

# Design and Implementation of Parallel Graph Algorithms on the Cray MTA-2

Joint work with  
 Prof. David A. Bader (Georgia Tech)  
 Jonathan Berry (Sandia National Labs)

Kamesh Madduri  
 kamesh@cc.gatech.edu

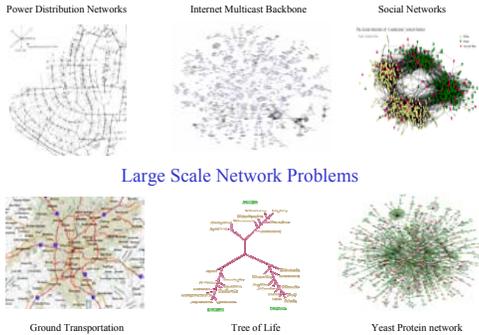
PhD student, College of Computing, Georgia Institute of Technology



## Abstract

We present fast parallel implementations of several fundamental graph theory problems on multithreaded architectures (e.g. the Cray MTA-2). The architectural features of the MTA-2 such as flat shared memory, fine-grained multithreading, and low-overhead synchronization aid the design of simple, scalable and high-performance graph algorithms. We test our implementations on large scale-free and sparse random graph instances, and report interesting results. For instance, Breadth-First Search on a scale-free graph of 500 million vertices and 2 billion edges takes 15 seconds on a 40-processor MTA-2 system with an absolute speedup of close to 30. This is a significant result in parallel computing, as prior implementations of parallel graph algorithms report very limited or no absolute speedup for irregular and sparse graphs

## Motivation



## Graph Algorithms

- memory intensive
- use sparse data structures
- poor cache locality
- tough to implement
- hard to achieve parallel speedup

## Current Parallel Systems

- physically distributed memory
- caches hide memory latency
- communication overhead
- coarse-grained parallelism
- no architectural support

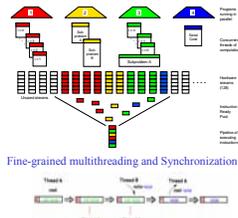
## Cray MTA-2

- flat shared memory
- no data cache
- hardware multi-threading to tolerate memory latency
- fine-grained synchronization
- 40 processors, 160 GB RAM
- 220 MHz clock
- 128 streams per processor
- 25 active threads to saturate



## Eldorado

- next-generation massively multithreaded supercomputer
- basic processing units – MTA-2
- uses the Cray XT3 interconnect
- 8192 processors, 128TB memory



## Graph Infrastructure

- based on generic programming ideas of the BOOST graph library
- extensible and modular
- several design levels and abstractions
- simple and efficient code
- C++, compiles on the Cray MTA-2, Linux
- Visitor level abstraction – users design simple components that can be plugged into the infrastructure
- Algorithm level – DFS, BFS, MST etc.
- Lower level abstractions – core data structures
- Visitation vs. Iteration
- Closures

## Algorithms

- Breadth-First Search
- st-connectivity
- Depth-First Search
- Connected Components
- Minimum Spanning Tree [Sollin]
- Shortest Path Algorithms
  - Dijkstra SSSP variants using *treaps* and *double-buckets*
  - Parallel SSSP [Meyer '03]
  - Thorup's SSSP alg., undirected graphs



## Algorithm Design Hurdles

- support for nested parallelism
- high-degree vertices
- hot spots
- race conditions

## Level-synchronized Parallel BFS

```

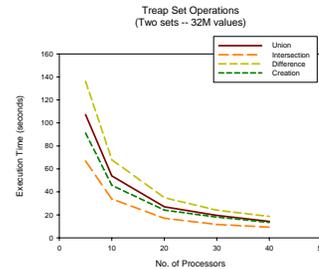
Input: G(V,E), source vertex s
Output: Array d [1..n] with d[v] holding the length of the shortest path from s to v in V, assuming unit-weight edges
1 for (all v in V) in parallel do
2   d[v] ← -1;
3   d[s] ← 0;
4   Q ← φ;
5   Enqueue s ← Q;
6   while (Q ≠ φ) do
7     for (all u in Q) in parallel do
8       Delete u ← Q;
9       for (each v adjacent to u) in parallel do
10        if (d[v] = -1) then
11          d[v] ← d[u] + 1;
12          Enqueue v ← Q;
    
```

```

/* While the Queue is not empty */
#pragma mta assert parallel
#pragma mta loop future
for (i=startIndex; i<endIndex; i++) {
  u = Q[i];
  /* Inspect all vertices adjacent to u */
  #pragma mta assert parallel
  for (j=0; j < degree[u]; j++) {
    v = neighbor[u][j];
    /* Check if v has been visited yet? */
    dist = readfc(&d[v]);
    if (dist == -1)
      writefc(&d[v], d[u] + 1);
    else
      writefc(&d[v], dist);
    /* Enqueue v */
    Q[fin_fetch_add(&count, 1)] = v;
  }
}
    
```

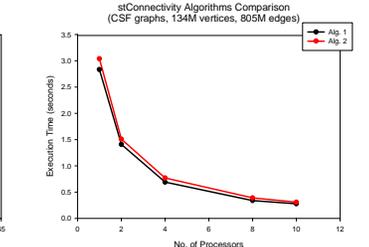
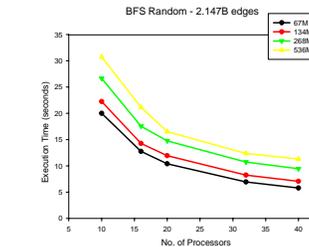
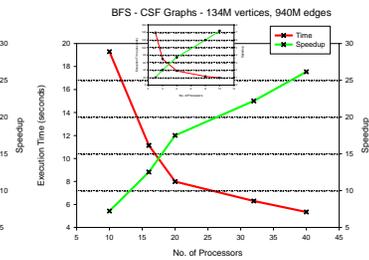
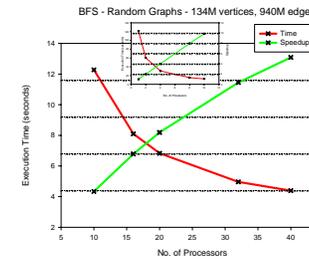
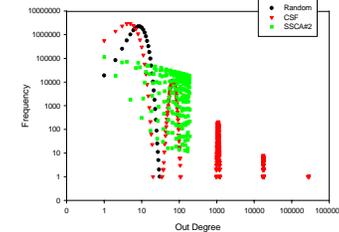
## Data Structures

- treaps
- van Emde Boas trees
- dynamic arrays
- hash tables
- priority queues
  - pairing heaps
  - fibonacci heaps



## Performance

Degree distributions of Test graphs (16M vertices, 150M edges)



- Cray MTA-2 – 40 processors: BFS on a graph of 528M vertices, 2.1B edges  
Scale-free – 17.32 sec., Random – 13.74 sec.
- IBM BlueGene/L – 32K processors: BFS\* on a random Poisson graph  
3.2B vertices, average degree of 10 – 4.9 sec.
- 128-node 2GHz dual Opteron cluster - Parallel BGL\*\*  
BFS: Random graph, 1M vertices and 15M edges  
1 node: 40 sec., 20 nodes: 10 sec., 70 nodes: 3 sec., 100 nodes : 10 sec.

\* [Chow '05] \*\* Lumsdaine, Parallel BGL Performance, <http://www.osl.nyu.edu/research/pbgl/performance/>

## Related Work

- D. A. Bader and K. Madduri, *Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2* (submitted)
- D.A. Bader, G. Cong, and J. Feo, *On the Architectural Requirements for Efficient Execution of Graph Algorithms*, ICPP 2005