Load Balancing Part 2: Static Load Balancing

Kathy Yelick
yelick@cs.berkeley.edu
www.cs.berkeley.edu/~yelick/cs194f07
Load Balancing Overview

Load balancing differs with properties of the tasks (chunks of work):

• **Tasks costs**
  • Do all tasks have equal costs?
  • If not, when are the costs known?
    • Before starting, when task created, or only when task ends

• **Task dependencies**
  • Can all tasks be run in any order (including parallel)?
  • If not, when are the dependencies known?
    • Before starting, when task created, or only when task ends

• **Locality**
  • Is it important for some tasks to be scheduled on the same processor (or nearby) to reduce communication cost?
  • When is the information about communication known?
Task Cost Spectrum

Schedule a set of tasks under one of the following assumptions:

**Easy:** The tasks all have equal (unit) cost.  

```
      ● ● ● ● ● 
 n items
```

```
      ● ● ● ● ● 
 p bins
```

**Harder:** The tasks have different, but known, times.

```
      ● ● ● ● ● 
 n items
```

```
      ● ● ● ● ● 
 p bins
```

**Hardest:** The task costs unknown until after execution.

```
      ● ● ● ● ● 
 n items
```

```
      ● ● ● ● ● 
 p bins
```

- branch-free loops
- sparse matrix-vector multiply
- GCM, circuits, search
Task Dependency Spectrum

Schedule a graph of tasks under one of the following assumptions:

**Easy:** The tasks can execute in any order.

**Harder:** The tasks have a predictable structure.

- wave-front
- out-tree
- in-tree
- general dag

balanced or unbalanced

**Hardest:** The structure changes dynamically (slowly or quickly)

search, sparse LU

dependence
free loops

matrix
computations
(dense, and some sparse, Cholesky)
Task Locality Spectrum (Communication)

Schedule a set of tasks under one of the following assumptions:

**Easy:** The tasks, once created, do not communicate.

**Harder:** The tasks communicate in a predictable pattern.

- Regular
- Irregular

**Hardest:** The communication pattern is unpredictable.

- Embarrassingly parallel
- PDE solver
- Discrete event simulation
Spectrum of Solutions

A key question is when certain information about the load balancing problem is known. Many combinations of answer leads to a spectrum of solutions:

- **Static scheduling.** All information is available to scheduling algorithm, which runs before any real computation starts.
  - *Off-line* algorithms make decisions before execution time

- **Semi-static scheduling.** Information may be known at program startup, or the beginning of each timestep, or at other well-defined points.
  - Offline algorithms may be used, between major steps.

- **Dynamic scheduling.** Information is not known until mid-execution.
  - *On-line* algorithms make decisions mid-execution
Solutions for Specific Problems

• For the solutions we have so far, locality is not considered, i.e., the techniques do not optimized for it
  • Loops with independent iterations
  • Divide-and-conquer problems with little/no communication (“bound” may be communicated in branch-and-bound search)
  • Computationally intensive tasks like matrix multiply

<table>
<thead>
<tr>
<th></th>
<th>Equal cost tasks</th>
<th>Unequal, but known cost</th>
<th>Unpredictable cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered “bag” of tasks</td>
<td>Trivial</td>
<td>Bin packing</td>
<td>Self Scheduling</td>
</tr>
<tr>
<td>Task tree (unknown shape)</td>
<td>Work stealing</td>
<td>Work stealing</td>
<td>Work stealing</td>
</tr>
<tr>
<td>Task graph (DAG)</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Solutions for Specific Problems

• If locality **is** important then we may need other solutions
• Two cases:
  • Task bag (independent tasks) that need to communicate → run on same processor (serialize) or “nearby”
  • Task graph (dependencies) → if two dependent tasks need to share data, try to schedule on same processor

<table>
<thead>
<tr>
<th></th>
<th>Equal cost tasks</th>
<th>Unequal, but known cost</th>
<th>Unpredictable cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered “bag” of tasks</td>
<td>Minimize surface to volume ratio; Array decomposition or graph partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task tree</td>
<td>Treat as general DAG if locality is really critical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task graph (DAG)</td>
<td>General scheduling problem</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Regular Meshes (e.g., Game of Life)

