Advanced Topics in Numerical Analysis: High Performance Computing

MATH-GA 2012.001 & CSCI-GA 2945.001

Georg Stadler
Courant Institute, NYU
stadler@cims.nyu.edu

Spring 2017, Thursday, 5:10–7:00PM, WWH #512

April 27, 2017

Organization

Today

- Organization, final projects planning
- Multigrid
- Partitioning and load balance

Organization

Final projects:

- There will be short final homework, posted tonight.
- Project presentation May 10/11.
- I will ask you early next week for a status report.
- Each group is expected to present 10 minutes (5-7 slides), and summarize their work in a final report.
- Final report must acknowledge sources of code and outside help. It also must include a short statement of who did what in the team. Don't plagarize!

Organization

Final projects:

- Please focus presentation/project description on HPC aspects:
  - What parallelization and why?
  - Flop rate, main computational kernel, memory access,. . .
  - Scalability (weak/strong). Please run your code not only on your Laptop but on a "real" computing device (Stampede, HPC, crunchy).
  - What's limits solving your problem faster, solving larger problems? Communication? Memory access? Amdahl's law?
  - Your personal experience/What have you learned?
  - Please put your code in a repository; I would like to post a link to your code and your final reports.

Partitioning and Load Balancing

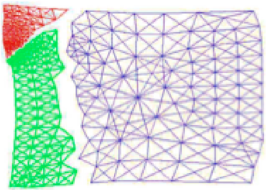
Thanks to Marsha Berger for letting me use many of her slides. Thanks to the Schloegel, Karypis and Kumar survey paper and Zoltan website for many of today's slides and pictures

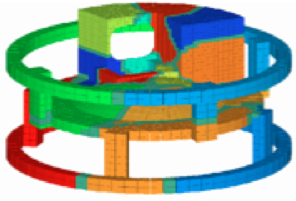
Partitioning

- Decompose computation into tasks to equi-distribute the data and work, minimize processor idle time.

  applies to grid points, elements, matrix rows, particles, . . .
- Map to processors to keep interprocessor communication low.

  communication to computation ratio comes from both the partitioning and the algorithm.

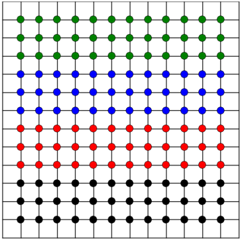


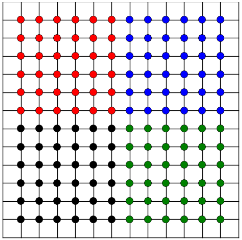


Partitioning

Data decomposition + Owner computes rule:

- Data distributed among the processors
- Data distribution defines work assignment
- Owner performs all computations on its data.
- Data dependencies for data items owned by different processors incur communication





Partitioning

- Static - all information available before computation starts

  use off-line algorithms to prepare before execution time; run as pre-processor, can be serial, can be slow and expensive, starts.

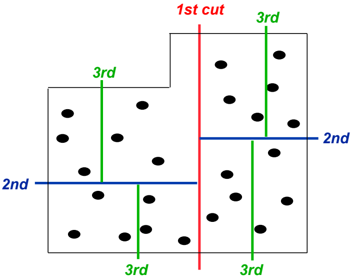
- Dynamic - information not known until runtime, work changes during computation (e.g. adaptive methods), or locality of objects change (e.g. particles move)

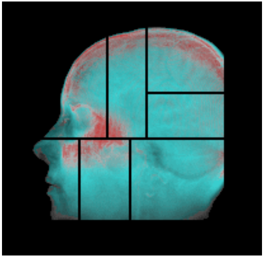
  use on-line algorithms to make decisions mid-execution; must run side-by-side with application, should be parallel, fast, scalable. Incremental algorithm preferred (small changes in input result in small changes in partitions)

will look at some geometric methods, graph-based methods, spectral methods, multilevel methods, diffusion-based balancing,...

Recursive Coordinate Bisection

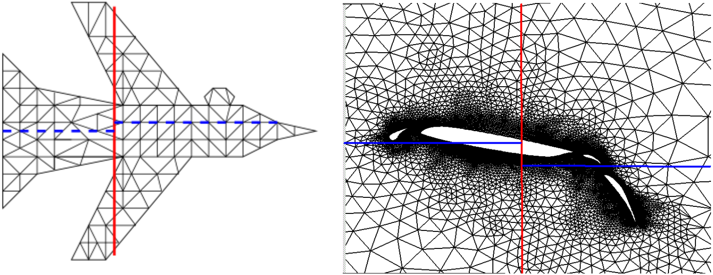
Divide work into two equal parts using cutting plane orthogonal to coordinate axis For good aspect ratios cut in longest dimension.





Can generalize to k-way partitions. Finding *optimal* partitions is NP hard. (There are optimality results for a class of graphs as a graph partitioning problem.)

Recursive Coordinate Bisection



+ Conceptually simple, easy to implement, fast.

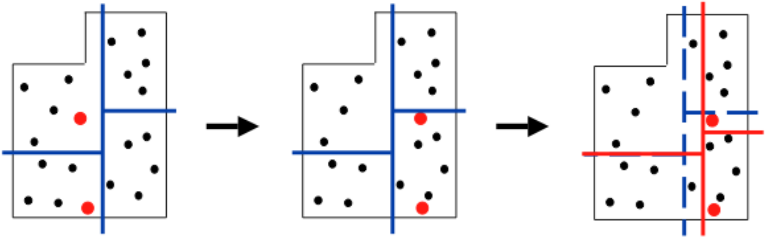
+ Regular subdomains, easy to describe

– Need coordinates of mesh points/particles.

– No control of communication costs.

– Can generate disconnected subdomains

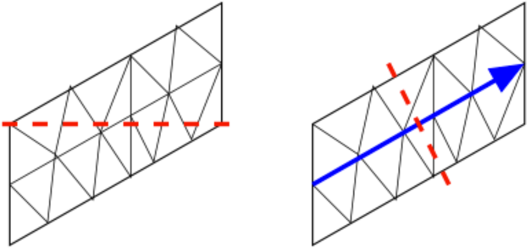
Recursive Coordinate Bisection



Implicitly incremental - small changes in data result in small movement of cuts

Recursive Inertial Bisection

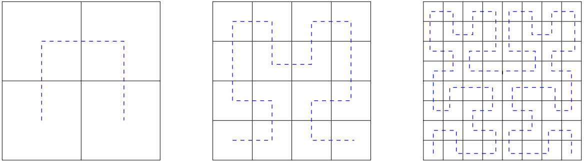
For domains not oriented along coordinate axes can do better if account for the angle of orientation of the mesh.



