The PageRank Derby

Karen Devine, Jon Berry and Steve Plimpton
Scalable Algorithms Department
Sandia National Laboratories

CSRI Student Seminar
July 2008
Motivation

• Graph algorithms perform extremely well on multithreaded architectures like the Cray MTA-2.
  – Latency tolerance is key.
• But is there a role for distributed memory computers in informatics?
  – More prevalent than MTA in research communities.
  – Less expensive than MTA.
  – Lots of expertise at Sandia.
• How do these platforms compare in performance?
  – This work includes the first apples-to-apples comparison of platforms on realistic data.
Massive Multithreading:
The Cray MTA and XMT

- Slow clock rate.
  - 220Mhz on MTA; 500Mhz on XMT.
- 128 “streams” per processor.
- Latency tolerant: Important for graph algorithms.
- Fine-grain parallelism.
- Simple, serial-like programming model with global address space.
- Advanced parallelizing compilers.
Distributed Memory: Clusters and RedStorm

- Fast clock rate (2+ Ghz).
- Local memory and cache.
- Data dependencies satisfied through message passing (e.g., MPI library).
- Communication more costly than computation.
- Coarse-grain parallelism.
- Parallelization done “manually.”
  - Data distribution and load balancing determined by application.
- Highly successful for wide range of PDE simulations.

Sandia’s RedStorm
PageRank

- Page, Brin, Motwani, Winograd; 1998
- Basis of Google’s web-page ranking system.
- Floating point computation on unstructured data.
- Premises:
  - Important pages link to other important pages.
  - Share of importance propagated is inversely proportional to number of outlinks.
PageRank

- Formulated as a Markov chain:
  - States $V$ are web pages.
  - Transition primarily according to hyperlinks $E$.
  - PageRank iterates to steady state.

<table>
<thead>
<tr>
<th>Markov Model</th>
<th>Graph</th>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>States $V$</td>
<td>Vertices $V$</td>
<td>Rows $A_i$</td>
</tr>
<tr>
<td>Probability of transition from state $i$ to state $k$.</td>
<td>Weighted directed edge $e_{ik}$</td>
<td>Nonzero entry $A_{ik}$</td>
</tr>
</tbody>
</table>
PageRank Example: Basic Model

- If \( v_i \) links to \( v_k \)...
  - User equally likely to follow any link on page.
  - Probability of moving from \( v_i \) to \( v_k \) = \( \frac{1}{\text{out}_\text{degree}(v_i)} \).

\[
A = \begin{bmatrix}
0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\]

PageRank Example: Basic Model

- If $v_i$ links to $v_k$...
  - User equally likely to follow any link on page.
  - Probability of moving from $v_i$ to $v_k = \frac{1}{\text{out-degree}(v_i)}$.


$$A = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$
PageRank Example: Special Cases

• If $v_i$ has no outlinks…
  – User equally likely to jump to any state.
  – Probability = $1 / |V|$

\[
\begin{bmatrix}
0 & 1/2 & 1/2 & 0 & 0 & 0 \\
1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\
1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\
0 & 0 & 0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

PageRank Example: Special Cases

- Weighted moves from each page:
  - Percentage of moves that follow a link == $\alpha$.
  - Percentage of moves that are jumps to another page == $(1- \alpha)$.
  - Typically, $\alpha \in [0.8, 0.9]$.

\[
A = \begin{bmatrix}
0 & \alpha/2 & \alpha/2 & 0 & 0 & 0 \\
1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\
\alpha/3 & \alpha/3 & 0 & 0 & \alpha/3 & 0 \\
0 & 0 & 0 & 0 & \alpha/2 & \alpha/2 \\
0 & 0 & 0 & \alpha/2 & 0 & \alpha/2 \\
0 & 0 & 0 & \alpha & 0 & 0 \\
\end{bmatrix}
\]

PageRank Example: Special Cases

- Weighted moves from each page:
  - Percentage of moves that follow a link $\equiv \alpha$.
  - Percentage of moves that are jumps to another page $\equiv (1-\alpha)$.
  - Typically, $\alpha \in [0.8, 0.9]$.

$$A = \begin{bmatrix}
\beta & \alpha/2+\beta & \alpha/2+\beta & \beta & \beta & \beta \\
1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\
\alpha/3+\beta & \alpha/3+\beta & \beta & \beta & \alpha/3+\beta & \beta \\
\beta & \beta & \beta & \beta & \alpha/2+\beta & \alpha/2+\beta \\
\beta & \beta & \beta & \alpha/2+\beta & \beta & \alpha/2+\beta \\
\beta & \beta & \beta & \alpha+\beta & \beta & \beta \\
\end{bmatrix}$$


$\beta = (1-\alpha)/|V| = (1-\alpha)/6$
PageRank Example: Special Cases

• Weighted moves from each page:
  – Percentage of moves that follow a link == $\alpha$.
  – Percentage of moves that are jumps to another page == $(1-\alpha)$.
  – Typically, $\alpha \in [0.8, 0.9]$.

