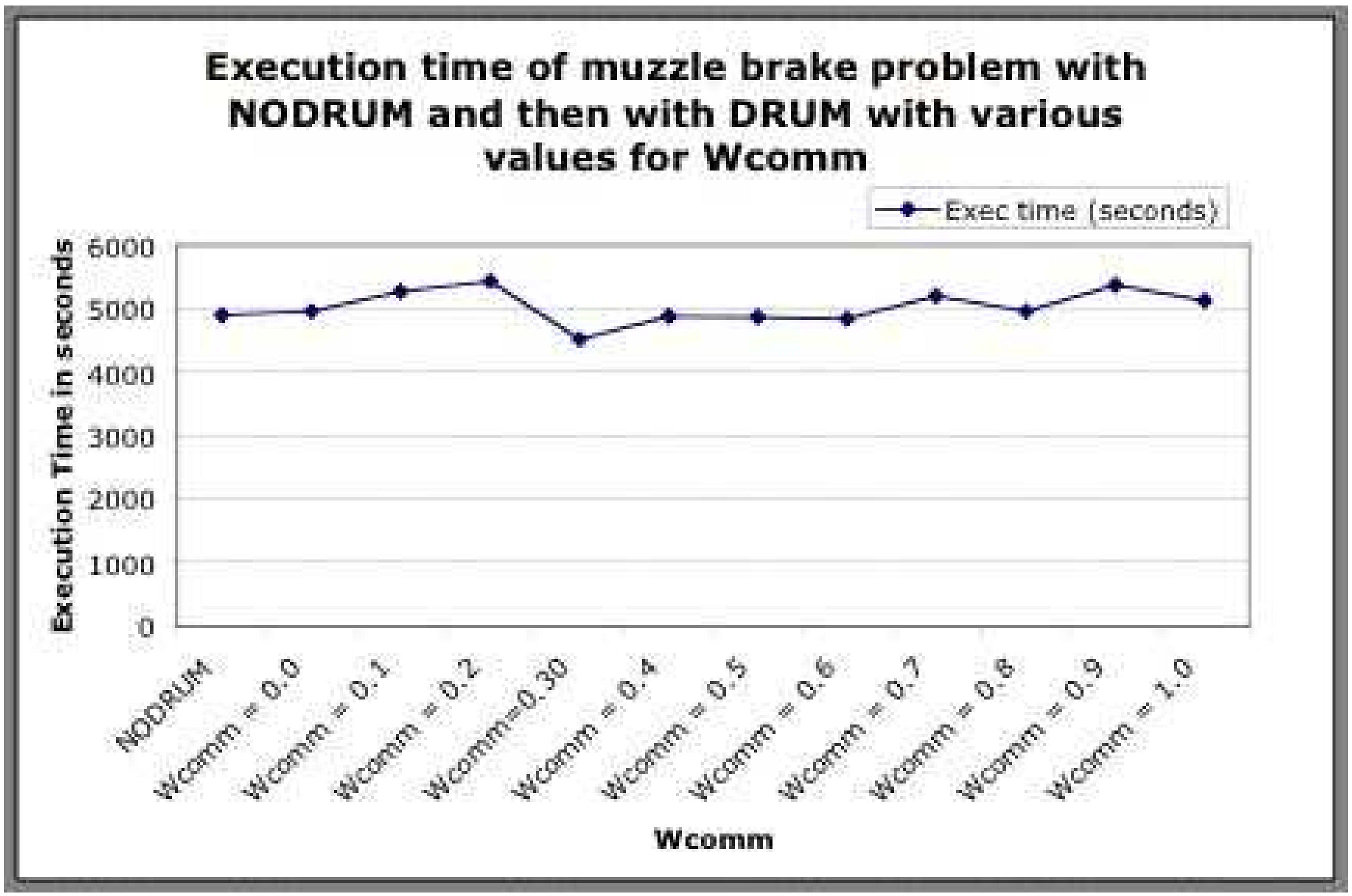
When is a non-zero communication weight beneficial?



