CSIS-335 Partitioning and Dynamic Load Balancing

Jim Teresco

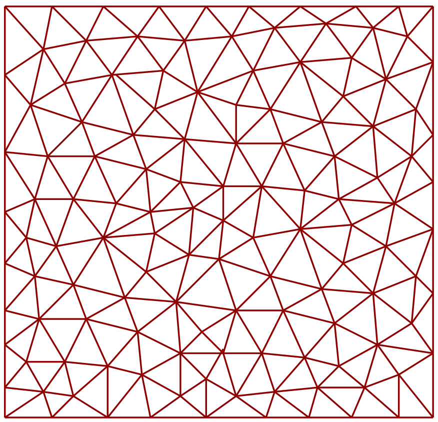
SIENA college
Computer Science

December 2-6, 2024

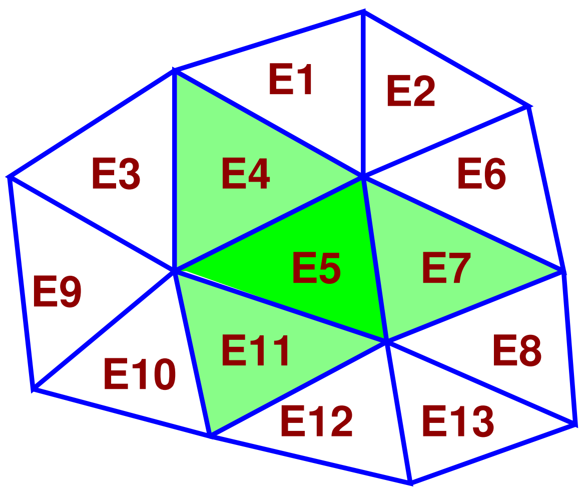
Yet another Powerpoint-free presentation!

Scientific Computation Basics

- Scientists and engineers in many disciplines now rely on simulation
- The governing equations and numerical methods vary
- An important class of problems may be solved using the finite element or related methods



  - discretize the domain into “elements”
  - elements form the “mesh”
  - solve on each element, “paste together” to obtain solution
  - an element’s solution depends on its and its neighbors’ previous values

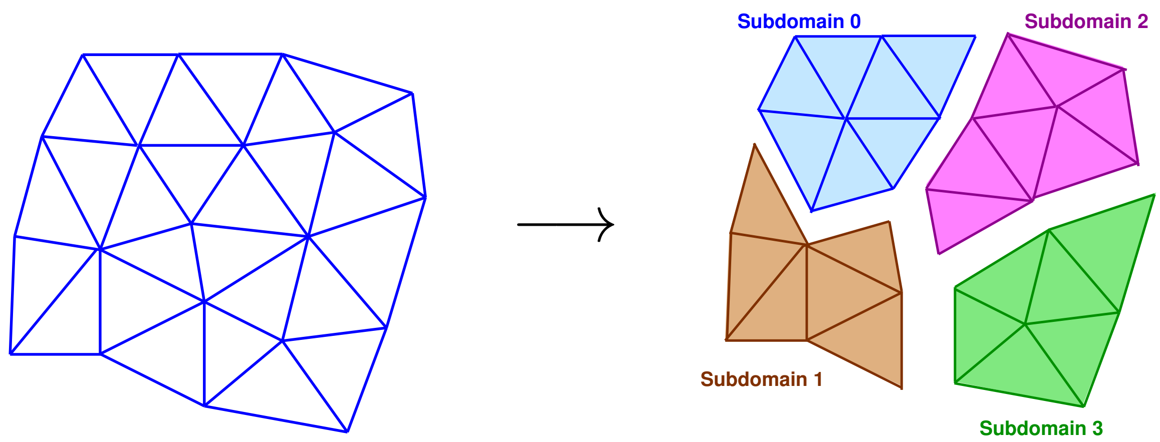


For example, at iteration $i+1$, the value at `E5` is a function of the iteration i value of `E5` and the iteration i values of `E4`, `E7`, and `E11`.

- We will think of these elements as our units of computational work

Parallel Scientific Computation

- Parallel approach: assign disjoint subsets of the mesh to separate processes
- To do this: *partition* the mesh into disjoint subdomains



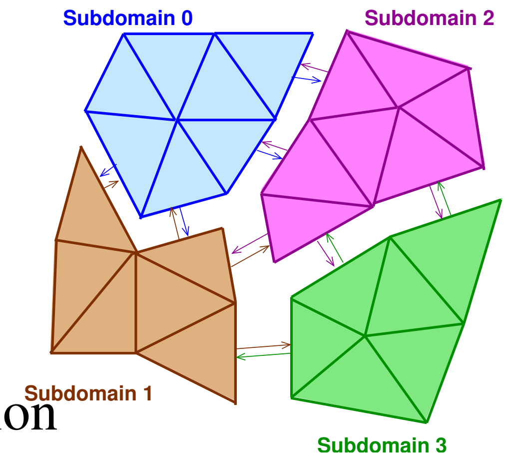
  - Goal: equal work to each, minimize necessary communication

- Partitioning is a major research field in its own right
  - geometric methods: use coordinates only (including Octrees)
  - graph-based methods: use mesh connectivity
  - not today's focus — we will assume good methods exist (they do!)

