Advanced Shortest Paths Algorithms on a Massively-Multithreaded Architecture

Joe Crobak, Jon Berry, Kamesh Madduri, and David Bader
Acknowledgement of Support

- National Science Foundation
  - CSR: A Framework for Optimizing Scientific Applications (06-14915)
  - CAREER: High-Performance Algorithms for Scientific Applications (06-11589; 00-93039)
  - ITR: Building the Tree of Life -- A National Resource for Phyloinformatics and Computational Phylogenetics (EF/BIO 03-31654)
  - ITR/AP: Reconstructing Complex Evolutionary Histories (01-21377)
  - DEB Comparative Chloroplast Genomics: Integrating Computational Methods, Molecular Evolution, and Phylogeny (01-20709)
  - ITR/AP(DEB): Computing Optimal Phylogenetic Trees under Genome Rearrangement Metrics (01-13095)
  - DBI: Acquisition of a High Performance Shared-Memory Computer for Computational Science and Engineering (04-20513).
- IBM PERCS / DARPA High Productivity Computing Systems (HPCS)
  - DARPA Contract NBCH30390004
- IBM Shared University Research (SUR) Grant
- Sony-Toshiba-IBM (STI)
- Microsoft Research
- Sun Academic Excellence Grant
- Cray Inc.
- Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000
Motivation

• Shortest path are at the core of several real-world graph problems, such as centrality metrics.

• Other Multithreaded SSSP algorithms perform more than optimal work.

• Support for simultaneous SSSP queries.
Our Contributions

• Experimental study of parallel version of Thorup’s algorithm for the single source shortest path problem.

• Memory-efficient mechanism for handling simultaneous shortest path queries.
Past Work on Parallel SSSP

- Relaxed heaps [DGST88], [BTZ98].
- Randomization [UY90].
- Delta-Stepping [MS03].
- Distributed memory implementations based on graph partitioning.
MTA $\Delta$-Stepping [MBBC07]

- Synthetic directed scale-free graph of 100 million vertices and 1 billion edges takes 9.73 seconds.
- Relative speedup of 31 on 40 processors of Cray MTA-2.
Parallel Shortest Path

• **Problem**: Dijkstra-based algorithms inherently serial.

• **Solutions**: Randomization, heuristics, bucketing ($\Delta$-stepping).
Thorup’s algorithm

- Undirected, integer weight edges
- Preprocess to build a tree structure called the *component hierarchy*.
- Scan edges out of source.
- Traverse the component hierarchy, which computes SSSP from source.
- \(O(n+m)\) time with \(n\) vertices and \(m\) edges.
Component Hierarchy
Traversing Component Hierarchy

• Components are *visited* recursively.
  – Determine which children to visit, and visit them recursively.

• If component represents a single vertex, then scan edges out of that vertex.
Component Bucketing

- Each component maintains the minimum distance value, $min-D$, for all unsettled vertices in its component.
- A component buckets its child $c$ based upon $c$’s $min-D$. 
Component Hierarchy in parallel.

- log(max weight) phases.
- Initially, graph contains only the vertices.
- At phase \( i \) add edges with weight \(<2^i\), collapse connected components into a node in the component hierarchy.
Example Component Hierarchy

Graph $G$

G without edges

G with $w=1$ edges

Leaves $\downarrow$ of CH

First level $\downarrow$ of CH
Example (cont.)
Traversing the Component Hierarchy in parallel

- Visiting a component
  - Discover which components are in lowest level bucket in parallel.

- Visiting a leaf component
  - Scan edges in parallel, update $min-D$ values.
Implementation details

- The number of children a component has varies from two to several thousand.
- Based upon the number of iterations, we either perform the following loop on all processors, a single processor, or in serial.

```c
int index=0;
#pragma mta assert nodep
for (int i=0; i<numChildren; i++) {
    CHNode *c = children_store[i];
    if (bucketOf[c->id] == thisBucket) {
        toVisit[index++] = child->id;
    }
}
```
Cray MTA-2 (XMT)

- **Tolerates latency** by multi-threading
  - hardware support for 128 threads on each processor
  - Globally hashed address space
  - No data cache
  - Single cycle context switch
  - Multiple outstanding memory requests
- Support for fine-grained, word-level synchronization
  - Full/empty bit associated with every memory word
- Flexibly supports dynamic load balancing
- Clock frequency: 220 MHz, largest machine: 40 processors
Experimental Setup

• Compared to DIMACS Reference solver for random graphs on a sequential compile on a Linux workstation.

• Compared to Delta-Stepping on random and scale free graphs on the Cray MTA-2.
### Sequential Performance

<table>
<thead>
<tr>
<th>Graph Family</th>
<th>Thorup</th>
<th>DIMACS reference solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rand UWD $2^{20} 2^{20}$</td>
<td>4.31s</td>
<td>1.66s</td>
</tr>
<tr>
<td>Rand UWD $2^{20} 2^2$</td>
<td>2.66s</td>
<td>1.24s</td>
</tr>
</tbody>
</table>
Random Graph Performance

Thorup’s Algorithm

![Graph showing time in seconds vs number of MTA Processors]

- "th-rand-uwd-2^25-2^25"
- "th-rand-pwd-3^25-3^25"
- "th-rand-uwd-2^24-2^22"

Time in Seconds

Number of MTA Processors
Scale Free Graph Performance

Thorup’s Algorithm

Time in Seconds

Number of MTA Processors
Simultaneous queries with shared CH

Simultaneous 40 Processor Thorup Runs from Multiple Sources

Time in Seconds

Number of Sources
Conclusions

- We experimentally evaluate a parallel implementation of Thorup’s algorithm for solving SSSP
- Our implementation demonstrates near-linear speedup on several low-diameter graph classes
- Our implementation supports simultaneous SSSP queries
- High diameter graphs are still a challenge to solve in parallel
Future Work

• Perform preprocessing for road networks.

• Phased implementation of Thorup’s algorithm.
Thank You

- Questions?

- Joe Crobak
  - crobakj@cs.rutgers.edu