Tutorial: The Zoltan Toolkit

Erik Boman, Cedric Chevalier, Karen Devine,

Vitus Leung, Sivasankaran Rajamanickam, Lee Ann Riesen,

Michael Wolf

Sandia National Laboratories, NM

Umit Çatalyürek, Doruk Bozdag

Ohio State University

ACTS Workshop: August 20, 2010

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

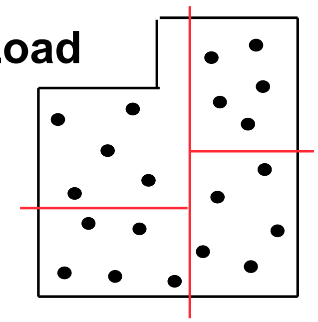
Outline

- High-level view of Zoltan
- Requirements, data models, and interface
- Load Balancing and Partitioning
- Matrix Ordering, Graph Coloring
- Utilities
- Isorropia

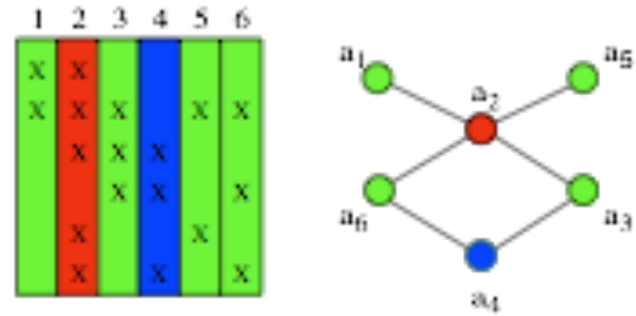
The Zoltan Toolkit

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

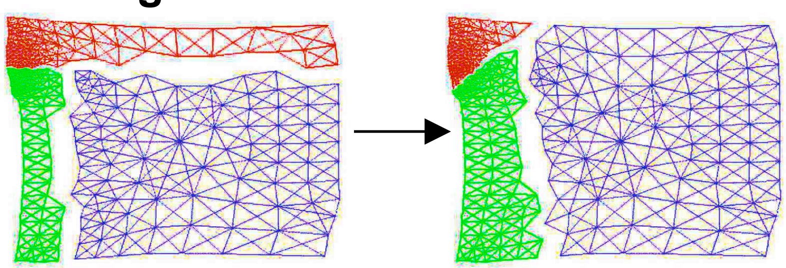
Dynamic Load Balancing



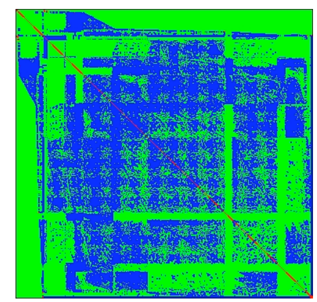
Graph Coloring



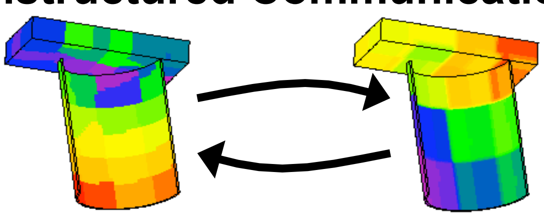
Data Migration



Matrix Ordering



Unstructured Communication



Distributed Data Directories

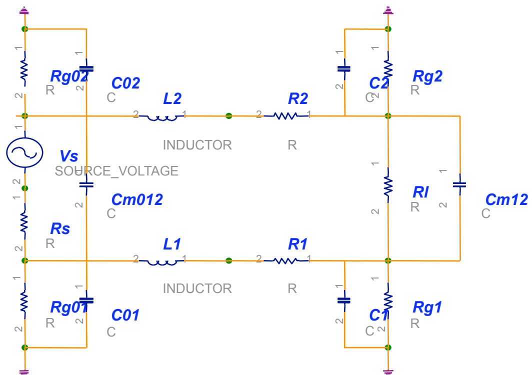
A	B	C	D	E	F	G	H	I
0	1	0	2	1	0	1	2	1

Zoltan System Assumptions

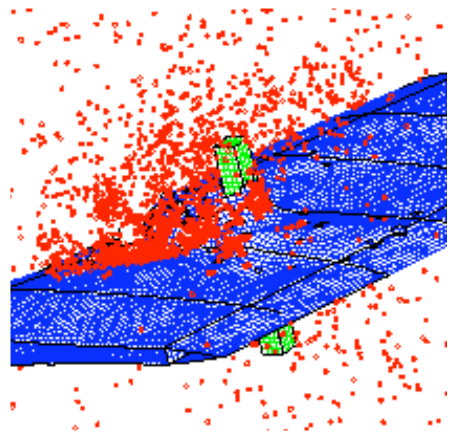
- **Assume distributed memory model.**
- **Data decomposition + "Owner computes":**
  - **The data is distributed among the processors.**
  - **The owner performs all computation on its data.**
  - **Data distribution defines work assignment.**
  - **Data dependencies among data items owned by different processors incur communication.**
- **Zoltan is available in Trilinos since version 9.0**
- **Requirements:**
  - **MPI (when running in parallel)**
  - **C compiler**
  - **Autotools or CMake.**

Zoltan Supports Many Applications

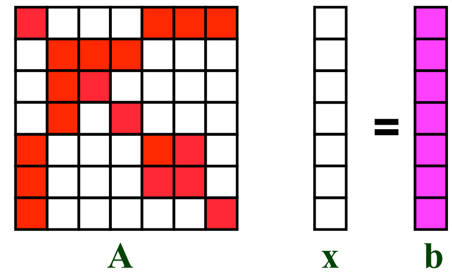
- **Different applications, requirements, data structures.**



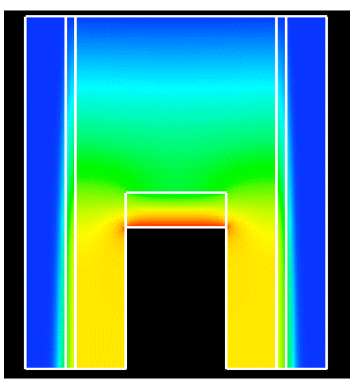
Parallel electronics networks



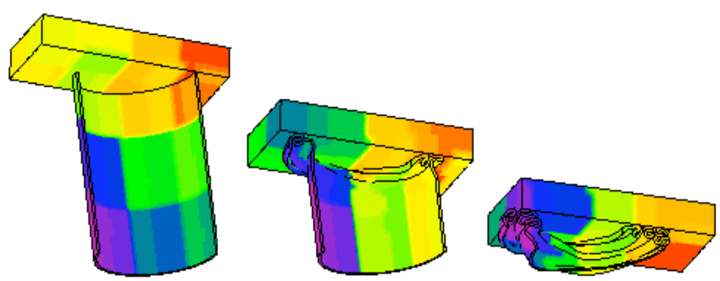
Particle methods



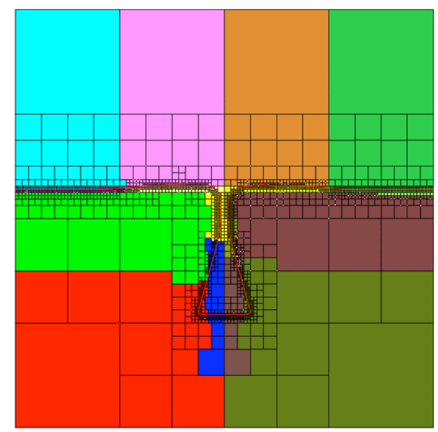
Linear solvers & preconditioners



Multiphysics simulations



Crash simulations



Adaptive mesh refinement

Zoltan's use in large-scale experiments and 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

SciDAC Collaboration: ITAPS

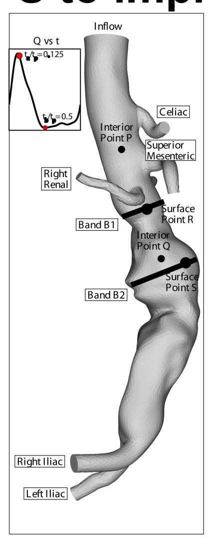
Sandia National Laboratories

- ITAPS developers at RPI use Zoltan for dynamic load balancing in their Flexible Mesh DataBase (FMDB) through iZoltan and iMeshP.
  - Initial partitioning of large meshes (1B elements) for up to 128K cores.
  - Dynamic repartitioning of adaptively refined meshes.
- FMDB is used by SLAC and PPPL for adaptive meshing.

- RPI also uses Zoltan for static parallel graph and hypergraph partitioning of non-adaptive simulations.
  - Achieved strong scalability up to 128K cores (BG/P) for CFD code PHASTA.
  - We continue work with ITAPS to improve robustness on >10K cores.

ITAPS
INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

Results courtesy of K. Jansen, M. Shephard, M. Zhou, T. Xie, O. Sahni; Rensselaer Polytechnic Institute.

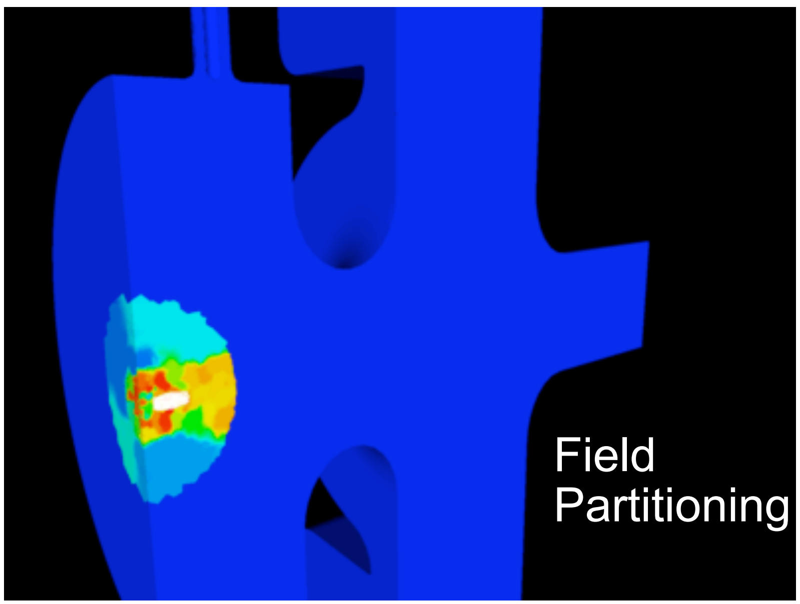


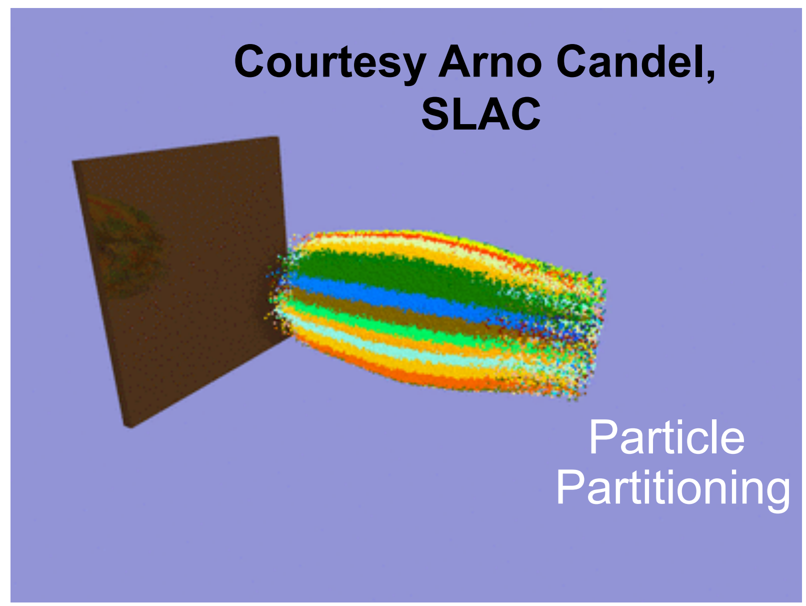
Number of cores	Time (s)	Efficiency
16k	222.03	1
32k	112.43	0.987
64k	57.09	0.972
128k	31.35	0.885

SciDAC Collaborations: 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)

