An Overview of Resource-Aware Parallel Computing

(With an Emphasis on Hierarchical Partitioning and Load Balancing)

Jim Teresco

Williams College
Department of Computer Science

Department of Computer Science
Williams College
Williamstown, Massachusetts

2005 SIAM Conference on Computational Science and Engineering

Minisymposium MS60/MS78: *Resource-Aware Parallel Computation*

Co-organizers: Jim Teresco (Williams College), Jamal Faik (Rensselaer)

Yet another Powerpoint-free presentation!

Overview

- Minisymposium introduction
  - resource-aware computation: why? how? who?
  - minisymposium talks and topics
- Hierarchical partitioning and load balancing
  - motivation
  - target applications
  - a hierarchical cluster
  - parallel adaptive scientific computation
  - dynamic load balancing
  - DRUM
  - hierarchical balancing idea
  - hierarchical balancing implementation
  - early results

Resource-Aware Parallel Computing

- Optimize software for performance on a particular computer
- Everyone does some:
  - compilers: portable source code
  - optimizing compilers: architecture-specific optimizations
- Another example: special-purpose memory management
  - allocate contiguous memory for linked structures
  - try to improve cache utilization
- Parallel computing introduces more variety
  - more need and opportunity for architecture-specific optimizations
  - heterogeneity and hierarchy of resources
  - wider variety of programming paradigms and tools

Target Computational Environments

- FreeBSD Lab, Williams: 12 dual 2.4 GHz Intel Xeon processor systems
- *Momentum*, RPI: SGI Origin 2000, 12 400 MHz MIPS R10000 processors
- *Bullpen Cluster*, Williams: 13 node Sun cluster, total of 4 300 MHz and 21 450 MHz UltraSparc II processors
- *ASCI Red*, Sandia National Labs: 4600+ nodes, each with 2 Intel Pentium II Xeon processors, first TeraOp machine in 1997
- *ASCI White*, LLNL: 512 nodes, each with 16 Power3 Nighthawk-2 processors, 12 TeraOps total, was world's fastest until 2002
- *ASCI Q*, LANL: 8192 HP AlphaServer 1.25 GHz processors, 15 TeraOps
- *Big Mac*, Virginia Tech: 1100 dual 2.0 GHz Apple G5 nodes, 17.6 TeraOps
- *ASCI Purple*, First 100+ Teraflop system, coming soon
- *Squall*, Williams College: Macintosh PowerBook, 1.25 GHz Power PC G4

Less Traditional Environments

- Internet Computing: “actor/theater” model of computation allowing a large distributed computation using resources that may be shared or unreliable
- Cray (formerly Tera) MTA: “multithreaded architecture” has fully multi-threaded hardware and OS
- Earth Simulator, Yokohama Institute for Earth Sciences, Japan: 640-node NEC system, each node with 8 vector processors, total of 5,120 CPUs, peak performance 40 TeraOps
- Computational Grids, such as the NSF TeraGrid: nodes at NCSA, San Diego Supercomputing Center, Argonne National Laboratory, Caltech, and the Pittsburgh Supercomputer Center, peak projected performance 20 Tera-Ops
- Many other Grids deployed or planned

Motivations

- Heterogeneous processor speeds
  - seem straightforward to deal with
  - does it matter?
  - assumptions of homogeneous processor speeds may be well-hidden
- 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

Motivations

- Relative speeds of processors/memory/networks
  - important even when targeting different homogeneous clusters
- Heterogeneous processor architectures (*e.g.*, Sparc + x86)
- Operating system support for programming paradigms
  - multithreading
  - priority thread scheduling
  - distributed shared memory
- Availability of tools (*e.g.*, MPI, OpenMP, Java, *etc*.)
  - may choose something less than optimal to maximize portability
- Transient resource availability
- Reliability (or lack thereof) of processors, networks
  - last year at SIAM PP04, several sessions on fault tolerance
- Scalability concerns
  - what works well for 10’s of processors may not for 1000’s+

What Can Be Adjusted?

- Choice of programming language (*e.g.*, Java for smoother portability)
- Choice of solution methods and algorithms
  - some approaches are better for multithreading
  - some approaches are better for 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
- Replication of computation
- Optimal message sizes

What Can Be Adjusted?

- Communication patterns (*e.g.*, ordering of collective communication)
- Optimal number of processors, processes, or threads
  - not necessarily one process/thread per processor
- Process placement
  - resource-aware initial allocation of processes to nodes
  - dynamic process migration
- Partitioning and dynamic load balancing
  - tradeoffs for imbalance *vs*. communication volume
  - variable-sized partitions
  - avoid communication across slowest interfaces

Who Can Make Adjustments?

- Compiler developers
- Low-level tool developers
  - MPI implementations
- Other tool developers
  - partitioners and dynamic load balancers
  - optimized numerical libraries
- Middleware
  - monitoring tools
  - autonomous migration systems
  - automated selection from among a group of available algorithms
- Application programmers
  - parallel programming paradigm
  - distribution of work: strict balance *vs.* minimal communication
  - frequency of dynamic load balancing
  - memory management techniques

What Is Needed?

- Knowledge of computing environment
  - manual specification
  - benchmarking *a priori*
  - discover automatically at run time
  - monitor dynamically
- Knowledge of software performance characteristics
  - performance models
  - studies to compare performance
- Tools to use this knowledge
  - middleware or libraries to hide architecture-aware details
  - partitioners and dynamic load balancers

Minisymposium: *Resource-Aware Parallel Computation*

Today:

- Remainder of this talk: *Hierarchical Partitioning and Load Balancing*
- *Scientific Computation on Heterogeneous Clusters using DRUM*
  Jamal Faik, RPI
- *Architecture-Aware Autonomic Adaptations within the CCA*
  Manish Parashar, Rutgers, **Jaideep Ray**, Sandia National Laboratories

