Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations

Roy D. Williams, 1990

Presented by Chris Eldred

Outline

- Summary
- Load Balancing
  - Types
- Quasi-Dynamic Load Balancing
- Optimization Problem
- Cost Function
- Algorithms
  - Simulated Annealing
  - ORB
  - ERB
- Finite Element Solver
- Results
- Conclusions
- Exascale/The Future

Summary

- Dynamic load balancing is important for good performance
  - Especially as the number of cores increases
- Eigenvalue Recursive Bisection (ERB) seems to be a good compromise between cost and efficiency
- Methods outlined here are applicable to a wide range of problems

What makes efficient DM code

- Processors have equal work
- Amount of communication is minimized
- Communication occurs in large messages rather than many small messages

- The problem of how to divide up a given computational problem between a set of processors is known as **load balancing**

Types of Loading Balancing

- By Inspection: Problem is solved analytically before the code is written
- Static: Problem is solved by a sequential machine prior to doing parallel runs
- Quasi-Dynamic: Problem is solved during program execution but changes are discrete and infrequent
- Dynamic: Problem is solved during program execution and changes are continuous and constant

Quasi-Dynamic Load Balancing

- Problem is solved during program execution but changes are discrete and infrequent
- For quasi-dynamic and dynamic loading balancing, a trade-off between overhead incurred by balancing and improved performance from balanced execution must be reached
- Only parallel methods for solving load balancing problem investigated
  - Scalable code

Optimization Problem

- Load balancing problem formulated as an optimization problem involving the minimization of a cost function
  - Cost function is designed to be independent of the details of the code
- Two parts to cost
  - Computational load per processor (imbalance)
  - Communication cost between processors
- Stated as a graph coloring problem

Optimization Problem part 2

- Undirected graph of N nodes (finite elements) colored with P colors (processors) to minimize H (cost function; depends on coloring)
- Hcalc is assumed to be minimized when Nq is the same for all q (equal distribution of nodes)
  - Does this always hold?- not really as number of processors increases

$$H = H_{calc} + \mu H_{comm}$$

Cost Function

$$H = \frac{P^2}{N^2} \sum_q N_q^2 + \mu \left(\frac{P}{N}\right)^{\frac{d-1}{d}} \sum_{e \leftrightarrow f} 1 - \delta_{p(e),\, p(f)}$$

- Assumes communication cost between all processor pairs is the same
- Ignores latency of message passing
- Ignores parallelism of communication
- Equally balanced mesh leads to a lower limit on the amount of boundary (and hence communication)

Simplified Cost Function Model

- Assume that N >> P >> 1
- Filling fraction: proportion of "proper share" of elements (N/P)

$$H = 1 + (1-\alpha)^2 + 2\mu\,(1-\delta_{\alpha,0}) \qquad \text{dimension} = 1$$

$$H = 1 + (1-\alpha)^2 + \mu\sqrt{6\alpha} \qquad \text{dimension} = 2$$

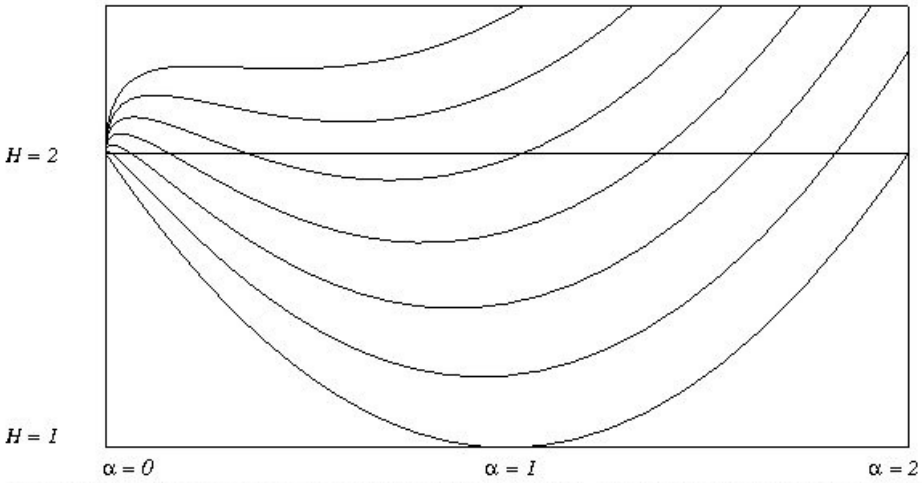


Figure 1: Cost function variation with filling fraction in two dimensions, for $\mu = 0$ (bottom curve) to 0.6 (upper curve) in steps of 0.1.

