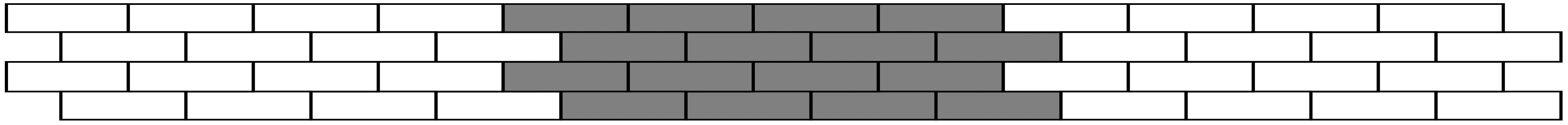
Decomposition of Complex Systems

- **Two types of decomposition**
  - **Geometric or Data decomposition**
  - **Algorithmic or Functional Decomposition**

- **Two types of computational domains**
  - **Homogeneous vs heterogeneous complex systems**
  - **Regular vs Irregular complex systems**

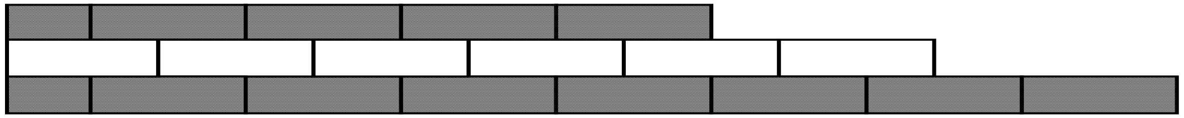
- **Example: a 120 km wall, Members: bricks (atomic units), Processing Elements: (Masons)**

- ***Geometric decomposition* (Vertical)**

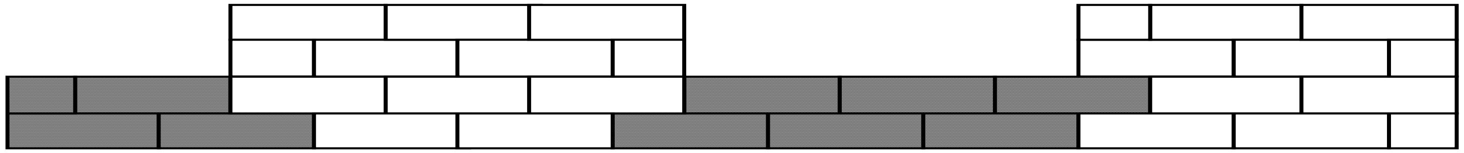


Decomposition of Complex Systems II

- Pipelined Decomposition



- Irregularities in Hadrian's Wall
  - E.g. Workers assigned to locations where the wall is higher should work on shorter lengths to obtain *LoadBalance*

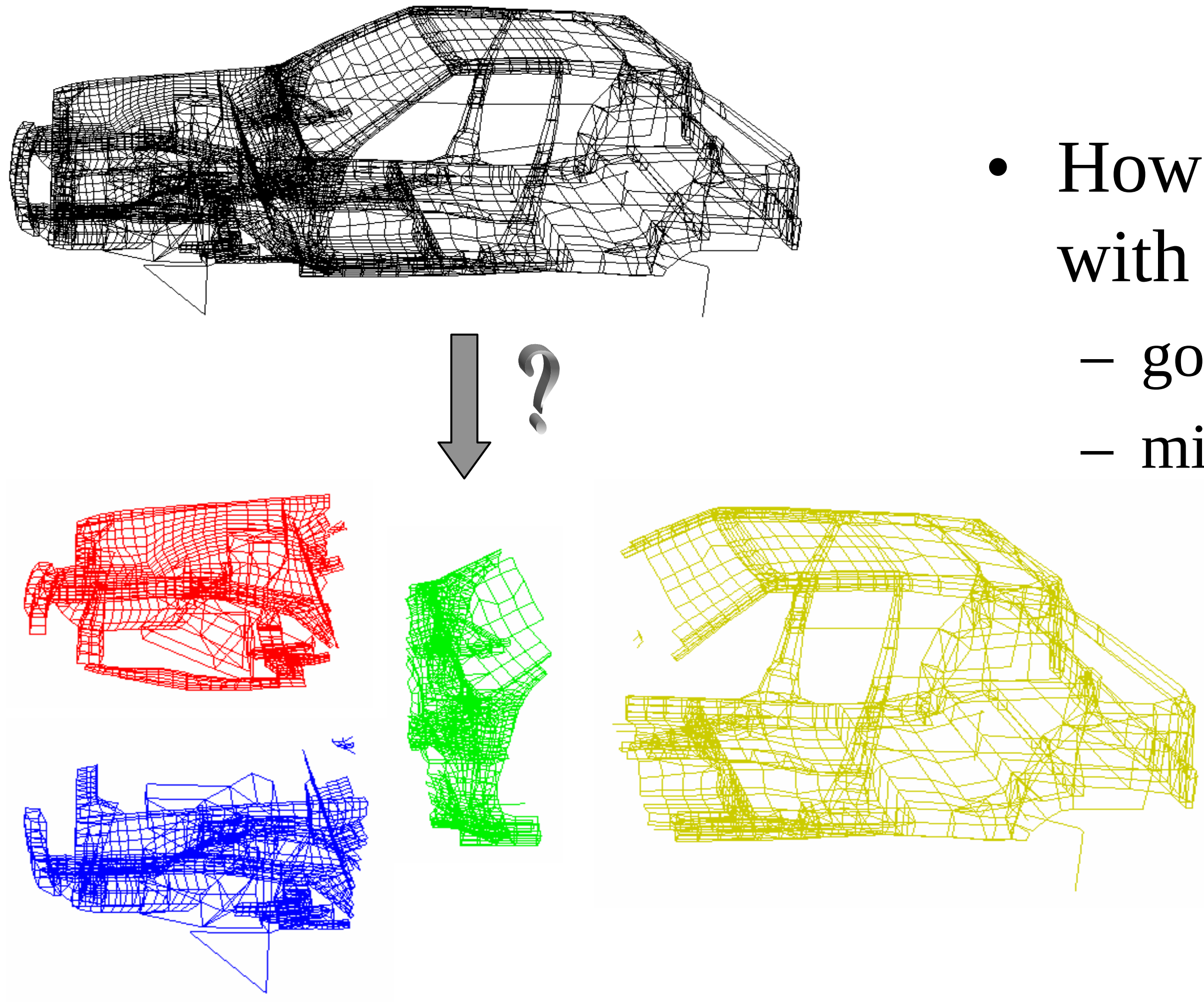


- Inhomogeneities in Hadrian's Wall
  - More than one type of member each with a different (unpredictable) construction time => Dynamic Planning = Dynamic LoadBalancing
  - For Instance a changing set of gargoyles in the wall.

Domain Decomposition

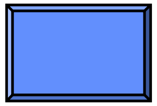
1. Data sets
2. Domain Decomposition vs Load Balancing
3. Domain Decomposition methods
4. An extended example

The Problem

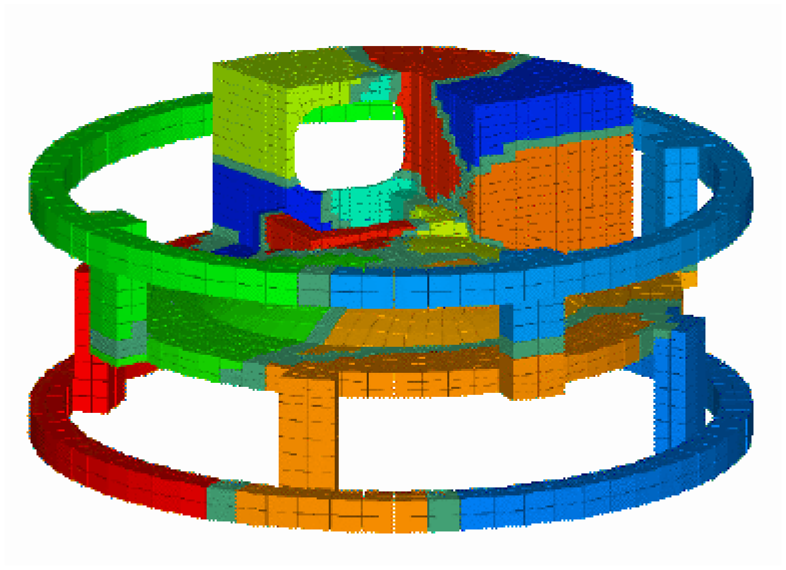


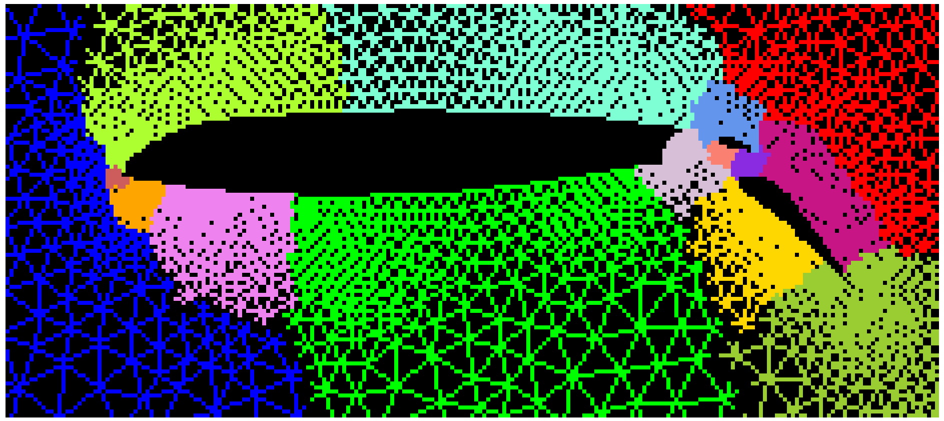
- How to decompose a grid, with
  - good load balance
  - minimal communication