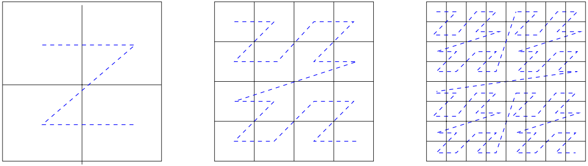
Use bisection line orthogonal to principal inertial axis (treat mesh elements as point masses). Project centers-of-mass onto this axis; bisect this ordered list. Typically gives smaller subdomain boundary.

Space-filling Curves

Linearly order a multidimensional mesh (nested hierarchically, preserves locality)



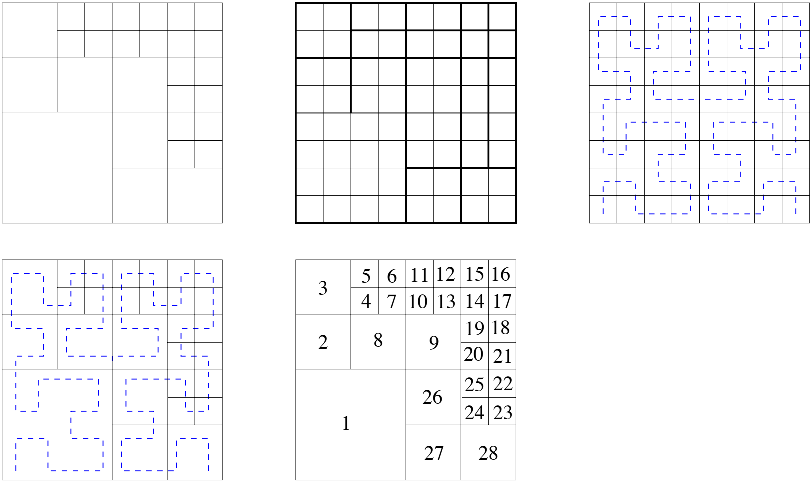
Peano-Hilbert ordering



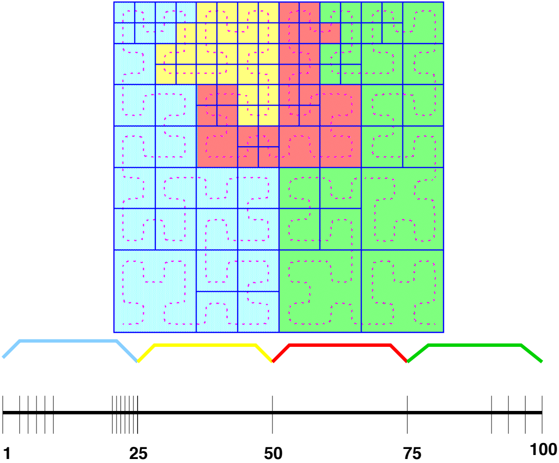
Morton ordering

Space-filling Curves

Easily extends to adaptively refined meshes

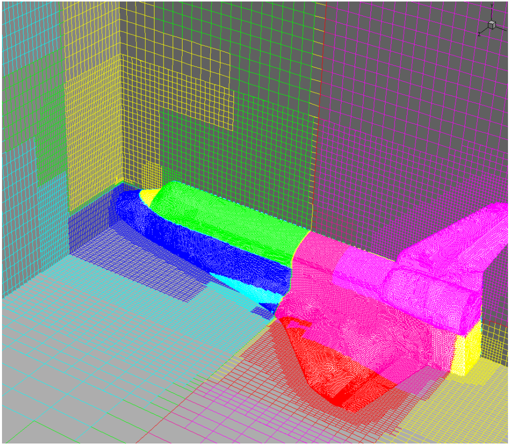


Space-filling Curves



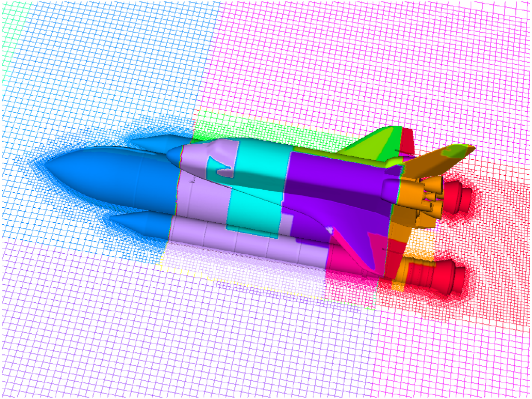
Partition work into equal chunks.

Space-filling Curves



+ Generalizes to uneven work loads - incorporate weights.
+ Dynamic on-the-fly partitioning for any number of nodes.
+ Good for cache performance

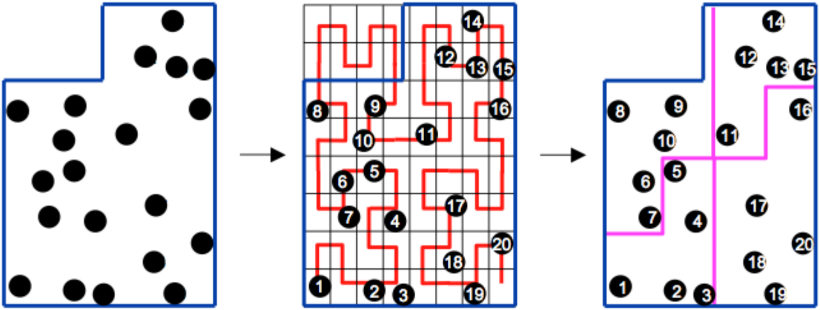
Space-filling Curves



- Red region has more communication - not compact
- Need coordinates

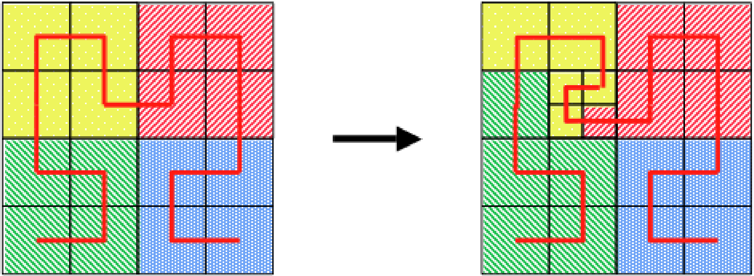
Space-filling Curves

Generalizes to other non-finite difference problems, e.g. particle methods, patch-based adaptive mesh refinement, smooth particle hydro.,

