Tutorial: Partitioning, Load Balancing and the Zoltan Toolkit

Erik Boman and Karen Devine
Discrete Algorithms and Math Dept.
Sandia National Laboratories, NM

CSCAPES Institute

SciDAC Tutorial, MIT, June 2007

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Outline

Part 1:
- Partitioning and load balancing
  - "Owner computes" approach
- Static vs. dynamic partitioning
- Models and algorithms
  - Geometric (RCB, SFC)
  - Graph & hypergraph

Part 2:
- Zoltan
  - Capabilities
  - How to get it, configure, build
  - How to use Zoltan with your application

Parallel Computing in CS&E

- Parallel Computing Challenge
  - Scientific simulations critical to modern science.
    - Models grow in size, higher fidelity/resolution.
    - Simulations must be done on parallel computers.
  - Clusters with 64-256 nodes are widely available.
  - High-performance computers have 100,000+ processors.
    - How can we use such machines efficiently?

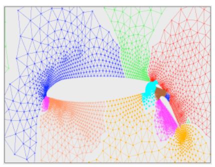
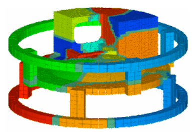
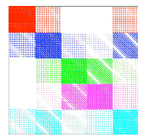
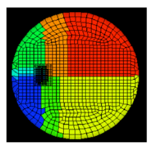
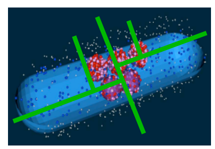
Parallel Computing Approaches

- We focus on distributed memory systems.
  - Two common approaches:
- Master–slave
  - A "master" processor is a global synchronization point, hands out work to the slaves.
- 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.

Slide 5

Partitioning and Load Balancing

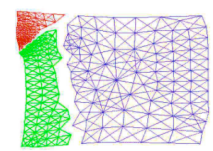
- Assignment of application data to processors for parallel computation.
- Applied to grid points, elements, matrix rows, particles, ….

Slide 6

Partitioning Goals

- Minimize total execution time by…
  - Minimizing processor idle time.
    - Load balance data and work.
  - Keeping inter-processor communication low.
    - Reduce total volume, max volume.
    - Reduce number of messages.

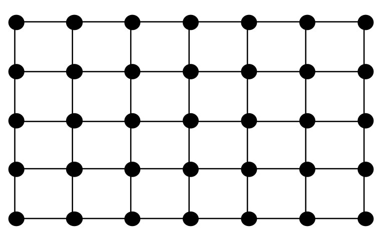


Partition of an unstructured finite element mesh for three processors

Slide 7

"Simple" Example (1)

- Finite difference method.
  - Assign equal numbers of grid points to processors.
  - Keep amount of data communicated small.

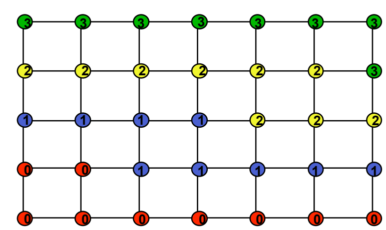


7x5 grid
5-point stencil
4 processors

Slide 8

"Simple" Example (2)

- Finite difference method.
  - Assign equal numbers of grid points to processors.
  - Keep amount of data communicated small.

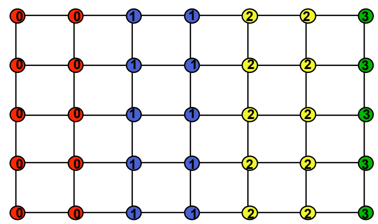


Max Data Comm: 14
Total Volume: 42
Max Nbor Proc: 2
Max Imbalance: 3%

First 35/4 points to processor 0; next 35/4 points to processor 1; etc.

"Simple" Example (3)

- Finite difference method.
  - Assign equal numbers of grid points to processors.
  - Keep amount of data communicated small.

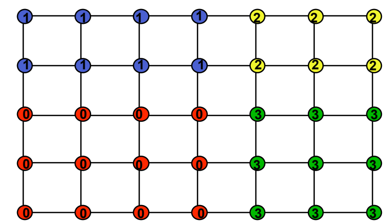


Max Data Comm: 10
Total Volume: 30
Max Nbor Proc: 2
Max Imbalance: 14%

One-dimensional striped partition

"Simple" Example (4)

- Finite difference method.
  - Assign equal numbers of grid points to processors.
  - Keep amount of data communicated small.



Max Data Comm: 7
Total Volume: 26
Max Nbor Proc: 2
Max Imbalance: 37%

Two-dimensional structured grid partition

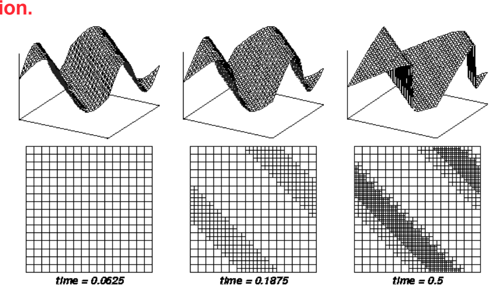
Static Partitioning

Initialize Application → Partition Data → Distribute Data → Compute Solutions → Output & End

- 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.

Dynamic Applications

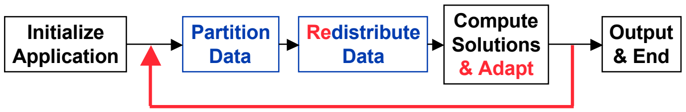
- Characteristics:
  - Work per processor is unpredictable or changes during a computation; and/or
  - Locality of objects changes during computations.
  - Dynamic redistribution of work is needed during computation.
- Example: adaptive mesh refinement (AMR) methods



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

Slide 13

Sandia National Laboratories



- 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.

Static vs. Dynamic: Usage and Implementation

Slide 14

Sandia National Laboratories

- Static:
  - Pre-processor to application.
  - Can be implemented serially.
  - May be slow, expensive.
  - File-based interface acceptable.
  - No consideration of existing decomposition required.

- Dynamic:
  - Must run side-by-side with application.
  - Must be implemented in parallel.
  - Must be fast, scalable.
  - Library application interface required.
  - Should be easy to use.
  - Incremental algorithms preferred.
    - Small changes in input result small changes in partitions.
    - Explicit or implicit incrementality acceptable.