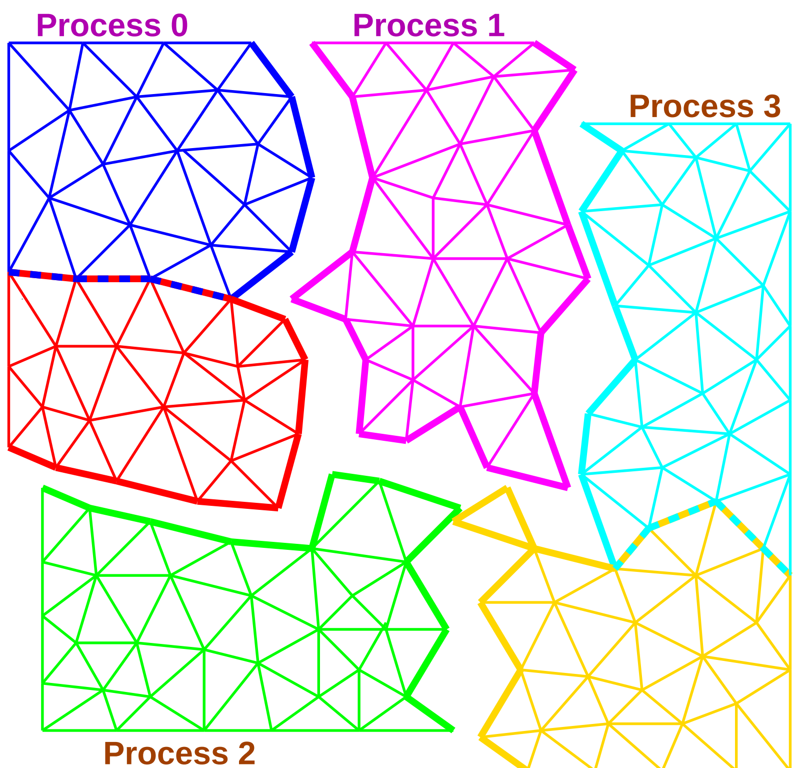
Tomorrow:

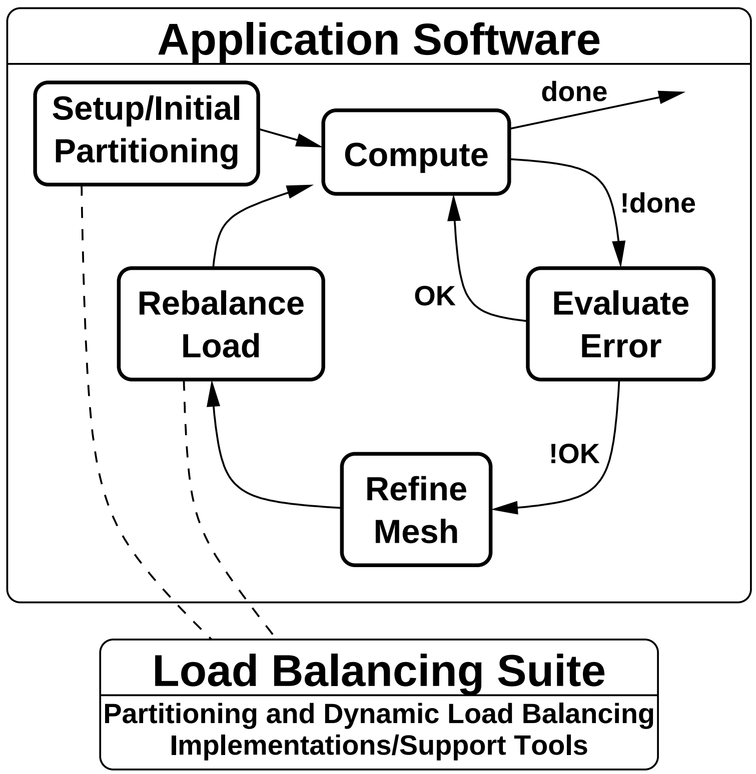
- *Automatic Deployment of MPI Applications on a Computational Grid*
  Sébastien Lacour, Argonne, and IRISA / INRIA Rennes, France and Argonne, Christian Pérez, IRISA / INRIA Rennes
- *Towards an Internet OS: Middleware for Adaptive Distributed Computing*
  Carlos Varela, RPI
- *Performance-Directed Resource Allocation*
  Seung-Hye Jang, Xingfu Wu, and Valerie Taylor, Texas A&M University

Resource-Aware Load Balancing for Scientific Computation on a Heterogeneous Cluster

Motivation: run large-scale parallel adaptive solution procedures in varying environments

- Finite element and related methods, parallelized by domain decomposition

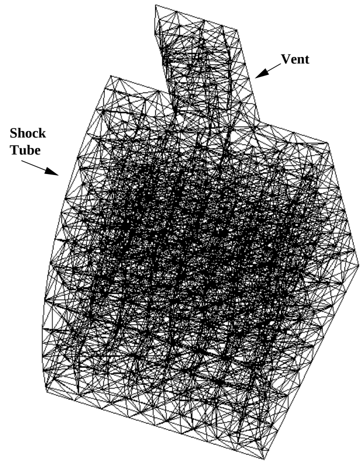


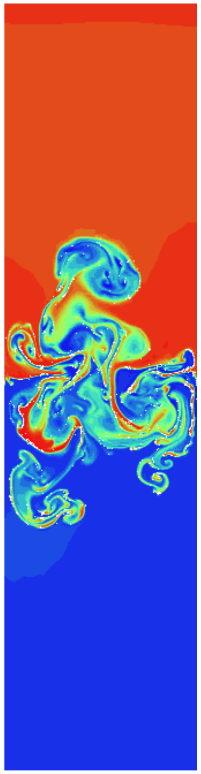


Example Parallel Adaptive Software

We wish to run several applications.

