DSMR: A Shared and Distributed Memory Algorithm for Single-Source Shortest Path Problem *

Saeed Maleki† Donald Nguyen‡ Andrew Lenharth‡
María Garzarán† David Padua† Keshav Pingali‡

†: Department of Computer Science, University of Illinois at Urbana-Champaign
‡: Department of Computer Science, The University of Texas at Austin
maleki1@illinois.edu, {ddn@cs,lenharth@ices}.utexas.edu, {garzaran,padua}@illinois.edu, pingali@cs.utexas.edu

Abstract

The Single-Source Shortest Path (SSSP) problem is to find the shortest paths from a source vertex to all other vertices in a graph. In this paper, we introduce the Dijkstra Strip-Mined Relaxation (DSMR) algorithm, an efficient parallel SSSP algorithm for shared and distributed memory systems. Our results show that, DSMR is faster than parallel Δ-Stepping by a factor of up-to 1.66.

1. Introduction

As the size of the input graphs increases, parallel graph algorithms are becoming increasingly important, as faster processing strategies are needed. *Scale-free networks* [1] such as Twitter's tweets graph are among the most important examples of today's large graphs.

This paper presents DSMR (Dijkstra Strip Mined Relaxation), a new shared and distributed memory parallel algorithm for the Single-Source Shortest Path (SSSP) problem that is particularly efficient on scale-free networks. Given a weighted graph G and a source vertex s in G, the SSSP problem computes the shortest distance from s to all vertices of G. SSSP is a classical problem and has many applications, such as transportation, robotics, and the computation of Betweenness Centrality. Our results show that DSMR is up-to between $1.50 - 1.66$ times faster than Δ-Stepping implementations.

2. Parallelizing SSSP

2.1 Background

We use the following notations: vertex v_0 is the source vertex, $w(v_iv_j)$ is the weight of edge v_iv_j and $d(v_i)$ is the *current* distance of v_i. The value of $d(v_i)$ typically changes as the algorithm advances in the computation. We use $d_f(v_i)$ to represent the final value of $d(v_i)$. Initially, $d(v_0)$ is set to 0 and $d(v_i)$ is set to ∞ for the other vertices. Then, a set of *relaxation* operations is used to compute successive values of the shortest distances for each vertex.

Relaxation: Relaxation is a basic operation used by SSSP algorithms. There are two types of relaxation operations: 1) *Relaxing an edge* v_iv_j which updates $d(v_j)$ to $\min\{d(v_j), d(v_i) + w(v_iv_j)\}$. 2) *Relaxing a vertex* v_i which relaxes all of its incident edges (outgoing edges in directed graphs). Relaxation of a vertex v_i becomes necessary when its distance, $d(v_i)$, is updated (updates always lower the distance). A vertex whose distance has been updated becomes *active* until the vertex is relaxed. SSSP starts by relaxing the source vertex, followed by the repetitive relaxation of the active vertices. However, since there could be multiple active vertices at a time, there are multiple possible orders of relaxation, which could be partial orders if parallel relaxations are allowed. We call the order of relaxation a *schedule*. We say that *the amount of work* of a SSSP algorithm is *the total number of edge relaxation* it does. Since the minimum number of relaxations needed to compute SSSP is equal to the number of edges, E, we say that the *overhead* of an algorithm is the number of edge relaxations minus E.

Among the existing SSSP algorithms, Dijkstra, Bellman-Ford, and Chaotic Relaxation, Δ-Stepping [6] has a reasonable balance between parallelism and overhead. Δ-Stepping proceeds iteratively. In each iteration i where $i \in \{0, 1, 2, \dots\}$, all active vertices v such that $i.\Delta \leq d(v) < (i + 1).\Delta$ are relaxed. Here Δ is a constant throughout the algorithm. However, as shown later, Δ-Stepping performs poorly when applied to scale-free graphs.