Two Types of Models/Algorithms

Slide 15

Sandia National Laboratories

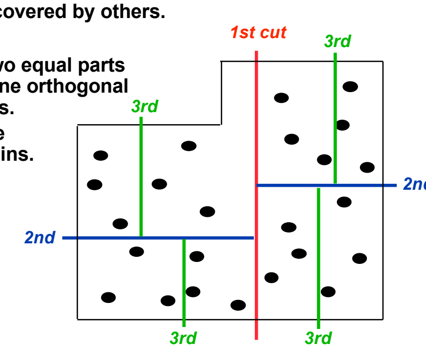
- Geometric
  - Computations are tied to a geometric domain.
  - Coordinates for data items are available.
  - Geometric locality is loosely correlated to data dependencies.
- Combinatorial (topological)
  - No geometry .
  - Connectivity among data items is known.
    - Represent as graph or hypergraph.

Recursive Coordinate Geometric Bisection (RCB)

Slide 16

Sandia National Laboratories

- Developed by Berger & Bokhari (1987) for AMR.
  - Independently discovered by others.
- Idea:
  - Divide work into two equal parts using a cutting plane orthogonal to a coordinate axis.
  - Recursively cut the resulting subdomains.

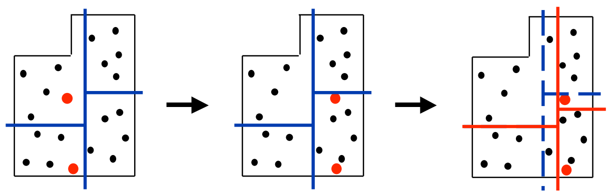


4

Slide 17

RCB Repartitioning

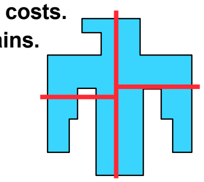
- Implicitly incremental.
- Small changes in data results in small movement of cuts.



Slide 18

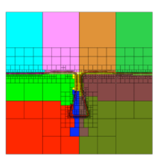
RCB Advantages and Disadvantages

- Advantages:
  - Conceptually simple; fast and inexpensive.
  - Regular subdomains.
    - Can be used for structured or unstructured applications.
    - All processors can inexpensively know entire decomposition.
  - Effective when connectivity info is not available.
- Disadvantages:
  - No explicit control of communication costs.
  - Can generate disconnected subdomains.
  - Mediocre partition quality.
  - Geometric coordinates needed.

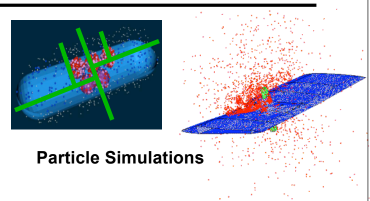


Slide 19

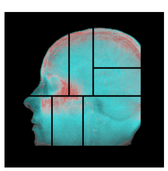
Applications of RCB



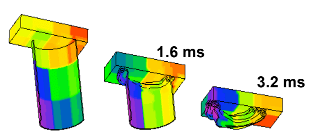
Adaptive Mesh Refinement



Particle Simulations



Parallel Volume Rendering

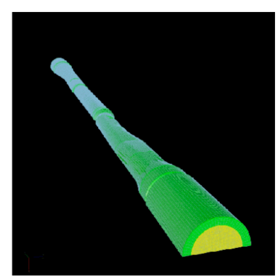


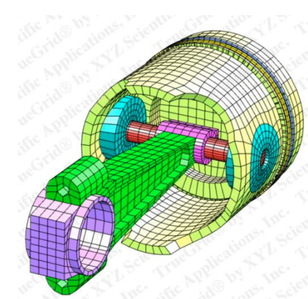
Crash Simulations and Contact Detection

Slide 20

Variations on RCB : RIB

- Recursive Inertial Bisection
  - Simon, Taylor, et al., 1991
  - Cutting planes orthogonal to principle axes of geometry.
  - Not incremental.

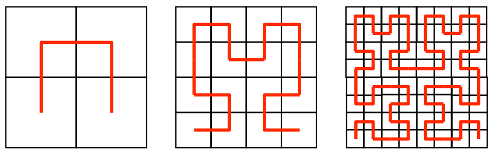




Slide 21

Space-Filling Curve Partitioning (SFC)

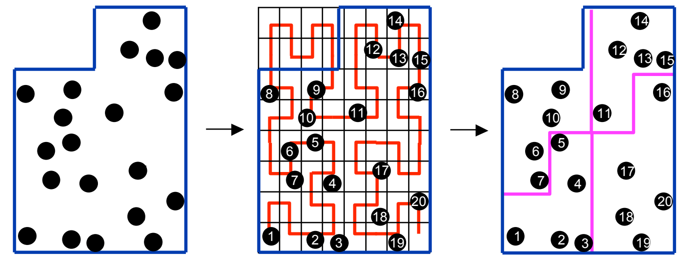
- Developed by Peano, 1890.
- Space-Filling Curve:
  - Mapping between R^3 to R^1 that completely fills a domain.
  - Applied recursively to obtain desired granularity.
- Used for partitioning by …
  - Warren and Salmon, 1993, gravitational simulations.
  - Pilkington and Baden, 1994, smoothed particle hydrodynamics.
  - Patra and Oden, 1995, adaptive mesh refinement.



Slide 22

SFC Algorithm

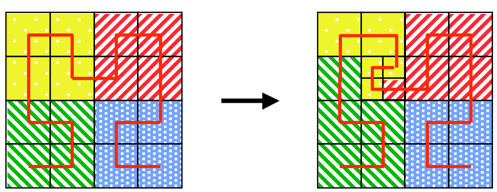
- Run space-filling curve through domain.
- Order objects according to position on curve.
- Perform 1-D partition of curve.



Slide 23

SFC Repartitioning

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



Slide 24

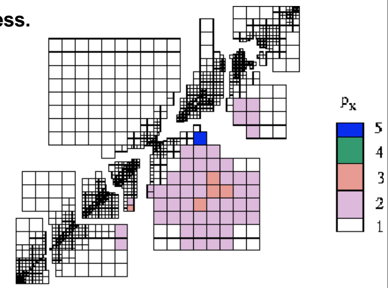
SFC Advantages and Disadvantages

- Advantages:
  - Simple, fast, inexpensive.
  - Maintains geometric locality of objects in processors.
  - Linear ordering of objects may improve cache performance.
- Disadvantages:
  - No explicit control of communication costs.
  - Can generate disconnected subdomains.
  - Often lower quality partitions than RCB.
  - Geometric coordinates needed.