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 Interface

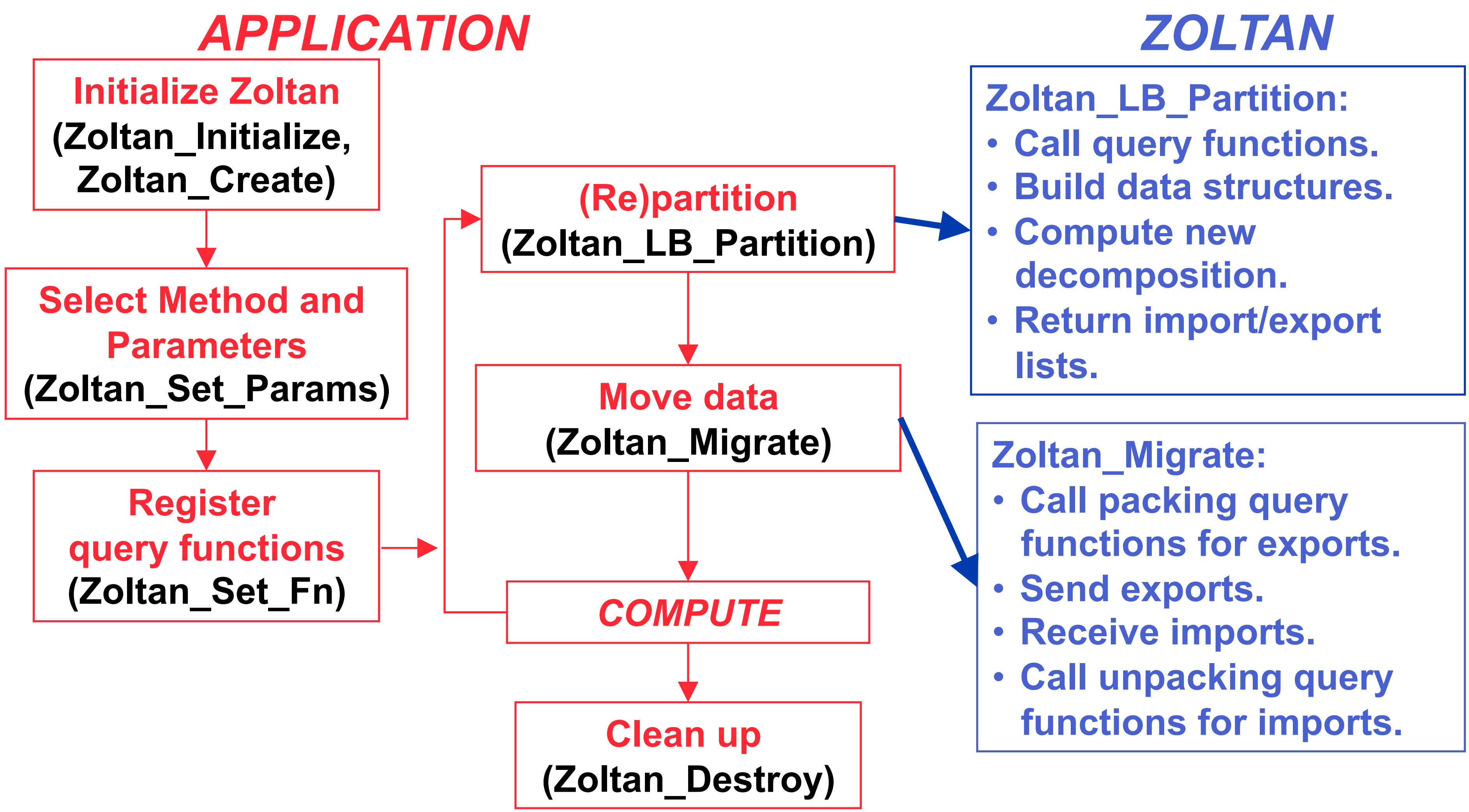
- Simple, easy-to-use interface.
  - Small number of callable Zoltan functions.
  - Callable from C, C++, Fortran.

- Requirement: Unique global IDs for objects to be partitioned/ordered/colored. For example:
  - Global element number.
  - Global matrix row number.
  - (Processor number, local element number)
  - (Processor number, local particle number)

Zoltan Application Interface

- 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.
- Query mechanism supports…
  - Geometric algorithms
    - Queries for dimensions, coordinates, etc.
  - Hypergraph- and graph-based algorithms
    - Queries for edge lists, edge weights, etc.
  - Tree-based algorithms
    - Queries for parent/child relationships, etc.
- Once query functions are implemented, application can access all Zoltan functionality.
  - Can switch between algorithms by setting parameters.

Zoltan Application Interface



Zoltan Query Functions

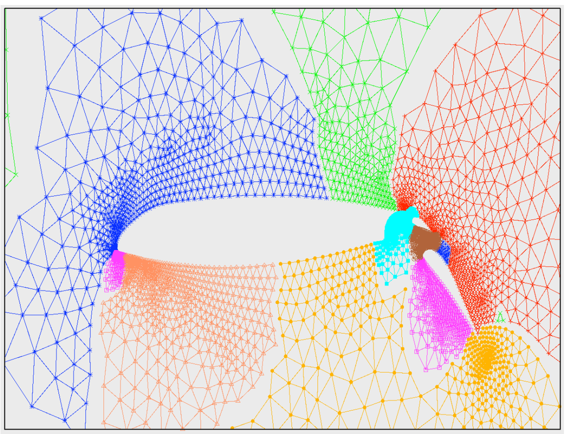
General Query Functions	
ZOLTAN_NUM_OBJ_FN	Number of items on processor
ZOLTAN_OBJ_LIST_FN	List of item IDs and weights.
Geometric Query Functions	
ZOLTAN_NUM_GEOM_FN	Dimensionality of domain.
ZOLTAN_GEOM_FN	Coordinates of items.
Hypergraph Query Functions	
ZOLTAN_HG_SIZE_CS_FN	Number of hyperedge pins.
ZOLTAN_HG_CS_FN	List of hyperedge pins.
ZOLTAN_HG_SIZE_EDGE_WTS_FN	Number of hyperedge weights.
ZOLTAN_HG_EDGE_WTS_FN	List of hyperedge weights.
Graph Query Functions	
ZOLTAN_NUM_EDGE_FN	Number of graph edges.
ZOLTAN_EDGE_LIST_FN	List of graph edges and weights.

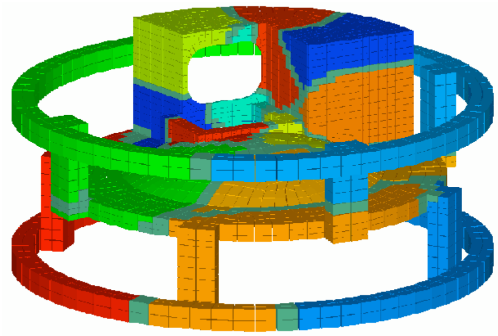
Using Zoltan in Your Application

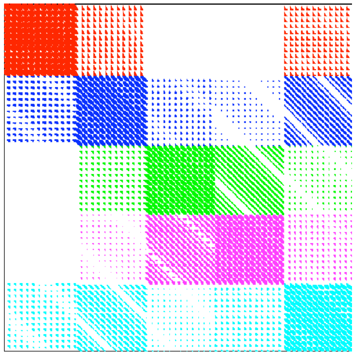
1. Decide what your objects are.
   - Elements? Grid points? Matrix rows? Particles?
2. Decide which tools (partitioning/ordering/coloring/utilities) and class of method (geometric/graph/hypergraph) to use.
3. Download Zoltan.
   - http://www.cs.sandia.gov/Zoltan (or http://trilinos.sandia.gov)
4. Write required query functions for your application.
   - Required functions are listed with each method in Zoltan User's Guide.
5. Call Zoltan from your application.
6. #include "zoltan.h" in files calling Zoltan.
7. Configure and build Zoltan.
8. Compile application; link with libzoltan.a.
   - mpicc application.c -lzoltan

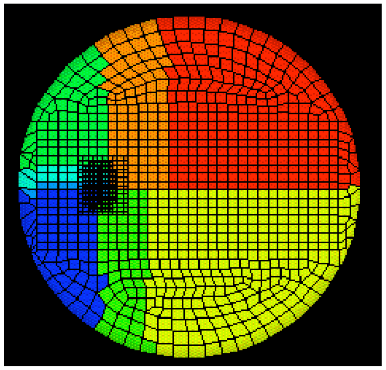
Partitioning and Load Balancing

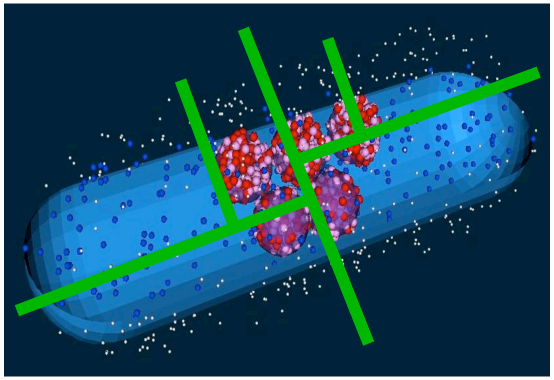
- Assignment of application data to processors for parallel computation.
- Applied to grid points, elements, matrix rows, particles, ….











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”);

Dynamic Repartitioning (a.k.a. Dynamic Load Balancing)

- **Dynamic repartitioning (load balancing) in an application:**
  - **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.**

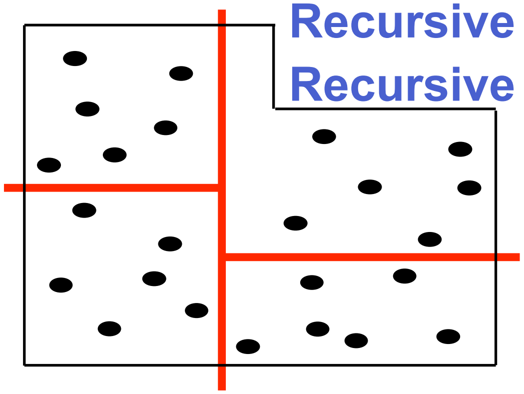
- **Zoltan_Set_Param(zz, “LB_APPROACH”, “REPARTITION”);**

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 the Zoltan Toolkit

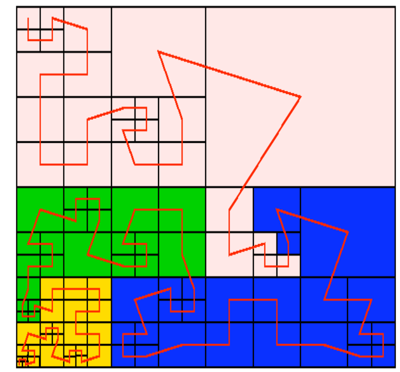
Geometric (coordinate-based) methods



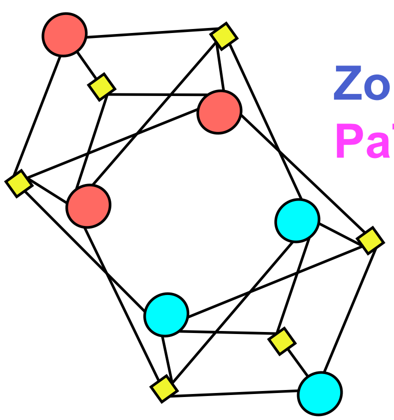
Recursive Coordinate Bisection (Berger, Bokhari)

Recursive Inertial Bisection (Taylor, Nour-Omid)

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



Combinatorial (topology-based) methods



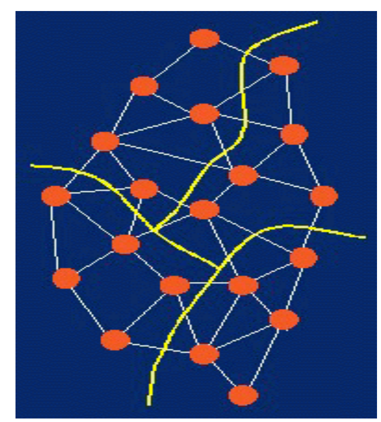
Zoltan Hypergraph Partitioning (PHG)

PaToH (Catalyurek & Aykanat)

Zoltan Graph Partitioning (PHG)

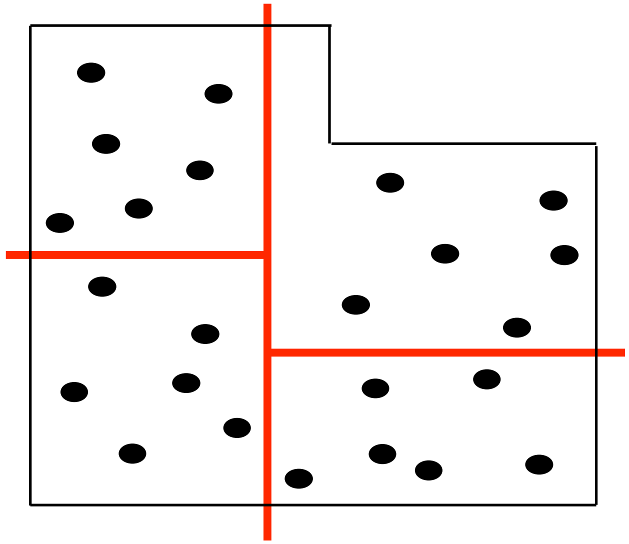
ParMETIS (Karypis, et al.)