Load Balancing Algorithms

- Three different load balancing algorithms were investigated:
  - Simulated Annealing (SA)
  - Orthogonal Recursive Bisection (ORB)
  - Eigenvector Recursive Bisection (ERB)
- These methods come from:
  - Statistical physics (SA)
  - Eigenvector analysis (ERB)
  - Geometric consideration (ORB)

Simulated Annealing

- Based on stochastic physics: simulated the slow cooling of a material
- Cost function H, temperate T and rules for changing state
- Changes are iteratively accepted or rejected by Metropolis criterion
  - If dH <0 accept, if not accept with probability exp(-dH/T)
- Changes must satisfy reachability

Simulated Annealing part 2

- Heuristics
  - Use knowledge of system to decide on how to change
  - Ex. Globules
  - Can be good or bad, influence reachability
- Collisional SA
  - Run SA in parallel
  - dH is not linear: parallel collisions
  - Can solve with rollback: expensive
- Potts Model
  - Divide changes into classes that do not collide

Simulated Annealing part 3

- Clustering: change connected sets of node at the same time
- Authors used simple collisional SA and ignored parallel collisions; also used clustering

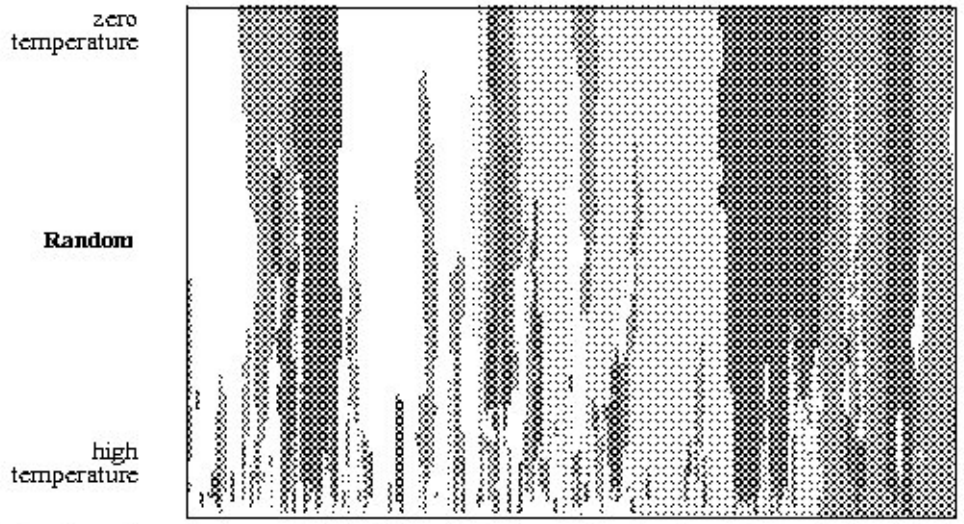


Figure 2: Simulated annealing of a ring graph of size 200, with the four graph colors shown by gray shades. The time history of the annealing runs vertically, with the maximum temperature and the starting configuration at the bottom; zero temperature and the final optimum at the top. The basic move is to change the color of a graph node to a random color.