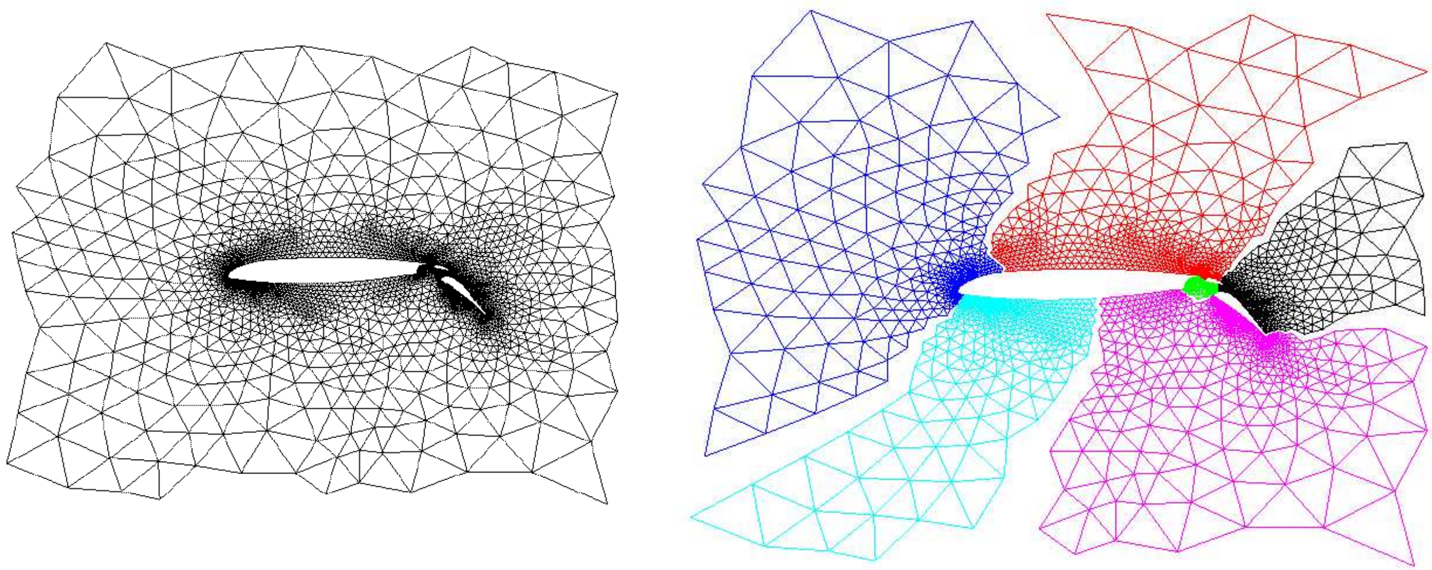
- Rensselaer's "LOCO"
  - parallel adaptive discontinuous Galerkin solution of compressible Euler equations in C.
  - "perforated shock tube" problem
- Rensselaer's "DG"
  - also discontinuous Galerkin methods, but in C++
  - using Algorithm-Oriented Mesh Database
  - Rayleigh-Taylor flow instabilities and others
- Mitchell's PHAML
  - Fortran 90, adaptive solutions of various PDEs
- Simmetrix, Inc. MeshSim-based applications
- Others





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!*

Example Parallel Adaptive Solution

Example adaptive computation: mesh-dens.mov

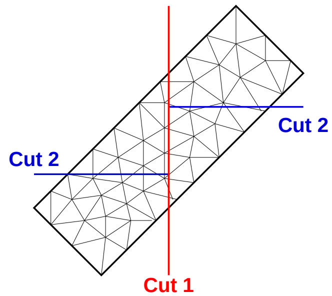
Real interest for parallel computing is in 3D transient problems.

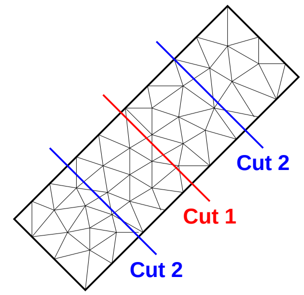
Mesh Partitioning/Load Balancing

Geometric methods, use only coordinate information

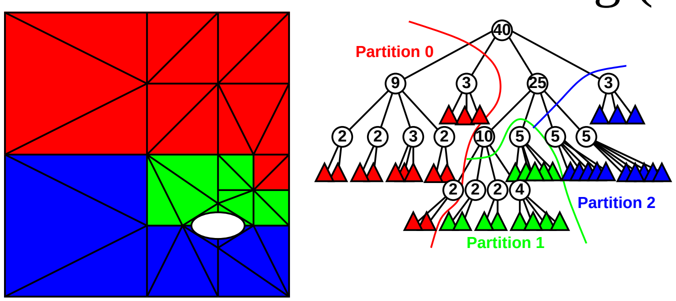
- Recursive methods, recursive cuts determined by

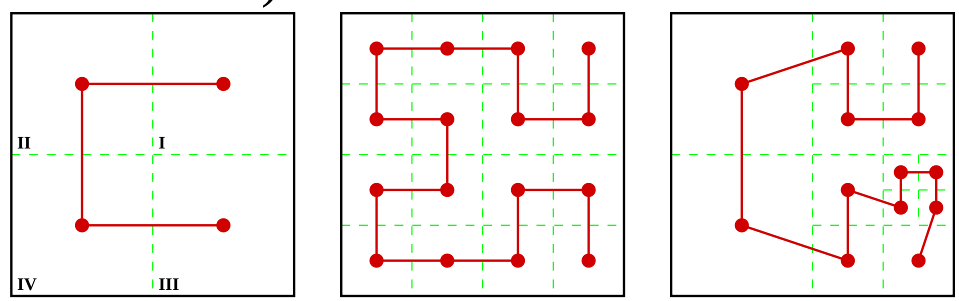
Coordinate Bisection (RCB) Inertial Bisection (RIB)





- Octree/SFC Partitioning (OCTPART/HSFC)



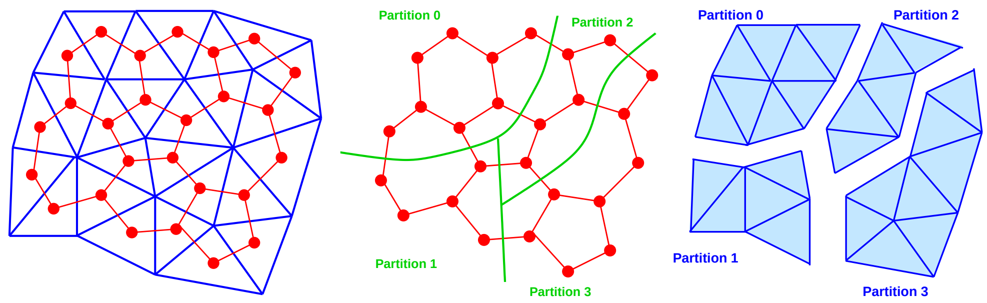


  - Morton, Grey Code, and Hilbert traversals available for OCTPART
  - Hilbert traversal for HSFC