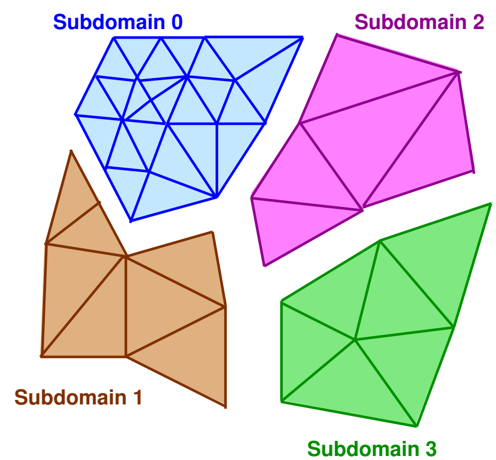
Parallel Scientific Computation: Complications

Many complications worth mentioning

- Distributed data structures
  - locating off-process data
  - interprocess boundary structures
  - interprocess links



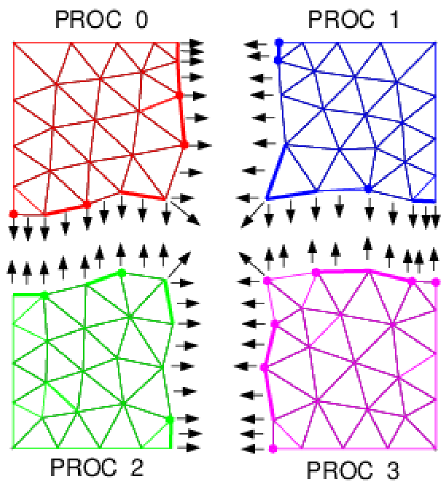
- Adaptivity – the mesh changes with the solution
  - a good partitioning will become imbalanced
  - need mesh migration
  - need dynamic load balancing methods

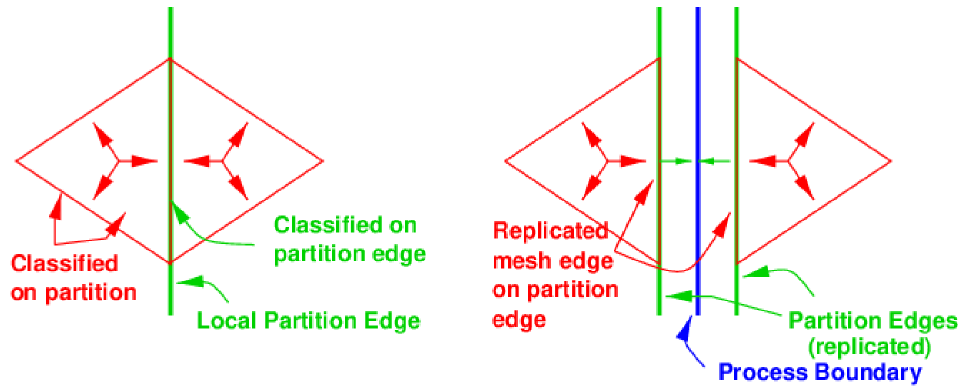


- Parallel development and debugging is hard
  - many threads of execution
  - relative execution speeds differ
  - message ordering adds nondeterminism
  - some errors do not surface except in large cases on many processors

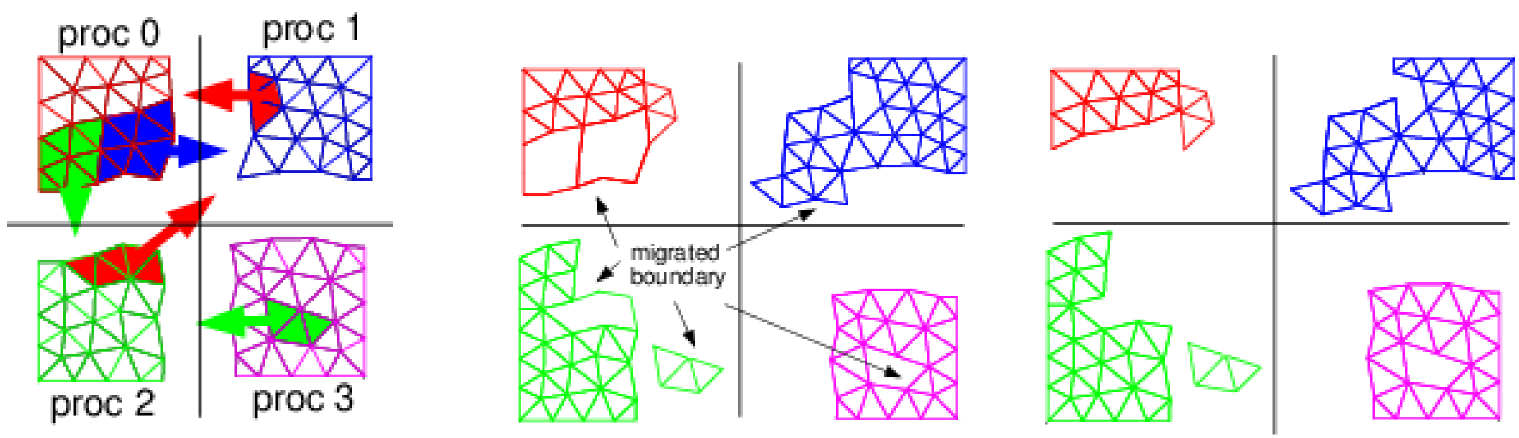
Parallel Mesh Data Structures

- interprocessor boundary structures





- Adaptivity leads to load imbalance: dynamic redisribution is necessary



(a) (b) (c)