Recursive Bisection

- Idea: Coloring with 2 colors is easier than many colors
- Split graph into 2 parts and minimize communication between then
- Continue recursively until the whole graph is colored
- Runs into issues when communication between pairs of processors is no longer equal
- Split is done by using a separator field
  - Use median of continuous field
  - Should be carefully chosen so that communication is minimzed

Orthogonal Recursive Bisection (ORB)

- Separator field based on position of elements in mesh (centers of mass)
- Can alternate directions of split (vertical/horizontal) or use more complicated measures (moment of inertia tensor)
- Ignores connectivity between elements- idea is that graph we are bisecting is related to geometry of the mesh elements

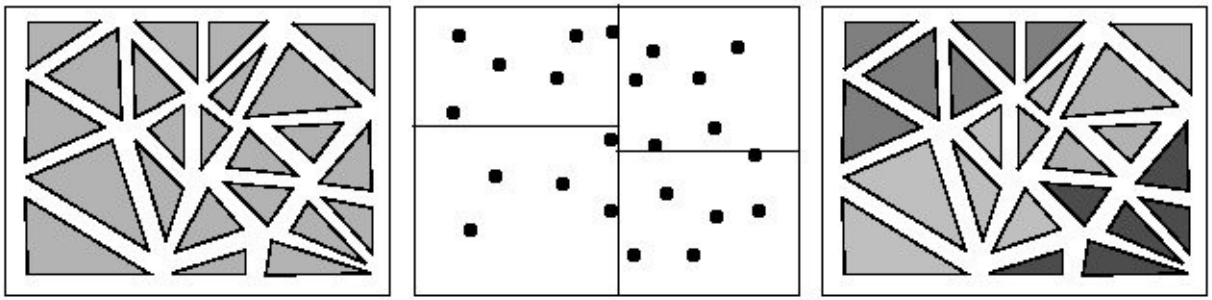
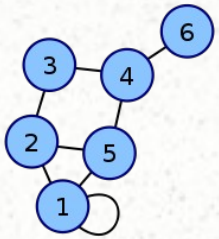


Figure 8: Load balancing by ORB for four processors. The elements (left) are reduced to points at their centers of mass (middle), then split into two vertically, then each half split into two horizontally. The result (right) shows the assignment of elements to processors.

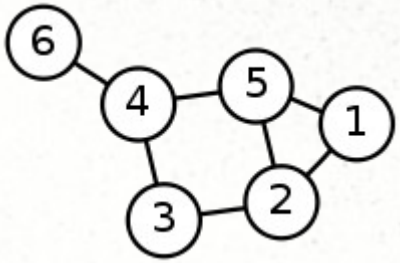
Eigenvector Recursive Bisection (ERB)

- Adjacency Matrix A



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

- Laplacian Matrix Q = D- A
  - D is the degree matrix of the graph



$$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

$$\ell_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

ERB part 2

- Assume graph is connected (makes sense)
- Choose eigenvector of Q associated with smallest positive eigenvalue (call it y)
- Use components of y as a separator field (find median value of ye and split graph based on that)
- Theorem by Fiedler says this procedure will produce connected subgraphs
- Use Lanczos method to calculate eigenvector
  - They used 30 vectors for 4000 nodes
- Could also use method based on 2nd highest eigenvector of adjacency matrix

Finite Element Solver

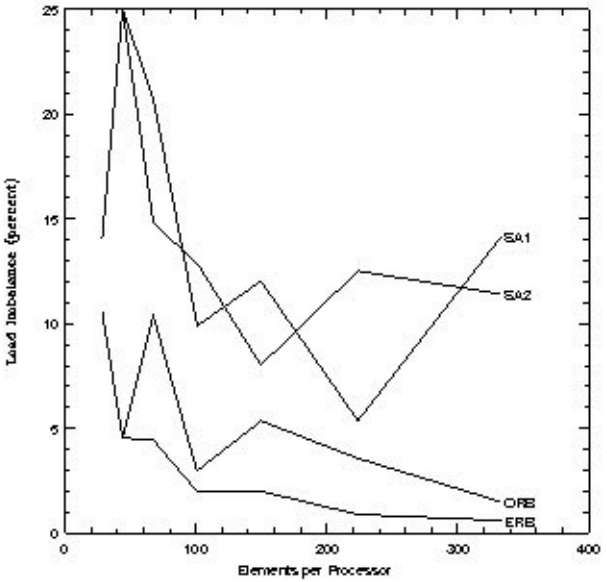
- Solver was implemented using DIME (Distributed Irregular Mesh Environment)
- Solution adaptive Laplacian solver (refinement occurs based on magnitude of gradient)
  - Jacobi iteration
  - Started with 280 elements all on one processor
  - Seven steps of: Solve, Refine, Balance
  - Final mesh had 5772 elements
  - 40% of elements refined at each stage