Deformable rod

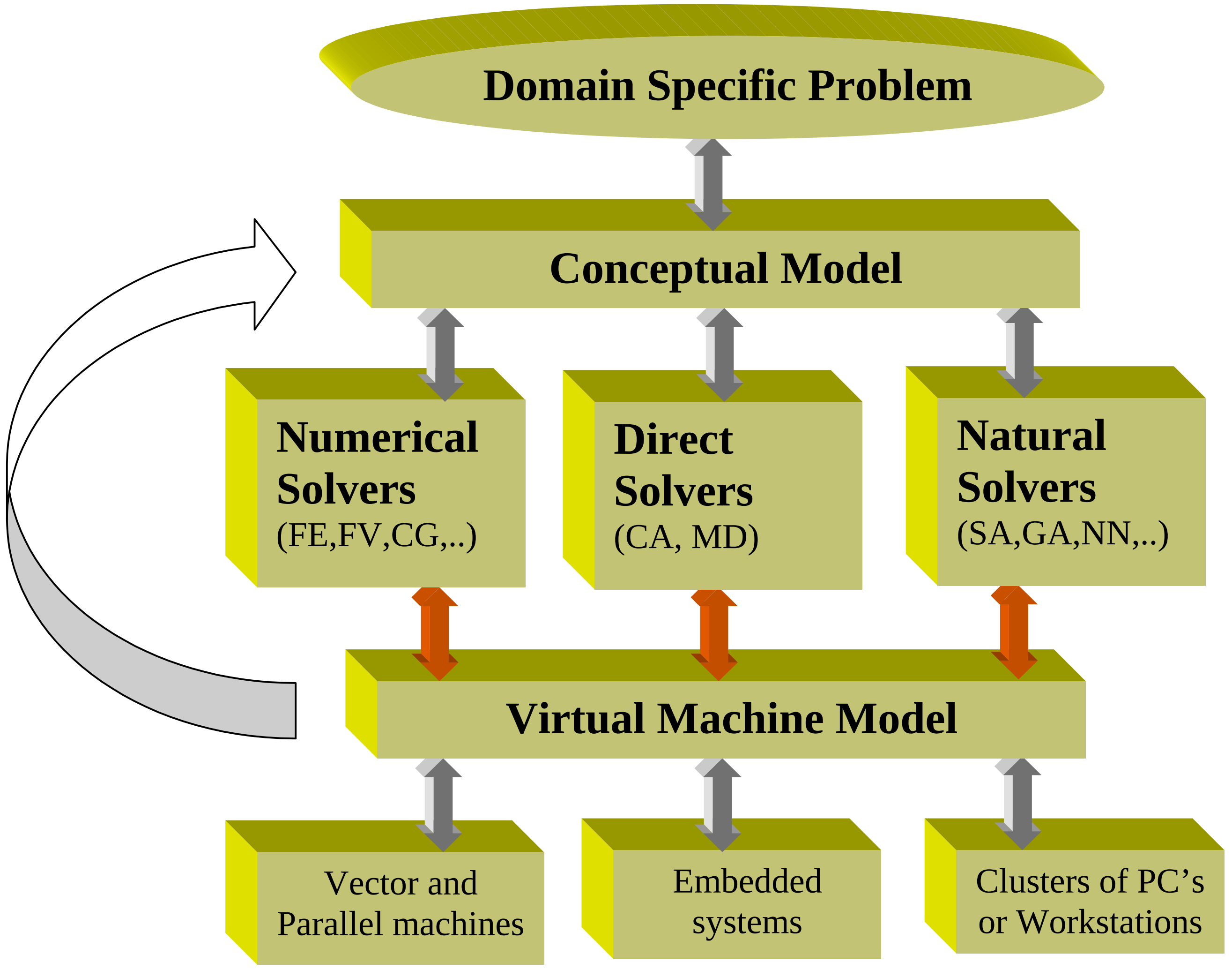


Other examples





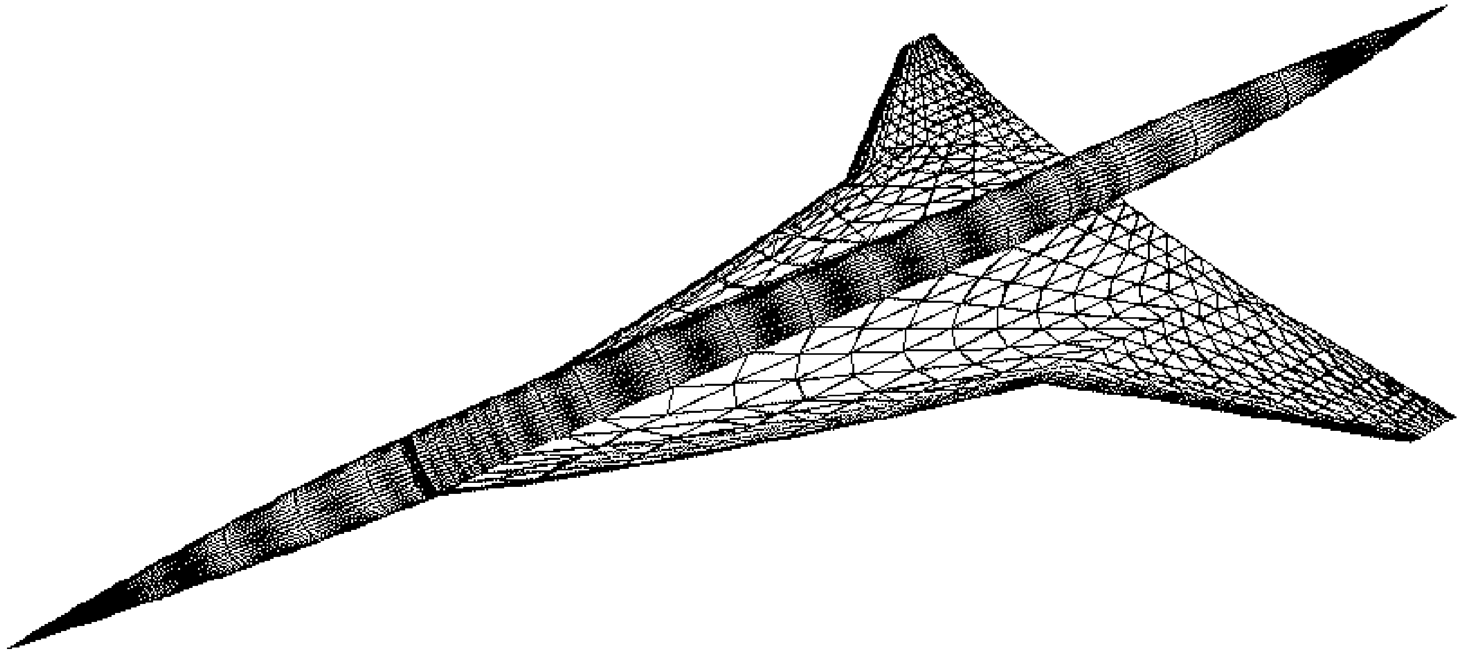
Modeling and Simulation Cycle


Domain Specific Problem
Conceptual Model
Numerical Solvers (FE,FV,CG,..)
Direct Solvers (CA, MD)
Natural Solvers (SA,GA,NN,..)
Virtual Machine Model
Vector and Parallel machines
Embedded systems
Clusters of PC's or Workstations

Data sets

- We restrict ourselves to
  - Simulations on complicated grids...
    - e.g. Irregular, body-fitted Finite Element grids
  - …with data locality….
    - i.e. updating grid elements only requires a local neighborhood
      - i.e. like the five-point stencil in the discretization of the Laplace equation on a Cartesian grid.
  - …and typically the grids are large.
    - Three dimensional, many grid elements.
- These grids allow
  - geometric decomposition
  - inducing communication along the boundaries of the domain.

Example of a Finite Element Grid

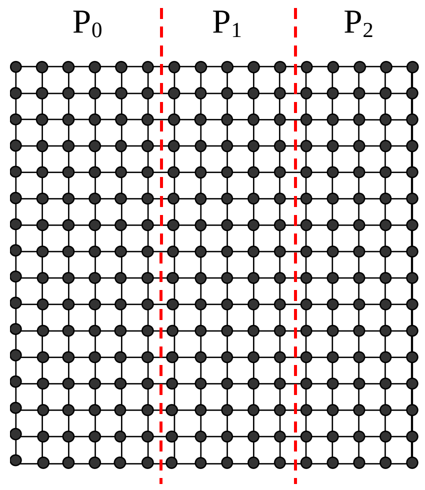


Properties of the grids

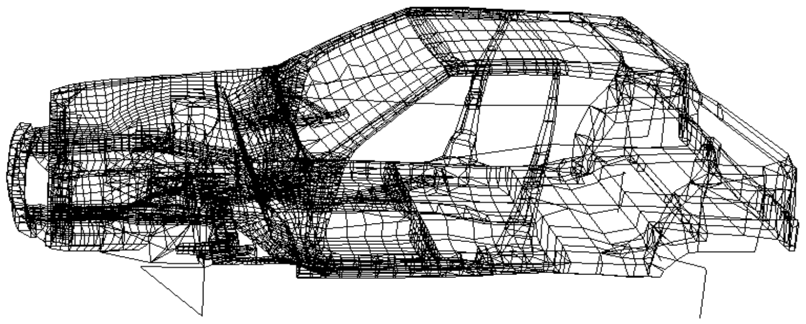
- Homogeneous versus inhomogeneous
  - – homogeneous if the amount of work associated with each element in the grid is the same
    - inhomogeneous grids are also called heterogeneous.
- Regular versus irregular
  - – regular if all elements are structured in some nice pattern, e.g a Cartesian lattice.

Examples of grids

- grids for the parallel Laplace equation in Numerical Solvers part.
  - Homogeneous and regular
    - well, really homogeneous ? What about the boundaries of the domain….



- grids for the car simulation
  - inhomogeneous (different materials) and irregular



The Domain Decomposition problem

- Find a geometric decomposition such that
  - all processors have the same workload ,
    - If processors have the same speed, all processors should have an equal amount of grid elements. If not, the amount of grid elements should be weighted by the computational speed.
  - And the communication between processors is minimized.
- Conflicting goals
  - no communication in sequential processing….
- Very hard optimization problem, in fact NP-hard.
  - We need heuristic methods to solve it.

Subtle difference

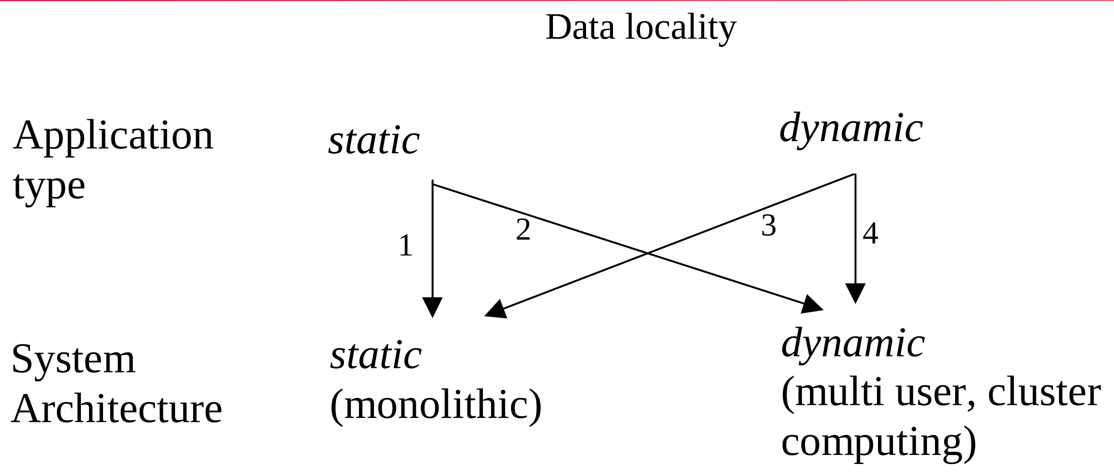
- There’s a subtle difference between domain decomposition and load balancing.
  - Redefine domain decomposition as

    “for a given problem with associated data set, an interconnectivity scheme for the data items and a given parallel machine, find a partitioning of the data domain into a number of subdomains such that

    a. the workload is evenly distributed over all processors in case they are fully dedicated to the problem in question and

    b. the communication is minimized.”
  - So, a good domain decomposition may still suffer from load imbalance if executed on a system with changing background load (i.e. processors that are also used by others).

Load Balancing Strategies



1. “Easy” case
   - by inspection (e.g. “trivial” cubical domains)
   - Static - non-trivial, but can be done before parallel job is started.
2. Load balancing by the run time support system -> e.g. Dynamic PVM
3. Need for rebalancing the application
   - Quasi dynamic
   - Dynamic
4. Rebalancing the application + support from run time system (very hard case !)

Cost function

- Express quality of a decomposition in terms of a cost function.

$$H = H_{calc} + \mu H_{comm}$$

  - H_{calc} is minimized if processors have equal work
  - H_{comm} is minimal when communication time is minimized
  - μ is a parameter that expresses the balance between computation and communication
    - proportional to fractional communication overhead.

H_{calc}

- Many choices are possible, we take

$$H_{calc} = \alpha \sum_{i=1}^{p} W_i^2$$

  - W_i is the load on a processor.
    - For homogeneous domains and identical processors equal to number of grid elements assigned to processor.
    - For heterogeneous domains and/or different processors, more information is needed.

H_{comm}

$$H_{comm} = \beta \sum_{i,j} b_{ij}$$

- where b_{ij} = 1 if element *i* and element *j* are connected and are in different processors, and b_{ij} = 0 otherwise.
- Total communication volume is counted.
  - Neglect possible parallelism in actual communications.
  - In practice no problem due to surface/volume effect.

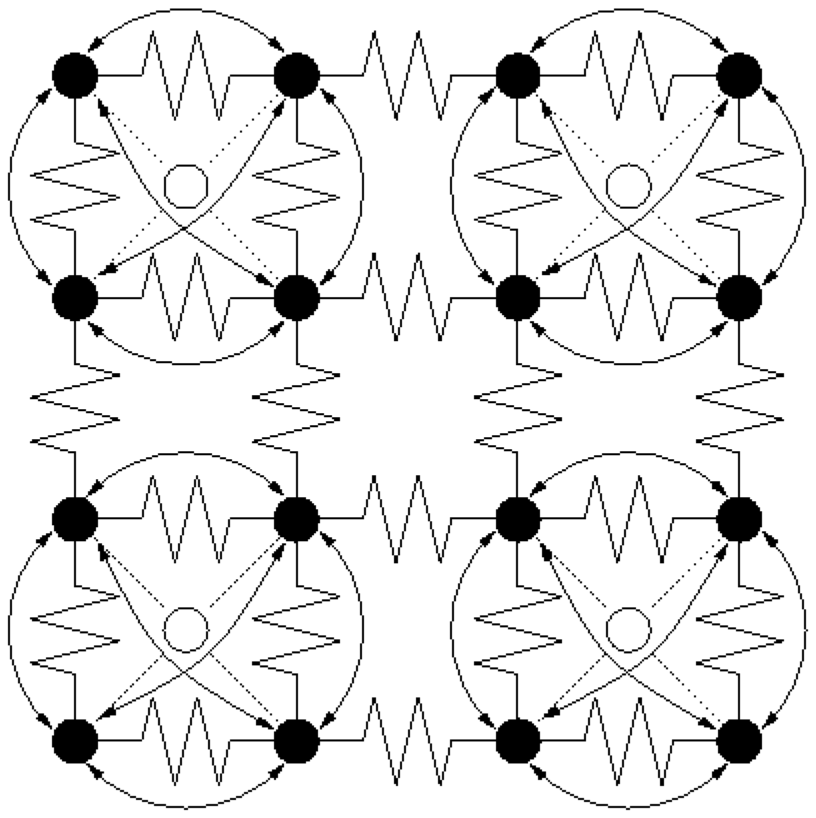
Final form of Cost function

$$H = \alpha \sum_{i=1}^{p} W_i^2 + \mu\beta \sum_{i,j} b_{ij}$$

- Choose α and β such that the optimal solution gives an O(1) contribution from each processor.

$$H = \frac{p^2}{N^2} \sum_{i=1}^{p} W_i^2 + \mu \left(\frac{p}{N} \right)^{\frac{d-1}{d}} \sum_{i,j} b_{ij}$$