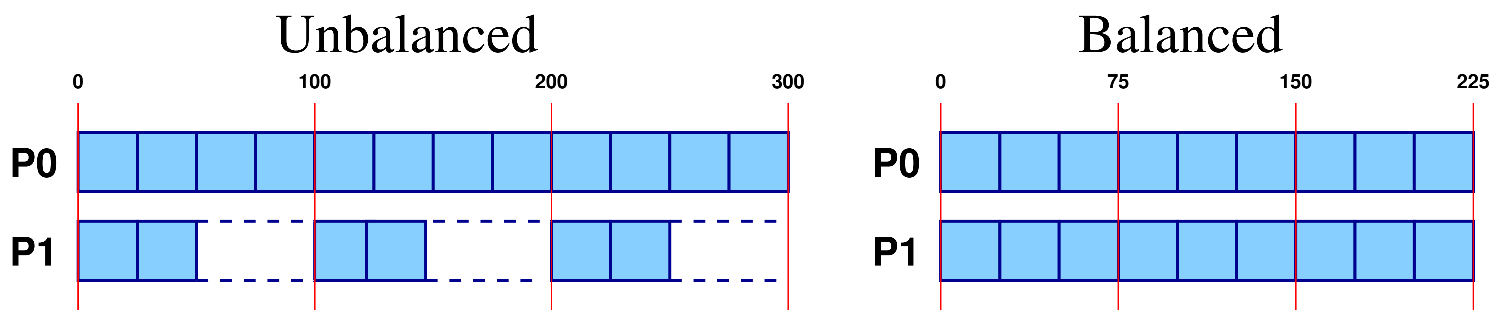
Parallel Mesh Data Structures

Example: Parallel Mesh Database

- Distributed structure supported by a "partition boundary" data structure
- Doubly-linked list of entities on partition boundaries
- Unique owner process for duplicate entities
- All copies know about the unique owner, owner knows about all remote copies
- This involves storing pointers to the memory of a different process!
- See `pmdb/include/private/pmdb_data_st.h`
- Partition Boundary Operators
  - Partition boundary query operators
  - Interprocessor link update operators
  - Scatter-Gather maps

Primary Research Efforts: Dynamic Load Balancing

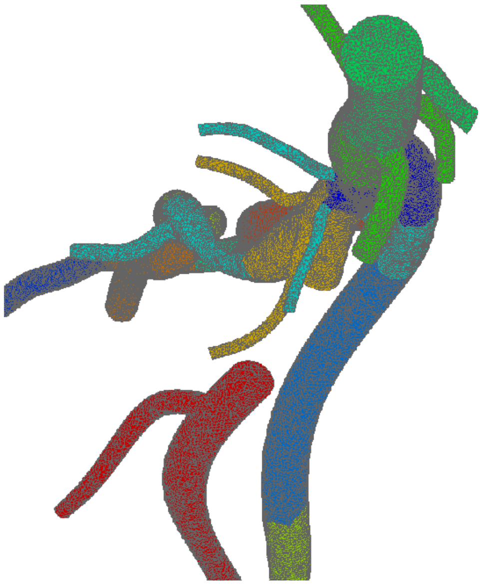
- Problem: keep the cooperating processors doing useful work
  - underloaded processors sit idle waiting for others to complete work

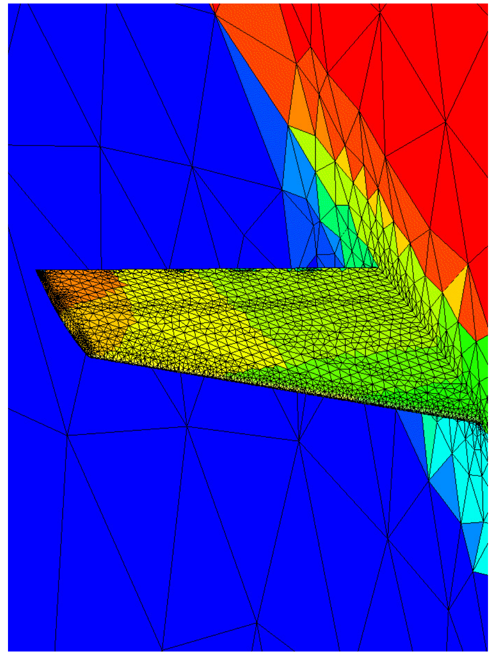


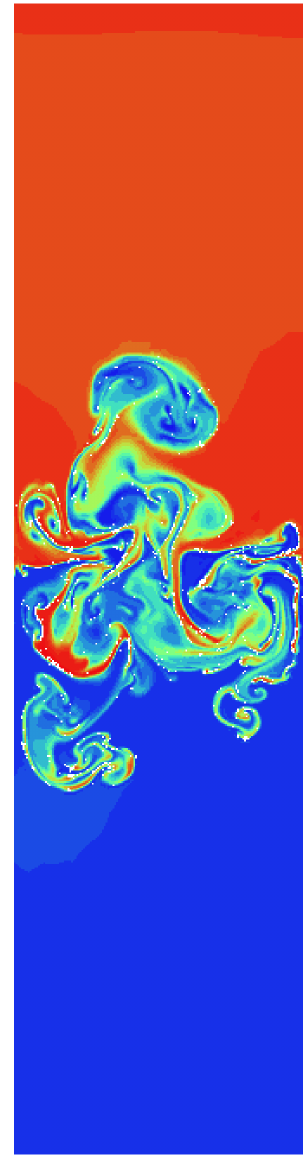
- Main application domain: parallel adaptive scientific computation
- Older work (mid-late 1990s): support tools
  - implementation of distributed data structures
  - dynamic load balancing algorithms and implementations
  - focus on large-scale parallel computers of the era
- More recently: resource-aware parallel computation
  - focus on clusters, in particular heterogeneous and hierarchical clusters
- Today: what do we need to do to take better advantage of all of these cores?

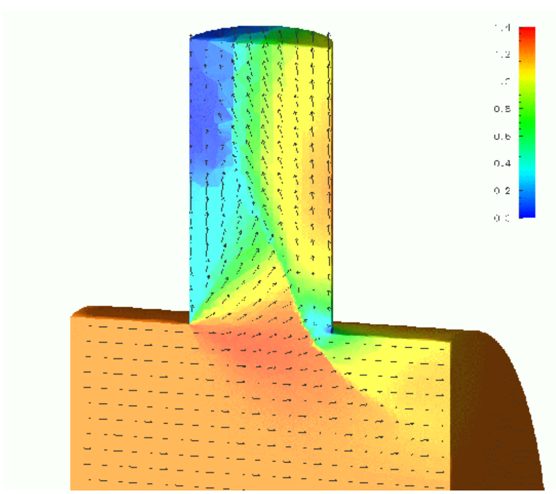
Parallel Scientific Computation Examples

