SAND2010-5835P

Exploring Feasibility of 2D Sparse Matrix Partitioning: Background

Erik Boman, Karen Devine, *Michael Wolf*
Sandia National Laboratories, NM

HMC CS Clinic Kickoff Meeting
September 2, 2010

cscapes

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000

Outline

- Data Partitioning
- Zoltan
- Matrix Partitioning
- Isorropia
- Project overview

Data Partitioning

- Process of determining how to assign computational objects to parts
  - Parallel computation
  - Objects typically mapped to different processors/cores based on part assignment
- Parts assignments are made to optimize some objective function
  - For instance, runtime
- Why do we care?
  - Interested in large-scale scientific application
  - Hundreds of thousands of cores -> millions of cores
  - Data partitioning necessary for applications to obtain good parallel efficiency on these systems

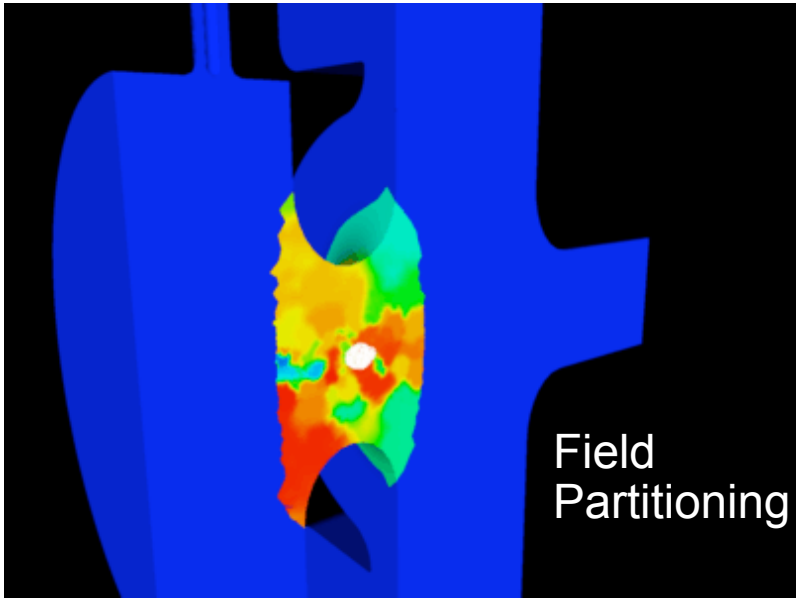
Large-Scale Simulations

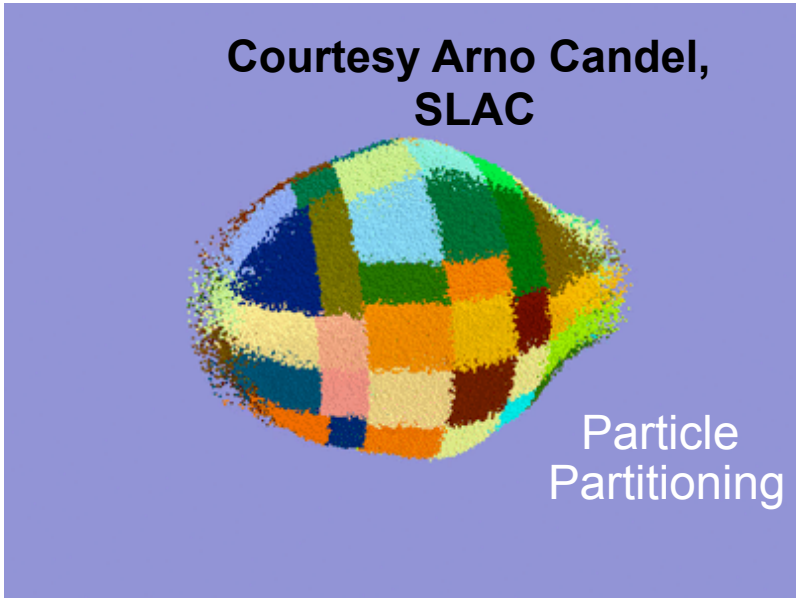
Partitioning Method	Application	Problem Size	Number of Processes	Number of Parts	Architecture	Source
Graph	PHASTA CFD	34M elements	16K	16K	BG/P	Zhou, et al., RPI
Hypergraph	PHASTA CFD	1B elements	4096	160K	Cray XT/5	Zhou, et al., RPI
Hypergraph	Sparta LB algorithms	800M zones	8192	262K	Hera (AMD Quadcore)	Lewis, LLNL
Geometric	Pic3P particle-in-cell	5B particles	24K	24K	Cray XT/4	Candel, et al., SLAC
Geometric	MPSalsa CFD	208M nodes	12K	12K	RedStorm	Lin, SNL
Geometric	Trilinos/ML Multigrid in ALEGRA shock physics	24.6M rows 1.2B non-zeros	24K	24K	RedStorm	Hu, et al., SNL

ComPASS (SLAC)

Enhanced Pic3P accelerator simulation capability with new partitioning scheme

- Pic3P solves Maxwell's equations with moving particles
- Our suggested load balance strategy: Use two different data decompositions
  - Fields partitioned with graph-based methods (ParMETIS)
  - Particles partitioned geometrically (Zoltan RCB 3D)
- Enables solution of larger problems: 24k CPUs, 750M DOFs, 5B particles





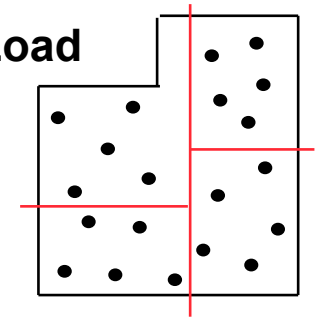
Example: LCLS RF gun, colors indicate distribution to different CPUs
(fields are computed only in causal region, using *p*-refinement)

Sandia National Laboratories

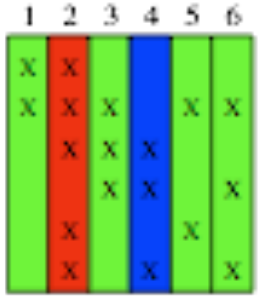
The Zoltan Toolkit

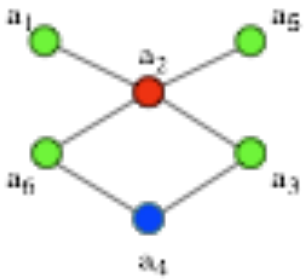
- **Library of data management services for unstructured, dynamic and/or adaptive computations.**

Dynamic Load Balancing

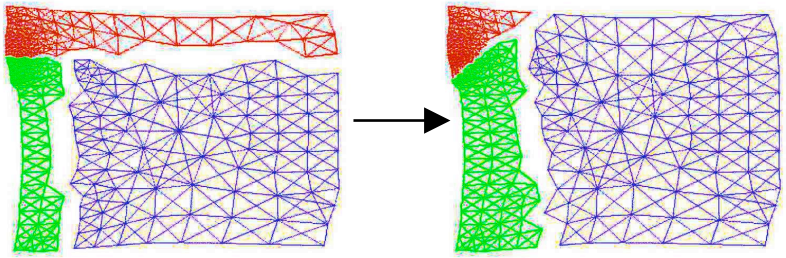


Graph Coloring

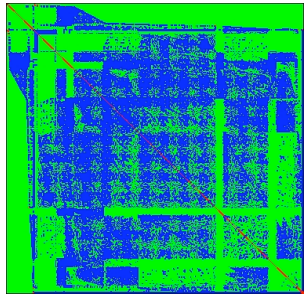




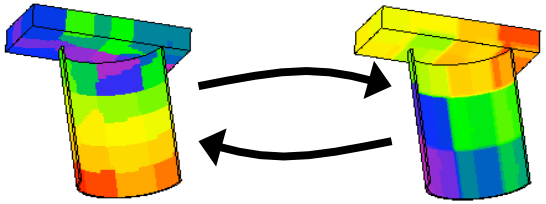
Data Migration



Matrix Ordering



Unstructured Communication



Distributed Data Directories

A	B	C	D	E	F	G	H	I
0	1	0	2	1	0	1	2	1

Zoltan Interface Design

- Common interface to each class of tools
- Tool/method specified with user parameters

- Data-structure neutral design
  - Supports wide range of applications and data structures
  - Imposes no restrictions on application's data structures
  - Application does not have to build Zoltan's data structures.

Zoltan Application Interface

- Simple, easy-to-use interface
  - Small number of callable Zoltan functions
  - Callable from C, C++, Fortran
- Requirement: Unique global IDs for objects to be partitioned/ordered/colored.
- Application interface:
  - Zoltan queries the application for needed info.
    - IDs of objects, coordinates, relationships to other objects.
  - Application provides simple functions to answer queries.
  - Small extra costs in memory and function-call overhead.
- Once query functions are implemented, application can access all Zoltan functionality.
  - Can switch between algorithms by setting parameters.

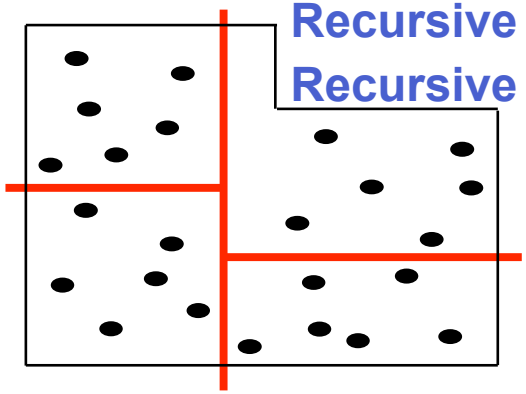
Zoltan Toolkit: Suite of Partitioners

- No single partitioner works best for all applications.
  - Trade-offs:
    - Quality vs. speed.
    - Geometric locality vs. data dependencies.
    - High-data movement costs vs. tolerance for remapping.
- Application developers may not know which partitioner is best for application.

- Zoltan contains suite of partitioning methods.
  - Application changes only one parameter to switch methods.
    - Zoltan_Set_Param(zz, “LB_METHOD”, “new_method_name”);
  - Allows experimentation/comparisons to find most effective partitioner for application.

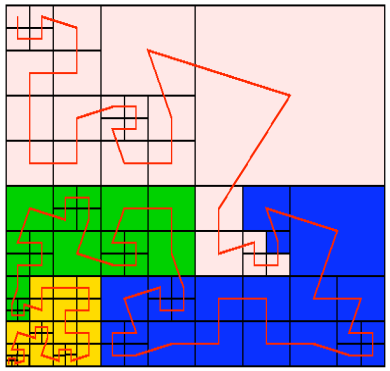
Partitioning Algorithms in Zoltan

Geometric (coordinate-based) methods



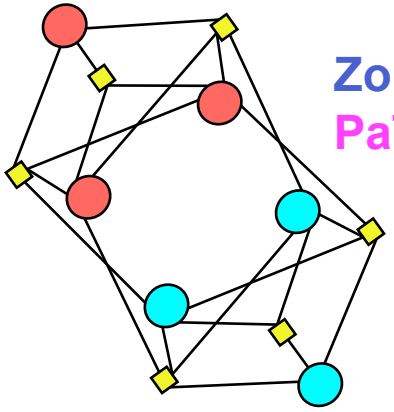
Recursive Coordinate Bisection (Berger, Bokhari)

Recursive Inertial Bisection (Taylor, Nour-Omid)

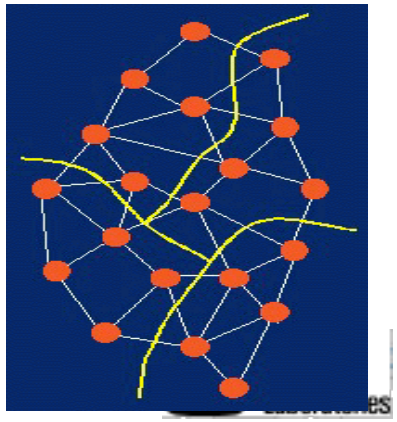


Space Filling Curve Partitioning
(Warren&Salmon, et al.)