$A = \begin{bmatrix}
\beta & \alpha/2+\beta & \alpha/2+\beta & \beta & \beta & \beta \\
1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\
\alpha/3+\beta & \alpha/3+\beta & \beta & \beta & \alpha/3+\beta & \beta \\
\beta & \beta & \beta & \beta & \alpha/2+\beta & \alpha/2+\beta \\
\beta & \beta & \beta & \alpha/2+\beta & \beta & \alpha/2+\beta \\
\beta & \beta & \beta & \alpha+\beta & \beta & \beta
\end{bmatrix}$

$\beta = (1-\alpha)/|V| = (1-\alpha)/6$

For efficiency: 
Keep the Sparsity

\[
A = \begin{bmatrix}
\beta & \alpha/2+\beta & \alpha/2+\beta & \beta & \beta & \beta \\
1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\
\alpha/3+\beta & \alpha/3+\beta & \beta & \beta & \alpha/3+\beta & \beta \\
\beta & \beta & \beta & \beta & \alpha/2+\beta & \alpha/2+\beta \\
\beta & \beta & \beta & \alpha/2+\beta & \beta & \alpha/2+\beta \\
\beta & \beta & \beta & \alpha+\beta & \beta & \beta \\
\end{bmatrix}
\]

\[
A = \alpha \begin{bmatrix}
0 & 1/2 & 1/2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\
0 & 0 & 0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix} + \alpha \begin{bmatrix}
0 \\
1/6 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix} + \beta ee^T
\]

\[
\beta = (1-\alpha)/|V| = (1-\alpha)/6
\]
Power Method Iteration

- PageRank is then Power Method iteration.
  - Propagate importance until converged.
  - Initialize PageRank vector $x$ to uniform distribution.
  - Do $x_{n+1}^T = x_n^T A$ until $|x_{n+1} - x_n| < tolerance$

- Key computational kernels:
  - Matrix-vector multiplication.
    - Link transitions
  - Loops over vector entries.
    - Adjustments for special cases
    - Norm and residual computations
MultiThreaded Implementation

- MultiThreaded Graph Library (MTGL)
  - Enables multithreaded graph algorithms.
  - Builds upon community standard (Boost Graph Library).
  - Abstracts data structures and other application details.
  - Hide some shared memory issues.
  - Preserves good multithreaded performance.
PageRank
MultiThreaded Implementation

• Use graph from Markov model.
• Store graph in compressed-row sparse format.
  – Vertex index array points to list of incoming edges’ source vertices.

```
Vertex: 1 2 3 4 5 6
Index: 0 1 3 4 6 8 10
IncomingEdgeList: 3 1 3 1 5 6 3 4 4 5
```

• Other representations would work with the same MTGL PageRank code.
**PageRank**

**MultiThreaded Implementation**

- Algorithm looks like serial code.

```c
#pragma mta assert nodep
// Loop over all vertices
for (int i=0; i<nVtx; i++) {
    double total=0.0;
    int begin = index[i];
    int end = index[i+1];
    // Loop over edges pointing to vertex i.
    for (int j=begin; j<end; j++) {
        int src = j;
        double r = rank[src];
        double incr = r/(double)degree[src];
        total += incr;
    }
    accumulate_rank[i] = total;
}
```

- Requirement for scaling:
  - Single thread contains the loop over incoming edges of a given vertex.
  - Enables compiler to generate code without hot spots.
PageRank Distributed Memory Implementation

- Use matrix-representation $A$.
- Iterate to steady-state: $x_{n+1}^T = x_n^T A$
- For efficiency, don’t store jump transitions in $A$.
  - Matrix-vector multiplication with sparse link transition matrix.
  - Loops over vectors for jump adjustments.