PT-Scotch (Pellegrini, et al.)

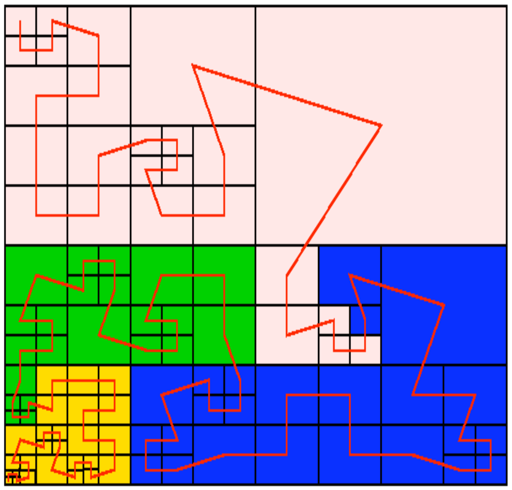


Geometric Partitioning

- 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.



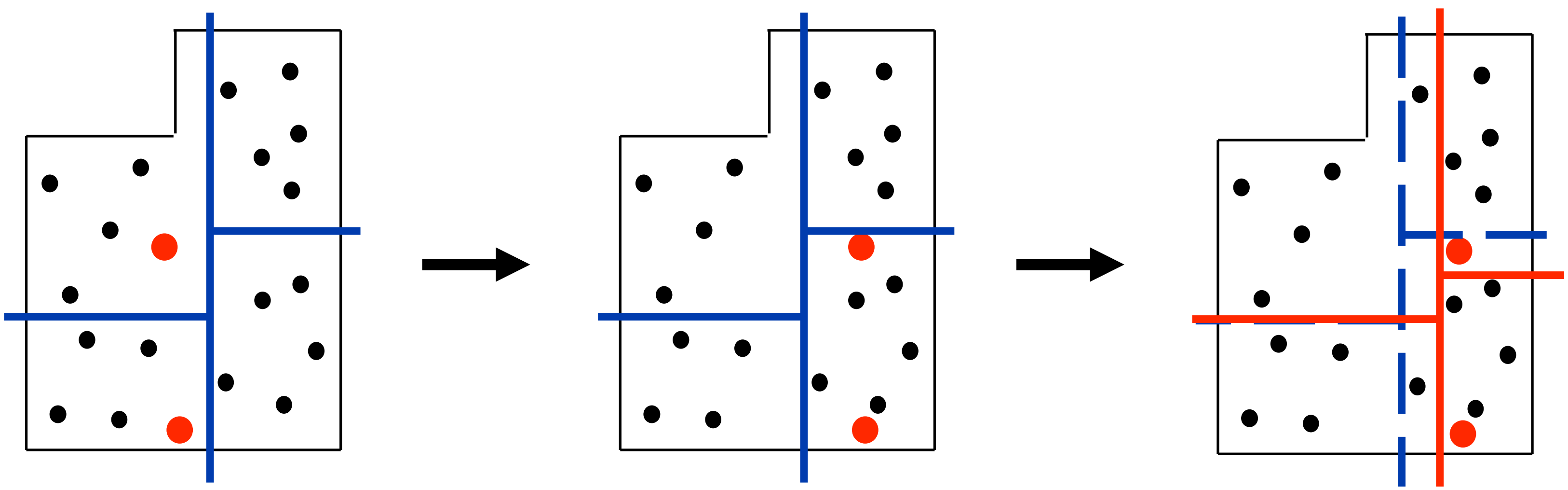
Recursive Coordinate Bisection (RCB)
Berger & Bokhari, 1987



Space Filling Curve Partitioning (HSFC)
Warren & Salmon, 1993;
Pilkington & Baden, 1994; Patra & Oden, 1995

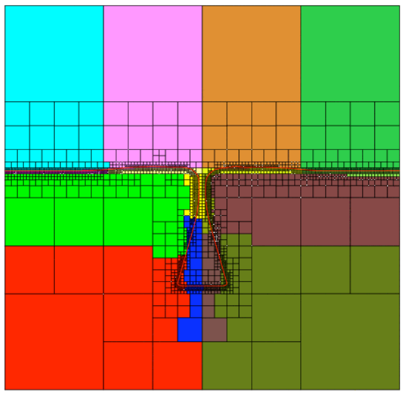
Geometric Repartitioning

- No explicit control of migration costs, but…
- Implicitly achieves low data redistribution costs
- For small changes in data, cuts move only slightly, resulting in little data redistribution.

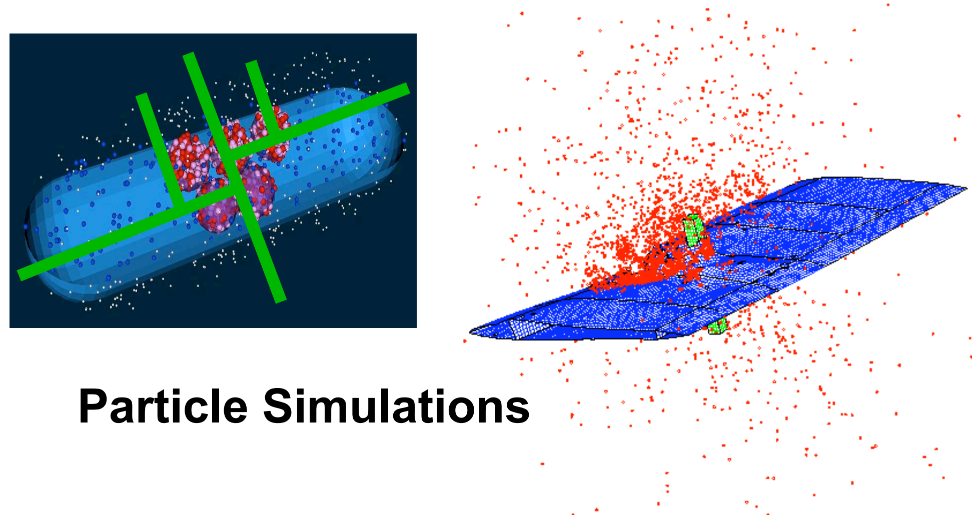


Recursive Coordinate Bisection (RCB)

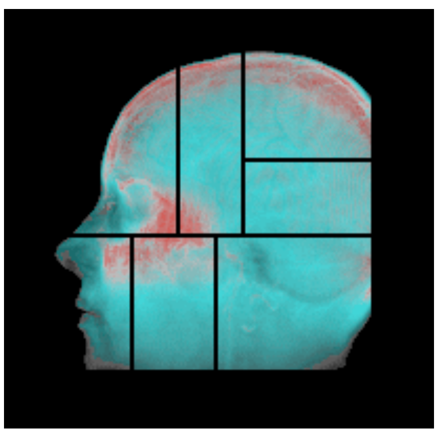
Applications of Geometric Partitioners



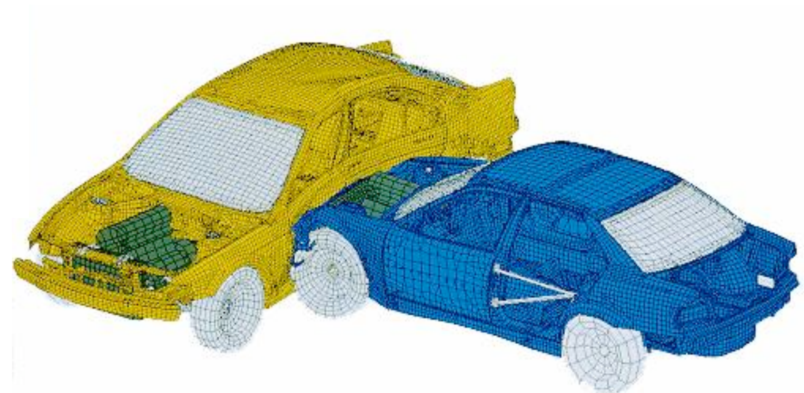
Adaptive Mesh Refinement



Particle Simulations



Parallel Volume Rendering



Crash Simulations and Contact Detection

Geometric Methods: Advantages and Disadvantages

- 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.

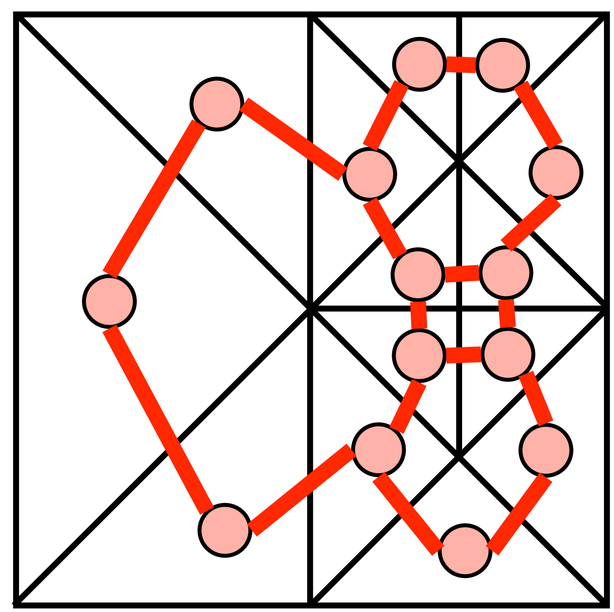
Geometric Partitioning: Query Functions

General Query Functions	
ZOLTAN_NUM_OBJ_FN	Number of items on processor
ZOLTAN_OBJ_LIST_FN	List of item IDs and weights.
Geometric Query Functions	
ZOLTAN_NUM_GEOM_FN	Dimensionality of domain.
ZOLTAN_GEOM_FN	Coordinates of items.
Hypergraph Query Functions	
ZOLTAN_HG_SIZE_CS_FN	Number of hyperedge pins.
ZOLTAN_HG_CS_FN	List of hyperedge pins.
ZOLTAN_HG_SIZE_EDGE_WTS_FN	Number of hyperedge weights.
ZOLTAN_HG_EDGE_WTS_FN	List of hyperedge weights.
Graph Query Functions	
ZOLTAN_NUM_EDGE_FN	Number of graph edges.
ZOLTAN_EDGE_LIST_FN	List of graph edges and weights.