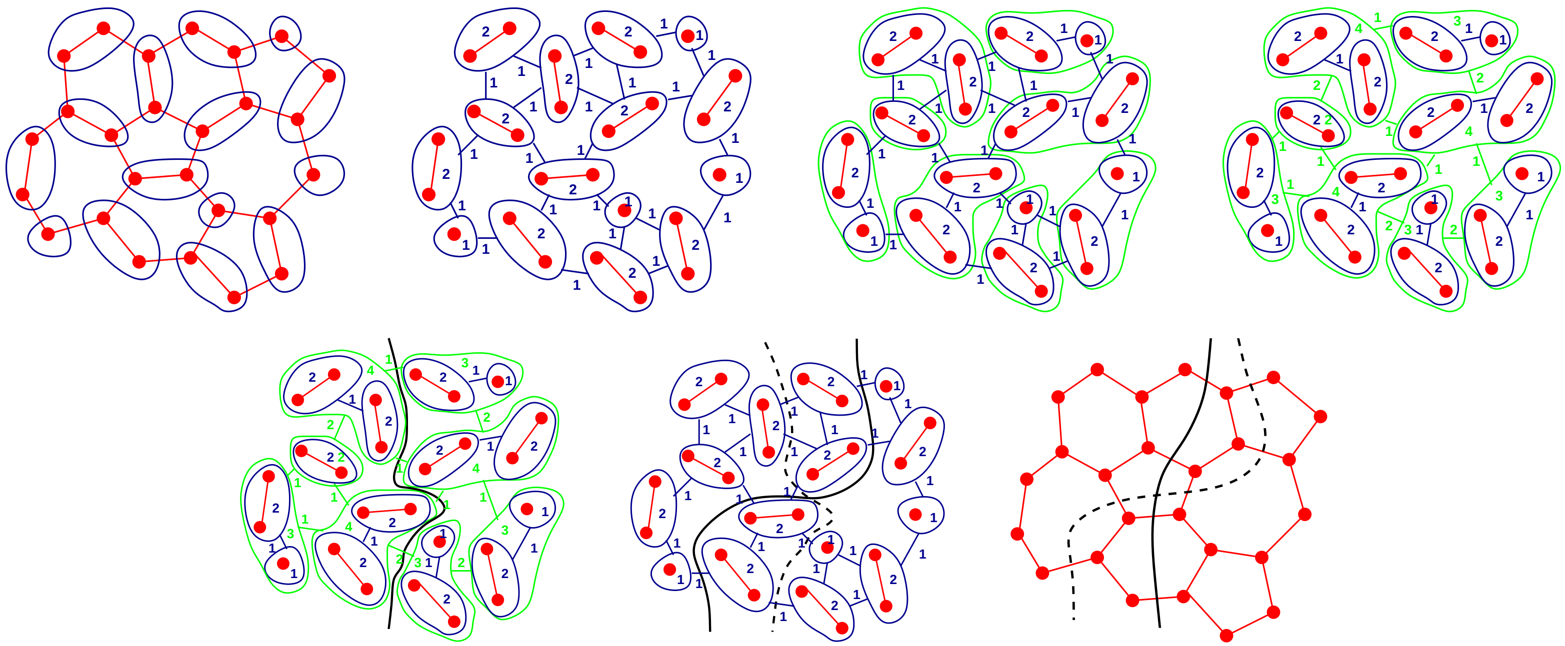
- Tend to be fast, and can achieve strict load balance
- “Unfortunate” cuts may lead to larger partition boundaries

Mesh Partitioning/Load Balancing

Graph-based methods use connectivity information



- Spectral methods (Chaco), Multilevel partitioning (ParMetis, Jostle)



- More expensive, but usually produce smallest partition boundaries
- May introduce some load imbalance to improve boundary sizes

Load Balancing Considerations

Many important factors must be considered

- Like a partitioner, a load balancer seeks
  - computational balance
  - minimization of communication
- 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
- No one approach will be best in all circumstances
  - depends on application
  - depends on parallel computing environment

Zoltan Toolkit

Includes suite of partitioning algorithms, developed at Sandia National Laboratories