Slide 25

Applications using SFC

- Adaptive hp-refinement finite element methods.
  - Assigns physically close elements to same processor.
  - Inexpensive; incremental; fast.
  - Linear ordering can be used to order elements for efficient memory access.

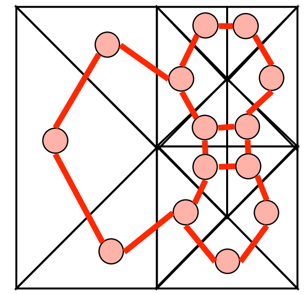


hp-refinement mesh; 8 processors.
Patra, et al. (SUNY-Buffalo)

Slide 26

Graph Partitioning

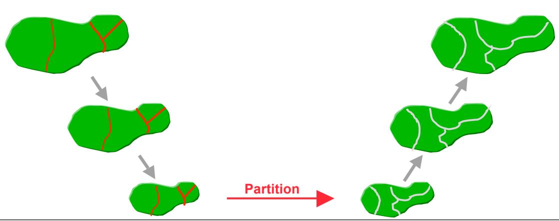
- Represent problem as a weighted graph.
  - Vertices = objects to be partitioned.
  - Edges = communication between objects.
  - Weights = work load or amount of communication.
- Partition graph so that …
  - Partitions have equal vertex weight.
  - Weight of edges cut by subdomain boundaries is small.



Slide 27

Multi-Level Graph Partitioning

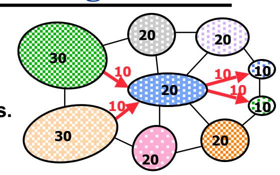
- Bui & Jones (1993); Hendrickson & Leland (1993); Karypis and Kumar (1995)
- Construct smaller approximations to graph.
- Perform graph partitioning on coarse graph.
- Propagate partition back, refining as needed.



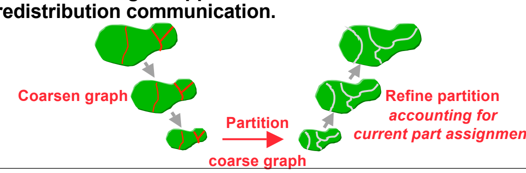
Slide 28

Graph Repartitioning

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



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

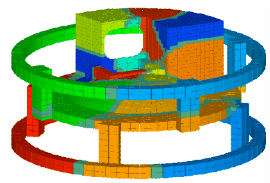


Graph Partitioning Advantages and Disadvantages

Slide 29

Sandia National Laboratories

- Advantages:
  - High quality partitions for many applications.
  - Explicit control of communication costs.
  - Widely used for static partitioning (Chaco, METIS, Jostle, Party, Scotch)
- Disadvantages:
  - More expensive than geometric approaches.
  - Not incremental.

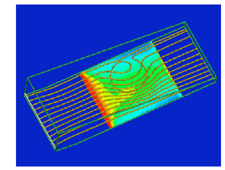


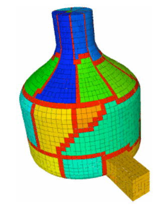
Applications using Graph Partitioning

Slide 30

Sandia National Laboratories

- Finite element analysis
- Multiphysics simulations
  - Difficult to estimate work in advance.
  - Rebalance infrequently; want high quality.
- Linear solvers and preconditioners
  - Square, structurally symmetric.
  - Decomposition of mesh induces good decomposition for solver.



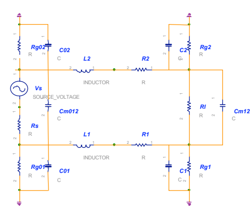


Applications using Graph Partitioning

Slide 31

Sandia National Laboratories

- XYCE (S. Hutchinson, R. Hoekstra, E. Keiter, SNL)
  - Massively parallel analog circuit simulator.
- Load balancing in XYCE.
  - Balance linear solve phase.
  - Equal number of rows while minimizing cut edges.
  - Partition matrix solver separately from matrix fill.
  - Trilinos solver library (Heroux, et al.) uses Zoltan to partition matrix.



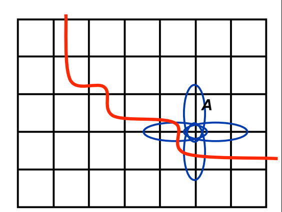
- Matrix structure more complex than mesh-based applications.
  - Is there a better partitioning model?

Flaws in the Graph Model

Slide 32

Sandia National Laboratories

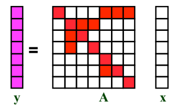
- Graph model and partitioning approach has been successful in scientific computing, BUT…
- Graph models assume # edge cuts = communication volume.
- In reality…
  - Edge cuts are not equal to communication volume.



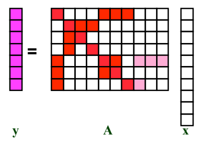
Slide 33

Graph Models: Applicability

- Graph models assume symmetric square problem.
  - Symmetric == undirected graph.
  - Square == inputs and outputs of operation are same size.
- Non-symmetric systems.
  - Require directed or bipartite graph.



- Rectangular systems.
  - Require decompositions for differently sized inputs and outputs.



Slide 34

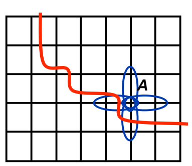
Is the Graph Model "good enough"?

- Mesh-based applications: Yes, maybe.
  - Graph partitioning works well in practice.
  - Geometric structure of meshes ensures…
    - Small separators and good partitions.
    - Low vertex degrees give small error in graph model.
- Irregular non-mesh applications: No.
  - Graph model is poor or does not apply.
  - Ex: circuit simulation, discrete optimization, data mining.
  - Nonsymmetric and rectangular matrices.