- Independent tasks (“bag” not DAG or tree)
- Load balancing $\rightarrow$ equal size partitions
- Locality $\rightarrow$ minimize perimeter using low aspect ratio partitions
  - Will hopefully reduce cache misses, false sharing

\[
\begin{align*}
\text{n*(p-1)} \\
\text{edge crossings}
\end{align*}
\]

\[
\begin{align*}
\text{2*n*(p^{1/2} - 1)} \\
\text{edge crossings}
\end{align*}
\]
Irregular Communication Patterns

- A *task interaction* graph shows which tasks communicate/share data with others
  - May be weighted by volume of data shared
  - If the data is constant, it may be replicated and doesn’t “count”

- The task interaction graph for the game-of-life is a regular 2D mesh

- For animations, simulations of complex structures, etc., unstructured meshes are used instead
Definition of Graph Partitioning

- Given a graph $G = (N, E, W_N, W_E)$
  - $N =$ nodes (or vertices),
  - $W_N =$ node weights
  - $E =$ edges
  - $W_E =$ edge weights

- Ex: $N =$ {tasks}, $W_N =$ {task costs}, edge $(j,k)$ in $E$ means task $j$ sends $W_E(j,k)$ words to task $k$

- Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_P$ such that
  - The sum of the node weights in each $N_j$ is “about the same”
  - The sum of all edge weights of edges connecting all different pairs $N_j$ and $N_k$ is minimized

- Ex: balance the work load, while minimizing communication

- Special case of $N = N_1 \cup N_2$: Graph Bisection
Definition of Graph Partitioning

- Given a graph $G = (N, E, W_N, W_E)$
  - $N =$ nodes (or vertices),
  - $W_N =$ node weights
  - $E =$ edges
  - $W_E =$ edge weights

- Ex: $N =$ \{tasks\}, $W_N =$ \{task costs\}, edge $(j,k)$ in $E$ means task $j$ sends $W_E(j,k)$ words to task $k$

- Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_p$ such that
  - The sum of the node weights in each $N_j$ is “about the same”
  - The sum of all edge weights of edges connecting all different pairs $N_j$ and $N_k$ is minimized (shown in black)

- Ex: balance the work load, while minimizing communication

- Special case of $N = N_1 \cup N_2$: Graph Bisection
Applications

• Telephone network design
  • Original application, algorithm due to Kernighan

• Load Balancing while Minimizing Communication

• Sparse Matrix times Vector Multiplication
  • Solving PDEs
  • $N = \{1,\ldots,n\}$, $(j,k)$ in $E$ if $A(j,k)$ nonzero,
  • $W_N(j) = \#\text{nonzeros in row } j$, $W_E(j,k) = 1$

• VLSI Layout
  • $N = \{\text{units on chip}\}$, $E = \{\text{wires}\}$, $W_E(j,k) = \text{wire length}$

• Sparse Gaussian Elimination
  • Used to reorder rows and columns to increase parallelism,
    and to decrease “fill-in”

• Data mining and clustering

• Physical Mapping of DNA
Sparse Matrix Vector Multiplication $y = y + A^*x$

Partitioning a Sparse Symmetric Matrix

... declare $A_{local}$, $A_{remote}$(1:num_procs), $x_{local}$, $x_{remote}$, $y_{local}$
$y_{local} = y_{local} + A_{local} \times x_{local}$
for all procs P that need part of $x_{local}$
  send(needed part of $x_{local}$, P)
for all procs P owning needed part of $x_{remote}$
  receive($x_{remote}$, P)
$y_{local} = y_{local} + A_{remote}(P)\times x_{remote}$
Cost of Graph Partitioning

• Many possible partitionings to search
• Just to divide in 2 parts there are:
  \[ \binom{n}{n/2} \approx \sqrt{\frac{2}{(n\pi)}} \cdot 2^n \] possibilities