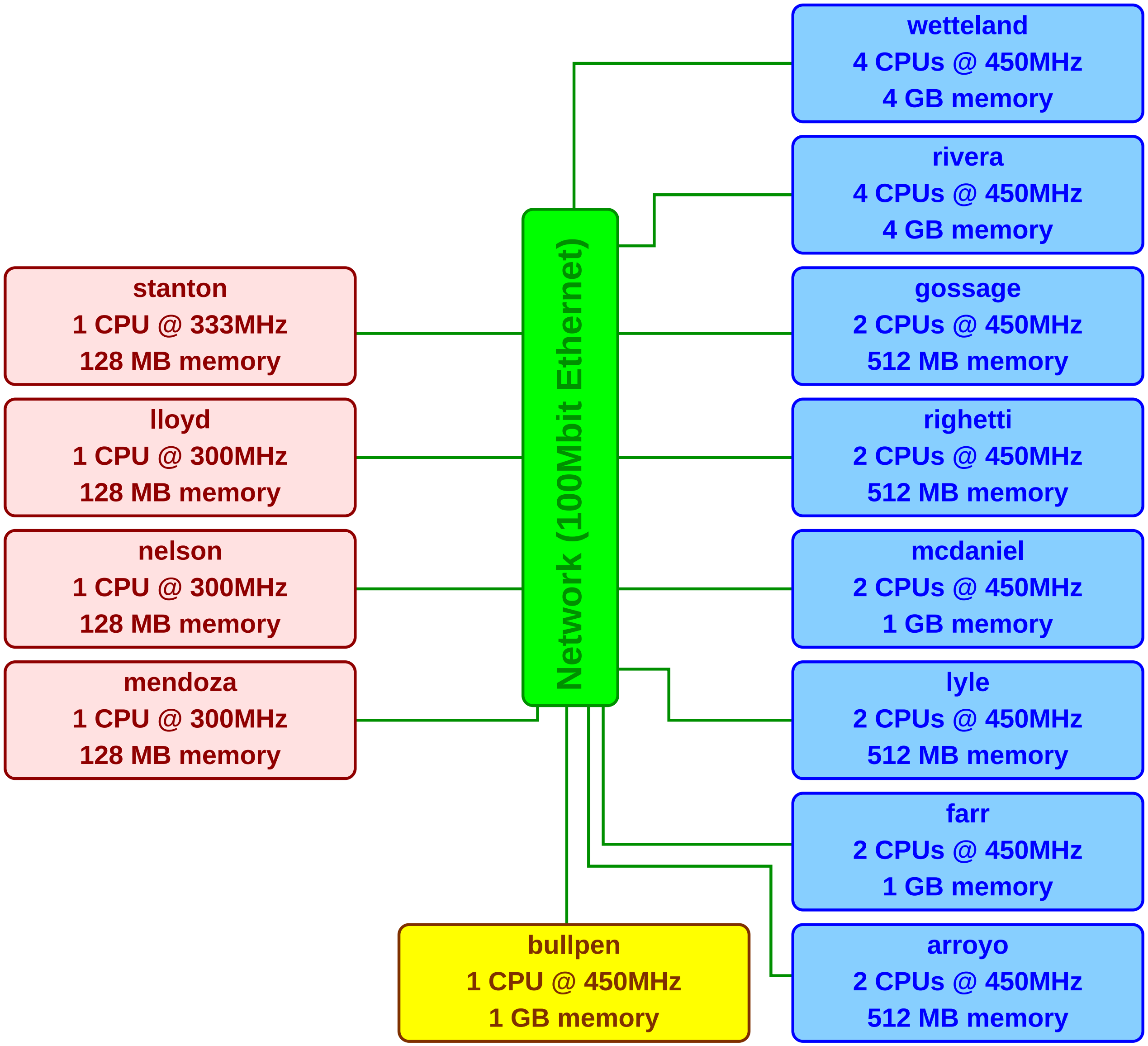
- General interface to a variety of partitioners and load balancers
- Application programmer can avoid the details of load balancing
- Interact with application through callback functions and migration arrays
  - "data structure neutral" design
- Switch among load balancers easily; experiment to find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

Bullpen Cluster

View from the door of TCL 312d.

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

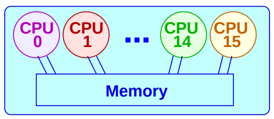
Bullpen Cluster



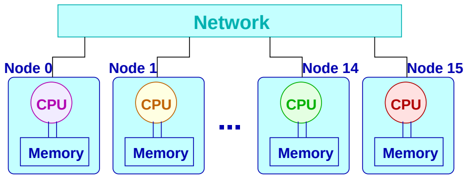
Hierarchical Partitioning Motivation

- We observe that
  - geometric partitioners are fast, give excellent balance
  - graph partitioners reduce boundary, may introduce load imbalence
  - in shared-memory environments, load balance is the key
  - in distributed environments with slow networks, reducing communication is the key
- May choose
  - a graph partitioner in the context of a slow network
  - geometric in SMPs
- What about clusters of SMPs or other hierarchical environments?
  - wish to reduce communication across slow networks
  - but maintain strict balance within a node