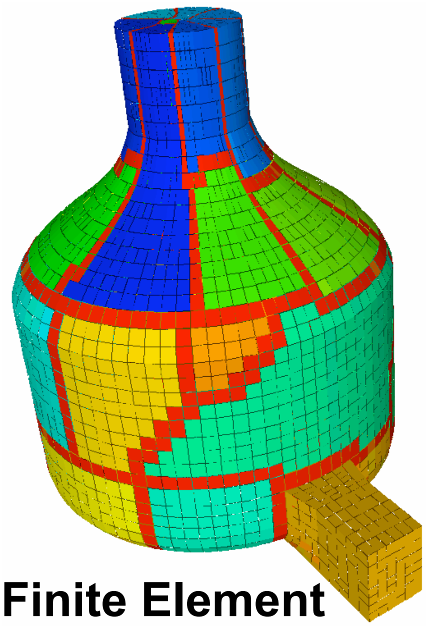
Graph Partitioning

- 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");

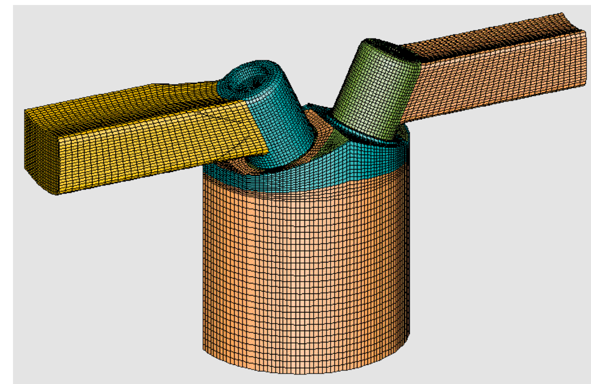
- Kernighan, Lin, Schweikert, Fiduccia, Mattheyes, Simon, Hendrickson, Leland, Kumar, Karypis, et al.
- Represent problem as a weighted graph.
  - Vertices = objects to be partitioned.
  - Edges = dependencies between two objects.
  - Weights = work load or amount of dependency.
- Partition graph so that …
  - Parts have equal vertex weight.
  - Weight of edges cut by part boundaries is small.



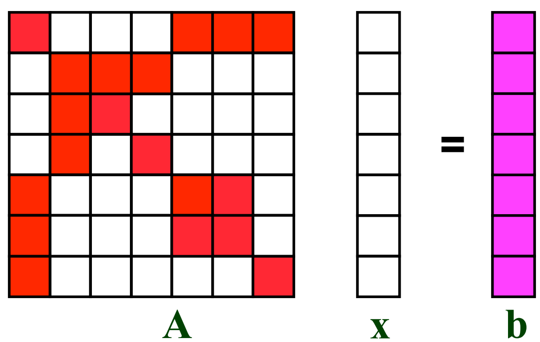
Applications using Graph Partitioning



Finite Element Analysis



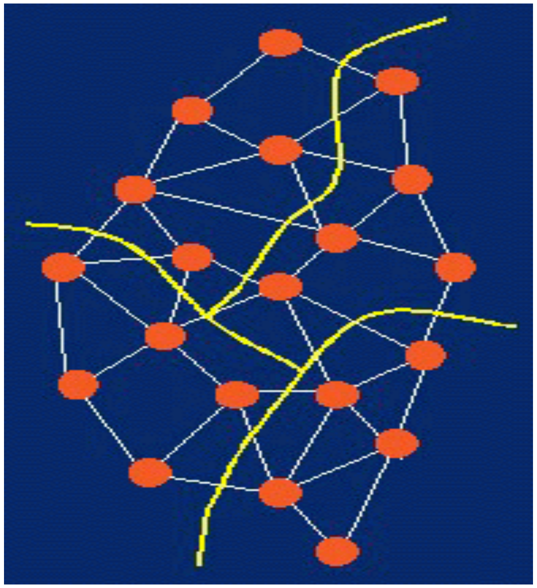
Multiphysics and multiphase simulations



Linear solvers & preconditioners
(square, structurally symmetric systems)

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.)
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.**

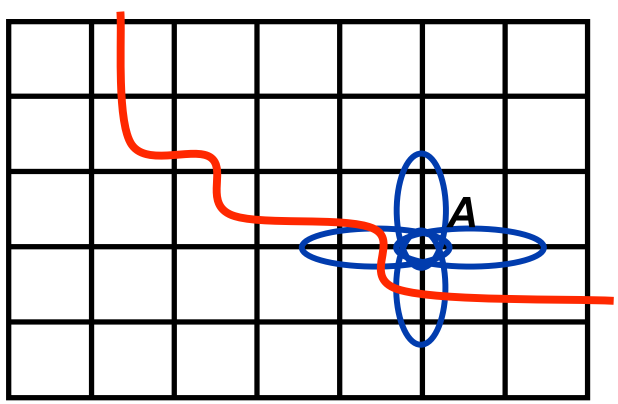
Graph Partitioning: Query Functions

General Query Functions	
ZOLTAN_NUM_OBJ_FN	Number of items on processor
ZOLTAN_OBJ_LIST_FN	List of item IDs and weights.
Geometric Query Functions	
ZOLTAN_NUM_GEOM_FN	Dimensionality of domain.
ZOLTAN_GEOM_FN	Coordinates of items.
Hypergraph Query Functions	
ZOLTAN_HG_SIZE_CS_FN	Number of hyperedge pins.
ZOLTAN_HG_CS_FN	List of hyperedge pins.
ZOLTAN_HG_SIZE_EDGE_WTS_FN	Number of hyperedge weights.
ZOLTAN_HG_EDGE_WTS_FN	List of hyperedge weights.
Graph Query Functions	
ZOLTAN_NUM_EDGE_FN	Number of graph edges.
ZOLTAN_EDGE_LIST_FN	List of graph edges and weights.

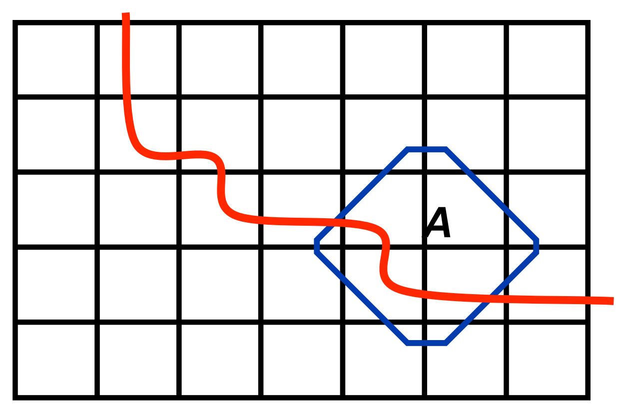
Hypergraph Partitioning

- Zoltan_Set_Param(zz, “LB_METHOD”, “HYPERGRAPH”);
- Zoltan_Set_Param(zz, “HYPERGRAPH_PACKAGE”, “ZOLTAN”); 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

Hypergraph Repartitioning

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

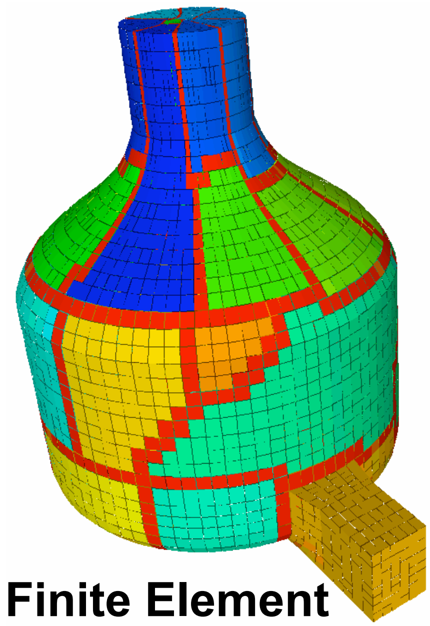
  Data redistribution volume

  + Application communication volume

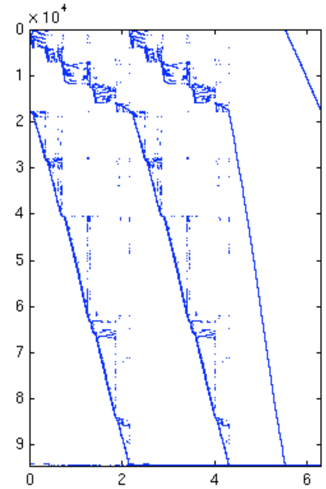
  Total communication volume
- **Data redistribution volume: callback returns data sizes**
  - **Zoltan_Set_Fn(zz, ZOLTAN_OBJ_SIZE_MULTI_FN_TYPE, myObjSizeFn, 0);**
- **Application communication volume = Hyperedge cuts * Number of times the communication is done between repartitionings.**
  - **Zoltan_Set_Param(zz, "PHG_REPART_MULTIPLIER", "100");**

Sandia National Laboratories

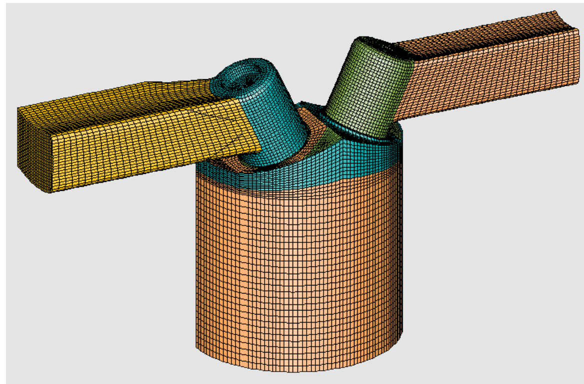
Hypergraph Applications



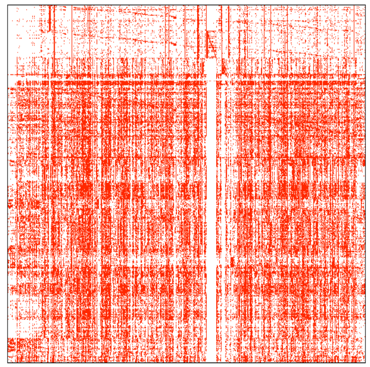
Finite Element Analysis



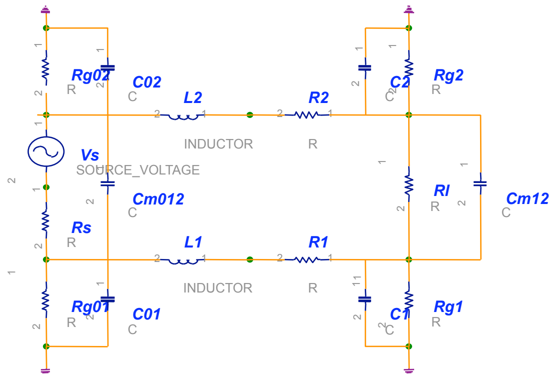
Linear programming for sensor placement



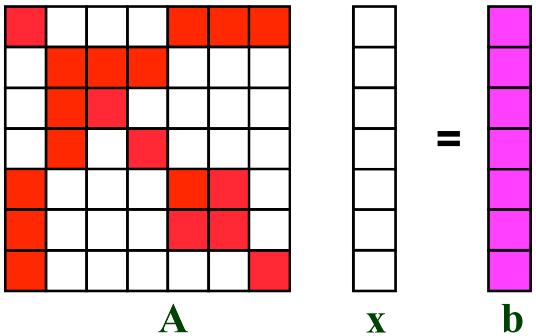
Multiphysics and multiphase simulations



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:
  - Usually more expensive than graph partitioning.

Hypergraph Partitioning with Hypergraph Query Functions

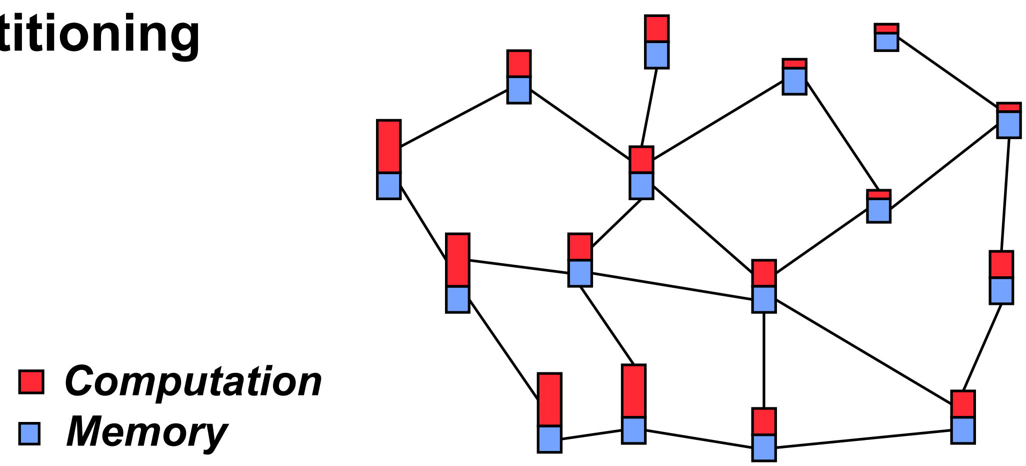
General Query Functions	
`ZOLTAN_NUM_OBJ_FN`	Number of items on processor
`ZOLTAN_OBJ_LIST_FN`	List of item IDs and weights.
Geometric Query Functions	
`ZOLTAN_NUM_GEOM_FN`	Dimensionality of domain.
`ZOLTAN_GEOM_FN`	Coordinates of items.
Hypergraph Query Functions	
`ZOLTAN_HG_SIZE_CS_FN`	Number of hyperedge pins.
`ZOLTAN_HG_CS_FN`	List of hyperedge pins.
`ZOLTAN_HG_SIZE_EDGE_WTS_FN`	Number of hyperedge weights.
`ZOLTAN_HG_EDGE_WTS_FN`	List of hyperedge weights.
Graph Query Functions	
`ZOLTAN_NUM_EDGE_FN`	Number of graph edges.
`ZOLTAN_EDGE_LIST_FN`	List of graph edges and weights.