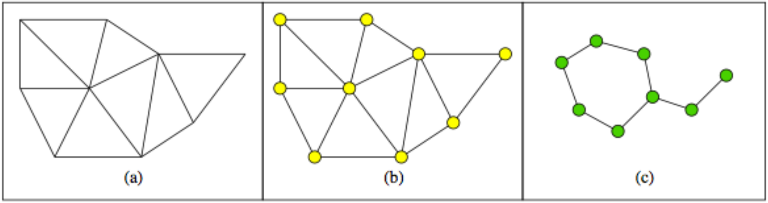


Space-filling Curves

Implicitly incremental - small changes in data results in small movement of cuts in linear ordering

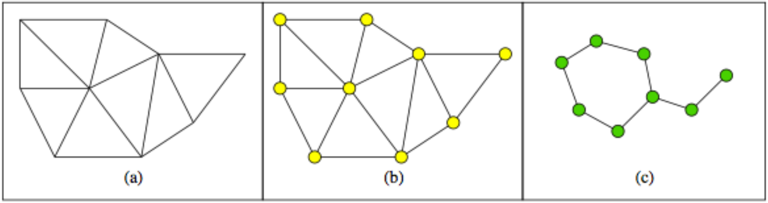


Graph Model of Computation



- for computation on mesh nodes, graph of the mesh is the graph of the computation; if there is an edge between nodes there is an edge between the vertices in the graph.
- for computation on the mesh elements the element is a vertex; put an edge between vertices if the mesh elements share an edge. This is the dual of the node graph.

Graph Model of Computation

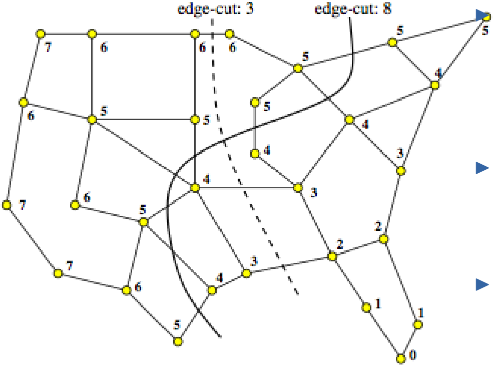


- for computation on mesh nodes, graph of the mesh is the graph of the computation; if there is an edge between nodes there is an edge between the vertices in the graph.
- for computation on the mesh elements the element is a vertex; put an edge between vertices if the mesh elements share an edge. This is the dual of the node graph.

Partition vertices into disjoint subdomains so each has same number. Estimate total communication by counting number of edges that connect vertices in different subdomains (the edge-cut metric).

Greedy Bisection Algorithm (also LND)

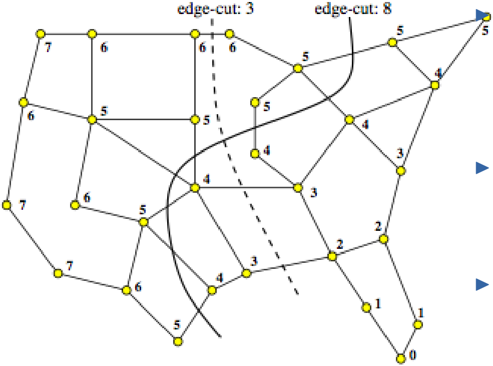
Put connected components together for min communication.



- Start with single vertex (peripheral vertex, lowest degree, endpoints of graph diameter)
- Incrementally grow partition by adding adjacent vertices (bfs)
- Stop when half the vertices counted (n/p for p partitions)

Greedy Bisection Algorithm (also LND)

Put connected components together for min communication.

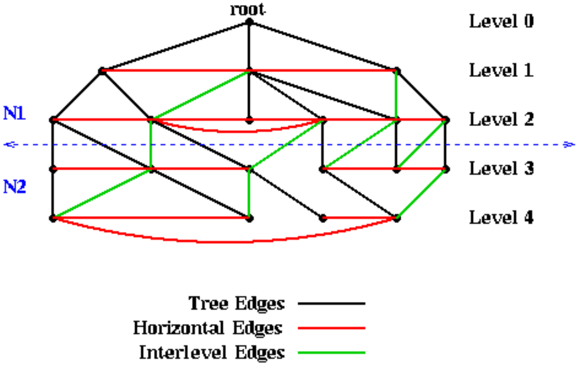


- Start with single vertex (peripheral vertex, lowest degree, endpoints of graph diameter)
- Incrementally grow partition by adding adjacent vertices (bfs)
- Stop when half the vertices counted (n/p for p partitions)

+ At least one component connected

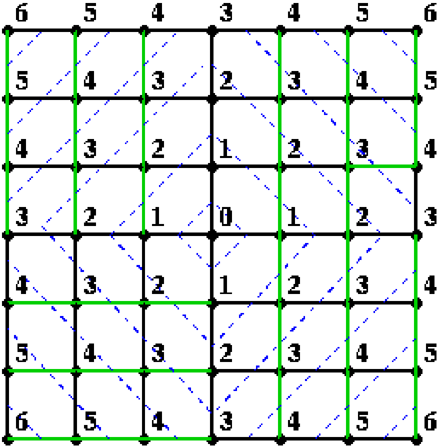
– Not best quality partitioning; need multiple trials.

Breadth First Search



- All edges between nodes in same level or adjacent levels.
- Partitioning the graph into nodes <= level L and >= L+1 breaks only tree and interlevel edges; no "extra" edges.

Breadth First Search

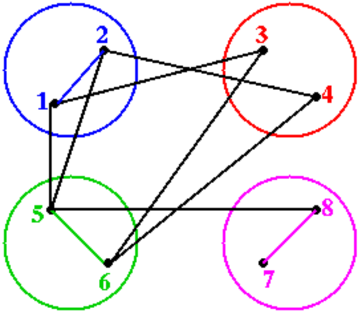


BFS of two dimensional grid starting at center node.

