New Challenges In Dynamic Load Balancing
Karen D. Devine, et al.

Presentation by
Nam Ma & J. Anthony Toghia
What is load balancing?

- Assignment of work to processors
- Goal: maximize parallel performance through minimal CPU idle time and interprocessor communication overhead
• Static
  • Applications with constant workloads (i.e. predictable)
  • Pre-processor to the computation
• Dynamic
  • Unpredictable workloads (e.g. adaptive finite element methods)
  • On-the-fly adjustment of application decomposition
Current challenges

- Most load balancers are custom written for an application
  - Lack of code reuse
  - Inability to compare different load balancing algorithms
  - Limited range of applications (symmetric, sparsely connected relationships)
  - Architecture dependence
• Library of expert implementations of related algorithms
  • General purpose application to a wide variety of algorithms
  • Code reuse / portability
  • More thorough testing
  • Comparison of various load balancers by changing a run-time parameter
• Data structure neutral design (data structures represented as generic objects with weights representing their computational costs)
Zoltan: additional tools

- Data migration tools to move data between old and new decompositions
- Callbacks to user defined constructors for data structures to support data-structure neutral migration
- Unstructured communication package (custom, complex migrations)
- All tools may be used independently (e.g. user can use Zoltan to compute decomposition but perform data migration manually or vice versa)
Tutorial: The Zoltan Toolkit

Karen Devine and Cedric Chevalier
Sandia National Laboratories, NM

Umit Çatalyürek
Ohio State University

CSCAPES Workshop, June 2008
The Zoltan Toolkit

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

Dynamic Load Balancing

Data Migration

Graph Coloring

Matrix Ordering

Unstructured Communication

Distributed Data Directories

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Zoltan Application Interface

APPLICATION

Initialize Zoltan
(Zoltan_Initialize, Zoltan_Create)

Select Method and Parameters
(Zoltan_Set_Params)

Register query functions
(Zoltan_Set_Fn)

ZOLTAN

COMPUTE

Re-partition
(Zoltan_LB_Partition)

Move data
(Zoltan_Migrate)

Clean up
(Zoltan_Destroy)

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

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

**Geometric (coordinate-based) methods**
- Recursive Coordinate Bisection (Berger, Bokhari)
- Recursive Inertial Bisection (Taylor, Nour-Omid)

**Combinatorial (topology-based) methods**
- Space Filling Curve Partitioning (Warren & Salmon, et al.)
- Refinement-tree Partitioning (Mitchell)
- Hypergraph Partitioning
- Hypergraph Repartitioning
- PaToH (Catalyurek & Aykanat)
- Zoltan Graph Partitioning
- ParMETIS (U. Minnesota)
- Jostle (U. Greenwich)
Geometric Partitioning

- Suitable for a class of applications, such as crash and particle simulations.
  - Geometric-coordinate based decomposition is a more natural and straightforward approach.
  - Graph partitioning is difficult or impossible.
  - Frequent changes in proximity require fast and dynamic repartitioning strategies.
- Based on two main algorithms
  - Recursive coordinate bisection
  - Space filling curve
Recursive Coordinate Bisection (RCB)

- Divide the domain into two equal subdomains using a cutting plane orthogonal to a coordinate axis.
- Recursively cut the resulting subdomains.

- A variation of RCB: Recursive Inertial Bisection, which computes cuts orthogonal to principle inertial axes of the geometry
Space Filling Curve (SFC)

- SFC: Mapping between $\mathbb{R}^n$ to $\mathbb{R}^1$ that completely fills a domain
- SFC Partitioning:
  - Run SFC through domain
  - Order objects according to position on curve
  - Perform 1-D partition of curve
Geometric Partitioning - Advantages and Disadvantages

• **Advantages**
  • Simple, fast, inexpensive
  • Effective when geometric locality is important
  • No connectivity needed
  • Implicitly incremental for repartitioning

• **Disadvantages**
  • No explicit control of communication costs
  • Need coordinate information
Repartitioning

- Implicitly incremental: small changes in workloads produce only small change in the decomposition
  → Reduce the cost of moving application data
An Application: Contact Detection in Crash Simulation

- Identify which partition’s subdomains intersect a given point (point assignment) or region (box assignment)
  - Point assignment: given a point, it returns the partition owning the region of space containing that point
  - Box assignment: given an axis-aligned region of space, it returns a list of partitions whose assigned regions overlap the specified box
Performance

- **Experimental Setup:**
  - Initialized 96 partitions on a 16-processor cluster
  - Performed 10,000 box-assignments

- **Results comparing RCB and SFC**

<table>
<thead>
<tr>
<th>Partitioner</th>
<th># of Intersecting Parts for 10 000 box-assignments</th>
<th>Partitioning Time</th>
<th>Time for 10,000 box-assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCB</td>
<td>10,931</td>
<td>0.71 secs</td>
<td>0.027 secs</td>
</tr>
<tr>
<td>HSFC</td>
<td>10,983</td>
<td>0.59 secs</td>
<td>0.176 secs</td>
</tr>
</tbody>
</table>
Graph Partitioning

Best for mesh-based partial differential equation (PDE) simulations

- Represent problem as a weighted graph.
  - Vertices = objects to be partitioned.
  - Edges = dependencies between two objects.
  - Weights = work load or amount of dependency.