Combinatorial (topology-based) methods



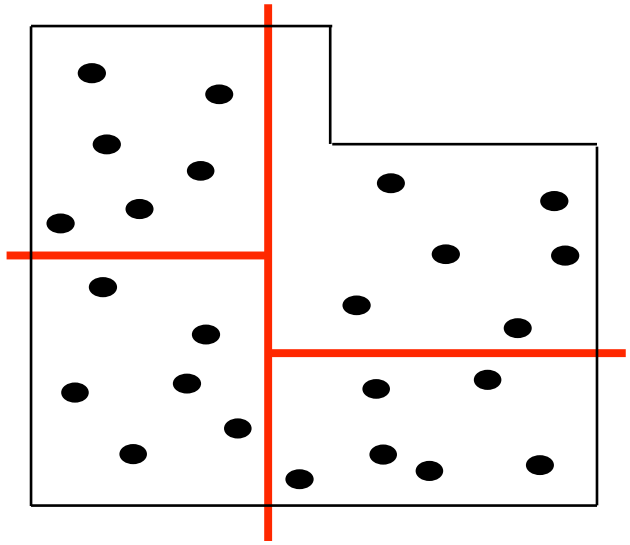
Zoltan Hypergraph Partitioning (PHG)
PaToH (Catalyurek & Aykanat)



Zoltan Graph Partitioning (PHG)
ParMETIS (Karypis, et al.)
PT-Scotch (Pellegrini, et al.)

Geometric Partitioning

- Zoltan_Set_Param(zz, “LB_METHOD”, “RCB”);
  Zoltan_Set_Param(zz, “LB_METHOD”, “RIB”);
  Zoltan_Set_Param(zz, “LB_METHOD”, “HSFC”);
- Partition based on geometric locality.
  - Assign physically close objects to the same processor.



Recursive Coordinate Bisection (RCB)
Berger & Bokhari, 1987

Geometric Methods

- Advantages:
  - Easiest partitioners to use.
  - 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.

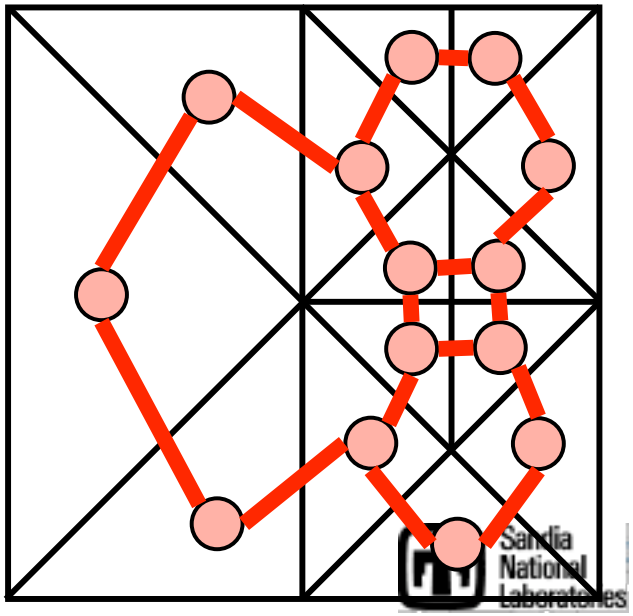
SLAC'S 55-cell Linear Accelerator with couplers: One-dimensional RCB partition reduced runtime up to 68% on 512 processor IBM SP3. (Wolf, Ko)

- Disadvantages:
  - No explicit control of communication volume.
  - Mediocre partition quality (in terms of volume).
  - Can generate disconnected subdomains for complex geometries.
  - Need coordinate information.

Graph Partitioning

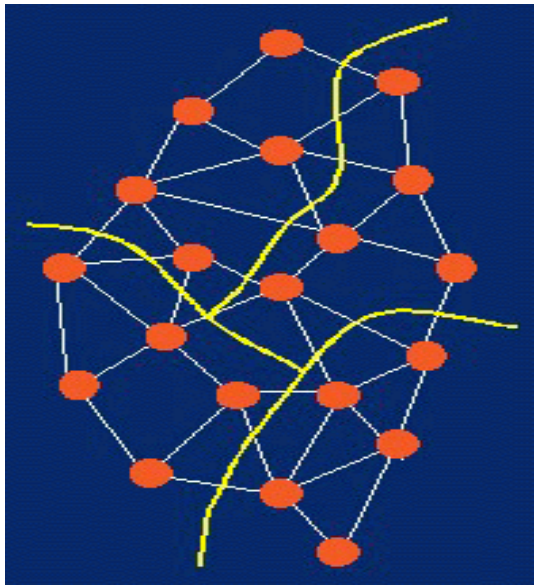
- **Zoltan_Set_Param(zz, "LB_METHOD", "GRAPH");**
- **Zoltan_Set_Param(zz, "GRAPH_PACKAGE", "PHG");** **or**
  Zoltan_Set_Param(zz, "GRAPH_PACKAGE", "PARMETIS"); **or**
  Zoltan_Set_Param(zz, "GRAPH_PACKAGE", "SCOTCH");

- **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

- 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.)
      Party (U. Paderborn)
      Scotch (U. Bordeaux)
    - Parallel: Zoltan (SNL)
      ParMETIS (U. Minn.)
      PJostle (U. Greenwich)
      PTScotch (U. Bordeaux)

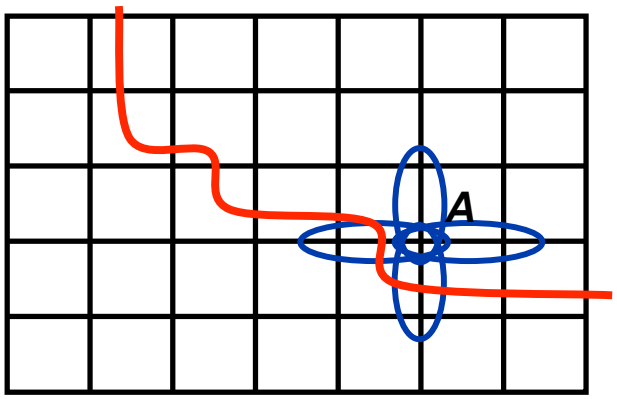


- Disadvantages:
  - More expensive than geometric methods.
  - Edge-cut model only approximates communication volume.

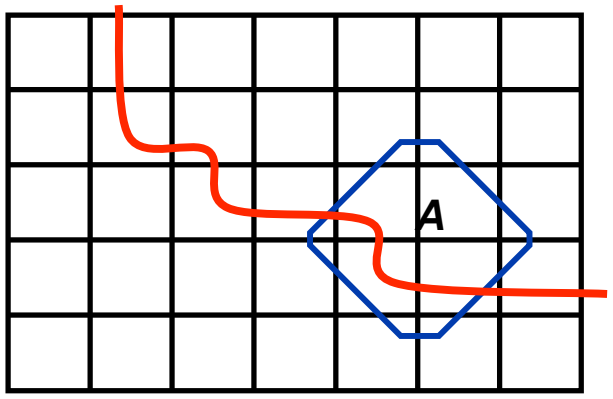
Hypergraph Partitioning

- Zoltan_Set_Param(zz, “LB_METHOD”, “HYPERGRAPH”);
- Zoltan_Set_Param(zz, “HYPERGRAPH_PACKAGE”, “PHG”); or
  Zoltan_Set_Param(zz, “HYPERGRAPH_PACKAGE”, “PATOH”);

- 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

Sandia National Laboratories

Hypergraph Partitioning

- 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:
  - Usually more expensive than graph partitioning

Sparse Matrix Partitioning

Parallel Matrix-Vector Multiplication

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} = \begin{bmatrix} 1 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 1 & 9 & 0 & 5 & 0 & 0 & 0 \\ 8 & 0 & 1 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 1 & 8 & 0 & 0 \\ 0 & 4 & 0 & 0 & 3 & 1 & 3 & 0 \\ 0 & 0 & 0 & 6 & 0 & 9 & 1 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}$$

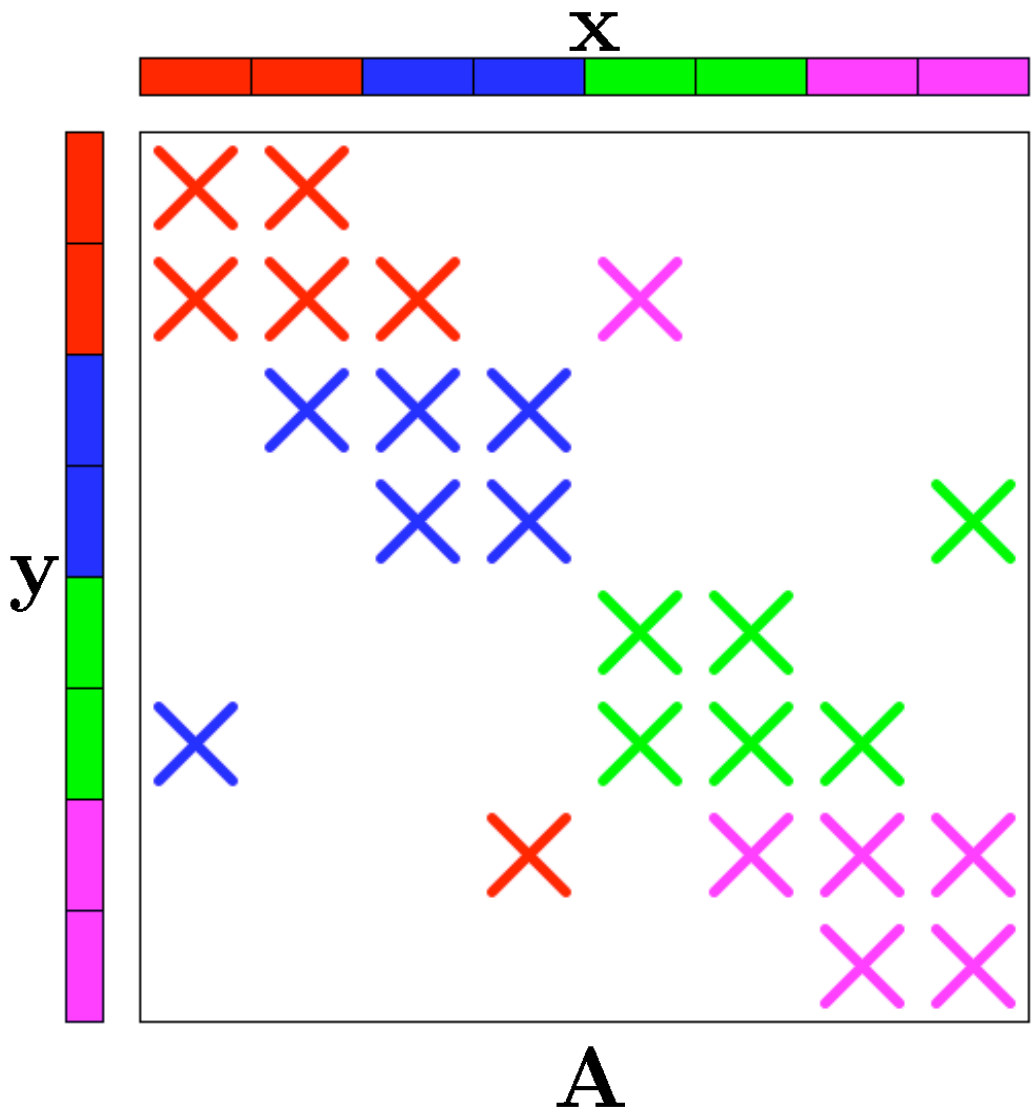
$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

- Vectors partitioned identically

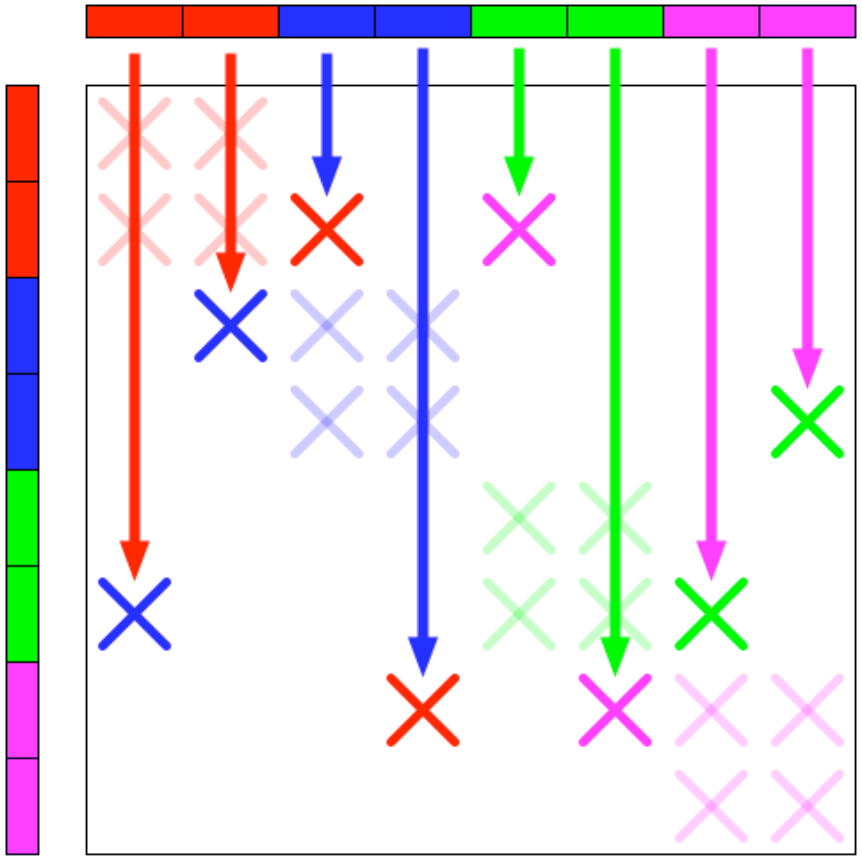
Objective

- Ideally we minimize total run-time
- Settle for easier objective
  - Work balanced
  - Minimize total communication volume
- Can partition matrices in different ways
  - 1-D
  - 2-D
- Can model communication in different ways
  - Graph
  - Bipartite graph
  - Hypergraph

