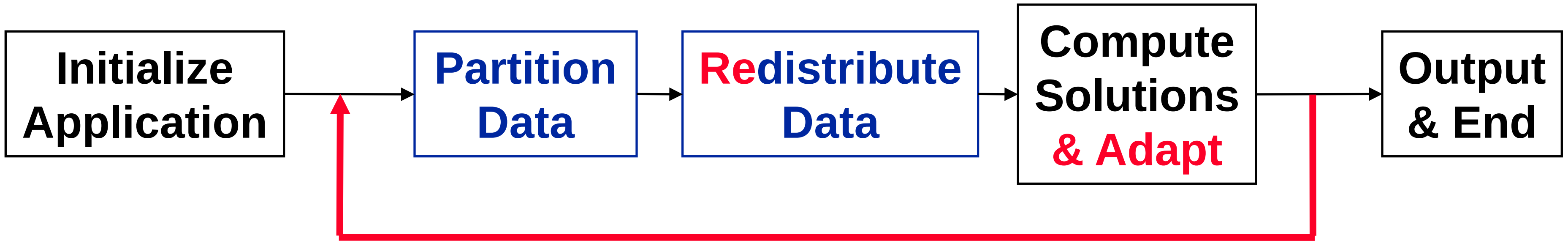
LOAD BALANCING

- Programs and algorithms as graphs
- Geometric Partitioning
- Graph Partitioning
  - Recursive Graph Bisection partitioning
  - Recursive Spectral Bisection
  - Multilevel Graph partitioning
- Hypergraph Partitioning
- Space Filling Curves and Assimilation Based Load Balancing

Requirements of Load Balancing Algorithm

- Balanced work loads.
  - Must minimize largest imbalance when attempting to provide each processor with an equal share of work.
- Must have low interprocessor communication costs.
  - Partitions with minimal communication costs are critical.
- Scalable partitioning time and memory use.
  - Scalability is especially important for frequent dynamic partitioning.
- Low data redistribution costs for dynamic partitioning.

Dynamic Load Balancing



Dynamic repartitioning (load balancing) in applications

- Data partition is computed.
- Data are distributed according to partition map.
- Application computes and, perhaps, adapts.
- Process repeats until the application is done.

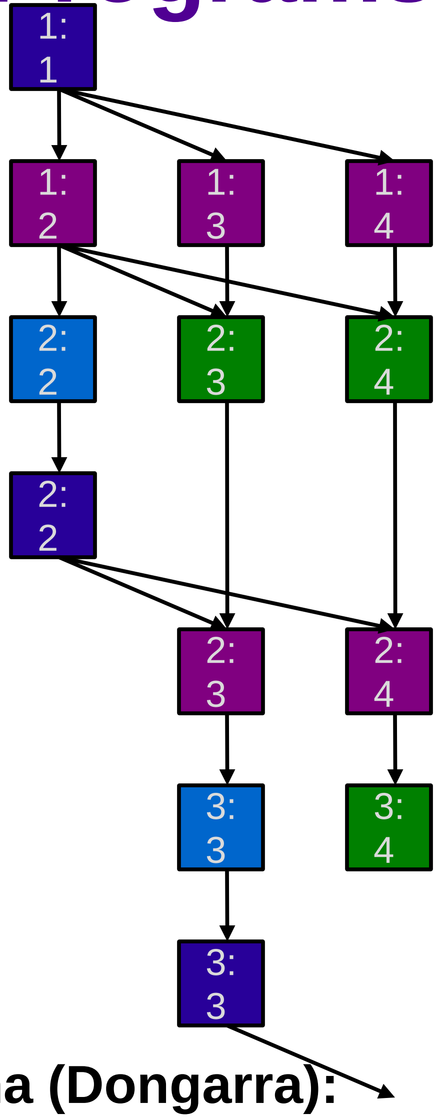
Ideal partition:

- Processor idle time is minimized.
- Inter-processor communication costs are kept low.
- Cost to redistribute data is also kept low.

Programs as Graphs

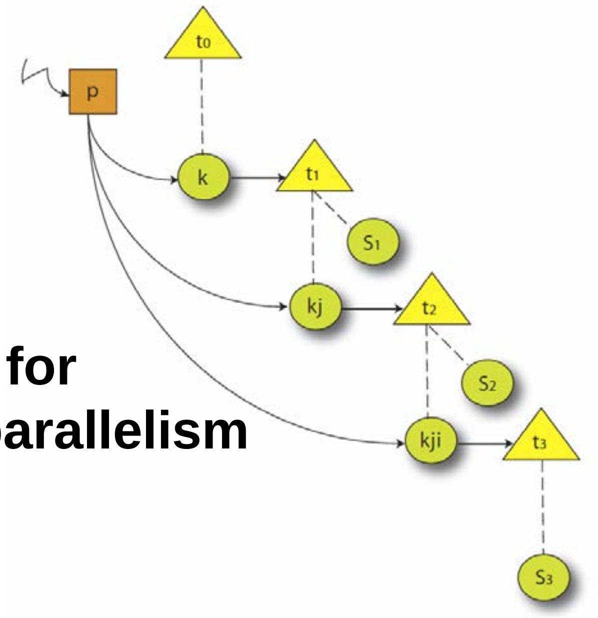
- Individual lines of code may be expressed as graphs
  - y = ax +b ; z = cy/e; is a graph connecting a,b,c,e,x,y and z the nodes of the graph are the variables and the edges represent calculations.
- The connections between functions (or different programs) may be represented as a graph.
- Partitioning a graph is thus on approach to load balancing. This is NP-hard (Non-deterministic polynomial time hard) and so we have to use heuristics to solve approximate problems close to the actual problem. (no general solution in polynomial time)

Graph Based View of Programs



Linear Algebra Applications are now written as Directed Acyclic Graphs

Some new programming languages are based areound the idea of expressing the program as a graph



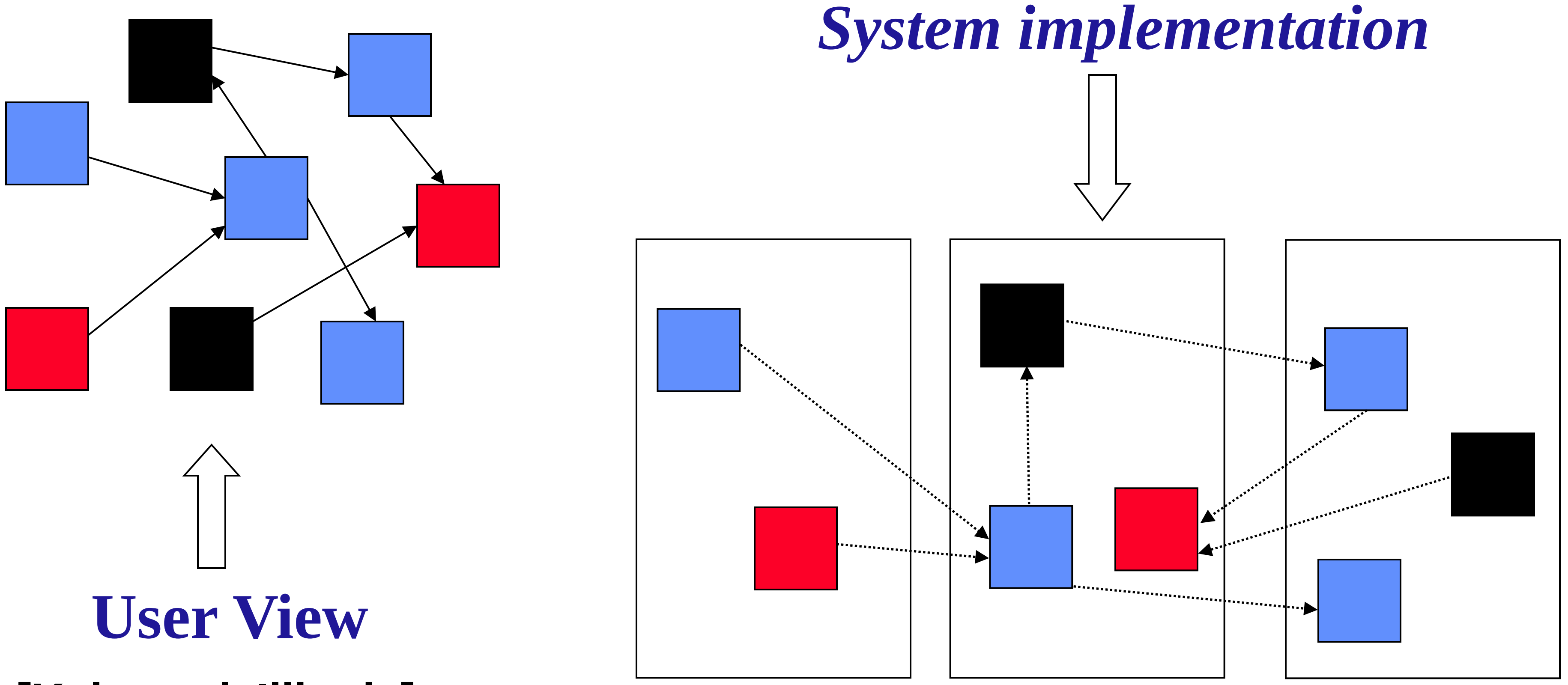
Intel CnC: new language for graph based parallelism

Plasma (Dongarra): DAG based Parallel linear algebra software

Charm ++ Virtualization: Object-based Parallelization

- **Idea: Divide the computation into a large number of objects**
  - –**Let the *system* map objects to processors**

User is only concerned with interaction between objects



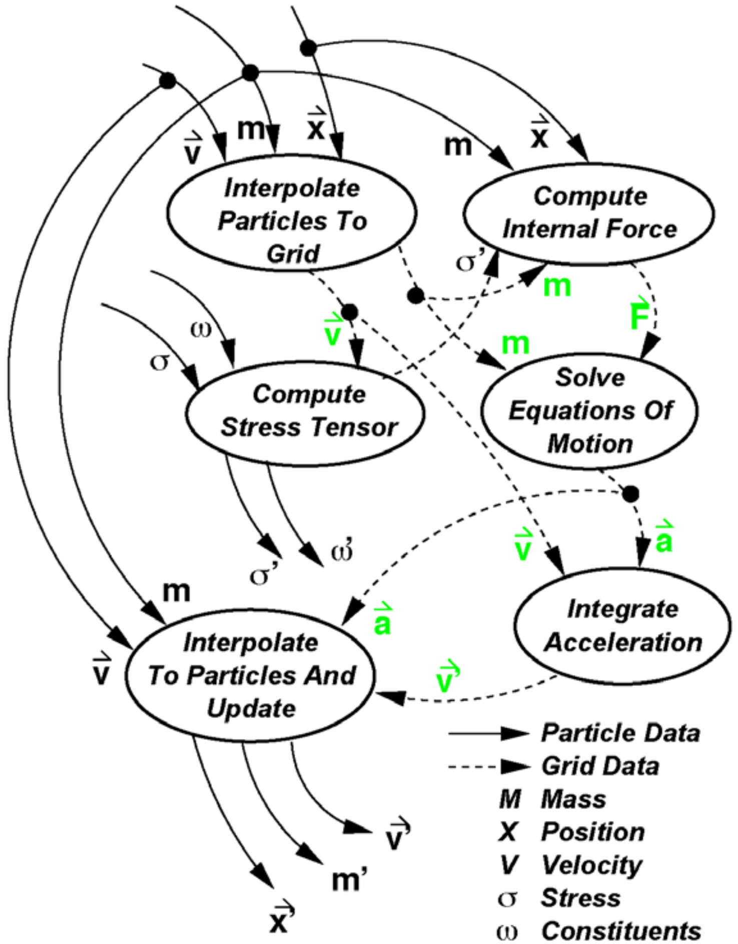
[Kale et al. Illinois]

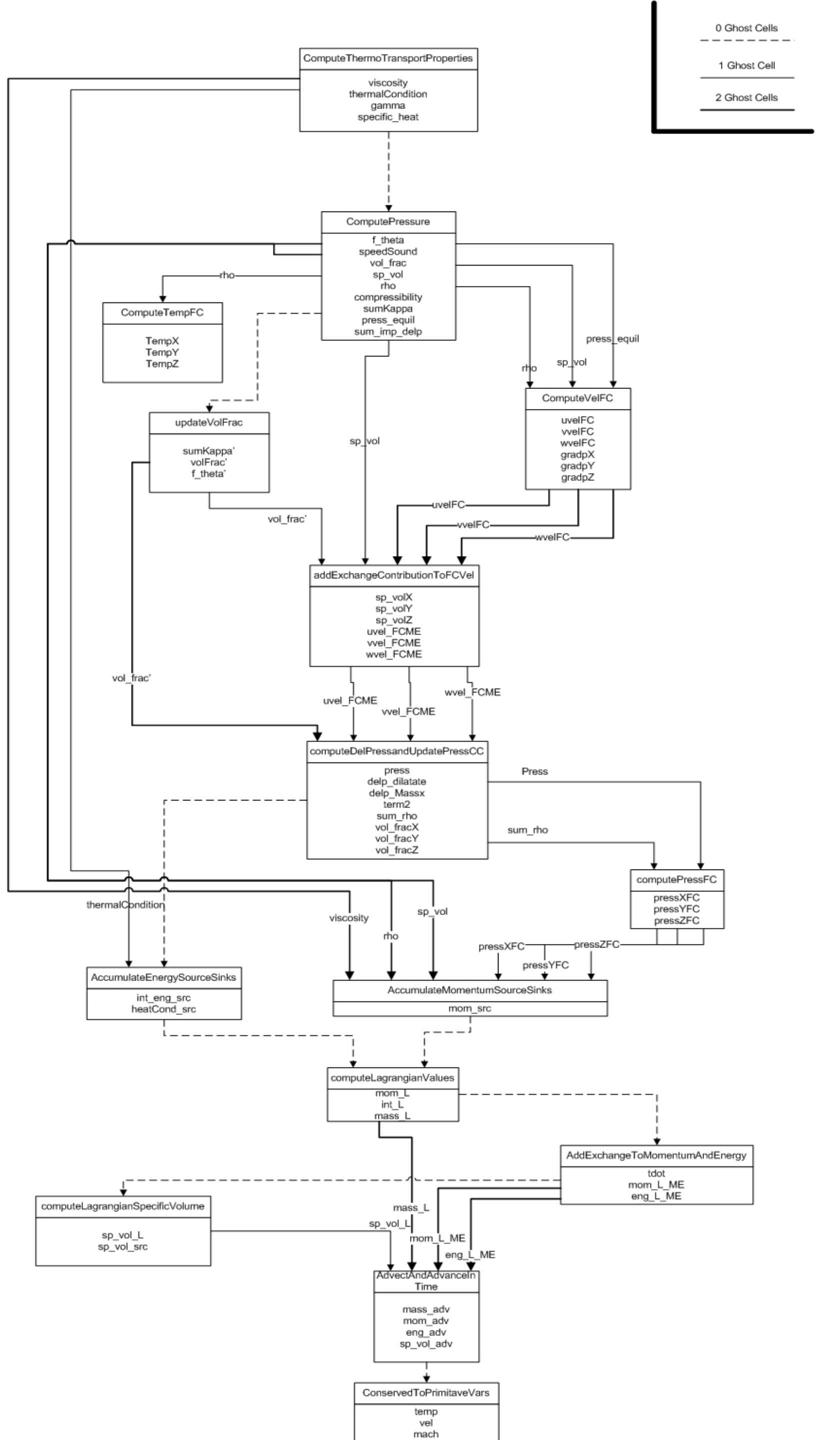
Code is expressed as Directed Acyclic Graph DAG