Four nodes removed from the main network and connected to the rest of the cluster through a slower 10Mbit Ethernet hub

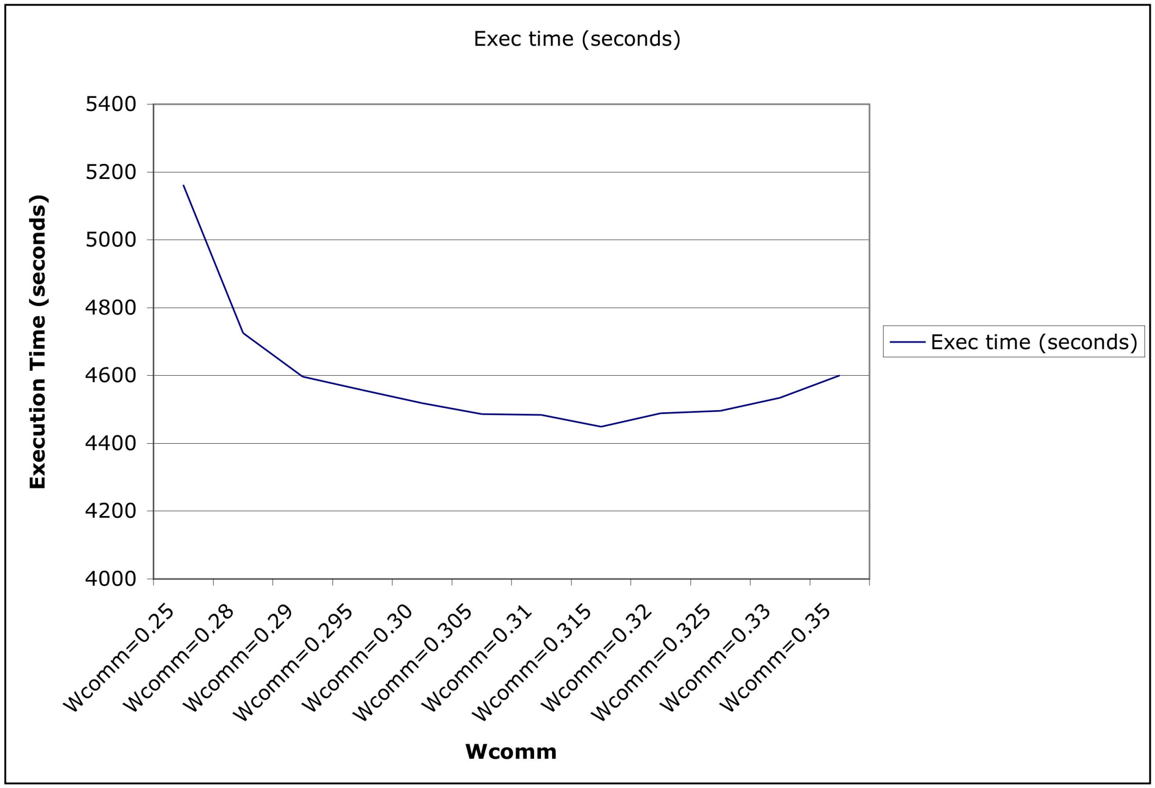
Communication Weight Study

- Three-dimensional perforated shock tube problem on modified cluster
  - 4 processes on nodes connected to main switch
  - 4 processes on nodes connected to slower switch
- Vary communication weight w^{comm}