DD : physical view

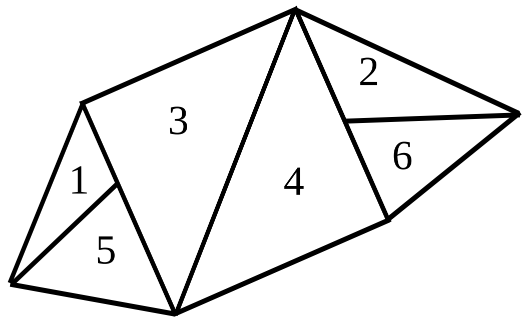


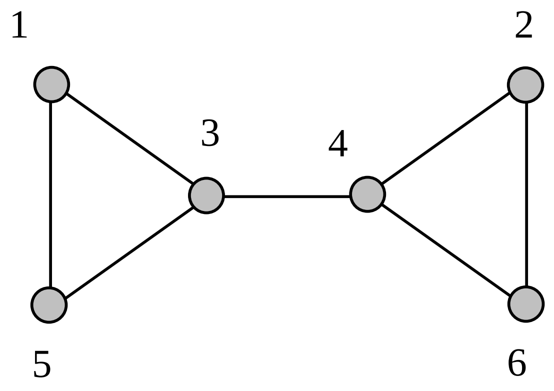
●	Particles
○	Processors
⩘	Attractive particle–particle communication forces
---	Attractive particle–processor redistribution forces
↔	Repulsive particle–particle work balancing forces

Sixteen data points optimally distributed on four processors

illustrating the physical analogy of domain decomposition

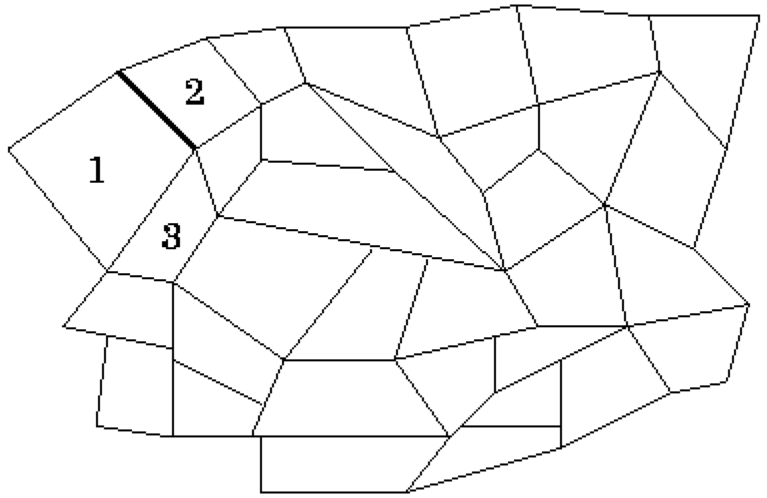
Dual Graph of a Mesh

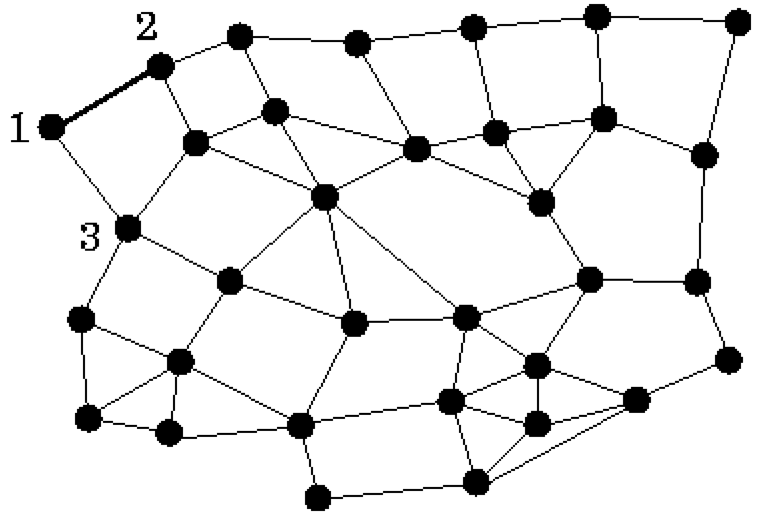




- Each element is represented by a vertex
- Vertices are connected by an edge if the corresponding elements are adjacent to each other

Example of a dual graph





An unstructured mesh consisting of quadrilateral elements (left) and its dual representation (right)

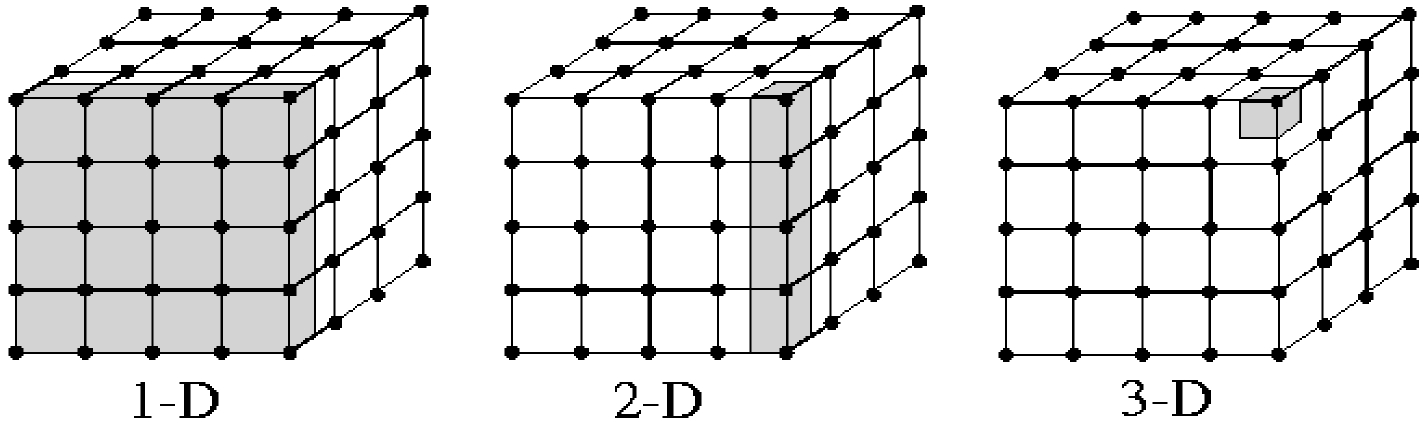
Domain Decomposition as graph partitioning

- Construct a dual graph of a mesh
- Graph bisection problem
  - Given, a graph G, with a set of vertices V and a set of edges E, $G = G(V,E)$, then find a partition $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, such that the size of the cut set E_c, defined as

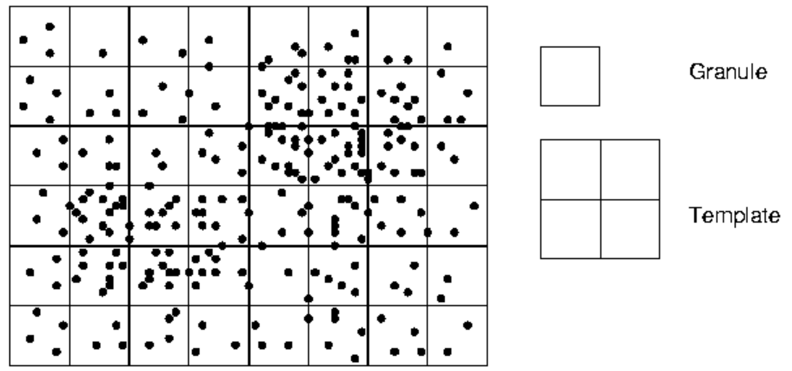
$$|E_c| = |\{e \mid e \in E; e = (v_1, v_2); v_1 \in V_1; v_2 \in V_2\}|$$

is minimized, subject to constraints on the decompostion. If all vertices have equal weight (homogeneous domain) the constraint could be $|V_1| = |V_2|$.

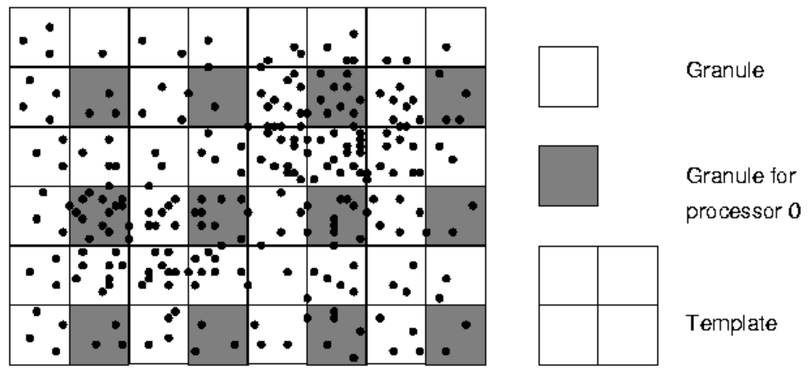
Some simple decompositions



Scattered Decomposition



Scattered decomposition for a 2 x 2 array of processors laid over the domain twelve times



The granules for which processor 0 is responsible

A lesson from Nature : Annealing

- Annealing
  - an experimental technique in which a crystal is first heated just below the melting temperature, and then cooled very slowly to produce perfect crystals.
    - Used a lot in e.g. semi-conductor technology
  - The crystal can choose from many configurations of the atoms, and the perfect crystal corresponds to the minimal energy configuration.
    - Annealing is an “optimization calculation”
    - if the crystal is not heated enough, or cooled to fast, it may get stuck in meta-stable states, corresponding to a local minimum in the energy landscape.
      - amorphous silicon, glassy structures.

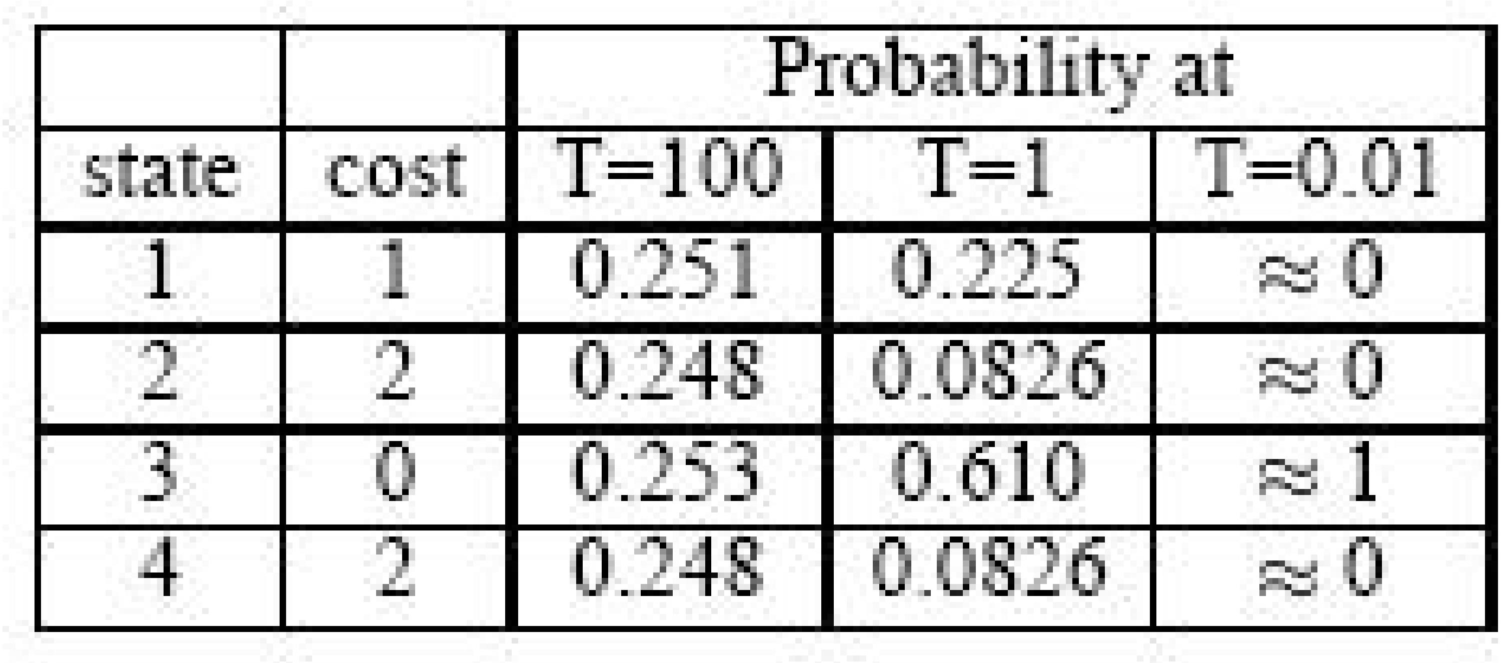
Boltzmann Distribution

- In thermal equilibrium, the probability that the crystal is in a state with energy E is given by the Boltzmann Distribution:

$$P(\mathrm{E} = E) = \frac{1}{Z(T)} \exp\left(-\frac{E}{kT}\right)$$

  - T is temperature, k the Boltzmann constant, Z a normalization function called the partition function. The exp($-E/kT$) is the Boltzmann factor.
  - at high temperature, the probability for any configuration is comparable, the space of all configurations is accessible to the system.
  - At very low temperatures, only the state with minimal energy has a finite probability.