Uintah Task Graphs

Fluid-flow solver

Particle method

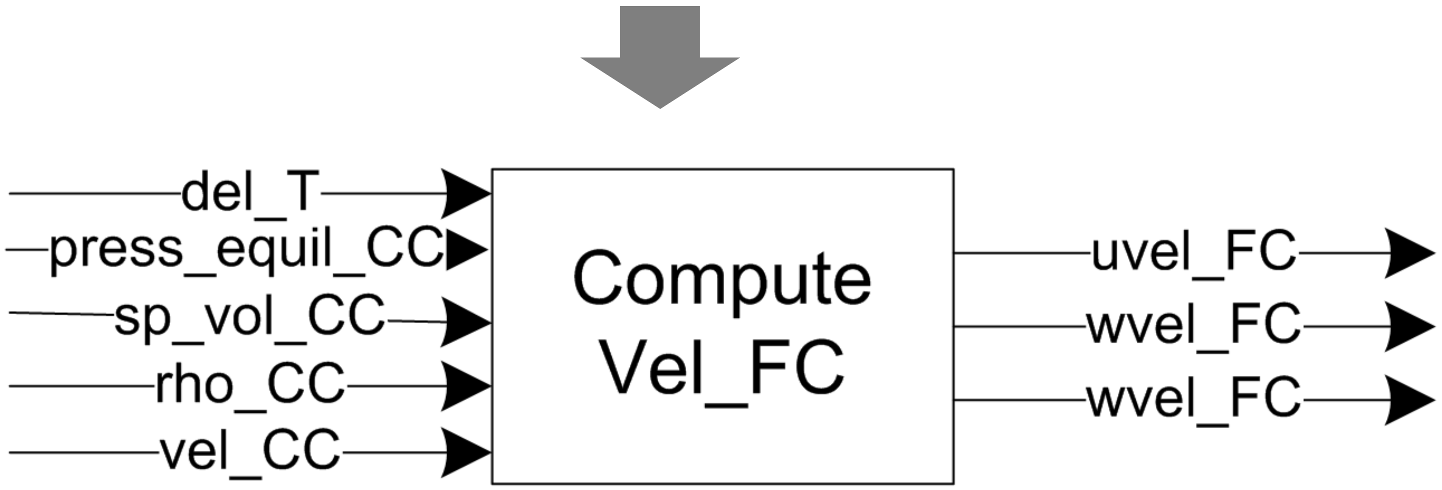




Example Uintah Task from the ICE Algorithm

Compute face-centered Velocities:

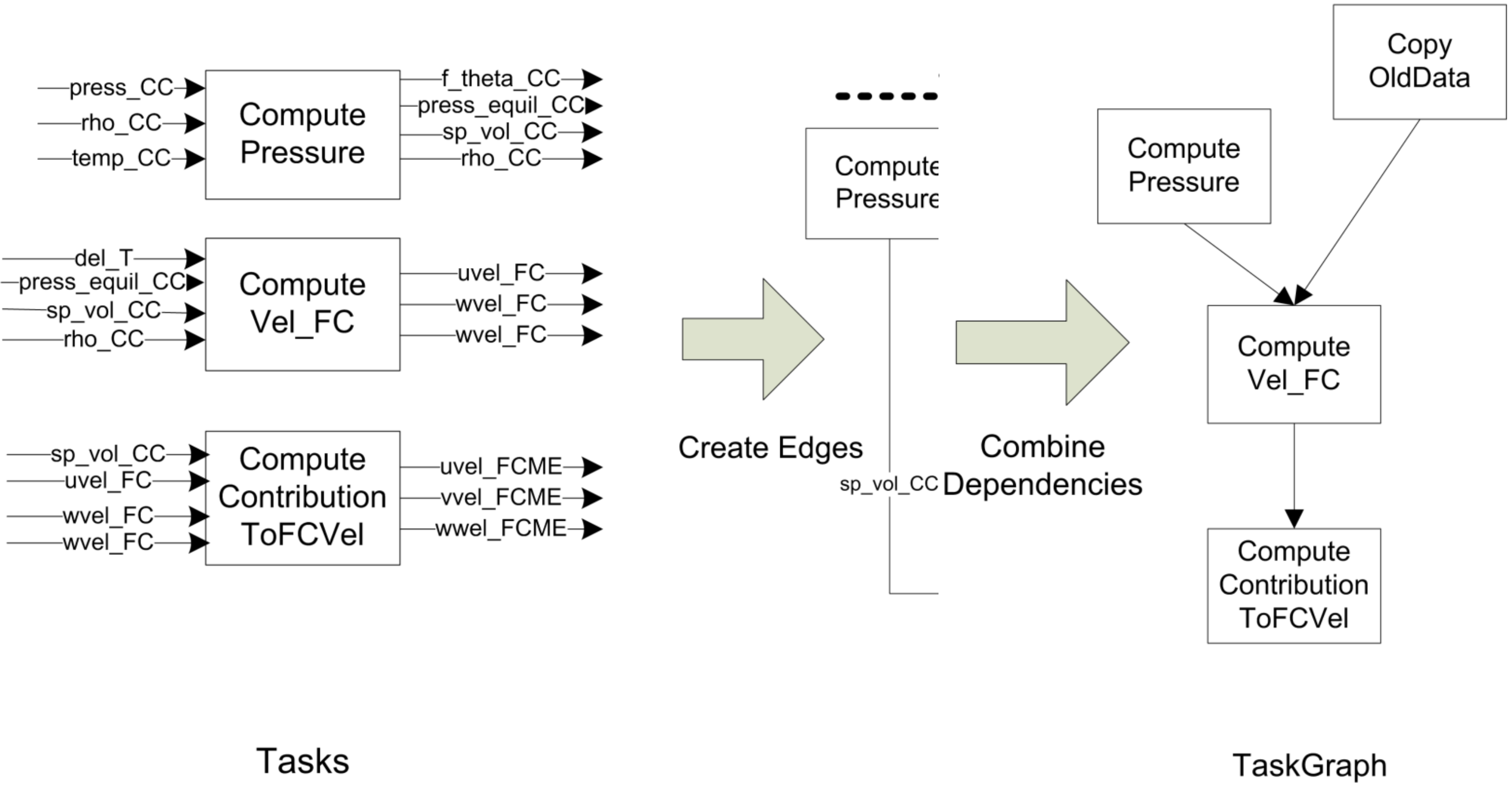
$$\vec{U^*}^f = f(\Delta t, P_{eq}, \vec{g}, \rho, \vec{U})$$



Input variables
(include boundary conditions)

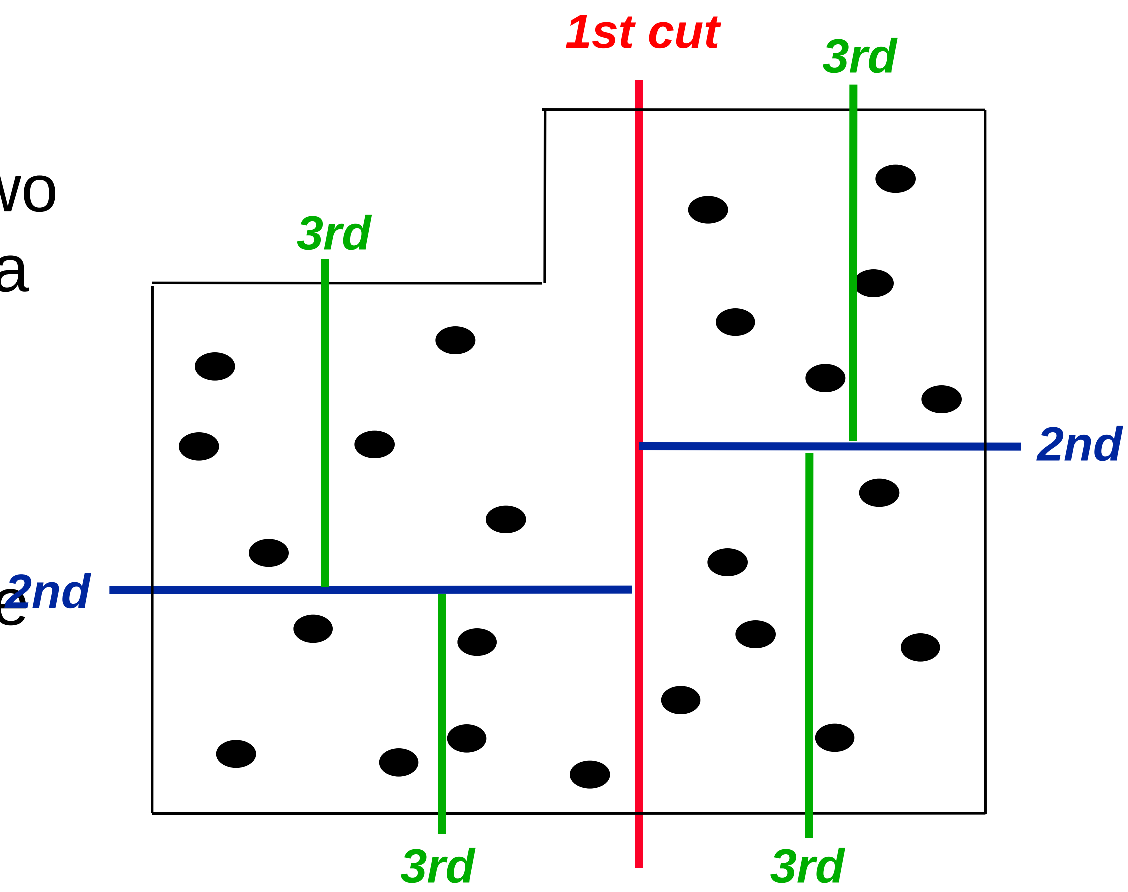
Output variables

Task Graph for a Real Application



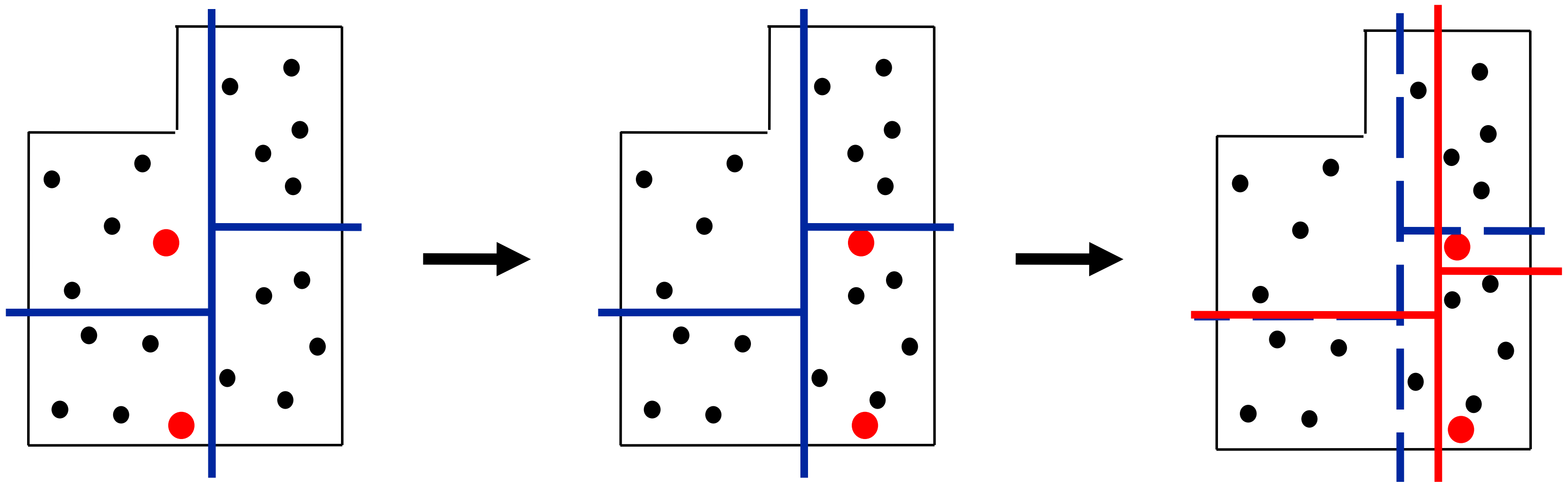
Geometric Partitioning

- **Recursive Coordinate Bisection**: Developed by Berger & Bokhari (1987) for Adaptive Mesh Refinement.
- Idea:
  - Divide work into two equal parts using a cutting plane orthogonal to a coordinate axis.
  - Recursively cut the resulting subdomains.



Geometric Repartitioning

- Implicitly achieves low data redistribution costs.
- For small changes in data, cuts move only slightly, resulting in little data redistribution.