Parallel Matrix-Vector Multiplication



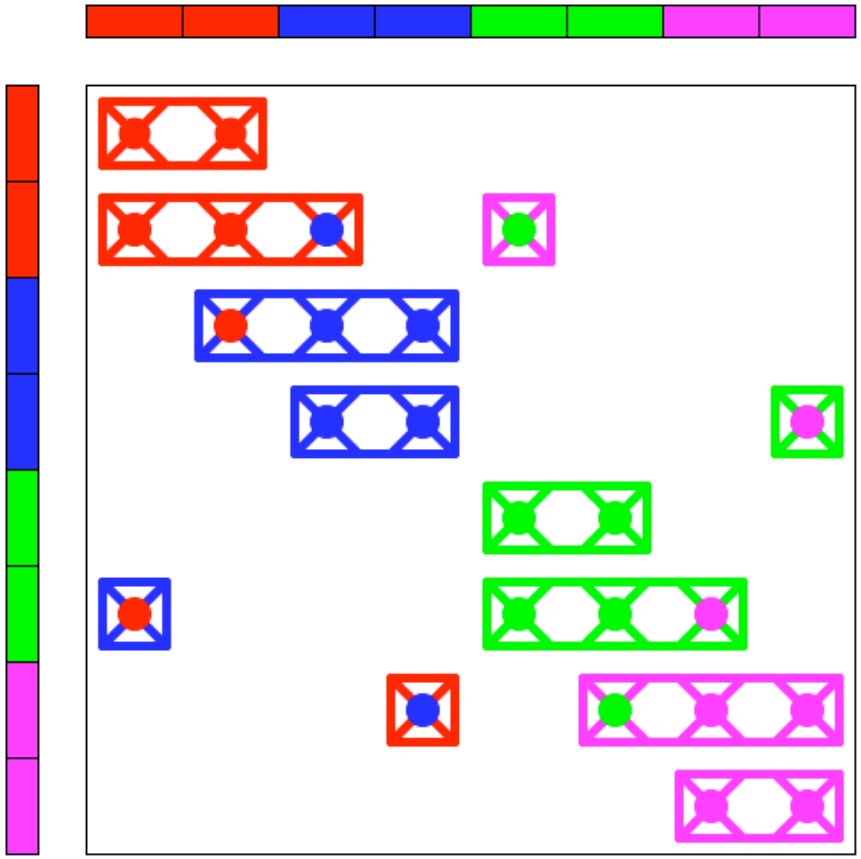
Parallel Matrix-Vector Multiplication Stage 1



"fan-out"

- x_j sent to remote processes that have nonzeros in column j

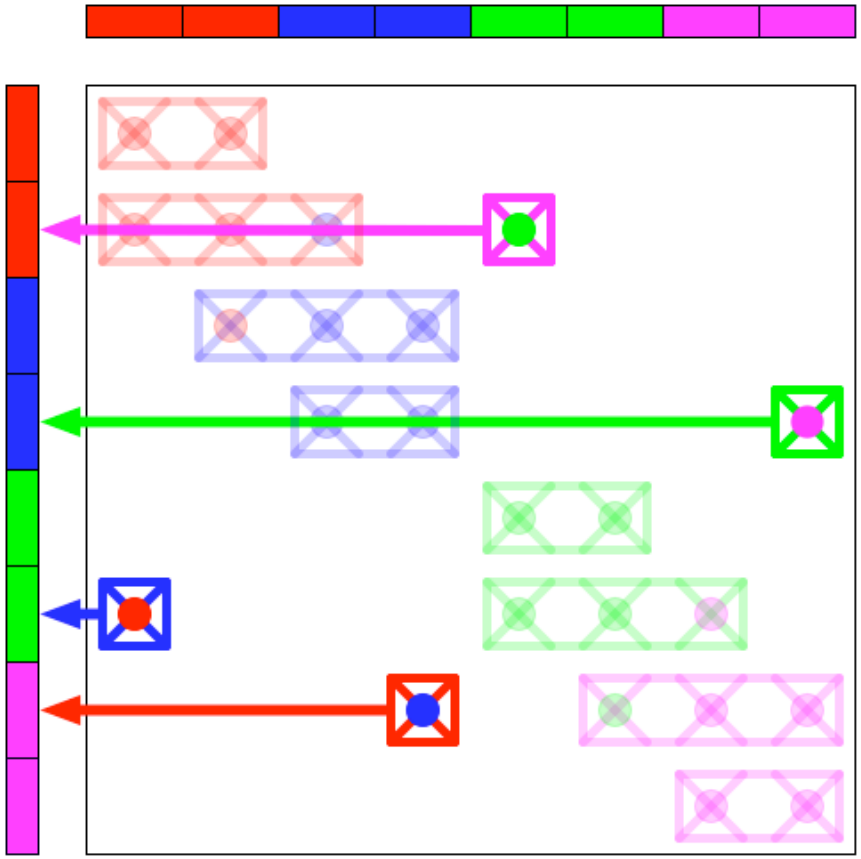
Parallel Matrix-Vector Multiplication Stage 2



$$y_i = \sum a_{ij} x_j,$$
$$\forall i, j : a_{ij} \neq 0$$

- Local partial inner-products

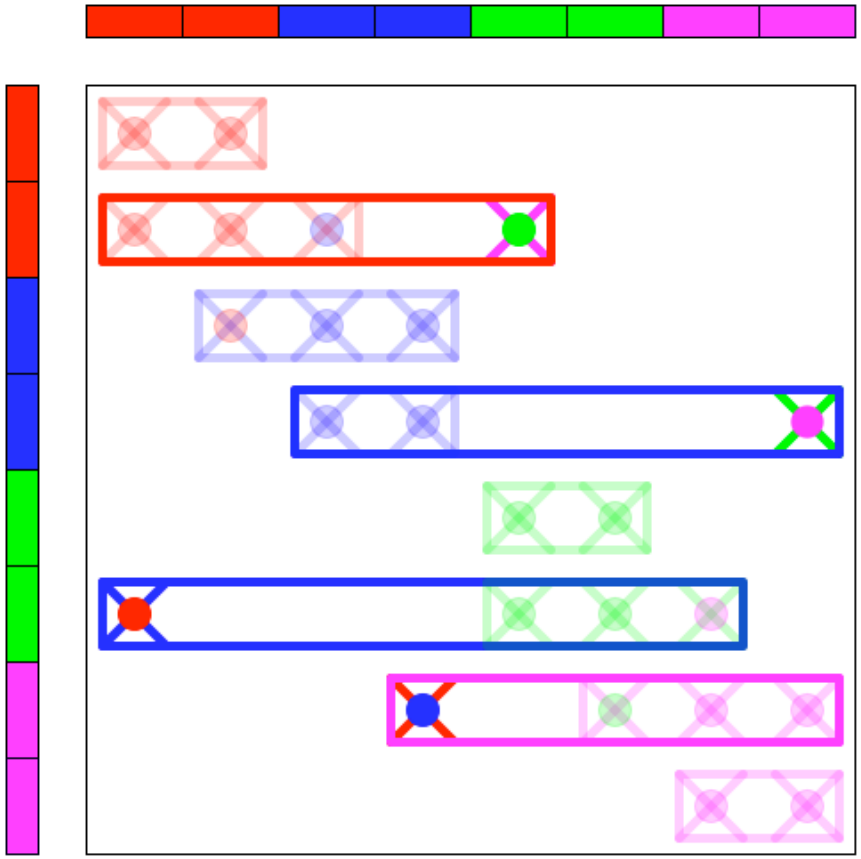
Parallel Matrix-Vector Multiplication Stage 3



"fan-in"

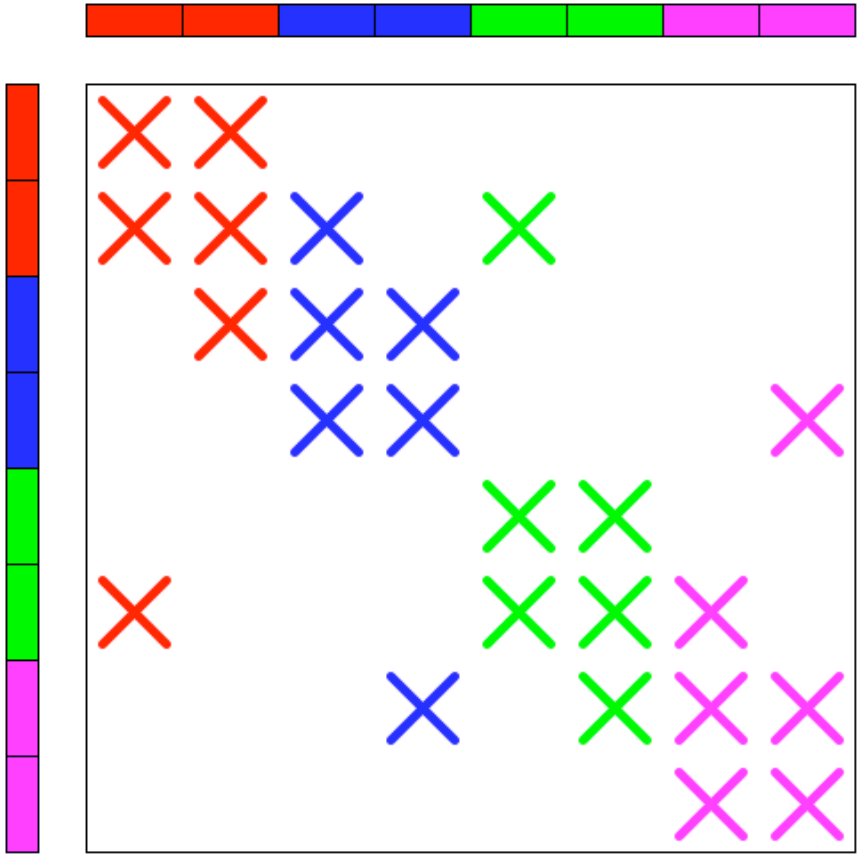
• Send partial inner-products to process that owns corresponding vector element y_i

Parallel Matrix-Vector Multiplication Stage 4



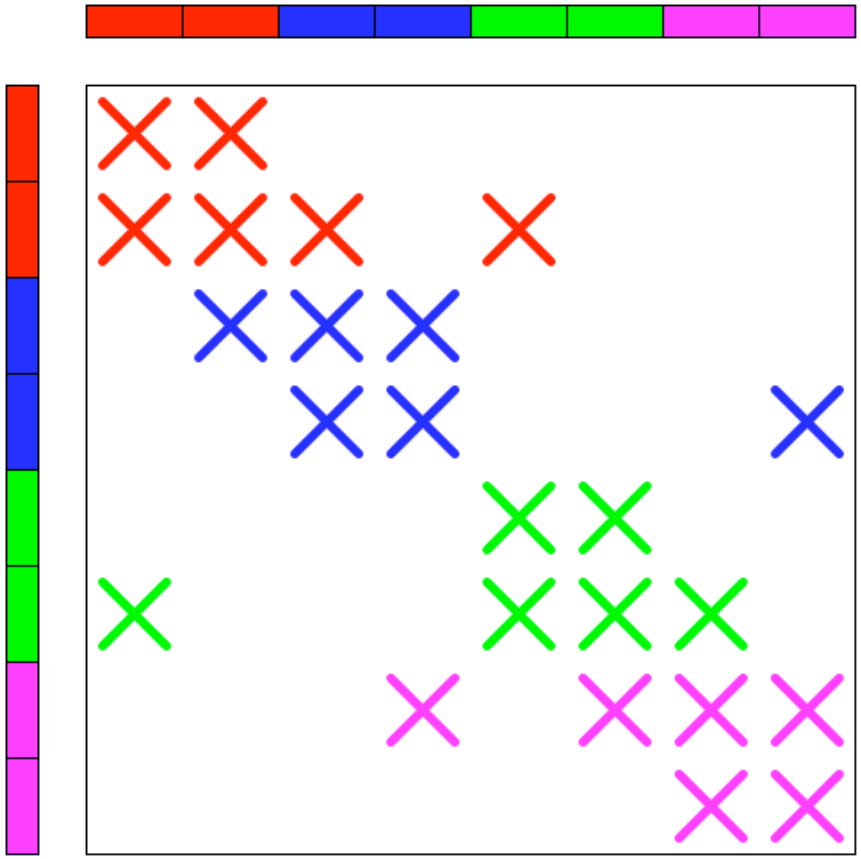
- Accumulate partial inner-products to obtain complete resulting vector