Simulated annealing



		Probability at		
state	cost	T=100	T=1	T=0.01
1	1	0.251	0.225	≈ 0
2	2	0.248	0.0826	≈ 0
3	0	0.253	0.610	≈ 1
4	2	0.248	0.0826	≈ 0

Simulated Annealing (SA)

- Start at a certain temperature (control variable)
- Equilibrate the system and sample configurations according to the Boltzmann distribution

  $\Rightarrow$ Metropolis Algorithm
- Slowly decrease the temperature, until the system settles into the, hopefully, global minimum.

  $\Rightarrow$ Cooling Schedule

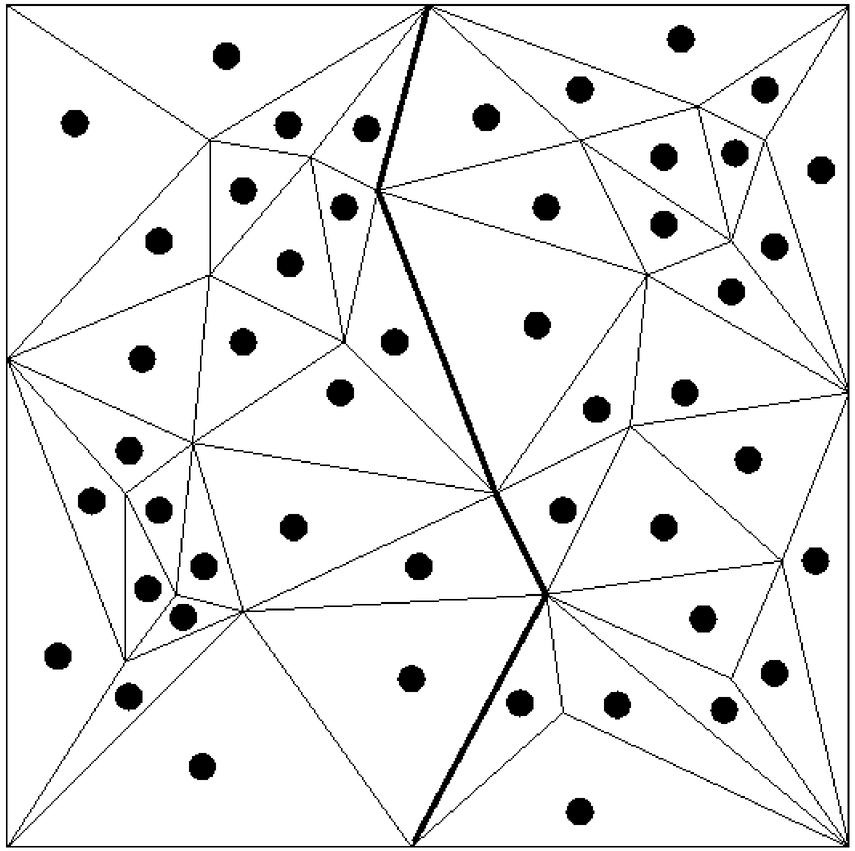
Recursive bisection methods

- Idea: Find a good graph bisection method
  - some heuristic method
- and apply this recursively.
- How good is this, do we find good partitions?
  - In general, NO
  - For well shaped meshes (as in most cases), YES
    - almost always close to optimal solution
  - see H.D. Simon, S-H Teng, “How good is recursive bisection?”, SIAM J. Sci. Comput. **18** (1997) 1436-1445.

Recursive Coordinate Bisection (RCB)

- Assign coordinates to vertices : $v_i = (x_i, y_i)$.
- Take longest dimension of domain, e.g. the x-direction.
- Sort vertices according to coordinate in longest dimension (e.g. the x-coordinates).
- Split the list in two.

Example RCB

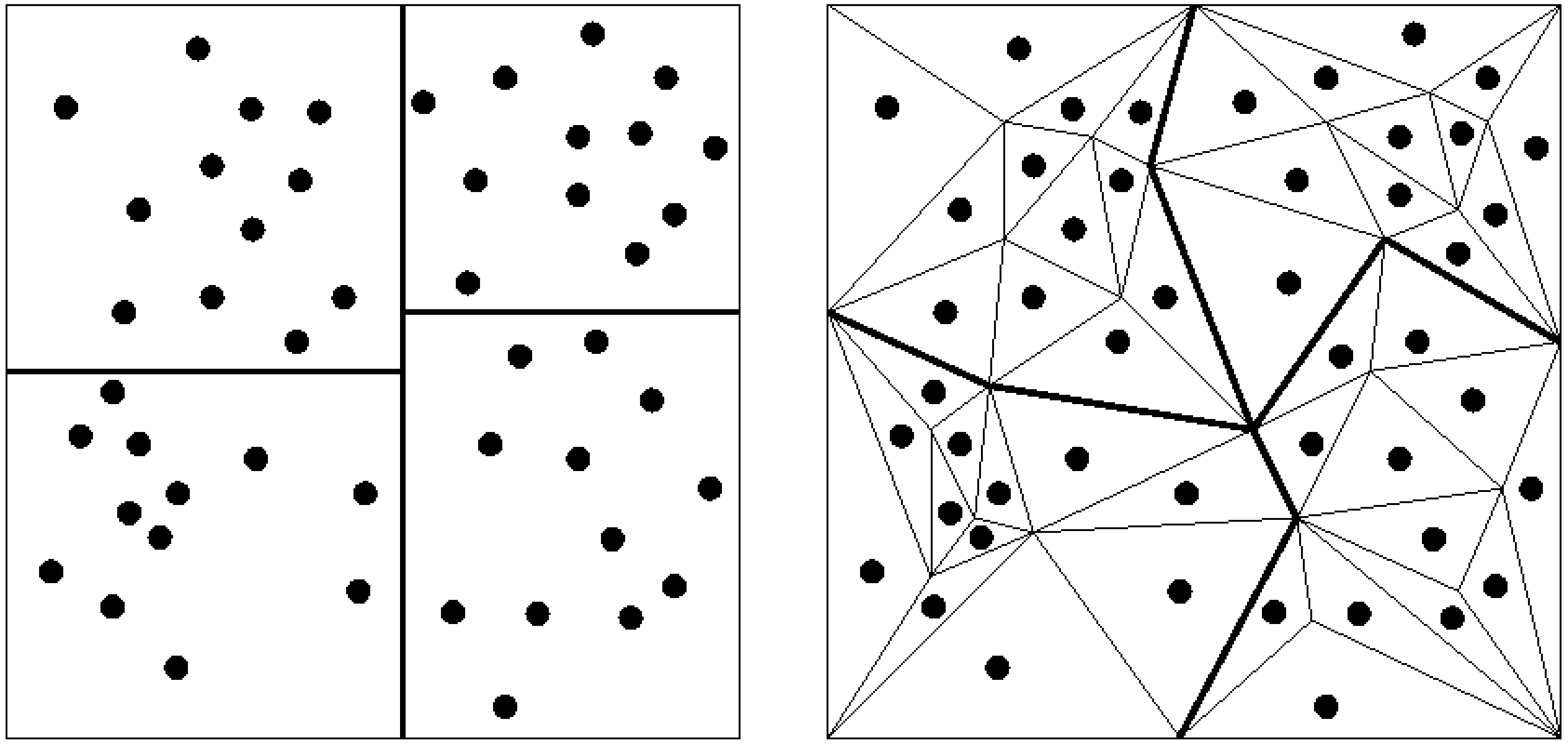


A single step of coordinate bisection for a simple mesh

Orthogonal Recursive Bisection (ORB)

- As RCB, but
- now do not split the domain in the longest dimension, but alternate over orthogonal directions, i.e. x-y-z-x-y-z- etc.

Example ORB

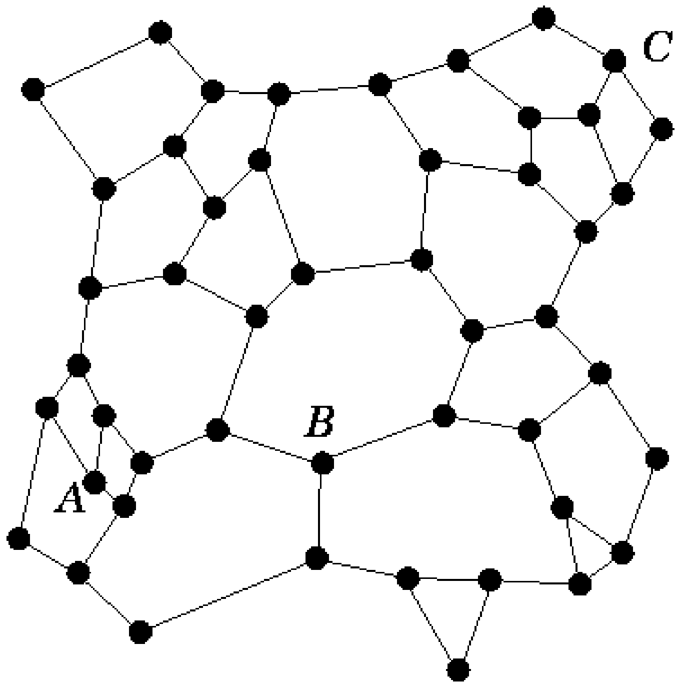

Two steps of ORB for a simple mesh. The domain is split vertically and then each half is split horizontally (left). The resulting subdomains are shown in the right figure

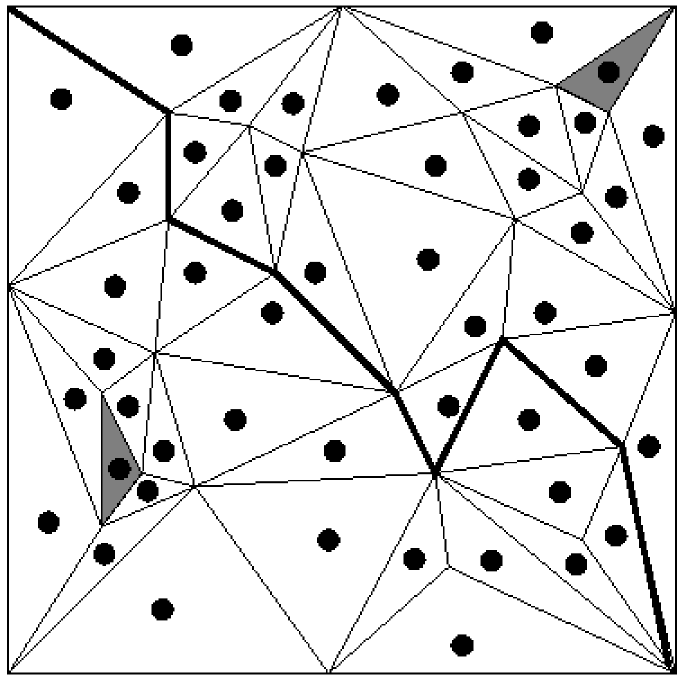
Recursive Graph Bisection (RGB)

- RCB and ORD do not use any connectivity information.
- RGB uses this by looking at the dual graph, and by defining a distance as d_{ij} = |shortest path from v_i to v_j|.

- Determine two vertices in the graph that have the maximum distance.
- Sort all vertices using distance to one of those extremes.
- Split the list.