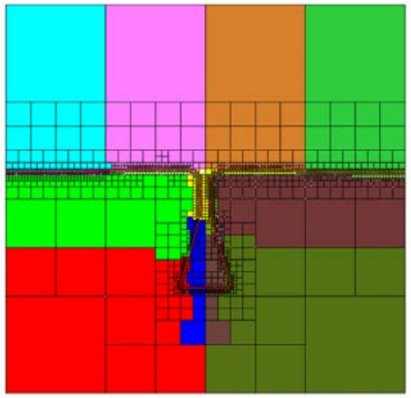


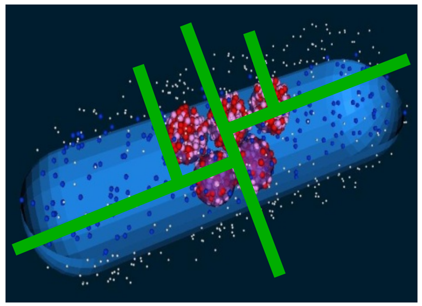
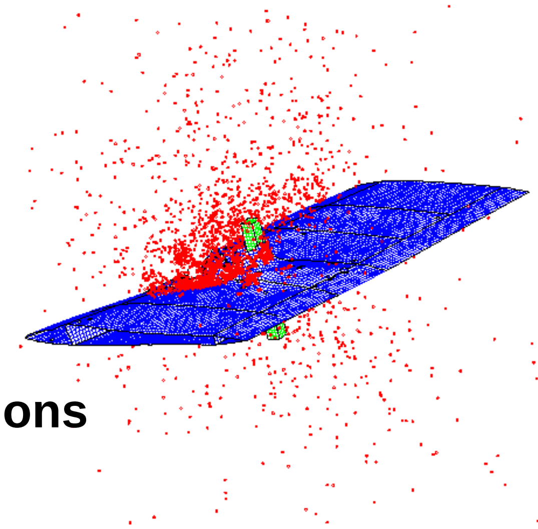
RCB Advantages and Disadvantages

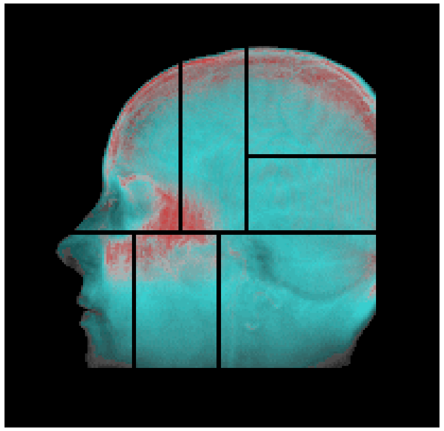
- Advantages:
  - Conceptually simple; fast and inexpensive.
  - All processors can inexpensively know entire partition (e.g., for global search in contact detection).
  - No connectivity info needed (e.g., particle methods).
  - Good on specialized geometries.

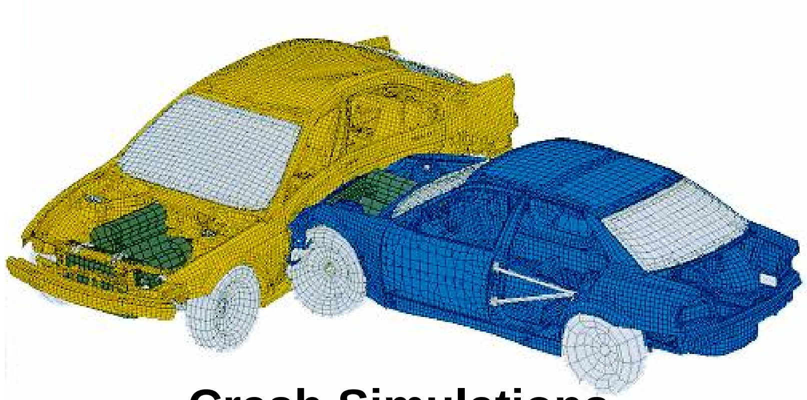
Disadvantages:
  - No explicit control of communication costs.
  - Mediocre partition quality.
  - Can generate disconnected subdomains for complex geometries.
  - Need coordinate information.

Applications of Geometric Methods


Adaptive Mesh Refinement



Particle Simulations

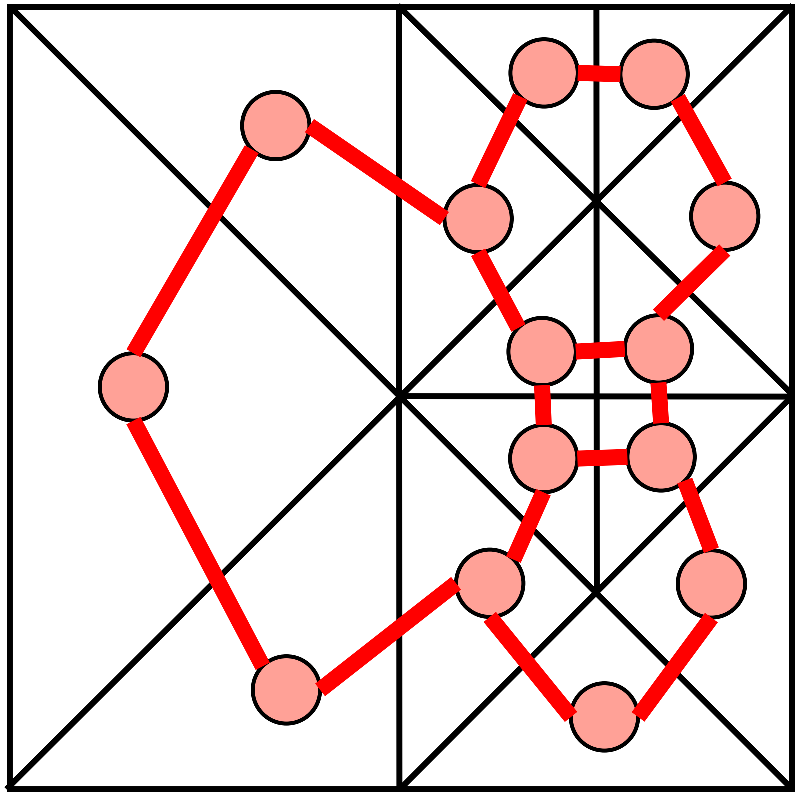

Parallel Volume Rendering


Crash Simulations
and Contact Detection

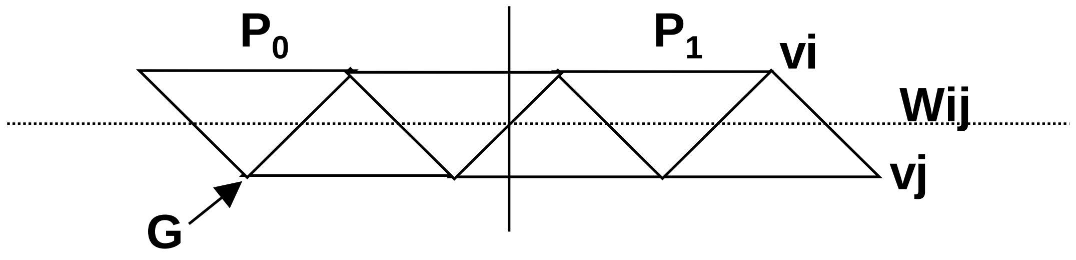
Graph Partitioning

- Kernighan, Lin, Schweikert, Fiduccia, Mattheyes, Simon, Hendrickson, Leland, Kumar, Karypis, et al.

- Represent problem as a weighted graph.
  - Vertices = objects to be partitioned.
  - Edges = dependencies between two objects.
  - Weights = work load or amount of dependency.
- Partition graph so that …
  - Parts have equal vertex weight.
  - Weight of edges cut by part boundaries is small.



Graph partitioning is the main approach



N vertices represent computation

m edges represent data dependencies

Partition into P sets such that:

- Equal # vertices per set
- Low induced communication

(more generally, weighted graphs …

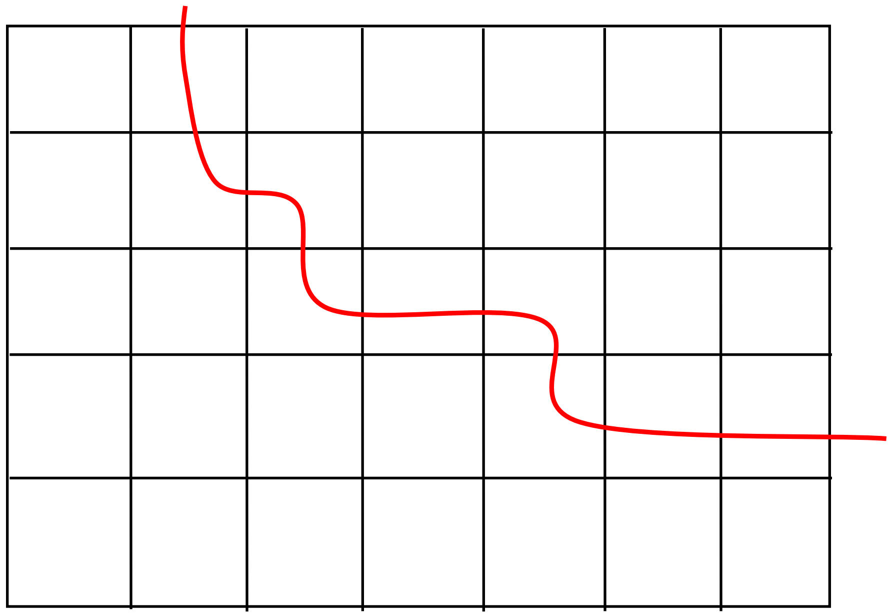
But how to model communication?

Mistake: using just edge cuts

- Generally believed that “Edge Cuts = Communication Cost”.
- This assumption is behind the use of graph partitioning.
- In reality:
  - Edge cuts are not equal to communication volume.
  - Communication volume is not equal to communication cost.

Communication volume

- Assume graph edges reflect data dependencies.
- Correct accounting of communication volume is:
  - Number of vertices on boundary of partition.



Edge cuts = 10.

Communication volume:

8 (from left partition to right)

7 (from right partition to left)

Why does graph partitioning Work?

Vast majority of applications are computational meshes.

- Geometric properties ensure that good partitions exist.
  - Communication/Computation = $n^{1/2}$ in 2D, $n^{2/3}$ in 3D.
  - Runtime is dominated by computation.

- Vertices have bounded numbers of neighbors.
  - Error in edge cut metric is bounded.
- Homogeneity ensures all processors have similar subdomains.
  - No processor has dramatically more communication.

Other applications aren't always so forgiving.

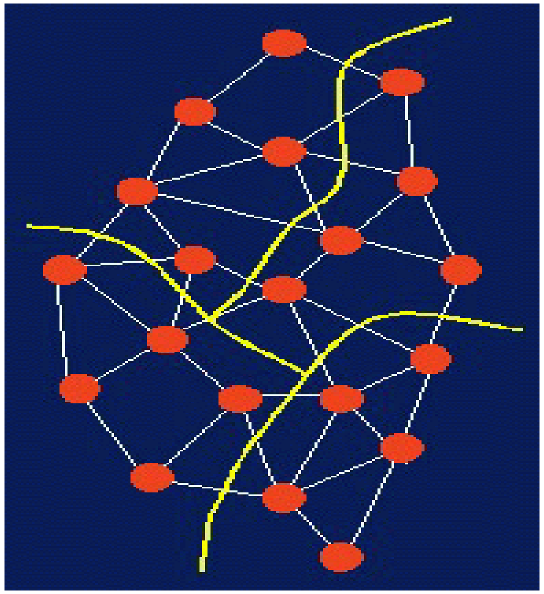
There is a good degree of luck involved!

Communication cost

- Cost of single message involves volume and latency.
- Cost of multiple messages involves congestion.
- Cost within application depends only on slowest processor.
- The model doesn’t optimize the right metrics

Graph Partitioning: Advantages and Disadvantages

- Advantages:
  - Highly successful model for mesh-based PDE problems.
  - Explicit control of communication volume gives higher partition quality than geometric methods.
  - Excellent software available.
    - Serial: Chaco (SNL)
      Jostle (U. Greenwich)
      METIS (U. Minn.)
      Scotch (U. Bordeaux)
    - Parallel: Zoltan (SNL)
      ParMETIS (U. Minn.)
      PJostle (U. Greenwich)
      PT-Scotch (U. Bordeaux)



- Disadvantages:
  - More expensive than geometric methods.
  - Edge-cut model only approximates communication volume.

