Dynamic Load Balancing and Partitioning using the Zoltan Toolkit

Karen Devine, Erik Boman, Vitus Leung, Lee Ann Riesen
Sandia National Laboratories

Umit Çatalyürek
Ohio State University

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

Dynamic Memory Debugging
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.
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.

What makes a partition “good,” especially at petascale?

- Balanced work loads.
  - Even small imbalances result in many wasted processors!
    • 50,000 processors with one processor 5% over average workload is equivalent to 2380 idle processors and 47,620 perfectly balanced processors.

- Low interprocessor communication costs.
  - Processor speeds increasing faster than network speeds.
  - Partitions with minimal communication costs are critical.

- Scalable partitioning time and memory use.
  - Scalability is especially important for dynamic partitioning.

- Low data redistribution costs (for dynamic partitioning).
  - Redistribution costs must be recouped through reduced total execution time.
Partitioning Algorithms in the Zoltan Toolkit

Geometric (coordinate-based) methods

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

Hypergraph and graph (connectivity-based) methods

- Hypergraph Partitioning
- Hypergraph Repartitioning
- PaToH (Catalyurek & Aykanat)

Zoltan Graph Partitioning
- ParMETIS (U. Minnesota)
- Jostle (U. Greenwich)

Geometric Partitioning: RCB

- Idea:
  - Divide work into two equal parts using a cutting plane orthogonal to a coordinate axis.
  - Recursively cut the resulting subdomains.
Geometric Partitioning: SFC

- Space-Filling Curve Partitioning
  - Gravitational simulations (Warren & Salmon, 1993)
  - Smoothed particle hydrodynamics (Pilkington & Baden, 1994)
  - Adaptive mesh refinement (Patra & Oden, 1995).

- SFC Partitioning Algorithm:
  - Run SFC through domain (mapping from $R^3$ to $R^1$).
  - Order objects according to position on curve.
  - Perform 1-D partitioning of curve.

Applications of Geometric Methods

- Adaptive Mesh Refinement
- Particle Simulations
- Parallel Volume Rendering
- Crash Simulations and Contact Detection
Geometric Methods
Advantages and Disadvantages

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

• Disadvantages:
  – No explicit control of communication costs.
  – Mediocre partition quality.
  – Need coordinate information.

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

Graph Partitioning

• Kernighan, Lin, Schweikert, Fiduccia, Mattheyses, 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

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)

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

- 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.

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
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.

Performance Results

• Experiments on Sandia’s Thunderbird cluster.
  – Dual 3.6 GHz Intel EM64T processors with 6 GB RAM.
  – Infiniband network.
• Compare RCB, SFC, graph (ParMETIS) 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

SLAC Linear Accelerator
2.9M x 2.9M
11.4M nonzeros

Cage15 DNA Electrophoresis
5.1M x 5.1M
99M nonzeros

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

Communication Volume:
Lower is Better

SLAC 6.0M LCLS
Number of parts = number of processors.

SLAC 2.9M Linear Accelerator

Xyce 680K circuit

Cage15 5.1M electrophoresis
Partitioning Time: Lower is better

SLAC 6.0M LCLS

SLAC 2.9M Linear Accelerator

Xyce 680K circuit

Cage15 5.1M electrophoresis

1024 parts. Varying number of processors.

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

Best Algorithms Paper Award at IPDPS07
“Hypergraph-based Dynamic Load Balancing for Adaptive Scientific Computations”
Catalyurek, Boman, Devine, Bozdag, Heaphy, & Riesen
Repartitioning Results: Lower is Better

SLAC 6.0M LCLS

Repartitioning
Time (secs)

Data Redistribution
Volume

Application
Communication
Volume

Xyce 680K circuit

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.
  - Suite of partitioners allows experimentation, comparisons.
Zoltan Interface Supports Many Applications

- Different applications, requirements, data structures.

- Multiphysics simulations
- Parallel electronics networks
- Particle methods
- Crash simulations
- Adaptive mesh refinement
- Linear solvers & preconditioners

Zoltan Interface

- Simple, easy-to-use interface.
  - Small number of callable Zoltan functions.
  - Callable from C, C++, Fortran90.

- Two ways to access Zoltan:
  - Through ITAPS mesh implementation.
  - Directly from application through native Zoltan interface.