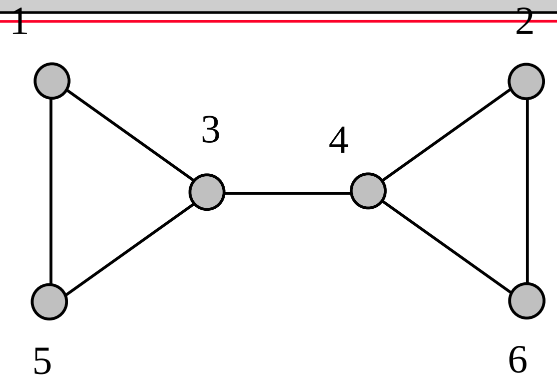
Example RGB





A single step of RGB for a simple mesh. The left figure shows the dual graph with the vertices A and C that are farthest apart. In the right figure the elements closer to C than to A are assigned to one subdomain and the other elements to the other subdomain

Recursive Spectral Bisection



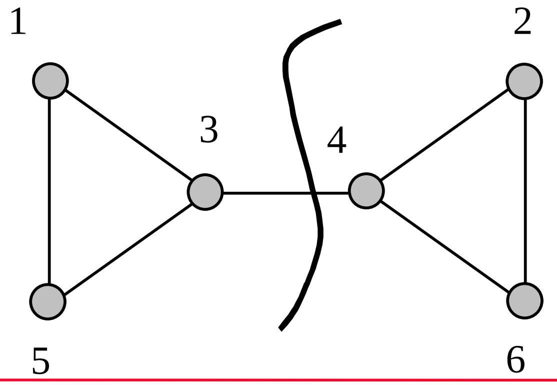
$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad \mathbf{D} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

$$\mathbf{L}_G = \mathbf{A} - \mathbf{D} = \begin{pmatrix} -2 & 0 & 1 & 0 & 1 & 0 \\ 0 & -2 & 0 & 1 & 0 & 1 \\ 1 & 0 & -3 & 1 & 1 & 0 \\ 0 & 1 & 1 & -3 & 0 & 1 \\ 1 & 0 & 1 & 0 & -2 & 0 \\ 0 & 1 & 0 & 1 & 0 & -2 \end{pmatrix}$$

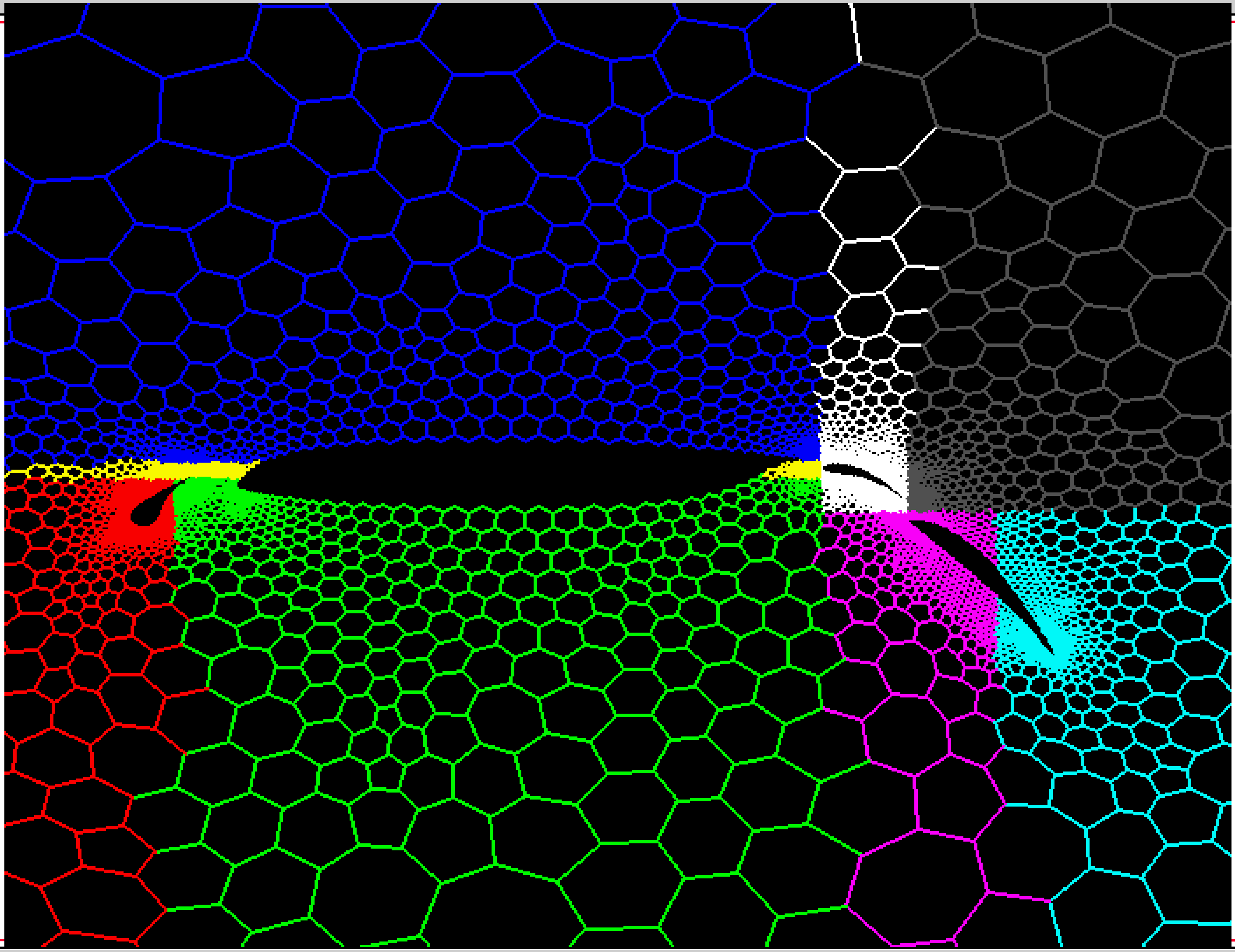
Fiedler vector, i.e. the second eigenvector of $\mathbf{L}_G$

$$\mathbf{e}_2 = \begin{pmatrix} -1.0 \\ 1.0 \\ -0.561 \\ 0.561 \\ -1.0 \\ 1.0 \end{pmatrix}, \lambda_2 = -0.438$$

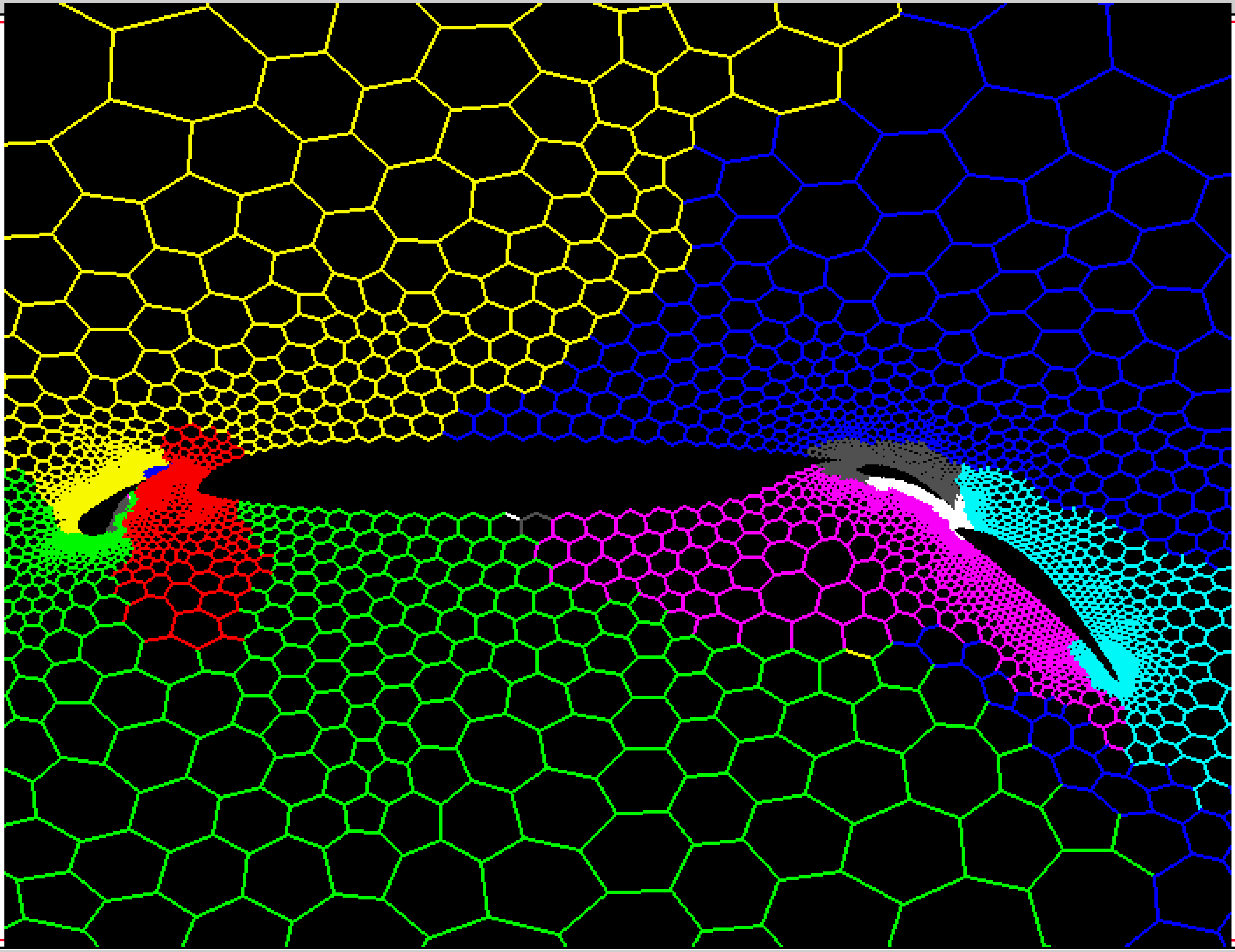
This gives the ordening {1,5,3,4,2,6}, so the bisection splits the graph in the subgraphs {1,3,5} and {2,4,6}.



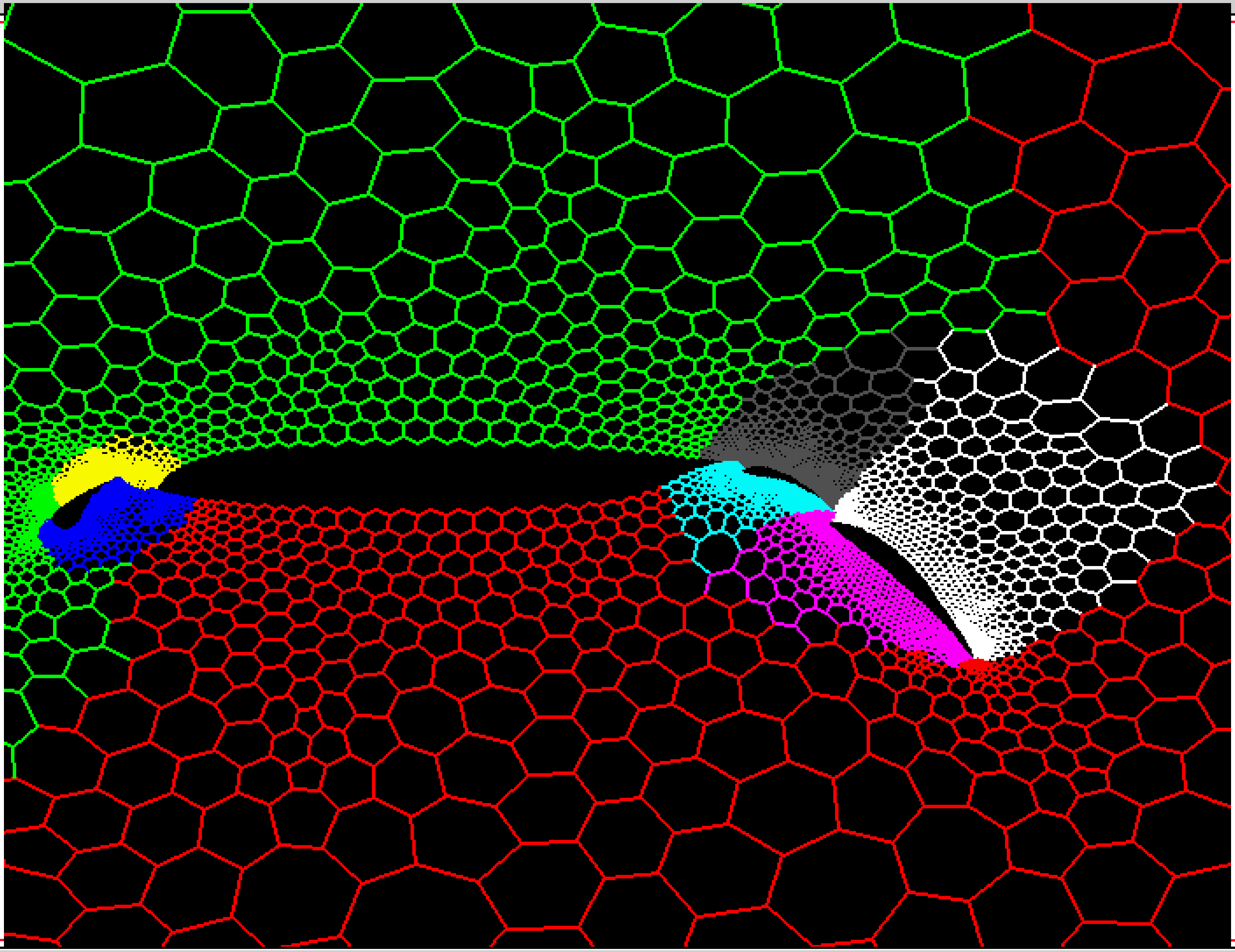
Airfoil with RCB (8 sub domains)



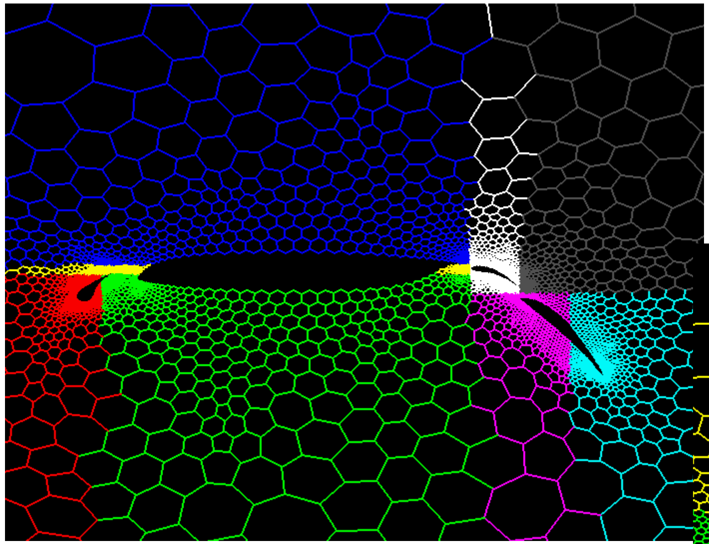
Airfoil with RGB (8 sub domains)