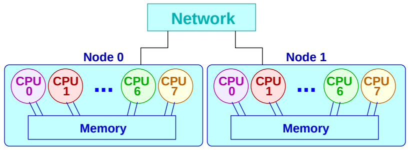
Hierarchical Load Balancing



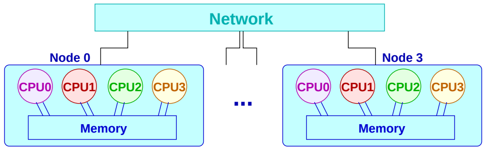
Fast Network



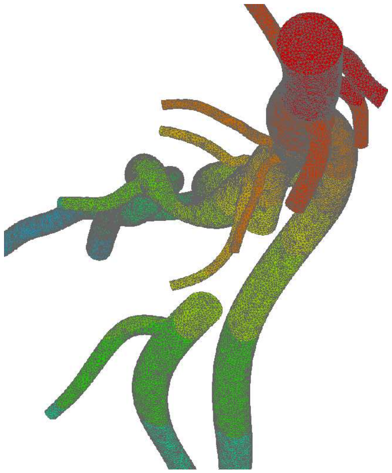
Slow Network



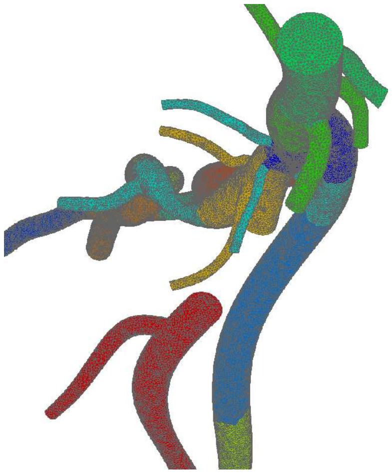
Combination



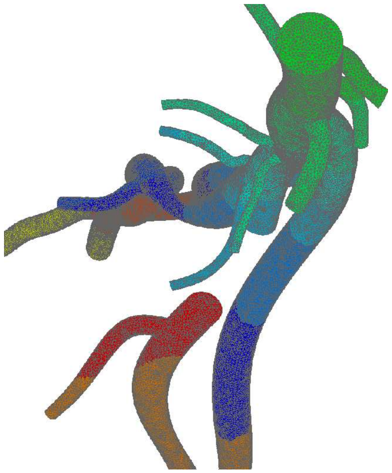
Combination



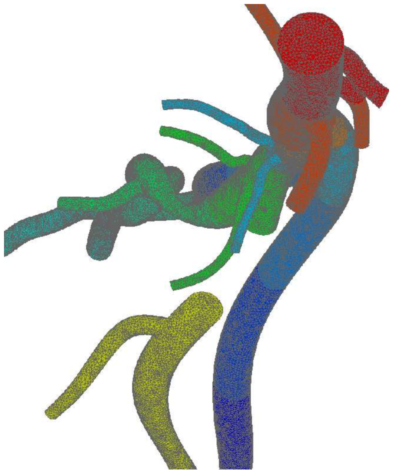
Strict Balance



Minimize Boundary



Mixed

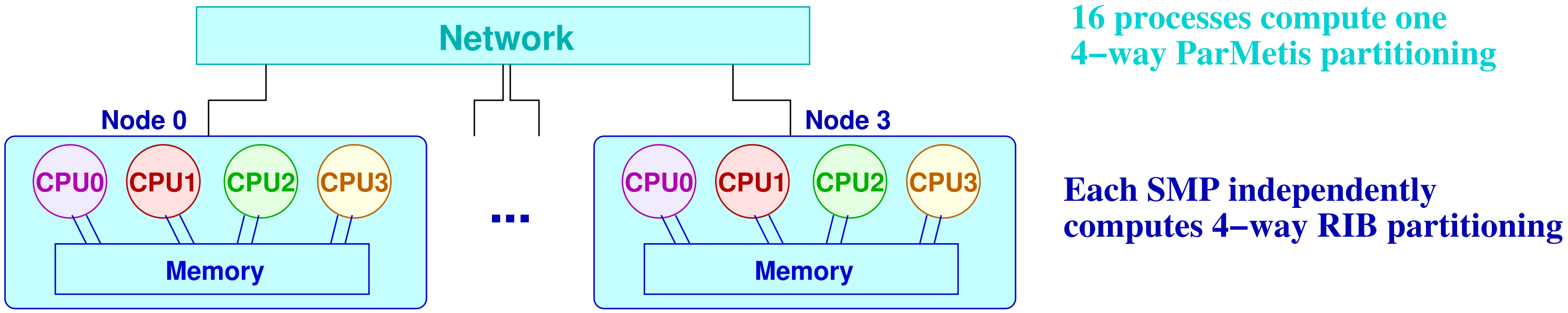


Mixed

- 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

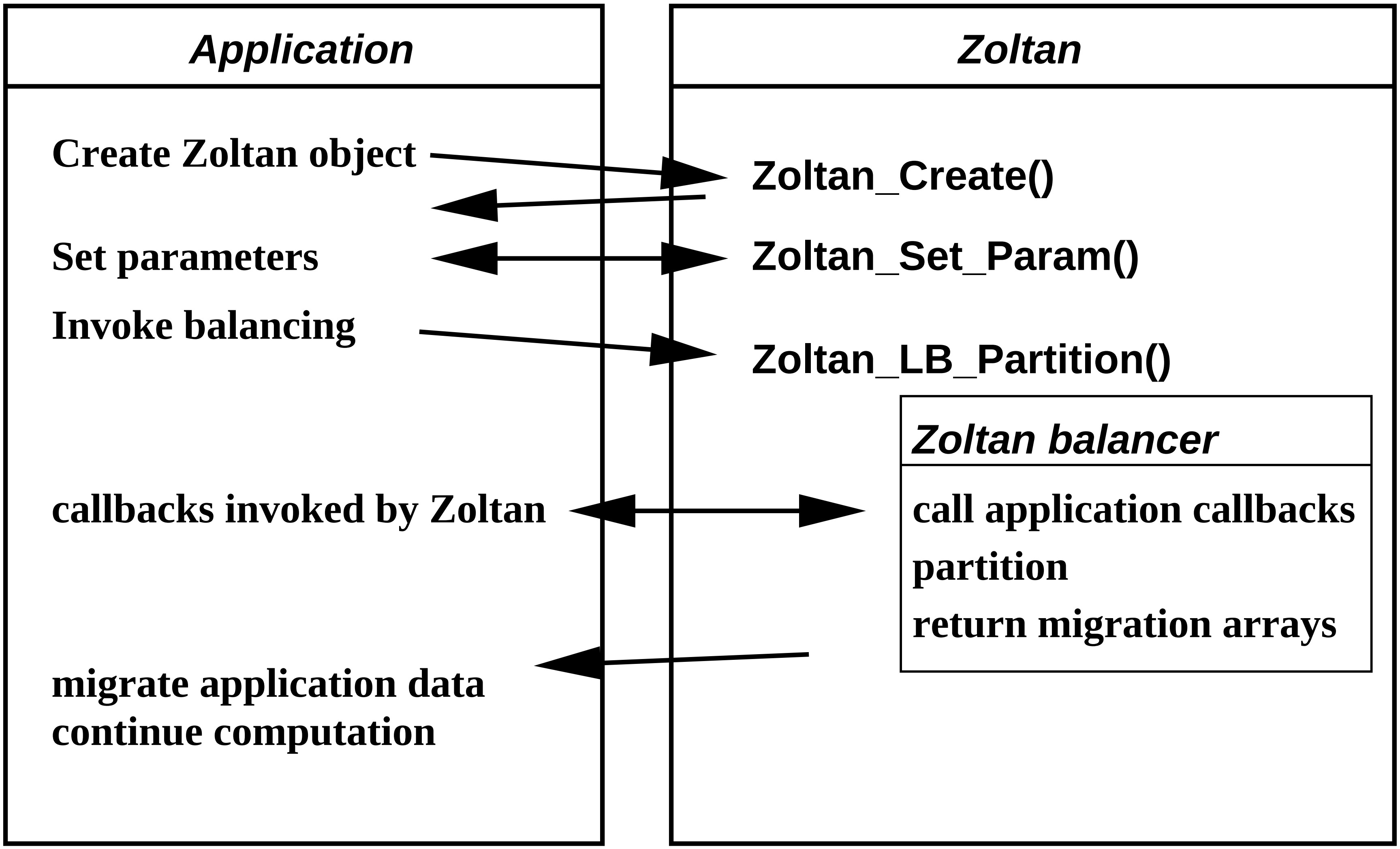
Hierarchical Partitioning and Load Balancing

- Automatic generation of hierarchical partitions using Zoltan method "HIER"