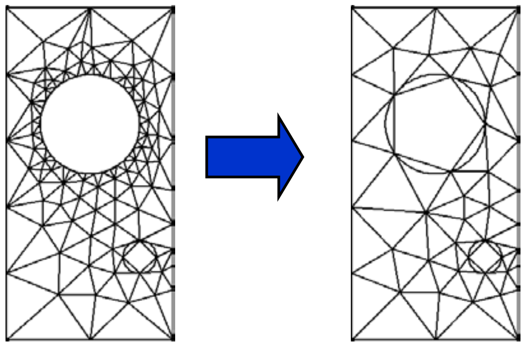
Multilevel KLFM

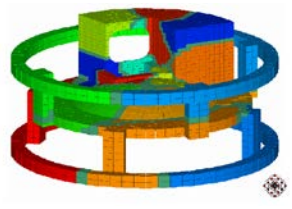
Kernighan-Lin / Fiduccia-Mathews

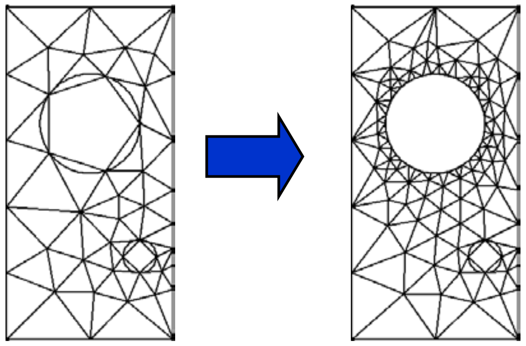
- Until graph is small enough:
  - coarsen graph

  Partition graph:

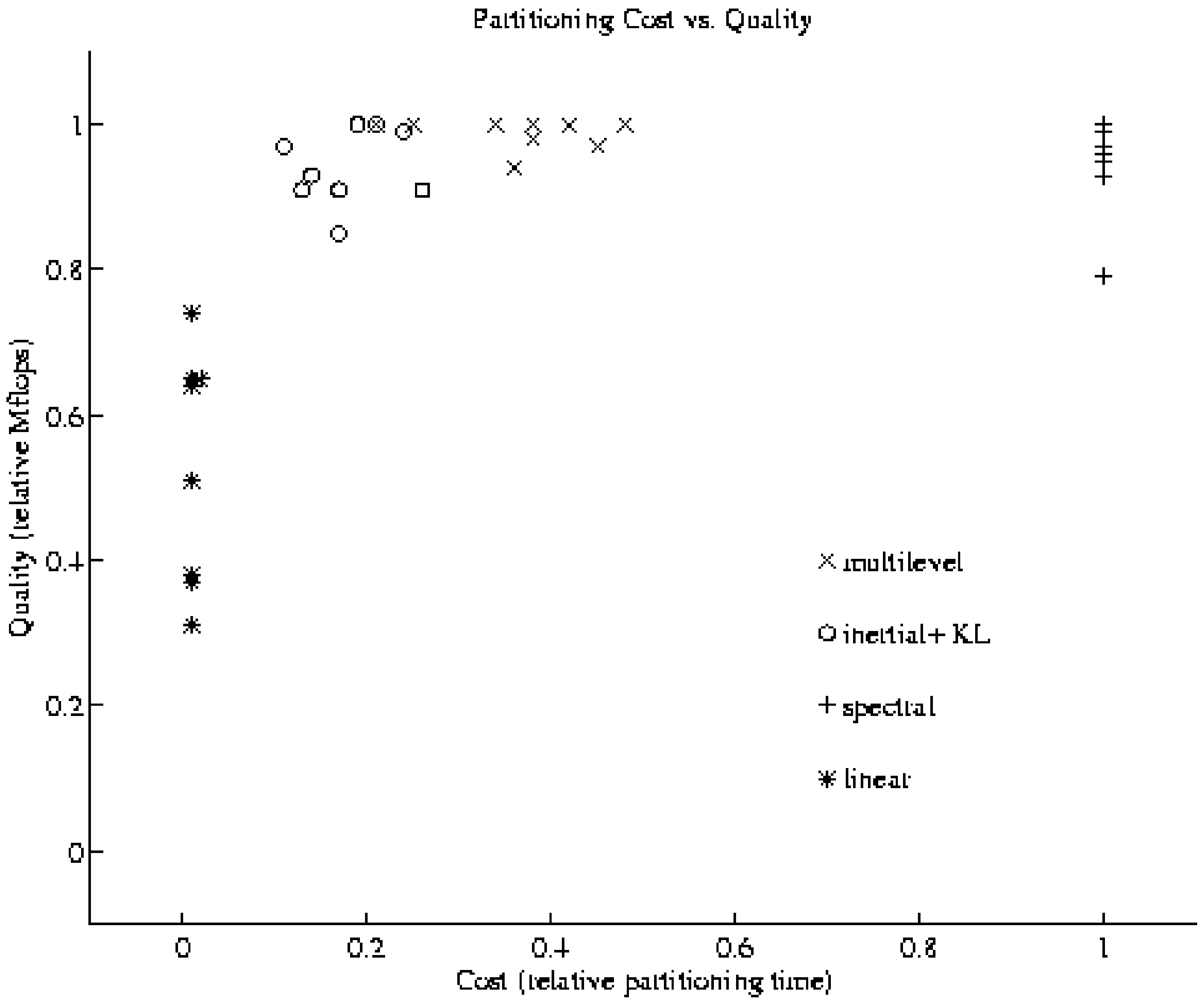
- Until graph = original graph:
  - uncoarsen graph
  - uncoarsen partition
  - locally refine partition





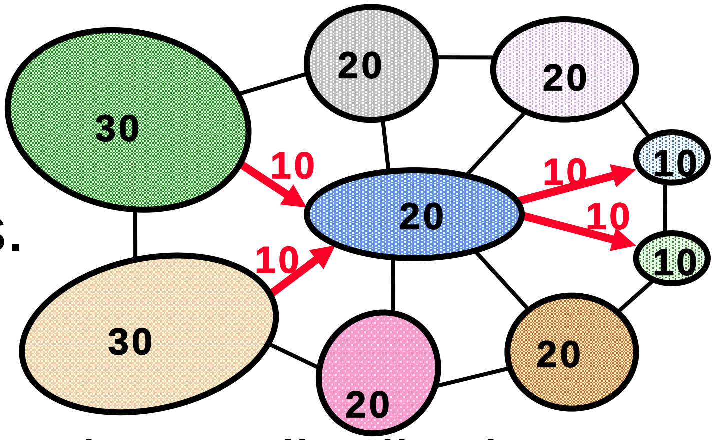


Performance of static partitioners

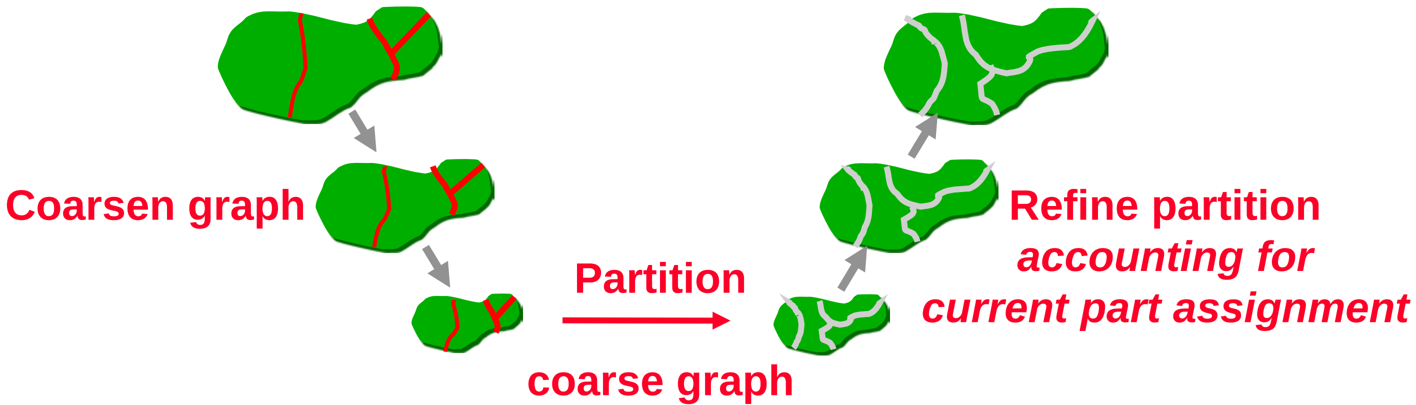


Graph Repartitioning

- Diffusive strategies (Cybenko, Hu, Blake, Walshaw, Schloegel, et al.)
  - Shift work from highly loaded processors to less loaded neighbors.
  - Local communication keeps data redistribution costs low.



  - Multilevel partitioners that account for data redistribution costs in refining partitions (Schloegel, Karypis)
  - Parameter weights application communication vs. redistribution communication.



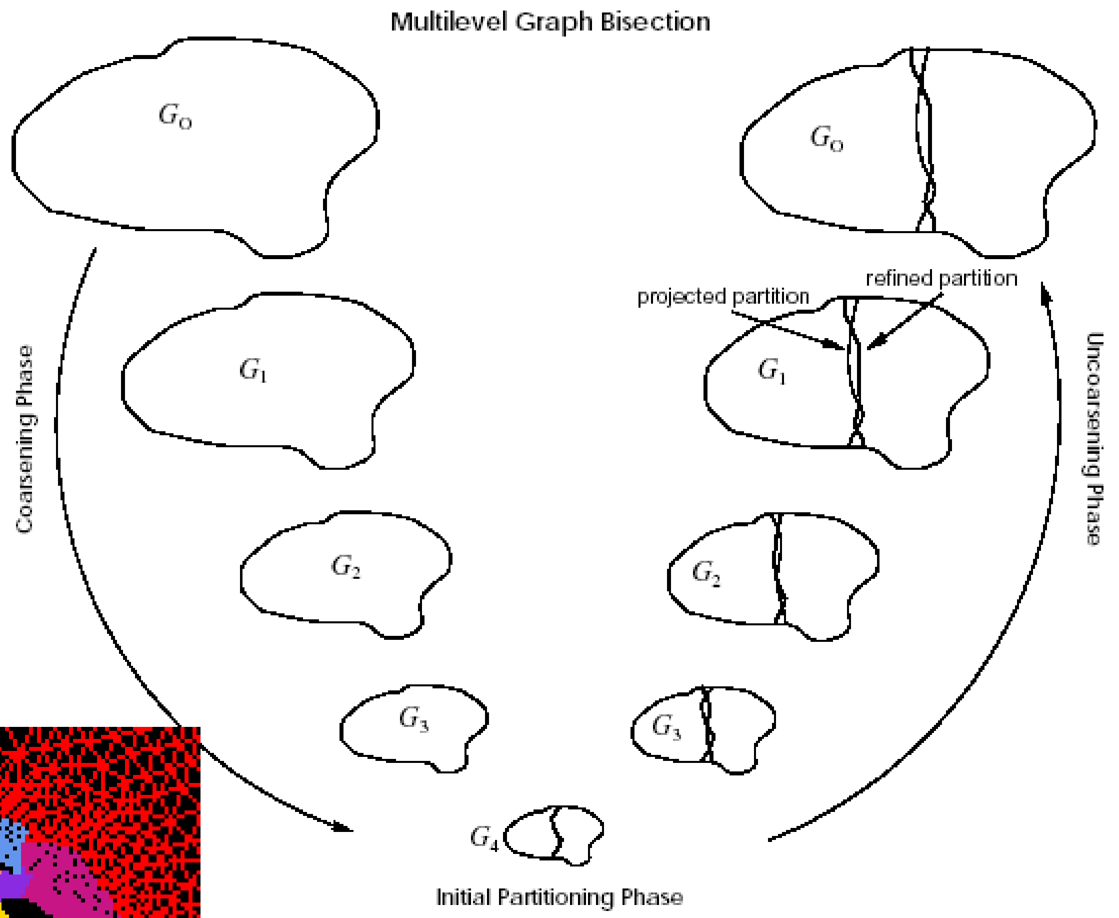
Multilevel Graph Partitioning

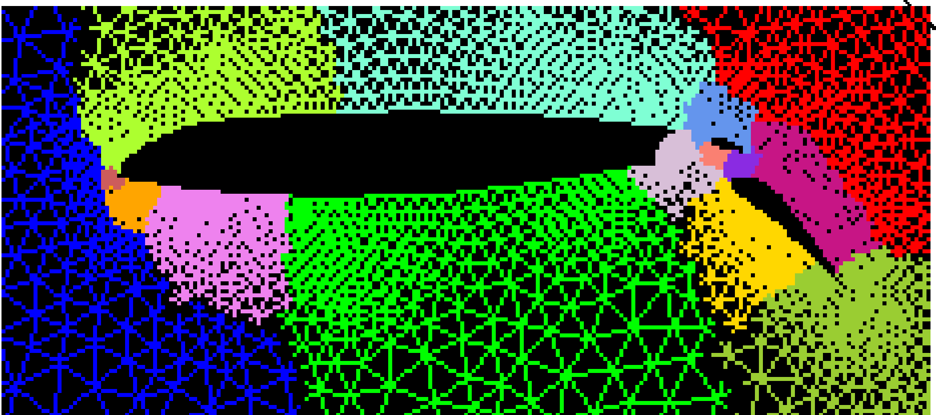
- 3 Phases

Coarsen

Partition

Uncoarsen



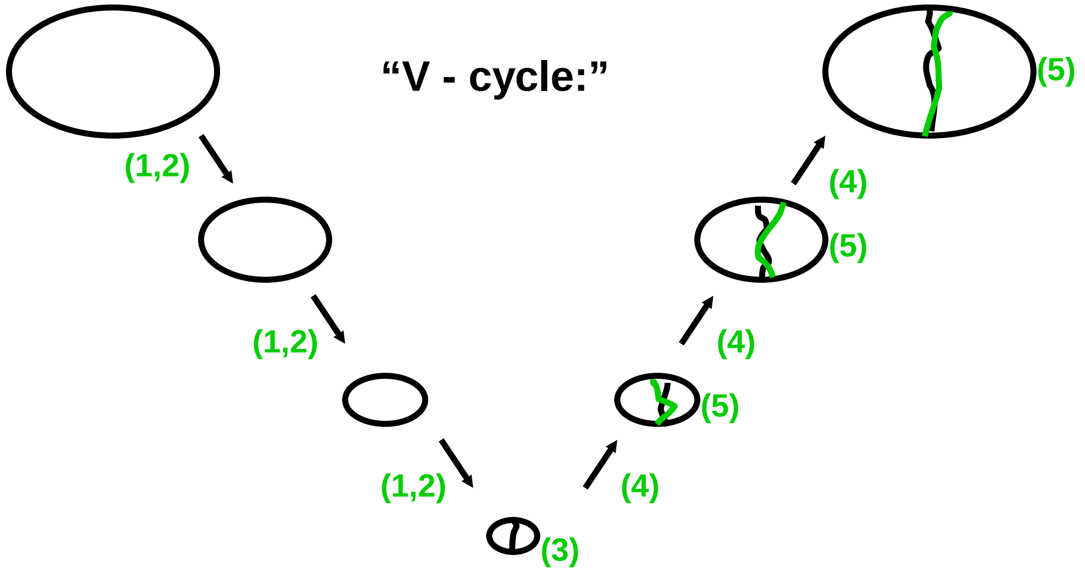


Multilevel Partitioning - High Level Algorithm

If G(N,E), is too big to partition efficiently, we

1) **Replace G(N,E) by a coarse approximation Gc(Nc,Ec), and**
2) **partition Gc instead and use partition of Gc to get a rough partitioning of G, and apply same idea recursively**
3) **until coarse graph can be partitioned**
4) **Expand back up to higher graph**
5) **Improve partition**