Airfoil with RSB *(8 sub domains)*

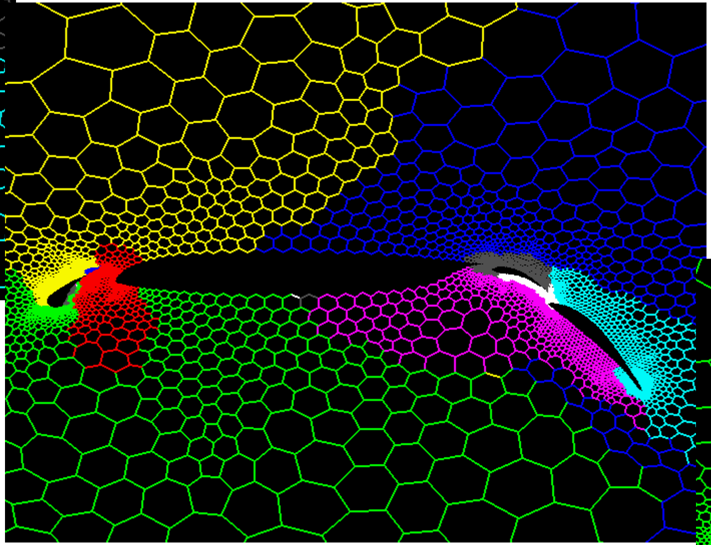


Comparison 1

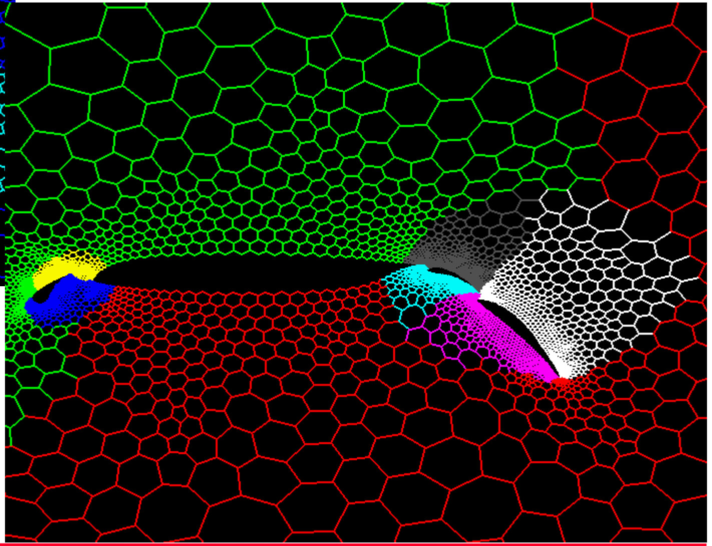


RCB

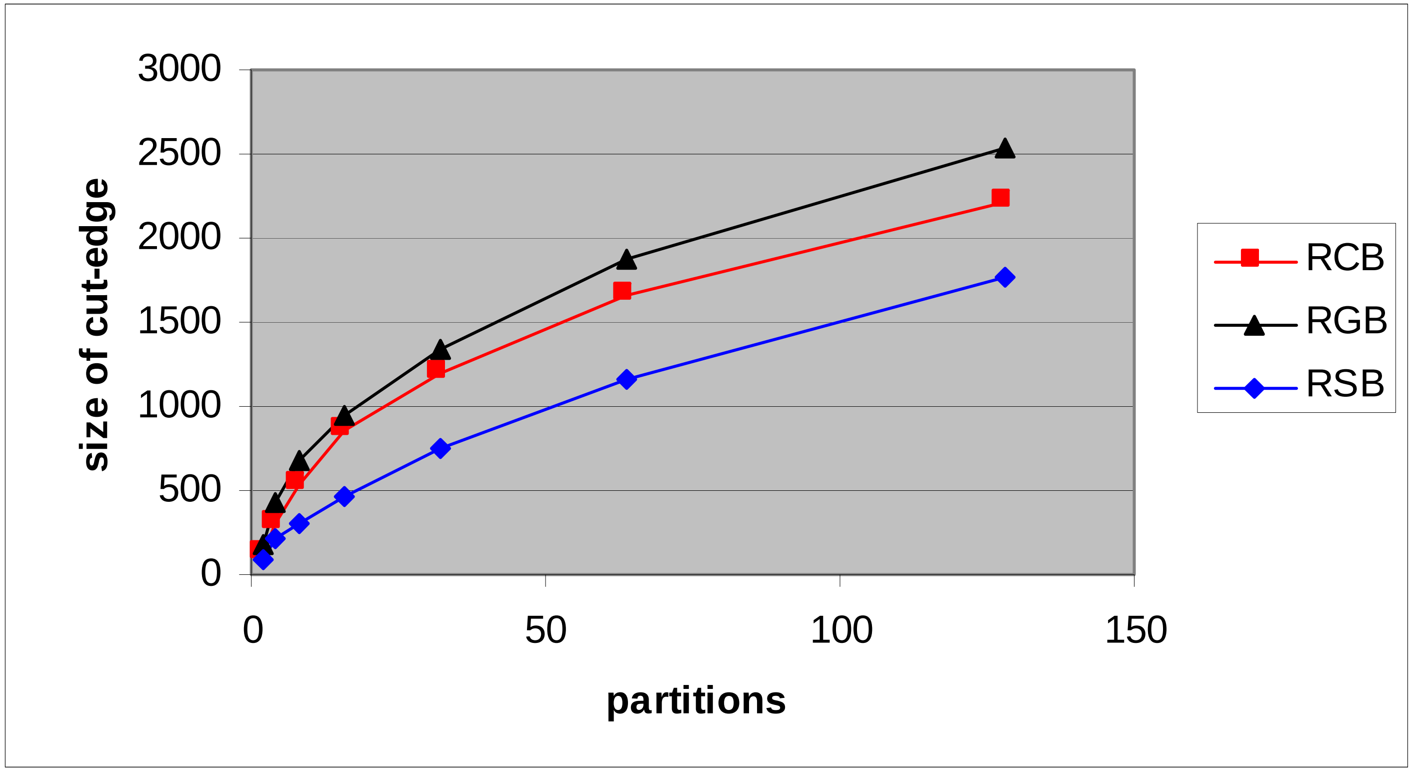
RGB



RSB



Comparison 2



Extended example

- See 3.3 in reader, measuring results.
- Taken from Fox et. Al, Parallel Computing Works, chapter 11.

Problem definition

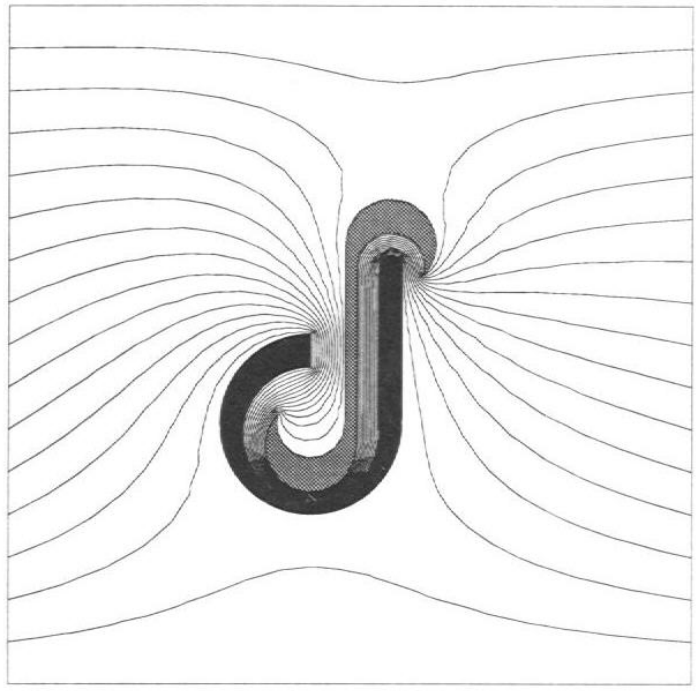


Figure 11.13: Solution of the Laplace Equation Used to Test Load-Balancing Methods. The outer boundary has voltage increasing linearly from −1.2 to 1.2 in the vertical direction, the light shade is voltage 1, and the dark shade voltage −1.

Computational Meshes

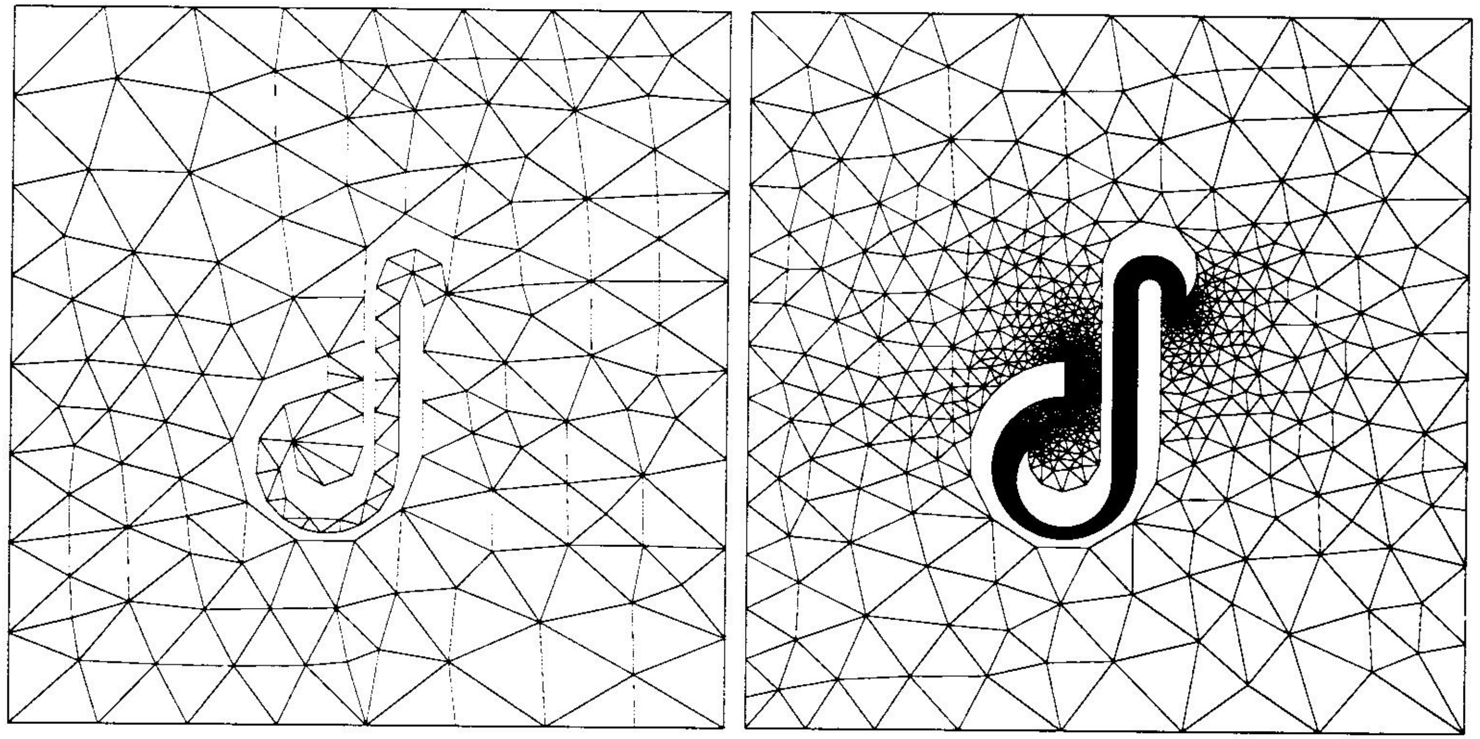


Figure 11.14: Initial and Final Meshes for the Load-Balancing Test. The initial mesh with 280 elements is essentially a uniform meshing of the square, and the final mesh of 5772 elements is dominated by the highly refined S-shaped region in the center.