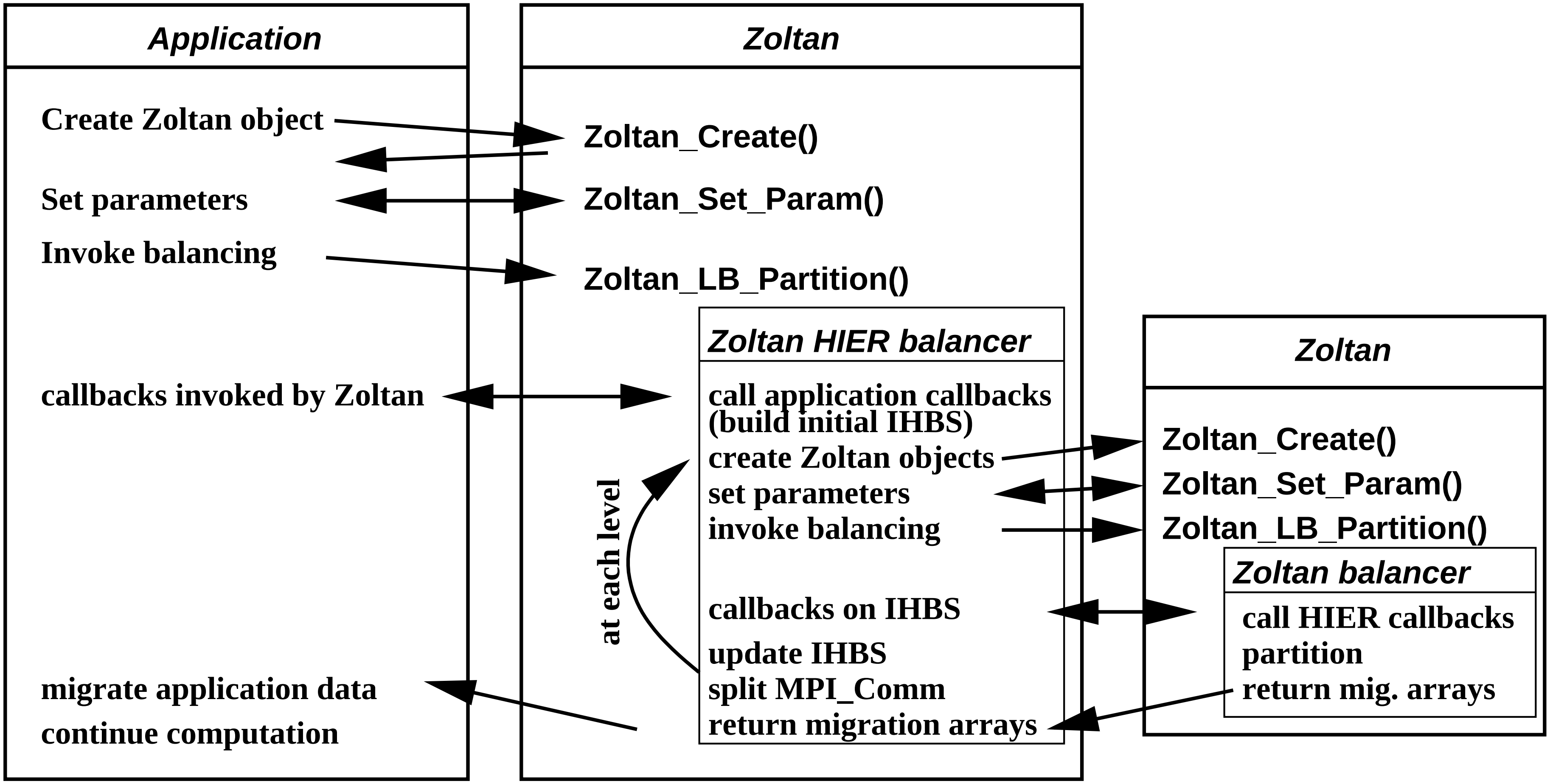


- Implemented during visit to Sandia's CSRI in 2003-04.
- Approach:
  - lightweight "intermediate structure" built from application callbacks
  - intermediate structure is augmented version of the structure built to feed to ParMetis
  - implement internal Zoltan callbacks to access intermediate structure
  - intermediate structure eliminates impact on Zoltan procedures
  - use Zoltan partitioners to partition at each of an arbitrary number of levels
  - object migration only at the end

Load Balancing with the Zoltan Toolkit



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

Specification of Zoltan HIER partitionings

- Small set of new Zoltan callbacks
  - set number of levels of hierarchy
  - at each level, set which partitions to be computed by each process
  - at each level, set LB method and parameters
- zoltanParams library: simple file-based configuration for HIER
  - provides the HIER-related callbacks
  - example

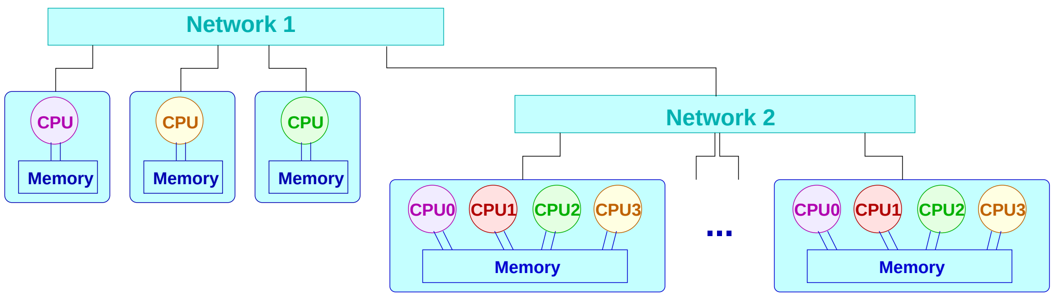
```
LB_METHOD HIER
2
0 0 1 1 2 2 3 3
LB_METHOD PARMETIS
PARMETIS_METHOD PARTKWAY
LEVEL END
0 1 0 1 0 1 0 1
LB_METHOD RCB
LEVEL END
```

  - can also be used for other Zoltan parameters
  - `http://www.cs.williams.edu/zoltanParams/`