$$x_{n+1}^T = x_n^T A$$

Row-based Distribution

- Trilinos solver framework (Heroux et al.)
- Epetra distributed matrix/vector classes.
- Default data distribution:
  - Each processor stores an equal number of rows, distributed linearly.
  - Vectors distributed with same map.
- Sparse point-to-point communication used in matrix-vector product.
  - Send only data needed; no zero-values.
  - Possibly communicate with all processors.
- Global communication needed for computing norms and residuals.
Block-Based Decomposition

- Matrix2D class (Plimpton)
- Logically arrange $P$ processors into array of size $\sqrt{P} \times \sqrt{P}$.
- Assign each processor a block of the matrix.
- Most communication done only along processor rows or columns.
  - Communicating with at most $\sqrt{P}$ neighbors.
  - Communications include all vector entries in row or column.
- Global communication needed for computing norms and residuals.
Recursive Matrix (R-MAT) Model

- (Chakrabarti, Zhan, Faloutsos; 2004).
- Commonly used to represent web and social networks.
- Power-law degree distributions.
- Generate edges $E$ through recursive operations in adjacency matrix $A$.
  - $A_{ik} > 0 \iff e_{ik} \in E$
  - Four parameters determine edge distribution: $a + b + c + d = 1.$
R-MAT Parameters

• “Nice” data parameters:
  – a = 0.45
  – b = 0.15
  – c = 0.15
  – d = 0.25

“Nice” R-MAT 14: |V|=16,384 ; |E|=131,072
Max degree: 112

• “Nasty” data parameters:
  – a = 0.57
  – b = 0.19
  – c = 0.19
  – d = 0.05

“Nasty” R-MAT 14: |V|=16,384 ; |E|=131,072
Max degree: 1,666
Experiments

• Data: R-MAT 25
  – Average degree = 8
  – $|V| = 2^{25} > 33M$; $|E| = 2^{28} > 268M$.
  – “Nice” max degree = 1,108
  – “Nasty” max degree = 230,207

• Architectures:
  – Distributed Memory:
    • Tbird cluster: 3.6GHz Intel EM64T; Infiniband CLOS network
    • Redstorm: 2.4GHz Operton; 3D Mesh network
  – Multithreaded Architectures:
    • Cray MTA: 220 MHz; Modified Caley network
    • Cray XMT: 500 MHz; 3D Torus network
Load Imbalance

- Number of rows per processor is uniform.
- Number of nonzeros per processor varies greatly.
  - Reduces scalability of matrix-vector multiplication.

**NICE data**

**NASTY data**
Multiconstraint Partitioning

• Ideal distribution would have...
  – Uniform number of rows per processor AND
  – Uniform number of nonzeros per processor.

• Multiconstraint Partitioning
  – (Karypis, Schloegel, Catalyurek)
  – Specify vector of weights for each row.
    • Weight for row $k = [1, \# \text{ nonzeros in row } k]$
  – Find distribution of rows that is balanced with respect to both components.

• PaToH Multiconstraint Hypergraph Partitioner
  (Catalyurek)
Multiconstraint Partitioning Results

**Row Distribution**

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>16</th>
<th>64</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Rows Per Processor</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Nonzero Distribution**

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>16</th>
<th>64</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nonzeros Per Processor</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RMAT-23 NASTY

<table>
<thead>
<tr>
<th>V</th>
<th>8.38M</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>67.1M</td>
</tr>
</tbody>
</table>
Max Vertex Degree: 94,561
PageRank Scalability with Rebalancing

- Execution time on Tbird.
- PaToH Multiconstraint Partitioning.

![Graph showing scalability with rebalancing]

RMAT-23 NASTY
$|V| = 8.38M$
$|E| = 67.1M$
Max Vertex Degree: 94,561
Conclusions

• Distributed memory clusters can process large unstructured data sets in some contexts.
  – Scalability demonstrated up to 1000 processors.

• Massively multithreaded architectures can outperform clusters, even with floating-point computation.
  – MTA and XMT can do more than chase pointers.

• Less programmer intervention needed on massively multithreaded architectures than distributed memory architectures.
  – Less effort spent on data layout and load balancing (but more fussing with compilers).
Future Work

• PageRank Derby
  – More architectures (e.g., Netezza database machine).
  – More programming paradigms (e.g., MapReduce).
• Comparing other graph algorithms.
  – Determine feasibility on distributed memory systems.
• Parallel multiconstraint hypergraph partitioning in Zoltan and Isorropia.