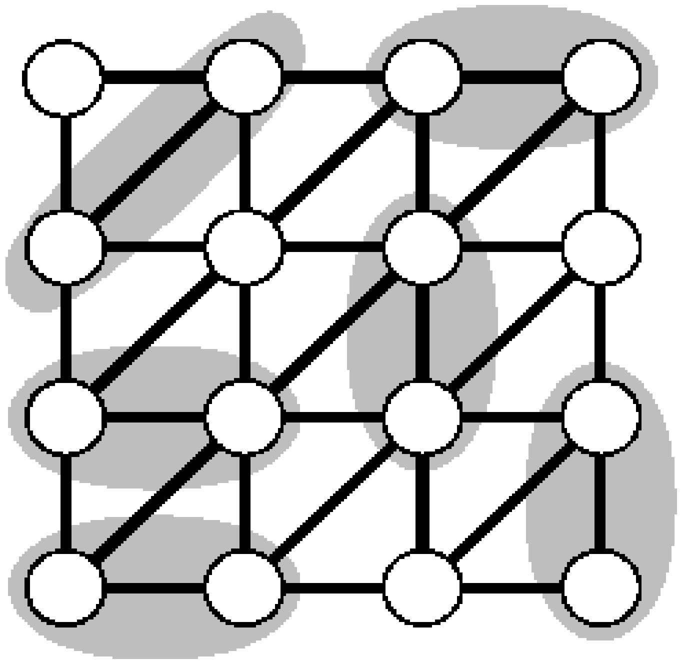
- Coarsen graph and expand partition using maximal matchings
- Improve partition using Kernighan-Lin

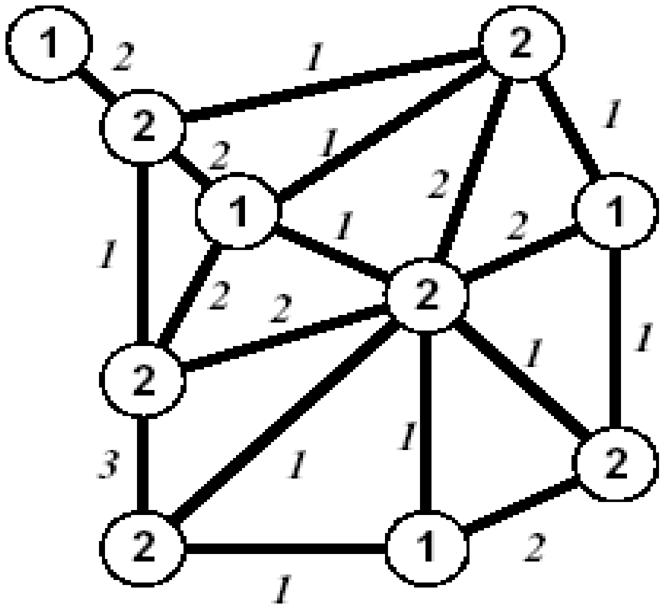


Graph Coarsening Example

Collapse adjacent vertices depending on “best” criteria

Shading shows collapsed nodes





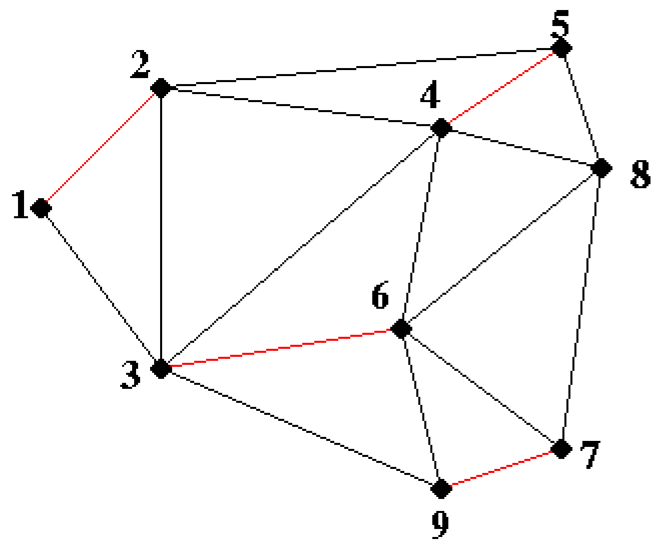
Maximal Matching

- *Definition*: A matching of a graph G(N,E) is a subset E_m of E such that no two edges in E_m share an endpoint
- *Definition:* A maximal matching of a graph G(N,E) is a matching E_m to which no more edges can be added and remain a matching
- A simple greedy algorithm computes a maximal matching:

Maximal Matching

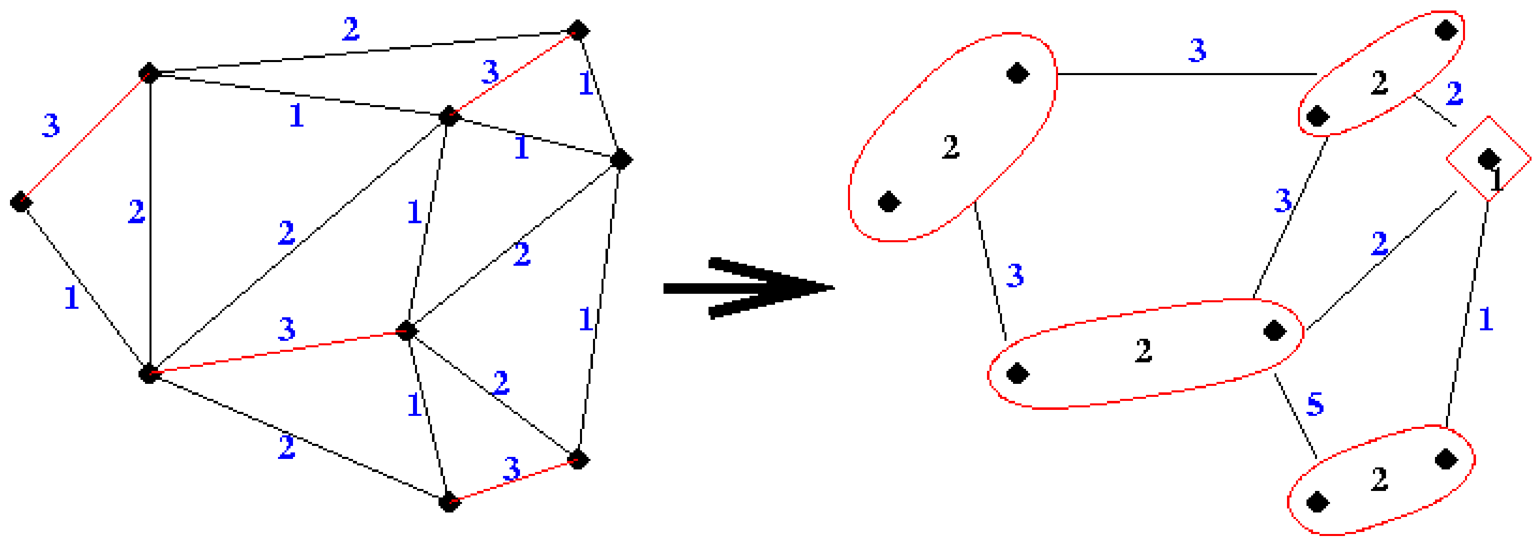
```
let $E_m$ be empty
mark all nodes in N as unmatched
for i = 1 to |N|      … visit the nodes in any order
    if i has not been matched
          if there is an edge e=(i,j)  where j is also unmatched,
              add e to $E_m$
              mark i and j as matched
          endif
    endif
endfor
```

Maximal Matching - Example



Example of Coarsening

How to coarsen a graph using a maximal matching



$G = (N, E)$

E_m is shown in red

Edge weights shown in blue

Node weights are all one

$G_c = (N_c, E_c)$

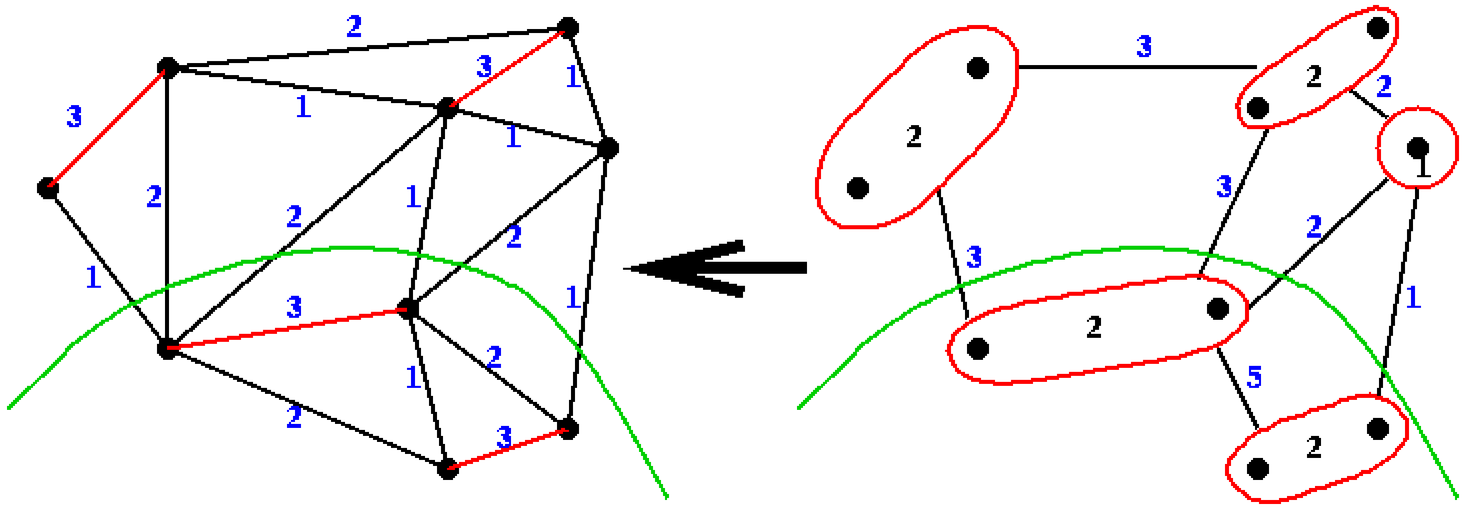
N_c is shown in red

Edge weights shown in blue

Node weights shown in black

Expanding a partition of G_c to a partition of G

Converting a coarse partition to a fine partition



Partition shown in green

Partitioning Phase

Find small edge-cut partition of the coarse graph into two sub-graphs of similar size

Use: Spectral Bisection, Kernighan-Lin, Fiduccia-Mattheyses etc

Kernighan-Lin Algorithm is iterative in nature
Starts with an initial partition of the graph
Searches for a subset of vertices from each part of the graph
such that a node swop gives a partition with a smaller edge-cut
Each search takes O(E log E)

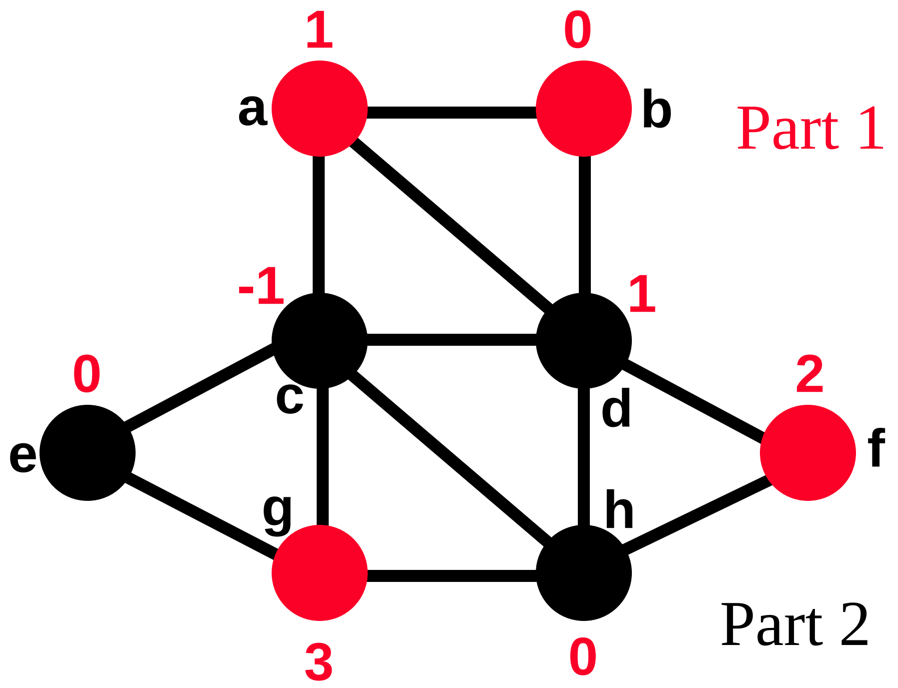
Fiduccia-Mattheyses Algorithm improves original KL algorithm
Reduces complexity to O(E) by using better data structures

Fiduccia-Mattheyses: Example (1)

Red nodes are in Part1;
Black nodes are in Part2.

The initial partition into two parts is arbitrary. In this case it cuts 8 edges.

The initial node gains are shown in Red.



Gain is difference in connectivity to black an red nodes

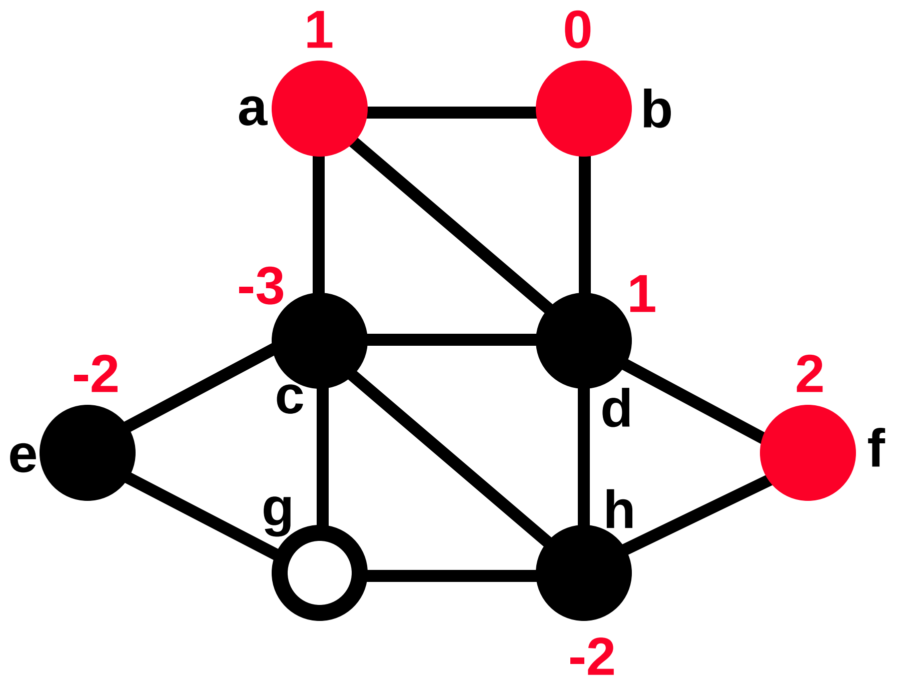
Nodes tentatively moved (and cut size after each pair):

none (8);

Fiduccia-Mattheyses: Example (2)

The node in Part1 with largest gain is g. Move it to Part2 and recompute the gains of its neighbors.