1-D Column Partitioning



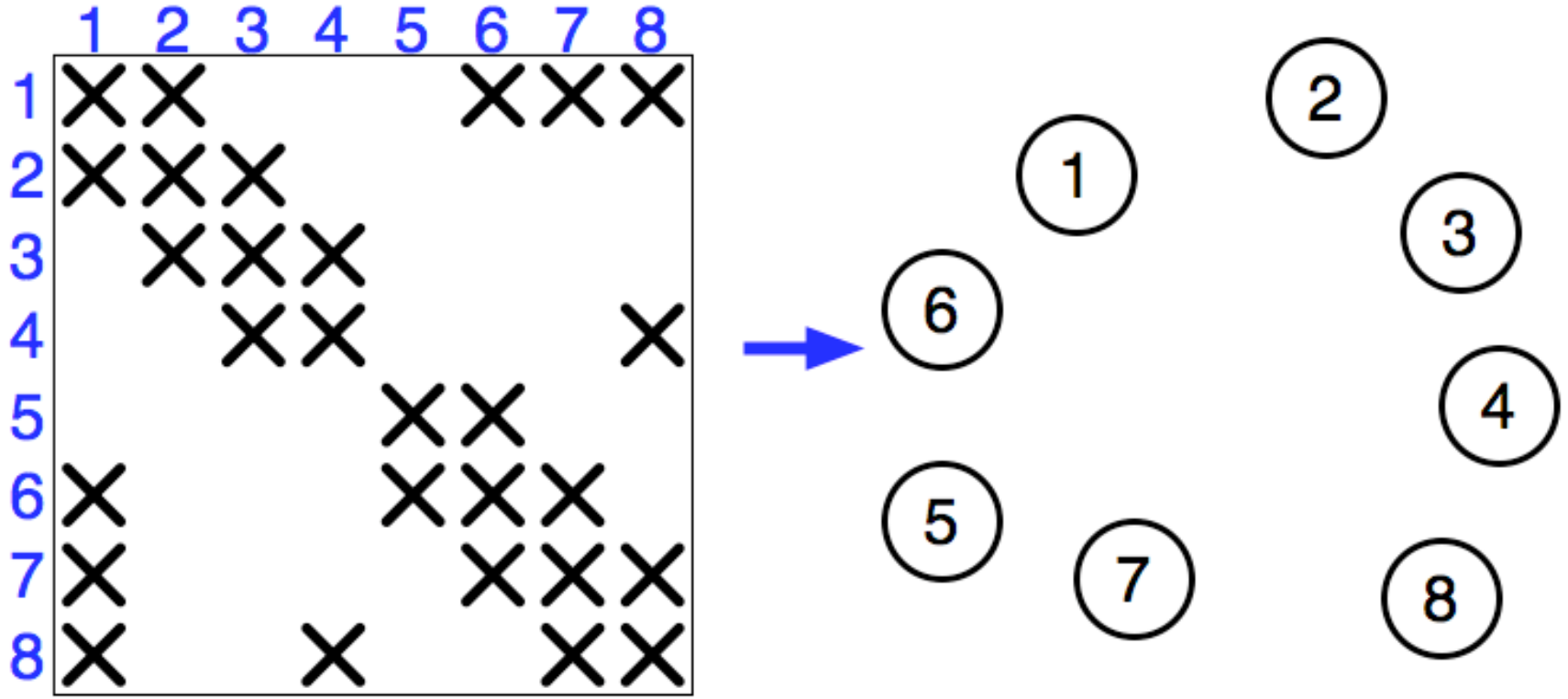
- Each process assigned nonzeros for set of columns

1-D Row Partitioning



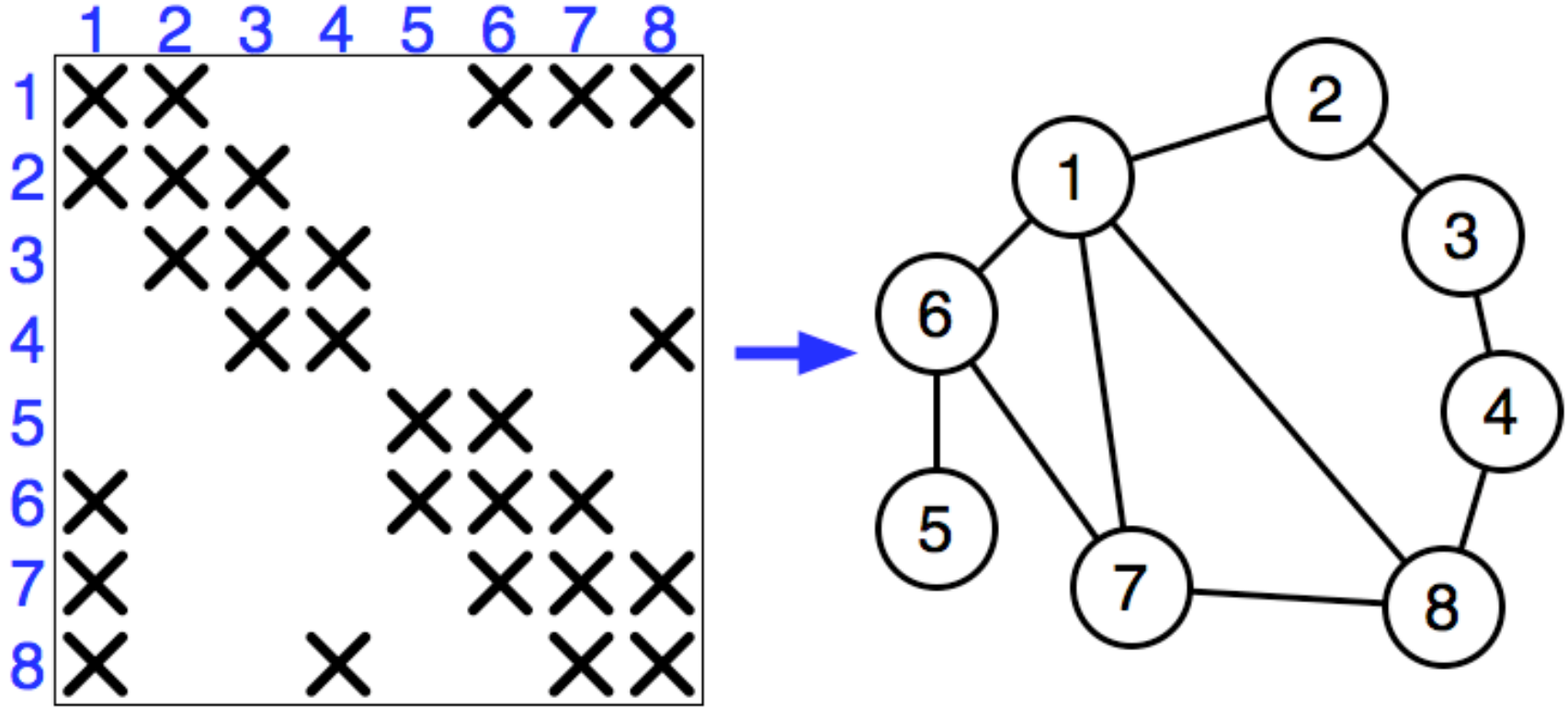
- Each process assigned nonzeros for set of rows

Graph Model of 1-D Partitioning



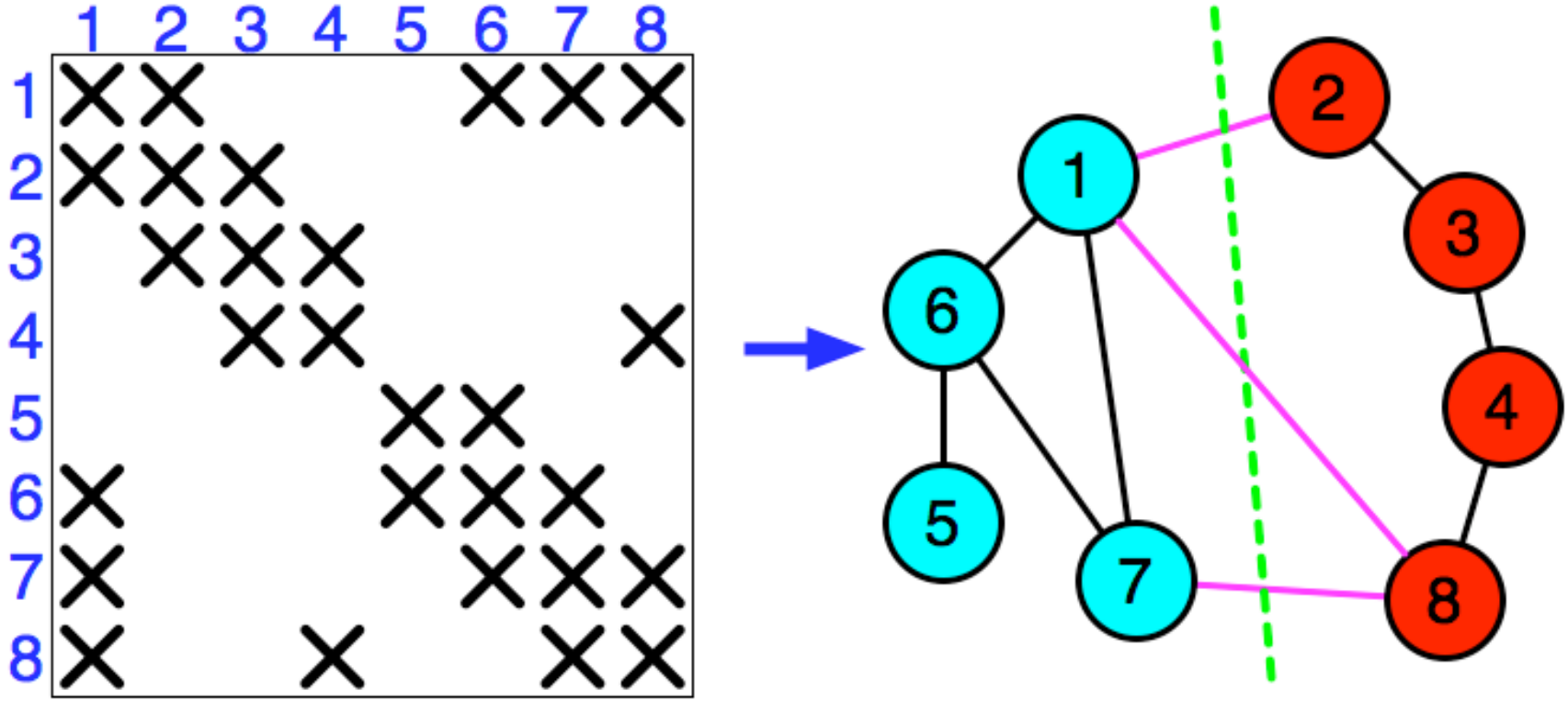
- Each row or column represented by graph vertex
  - Weighted by number of nonzeros in row/column