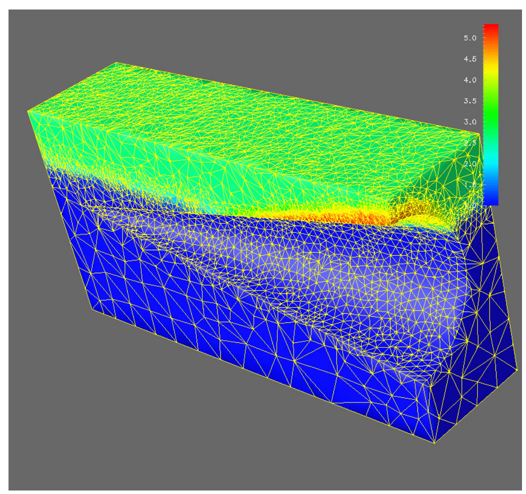
These kinds of methods apply to a wide variety of problems.



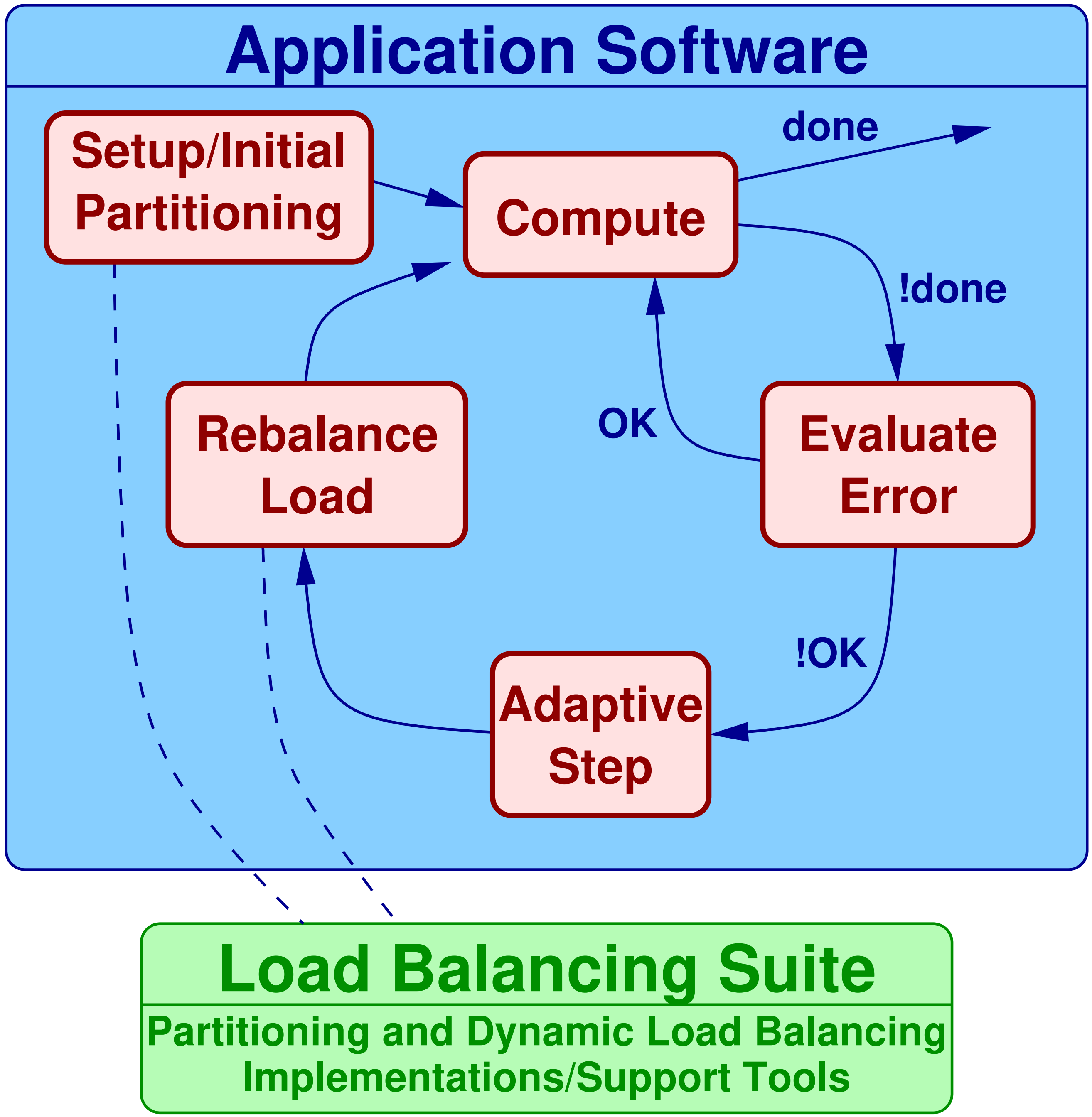








Parallel Adaptive Computation Flow



Partition Quality

What makes for a “high-quality” partitioning of a mesh?

- Big goals for partitioning and dynamic load balancing
  - divide the work evenly for computational balance
  - minimize the needed interprocess communication
- Additional goal dynamic load balancing
  - minimize the change from the current partitioning in the rebalancing

These goals are often competing!