Graph Partitioning for Sparse Matrix Vector Mult.

Compute $y = Ax$, A sparse symmetric matrix,
Vertices v_i represent x_i, y_i.
Edge (i,j) for each nonzero A_{ij}

$$\mathbf{A} = \begin{array}{c|cc|cc|cc|cc|c} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \\ \hline & \bullet & \bullet & \bullet & & \bullet & & & & 1 \\ & \bullet & \bullet & & \bullet & \bullet & & & & 2 \\ \hline & \bullet & & \bullet & & & \bullet & & & 3 \\ & & \bullet & & \bullet & & \bullet & & & 4 \\ \hline & \bullet & \bullet & & & & \bullet & & \bullet & 5 \\ & & & \bullet & \bullet & \bullet & & & & 6 \\ \hline & & & & & & & \bullet & \bullet & 7 \\ & & & & & \bullet & & \bullet & \bullet & 8 \end{array}$$



Black lines represent communication.

Graph Partitioning for Sparse Matrix Factorization

Nested dissection for fill-reducing orderings for sparse matrix factorizations.
Recursively repeat:

- Compute vertex separator, bisect graph,

  edge separator = smallest subset of edges such that removing them divided graph into 2 disconnected subgraphs)

  vertex separator = can extend edge separator by connecting each edge to one vertex, or compute directly.

- Split a graph into roughly equal halves using the vertex separator

At each level of recursion number the vertices of the partitions, number the separator vertices last. Unknowns ordered from n to 1.

Smaller separators $\Rightarrow$ less fill and less factorization work

Spectral Bisection

Gold standard for graph partitioning (Pothen, Simon, Liou, 1990)

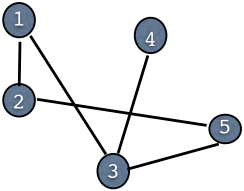
Let

$$x_i = \begin{cases} -1 & i \in A \\ 1 & i \in B \end{cases} \qquad \sum_{(i,j)\in E} (x_i - x_j)^2 = 4 \cdot \# \text{ cut edges}$$

Goal: find x to minimize quadratic objective function (edge cuts) for integer-valued $x = \pm 1$. Uses Laplacian L of graph G:

$$l_{ij} = \begin{cases} d(i) & i = j \\ -1 & i \neq j, (i,j) \in E \\ 0 & otherwise \end{cases}$$

Spectral Bisection



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

- A = adjacency matrix; D diagonal matrix
- L is symmetric, so has real eigenvalues and orthogonal evecs.
- Since row sum is 0, $Le = 0$, where $e = (111\ldots1)^t$
- Think of second eigenvector as first "vibrational" mode

Spectral Bisection

Note that

$$x^t L x = x^t D x - x^t A x = \sum_{i=1}^{n} d_i x_i^2 - 2 \sum_{(i,j)\in E} x_i x_j = \sum_{(i,j)\in E} (x_i - x_j)^2$$

Using previous example, $$x^t A x = (x_1\, x_2\, x_3\, x_4\, x_5) \begin{pmatrix} x_2 + x_3 \\ x_1 + x_5 \\ x_1 + x_4 + x_5 \\ x_3 + x_4 \\ x_2 + x_3 + x_5 \end{pmatrix}$$

So finding x to minimize cut edges looks like minimizing $x^t L x$ over vectors $x = \pm 1$ and $\sum_{i=1}^{n} x_i = 0$ (balance condition).

Spectral Bisection

- Integer programming problem difficult.
- Replace $x_i = \pm 1$ with $\sum_{i=1}^{n} x_i^2 = n$

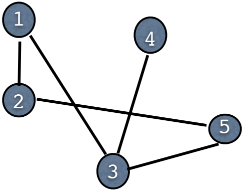
$$\begin{aligned} \min_{\substack{\sum x_i = 0 \\ \sum x_i^2 = n}} x^t L x &= x_2^t L x_2 \\ &= \lambda_2 \, x_2^t \cdot x_2 \\ &= \lambda_2 \, n \end{aligned}$$

- λ_2 is the smallest positive eval of L, with evec x_2, (assuming G is connected, $\lambda_1 = 0, x_1 = e$)
- x_2 satisfies $\sum x_i = 0$ since orthogonal to x_1, $e^t x_1 = 0$
- x_2 called Fiedler vector (properties studied by Fiedler in 70's).

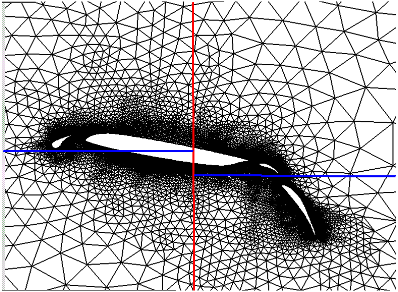
Spectral Bisection

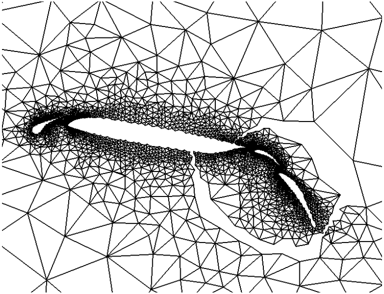
- Assign vertices according to the sign of the x_2. Almost always gives connected subdomains, with significantly fewer edge cuts than RCB. (Thrm. (Fiedler) If G is connected, then one of A,B is. If $\nexists i, x_{2_i} = 0$ then other set is connected too).
- Recursively repeat (or use higher order evecs)

$$v2 = \begin{pmatrix} .256 \\ .437 \\ -.138 \\ -.811 \\ .256 \end{pmatrix}$$



Spectral Bisection





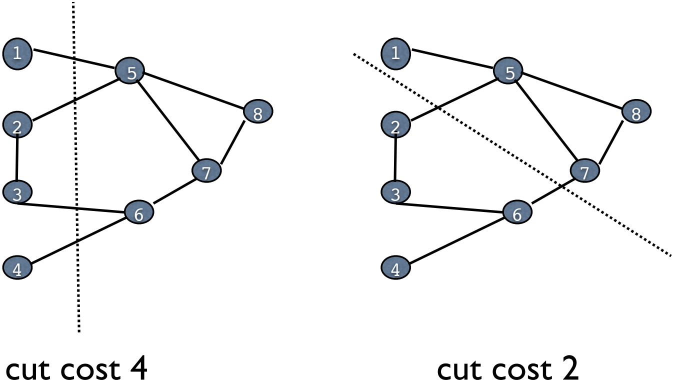
+ High quality partitions

– How find second eval and evec? (Lanczos, or CG, ... how do this in parallel, when you don't yet have the partition?)