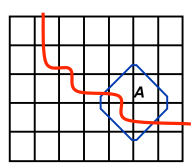
Slide 35

Hypergraph Partitioning

- Hypergraph model (Aykanat & Catalyurek)
  - Vertices represent computations.
  - Hyperedges connect all objects which produce/use datum.
    - Hyperedges connect one or more vertices (cf. graph edge always two)
  - Greater expressiveness than graph partitioners.
    - Non-symmetric data dependencies.
    - Rectangular matrices.
  - Cut hyperedges == communication volume!



Graph model only approximates communication volume.



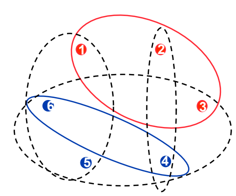
Hypergraph model accurately measures communication volume.

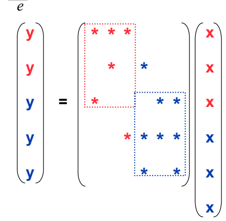
Slide 36

Matrices and Hypergraphs

- View matrix as hypergraph (Çatalyürek & Aykanat)
  - Vertices == columns
  - Edges == rows
- Partition vertices (columns in matrix)
- Communication volume associated with edge e:

  CV_e = (# processors in edge e) - 1
- Total communication volume = $\sum_e CV_e$





Slide 37

Hypergraph Repartitioning

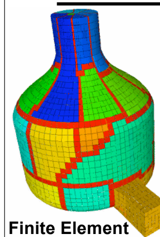
- Augment hypergraph with data redistribution costs.
  - Account for data's current processor assignments.
  - Weight hyperedges by data redistribution size or frequency of use.
- Hypergraph partitioning then attempts to minimize *total communication volume*:

  Data redistribution volume
  + Application communication volume
  Total communication volume

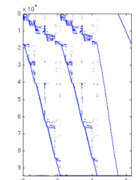
- Trade-off between application volume and redistribution cost controlled by single knob (user parameter).
  - PHG_REPART_MULTIPLIER should be (roughly) number of application communications between repartitions.
- Can re-use existing algorithms and software.
  - This approach also works for graphs.

Slide 38

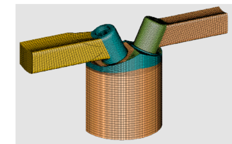
Hypergraph Applications



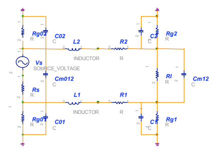
Finite Element Analysis



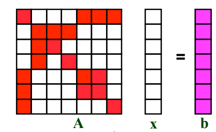
Linear programming for sensor placement



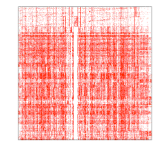
Multiphysics and multiphase simulations



Circuit Simulations



Linear solvers & preconditioners (no restrictions on matrix structure)



Data Mining

Slide 39

Hypergraph Partitioning: Advantages and Disadvantages

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

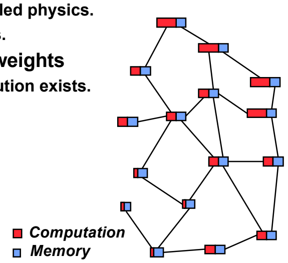
Slide 40

Using Weights

- Some data items may have more work than others.
- Solution: Specify work (load) using weights.
  - By default, all data items have unit weights.
  - Objective is to balance sum of weights.
- Geometric methods:
  - Add a weight for each point.
- Graph/hypergraph methods:
  - One weight per vertex.
  - Can also weight edges with communication size.

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 meshes in loosely coupled physics.
    - Balance multi-phase simulations.
  - Extend algorithms to multiple weights
    - Difficult. No guarantee good solution exists.



Heterogeneous Architectures

- Clusters may have different types of processors.
- Assign "capacity" weights to processors.
  - Compute power (speed)
  - Memory
- Partitioner should balance with respect to processor capacity.

Example & Recap

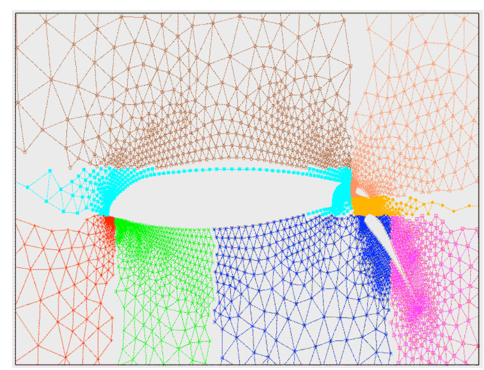
- Hammond airfoil mesh
- 2d mesh, triangular elements
  - 5K vertices
  - 13K edges
- Partition into 8 parts

RCB

Total Volume:

826

Max #mesg: 6



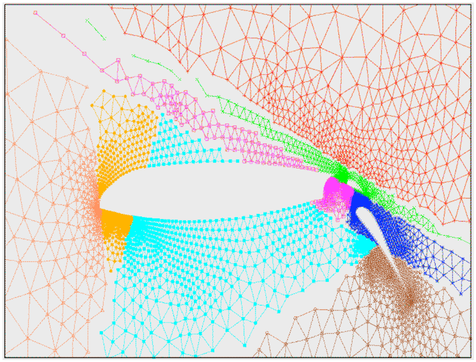
Slide 45

Sandia National Laboratories

RIB

Total volume: 922

Max #mesg: 5



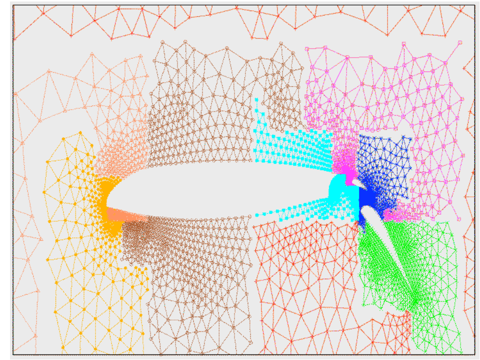
Slide 46

Sandia National Laboratories

HSFC

Total volume: 1000

Max #mesg: 6



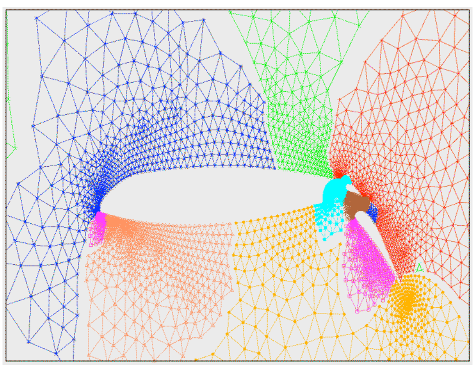
Slide 47

Sandia National Laboratories

Graph

Total volume: 472

Max #mesg: 5



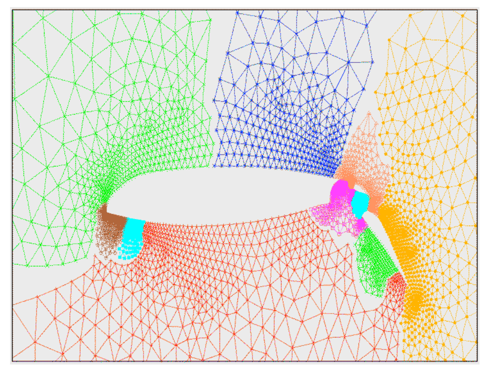
Slide 48

Sandia National Laboratories

Hypergraph

Total volume: 464

Max #mesg: 6



Slide 49

Coffee Break!

Slide 50

Software

- Geometric partitioners
  - Often embedded in application code;
    - Cannot easily be re-used.
- Graph/hypergraph partitioners
  - Multilevel partitioners are so complex they can take several man-years to implement.
  - Abstraction allows partitioners to be used across many applications.

Slide 51

Software

- 1990s: Many graph partitioners
  - Chaco (Sandia)
  - Metis/ParMetis (U. Minnesota)
  - Jostle/PJostle (U. Greenwich)
  - Scotch (U. Bordeaux)
  - Party (U. Paderborn)
- Great advance at the time, but…
  - Single algorithm is not best for all applications.
  - Interface requires application to build specific graph data structure.

Slide 52

Our Approach: Zoltan Toolkit

- Construct applications from smaller software “parts.”
- “Tinker-toy parallel computing” -- B. Hendrickson
- Toolkits include …
  - Services applications commonly need.
  - Support for wide range of applications.
  - Easy-to-use interfaces.
  - Data-structure neutral design.
- Toolkits avoid …
  - Prescribed data structures
  - Heavy framework
  - Limited freedom for application developers.
- Zoltan: Toolkit of Parallel Data Management Tools for Parallel, Unstructured Applications.

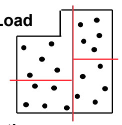
Hasbro, Inc.

Slide 53

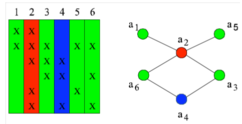
The Zoltan Toolkit

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

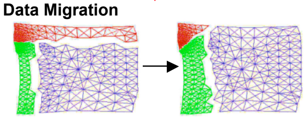
Dynamic Load Balancing



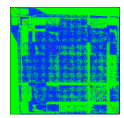
Graph Coloring



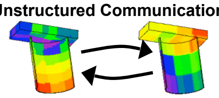
Data Migration



Matrix Ordering



Unstructured Communication



Distributed Data Directories

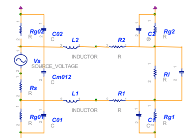
A	B	C	D	E	F	G	H	I
0	1	0	2	1	0	1	2	1

Dynamic Memory Debugging

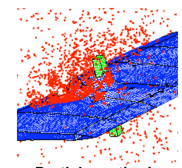
Slide 54

Zoltan Supports Many Applications

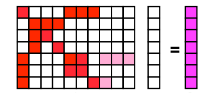
- Different applications, requirements, data structures.



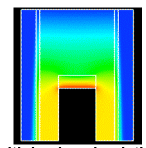
Parallel electronics networks



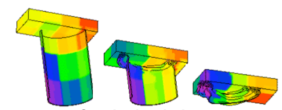
Particle methods



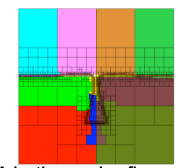
Linear solvers & preconditioners



Multiphysics simulations



Crash simulations



Adaptive mesh refinement

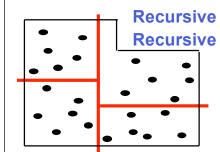
Slide 55

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.

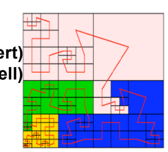
Slide 56

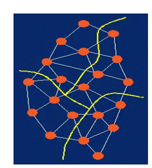
Zoltan Toolkit: Suite of Partitioners



Recursive Coordinate Bisection (Berger, Bokhari)
Recursive Inertial Bisection (Taylor, Nour-Omid)

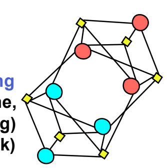
Space Filling Curves (Peano, Hilbert)
Refinement-tree Partitioning (Mitchell)





Graph Partitioning
ParMETIS (Karypis, Schloegel, Kumar)
Jostle (Walshaw)

Hypergraph Partitioning & Repartitioning
(Catalyurek, Aykanat, Boman, Devine, Heaphy, Karypis, Bisseling)
PaToH (Catalyurek)



Slide 57

Zoltan Interface Design

- Common interface to each class of partitioners.
- Partitioning 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.

Slide 58

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. For example:
  - Global element number.
  - Global matrix row number.
  - (Processor number, local element number)
  - (Processor number, local particle number)

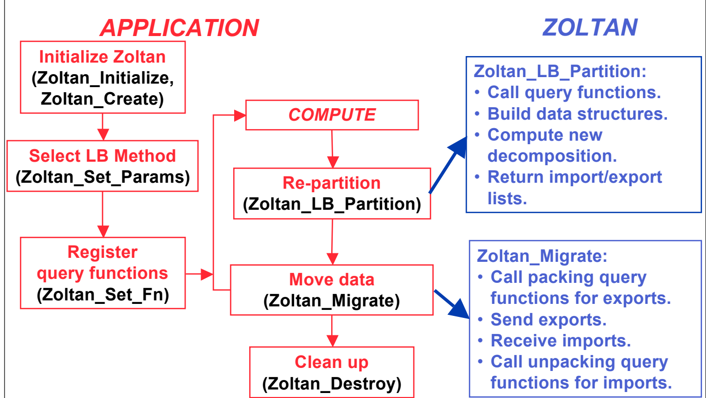
Slide 59

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.