• Choosing optimal partitioning is NP-complete
  • (NP-complete = we can prove it is as hard as other well-known hard problems in a class Nondeterministic Polynomial time)
  • Only known exact algorithms have cost = exponential(n)

• We need good heuristics
Overview of heuristics
First Heuristic: Repeated Graph Bisection

• To partition $N$ into $2^k$ parts
  • bisect graph recursively $k$ times
• Henceforth discuss mostly graph bisection
Edge Separators vs. Vertex Separators

- **Edge Separator**: $E_s$ (subset of $E$) separates $G$ if removing $E_s$ from $E$ leaves two ~equal-sized, disconnected components of $N$: $N_1$ and $N_2$

- **Vertex Separator**: $N_s$ (subset of $N$) separates $G$ if removing $N_s$ and all incident edges leaves two ~equal-sized, disconnected components of $N$: $N_1$ and $N_2$

- Making an $N_s$ from an $E_s$: pick one endpoint of each edge in $E_s$
  - $|N_s| \leq |E_s|$

- Making an $E_s$ from an $N_s$: pick all edges incident on $N_s$
  - $|E_s| \leq d \times |N_s|$ where $d$ is the maximum degree of the graph

- We will find Edge or Vertex Separators, as convenient
Overview of Bisection Heuristics

- Partitioning with Nodal Coordinates
  - Each node has x,y,z coordinates → partition space

- Partitioning without Nodal Coordinates
  - E.g., Sparse matrix of Web documents
    - $A(j,k) = \# \text{ times keyword } j \text{ appears in URL } k$
  - Multilevel acceleration (advanced topic)
    - Approximate problem by “coarse graph,” do so recursively
Partitioning with Nodal Coordinates
i.e., nodes at point in (x,y) or (x,y,z) space
Nodal Coordinates: How Well Can We Do?

• A planar graph can be drawn in plane without edge crossings
• Ex: m x m grid of m² nodes: ∃ vertex separator Nₛ with |Nₛ| = m = sqrt(|N|) (see last slide for m=5)
• Theorem (Tarjan, Lipton, 1979): If G is planar, ∃ Nₛ such that
  • N = N₁ U Nₛ U N₂ is a partition,
  • |N₁| <= 2/3 |N| and |N₂| <= 2/3 |N|
  • |Nₛ| <= sqrt(8 * |N|)
• Theorem motivates intuition of following algorithms
Nodal Coordinates: Inertial Partitioning

- For a graph in 2D, choose a line with half the nodes on one side and half on the other
  - In 3D, choose a plane, but consider 2D for simplicity
- Choose a line $L$, and then choose a line $L^\perp$ perpendicular to it, with half the nodes on either side

1. Choose a line $L$ through the points $L$ given by $a^*(x-x_{\text{bar}}) + b^*(y-y_{\text{bar}}) = 0$,
   with $a^2 + b^2 = 1$; $(a,b)$ is a unit vector $\perp$ to $L$
2. Project each point to the line
   For each $n_j = (x_j,y_j)$, compute coordinate
   $$S_j = -b^*(x_j-x_{\text{bar}}) + a^*(y_j-y_{\text{bar}})$$ along $L$
3. Compute the median
   Let $S_{\text{bar}} = \text{median}(S_1,\ldots,S_n)$
4. Use median to partition the nodes
   Let nodes with $S_j < S_{\text{bar}}$ be in $N_1$, rest in $N_2$
Inertial Partitioning: Choosing L

- Clearly prefer L on left below

- Mathematically, choose L to be a total least squares fit of the nodes
  - Minimize sum of squares of distances to L (green lines on last slide)
  - Equivalent to choosing L as axis of rotation that minimizes the moment of inertia of nodes (unit weights) - source of name
\[
\Sigma_j \text{ (length of } j\text{-th green line)}^2
= \Sigma_j \left[ (x_j - \text{xbar})^2 + (y_j - \text{ybar})^2 - (-b*(x_j - \text{xbar}) + a*(y_j - \text{ybar}))^2 \right]
\]
\[
\text{... Pythagorean Theorem}
\]
\[
= a^2 \cdot \Sigma_j (x_j - \text{xbar})^2 + 2*a*b* \Sigma_j (x_j - \text{xbar})(x_j - \text{ybar}) + b^2 \Sigma_j (y_j - \text{ybar})^2
\]
\[
= a^2 \cdot X1 + 2*a*b* X2 + b^2 \cdot X3
\]
\[
= [a \ b] \cdot \begin{bmatrix} X1 & X2 \\ X2 & X3 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}
\]