Partition Quality

- Computational balance
  - about the same number of elements per partition
  - use a weighting if computational costs vary
- Minimize interprocess communication
  - communication is necessitated by elements whose neighbors are on a different process
  - Possible metrics (Bottasso, et. al, 1995)
    - a partition's *surface index* is the percentage of all element adjacencies are on a partition boundary
    - the *maximum local surface index* is the largest surface index of any partition
    - the *global surface index* is the percentages of all element adjacencies are on any partition boundary

Partition Quality

Surface index implications

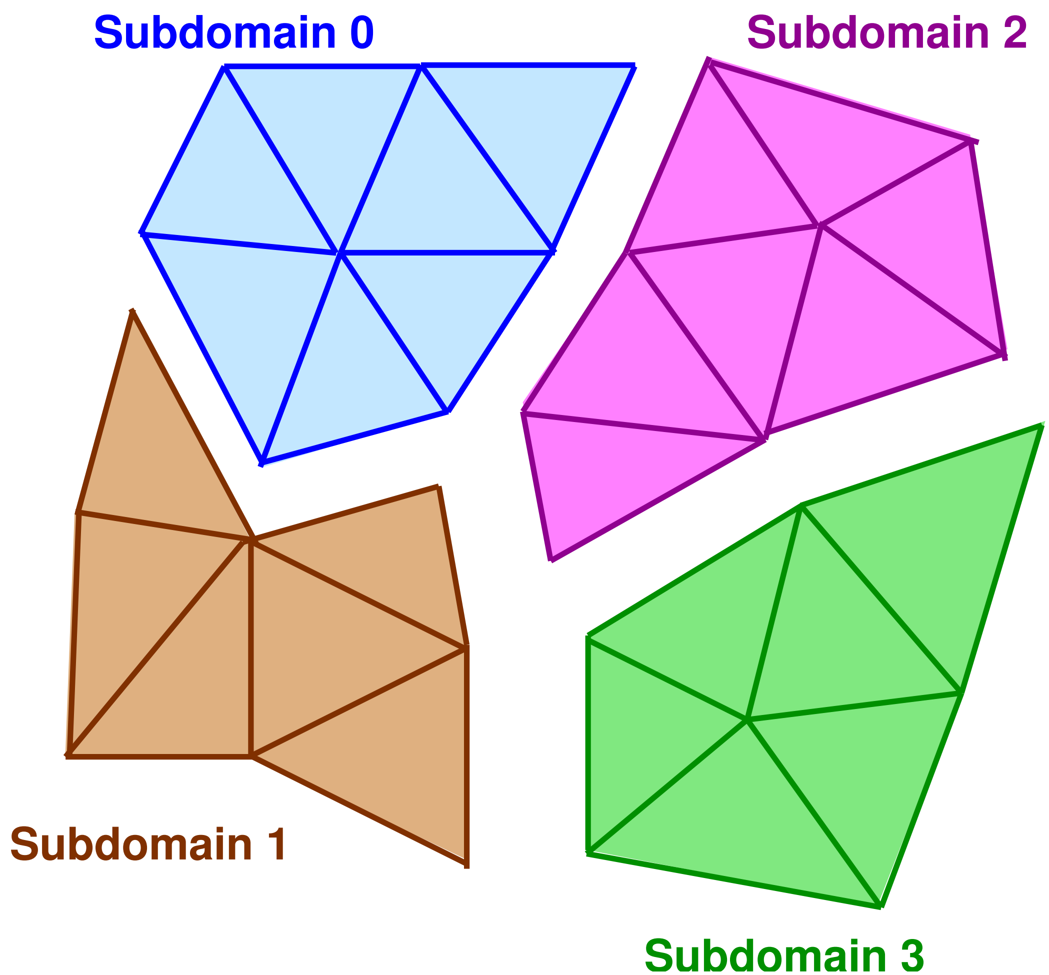
- *maximum local surface index*
  - largest "surface-to-volume ratio" (in 3D meshes) on any process
  - worst case communication relative to on-process data
- *global surface index*
  - total communication volume among all processes normalized by computation - related to the number of "cuts" that a partition creates

More important in some circumstances (high-latency communications) are

- *maximum interprocess adjacency*
  - maximum percentage of other processes with which some process must communicate
- *average interprocess adjacency*
  - average percentage of adjacent processes across all processes

Partition Quality

Let's compute partitioning metrics for the earlier simple example:



Mesh Partitioning/Load Balancing Methods

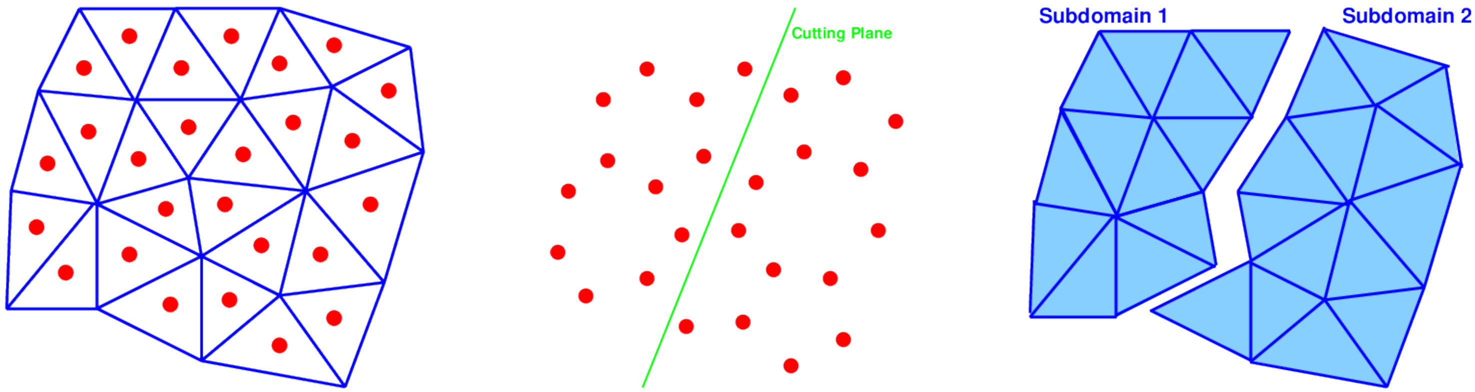
Procedures can be categorized by several features and characteristics

- Computational cost
- Parallel efficiency
- Resulting partition quality – may trade balance for better boundary size
- Incrementality
  - are new partitions as similar as possible to previous partitions?
- Input information required
  - geometric methods – use coordinate information only
  - graph-based methods – use connectivity information
  - hybrid methods – use both coordinates and connectivity

