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

Kamesh Madduri
kamesh@cc.gatech.edu
PhD student, College of Computing, Georgia Institute of Technology

We present fast parallel implementations of several fundamental graph theory problems on multithreaded architectures like the Cray MTA-2. The architectural features of the MTA-2 such as the shared memory, fine-grained multithreading, and low-overhead synchronization all the design of graph, scalable and high-performance graph algorithms. We use our implementations on large-scale graphs to show the benefits of parallel graph algorithms over sequential algorithms, and to demonstrate the applicability of multi-threaded computing to large-scale applications. First, we solve a scale-free graph of 500 million vertices and 2 billion edges in 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 speedups.

Abstract

Motivation

Large Scale Network Problems

Graph Algorithms

Current Parallel Systems

Graph Infrastructure

Algorithm Design Hurdles

Level-synchronized Parallel BFS

MTA-2 Programming is simple! (code corresponding to steps 7-12)

Performance

Related Work

Based on generic programming ideas of the BOOST graph library – users design simple components that can be plugged into the infrastructure

Algorithm level – DFS, BFS, MST etc.

Lower level abstractions – core data structures

Visualization vs. Iteration

Closures

Motivation

Visitor level abstraction – users design simple and 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 speedups.

Graph Algorithms

• 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

BFS – Random Graphs – 134M vertices, 940M edges

Dijkstra SSSP variants using treaps and double-buckets

Parallel SSSP [Meyer ’03]

Thorup’s SSSP algo., undirected graphs

MTA-2 Programming is simple! (code corresponding to steps 7-12)

While the Queue is not empty

Enqueue v

BFS Random - 2.147B edges

D. A. Bader and K. Madduri, 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

Visitation vs. Iteration

Power Distribution Networks

Internet Multicast Backbone

Social Networks

Ground Transportation

Tree of Life

Yeast Protein network

Input: G(V, E), source vertex s
Output: Array of [v,E] with d[v] holding the length of the shortest path from s to v in V, assuming unit-weight edges

• Breadth-First Search

• x-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 algo., undirected graphs

Multithreading and Synchronization

Fine-grained multithreading and synchronization

• Treaps

• van Emde Boas trees

• dynamic arrays

• hash tables

• priority queues

• pairing heaps

• fibonacci heaps

Cray MTA-2 – 40 processors: BFS on a graph of 528M vertices, 2.1B edges

Sources: Eldorado [Feo 2005], Cray MTA-2 User manual, Karo Designs (http://www.karo.com)