Minimized by choosing
(xbar, ybar) = (\Sigma_j x_j, \Sigma_j y_j) / n = center of mass
(a,b) = eigenvector of smallest eigenvalue of \[
\begin{bmatrix} X1 & X2 \\ X2 & X3 \end{bmatrix}
\]

(a,b) is unit vector perpendicular to L
Nodal Coordinates: Random Spheres

• Generalize nearest neighbor idea of a planar graph to higher dimensions
  • Any graph can fit in 3D with edge crossings
  • Capture intuition of planar graphs of being connected to “nearest neighbors” but in higher than 2 dimensions

• For intuition, consider graph defined by a regular 3D mesh
  • An n by n by n mesh of $|N| = n^3$ nodes
    • Edges to 6 nearest neighbors
    • Partition by taking plane parallel to 2 axes
    • Cuts $n^2 = |N|^{2/3} = O(|E|^{2/3})$ edges

• For the general graphs
  • Need a notion of “well-shaped” like mesh
Random Spheres: Well Shaped Graphs

• Approach due to Miller, Teng, Thurston, Vavasis
• Def: A k-ply neighborhood system in d dimensions is a set \{D_1,\ldots,D_n\} of closed disks in \( \mathbb{R}^d \) such that no point in \( \mathbb{R}^d \) is strictly interior to more than k disks
• Def: An \((\alpha,k)\) overlap graph is a graph defined in terms of \( \alpha \geq 1 \) and a k-ply neighborhood system \{D_1,\ldots,D_n\}: There is a node for each \( D_j \), and an edge from \( j \) to \( i \) if expanding the radius of the smaller of \( D_j \) and \( D_i \) by \( >\alpha \) causes the two disks to overlap

Ex: n-by-n mesh is a \((1,1)\) overlap graph
Ex: Any planar graph is \((\alpha,k)\) overlap for some \( \alpha,k \)

2D Mesh is \((1,1)\) overlap graph
Generalizing Lipton/Tarjan to Higher Dimensions

• *Theorem* (Miller, Teng, Thurston, Vavasis, 1993): Let $G=(N,E)$ be an $(\alpha,k)$ overlap graph in $d$ dimensions with $n=|N|$. Then there is a vertex separator $N_s$ such that
  • $N = N_1 \cup N_s \cup N_2$ and
  • $N_1$ and $N_2$ each has at most $n^{(d+1)/(d+2)}$ nodes
  • $N_S$ has at most $O(\alpha \cdot k^{1/d} \cdot n^{(d-1)/d})$ nodes

• When $d=2$, same as Lipton/Tarjan

• Algorithm:
  • Choose a sphere $S$ in $\mathbb{R}^d$
  • Edges that $S$ “cuts” form edge separator $E_S$
  • Build $N_S$ from $E_S$
  • Choose $S$ “randomly”, so that it satisfies Theorem with high probability
Stereographic Projection

• Stereographic projection from plane to sphere
  • In d=2, draw line from p to North Pole, projection p’ of p is where the line and sphere intersect

\[ p = (x,y) \quad p' = \frac{2x,2y,x^2 + y^2 -1}{x^2 + y^2 + 1} \]