Graph Model of 1-D Partitioning



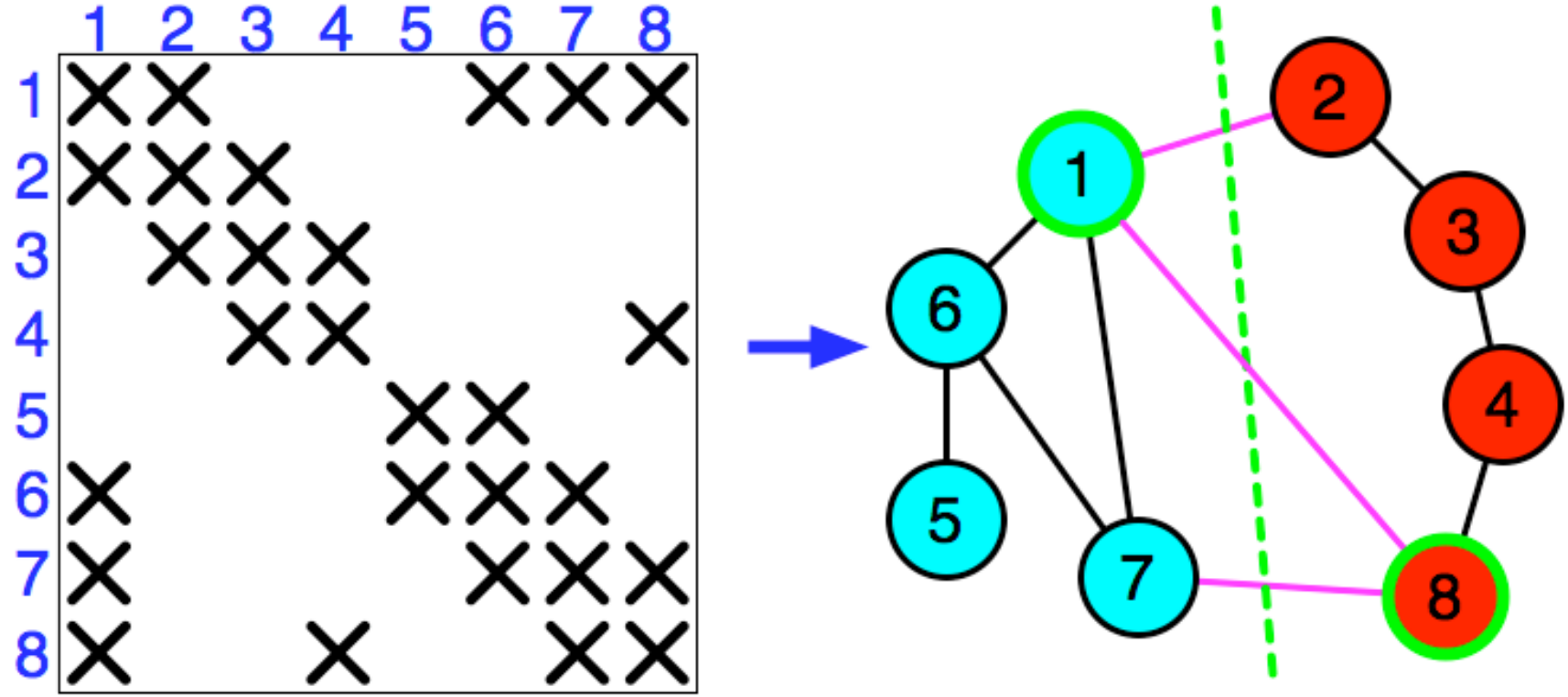
- Nonzeros represented by edges between 2 vertices (corresponding to nonzero row, col)

Graph Model of 1-D Partitioning



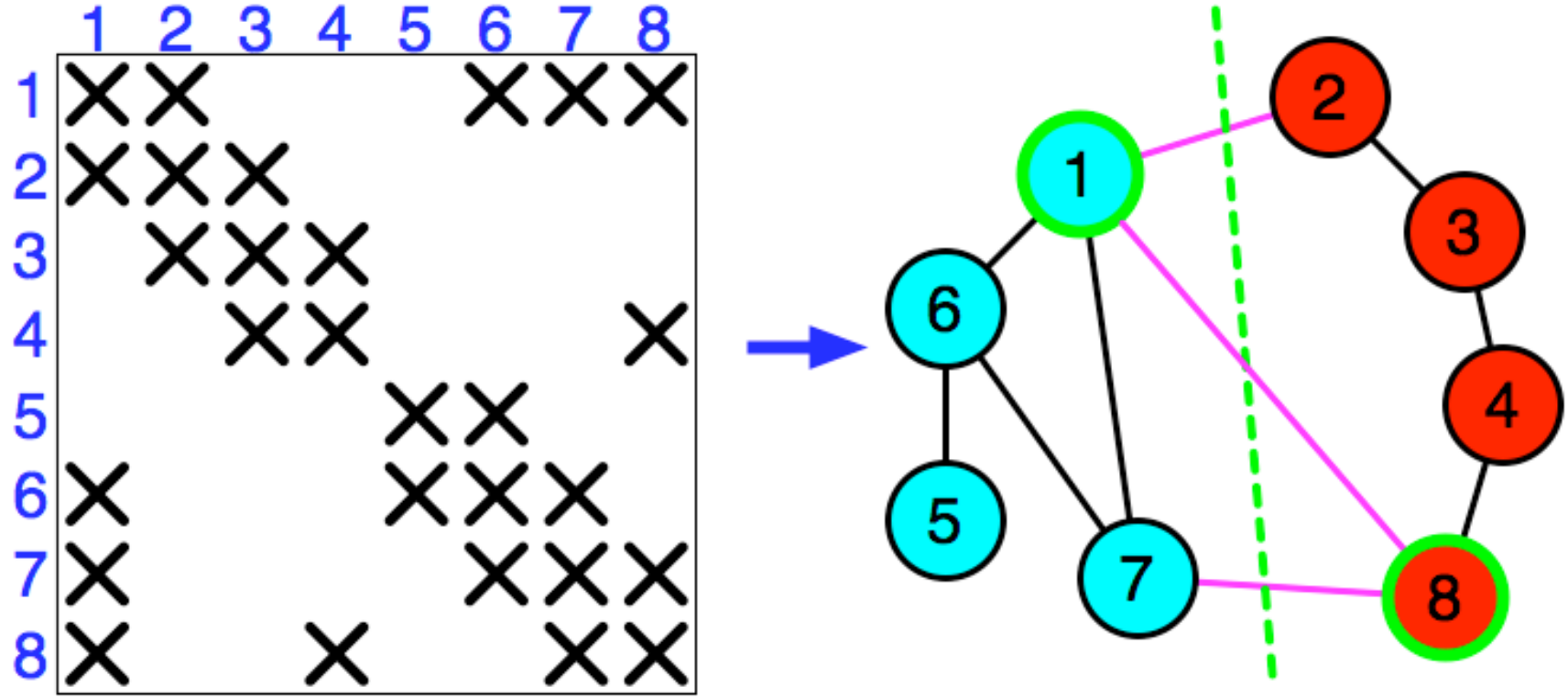
- Partition into k equal sets
  - Such that number of cut edges is minimized

Graph Model Shortcomings



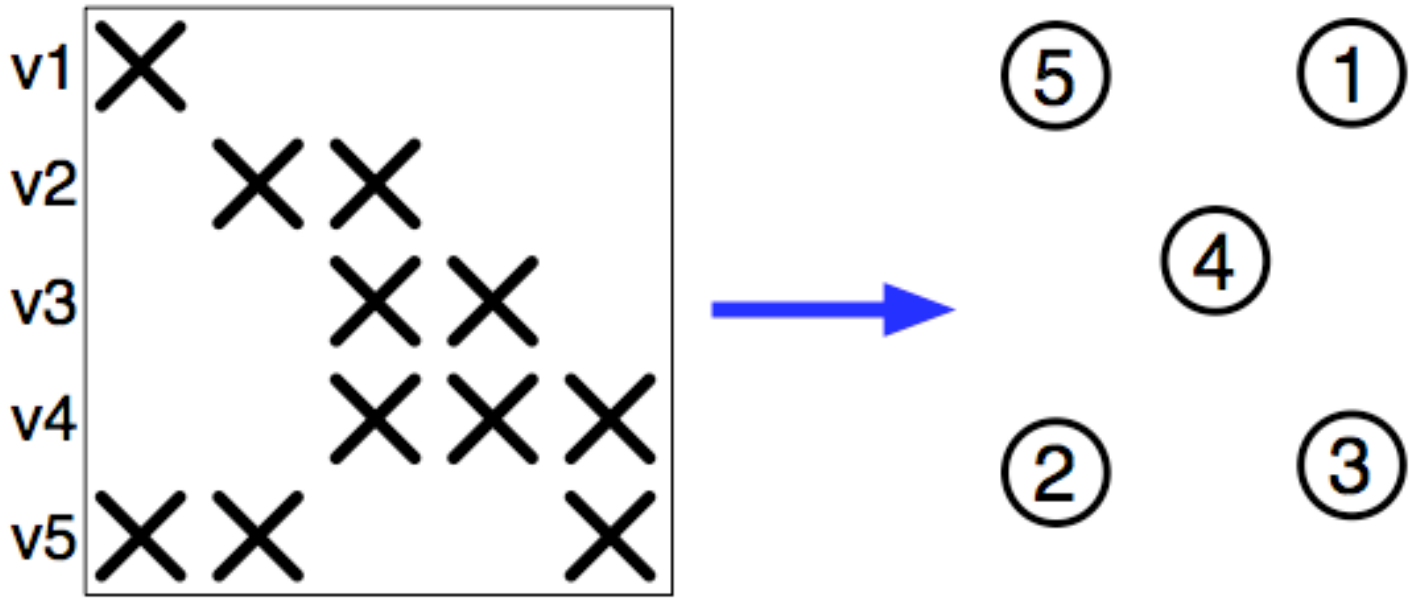
- Inaccurate approximation of communication volume
  - Approximate volume: 6
  - Actual volume: 4

Graph Model Shortcomings



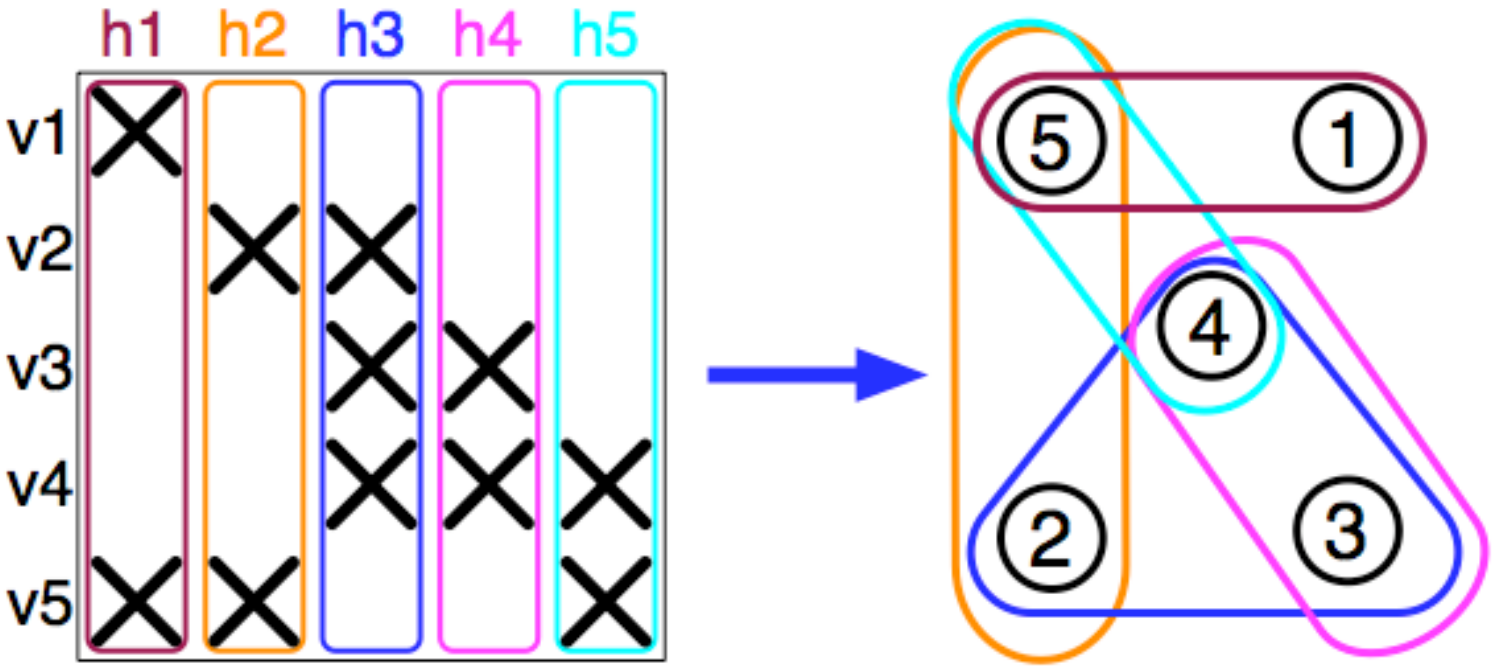
- Requires symmetric nonzero pattern
- NP-hard to solve optimally