Tentatively moved nodes are hollow circles. After a node is tentatively moved its gain doesn't matter any more.



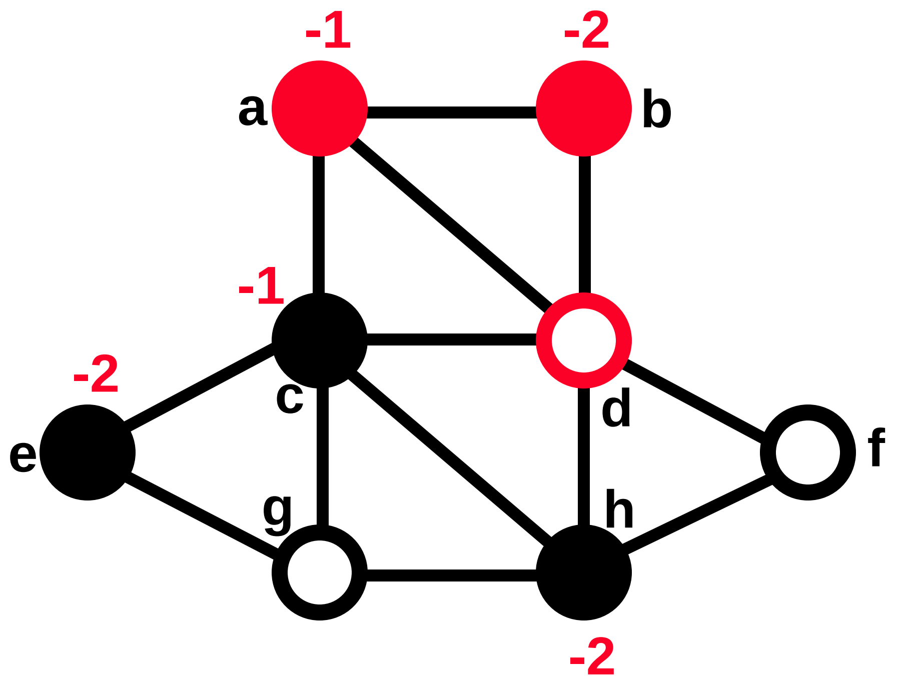
Gain = reduction in edge cut due to swap

Nodes tentatively moved (and cut size after each pair):

none (8); g,

Fiduccia-Mattheyses: Example (4)

The unmoved node in Part1 with largest gain is f. We tentatively move it to Part2 and recompute the gains of its neighbors.



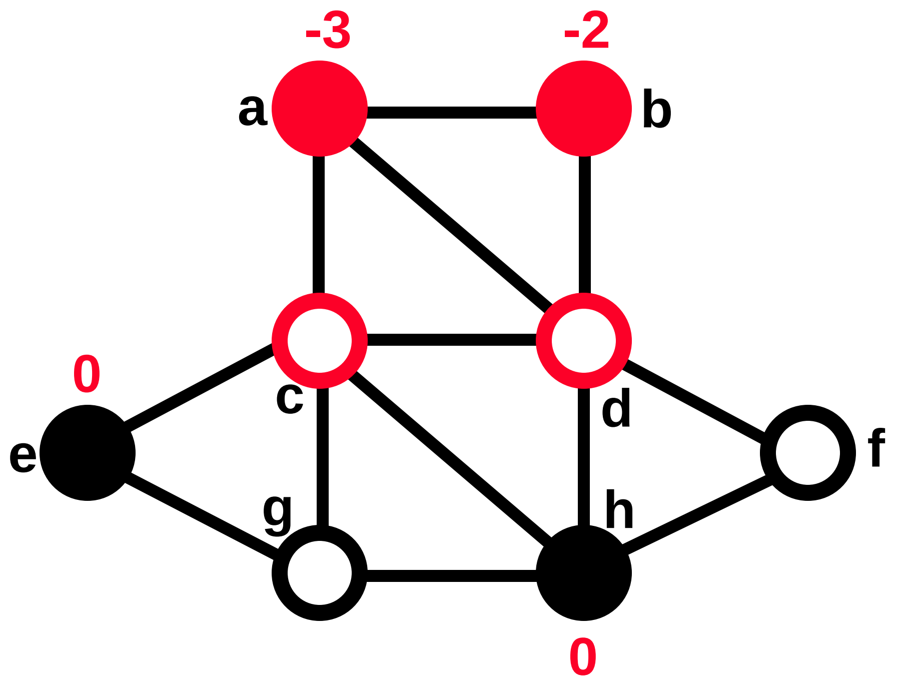
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f

Fiduccia-Mattheyses: Example (5)

The unmoved node in Part2 with largest gain is c. Move it to Part1 and recompute the gains of its neighbors.

After this tentative swap, the cut size is 5.



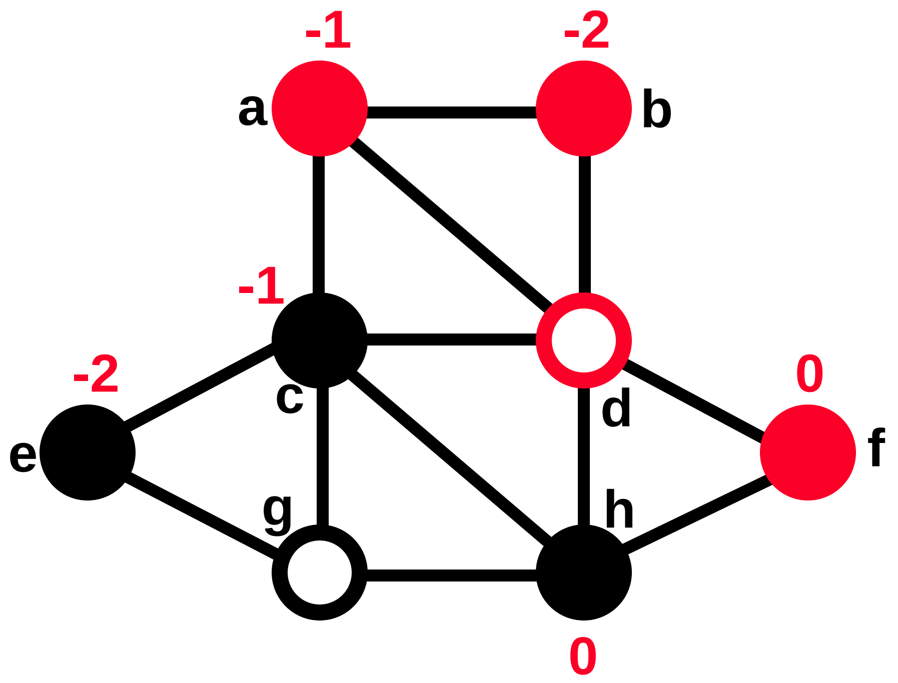
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5);

Fiduccia-Mattheyses: Example (3)

The node in Part2 with largest gain is d. Move it to Part1 and recompute the gains of its neighbors.

After this first tentative swap, the cut size is 4.

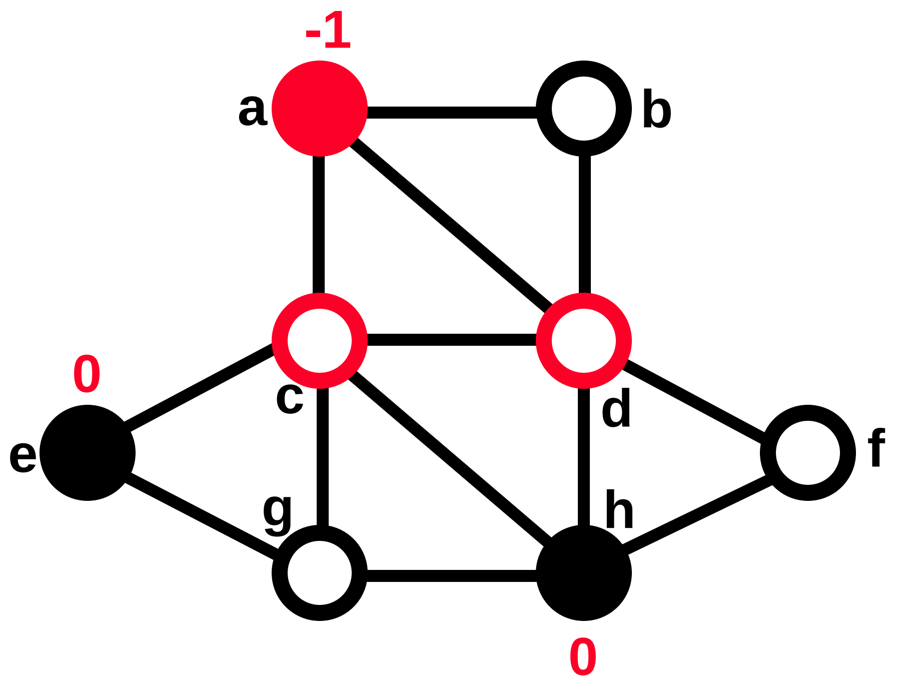


Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4);

Fiduccia-Mattheyses: Example (6)

The unmoved node in Part1 with largest gain is b. Move it to Part2 and recompute the gains of its neighbors.



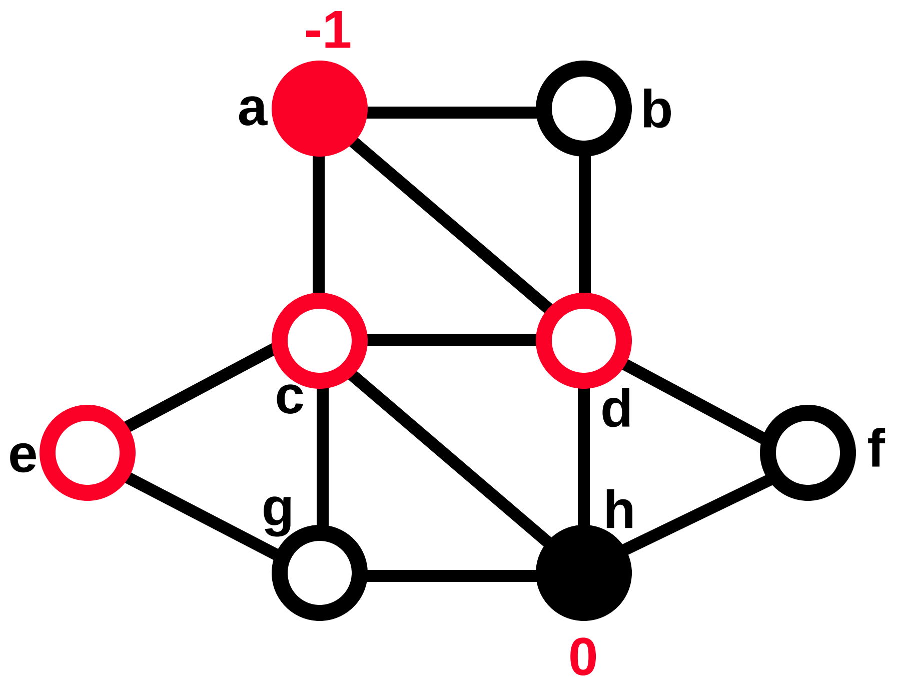
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5); b

Fiduccia-Mattheyses: Example (7)

There is a tie for largest gain between the two unmoved nodes in Part2. We choose one (say e) and tentatively move it to Part1. It has no unmoved neighbors so no gains are recomputed.

After this tentative swap the cut size is 7.

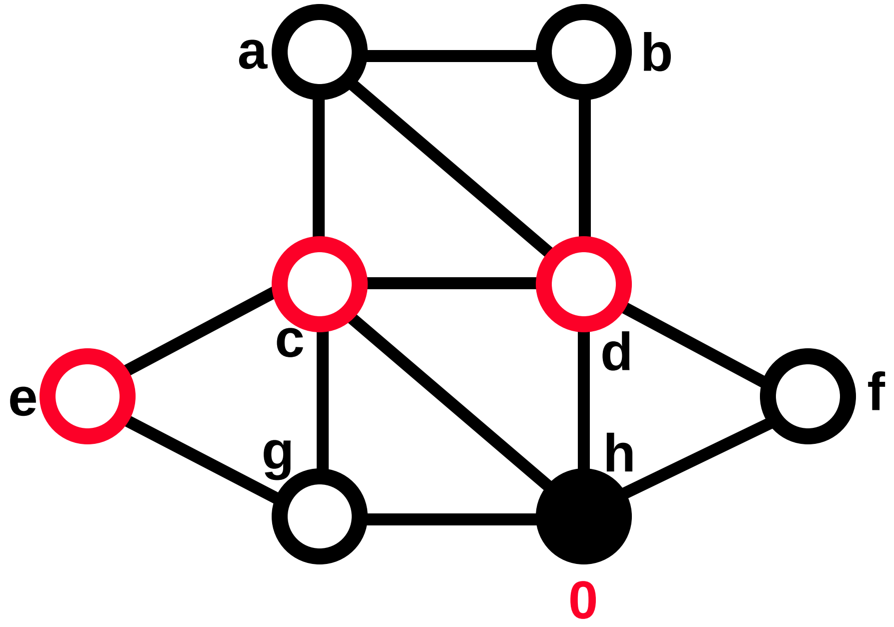


Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5); b, e (7);

Fiduccia-Mattheyses: Example (8)

The unmoved node in Part1 with the largest gain (the only one) is a. We tentatively move it to Part2. It has no unmoved neighbors so no gains are recomputed.