• Similar in higher dimensions
Choosing a Random Sphere

- Do stereographic projection from $\mathbb{R}^d$ to sphere $S$ in $\mathbb{R}^{d+1}$
- Find centerpoint of projected points
  - Any plane through centerpoint divides points evenly
  - There is a linear programming algorithm, cheaper heuristics
- *Conformally map* points on sphere
  - *Rotate* points around origin so centerpoint at $(0, \ldots, 0, r)$ for some $r$
  - *Dilate* points (unproject, multiply by $\sqrt{\frac{1-r}{1+r}}$, project)
    - this maps centerpoint to origin $(0, \ldots, 0)$, spreads points around $S$
- Pick a random plane through origin
  - Intersection of plane and sphere $S$ is “circle”
- Unproject circle
  - yields desired circle $C$ in $\mathbb{R}^d$
- Create $N_s$: $j$ belongs to $N_s$ if $\alpha * D_j$ intersects $C$
Random Sphere Algorithm (Gilbert)
Random Sphere Algorithm (Gilbert)
Figure 3: Projected mesh points. The large dot is the center point.
Random Sphere Algorithm (Gilbert)
Random Sphere Algorithm (Gilbert)
Figure 5: The separating circle projected back to the plane.
Nodal Coordinates: Summary

- Other variations on these algorithms
- Algorithms are efficient
- Rely on graphs having nodes connected (mostly) to “nearest neighbors” in space
  - algorithm does not depend on where actual edges are!
- Common when graph arises from physical model
- Ignores edges, but can be used as good starting guess for subsequent partitioners that do examine edges
- Can do poorly if graph connection is not spatial:

- Details at
  - www.cs.berkeley.edu/~demmel/cs267/lecture18/lecture18.html
  - www.cs.ucsb.edu/~gilbert
  - www.cs.bu.edu/~steng
Partitioning without Nodal Coordinates
E.g., In the WWW, nodes are web pages
Coordinate-Free: Breadth First Search (BFS)

- Given $G(N,E)$ and a root node $r$ in $N$, BFS produces
  - A subgraph $T$ of $G$ (same nodes, subset of edges)
  - $T$ is a tree rooted at $r$
  - Each node assigned a level = distance from $r$
Partitioning via Breadth First Search

- BFS identifies 3 kinds of edges
  - Tree Edges - part of T
  - Horizontal Edges - connect nodes at same level
  - Interlevel Edges - connect nodes at adjacent levels
- No edges connect nodes in levels differing by more than 1 (why?)
- BFS partitioning heuristic
  - \( N = N_1 \cup N_2 \), where
    - \( N_1 = \{ \text{nodes at level } \leq L \} \)
    - \( N_2 = \{ \text{nodes at level } > L \} \)
  - Choose \( L \) so \( |N_1| \) close to \( |N_2| \)

BFS partition of a 2D Mesh using center as root:
- \( N_1 = \) levels 0, 1, 2, 3
- \( N_2 = \) levels 4, 5, 6
Coordinate-Free: Kernighan/Lin

• Take a initial partition and iteratively improve it
  • Kernighan/Lin (1970), cost = O(|N|^3) but easy to understand
  • Fiduccia/Mattheyses (1982), cost = O(|E|), much better, but more complicated

• Given G = (N,E,W_E) and a partitioning N = A U B, where |A| = |B|
  • T = cost(A,B) = Σ {W(e) where e connects nodes in A and B}
  • Find subsets X of A and Y of B with |X| = |Y|
  • Swapping X and Y should decrease cost:
    • newA = A - X U Y and newB = B - Y U X
    • newT = cost(newA , newB) < cost(A,B)

• Need to compute newT efficiently for many possible X and Y, choose smallest
Kernighan/Lin: Preliminary Definitions

- $T = \text{cost}(A, B), \quad \text{new}T = \text{cost}(\text{new}A, \text{new}B)$
- Need an efficient formula for new$T$; will use
  - $E(a) = \text{external cost of } a \text{ in } A = \sum \{W(a,b) \text{ for } b \text{ in } B\}$
  - $I(a) = \text{internal cost of } a \text{ in } A = \sum \{W(a,a') \text{ for other } a' \text{ in } A\}$
  - $D(a) = \text{cost of } a \text{ in } A = E(a) - I(a)$
  - $E(b), I(b) \text{ and } D(b) \text{ defined analogously for } b \text{ in } B$
- Consider swapping $X = \{a\}$ and $Y = \{b\}$
  - new$A = A - \{a\} U \{b\}, \quad \text{new}B = B - \{b\} U \{a\}$
  - new$T = T - (D(a) + D(b) - 2*W(a,b)) = T - \text{gain}(a,b)$
    - gain$(a,b)$ measures improvement gotten by swapping $a$ and $b$
- Update formulas
  - new$D(a') = D(a') + 2*W(a',a) - 2*W(a',b) \quad \text{for } a' \text{ in } A, a' \neq a$
  - new$D(b') = D(b') + 2*W(b',b) - 2*W(b',a) \quad \text{for } b' \text{ in } B, b' \neq b$
**Kernighan/Lin Algorithm**