Hypergraph Model of 1-D (Row) Partitioning



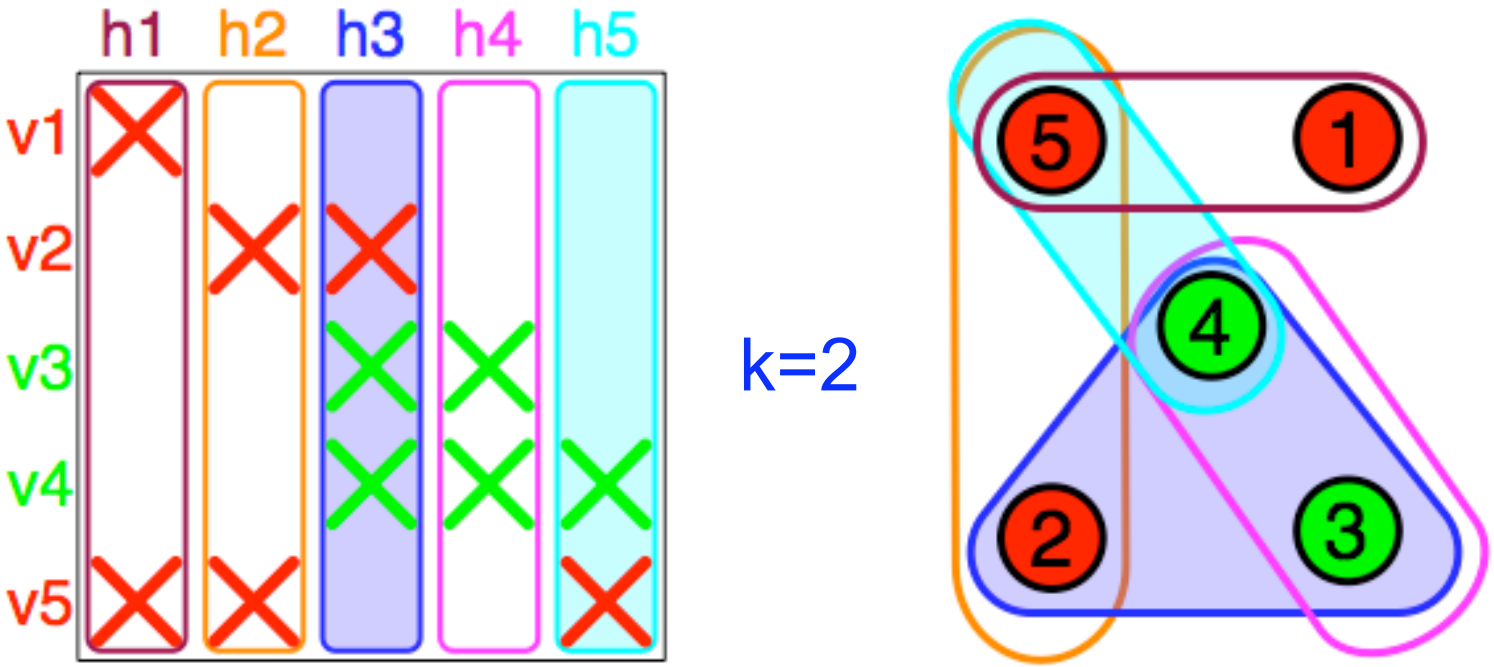
- Nonzero pattern can be unsymmetric
- Rows represented by vertices in hypergraph
  - Weighted by number of nonzeros in row

Hypergraph Model of 1-D (Row) Partitioning



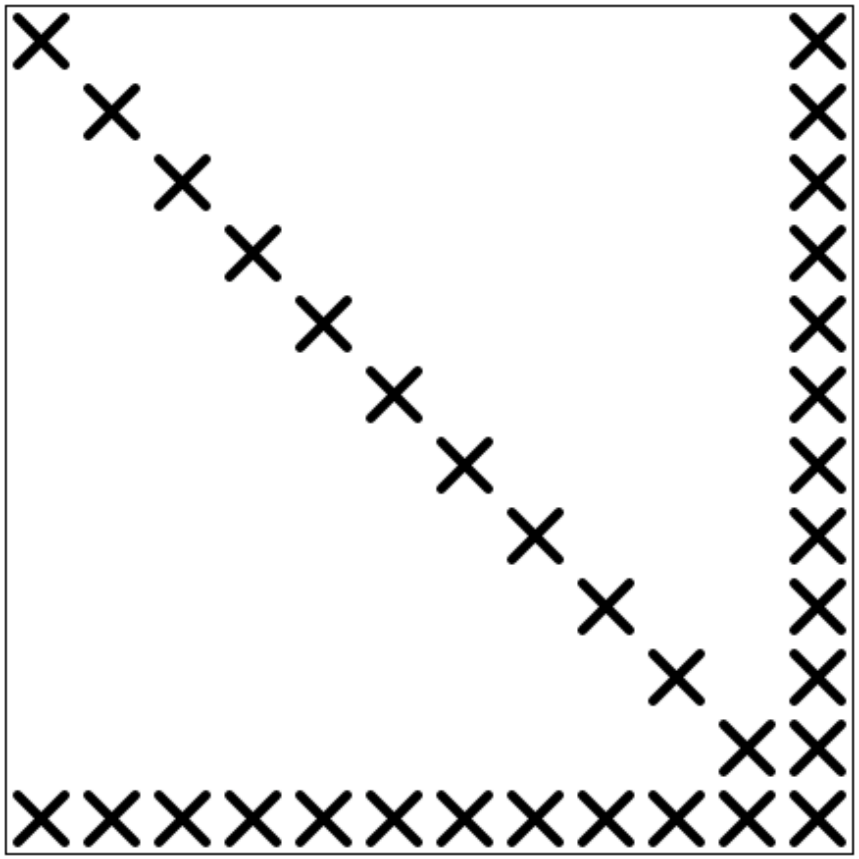
- Columns represented by hyperedges in hypergraph

Hypergraph Model of 1-D (Row) Partitioning



- Partition vertices into k equal sets
- Hyperedge cut = communication volume
  - Aykanat and Catalyurek (1996)
- NP-hard to solve optimally

When 1-D Partitioning is Inadequate

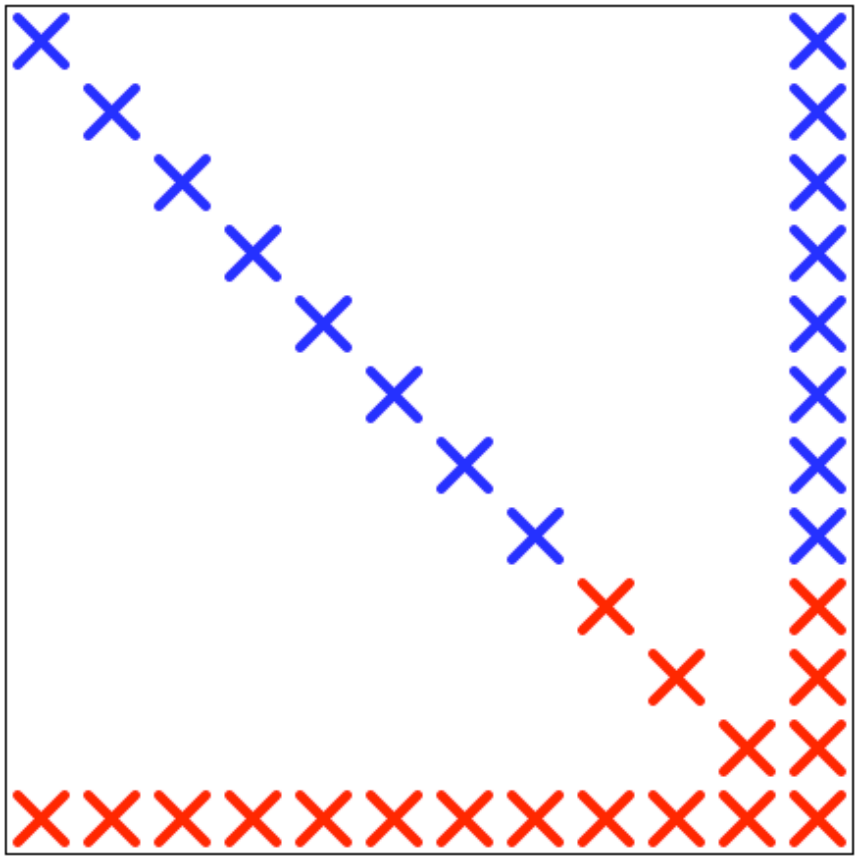


"Arrowhead" matrix

n=12

nnz=30

When 1-D Partitioning is Inadequate



n=12

nnz=30

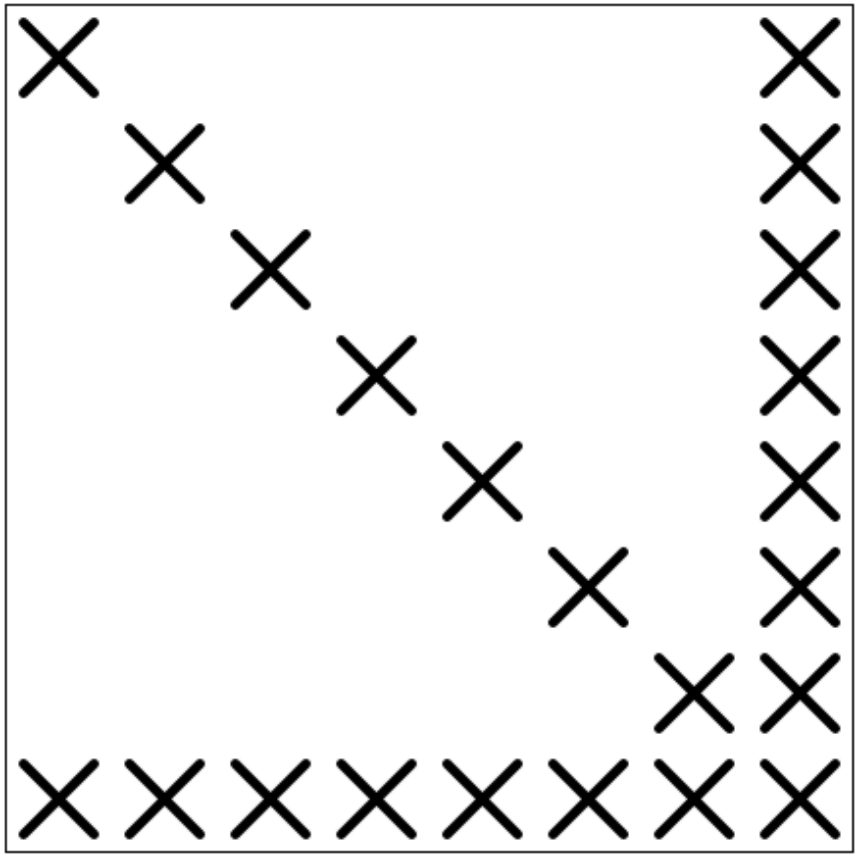
volume = 9

- For nxn matrix for any 1-D bisection:
  - nnz = 3n-2
  - Volume ≈ 3/4*n

2-D Partitioning Methods

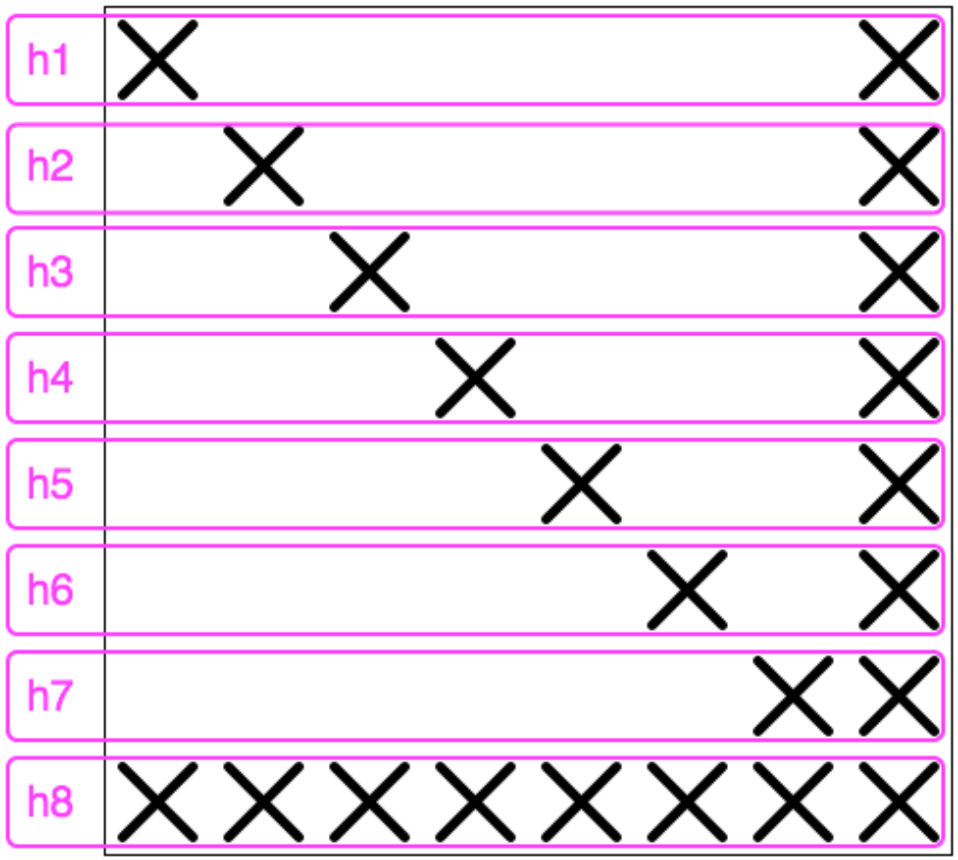
- More flexibility
- Yield lower communication volume for many problems