Hypergraph Partitioning with Graph Query Functions

General Query Functions	
`ZOLTAN_NUM_OBJ_FN`	Number of items on processor
`ZOLTAN_OBJ_LIST_FN`	List of item IDs and weights.
Geometric Query Functions	
`ZOLTAN_NUM_GEOM_FN`	Dimensionality of domain.
`ZOLTAN_GEOM_FN`	Coordinates of items.
Hypergraph Query Functions	
`ZOLTAN_HG_SIZE_CS_FN`	Number of hyperedge pins.
`ZOLTAN_HG_CS_FN`	List of hyperedge pins.
`ZOLTAN_HG_SIZE_EDGE_WTS_FN`	Number of hyperedge weights.
`ZOLTAN_HG_EDGE_WTS_FN`	List of hyperedge weights.
Graph Query Functions	
`ZOLTAN_NUM_EDGE_FN`	Number of graph edges.
`ZOLTAN_EDGE_LIST_FN`	List of graph edges and weights.

Multi-Criteria Load-Balancing

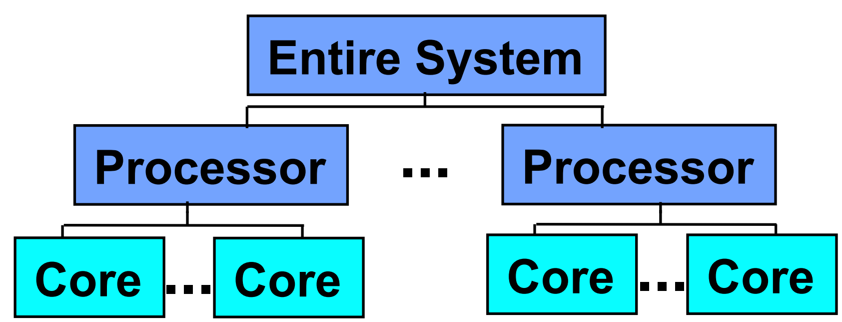
- Multiple constraints or objectives
  - Compute a single partition that is good with respect to multiple factors.
    - Balance both computation and memory
    - Balance multi-phase simulations
  - Extend algorithms to multiple weights
    - Difficult. No guarantee good solution exists.
- Zoltan_Set_Param(zz, “OBJ_WEIGHT_DIM”, “2”);
  - Available in RCB, RIB and ParMETIS graph partitioning



Heterogeneous Architectures

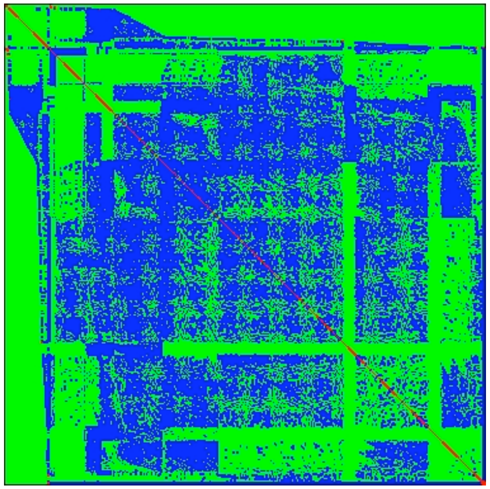
- Clusters may have different types of processors.
- Assign "capacity" weights to processors.
  - E.g., Compute power (speed).
  - Zoltan_LB_Set_Part_Sizes(…);
    - Note: Can use this function to specify part sizes for any purpose.
- Balance with respect to processor capacity.

- Hierarchical partitioning: Allows different partitioners at different architecture levels.
  - Zoltan_Set_Param(zz, "LB_METHOD", "HIER");
  - Requires three additional callbacks to describe architecture hierarchy.
    - ZOLTAN_HIER_NUM_LEVELS_FN
    - ZOLTAN_HIER_PARTITION_FN
    - ZOLTAN_HIER_METHOD_FN

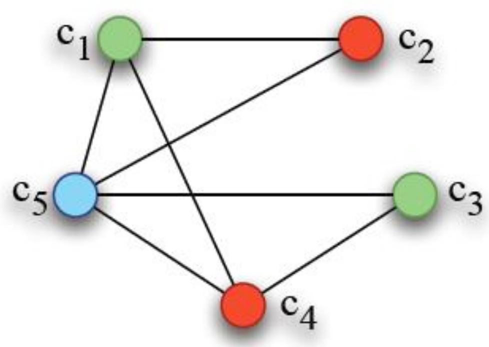


Zoltan Ordering

- Global ordering produces fill-reducing permutations for sparse matrix factorization.
  - Interface to PT-Scotch (Pellegrini, Chevalier; INRIA-LaBRi)
  - Interface to ParMETIS (Karypis et al.; U. Minnesota)
- Local ordering improves cache utilization.
  - Space-filling curve ordering of in-processor data.
- Ordering algorithms use the same callback function interface as partitioning algorithms.



Zoltan Graph Coloring



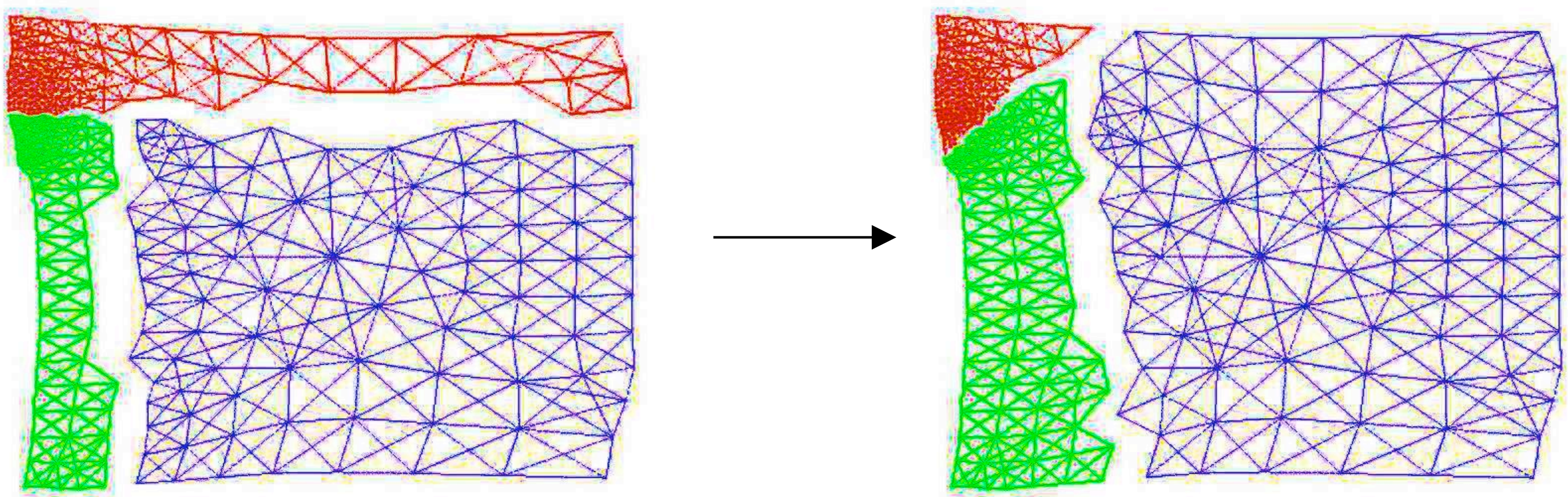
- **Parallel distance-1 and distance-2 graph coloring.**
- **Graph built using same application interface and code as graph partitioners.**
- **Generic coloring interface; easy to add new coloring algorithms.**
- **Algorithms**
  - **Distance-1: Bozdag, Gebremedhin, Manne, Boman, Catalyurek**
  - **Distance-2: Bozdag, Catalyurek, Gebremedhin, Manne, Boman, Ozguner**

Other Zoltan Functionality

- Tools needed when doing dynamic load balancing:
  - Data Migration
  - Unstructured Communication Primitives
  - Distributed Data Directories
- Functionalities described in Zoltan User's Guide
  - http://www.cs.sandia.gov/Zoltan/ug_html/ug.html

Zoltan Data Migration Tools

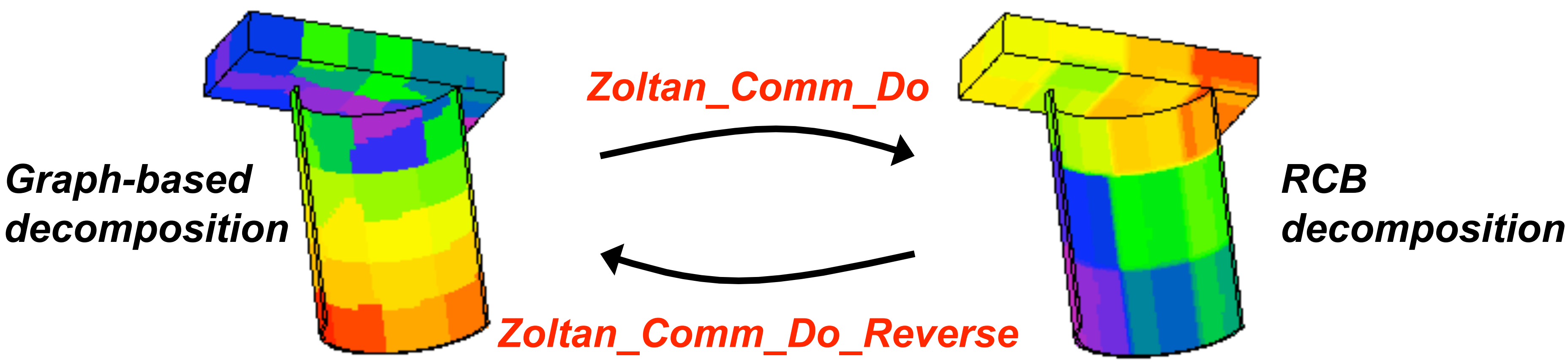
- After partition is computed, data must be moved to new decomposition.
  - Depends strongly on application data structures
  - Complicated communication patterns
- Zoltan can help!
  - Application supplies query functions to pack/unpack data.
  - Zoltan does all communication to new processors.



Zoltan Unstructured Communication Package

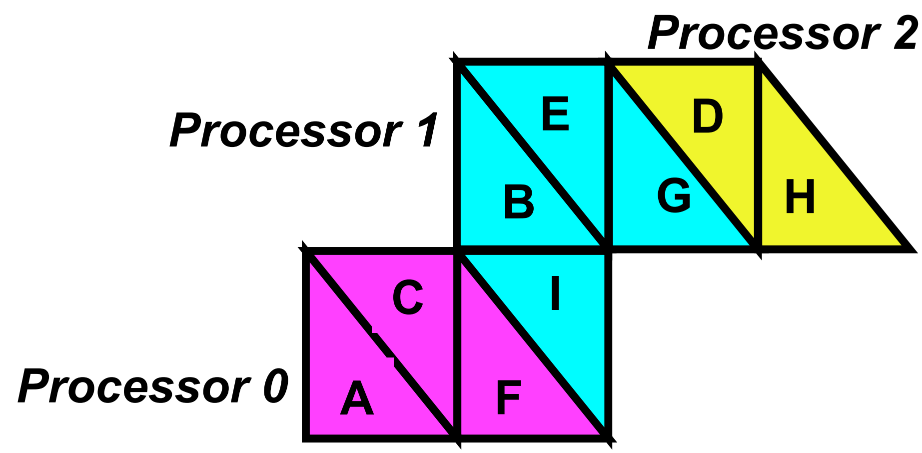
Sandia National Laboratories

- Simple primitives for efficient irregular communication.
  - Zoltan_Comm_Create: Generates communication plan.
    - Processors and amount of data to send and receive.
  - Zoltan_Comm_Do: Send data using plan.
    - Can reuse plan. (Same plan, different data.)
  - Zoltan_Comm_Do_Reverse: Inverse communication.
- Used for most communication in Zoltan.