Compute \( T = \text{cost}(A, B) \) for initial \( A, B \) \( \quad \ldots \quad \text{cost} = O(|N|^2) \)

Repeat

\[ \ldots \text{One pass greedily computes } |N|/2 \text{ possible } X, Y \text{ to swap, picks best} \]

Compute costs \( D(n) \) for all \( n \) in \( N \) \( \quad \ldots \quad \text{cost} = O(|N|^2) \)

Unmark all nodes in \( N \) \( \quad \ldots \quad \text{cost} = O(|N|) \)

While there are unmarked nodes \( \quad \ldots \quad |N|/2 \text{ iterations} \)

\[ \text{Find an unmarked pair } (a, b) \text{ maximizing } \text{gain}(a, b) \quad \ldots \quad \text{cost} = O(|N|^2) \]

Mark \( a \) and \( b \) (but do not swap them) \( \quad \ldots \quad \text{cost} = O(1) \)

Update \( D(n) \) for all unmarked \( n \),

\[ \quad \ldots \quad \text{as though } a \text{ and } b \text{ had been swapped} \quad \ldots \quad \text{cost} = O(|N|) \]

Endwhile

\[ \ldots \text{At this point we have computed a sequence of pairs} \]

\[ \ldots \quad (a_1, b_1), \ldots, (a_k, b_k) \quad \text{and gains } \text{gain}(1), \ldots, \text{gain}(k) \]

\[ \ldots \quad \text{where } k = |N|/2, \text{ numbered in the order in which we marked them} \]

Pick \( m \) maximizing \( \text{Gain} = \sum_{k=1}^{m} \text{gain}(k) \) \( \quad \ldots \quad \text{cost} = O(|N|) \)

\[ \ldots \text{Gain is reduction in cost from swapping } (a_1, b_1) \text{ through } (a_m, b_m) \]

If \( \text{Gain} > 0 \) then \( \ldots \text{it is worth swapping} \)

Update \( \text{newA} = A - \{ a_1, \ldots, a_m \} \cup \{ b_1, \ldots, b_m \} \) \( \quad \ldots \quad \text{cost} = O(|N|) \)

Update \( \text{newB} = B - \{ b_1, \ldots, b_m \} \cup \{ a_1, \ldots, a_m \} \) \( \quad \ldots \quad \text{cost} = O(|N|) \)

Update \( T = T - \text{Gain} \) \( \quad \ldots \quad \text{cost} = O(1) \)

endif

Until \( \text{Gain} <= 0 \)
Comments on Kernighan/Lin Algorithm

• Most expensive line shown in red, \( O(n^3) \)
• Some gain\( (k) \) may be negative, but if later gains are large, then final Gain may be positive
  • can escape “local minima” where switching no pair helps
• How many times do we Repeat?
  • K/L tested on very small graphs (\(|N|\leq 360\)) and got convergence after 2-4 sweeps
  • For random graphs (of theoretical interest) the probability of convergence in one step appears to drop like \( 2^{-|N|/30} \)
Coordinate-Free: Spectral Bisection

- Based on theory of Fiedler (1970s), popularized by Pothen, Simon, Liou (1990)
- Motivation, by analogy to a vibrating string
- Implementation via the Lanczos Algorithm
  - To optimize sparse-matrix-vector multiply, we graph partition
  - To graph partition, we find an eigenvector of a matrix associated with the graph
  - To find an eigenvector, we do sparse-matrix vector multiply
  - No free lunch ...