Slide 60

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.

For geometric partitioning (RCB, RIB, HSFC), use …

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.

For graph partitioning, coloring & ordering, use …

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.

For hypergraph partitioning and repartitioning, use …

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.

16

Slide 65

Or can use graph queries to build hypergraph.

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.

Slide 66

Using Zoltan in Your Application

1. Decide what your objects are.
   - Elements? Grid points? Matrix rows? Particles?
2. Decide which class of method to use (geometric/graph/hypergraph).
3. Download and build Zoltan.
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. Compile; link application with libzoltan.a.
   - mpicc application.c -lzoltan

Slide 67

Typical Applications

- **Unstructured meshes:**
  - Nodes, edges, and faces all need be distributed.
  - Several choices:
    - Nodes are Zoltan objects (primal graph)
    - Faces are Zoltan objects (dual graph)
- **Sparse matrices:**
  - Partition rows or columns?
  - Balance rows or nonzeros?
- **Particle methods:**
  - Partition particles or cells weighted by particles?

Slide 68

Zoltan: Getting Started

- Requirements:
  - C compiler
  - GNU Make (gmake)
  - MPI library (Message Passing Interface)
- Download Zoltan from Zoltan web site
  - http://www.cs.sandia.gov/Zoltan
  - Select "Download Zoltan" button.
  - Submit the registration form.
  - Choose the version you want; we suggest the latest version v3.0!
  - Downloaded file is zoltan_distrib_v3.0.tar.gz.

Configuring and Building Zoltan

- Create and enter the Zoltan directory:
  - gunzip zoltan_distrib_v3.0.tar.gz
  - tar xf zoltan_distrib_v3.0.tar
  - cd Zoltan
- Configure and make Zoltan library
  - Not autotooled; uses manual configuration file.
  - "make zoltan" attempts a generic build; library libzoltan.a is in directory Obj_generic.
  - To customize your build:
    - cd Utilities/Config; cp Config.linux Config.your_system
    - Edit Config.your_system
    - cd ../..
    - setenv ZOLTAN_ARCH your_system
    - make zoltan
    - Library libzoltan.a is in Obj_your_system

Config file

```
DEFS                    =
RANLIB                  = ranlib
AR                      = ar r

CC                      = mpicc -Wall
CPPC                    = mpic++
INCLUDE_PATH            =
DBG_FLAGS               = -g
OPT_FLAGS               = -O
CFLAGS                  = $(DBG_FLAGS)

F90                     = mpif90
LOCAL_F90               = f90
F90CFLAGS               = -DFMANGLE=UNDERSCORE -DNO_MPI2
FFLAGS                  =
SPPR_HEAD               = spprinc.most
F90_MODULE_PREFIX       = -I
FARG                    = farg_typical

MPI_LIB                 =
MPI_LIBPATH             =

PARMETIS_LIBPATH        = -L/Users/kddevin/code/ParMETIS3_1
PARMETIS_INCPATH        = -I/Users/kddevin/code/ParMETIS3_1
#PATOH_LIBPATH           = -L/Users/kddevin/code/PaToH
#PATOH_INCPATH           = -I/Users/kddevin/code/PaToH
```


Simple Example

- Zoltan/examples/C/zoltanSimple.c
- Application data structure:
  - int MyNumPts;
    - Number of points on processor.
  - int *Gids;
    - array of Global ID numbers of points on processor.
  - float *Pts;
    - Array of 3D coordinates of points on processor (in same order as Gids array).

Example zoltanSimple.c: Initialization

```
/* Initialize MPI */
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &me);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

/*
** Initialize application data.  In this example,
** create a small test mesh and divide it across processors
*/

exSetDivisions(32);    /* rectilinear mesh is div X div X div */

MyNumPts = exInitializePoints(&Pts, &Gids, me, nprocs);

/*  Initialize Zoltan */
rc = Zoltan_Initialize(argc, argv, &ver);

if (rc != ZOLTAN_OK){
  printf("sorry...\n");
  free(Pts); free(Gids);
  exit(0);
}
```

Example zoltanSimple.c: Prepare for Partitioning

Slide 73

```
/* Allocate and initialize memory for Zoltan structure */
zz = Zoltan_Create(MPI_COMM_WORLD);

/* Set general parameters */
Zoltan_Set_Param(zz, "DEBUG_LEVEL", "0");
Zoltan_Set_Param(zz, "LB_METHOD", "RCB");
Zoltan_Set_Param(zz, "NUM_GID_ENTRIES", "1");
Zoltan_Set_Param(zz, "NUM_LID_ENTRIES", "1");
Zoltan_Set_Param(zz, "RETURN_LISTS", "ALL");

/* Set RCB parameters */
Zoltan_Set_Param(zz, "KEEP_CUTS", "1");
Zoltan_Set_Param(zz, "RCB_OUTPUT_LEVEL", "0");
Zoltan_Set_Param(zz, "RCB_RECTILINEAR_BLOCKS", "1");

/* Register call-back query functions. */
Zoltan_Set_Num_Obj_Fn(zz, exGetNumberOfAssignedObjects, NULL);
Zoltan_Set_Obj_List_Fn(zz, exGetObjectList, NULL);
Zoltan_Set_Num_Geom_Fn(zz, exGetObjectSize, NULL);
Zoltan_Set_Geom_Multi_Fn(zz, exGetObject, NULL);
```

Example zoltanSimple.c: Partitioning

Slide 74

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)

```
/* Perform partitioning */
rc = 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);
```

Example zoltanSimple.c: Use the Partition

Slide 75

```
/* Process partitioning results;
** in this case, print information;
** in a "real" application, migrate data here.
*/
if (!rc){
  exPrintGlobalResult("Recursive Coordinate Bisection",
                      nprocs, me,
                      MyNumPts, numImport, numExport, changes);
}
else{
  free(Pts);
  free(Gids);
  Zoltan_Destroy(&zz);
  MPI_Finalize();
  exit(0);
}
```