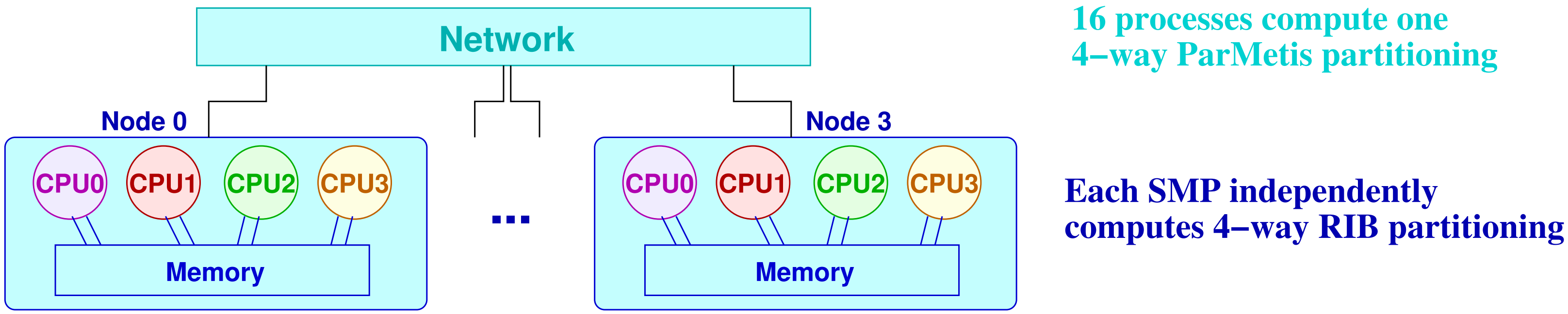
Communication Weight Study

- Best w^{comm} values in the 0.30-0.33 range
- More examples with finer adjustments to w^{comm}
- With $w^{comm} = 0.315$, processes on nodes connected to slow switch assigned power 0.09, others assigned 0.15



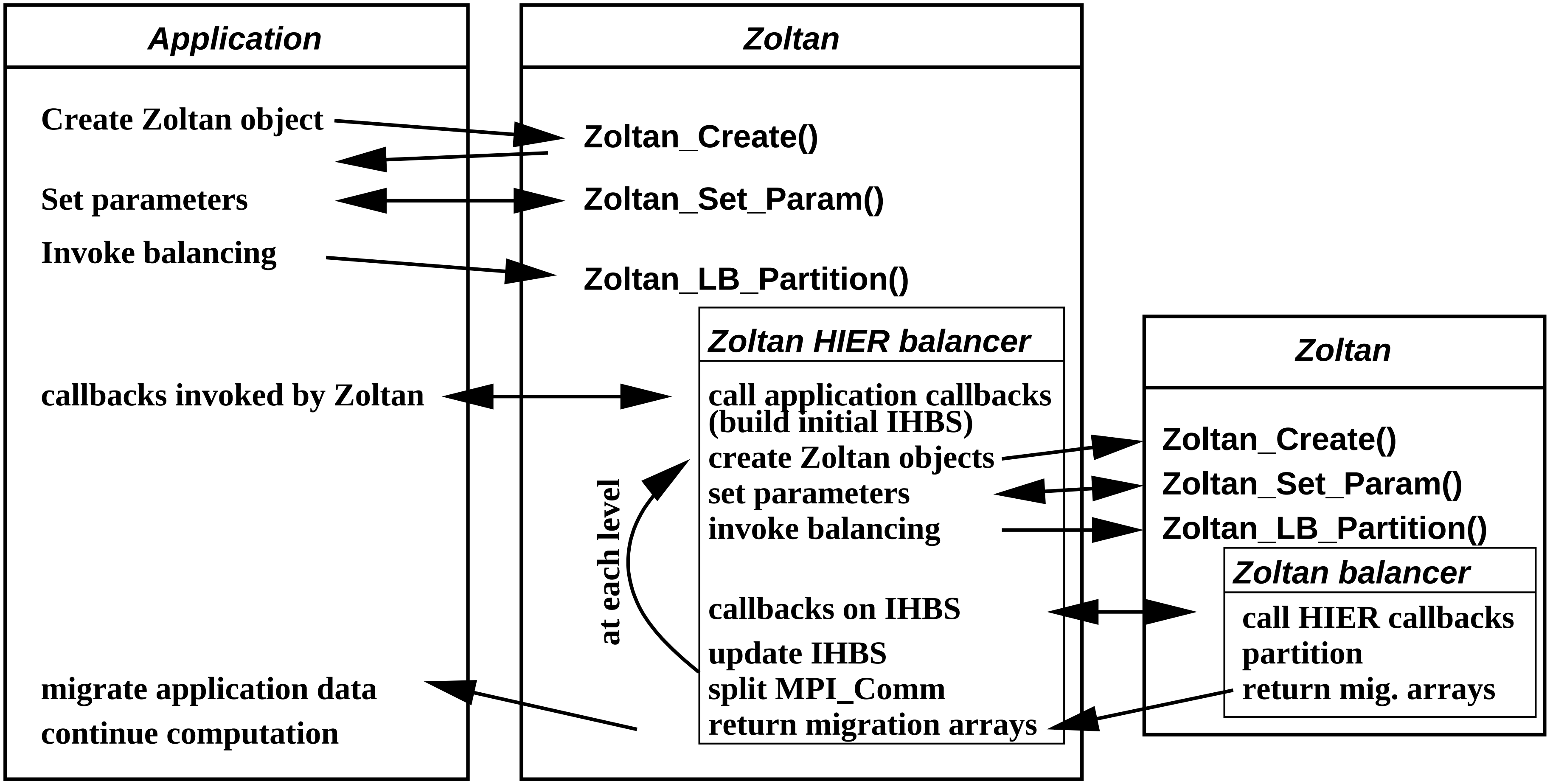
Hierarchical Partitioning and Load Balancing