2-D Method: Fine-Grain Hypergraph Model



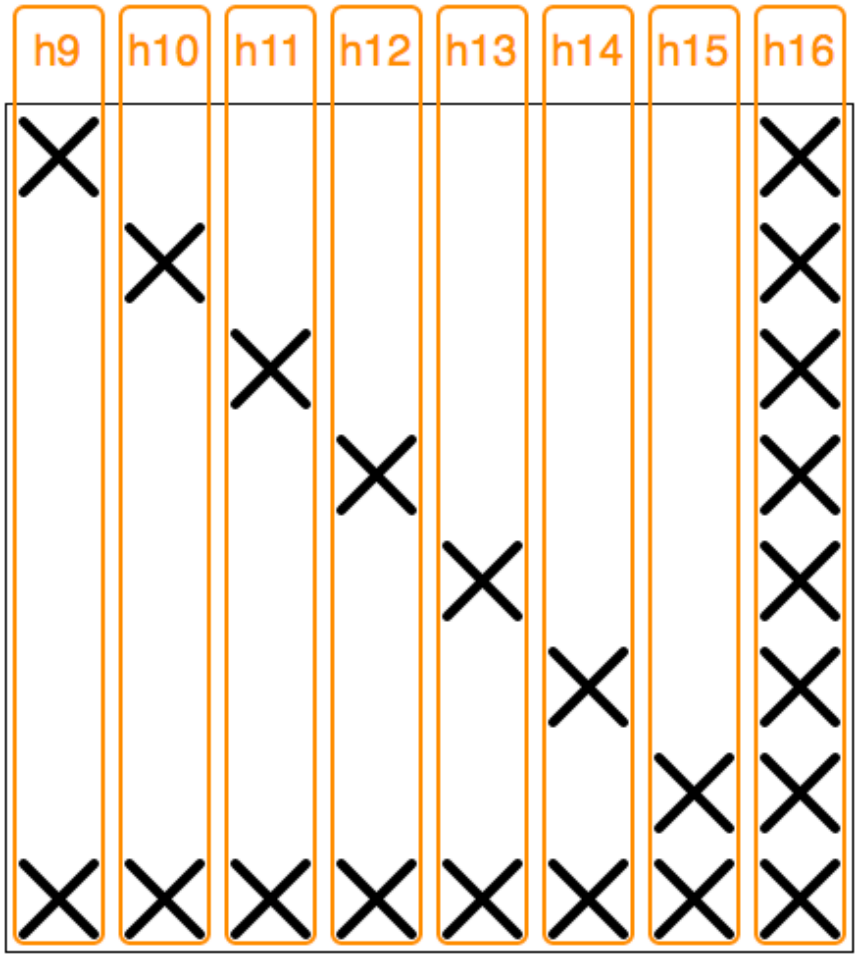
- Catalyurek and Aykanat (2001)
- Assign each nz separately
- Nonzeros represented by vertices in hypergraph

2-D Method: Fine-Grain Hypergraph Model



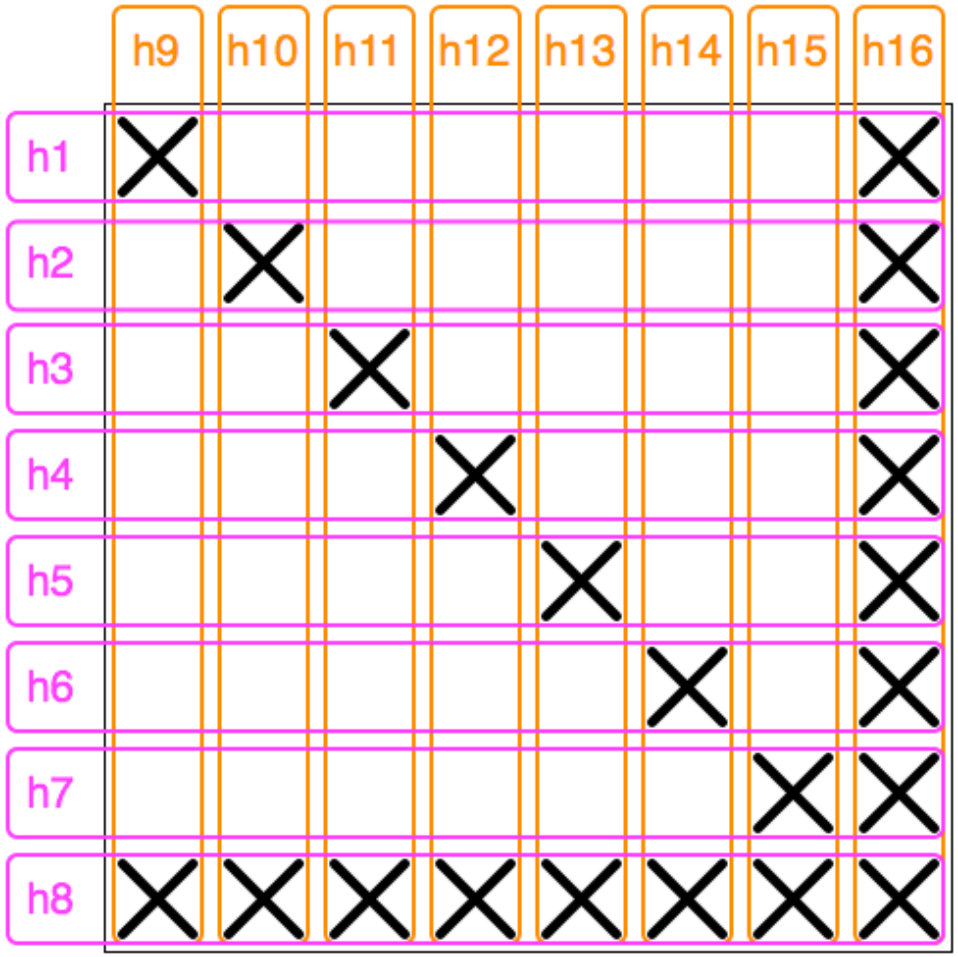
- Rows represented by hyperedges

2-D Method: Fine-Grain Hypergraph Model



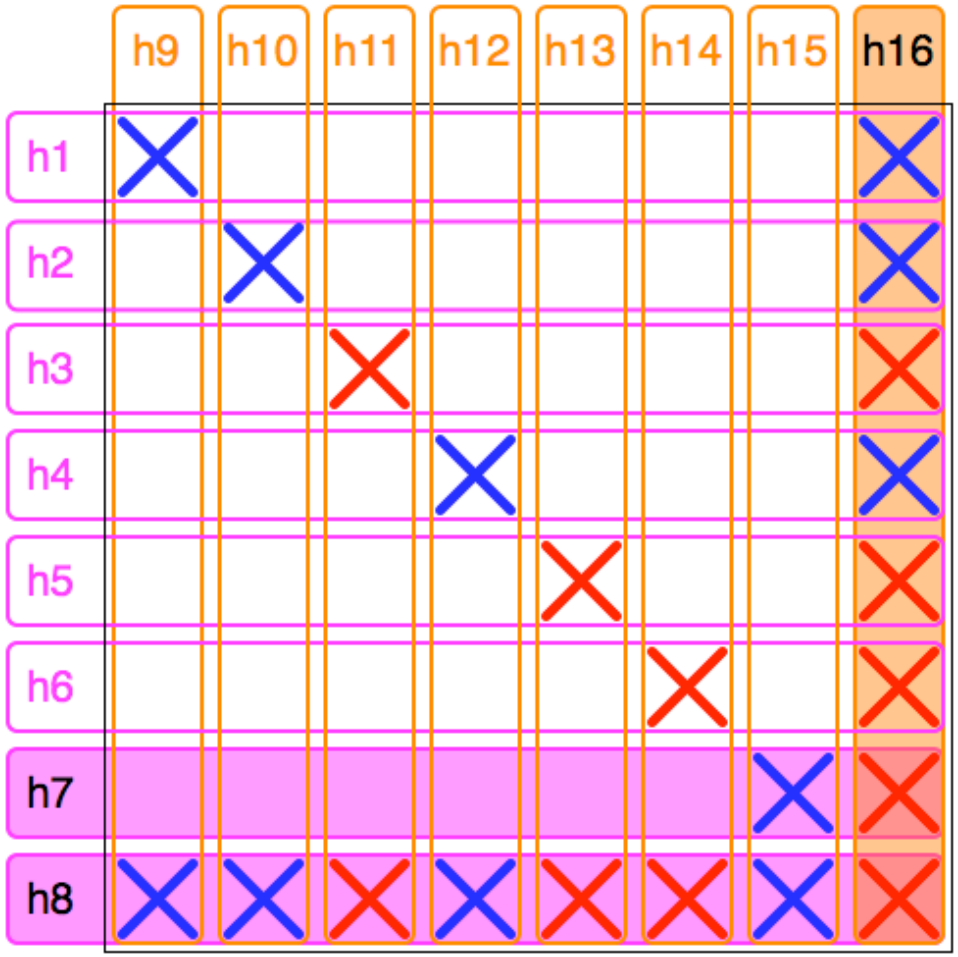
- Columns represented by hyperedges

2-D Method: Fine-Grain Hypergraph Model



- 2n hyperedges

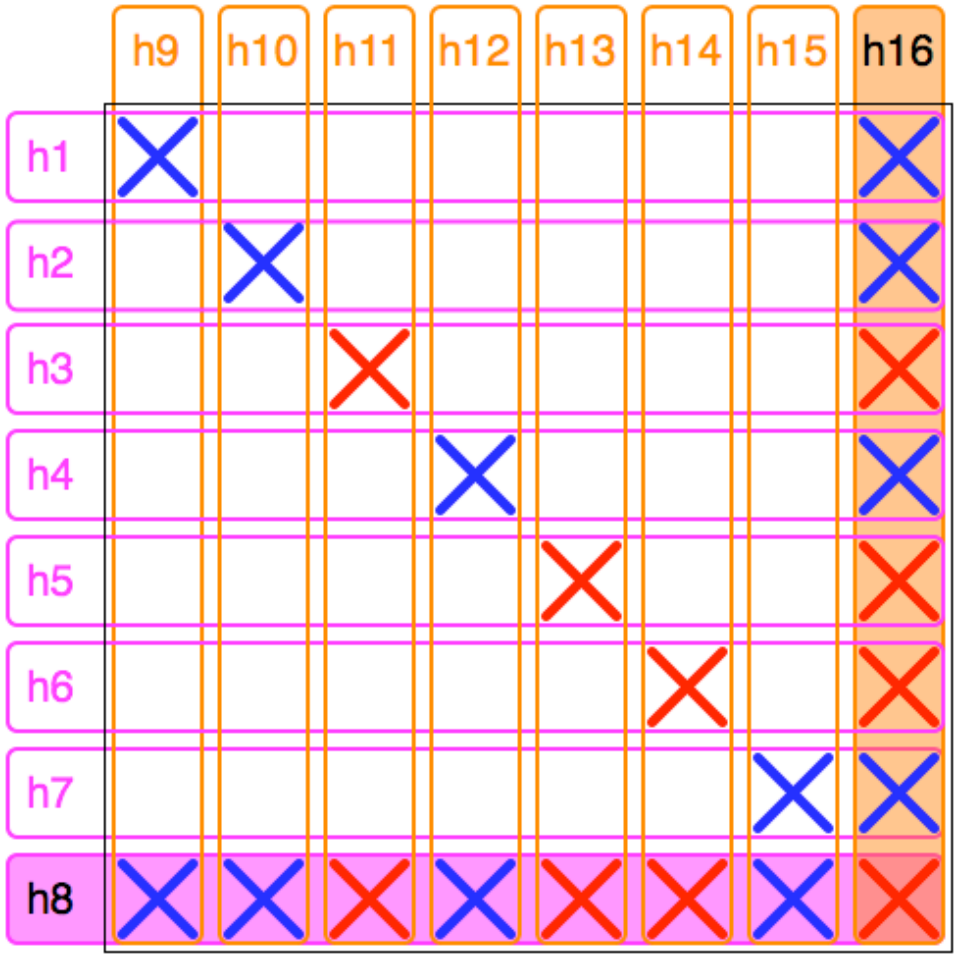
2-D Method: Fine-Grain Hypergraph Model



k=2, volume = 3

- Partition vertices into k equal sets
- Volume = hypergraph cut
- Minimum volume partitioning when optimally solved
- Larger NP-hard problem