Geometric Mesh Partitioning/Load Balancing

Use only coordinate information

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



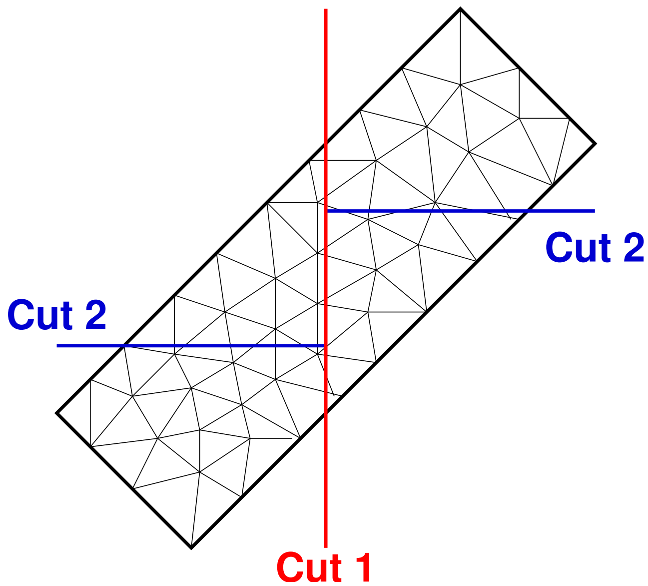
- Tend to be fast, and can achieve strict load balance
- "Unfortunate" cuts may lead to larger partition boundaries
  - cut through a highly refined region
- May be the only option when only coordinates are available
- May be especially beneficial when spatial searches are needed
  - contact problems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

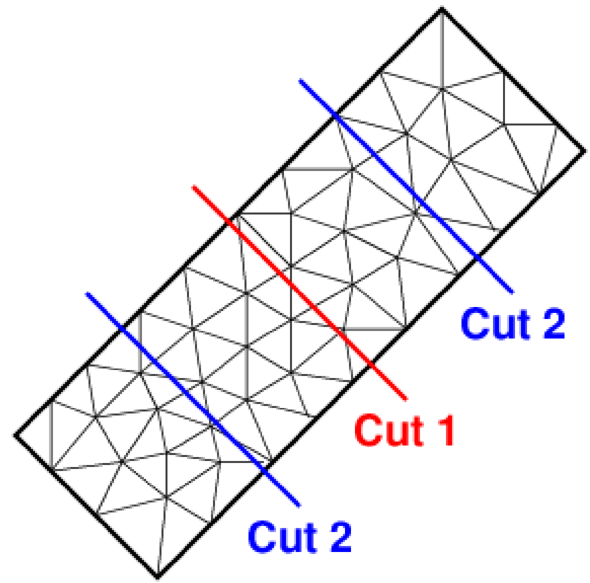
Recursive Bisection Mesh Partitioning/Load Balancing

Simple geometric methods

- Recursive methods, recursive cuts determined by

Coordinate Bisection (RCB) Inertial Bisection (RIB)



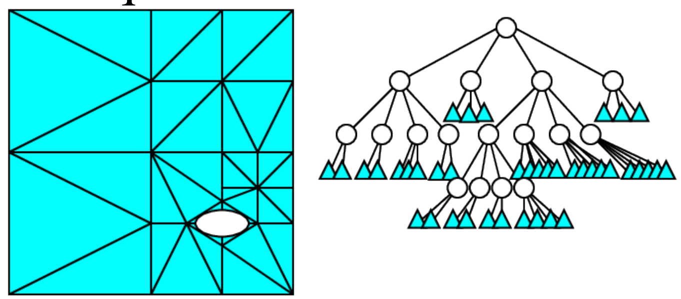


- Simple and fast
- RCB is incremental
- Partition quality may be poor
- Boundary size may be reduced by a post-processing “smoothing” step

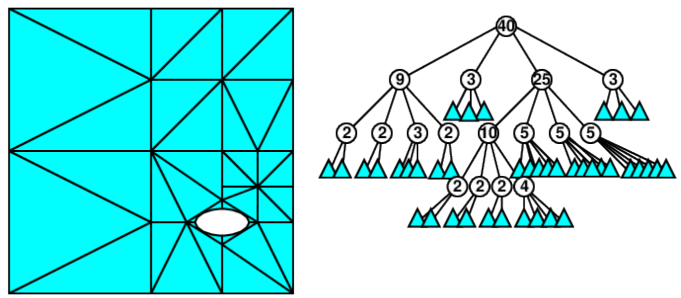
Octree/SFC Mesh Partitioning/Load Balancing

- Quadtree/Octree structure may be used to coarsen the structure

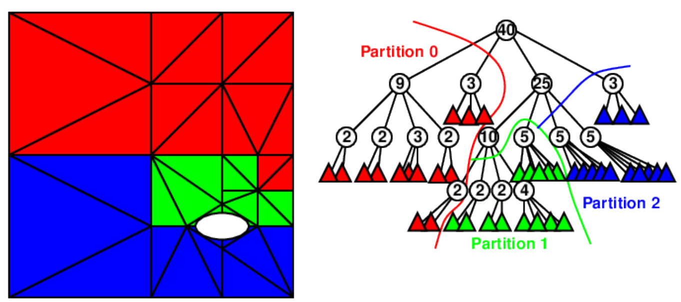
1. insert elements into a quadtree/octree structure