- Goal:
  - use different algorithms in different parts of the execution environment
  - tailor mesh (or other) partitions for network hierarchy, SMP nodes



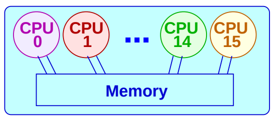
- Takes advantage of characteristics of available procedures
  - graph partitioners such as those in Parmetis
    - minimize inter-partition boundaries, but may introduce imbalance
  - geometric methods such as inertial recursive bisection
    - achieve strict balance, but may have large boundaries
  - studies with Ungar and Bennett at Williams
- Need ability to produce: variable-sized partitions, $k \neq p$ partitions

Hierarchical Load Balancing with the Zoltan Toolkit

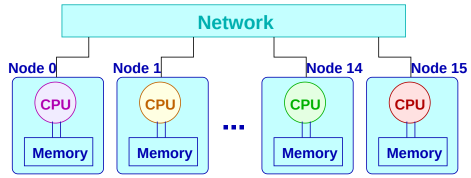


- IHBS = internal hierarchical balancing structure
  - Parmetis-style arrays, augmented to maintain internal migration
- Can do any number of levels and use any combination of procedures
- Application is not modified; existing Zoltan procedures are not modified
- In Zoltan development version, expected to include in next release

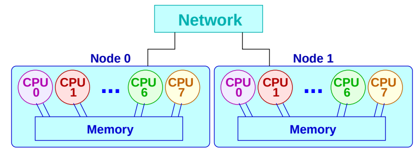
Hierarchical Load Balancing



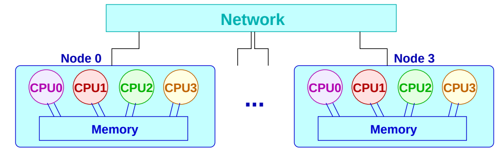
Fast Network



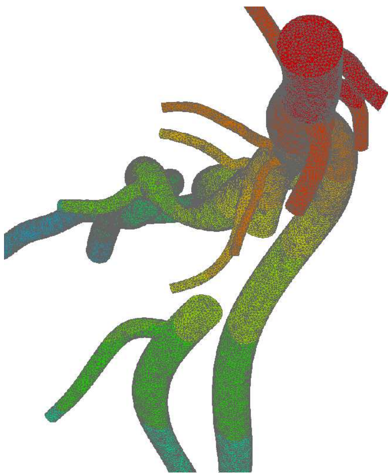
Slow Network



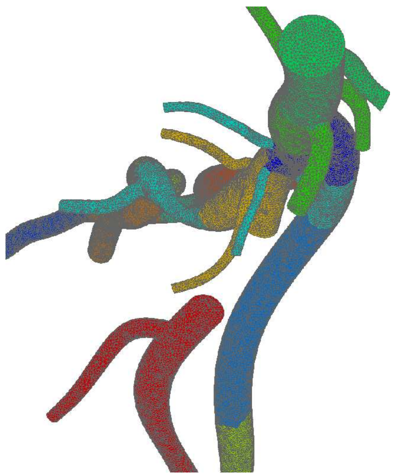
Combination



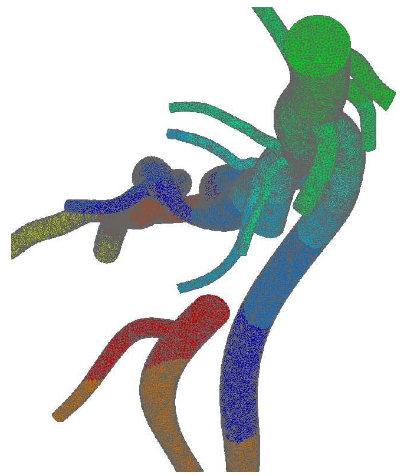
Combination



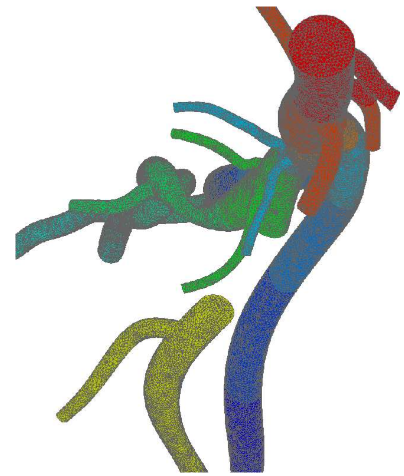
Strict Balance



Minimize Boundary