- Coming in FY08:
  - Matrix-based interface through Trilinos/Isorropia.
Zoltan ITAPS Interface

- Interoperable Tools for Advanced Petascale Simulations
- ITAPS iMesh implementation provides information to Zoltan for partitioning.
  - Number of mesh entities, connectivity, coordinates.
- Given a loaded iMesh_Instance, the application ...
  - Constructs an ITAPSZoltan object;
  - Specifies number of parts, partitioning method, etc.; and
  - Invokes partitioning.
- Parts returned as tagged entity sets.
- Initial ITAPS-compliant implementation available.
  - https://svn.scorec.rpi.edu/wsvn/TSTT/Distributions/

Zoltan Native Interface Design

- 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.

- 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)
Zoltan Native Interface

• Application interface:
  – Zoltan queries the application for needed info.
    • IDs of objects, object weights, part assignments.
    • Geometric algorithms: dimensions, coordinates.
    • Connectivity-based algorithms: edge lists, edge weights.
  – Application provides simple functions to answer queries.

• Once query functions are implemented, application can access all Zoltan functionality.
  – Can switch between algorithms by setting parameters.

Zoltan Application Interface

APPLICATION

Initialize Zoltan (Zoltan_Initialize, Zoltan_Create)

Select LB Method (Zoltan_Set_Params)

Register query functions (Zoltan_Set_Fn)

ZOLTAN

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

(Re-)partition (Zoltan_LB_Partition)

Move data (Zoltan_Migrate)

COMPUTE

Clean up (Zoltan_Destroy)

Zoltan_Migrate:
• Call packing query functions for exports.
• Send exports.
• Receive imports.
• Call unpacking query functions for imports.
Aiming for Petascale

• Reducing communication costs for applications.
  – Reducing communication volume.
    • Two-dimensional sparse matrix partitioning (Catalyurek, Aykanat, Bisseling).
    • Partitioning non-zeros of matrix rather than rows/columns.
  – Reducing message latency.
    • Minimize maximum number of neighboring parts (with Kumfert, LLNL).
    • Balancing both computation and communication (Pinar & Hendrickson); balance criterion is complex function of the partition instead of simple sum of object weights.
  – Reducing communication overhead.
    • Map parts onto processors to take advantage of network topology.
    • Minimize distance messages travel in network.

Aiming for Petascale

• Hierarchical partitioning in Zoltan v3.
  – Partition for multicore/manycore architectures.
    • Partition hierarchically with respect to chips and then cores.
    • Similar to strategies for clusters of SMPs (Teresco, Faik).
    • Treat core-level parts as separate threads or MPI processes.
  – Support 100Ks processors.
    • Reduce collective communication operations during partitioning.
    • Allow more localized partitioning on subsets of processors.
Aiming for Petascale

• Improving scalability of partitioning algorithms.
  – Hybrid partitioners (particularly for mesh-based apps.)
    • Use inexpensive geometric methods for initial partitioning;
      refine with high quality hypergraph/graph-based algorithms.
    • Use geometric information to accelerate multilevel
      hypergraph/graph-based partitioners.
  – Refactored partitioners for bigger data sets and processor arrays.

Aiming for Petascale

• Developing specialized partitioning strategies.
  – E.g., for particle-in-cell applications, multiscale.
• Data ordering within a processor.
  – Better memory performance.
  – Multicore support.
• Testing and performance evaluation.
  – Examine effectiveness of partitions in applications.

• Wanted: Collaborations with application developers!
For More Information…

  – Download Zoltan v3 (open-source software).

• ITAPS Interface to Zoltan:
  [https://svn.scorec.rpi.edu/wsvn/TSTT/Distributions/](https://svn.scorec.rpi.edu/wsvn/TSTT/Distributions/)

Thanks

SciDAC CSCAPES Institute (A. Pothen, Old Dominion U., PI)
SciDAC ITAPS Center (L. Diachin, LLNL, PI)
NNSA ASC Program

- S. Attaway (SNL)
- C. Aykanat (Bilkent U.)
- A. Bauer (RPI)
- R. Bisseling (Utrecht U.)
- D. Bozdag (Ohio St. U.)
- T. Davis (U. Florida)
- J. Faik (RPI)
- J. Flaherty (RPI)
- R. Heaphy (SNL)
- B. Hendrickson (SNL)
- M. Heroux (SNL)
- K. Ko (SLAC)
- G. Kumfert (LLNL)
- L.-Q. Lee (SLAC)
- V. Leung (SNL)
- G. Lonsdale (NEC)
- X. Luo (RPI)
- L. Musson (SNL)
- S. Plimpton (SNL)
- J. Shadid (SNL)
- M. Shephard (RPI)
- C. Silvio (SNL)
- J. Teresco (Mount Holyoke)
- C. Vaughan (SNL)
- M. Wolf (U. Illinois)