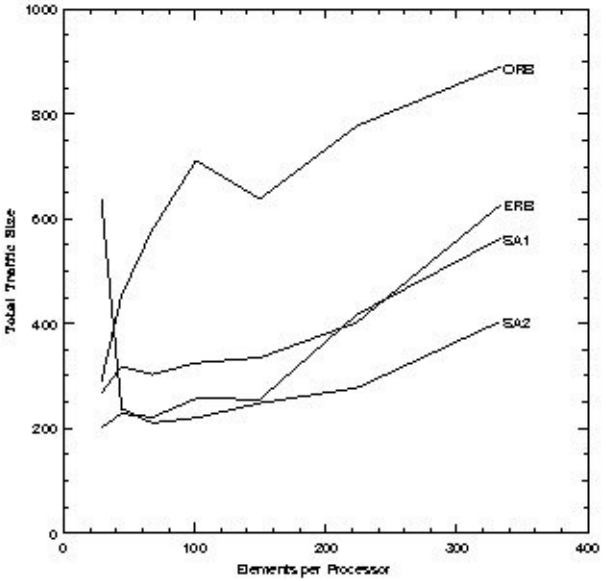
Testing

- Load balancing had two stages:
  - Decide where each element goes
  - Migrate all elements at once
  - Care had to be taken to avoid deadlock
- Two SA Runs
  - SA1 (low T, 500 stages)
  - SA2 (high T, 5000 stages)
- ORB Run
- ERB Run

Results

- Machine-Independent Results:
  - Load Imbalance
  - Total Traffic Size
  - Number of Messages





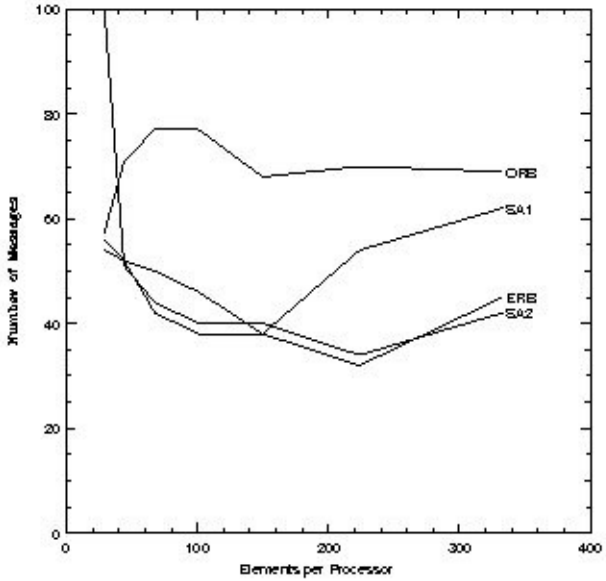


Figure 12: Machine-independent measures of load balancing performance. Left, percentage load imbalance; lower left, total amount of communication; below, total number of messages.

Results part 2

- Machine-Dependent Results:
  - Flops per iteration
  - Communication time per iteration