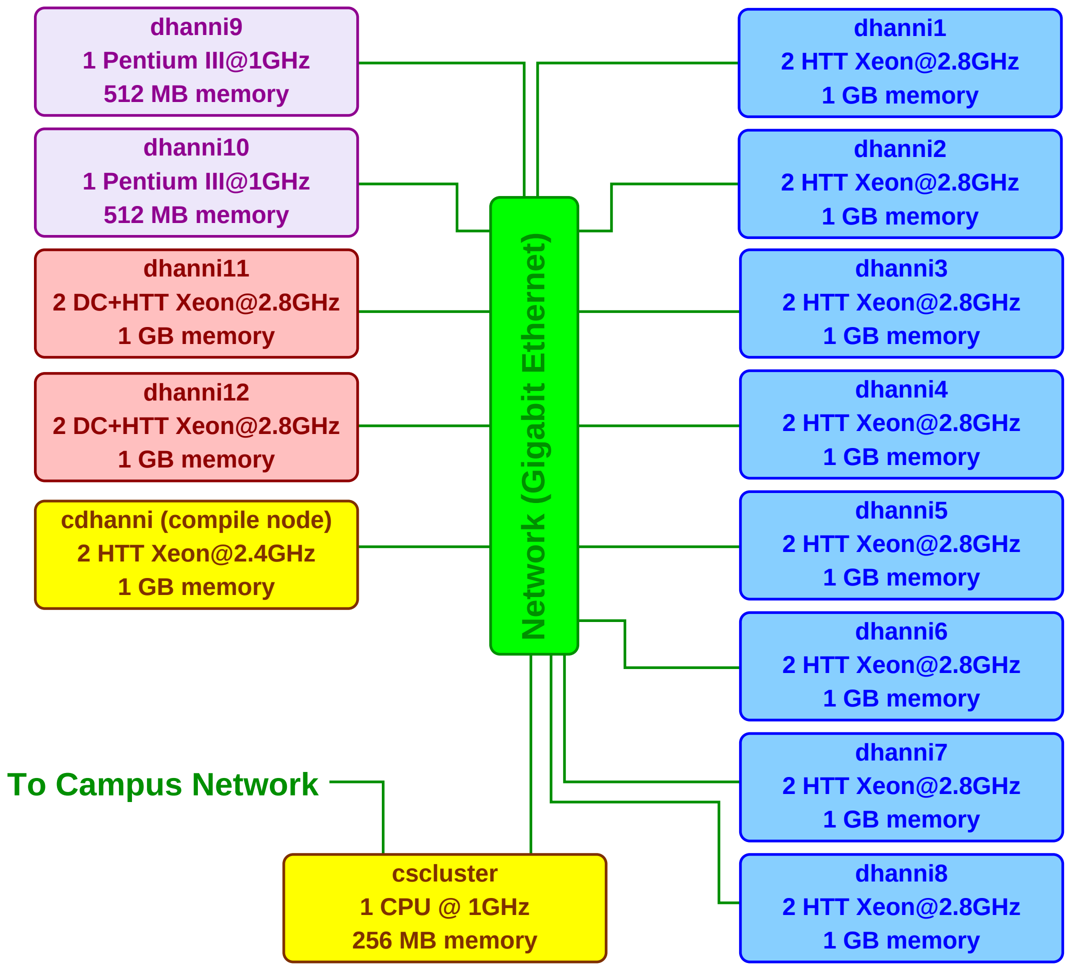
Mixed



Mixed

- 1,103,018-element mesh of human arteries, partitioned using RIB and Parmetis
- Minimize communication across slow networks, balance strictly within SMPs
  - for the 2 8-way node case, only 0.3% of faces are on “slow” boundary

Dhanni Cluster at Williams College



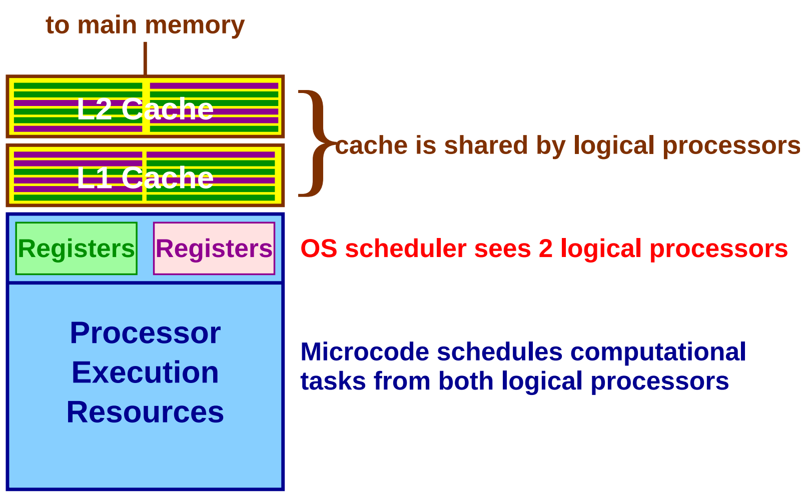
Legend: DC=dual core, HTT=hyperthreaded

Blue nodes have 4 logical processors, red nodes have 8 logical processors

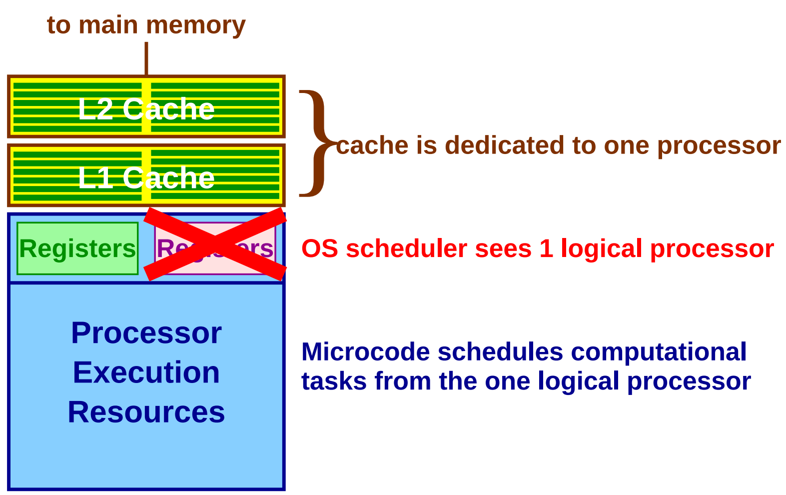
Processor Heterogeneity on Dhanni

- "Slow" vs. "Fast" nodes – same issues as Bullpen
- But... hyperthreaded and multi-core nodes
  - A hyperthreaded processor appears to the OS as 2 "logical" processors