Kernighan-Lin Algorithm

- Heuristic for graph partitioning (even 2 way partitioning with unit weights is NP complete)

- Needs initial partition to start, iteratively improve it by making small local changes to improve partition quality (vertex swaps that decrease edge-cut cost)



Kernighan-Lin Algorithm

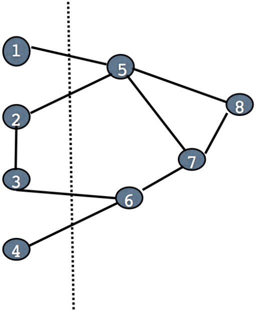
More precisely, the problem is:

- Given: an undirected graph $G(V, E)$ with $2n$ vertices, edges $(a, b) \in E$ with weights $w(a, b)$
- Find: sets A and B, so that $V = A \cup B$, $A \cap B = 0$, and $|A| = |B| = n$ that minimizes the cost $\sum_{(a,b) \in AxB} w(a, b)$
- Approach: Take initial partition and iteratively improve it. Exchange two vertices and see if cost of cut size is reduced. Select best pair of vertices, lock them, continue. When all vertices locked one iteration is done.

Original algorithm $O(n^3)$. Complicated improvement by Fiduccia-Mattheyses is $O(|E|)$.

Kernighan-Lin Algorithm

- Let C = cost(A,B)
- E(a) = external cost of a in A
  $= \sum_{b \in B} w(a,b)$
- I(a) = internal cost of a in A
  $= \sum_{a' \in A, a' \neq a} w(a,a')$
- D(a) = cost of a in A = E(a) - I(a)



D(6) = 1   D(1) = 1
D(3) = 0   newD(3) =
-2

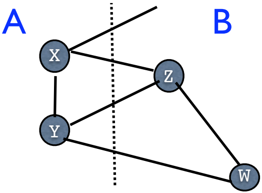
Consider swapping X={a} and Y={b}.
(newA = A - X ∪ Y   newB = B - Y ∪ X)

newC = C - (D(a) + D(b) - 2*w(a,b)) = C - gain(a,b)

newD(a') = D(a') + 2 w(a',a) - 2 w(a',b) for $a' \in A, a' \neq a$
newD(b') = D(b') + 2 w(b',b) - 2 w(b',a) for $b' \in B, b' \neq b$

Kernighan-Lin Algorithm

- Let C = cost(A,B)
- E(a) = external cost of a in A
  $= \sum_{b \in B} w(a, b)$
- I(a) = internal cost of a in A
  $= \sum_{a' \in A, a' \neq a} w(a, a')$
- D(a) = cost of a in A = E(a) - I(a)



D(Y) = 1   D(Z) = 1

newC = C

Consider swapping X={a} and Y={b}.
(newA = A - X ∪ Y   newB = B - Y ∪ X)

newC = C - (D(a) + D(b) - 2*w(a,b)) = C - gain(a,b)

newD(a') = D(a') + 2 w(a',a) - 2 w(a',b) for $a' \in A, a' \neq a$
newD(b') = D(b') + 2 w(b',b) - 2 w(b',a) for $b' \in B, b' \neq b$

Kernighan-Lin Algorithm

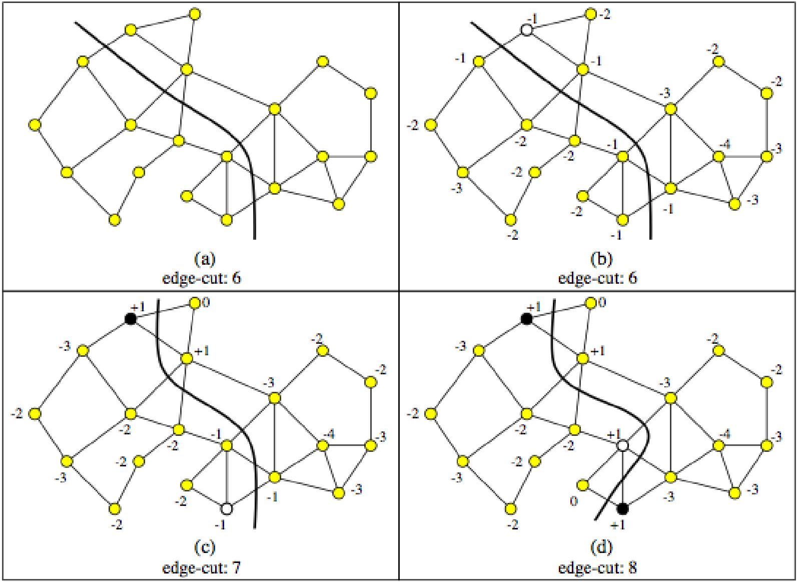
```
Compute C = cost(A,B) for initial A,B

Repeat
  Compute costs  D for all verts
  Unmark all nodes
  While there are unmarked nodes
    Find unmarked pair (a,b) maximizing gain(a,b)
    Mark a and b (do not swap)
    Update D for all unmarked verts (as if a,b swapped)
  End

  Pick sets of pairs maximizing gain
  if (Gain>0) then actually swap
     Update A' = A - {a1,a2,...am} + {b1,b2,...bm}
            B' = B - {b1,b2,...bm} + {a1,a2,...,am}
            C' = C - Gain

Until Gain<0
```