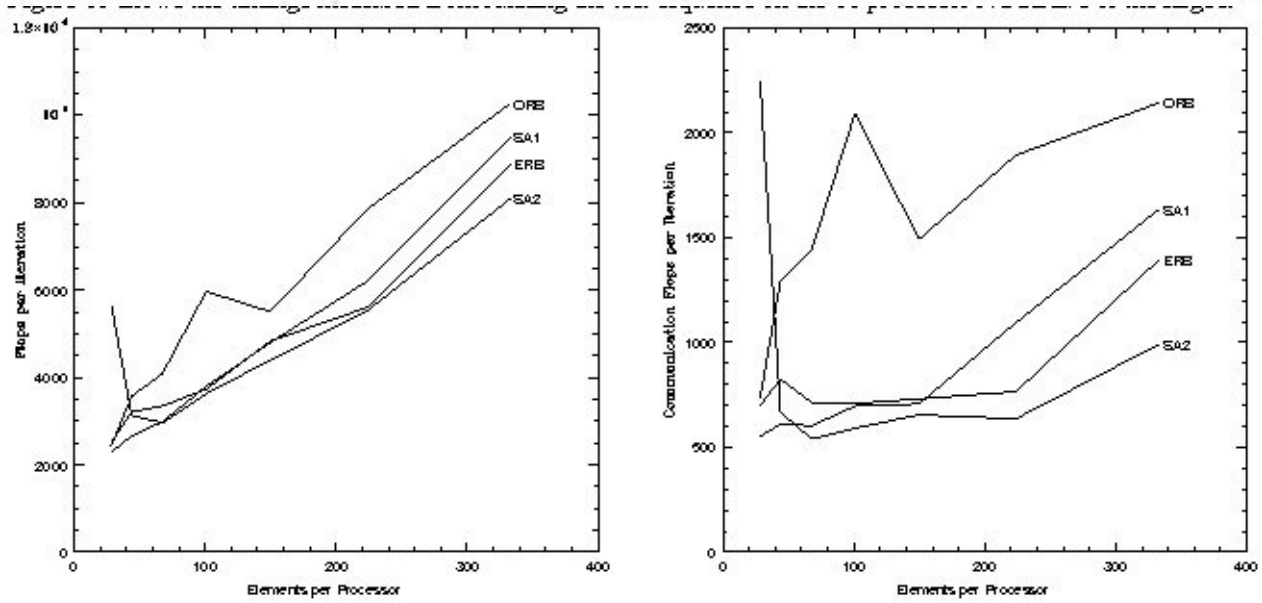


Figure 13: Machine-dependent measures of load balancing performance. Left, running time per Jacobi iteration in units of the time for a floating-point operation (flop); right, time spent doing local communication in flops.

Results part 3

- Measurements for Dynamic Load Balancing:
  - Elements migrated
  - Total time for load balancing

Method	Time(minutes)
ORB	5
ERB	11
SA1	25
SA2	230

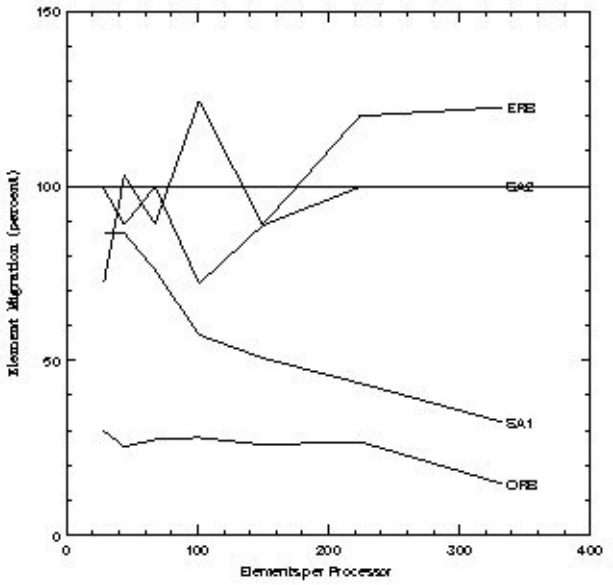


Figure 14: Percentage of elements migrated during each load balancing stage. The percentage may be greater than 100 because the recursive bisection methods may cause the same element to be migrated several times.

Conclusions

- Evaluate load balancing algorithms by:
  - Quality of solution (time per iteration)
  - Time to do load balancing
  - Portability and compactness
- ORB: Cheap, portable, compact, relatively poor performance, ignores connectivity of element graph
- SA: Expensive, portable, have to set many parameters, excellent performance
- ERB: Compromise between ORB and SA, based on eigenvector calculations that are already well-implemented in existing parallel libraries

Exascale/The Future

- It is like that future machines will have O(100K-1M) cores
- Static or inspection-based load balancing will not work for that many cores- need to be able to do dynamic or quasi-dynamic load balancing
- Many types of problems and work decompositions can be expressed as graphs
  - This work is applicable well beyond finite element solvers and adaptive meshes