Decompositions at stage 4 and 5

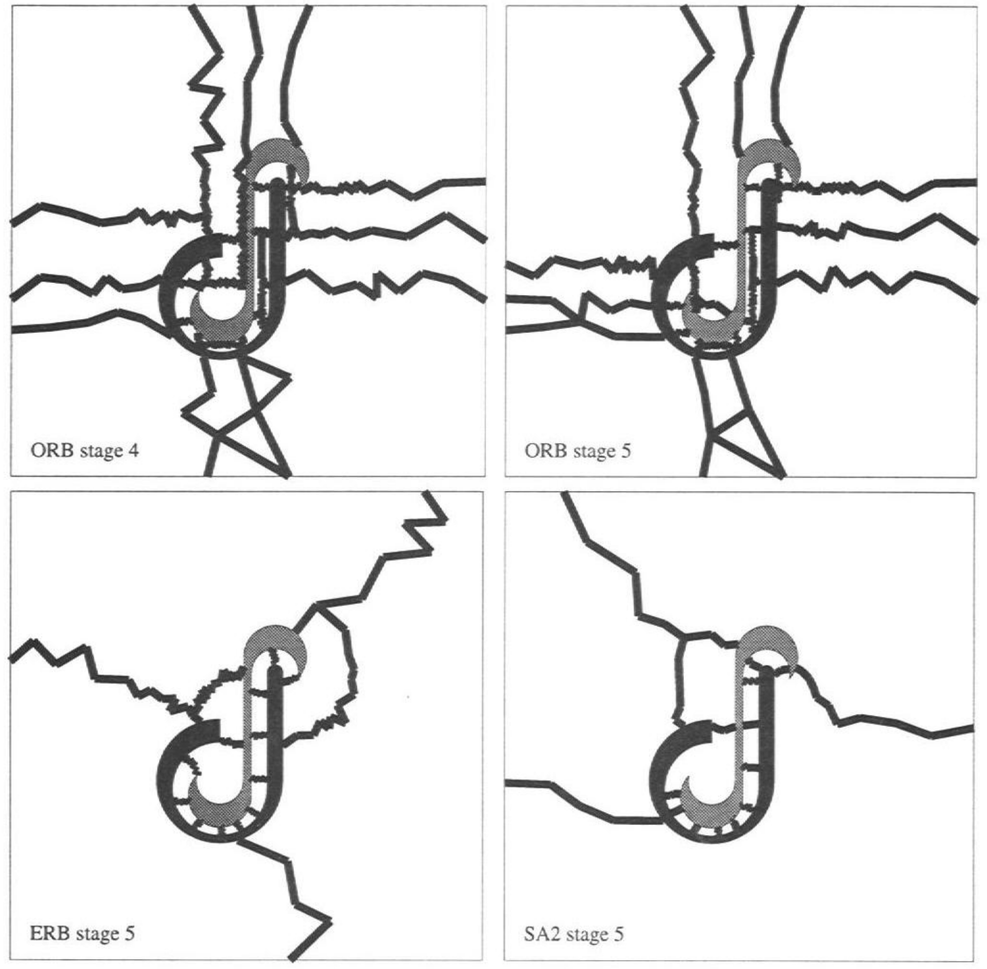


Figure 11.15: Processor Divisions Resulting from the Load-Balancing Algorithms. Top, ORB at the fourth and fifth stages; lower left, ERB at the fifth stage; lower right, SA2 at the fifth stage.

Machine independent measures

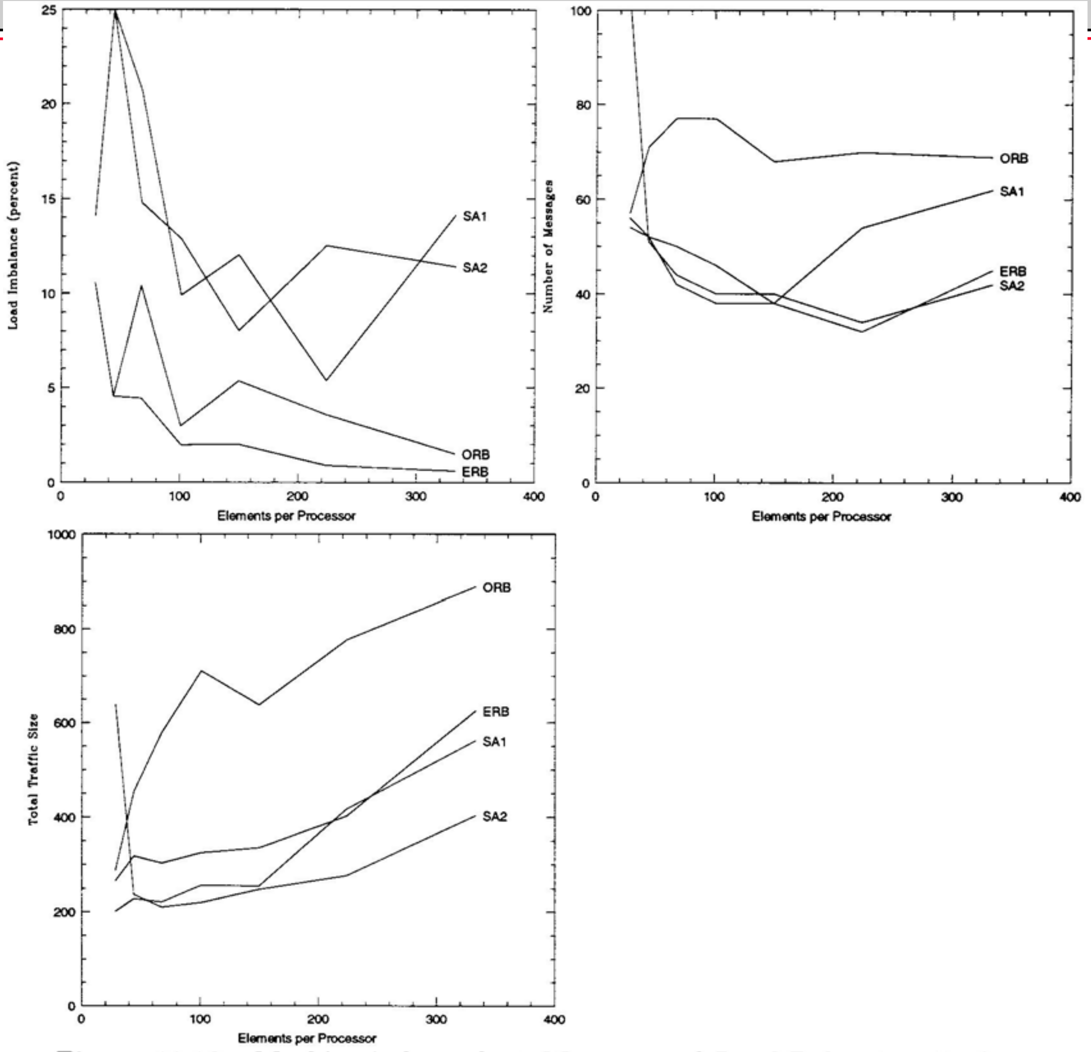


Figure 11.16: Machine-independent Measures of Load-Balancing Performance. Left, percentage load imbalance; lower left, total amount of communication; right, total number of messages.

Machine dependent measures

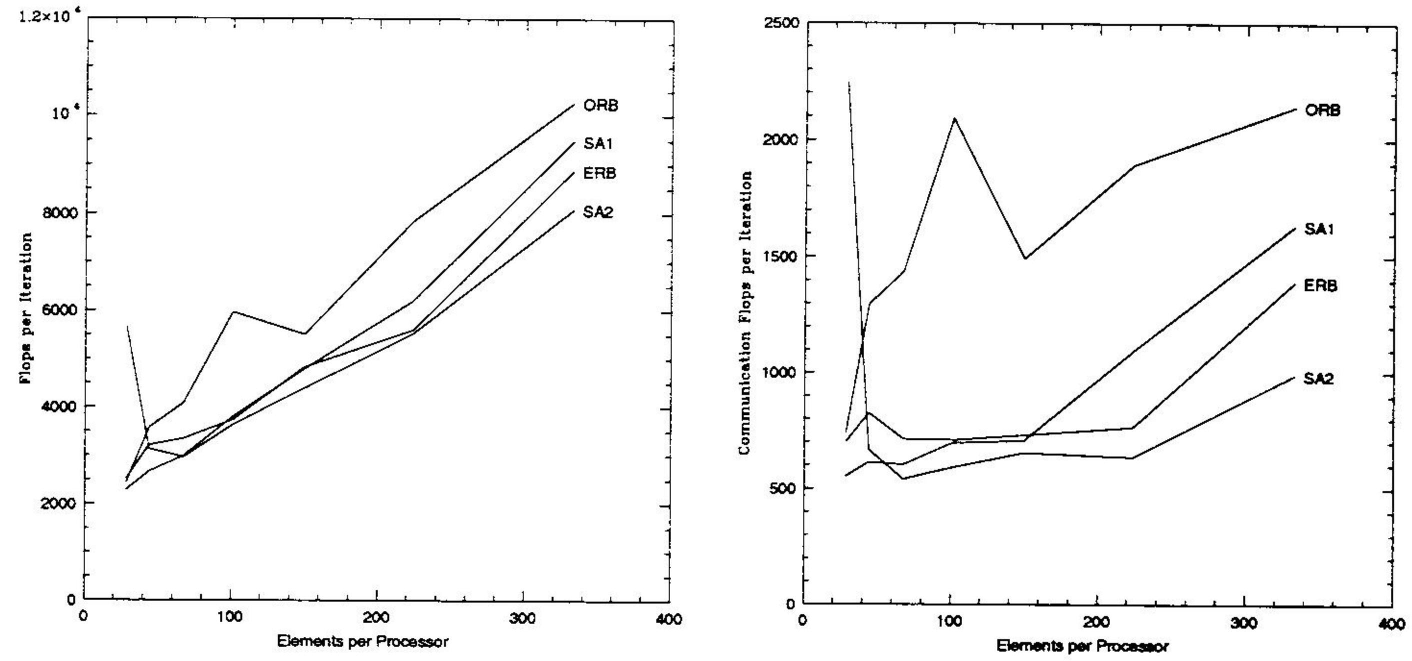


Figure 11.17: Machine-dependent Measures of Load-Balancing Performance. Left, running time per Jacobi iteration in units of the time for a floating-point operation (flop); right, time spent doing local communication in flops.

Dynamic load balancing

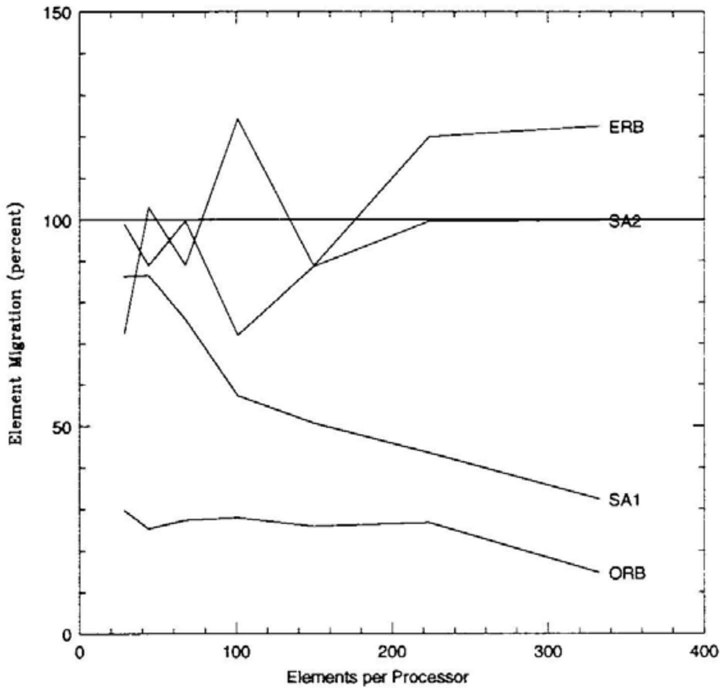


Figure 11.18: Percentage of Elements Migrated During Each Load-Balancing Stage. The percentage may be greater than 100 because the recursive bisection methods may cause the same element to be migrated several times.

More graph partitioning

- Static Graph Partitioning
  - Geometric
  - Combinatorial
  - Spectral
  - Combinatorial optimization
  - Multilevel methods
- Adaptive Computations and Repartitioning
- Parallel Graph Partitioning
- Graph Partitioning Libraries

Static Graph Partitioning 1

- Geometric Techniques
  - only use coordinate information of mesh elements.
  - typically very fast.
  - produce worse results compared to schemes that take connectivity into account.

A. Recursive Coordinate Bisection (RCB)

- a.k.a. Coordinate Nested Dissection (CND).