Kernighan-Lin Algorithm



(a)
edge-cut: 6

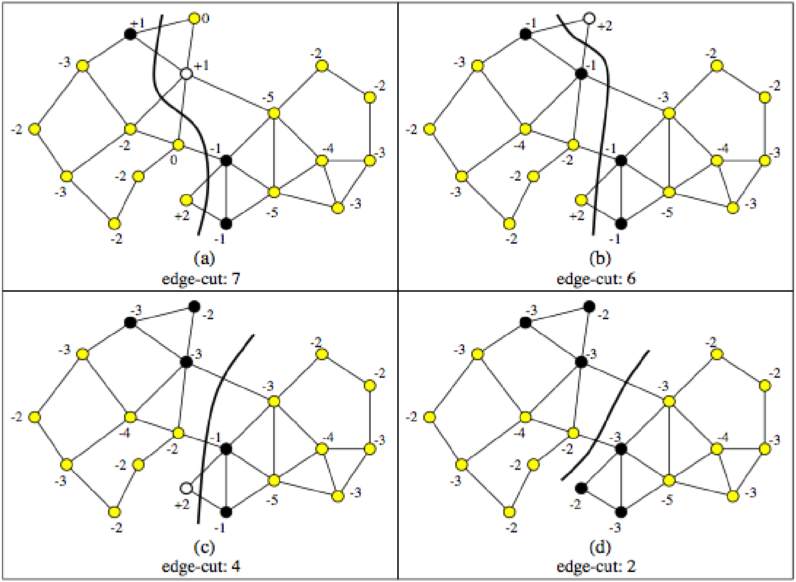
(b)
edge-cut: 6

(c)
edge-cut: 7

(d)
edge-cut: 8

KL can sometimes climb out of local minima...

Kernighan-Lin Algorithm



(a)
edge-cut: 7

(b)
edge-cut: 6

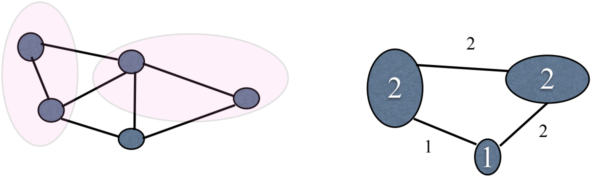
(c)
edge-cut: 4

(d)
edge-cut: 2

gets better solution; but need good partitions to start

Graph Coarsening

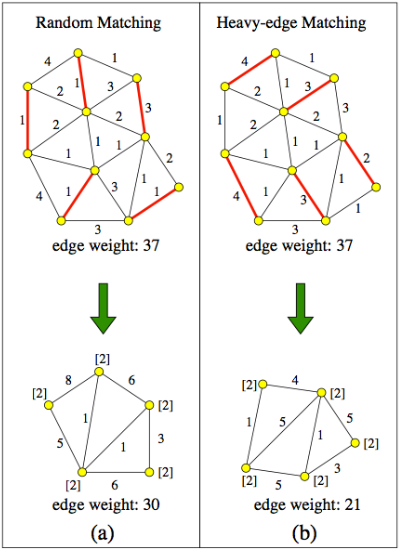
- Adjacent vertices are combined to form a multinode at next level, with weight equal to the sum of the original weights. Edges are the union of edges of the original vertices, also weighted. Coarser graph still *represents* original graph.



- Graph collapse uses maximal matching = set of edges, no two of which are incident on the same vertex. The matched vertices are collapsed into the multinode. Unmatched vertices copied to next level.

- Heuristics that combine 2 vertices sharing edge with heaviest weight, or randomly chosen unmatched vertex, ...

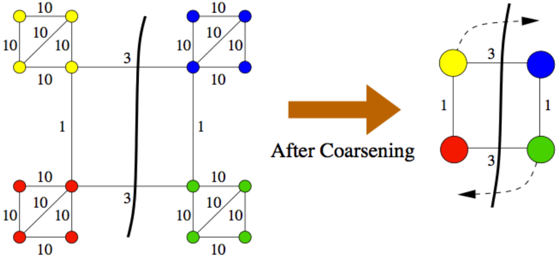
Graph Coarsening



Fewer remaining visible edges on coarsest grid ⇒ easier to partition

Multilevel Graph Partitioning

- Coarsen graph
- Partition the coarse graph
- Refine graph, using local refinement algorithm (e.g.K-L)
    - vertices in larger graph assigned to same set as coarser graph's vertex.
    - since vertex weight conserved, balance preserved
    - similarly for edge weights



Moving one node with K-L on coarse graph equivalent to moving large number of vertices in original graph but much faster.

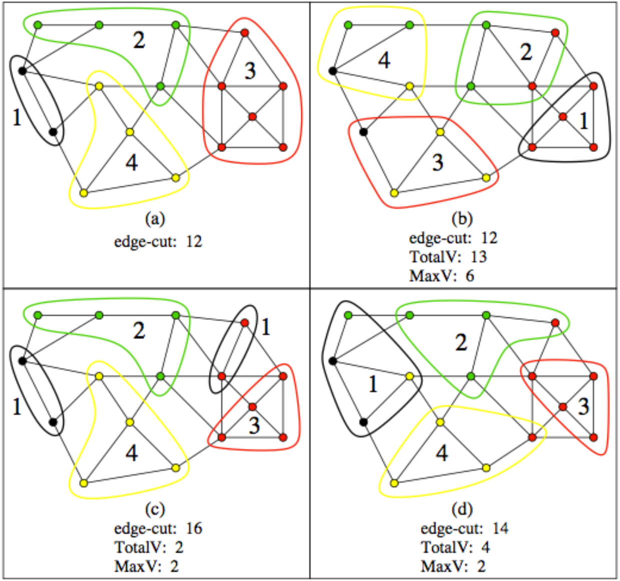
Re-Partitioning

when workload changes dynamically, need to re-partition as well as minimizing redistribution cost. Options include:

- partition from scratch (use incremental partitioner, or try to map on to processors well) called scratch-remap
- give away excess, called cut-and-paste repartitioning
- diffusive repartitioning

Should you minimize sum of vertices changing subdomains (total volume of communication = TotalV), or max volume per processor (called maxV).