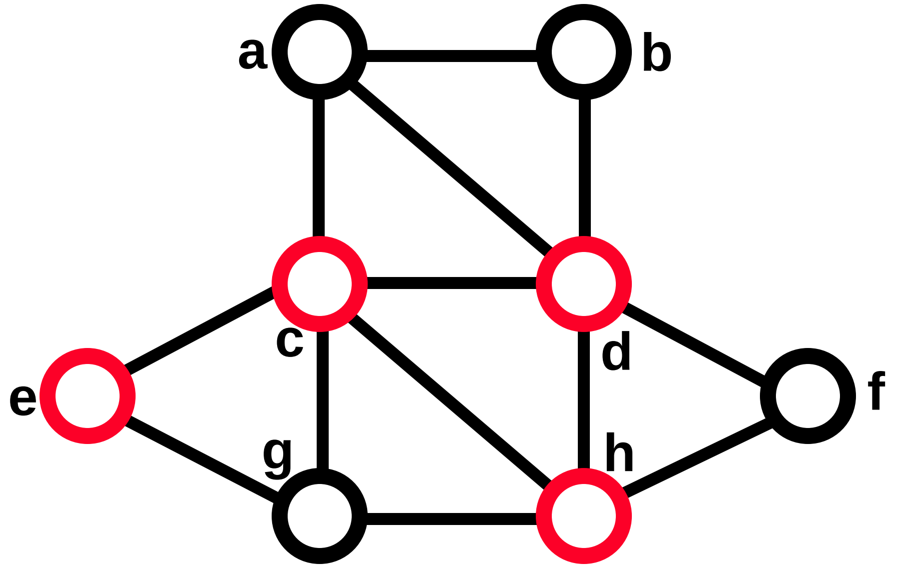
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5); b, e (7); a

Fiduccia-Mattheyses: Example (9)

The unmoved node in Part2 with the largest gain (the only one) is h. We tentatively move it to Part1.

The cut size after the final tentative swap is 8, the same as it was before any tentative moves.



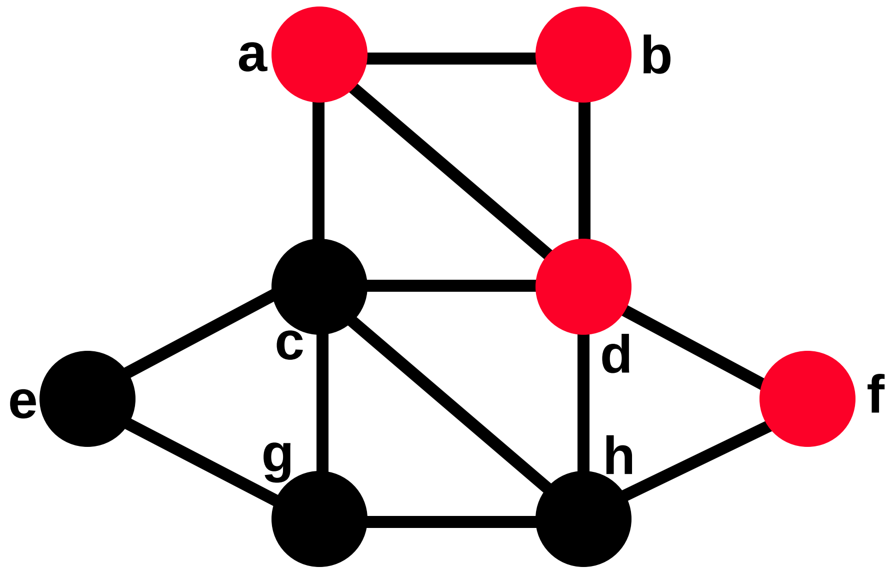
Nodes tentatively moved (and cut size after each pair):

none (8); g, d (4); f, c (5); b, e (7); a, h (8)

Fiduccia-Mattheyses: Example (10)

After every node has been moved, look back at the sequence and see that the smallest cut was 4, after swapping g and d. Make that swap permanent and undo all the later tentative swaps.

This is the end of the first improvement step.



Nodes tentatively moved (and cut size after each pair):

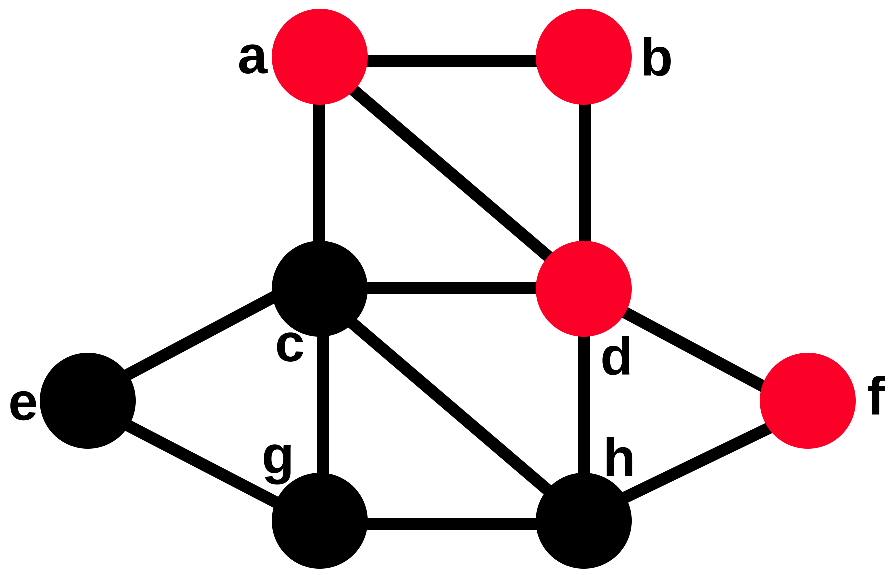
none (8); **g, d <u>(4);</u>** f, c (5); b, e (7); a, h (8)

Fiduccia-Mattheyses: Example (11)

Now recompute the gains and do another improvement step starting from the new size-4 cut.

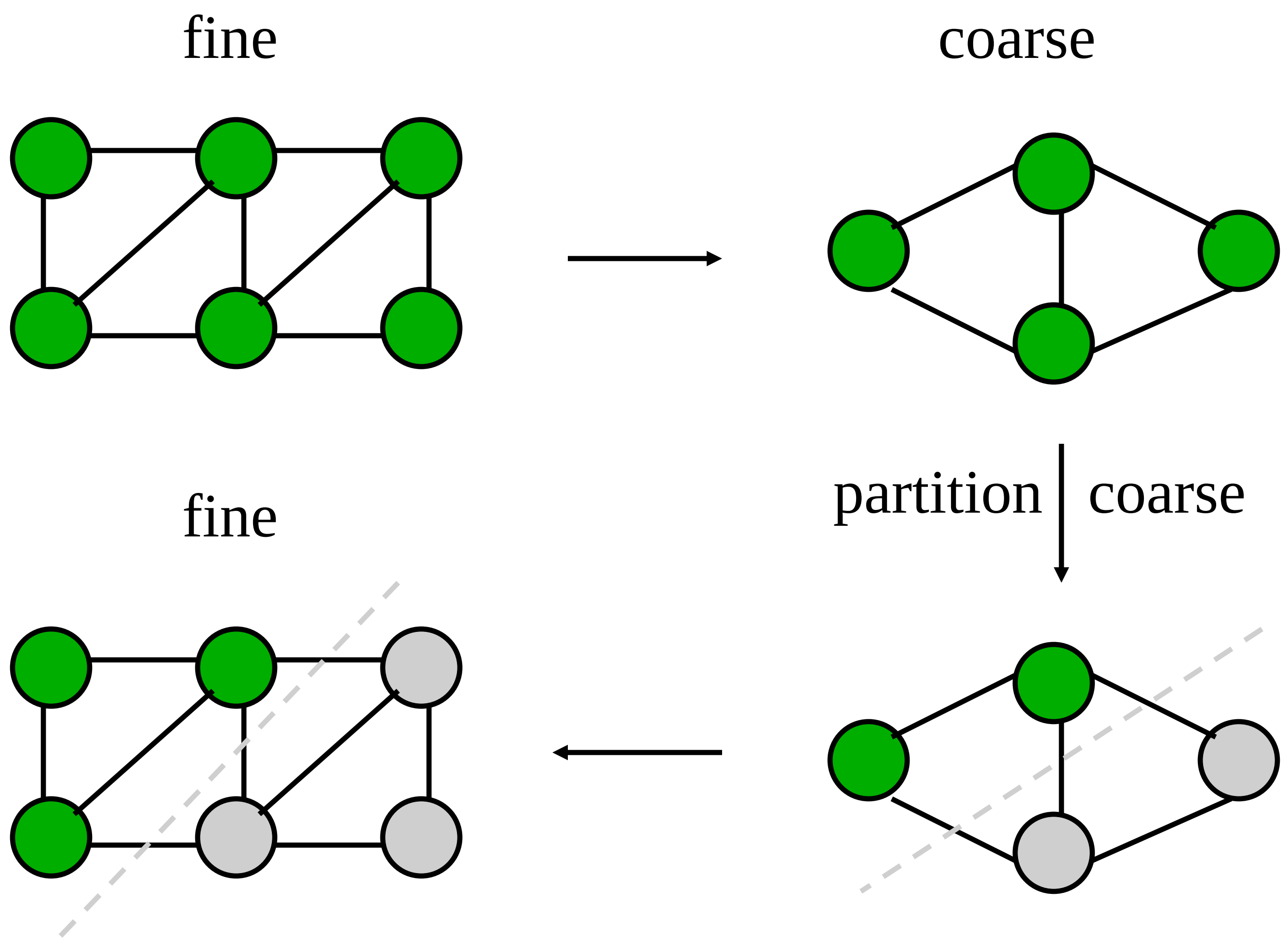
(not shown).

The second improvement step doesn't change the cut size, so the algorithm ends with a cut of size 4.



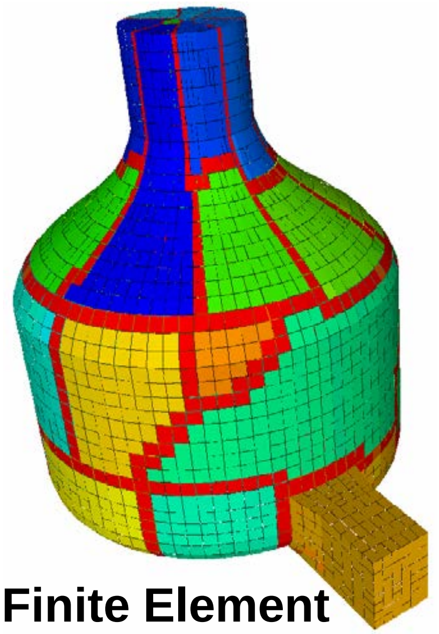
keep doing improvement steps while cut size reduced

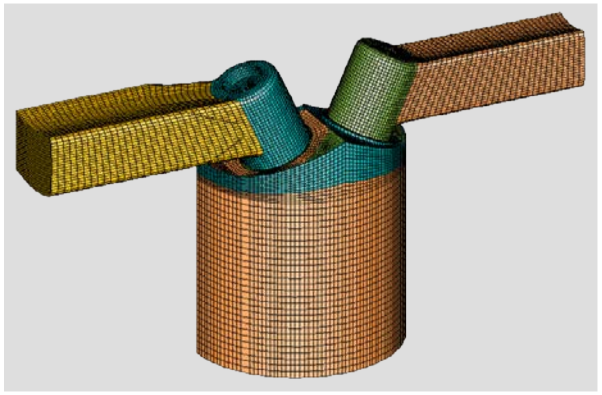
Uncoarsening Phase

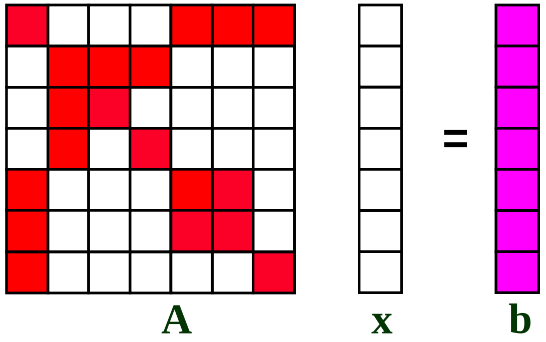


Algorithms such as these are at the heart of modern partitioners

Applications using Graph Partitioning


Finite Element Analysis

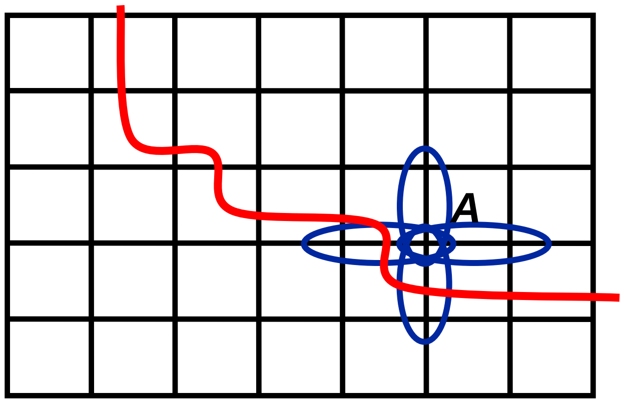

Multiphysics and multiphase simulations



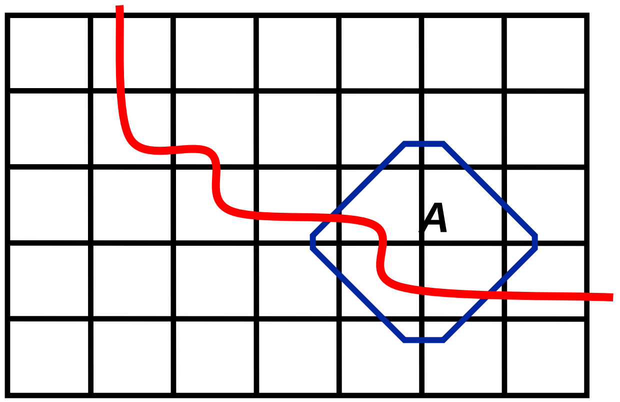
Linear solvers & preconditioners
(square, structurally symmetric systems)

Hypergraph Partitioning

- Alpert, Kahng, Hauck, Borriello, Çatalyürek, Aykanat, Karypis, et al.
- Hypergraph model:
  - Vertices = objects to be partitioned.
  - Hyperedges = dependencies between two *or more* objects.
- Partitioning goal: Assign equal vertex weight while minimizing hyperedge cut weight.