Zoltan Distributed Data Directory

- **Helps applications locate off-processor data.**
- **Rendezvous algorithm (Pinar, 2001).**
  - **Directory distributed in known way (hashing) across processors.**
  - **Requests for object location sent to processor storing the object's directory entry.**



Directory Index →	A	B	C
Location →	0	1	0

Processor 0

D	E	F
2	1	0

Processor 1

G	H	I
1	2	1

Processor 2

Interfaces to Zoltan

- C, C++ and F90 interfaces in Zoltan
- Mesh-based interface in ITAPS
- Isorropia: matrix-based interface in Trilinos

ITAPS Dynamic Services: Mesh-based Interface to Zoltan

- **Interoperable Technologies for Advanced Petascale Simulations (L. Diachin, LLNL, PI)**
  - **SciDAC2 CET.**
- **ITAPS Goals:**
  - **Develop the next generation of meshing and geometry tools for petascale computing.**
    - **E.g., adaptive mesh refinement, shape optimization.**
  - **Improve applications' ability to use these tools.**
    - **"Standardization" of mesh interfaces.**
- **Dynamic Services toolkit:**
  - **ITAPS-compliant mesh interface to Zoltan tools.**
  - **Integration with ITAPS iMeshP parallel mesh interface to be released FY09.**

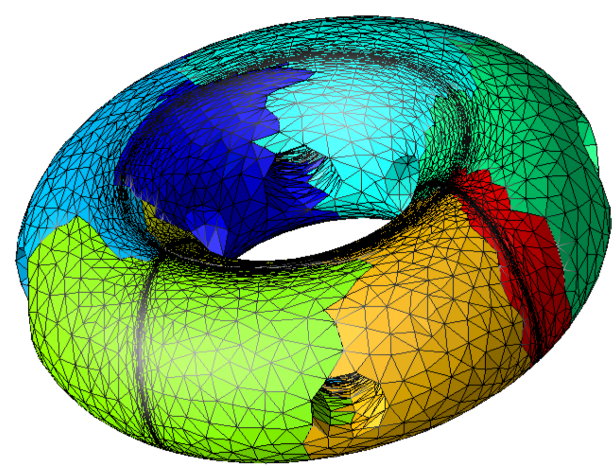


Image courtesy of M. Shephard, RPI

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)**
  - **Epetra: parallel linear algebra package**

- **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**

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**

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
- ITAPS Home Page
  - http://www.itaps.org
- CSCAPES Home Page
  - http://www.cscapes.org
- Email
  - zoltan-dev@software.sandia.gov

The End

Partitioning Interface

Zoltan computes the difference (Δ) from current distribution

Choose between:

a) Import lists (data to import from other procs)
b) Export lists (data to export to other procs)
c) Both (the default)

```
err = Zoltan_LB_Partition(zz,
    &changes, /* Flag indicating whether partition changed */
    &numGidEntries, &numLidEntries,
    &numImport, /* objects to be imported to new part */
    &importGlobalGids, &importLocalGids, &importProcs, &importToPart,
    &numExport, /* # objects to be exported from old part */
    &exportGlobalGids, &exportLocalGids, &exportProcs, &exportToPart);
```

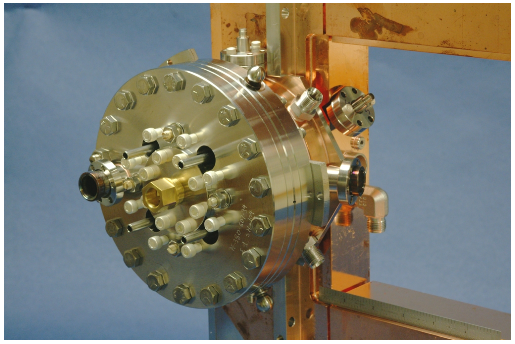
Extra Slides

- Experimental results: Partitioning

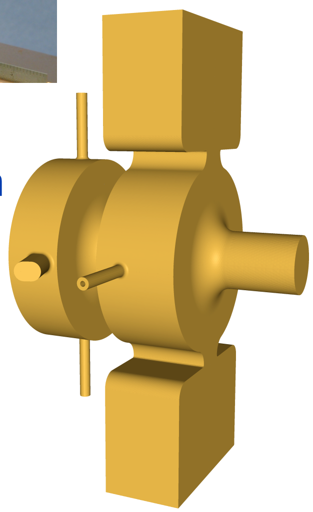
Performance Results

- Experiments on Sandia's Thunderbird cluster.
  - Dual 3.6 GHz Intel EM64T processors with 6 GB RAM.
  - Infiniband network.
- Compare RCB, HSFC, graph and hypergraph methods.
- Measure …
  - Amount of communication induced by the partition.
  - Partitioning time.

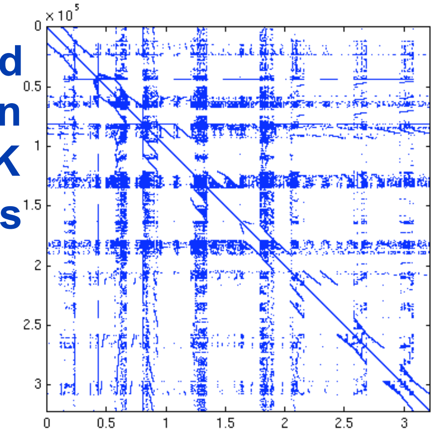
Test Data



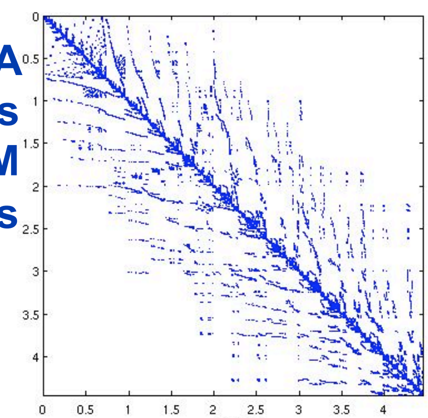
SLAC *LCLS
Radio Frequency Gun
6.0M x 6.0M
23.4M nonzeros



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



Cage15 DNA
Electrophoresis
5.1M x 5.1M
99M nonzeros



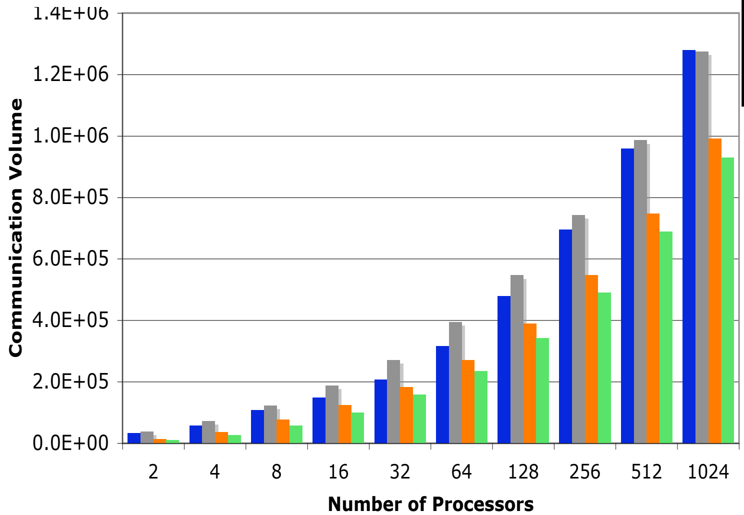
SLAC Linear Accelerator
2.9M x 2.9M
11.4M nonzeros

Communication Volume: Lower is Better

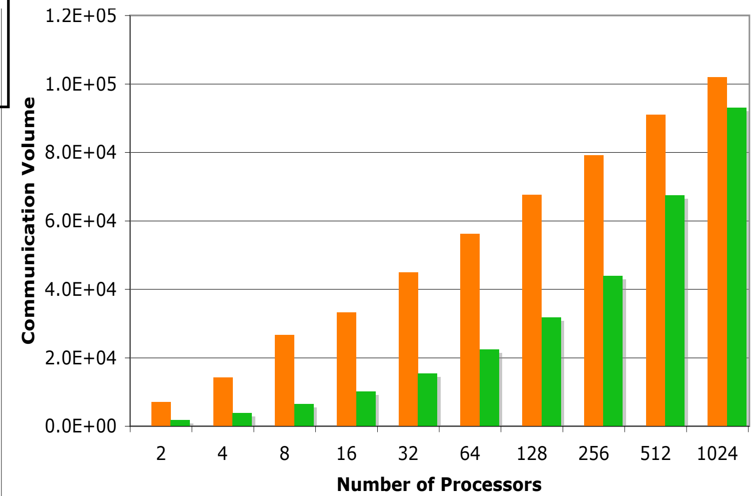
Sandia National Laboratories

Number of parts = number of processors.