Re-Partitioning



(b) from scratch (c) cut-and-paste, (d) diffusive

Diffusion-based Partitioning

- Iterative method used for re-partitioning - migrate tasks from overutilized processors to underutilized ones.
- Variations on which nodes to move, how many to move at one time.
- Based on Cybenko model

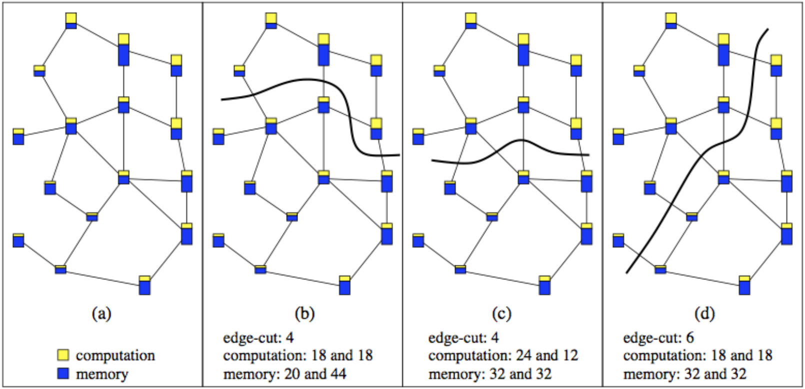
$$w_i^{t+1} = w_i^t + \sum_j \alpha_{ij}(w_j^t - w_i^t)$$

  if $w_j - w_i > 0$ processor j gives work to i, else other way around.
- At *steady state* the temperature is constant (computational load is equal)

Slow to converge, use multilevel version, or recursive bisection verion. Solve optimization problem to minimize norm of data movement (1- or 2-norm).

Multiphase/Multiconstraint Graph Partitioning

- Many simulations have multiple phases - e.g. first compute fluid step, next compute the structural deformation, move geometry,...
- Each step has different CPU and memory requirements. Would like to load balance each phase.
  - single partition that balances all phases?
  - multiple partition with redistribution between phases?



(a)

🟨 computation
🟦 memory

(b)

edge-cut: 4
computation: 18 and 18
memory: 20 and 44

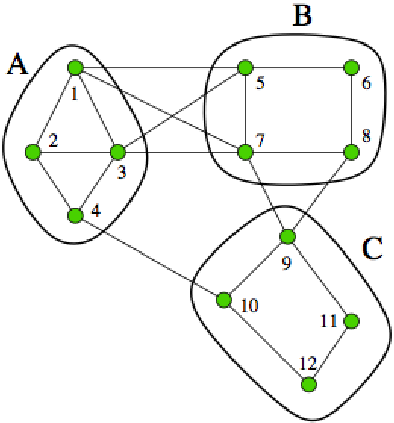
(c)

edge-cut: 4
computation: 24 and 12
memory: 32 and 32

(d)

edge-cut: 6
computation: 18 and 18
memory: 32 and 32

Issues with Edge Cut Approximation

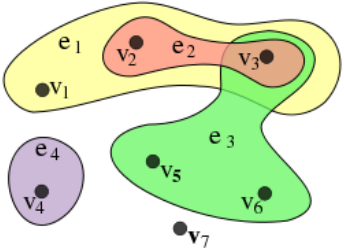


- 7 edges cut
- 9 items communicated
- vertex 1 in A connected to two vertices in B but it only needs to be sent once.

Edge cuts $\neq$ Communication volume
Communication volume $\neq$ Communication cost

Hypergraphs

Hypergraph $H = (V, E)$ where E is a
hyperedge = subset of V, i.e. connects more than two vertices



$$e_1 = \{v_1, v_2, v_3\}$$
$$e_2 = \{v_2, v_3\}$$
$$e_3 = \{v_3, v_5, v_6\}$$
$$e_4 = \{v_4\}$$

k-way partitioning: find $P = \{V_o, ..., V_{k-1}\}$ to minimize

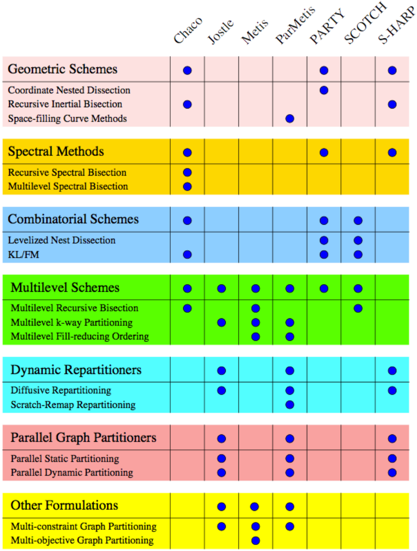
$$\text{cut(H,P)} = \Sigma_{i=0}^{|E|-1} (\lambda_i(H, P) - 1)$$

$\lambda_i(H, P)$ = number of partitions spanned by hyperedge i

Other Issues

- Heterogeneous machines
- Aspect ratio of subdomains (needed for convergence rate of iterative solvers)

Software Packages

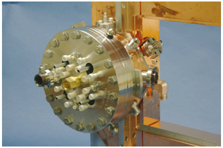


	Chaco	Jostle	Metis	ParMetis	PARTY	SCOTCH	S-HARP
Geometric Schemes	●				●		●
Coordinate Nested Dissection					●		
Recursive Inertial Bisection	●						●
Space-filling Curve Methods				●			
Spectral Methods	●				●		●
Recursive Spectral Bisection	●						
Multilevel Spectral Bisection	●						
Combinatorial Schemes	●				●	●	
Levelized Nest Dissection					●	●	
KL/FM	●				●	●	
Multilevel Schemes	●	●	●	●	●	●	
Multilevel Recursive Bisection	●		●			●	
Multilevel k-way Partitioning		●	●	●			
Multilevel Fill-reducing Ordering			●	●			
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			●				

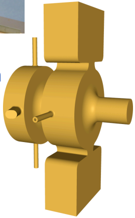
Also, graph partitioning archive at Univ. of Greenwich by Walshaw.

Slide 91

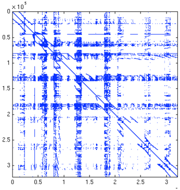
Test Data



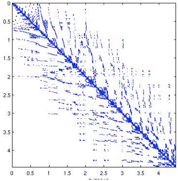
SLAC *LCLS
Radio Frequency Gun
6.0M x 6.0M
23.4M nonzeros