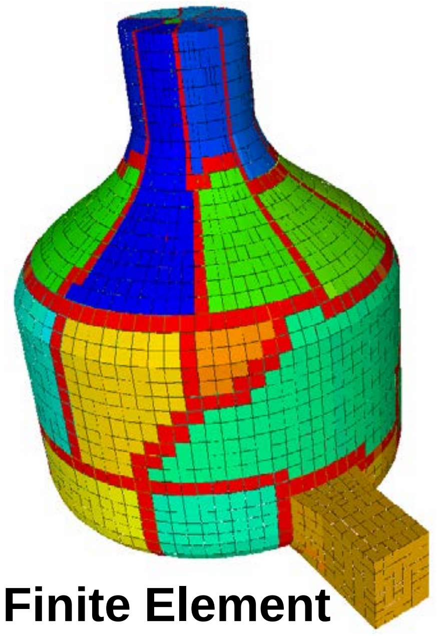


Graph Partitioning Model

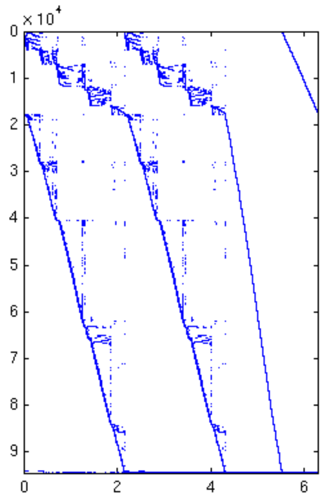


Hypergraph Partitioning Model

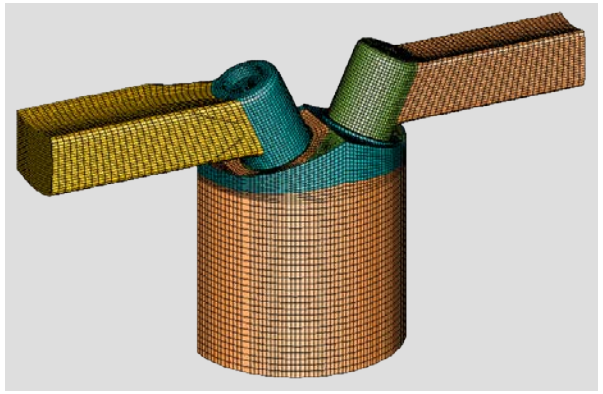
Hypergraph Applications



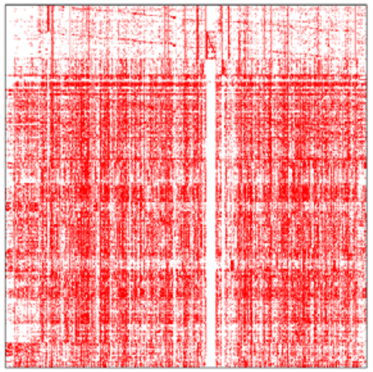
Finite Element Analysis



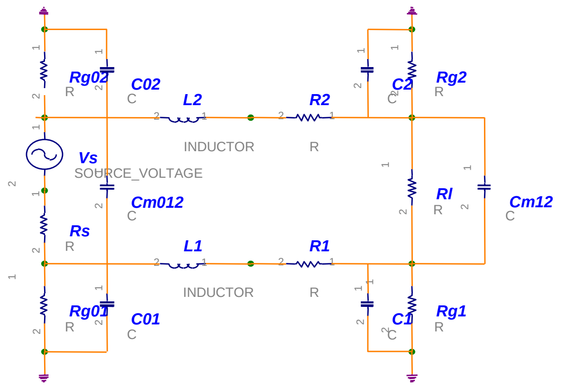
Linear programming for sensor placement



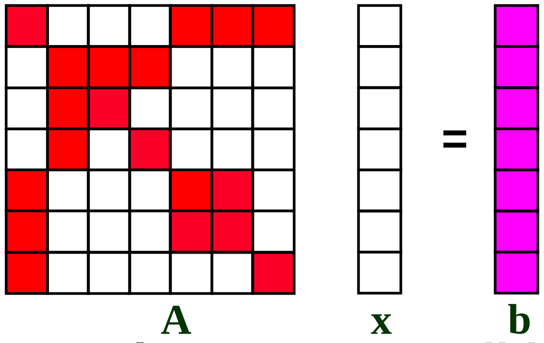
Multiphysics and multiphase simulations



Data Mining



Circuit Simulations



Linear solvers & preconditioners

(no restrictions on matrix structure)

Hypergraph Partitioning: Advantages and Disadvantages

- Advantages:
  - Communication volume reduced 30-38% on average over graph partitioning (Catalyurek & Aykanat).
    - 5-15% reduction for mesh-based applications.
  - More accurate communication model than graph partitioning.
    - Better representation of highly connected and/or non-homogeneous systems.
  - Greater applicability than graph model.
    - Can represent rectangular systems and non-symmetric dependencies.
- Disadvantages:
  - More expensive than graph partitioning.

Hypergraph Repartitioning

- Augment hypergraph with data redistribution costs.
  - Account for data's current processor assignments.
  - Weight dependencies by their size and frequency of use.
- Hypergraph partitioning then attempts to minimize *total communication volume*:

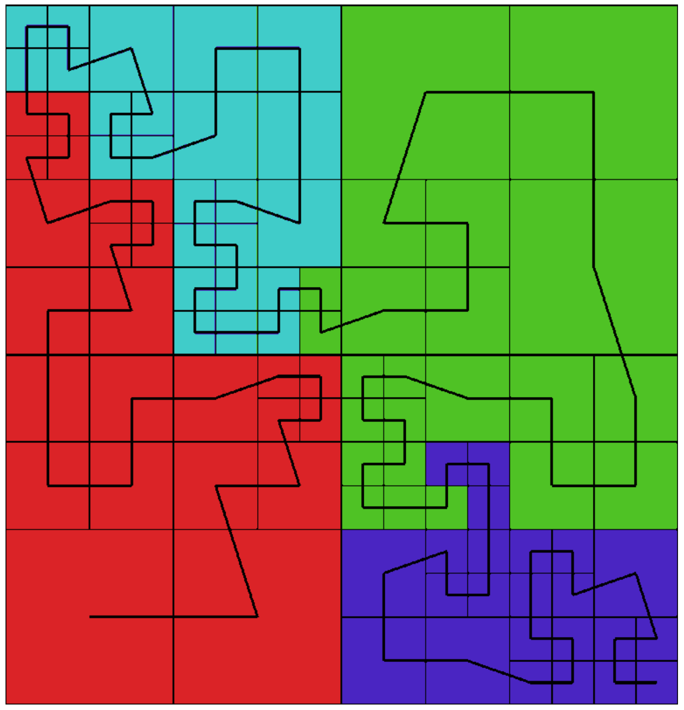
  Data redistribution volume
  + Application communication volume

  Total communication volume

Load Balancing via Space Filling Curves

- **Improve Load Balance**
  - Cost Estimation Algorithms can be based on measurements
  - Use load balancing algorithms based on patches and a new fast space filling curve algorithm
  - Space filling curve deals with a mesh representation of work.
  - A single line is passed through all the mesh.
  - Partitions of this line form the assignment of work to a processor

In this case the graph connects the centroids of all adjacent cells



Load Balancing Weight Estimation

Need to assign the same workload to each processor.

- Algorithmic Cost Models based on discretization method
  - Vary according to simulation+machine
  - Requires accurate information from the user
- Time Series Analysis
  - Used to forecast time for execution on each patch
  - Automatically adjusts according to simulation and architecture with no user interaction

Simple Exponential Smoothing:

Er,t: Estimated Time Or,t: Observed Time α: Decay Rate

$$E_{r,t+1} = \alpha\, O_{r,t} + (1-\alpha)\, E_{r,t}$$

$$= \alpha \underbrace{(O_{r,t} - E_{r,t})}_{\text{Error in last prediction}} + E_{r,t}$$

Time Series Analysis

Simple Exponential Smoothing:

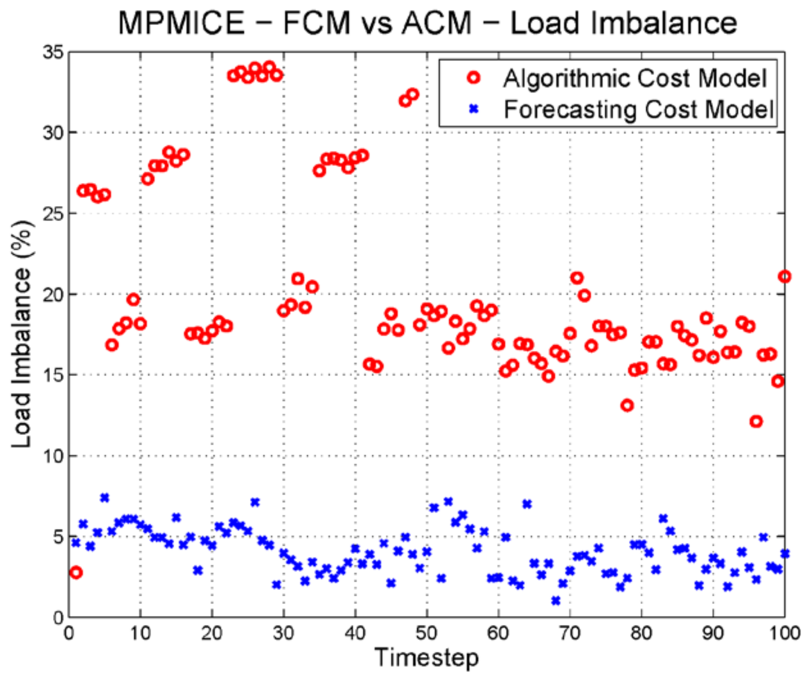
$E_{r,t}$: Estimated Time $O_{r,t}$: Observed Time α: Decay Rate

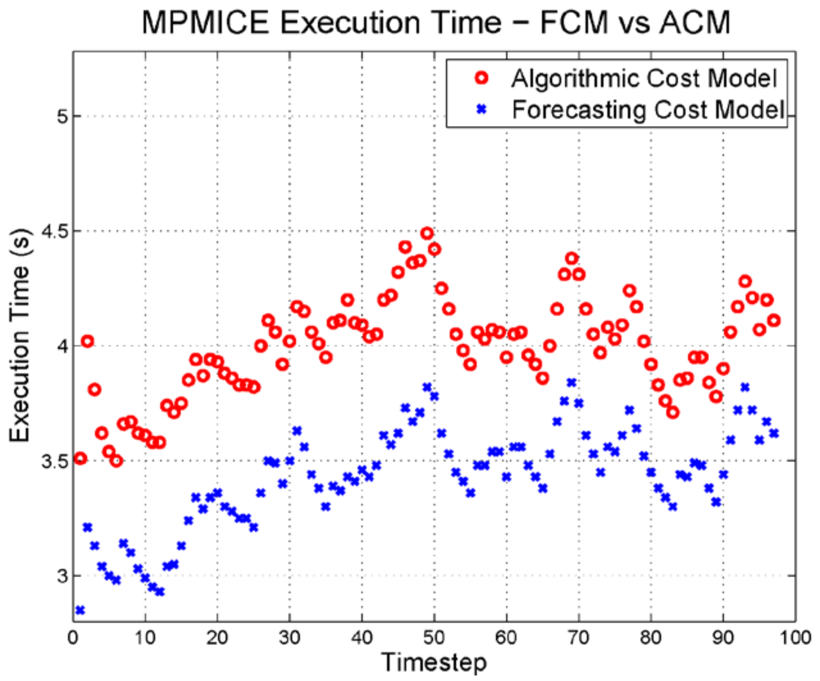
$$E_{r,t+1} = \alpha \; O_{r,t} + (1 - \alpha) \; E_{r,t}$$

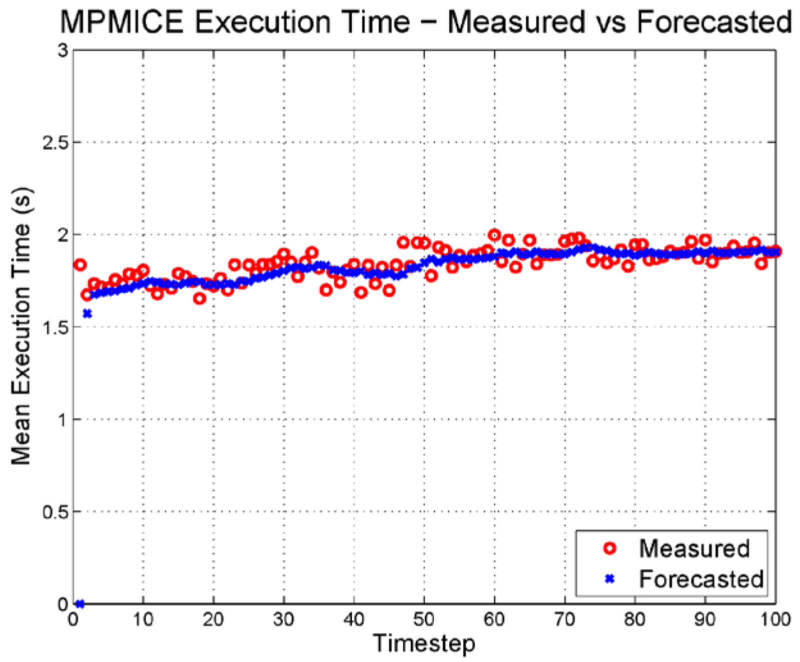
$$= \alpha \underbrace{(O_{r,t} - E_{r,t})}_{\text{Error in last prediction}} + E_{r,t}$$

Compute on fixed intervals (regions)

- Allows patches to change







Comparison between Forecast Cost Model FCM & Algorithmic Cost Model Particles + Fluid code FULL SIMULATION

SUMMARY

- Many partitioning methods and good software available
- Scalability is an issue and much work remains to be done to scale to the next generation of parallel machines
- An old but valuable critique of graph partitioning follows