Review of
Scalability! But at what COST?

Frank McSherry, Michael Isard, Derek G. Murray

Review by Lee Savoie

“You can have a second computer once you’ve shown you know how to use the first one.”
-Paul Barham
Speedup

\[ \text{speedup} = \frac{\text{serial execution time}}{\text{parallel execution time}} \]

- \[ \text{speedup} = \frac{1000 \text{ seconds}}{250 \text{ seconds}} = 4 \]
- \[ \text{speedup} = \frac{1000 \text{ seconds}}{280 \text{ seconds}} = 3.6 \]
Scalability

The graph shows the speed-up of two systems, A and B, as a function of the number of cores. System A has a higher speed-up compared to system B, indicating better scalability.
Background

• Big data systems
• Graph problems – non-trivial to parallelize
Problem

• How should we measure the performance of parallel code?
• Common practice is to calculate speedup by comparing a system with itself.
• Good speedup shows scalability but not performance.
  • “any system can scale arbitrarily well with a sufficient lack of care in its implementation.”

\[
\text{speedup} = \frac{\text{serial execution time}}{\text{parallel execution time}}
\]

• Sometimes we are parallelizing the overhead introduced by parallelization.
• Parallelism is often about performance.
Solution

- COST: the Configuration that Outperforms a Single Thread
- How many cores must we run the parallel code on to exceed the performance of the serial code?
- Measures overhead of parallelism.
Methodology

- Find published performance results from big data systems.
- Write a simple, single threaded implementation that solves the same problem.
- Run the serial implementation on a laptop using the same input as the published results.
- Compare the performance achieved by the laptop with the published performance.
Results: PageRank

<table>
<thead>
<tr>
<th>scalable system</th>
<th>cores</th>
<th>twitter</th>
<th>uk-2007-05</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphChi [10]</td>
<td>2</td>
<td>3160s</td>
<td>6972s</td>
</tr>
<tr>
<td>Stratosphere [6]</td>
<td>16</td>
<td>2250s</td>
<td>-</td>
</tr>
<tr>
<td>X-Stream [17]</td>
<td>16</td>
<td>1488s</td>
<td>-</td>
</tr>
<tr>
<td>Spark [8]</td>
<td>128</td>
<td>857s</td>
<td>1759s</td>
</tr>
<tr>
<td>Giraph [8]</td>
<td>128</td>
<td>596s</td>
<td>1235s</td>
</tr>
<tr>
<td>GraphLab [8]</td>
<td>128</td>
<td>249s</td>
<td>833s</td>
</tr>
<tr>
<td>GraphX [8]</td>
<td>128</td>
<td>419s</td>
<td>462s</td>
</tr>
<tr>
<td>Single thread (SSD)</td>
<td>1</td>
<td>300s</td>
<td>651s</td>
</tr>
<tr>
<td>Single thread (RAM)</td>
<td>1</td>
<td>275s</td>
<td>-</td>
</tr>
<tr>
<td>Hilbert order (SSD)</td>
<td>1</td>
<td>242s</td>
<td>256s</td>
</tr>
<tr>
<td>Hilbert order (RAM)</td>
<td>1</td>
<td>110s</td>
<td>-</td>
</tr>
</tbody>
</table>

- Note: reported Hilbert order performance does not include 179 seconds of pre-processing for the twitter input.
# Results: Connected Components

<table>
<thead>
<tr>
<th>scalable system</th>
<th>cores</th>
<th>twitter</th>
<th>uk-2007-05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratosphere [6]</td>
<td>16</td>
<td>950s</td>
<td>-</td>
</tr>
<tr>
<td>X-Stream [17]</td>
<td>16</td>
<td>1159s</td>
<td>-</td>
</tr>
<tr>
<td>Spark [8]</td>
<td>128</td>
<td>1784s</td>
<td>≥ 8000s</td>
</tr>
<tr>
<td>Giraph [8]</td>
<td>128</td>
<td>200s</td>
<td>≥ 8000s</td>
</tr>
<tr>
<td>GraphLab [8]</td>
<td>128</td>
<td>242s</td>
<td>714s</td>
</tr>
<tr>
<td>GraphX [8]</td>
<td>128</td>
<td>251s</td>
<td>800s</td>
</tr>
<tr>
<td>Single thread (SSD)</td>
<td>1</td>
<td>153s</td>
<td>417s</td>
</tr>
<tr>
<td>Union-Find (SSD)</td>
<td>1</td>
<td>15s</td>
<td>30s</td>
</tr>
</tbody>
</table>
COST: PageRank

- Left: time per warm iteration, right: time for ten iterations from cold start.
Another Example

Execution Time - FT

Speedup - FT
Analysis: why high COST?

- Parallel implementations add overhead.
- Hardware may value capacity or throughput over performance.
- System implementations add overhead (garbage collection, bounds checking, etc).
- Research systems may not be fully optimized.
- Parallel systems have other advantages: integration, security, reliability.
  - But you don’t need reliability if you run on a single core.
- Programming models may favor or require inefficient algorithms for some problems.
- Programming models can prevent some kinds of optimizations.
Future Questions

• How should we measure the performance of parallel code?
• Some systems are scalable and perform well.
  • What did they do right?
  • Can we apply this to other parallel systems?
• What is a good COST or a bad COST?
• What is an appropriate amount of overhead for a parallel system?
• Measuring COST requires implementing a system twice. Can we get the same information without the extra work?
• Anything else?
Critique

• The comparisons in this paper are not completely fair.
  • This is acknowledged in the paper.
• Speedup can compare a good serial implementation with a good parallel implementation. Do we need a new metric?
• Perhaps scalability is the right metric. Speed doesn’t help if we can only solve small problems.
  • On the other hand, the serial codes in this paper processed millions of vertexes and billions of edges in under a minute.
• Perhaps the assumptions that underlie big data systems are no longer valid.
  • Should we start building smaller systems with more powerful components?
• This paper brings an important issue to light: parallel code only helps if it’s better than equivalent serial code.
This Semester’s Questions

• How does implementation of programming constructs impact design?
  • The programming model determines which algorithms are available. This can cause programmers to use inefficient algorithms that fit the model, thus wasting resources.

• How should programming models be evaluated?
  • Speedup is important.
  • So is COST.