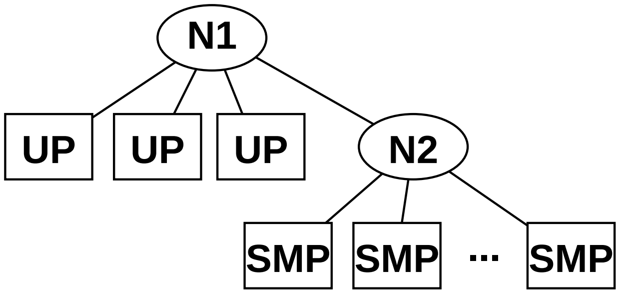
DRUM: Dynamic Resource Utilization Model

A run-time model of the parallel execution environment

(and topic of the next talk)



Computing Environment

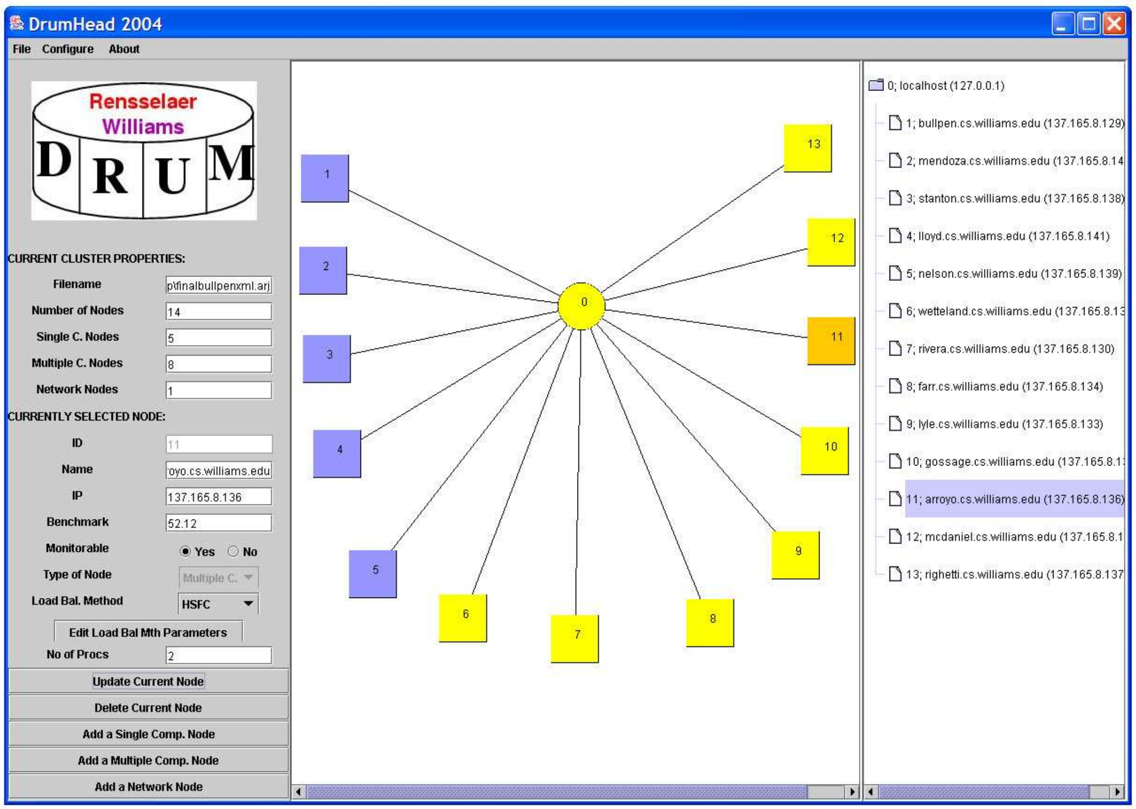


Machine Model

- Tree structure based on network hierarchy
- Computation nodes, assigned “computing power”
  - UP – uniprocessor node
  - SMP – symmetric multiprocessing node
- Communication nodes
  - network characteristics (bandwidth, latency)
  - assigned a computing power as a function of children
- Combine static capability information and dynamic monitoring feedback
- Powers guide creation of weighted partitions using existing procedures

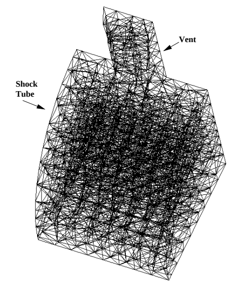
DRUM-guided Hierarchical Partitioning

- Coming soon: HIER partitions guided by DRUM machine model
  - network topology and load balancing parameters specified by DRUM's *DrumHead* configuration tool
  - stored in DRUM's XML-format configuration file
  - DRUM provides callbacks for Zoltan



Preliminary Hierarchical Balancing Results

- Run three-dimensional adaptive simulation



  - discontinuous Galerkin solution of a perforated shock tube
  - start with 69,572 tetrahedral elements, after 4 adaptive refinements, 254,510
  - cluster of multiprocessors: 4 2-way SMPs, 2 4-way SMPs
- measure time to solution for all traditional and hierarchical procedures
- Best hierarchical balancing combination:
  - ParMetis multilevel graph partitioning for inter-node partitioning
  - inertial recursive bisection within each node
- Results depend on how much imbalance is introduced by graph partitioning
  - when little imbalance occurs, graph partitioning produces the best results
  - otherwise, hierarchical partitions help

Hierarchical Partitioning and Load Balancing Current/Future

- Test on more applications, parallel environments (including Grids)
- More verification and testing to include in next Zoltan release
- Better integration with DRUM machine model
- Many efficiency improvements
  - avoid IHBS updates when not needed
  - maintain IHBS between successive rebalancings
  - avoid building redundant structures (*e.g.*, ParMetis applied first)
- Use IHBS to allow incremental enhancements through post-processing
- Use IHBS to compute multiple “candidate” partitionings
  - compute statistics about each
  - only accept and use the one deemed best
- Apply hierarchical structure to other parts of the computation

Other Current Approaches?

Minisymposium speakers today and tomorrow will tell us.

Acknowledgements

The primary funding for the on 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
- Sun multiprocessors at Sandia National Laboratories
- Workstations and multiprocessors in Computer Science and at SCOREC at RPI
- The PASTA Laboratory at Union College

Thanks also to the SIAM CSE05 organizers and all participants for their support of this minisymposium.