Xyce 680K ASIC Stripped
Circuit Simulation
680K x 680K
2.3M nonzeros

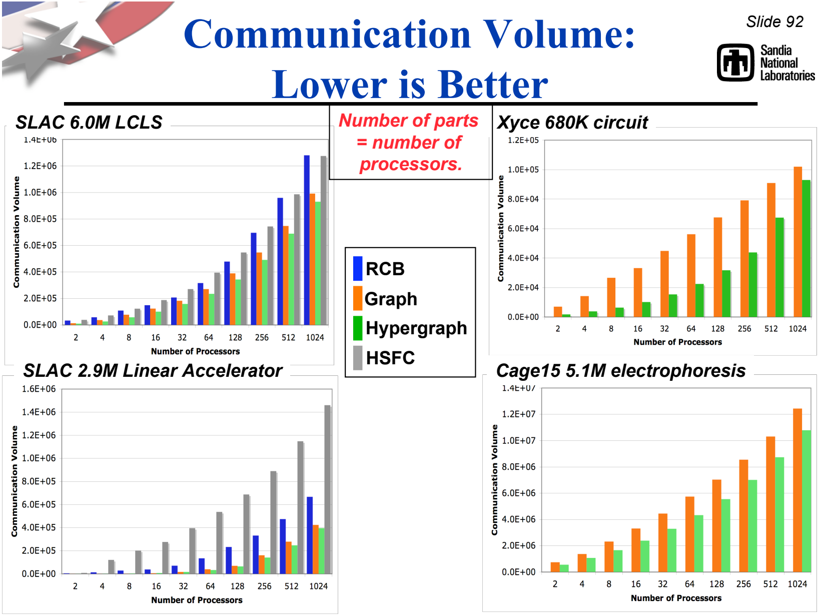


Cage15 DNA
Electrophoresis
5.1M x 5.1M
99M nonzeros



SLAC Linear Accelerator
2.9M x 2.9M
11.4M nonzeros

from Zoltan tutorial slides, by Erik Boman and Karen Devine



Communication Volume: Lower is Better

Slide 92

Sandia National Laboratories

Number of parts = number of processors.

- 🟦 RCB
- 🟧 Graph
- 🟩 Hypergraph
- ▨ HSFC

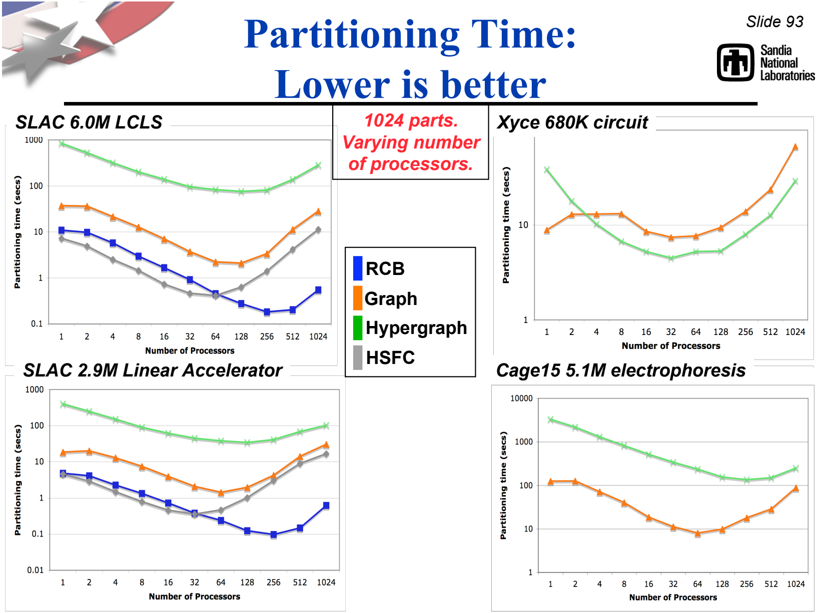
SLAC 6.0M LCLS

Xyce 680K circuit

SLAC 2.9M Linear Accelerator

Cage15 5.1M electrophoresis

from Zoltan tutorial slides, by Erik Boman and Karen Devine



Slide 93

Sandia National Laboratories

Partitioning Time: Lower is better

1024 parts. Varying number of processors.

🟦 RCB
🟧 Graph
🟩 Hypergraph
⬜ HSFC

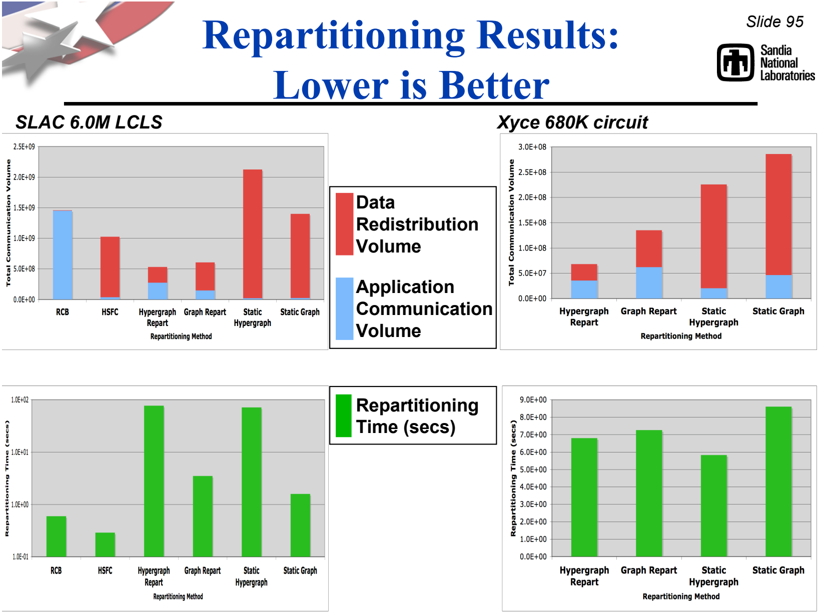
SLAC 6.0M LCLS

Xyce 680K circuit

SLAC 2.9M Linear Accelerator

Cage15 5.1M electrophoresis

from Zoltan tutorial slides, by Erik Boman and Karen Devine



Slide 95

Sandia National Laboratories

Repartitioning Results: Lower is Better

SLAC 6.0M LCLS

Xyce 680K circuit

🟥 Data Redistribution Volume

🟦 Application Communication Volume

🟩 Repartitioning Time (secs)

from Zoltan tutorial slides, by Erik Boman and Karen Devine

References

- *Graph Partitioning for High Performance Scientific Simulations* by K. Schloegel, G. Karypis and V. Kumar. in CRPC Parallel Computing Handbook, (2000). (University of Minnesota TR 0018)
- *Load Balancing Fictions, Falsehoods and Fallacie* by Bruce Hendrickson Applied Math Modelling, (preprint from his website; many other relevant papers there too).
- Zoltan tutorial by E. Boman and K. Devine http://www.cs.sandia.gov/~kddevin/papers/Zoltan_Tutorial_Slides.pdf