+ very fast

+ requires little memory

+ easy to parallelize

- low quality partitions

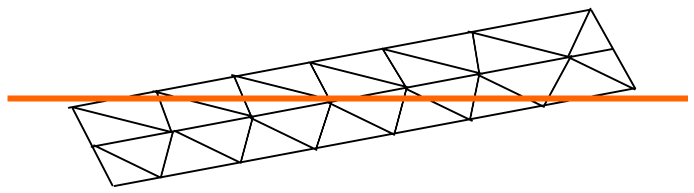
- disconnected subdomains

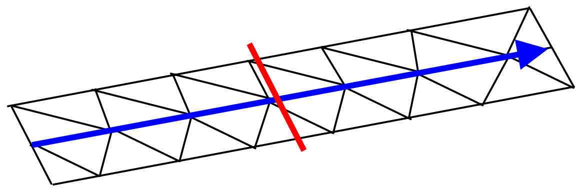
Static Graph Partitioning 2

- Geometric Techniques continued

  B. Recursive Inertial Bisection (RIB)

  - CND only bisects normal to coordinate axes
  - RIB improves this
    - Compute inertial axes of Mesh
    - Project onto this axes, and sort





CND versus RIB

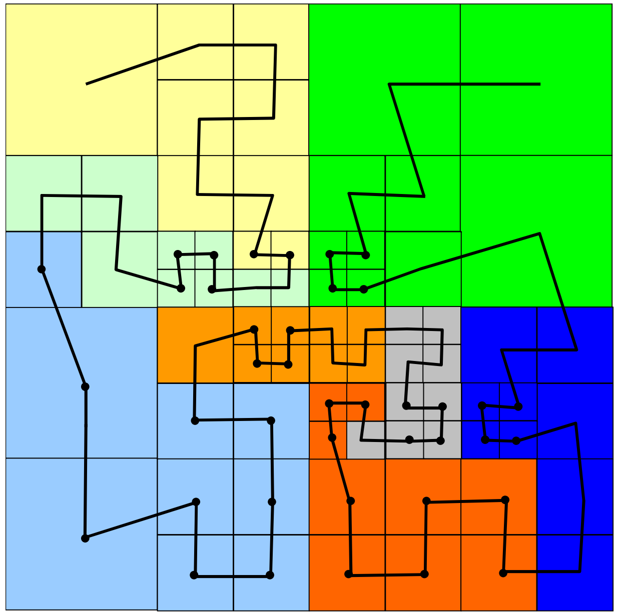
Static Graph Partitioning 3

- Geometric Techniques continued
  - **C**. Space Filling Curves
    - CND and RIB only sort in one dimension
    - Using more dimensions may improve results
    - Order elements along a space filling curve
      - e.g. a Peano-Hilbert curve
    - Do a *k*-way partitioning immediately on this sorted list.

+ very fast

+ typically better than CND and RIB

+ works well for specific applications, e.g. *N*-body simulations.



Static Graph Partitioning 4

- Combinatorial Techniques
  - Attempt to group together highly connected vertices, regardless if their coordinates are close together or far apart.
  - So, only based on adjacency information of the graph.

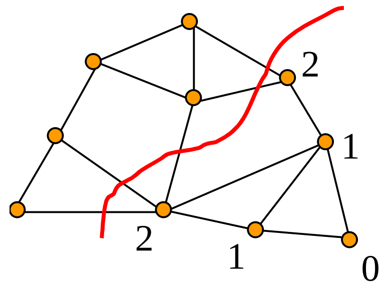
A. Levelized Nested Dissection (LND)

- idea is to put vertices together, by starting with a single vertex and then grow incrementally with other vertices that are connected to it.

Static Graph Partitioning 5

A. Levelized Nested Dissection (LND) continued

– The algorithm is as follows
  - Select a single vertex, assign number 0.
  - All vertices adjacent to 0 (i.e. connected to 0) are assigned number 1.
  - Repeat this now for number 1, numbering all non-numbered vertices with increasing numbers.
  - Terminate if half of the vertices have been assigned a number.
  - The numbered vertices go in one domain, the remaining in the other.



– LND tends to perform better if initial vertex is one of a pair with largest distance on the graph.
  - In that case LND is equal to Recursive Graph Bisection.

– At least one of the domains is connected.

– Produces comparable or better partitioning than geometric schemes.

– Usually, multiple LND runs are performed, using different initial vertices, and the best partitioning is chosen.

Static Graph Partitioning 6

- Combinatorial Techniques

  B. Kernigham-Lin / Fiduccia-Mattheyes Algorithm (KL/FM)

  - partition refinement
    - i.e. given an initial partition, try to improve it as good as possible.
    - Given a partition A, B find two equally sized subsets $x \in A$ and $y \in B$ such that swapping x to B and y to A gives the best possible reduction of the edge-cut.
  - KL/FM are greedy algorithms to do this.
  - KL is the original algorithm, FM is a refinement that gives the same results but is much faster.

Static Graph Partitioning 7

- Combinatorial Techniques

  B. Kernigham-Lin / Fiduccia-Mattheyes Algorithm (KL/FM)

  - The KL algorithm

    start with initial partition

    in one pass do the following

      pick one vertex from one sub domain, and form pairs of vertices by picking vertices from other sub domains

      The pair that gives the best improvement to the edge-cut (or, the smallest increase in edge-cut, i.e. jump out of local minimum) is chosen and the vertices are swapped.

      This is repeated for the remaining vertices, where once moved vertices are not considered anymore in forming pairs.

      When all vertices have been moved, the pass ends.

      At this point, the state that, during the pass, had the smallest edge-cut is chosen and restored.

    Several passes are performed, usually the algorithm takes just a few passes to converge.

  - Each pass of KL takes O(|V|^2) operations.

Static Graph Partitioning 8

- Combinatorial Techniques

  B. Kernigham-Lin / Fiduccia-Mattheyes Algorithm (KL/FM)

  - The FM algorithm
    - – a modification of the KL algorithm
      - improves run time without decreasing the effectiveness (at least for FE grids in scientific computing).
    - – Only use single vertices to move. Calculate for each vertex in a domain the gain (or loss) in quality if moved. Then create priority queues for both domains, and by handling these priority queues efficiently, FM is able to get good refinement (comparable to KL) in O(|E|), which is a huge improvement.

Static Graph Partitioning 9

- Spectral Techniques

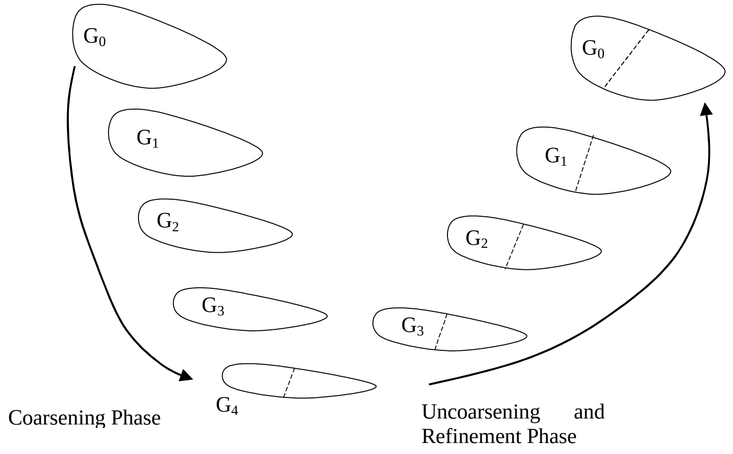
  Recursive Spectral Bisection (RSB)

  + High quality partitioning

  - calculating Fiedler vector is very expensive

Static Graph Partitioning 10

- Multi-Level Schemes
  - Three phases
    - (1) Graph Coarsening
      - Construct series of graphs by collapsing together selected vertices.
      - Repeat a few times, until a sufficiently small graph is obtained.
    - (2) Initial Partitioning
      - On the coarsest graph, any fast partitioning method will do.
    - (3) Multilevel refinement
      - On each level, a partitioning refinement using e.g. KL/FM type algorithms is applied.



Static Graph Partitioning 11

- Multi-Level Schemes

A. Multilevel Recursive Bisection

- Works very well
  - KL/FM become very powerful in the multilevel context.
- Robust, outperfoms RSB in quality of partitions, and is much faster.
- Choice of initial partitioning on the coarsest level does hardly have an effect on the final result.

B. Multilevel k-way Partitioning

- Using k-way refinement schemes (i.e. generalizations of KL/FM).
- Comparable or even better partitioning than Multilevel Recursive Bisection.
- Very fast, typically 2 orders of magnitude faster than spectral methods.

Static Graph Partitioning 12

	Needs Coordinates	Quality	Local View	Global view	Run Time	Degree of parallelism
Recursive Spectral Bisection	no	●●●●	O	●●●●	♦ ♦ ♦ ♦	■■
Multilevel Partitioning	no	●●●●●●	●●	●●●●	♦ ♦	■■
Levelized Nested Dissection	no	●●	O	●●	♦ ♦	■■
Coordinate Nested Dissection	yes	●	O	●	♦	■■■
Recursive Inertial Bisection	yes	●●	O	●●	♦	■■■

Remember Load Balancing Strategies

1. "Easy" case

   - by inspection (e.g. "trivial" cubical domains)

   - Static - non-trivial, but can be done before parallel job is started.

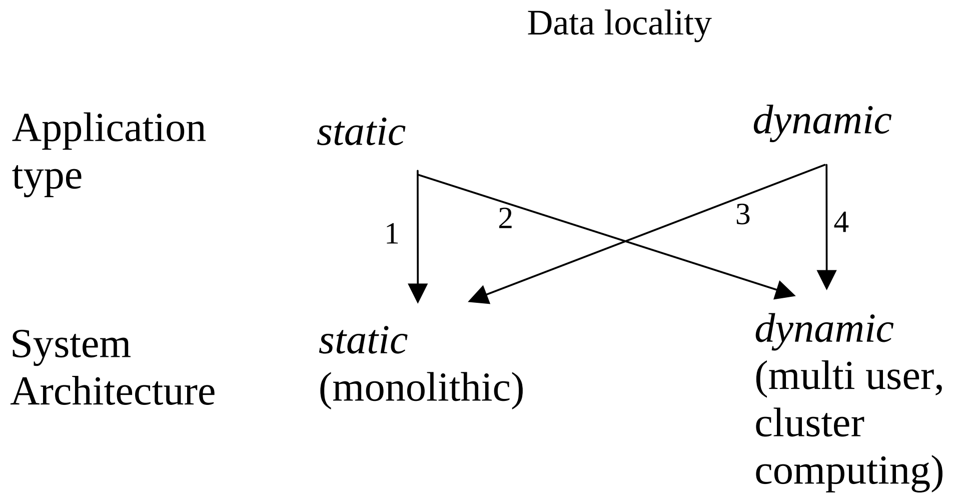
2. Load balancing by the run time support system -> e.g. Dynamic PVM

3. Need for rebalancing the application

   - Quasi dynamic

   - Dynamic

4. Rebalancing the application + support from run time system (very hard case !)



We now focus on case 3, rebalancing the application

Adaptive Computations and Repartitioning

- Adaptive computations
  - – e.g. through adaptive grid refinements or changing workload in elements during the simulation.
  - – Causes load imbalance.
- Adaptive Graph Partitioning
  - – same requirements as static graph partitioning +
  - – amount of data that must be redistributed among processors should be minimized.
    - So, not only the computational weight of a vertex is a parameter, but also it’s data size.

Repartitioning Approaches

- Scratch-Remap Repartitioning
  - – Just forget the previous partitioning and do it again from scratch.
    - remember the SA with high temperature in the extended example.
  - – Loss of information, and need for enormous amount of data to be redistributed.
- Diffusion Based Schemes
  - – incremental changes in partitioning, by moving (diffusing) vertices to neighboring processors.

Parallel Graph Partitioning

- Reasons to do partitioning in parallel
  - memory constrains due to very large meshes
    - just don’t fit in memory of a single processor
  - the number of available processors (which dictates the number of partitions) may only be known at runtime.
    - Especially true in grid-computing environments.
  - For adaptive repartitioning, the application is already running on a parallel system.
- No details in this lecture
  - many parallel versions of partitioning schemes are available
    - in partitioning libraries !

Libraries for (parallel) Graph Partitioning

- Many libraries are available
  - Chaco
    - http://www.cs.sandia.gov/CRF/chac.html
  - JOSTLE
    - http://www.gre.ac.uk/~jjg01/
  - MeTiS and ParMeTiS
    - http://www-users.cs.umn.edu/~karypis/metis/index.html
    - http://www-users.cs.umn.edu/~karypis/metis/parmetis/index.html
  - PARTY
    - http://www.unipaderborn.de/fachbereich/AG/monien/RESEARCH/PART/party.html
  - SCOTCH
    - http://dept-info.labri.u-bordeaux.fr/~pelegrin/scotch/
  - S-HARP
    - http://www.cs.njit.edu/sohn/sharp/index.html

Comparison of libraries
static partitioners

	Chaco	Jostle	Metis	ParMetis	Party	Scotch	S-Harp
Geometric Schemes	●			●	●		●
Coordinate Nested Dissection					●		
Recursive Inertial Dissection	●						●
Space-filling Curve Methods				●			
Spectral Methods	●				●		●
Recursive Spectral Bisection	●						
Multilevel Spectral Bisection	●						
Combinatorial Schemes	●				●	●	
Levelized Nester Dissection					●	●	
KL/FM	●				●	●	
Multilevel Schemes	●	●	●	●	●	●	
Multilevel Recursive Bisection	●		●			●	
Multilevel k-way Partitioning		●	●	●			
Multilevel fill-reducing ordening			●	●			

Comparison of libraries repartitioners, parallel and others

	Chaco	Jostle	Metis	ParMetis	Party	Scotch	S-Harp
Dynamic Repartitioners		●		●			●
Diffusive Repartitioning		●		●	●		●
Scratch-remap Repartitioning				●			
Parallel Graph Partitioners		●		●			●
Parallel Static Partitioning		●		●			●
Parallel Dynamic Partitioning		●		●			●
Other Formulations		●	●	●			
Multi-constraint Graph Partitioning		●	●	●			
Multi-objective Graph Partitioning			●				