2. assign weights to octants



3. partition through SFC-based truncated tree traversal

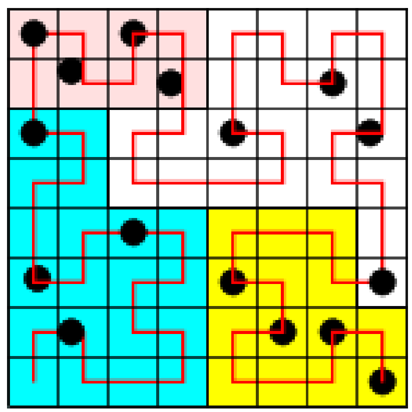


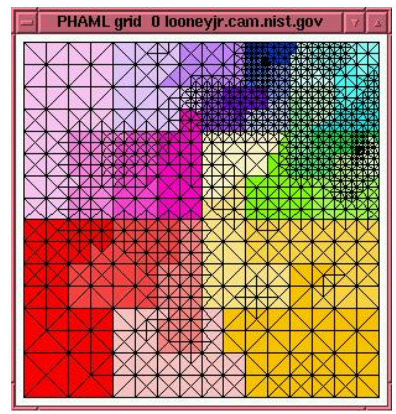
- SFC/Octree methods produce medium-quality decompositions produced at low cost

SFC Mesh Partitioning/Load Balancing

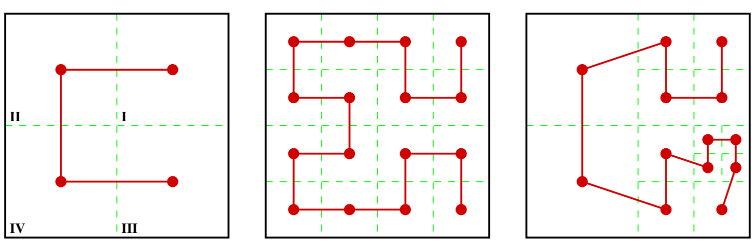
Another geometric method

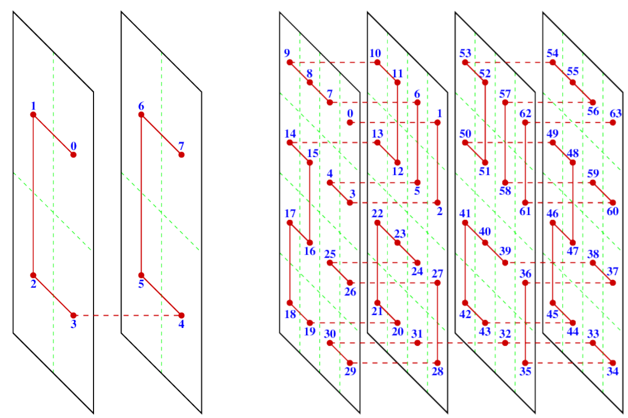
- Use the locality-preserving properties of space-filling curves (SFCs)
- Each element is assigned a coordinate along an SFC
  - a linearization of the objects in two- or three-dimensional space





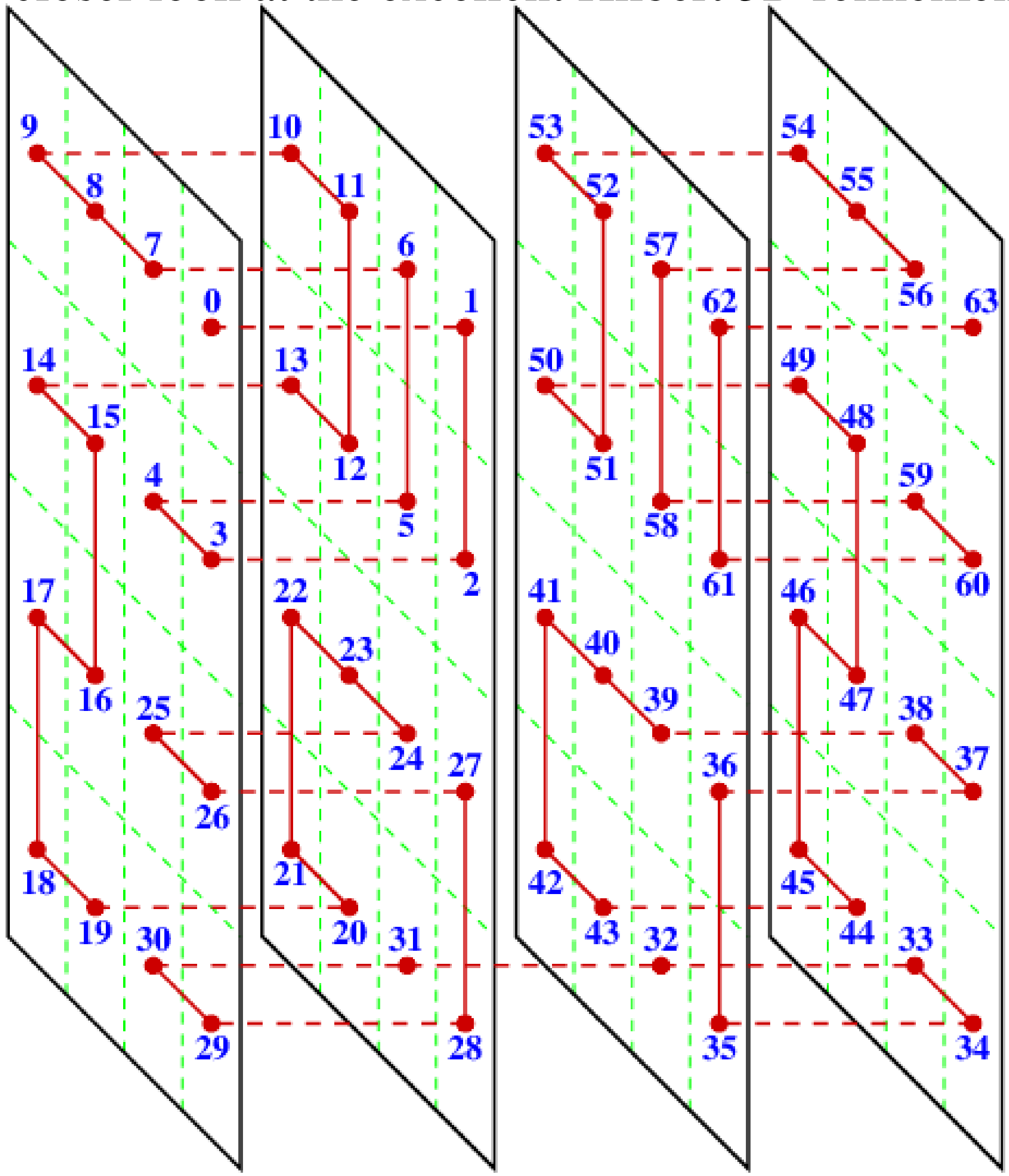
- Hilbert SFC is most effective (check out HDX SFC traversal examples)