Hyperthreading enabled



Hyperthreading disabled



Hyperthreading must be enabled/disabled in system BIOS before bootup

Processor Heterogeneity on Dhanni

- How does hyperthreading come into play?
  - Benefit of better hardware utilization must outweigh costs of cache misses
  - Initial tests with DRUM's 3D Cellular Automata example program
  - With hyperthreading disabled (in system BIOS):

    4 nodes, 1 processes per node: 182.1 seconds
    4 nodes, 2 processes per node: 91.9 seconds

  - With hyperthreading enabled:

    4 nodes, 1 processes per node: 182.4 seconds
    4 nodes, 2 processes per node: 142.5 seconds
    4 nodes, 3 processes per node: 110.1 seconds
    4 nodes, 4 processes per node: 82.1 seconds

- Expectation: effect of hyperthreading can be difficult to predict
- Expectation: DRUM's dynamic monitors will "do the right thing"

Ongoing and Future Work in DRUM

- Apply DRUM to other applications (volunteers? *terescoj@cs.williams.edu*)
- Further DRUM management of the computation – triggering load balancing
- Prepare a public release of DRUM (currently available by request)
- Apply to Grid environments
  - more heterogeneity: more need for and more benefit from DRUM
  - more hierarchy: hierarchical balancing
  - take advantage of other Grid-aware discovery and monitoring tools
- Hyperthreaded and dual-core nodes
  - test current DRUM version in these environments
  - enhance DRUM for such environments – discovery, benchmarks
  - hierarchical partitions?

Balancing at Other Levels

- Process-level load balancing
  - migrate MPI processes among computing nodes
  - developing middleware MPI/IOS system with Varela and ElMaghraoui (RPI)
    - uses MPI-2 functionality to migrate MPI processes
    - migrate processes only at “convenient” times for the application using Process Checkpoint and Migration (PCM) library
  - migration is expensive
  - support for transient environments
- System-level load balancing
  - migrate entire virtual operating systems, enabled by Xen project
  - migration is expensive, but migrating systems can continue operating until the final step
  - just-completed senior honors thesis by Travis Vachon
  - initial focus on data center environments, but is this appropriate for HPC applications?

Closing Remarks

- Accounting for heterogeneity and hierarchy can improve efficiency
- DRUM and hierarchical balancing software are not dependent on specific application software or mesh structures
  - DRUM is a standalone library (but works well with Zoltan)
  - HIER is part of the Zoltan Toolkit
- DRUM and hierarchical balancing can be transparent to applications
- Tools like DRUM more important in more heterogeneous environments
- Heterogeneity was intentional here but will arise over time as nodes are added to clusters
  - “one man’s trash is another man’s new cluster node”
  - multi-core and hyperthreaded processors
  - also think of networks of workstations, grid environments
- Focus to date on application-level load balancing, but process- and system-level balancing may be appropriate in some circumstances

Acknowledgements

First, thank you for having me here today and for your attention.

The primary funding for initial DRUM development and for the hierarchical partitioning and dynamic load balancing work has been through Sandia National Laboratories by contract 15162 and by the Computer Science Research Institute. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Portions of this work have also been supported by the following sponsors:

- Williams College Summer Science Program
- Simmetrix, Inc.

Computer systems used include:

- The "Bullpen Cluster" of Sun servers and FreeBSD lab at Williams College
- The "Dhanni Cluster" of Linux servers at Williams College
- Workstations and multiprocessors at Sandia National Laboratories and Rensselaer Polytechnic Institute
- The PASTA Laboratory at Union College (long, long ago)