2.2 Dijkstra Strip Mined Relaxation

Dijkstra Strip Mined Relaxation (DSMR) is our new parallel SSSP algorithm. First, DSMR partitions the set of vertices into P (total number of processors) subsets $U_1, U_2, \dots, U_P$. Using these subsets, it creates P subgraphs. The subgraph assigned to processor k consists of the vertices in U_k plus all vertices adjacent to those in U_k. The adjacent vertices added to complete each subgraph are called *halo* vertices. Processor i computes the shortest distance from the source to all vertices in U_i minus the halo vertices.

DSMR proceeds in three stages: 1) Each processor relaxes the active vertices in its subgraph in *distance order* using only local information until it has relaxed exactly D edges. 2) After D edges have been relaxed, all processors reach a collective all-to-all communication that updates the distances of halo vertices across all processors. 3) Each processor locally finds the active vertices it owns. These 3 stages continue until there are no more active vertices. Large D values cause late distance updates and work overhead and small D values cause frequent collective communications increasing communication cost. To study how DSMR's overhead compares with Δ-Stepping's, we studied the Degree-Distance and Overhead Distributions.

* The work presented in this paper has been supported by the National Science Foundation grant CNS 1111407. This research used resources of the Argonne Leadership Computing Facility, which is a DOE office of Science User Facility supported under Contract DE-AC02-06CH11357.

PPoPP '16, March 12-16, 2016, Barcelona, Spain
Copyright © 2016 ACM 978-1-4503-4092-2/16/03…$15.00
DOI: http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2851141.2851183

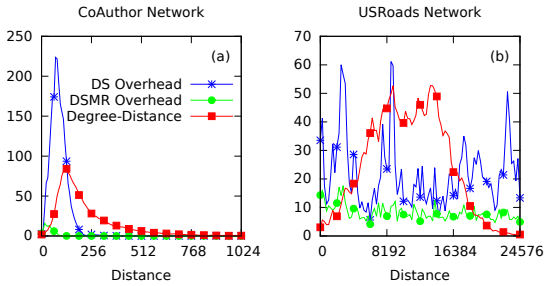

Figure 1: Overhead distribution of Δ-Stepping (DS) algorithm and DSMR algorithm compared with degree-distance distribution.

2.3 Degree-Distance and Overhead Distributions

Degree-Distance Distribution: Degree-Distance distribution $y(x)$, is measured after computing the final shortest distances. $y(x)$ is the total number of edges that are connected to vertices with shortest distance x. Formally, $y(x) = \sum_{v:d_f(v)=x} degree(v)$. The Degree-Distance plot (red plot) in Figure 1 shows the degree-distance distribution from a random source vertex for Co-Author and US Roads networks, a scale-free network and a non-scale-free network, respectively.

The Coauthor Network's plot has a narrow Gaussian shape with a long tail while the US Roads network's plot has a wide Gaussian shape with short head and tail. For the Co-Author, the Δ-Stepping algorithm relaxes significantly more edges in earlier iterations than in later iterations since in each iteration, it relaxes vertices in a Δ range of distances (fixed length successive non-overlapping intervals on the horizontal axis). This imbalance causes significant work or communication overhead as described next. However, DSMR has constant amount of work in each iteration. For US Roads plot in Figure 1(b), the amount of work for Δ-Stepping is roughly uniform across iterations, making Δ-Stepping suitable for this graph.

Overhead Distribution: The work overhead of a SSSP algorithm associated with a vertex v is caused by relaxing it before it reaches its final distance $d_f(v)$. We associate the source of this overhead with the first vertex u that should have been relaxed before v to avoid the overhead. For each such u, $degree(v)$, the amount of work overhead, is accumulated into the overhead distribution, $z(x)$, where $x = d_f(u)$. The blue and green plots in Figure 1 show the overhead distribution of Δ-Stepping (DS) and DSMR for the two networks, respectively. The values of D and Δ are chosen such that the number of collective communications are almost the same for both algorithms. As it can be seen, the overhead distribution of DSMR is roughly uniform for both algorithm while this is not the case for Δ-Stepping. This is the main reason why DSMR performs better than Δ-Stepping.