- Partition graph so that …
  - Parts have equal vertex weight.
  - Weight of edges cut by part boundaries is small.
Graph Partitioning: Advantages 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

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

- 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 partitioning

- Coarsening phase (Clustering)
  - Hierarchical (simultaneous clustering)
    - 1. Pick an unreached vertex (random)
    - 2. Connect it to another vertex based on selection criteria (e.g. shortest edge)
  - Agglomerative (build 1 cluster at a time)
    - $N_{u,Cuv}$ Connectivity value of vertex $N = \#$ of edges connected to $N$
    - $W_{u,Cv}$, Weight = number of vertices in any cluster
    - 1. Choose cluster or singleton with highest $N_{u,Cuv} / W_{u,Cv}$
    - 2. Choose edge based on selection criteria
- Partitioning phases (Bisecting hypergraph)
- Uncoarsening phase
  - Project coarsened, bisected graph back to previous level
  - Refine bisection by running boundary force method (BFM)
Hypergraph Example
Kernighan/Lin Algorithm

- $E(a) = \text{external cost of } a \text{ in } A = \sum \{W(a,b) \text{ for } b \text{ in } B\}$
- $I(a) = \text{internal cost of } a \text{ in } A = \sum \{W(a,a') \text{ for other } a' \text{ in } A\}$
- $D(a) = \text{total cost of } a \text{ in } A = E(a) - I(a)$
- Consider swapping $a$ in $A$ and $b$ in $B$
  - New Partition Cost = Old Partition Cost – $D(a) – D(b) + 2*W(a,b)$
- Compute $D(n)$ for all nodes

Repeat
- Unmark all nodes
- While there are unmarked pairs
  - Find an unmarked pair $(a,b)$
  - Mark $a$ and $b$ (but do not swap)
  - Update $D(n)$ for all unmarked nodes, as if $a$ and $b$ had been swapped
  - Pick maximizing gain
  - If Gain > 0 then swap

Until gain $\leq 0$
- Worst case $O(N^2 \log N)$
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”);`
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.
Table 3
Comparison of graph and hypergraph partitioning for HexFEM matrix (Example 3).

<table>
<thead>
<tr>
<th>Partitioning Method</th>
<th>Imbalance (Max / Avg Work)</th>
<th># of Neighbor Partitions per Partition</th>
<th>Communication Volume over all Partitions</th>
<th>Reduction of Total Communication Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
</tr>
<tr>
<td>Graph method (METIS PartKWay)</td>
<td>1.03</td>
<td>4</td>
<td>3.6</td>
<td>1659</td>
</tr>
<tr>
<td>Best Zoltan hypergraph method (RRM)</td>
<td>1.013</td>
<td>4</td>
<td>3.6</td>
<td>1164</td>
</tr>
<tr>
<td>Worst Zoltan hypergraph method (RHP)</td>
<td>1.019</td>
<td>4</td>
<td>2.8</td>
<td>2209</td>
</tr>
</tbody>
</table>

HexFEM Matrix:
- Hexahedral 3D structured-mesh finite element method.
- 32,768 rows
- 830,584 non-zeros
- Five partitions
Hypergraph results – asymmetric, sparse graph

Table 4
Comparison of graph and hypergraph partitioning for PolyDFT matrix (Example 3).

<table>
<thead>
<tr>
<th>Partitioning Method</th>
<th>Imbalance (Max / Avg Work)</th>
<th># of Neighbor Partitions per Partition</th>
<th>Communication Volume over all Partitions</th>
<th>Reduction of Total Communication Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
</tr>
<tr>
<td>Graph method (METIS PartKWay)</td>
<td>1.03</td>
<td>7</td>
<td>6</td>
<td>7382</td>
</tr>
<tr>
<td>Best Zoltan hypergraph method (MXG)</td>
<td>1.018</td>
<td>5</td>
<td>4</td>
<td>3493</td>
</tr>
<tr>
<td>Worst Zoltan hypergraph method (GRP)</td>
<td>1.03</td>
<td>6</td>
<td>5.25</td>
<td>5193</td>
</tr>
</tbody>
</table>
Resource-aware Load Balancing

- **Heterogeneous Architectures**
  - Clusters may have different types of processors with various capacity
- **Assign “capacity” weight to processors**
- **Balance with respect to processors capacity**
- **Hierarchical partitioning: Allows different partitioners at different architecture levels**
Dynamic Resource Utilization Model (DRUM)

- DRUM provides applications aggregated information about the computation and communication capabilities of an execution environment.
- The tree constructed by DRUM represents a heterogeneous network.
  - Leaves represent individual computation nodes (i.e. single processors (SP) or SMPs)
  - Non-leaf nodes represent routers or switches, having an aggregate power.
Tree represents a heterogeneous network by DRUM
Power Representation

- Power of a node is computed as weighted sum of a processing power $p_n$ and a communication power $c_n$

$$power_n = w_n^{\text{comm}} c_n + w_n^{\text{cpu}} p_n,$$

$$w_n^{\text{comm}} + w_n^{\text{cpu}} = 1.$$

- For each node $n$ in $L_i$ (the set of nodes at level $i$) in the network, the final power is computed by

$$power_n = pp_n \left( w_n^{\text{comm}} \frac{c_n}{\sum_{j=1}^{\mid L_i \mid} c_j} + w_n^{\text{cpu}} \frac{p_n}{\sum_{j=1}^{\mid L_i \mid} p_j} \right),$$

where $pp_n$ is the power of the parent of node $n$. 
For More Information...

• Zoltan Home Page
  – User’s and Developer’s Guides
  – Download Zoltan software under GNU LGPL.

• Email:
  – {kddevin,ccheval,egboman}@sandia.gov
  – umit@bmi.osu.edu
Questions and Comments