Example zoltanSimple.c: Cleanup

Slide 76

```
/* Free Zoltan memory allocated by Zoltan_LB_Partition. */
Zoltan_LB_Free_Part(&importGlobalGids, &importLocalGids,
                    &importProcs, &importToPart);
Zoltan_LB_Free_Part(&exportGlobalGids, &exportLocalGids,
                    &exportProcs, &exportToPart);

/* Free Zoltan memory allocated by Zoltan_Create. */
Zoltan_Destroy(&zz);

/* Free Application memory */
free(Pts); free(Gids);

/**********************
** all done ***********
**********************/

MPI_Finalize();
```

Slide 77

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;
}
```

Slide 78

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]);
  }
}
```

Slide 79

Example Graph Callbacks

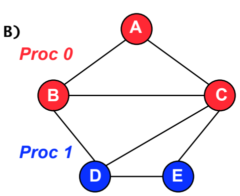
```
void ZOLTAN_NUM_EDGES_MULTI_FN(void *data,
  int num_gid_entries, int num_lid_entries,
  int num_obj, ZOLTAN_ID_PTR global_id, ZOLTAN_ID_PTR local_id,
  int *num_edges, int *ierr);
```

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
```



Slide 80

Example Graph Callbacks

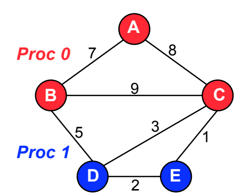
```
void ZOLTAN_EDGE_LIST_MULTI_FN(void *data,
  int num_gid_entries, int num_lid_entries,
  int num_obj, ZOLTAN_ID_PTR global_ids, ZOLTAN_ID_PTR local_ids,
  int *num_edges,
  ZOLTAN_ID_PTR nbor_global_id, int *nbor_procs,
  int wdim, float *nbor_ewgts,
  int *ierr);
```

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
```



Slide 81

More Details on Query Functions

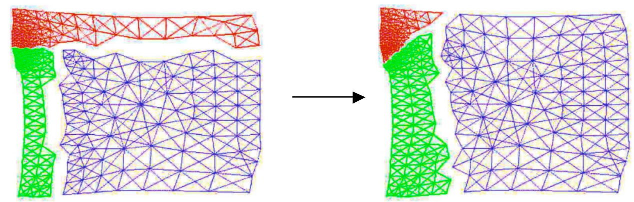
- void* data pointer allows user data structures to be used in all query functions.
  - To use, cast the pointer to the application data type.
- Local IDs provided by application are returned by Zoltan to simplify access of application data.
  - E.g. Indices into local arrays of coordinates.
- `ZOLTAN_ID_PTR` 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)
```

Slide 82

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.



Slide 83

Using Zoltan's Data Migration Tools

- Required migration query functions:
  - ZOLTAN_OBJ_SIZE_MULTI_FN:
    - Returns size of data (in bytes) for each object to be exported to a new processor.
  - ZOLTAN_PACK_MULTI_FN:
    - Remove data from application data structure on old processor;
    - Copy data to Zoltan communication buffer.
  - ZOLTAN_UNPACK_MULTI_FN:
    - Copy data from Zoltan communication buffer into data structure on new processor.