3. Results

We used two machines for our evaluation: a shared memory machine with 40 cores (4 10-core Inten Xeon E7-4860) and Mira, a distributed-memory cluster at Argonne National lab with 16-core nodes (PowerPC A2). The four graphs for this study are Co-Author network [7], US roads network [4], R-MAT [3] and Orkut [9].

Figure 2 compares DSMR, our implementation of Δ-Stepping (DS), the Δ-Stepping from the Elixir collection [8] implemented in the Galois system [5], and the version of Δ-Stepping described in [2] (IPDPS-DS). Plots (a) and (b) show the results for the Co-Author and US Roads networks on the shared memory machine and plots (c) and (d) show the results for Orkut and R-MAT graphs on Mira. Plots (a), (b) and (c) show strong scaling results while

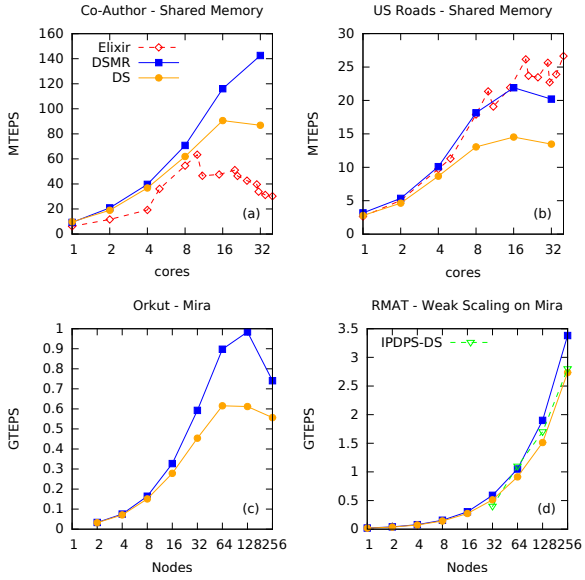

Figure 2: Comparison of DSMR and different versions of Δ-Stepping on the shared and distributed memory systems

plot (d) shows weak scaling results. For each algorithm, the best parameters (Δ in Δ-Stepping and D in DSMR) were searched from a random source vertex. We found these parameters to be stable when changing the source vertex and, therefore, we executed all algorithms with them for 100 other random source vertices. The performance results are presented in TEPS (Traversed Edges Per Second) which is $|E(G)|/T$, where T is the running time in seconds. The X axis represents different number of cores in plot (a) and (b) and different number of nodes (16 core/node) in plots (c) and (d). The Y axis shows the average TEPS (either in Mega or Giga) of the 100 random source vertices.

As can be seen from all plots in Figure 2, DSMR is faster than the Δ-Stepping implementations except for the US Roads network where DSMR is $0.75\times$ slower than Elixir. For the other cases, DSMR is between $1.50\times$ to $1.66\times$ faster than all the other Δ-Stepping implementations. Also, note that DSMR speed up over Δ-Stepping improves as the number of cores is increased.

References

[1] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science'99*.

[2] V. Chakaravarthy, F. Checconi, F. Petrini, and Y. Sabharwal. Scalable single source shortest path algorithms for massively parallel systems. In *IPDPS'14*.

[3] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *SDM'04*.

[4] C. Demetrescu, A. V. Goldberg, and D. S. Johnson. Implementation challenge for shortest paths. In *Encyclopedia of Algorithms'08*.

[5] Galois System. http://iss.ices.utexas.edu/?p=projects/galois.

[6] U. Meyer and P. Sanders. Delta-stepping: A parallelizable shortest path algorithm. *J. Algorithms'03*.

[7] G. Palla, I. J. Farkas, P. Pollner, I. Dernyi, and T. Vicsek. Fundamental statistical features and self-similar properties of tagged networks. *New Journal of Physics*.

[8] D. Prountzos, R. Manevich, and K. Pingali. Elixir: A system for synthesizing concurrent graph programs. OOPSLA '12.

[9] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *MDS '12*.