2-D Method: Fine-Grain Hypergraph Model



Volume = 2

- Loosening load-balancing restriction we can obtain minimum cut (for non-trivial partitioning)

Trilinos and Isorropia

- Trilinos (M. Heroux, SNL, PI)
  - Framework for solving large-scale scientific problems
  - Focus on packages (independent pieces of software that are combined to solve these problems)
- Isorropia
  - Trilinos package for combinatorial scientific computing
  - Partitioning, coloring, ordering algorithms applied to Epetra matrices
  - Utilizes many algorithms in Zoltan
  - “Zoltan for sparse matrices”
- Partitioning methods
  - 1D linear/block, cyclic, random
  - 1D hypergraph
  - 1D graph
  - 2D fine-grain hypergraph

Sandia National Laboratories

Isorropia Partitioning: Example 1

```
using Isorropia::Epetra::Partitioner;

ParameterList params;
params.set("PARTITIONING_METHOD", "HYPERGRAPH");
params.set("BALANCE_OBJECTIVE","NONZEROS");
params.set("IMBALANCE_TOL","1.03");

// rowmatrix is an Epetra_RowMatrix
Partitioner partitioner(rowmatrix,params,false);
partitioner.partition();
```

- Simple partitioning of rowmatrix
  - 1D row hypergraph partitioning
  - Balancing number of nonzeros
  - Load imbalance tolerance of 1.03

Isorropia Partitioning: Example 2

```
using Isorropia::Epetra::Partitioner2D;

ParameterList params;
params.set("PARTITIONING_METHOD", "HGRAPH2D_FINEGRAIN");
params.set("IMBALANCE_TOL","1.03");

// rowmatrix is an Epetra_RowMatrix
Partitioner2D partitioner(rowmatrix, params, false);
partitioner.partition();
```

- 2D partitioning of rowmatrix
  - 2D fine-grain hypergraph partitioning
  - Balancing number of nonzeros (implicit)
  - Load imbalance tolerance of 1.03

Isorropia: Redistributing Matrix Data

```
partitioner->partition();

// Set up Redistributor based on partition
Isorropia::Epetra::Redistributor rd(partitioner);

// Redistribute data
newmatrix = rd.redistribute(*rowmatrix,true);
```

- After partitioning matrix
  - Build Redistributor from new partition
  - Redistribute data based on new partition
  - Obtain new matrix

Isorropia: Redistributing Matrix Data

```
using Isorropia::Epetra::createBalancedCopy;

ParameterList params;
params.set("IMBALANCE_TOL","1.03");
params.set("BALANCE_OBJECTIVE","NONZEROS");
params.set("PARTITIONING_METHOD", "HYPERGRAPH");

// crsmatrix and newmatrix are Epetra_CrsMatrix
newmatrix = createBalancedCopy(*crsmatrix, params);
```

- Shortcut
  - Combines partitioning/redistibution of data

HMC Clinic Project

- Implement partition evaluation functionality
- Python interface to Isorropia
  - Graphical capabilities for visualizing the matrix partition
- Implement 2D partitioning algorithm(s)
- Implement vector partitioning algorithm(s)
- Empirical study comparing 1D and 2D partitioning method

For More Information...

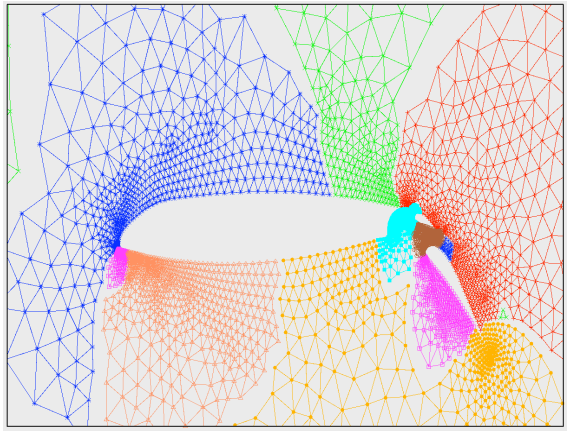
- Zoltan Home Page
  - http://www.cs.sandia.gov/Zoltan
  - User's and Developer's Guides
  - Tutorial: "Getting Started with Zoltan: A Short Tutorial"
  - Download Zoltan software under GNU LGPL
- Trilinos Home Page
  - http://trilinos.sandia.gov
- CSCAPES Home Page
  - http://www.cscapes.org

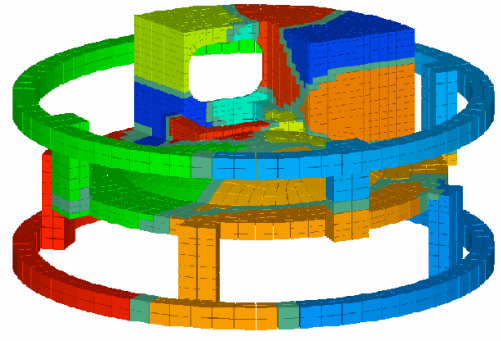
End

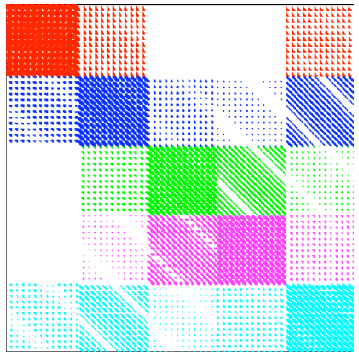
Extra

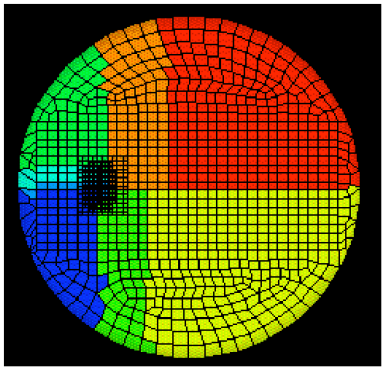
Partitioning and Load Balancing

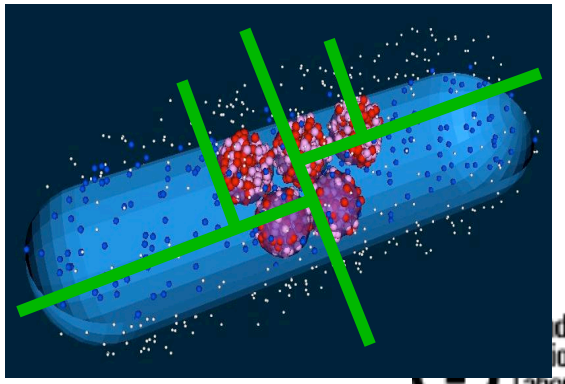
- Assignment of application data to processors for parallel computation.
- Applied to grid points, elements, matrix rows, particles, ….











Static Partitioning

- Static partitioning in an application:
  - Data partition is computed.
  - Data are distributed according to partition map.
  - Application computes.
- Ideal partition:
  - Processor idle time is minimized.
  - Inter-processor communication costs are kept low.
- Zoltan_Set_Param(zz, “LB_APPROACH”, “PARTITION”);

Zoltan Application Interface

APPLICATION

Initialize Zoltan
(Zoltan_Initialize, Zoltan_Create)

Select Method and Parameters
(Zoltan_Set_Params)

Register query functions
(Zoltan_Set_Fn)

(Re)partition
(Zoltan_LB_Partition)

Move data
(Zoltan_Migrate)

COMPUTE

Clean up
(Zoltan_Destroy)

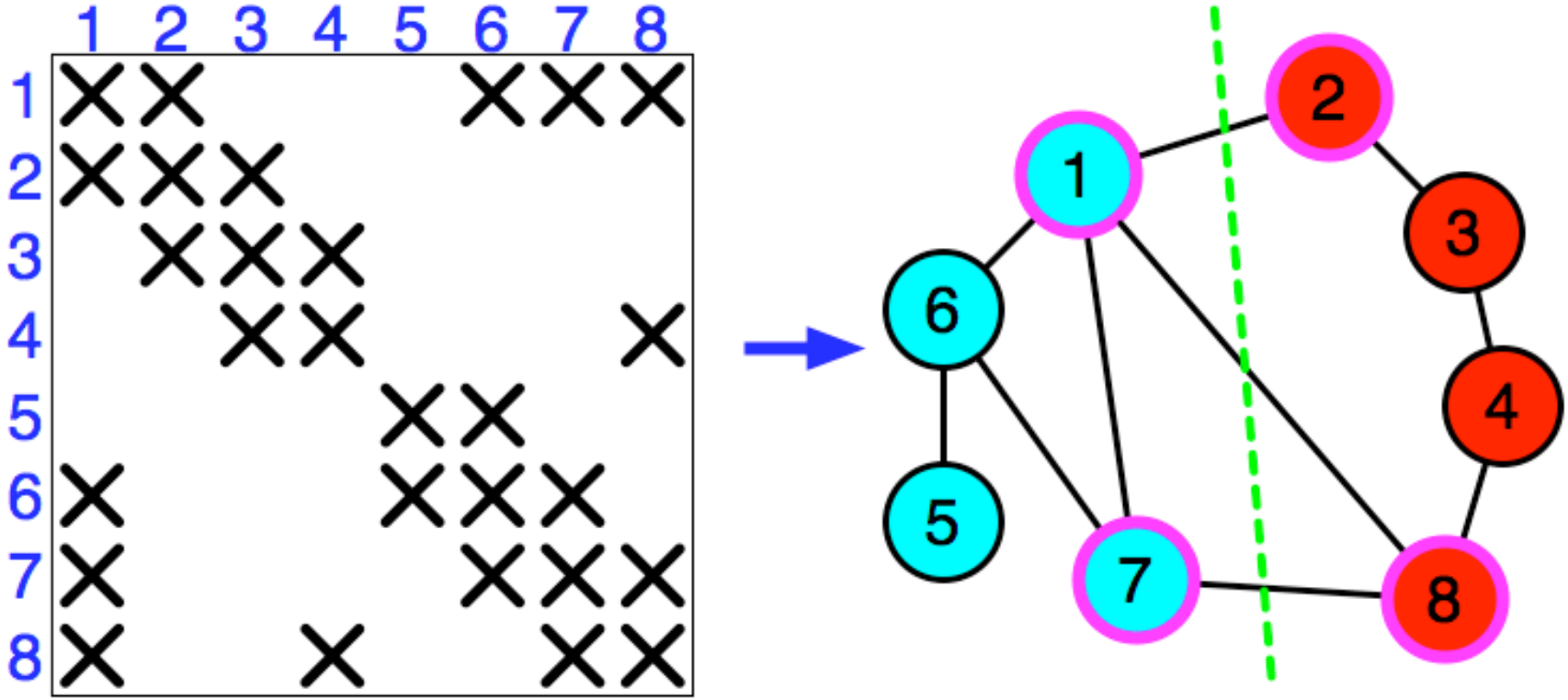
ZOLTAN

Zoltan_LB_Partition:
- Call query functions.
- Build data structures.
- Compute new decomposition.
- Return import/export lists.

Zoltan_Migrate:
- Call packing query functions for exports.
- Send exports.
- Receive imports.
- Call unpacking query functions for imports.

Sandia National Laboratories

Graph Model Revisited



- Bisection: count boundary vertices
- Slightly more complicated for k>2