```
int Zoltan_Migrate(struct Zoltan_Struct *zz,
       int num_import, ZOLTAN_ID_PTR import_global_ids,
       ZOLTAN_ID_PTR import_local_ids, int *import_procs,
       int *import_to_part,
       int num_export, ZOLTAN_ID_PTR export_global_ids,
       ZOLTAN_ID_PTR export_local_ids, int *export_procs,
       int *export_to_part);
```

Slide 84

Other Zoltan Functionality

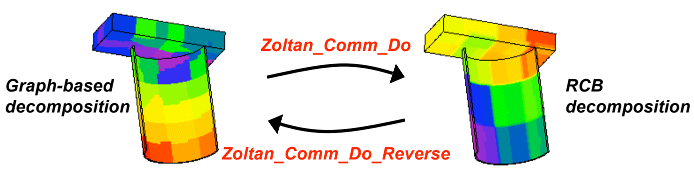
- Tools needed when doing dynamic load balancing:
  - Unstructured Communication Primitives
  - Distributed Data Directories
- Tools closely related to graph partitioning:
  - Graph coloring
  - Matrix ordering
  - These tools use the same query functions as graph partitioners.
- All functionality described in Zoltan User's Guide.
  - http://www.cs.sandia.gov/Zoltan/ug_html/ug.html

21

Slide 85

Zoltan Unstructured Communication Package

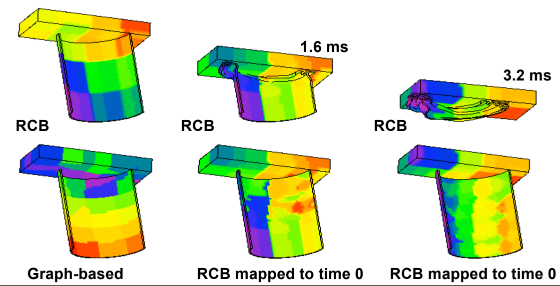
- Simple primitives for efficient irregular communication.
  - Zoltan_Comm_Create: Generates communication plan.
    - Processors and amount of data to send and receive.
  - Zoltan_Comm_Do: Send data using plan.
    - Can reuse plan. (Same plan, different data.)
  - Zoltan_Comm_Do_Reverse: Inverse communication.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



Slide 86

Example Application: Crash Simulations

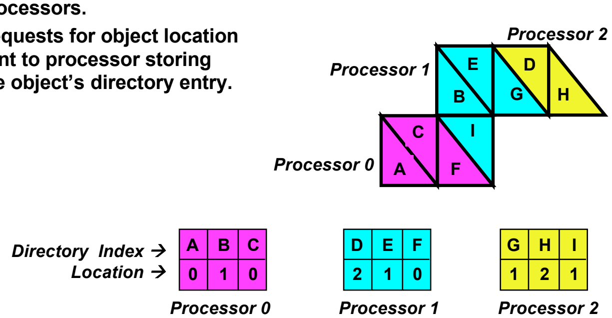
- Multiphase simulation:
  - Graph-based decomposition of elements for finite element calculation.
  - Dynamic geometric decomposition of surfaces for contact detection.
  - Migration tools and Unstructured Communication package map between decompositions.



Slide 87

Zoltan Distributed Data Directory

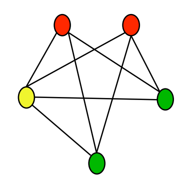
- Helps applications locate off-processor data.
- Rendezvous algorithm (Pinar, 2001).
  - Directory distributed in known way (hashing) across processors.
  - Requests for object location sent to processor storing the object's directory entry.



Slide 88

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.
- Implemented algorithms due to Bozdag, Catalyurek, Gebremedhin, Manne, Boman, 2005.

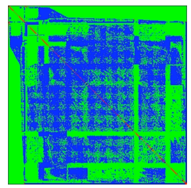


Slide 89

Sandia National Laboratories

Zoltan Matrix Ordering Interface

- Produce fill-reducing ordering for sparse matrix factorization.
- Graph built using same application interface and code as graph partitioners.
- Generic ordering interface; easy to add new ordering algorithms.
- Specific interface to ordering methods in ParMETIS (Karypis, et al., U. Minnesota).



Slide 90

Sandia National Laboratories

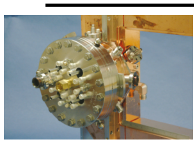
Performance Results

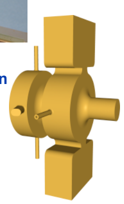
- Experiments on Sandia's Thunderbird cluster.
  - Dual 3.6 GHz Intel EM64T processors with 6 GB RAM.
  - Infiniband network.
- Compare RCB, graph and hypergraph methods.
- Measure …
  - Amount of communication induced by the partition.
  - Partitioning time.

Slide 91

Sandia National Laboratories

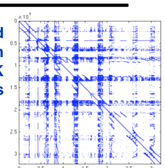
Test Data



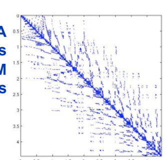


SLAC *LCLS
Radio Frequency Gun
6.0M x 6.0M
23.4M nonzeros

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

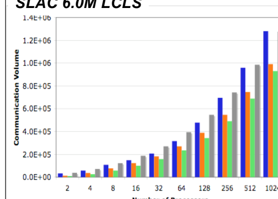
Slide 92

Sandia National Laboratories

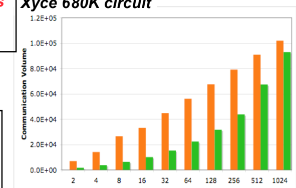
Communication Volume: Lower is Better

Number of parts = number of processors.

SLAC 6.0M LCLS

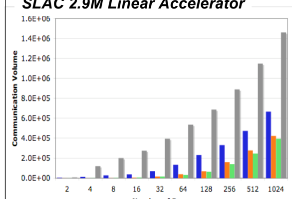


Xyce 680K circuit

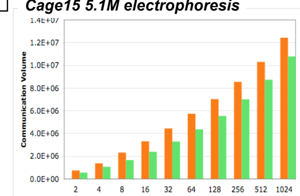


RCB
Graph
Hypergraph
HSFC

SLAC 2.9M Linear Accelerator



Cage15 5.1M electrophoresis

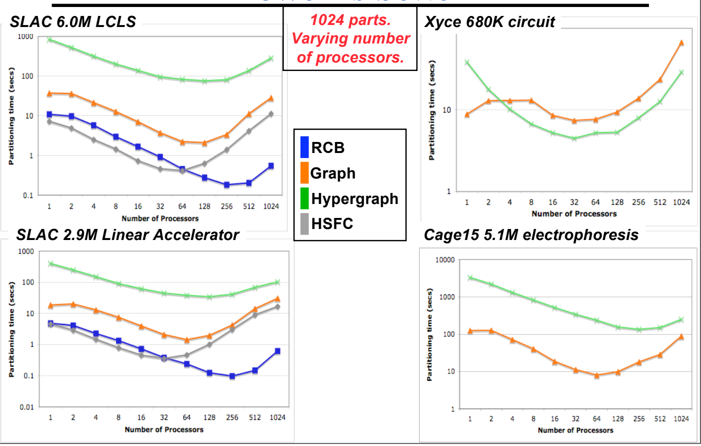


23

Partitioning Time: Lower is better

Slide 93

Sandia National Laboratories



Repartitioning Experiments

Slide 94

Sandia National Laboratories

- 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

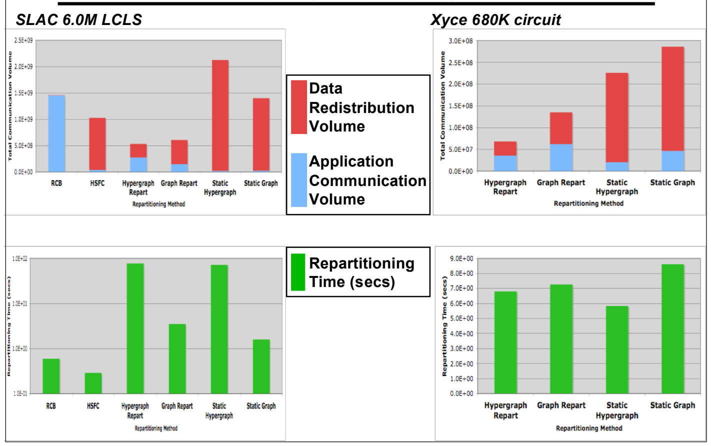
  + Application communication volume

  Total communication volume

Repartitioning Results: Lower is Better

Slide 95

Sandia National Laboratories



Summary

Slide 96

Sandia National Laboratories

- No one-size-fits-all solutions for partitioning.
- Different methods for different applications
  - Geometric vs. combinatorial/topological
  - Static vs. dynamic problem
- Zoltan toolkit has it all (almost…)
  - Provides collection of load-balance methods
  - Also provides other common parallel services
  - Frees the application developer to focus on his/her specialty area
  - Easy to test and compare different methods

Slide 97

For More Information...

- Zoltan Home Page
  - http://www.cs.sandia.gov/Zoltan
  - User's and Developer's Guides
  - Download Zoltan software under GNU LGPL.
- Email:
  - {egboman,kddevin}@sandia.gov

Slide 98

The End

Slide 99

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 (Proc 0: A, B; Proc 1: C, D) × Hyperedges

Hyperedges	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

Slide 100

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 (Proc 0: A, B; Proc 1: C, D) × Hyperedges

Hyperedges	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