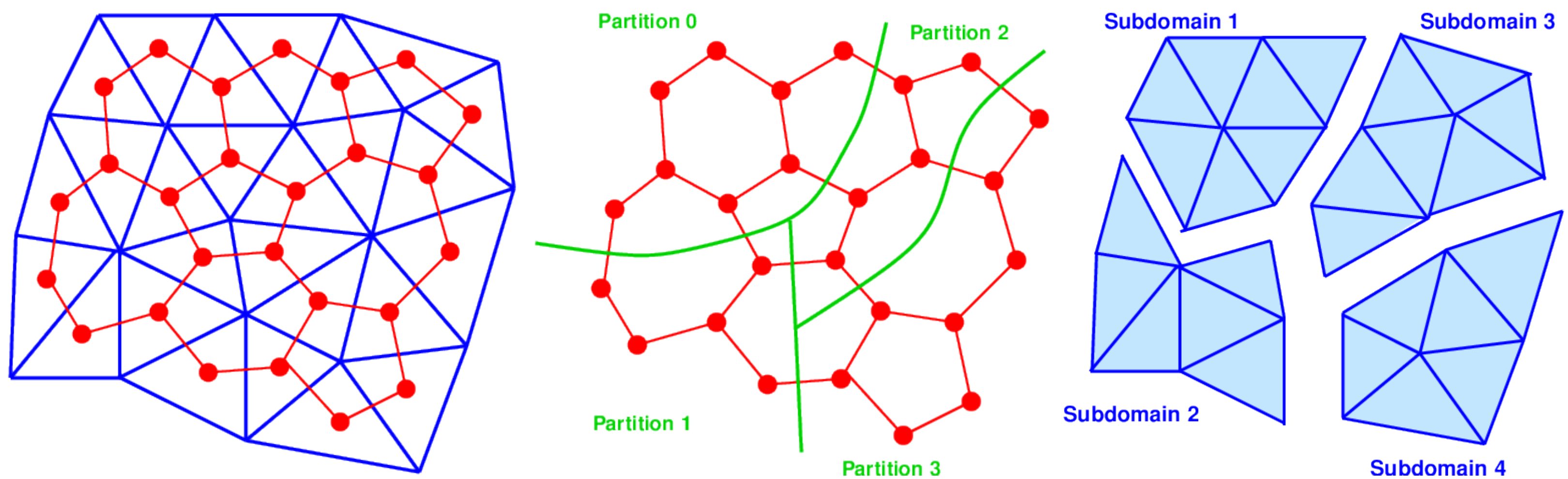
SFC Mesh Partitioning/Load Balancing

A closer look at the excellent Hilbert 3D refinement.



Graph-Based Mesh Partitioning/Load Balancing

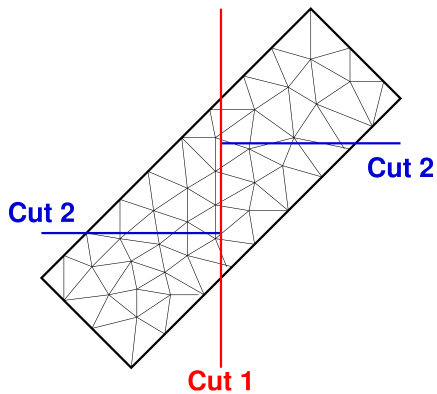
Use connectivity information

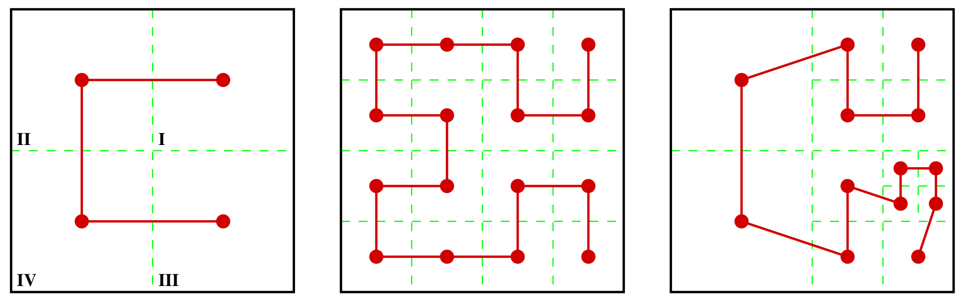


- Spectral methods (Chaco)
  - prohibitively expensive and difficult to parallelize
  - produces excellent partitions
- Multilevel partitioning (Parmetis, Jostle)
  - much faster than spectral, but still more expensive than geometric
  - quality of partitions approaches that of spectral methods
- May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

Dynamic Load Balancing

- Partitioning/dynamic load balancing important for efficiency
- Usual concerns: computational balance, communication minimization
- It's not just graph partitioning





- Experts have developed reusable software libraries: Parmetis, Zoltan, etc.
- Still an active research area
- A common feature of many codes, applications

See: chapter "Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations" by J. D. Teresco, K. D. Devine, J. E. Flaherty, in LNCSE 51, **Numerical Solution of Partial Differential Equations on Parallel Computers**, Bruaset, Are Magnus; Tveito, Aslak (Eds.), Springer, 2006.

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/`