SLAC 6.0M LCLS

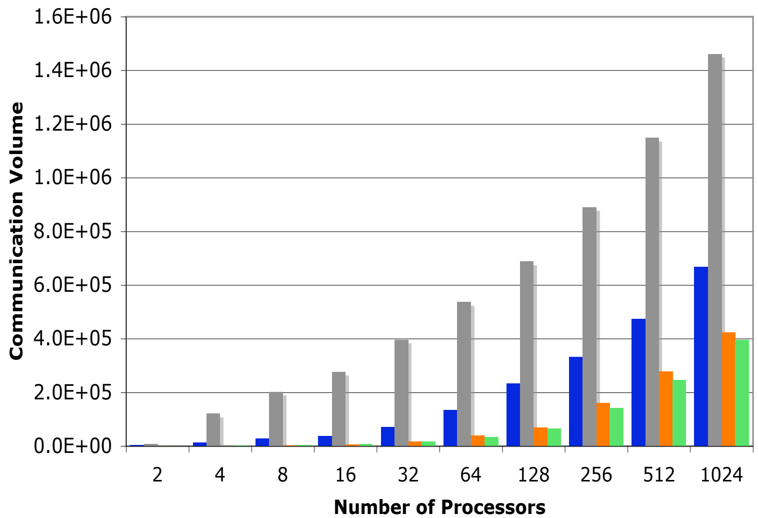


Xyce 680K circuit

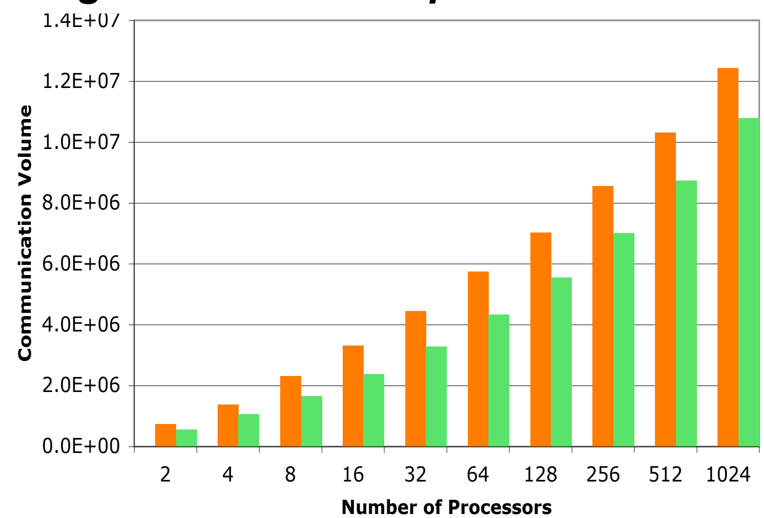


- RCB
- HSFC
- Graph
- Hypergraph

SLAC 2.9M Linear Accelerator



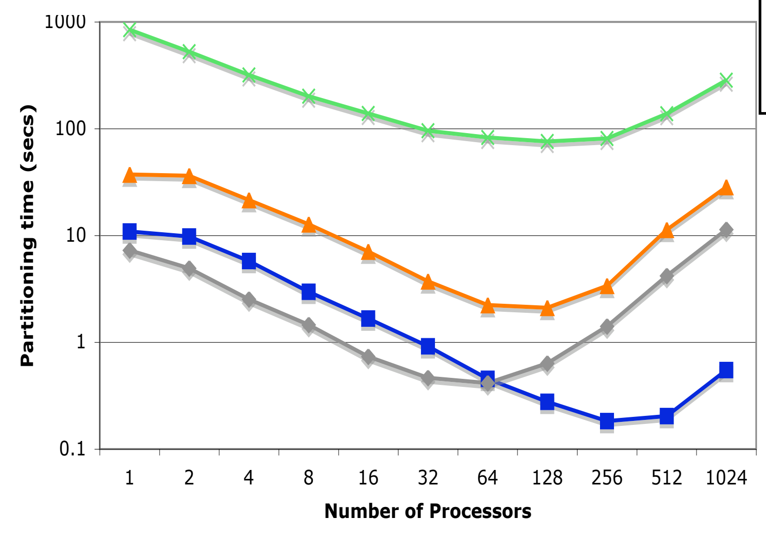
Cage15 5.1M electrophoresis



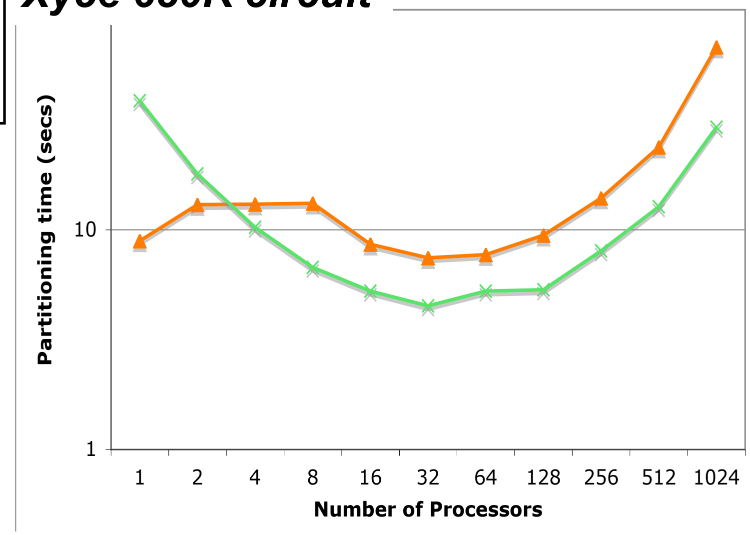
Partitioning Time: Lower is better

1024 parts. Varying number of processors.

SLAC 6.0M LCLS

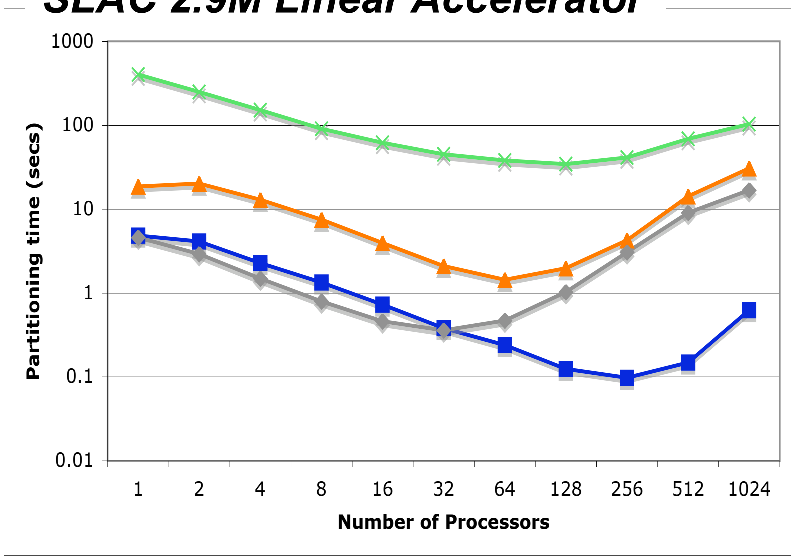


Xyce 680K circuit

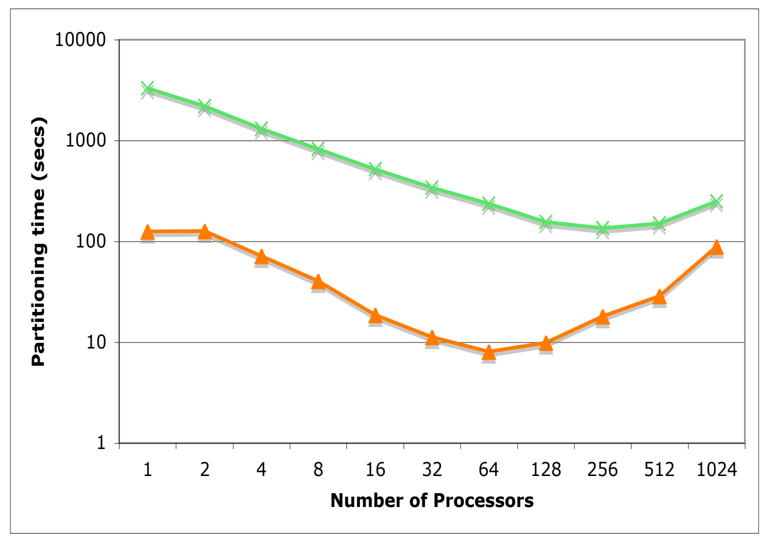


- RCB
- HSFC
- Graph
- Hypergraph

SLAC 2.9M Linear Accelerator



Cage15 5.1M electrophoresis



Extra Slides

- **Experimental results: Repartitioning**

Repartitioning Experiments

- Experiments with 64 parts on 64 processors.
- Dynamically adjust weights in data to simulate, say, adaptive mesh refinement.
- Repartition.
- Measure repartitioning time and total communication volume:

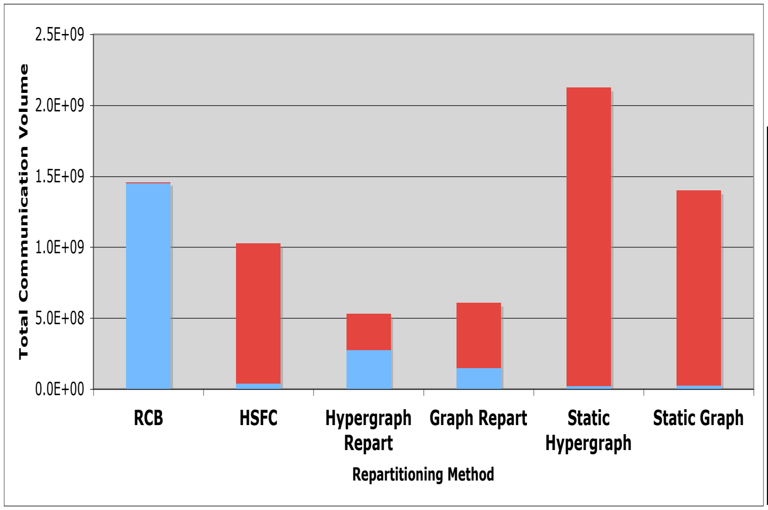
  Data redistribution volume

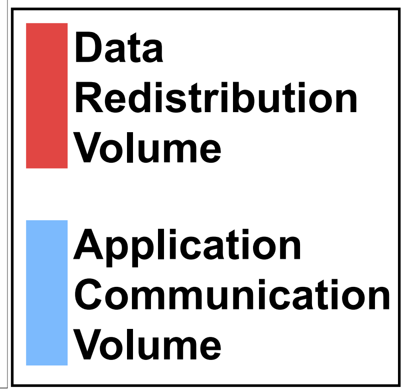
  + Application communication volume

  Total communication volume

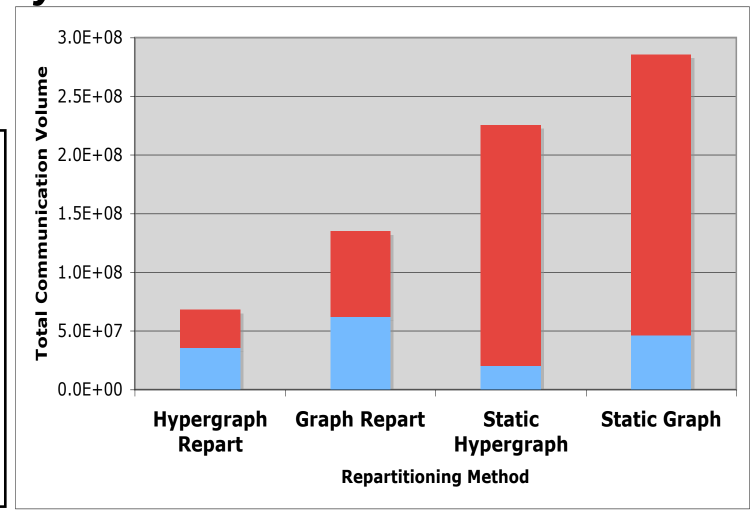
Repartitioning Results: Lower is Better

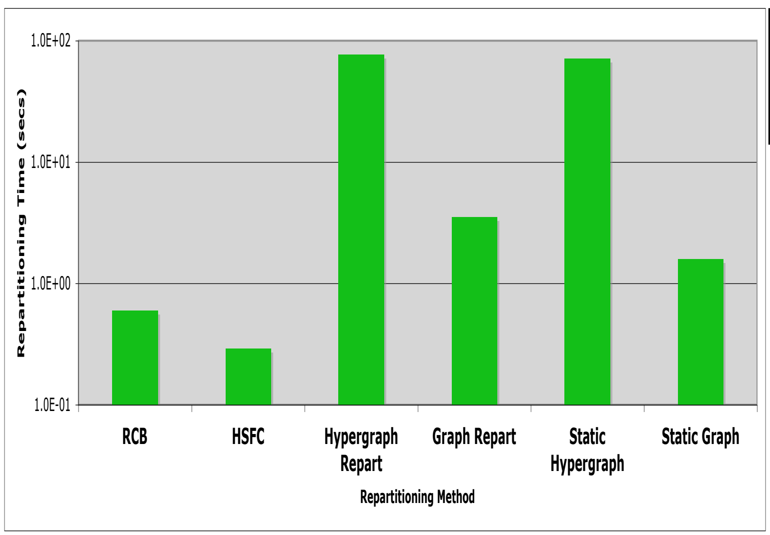
SLAC 6.0M LCLS

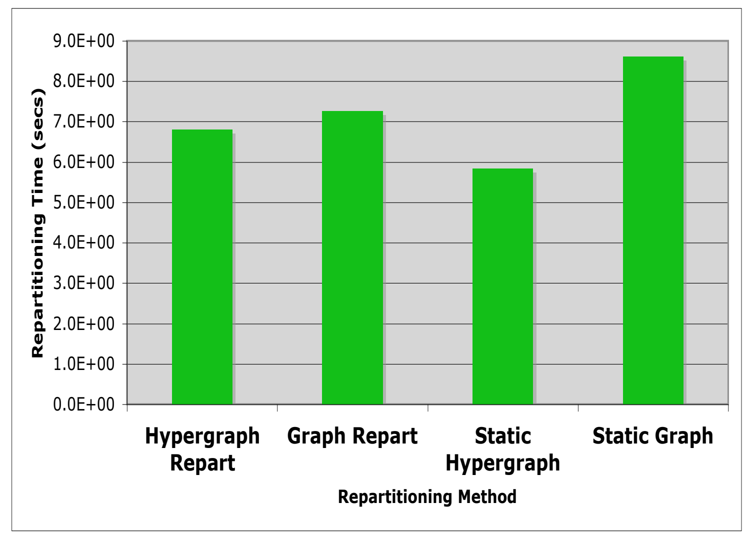




Xyce 680K circuit







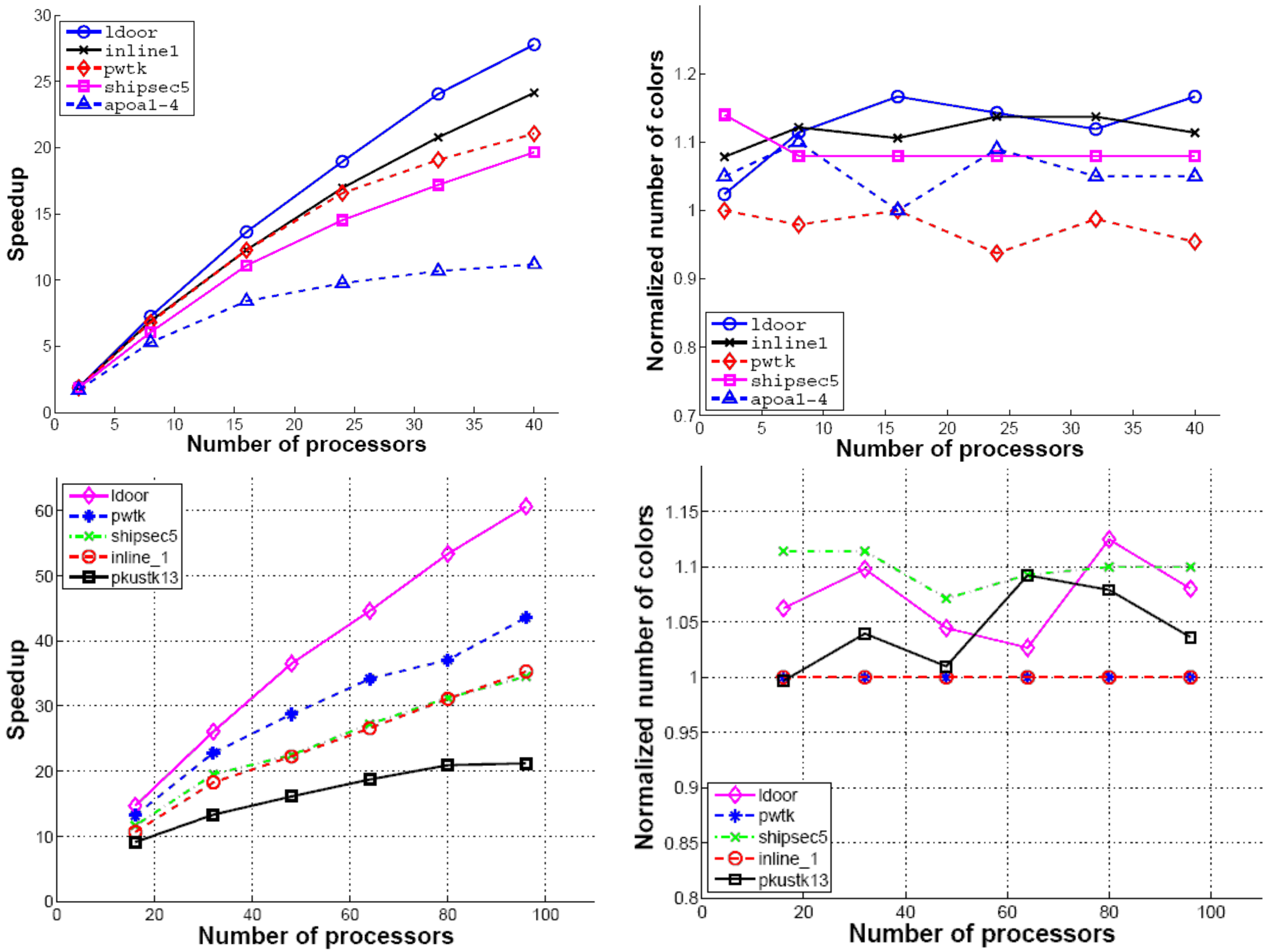
Extra Slides

- **Experimental results: Coloring**

A Parallel Coloring Framework

- Color vertices iteratively in rounds using a first fit strategy
- Each round is broken into supersteps
  - Color a certain number of vertices
  - Exchange recent color information
- Detect conflicts at the end of each round
- Repeat until all vertices receive consistent colors

Experimental Results



Extra Slides

- More details on callback/query functions.

More Details on Query Functions

- **void* data pointer allows user data structures to be used in all query functions.**
  - **To use, cast the pointer to the application data type.**
- **Local IDs provided by application are returned by Zoltan to simplify access of application data.**
  - **E.g. Indices into local arrays of coordinates.**
- **ZOLTAN_ID_PTR is pointer to array of unsigned integers, allowing IDs to be more than one integer long.**
  - **E.g., (processor number, local element number) pair.**
  - **numGlobalIds and numLocalIds are lengths of each ID.**
- **All memory for query-function arguments is allocated in Zoltan.**

```
void ZOLTAN_GET_GEOM_MULTI_FN(void *userDefinedData,
                              int numGlobalIds, int numLocalIds, int numObjs,
                              ZOLTAN_ID_PTR gids, ZOLTAN_ID_PTR lids,
                              int numDim, double *pts, int *err)
```

Example zoltanSimple.c: ZOLTAN_OBJ_LIST_FN


```
void exGetObjectList(void *userDefinedData,
                     int numGlobalIds, int numLocalIds,
                     ZOLTAN_ID_PTR gids, ZOLTAN_ID_PTR lids,
                     int wgt_dim, float *obj_wgts,
                     int *err)
{
/* ZOLTAN_OBJ_LIST_FN callback function.
** Returns list of objects owned by this processor.
** lids[i] = local index of object in array.
*/
  int i;

  for (i=0; i<NumPoints; i++)
  {
    gids[i] = GlobalIds[i];
    lids[i] = i;
  }

  *err = 0;

  return;
}
```

Example zoltanSimple.c: ZOLTAN_GEOM_MULTI_FN

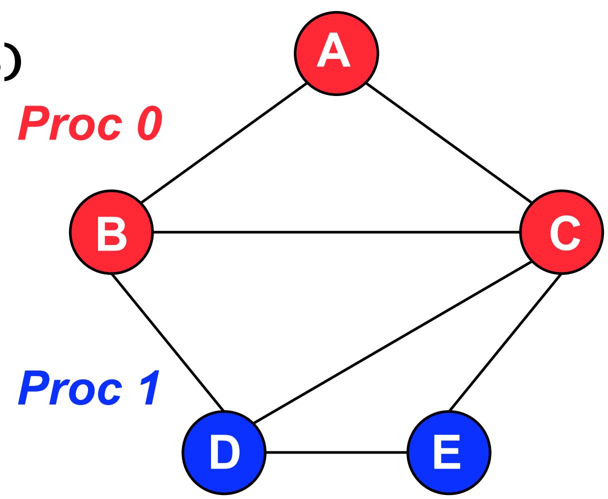

```
void exGetObjectCoords(void *userDefinedData,
                       int numGlobalIds, int numLocalIds, int numObjs,
                       ZOLTAN_ID_PTR gids, ZOLTAN_ID_PTR lids,
                       int numDim, double *pts, int *err)
{
/* ZOLTAN_GEOM_MULTI_FN callback.
** Returns coordinates of objects listed in gids and lids.
*/
  int i, id, id3, next = 0;
  if (numDim != 3) {
    *err = 1; return;
  }
  for (i=0; i<numObjs; i++){
    id = lids[i];
    if ((id < 0) || (id >= NumPoints)) {
      *err = 1; return;
    }
    id3 = lids[i] * 3;
    pts[next++] = (double)(Points[id3]);
    pts[next++] = (double)(Points[id3 + 1]);
    pts[next++] = (double)(Points[id3 + 2]);
  }
}
```

Example Graph Callbacks

```
void ZOLTAN_NUM_EDGES_MULTI_FN(void *data,
  int num_gid_entries, int num_lid_entries,
  int num_obj, ZOLTAN_ID_PTR global_id, ZOLTAN_ID_PTR local_id,
  int *num_edges, int *ierr);
```

```
Proc 0 Input from Zoltan:
  num_obj = 3
  global_id = {A,C,B}
  local_id  = {0,1,2}

Output from Application on Proc 0:
  num_edges = {2,4,3}
              (i.e., degrees of vertices A, C, B)
  ierr = ZOLTAN_OK
```

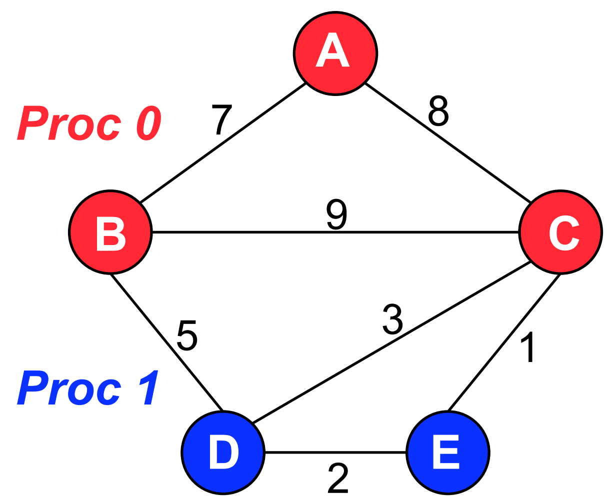


Example Graph Callbacks

```
void ZOLTAN_EDGE_LIST_MULTI_FN(void *data,
  int num_gid_entries, int num_lid_entries,
  int num_obj, ZOLTAN_ID_PTR global_ids, ZOLTAN_ID_PTR local_ids,
  int *num_edges,
  ZOLTAN_ID_PTR nbor_global_id, int *nbor_procs,
  int wdim, float *nbor_ewgts,
  int *ierr);
```

```
Proc 0 Input from Zoltan:
  num_obj = 3
  global_ids = {A, C, B}
  local_ids  = {0, 1, 2}
  num_edges  = {2, 4, 3}
  wdim = 0 or EDGE_WEIGHT_DIM parameter value

Output from Application on Proc 0:
  nbor_global_id = {B, C, A, B, E, D, A, C, D}
  nbor_procs     = {0, 0, 0, 0, 1, 1, 0, 0, 1}
  nbor_ewgts     = if wdim then
                   {7, 8, 8, 9, 1, 3, 7, 9, 5}
  ierr = ZOLTAN_OK
```



Example Hypergraph Callbacks

```
void ZOLTAN_HG_SIZE_CS_FN(void *data, int *num_lists, int *num_pins,
  int *format, int *ierr);
```

```
Output from Application on Proc 0:
  num_lists = 2
  num_pins = 6
  format = ZOLTAN_COMPRESSED_VERTEX
           (owned non-zeros per vertex)
  ierr = ZOLTAN_OK

OR

Output from Application on Proc 0:
  num_lists = 5
  num_pins = 6
  format = ZOLTAN_COMPRESSED_EDGE
           (owned non-zeros per edge)
  ierr = ZOLTAN_OK
```

Vertices

Hyperedges	Proc 0		Proc 1	
	A	B	C	D
a	X			X
b		X		X
c			X	X
d		X		X
e	X		X	X
f	X	X	X	X

Example Hypergraph Callbacks

```
void ZOLTAN_HG_CS_FN(void *data, int num_gid_entries,
  int nvtxedge, int npins, int format,
  ZOLTAN_ID_PTR vtxedge_GID, int *vtxedge_ptr, ZOLTAN_ID_PTR pin_GID,
  int *ierr);
```

```
Proc 0 Input from Zoltan:
  nvtxedge = 2 or 5
  npins = 6
  format = ZOLTAN_COMPRESSED_VERTEX or
           ZOLTAN_COMPRESSED_EDGE

Output from Application on Proc 0:
  if (format = ZOLTAN_COMPRESSED_VERTEX)
      vtxedge_GID = {A, B}
      vtxedge_ptr = {0, 3}
      pin_GID = {a, e, f, b, d, f}
  if (format = ZOLTAN_COMPRESSED_EDGE)
      vtxedge_GID = {a, b, d, e, f}
      vtxedge_ptr = {0, 1, 2, 3, 4}
      pin_GID = {A, B, B, A, A, B}
  ierr = ZOLTAN_OK
```

Vertices

Hyperedges	Proc 0		Proc 1	
	A	B	C	D
a	X			X
b		X		X
c			X	X
d		X		X
e	X		X	X
f	X	X	X	X