A Scalable General Purpose System for Large-Scale Graph Processing

Sun, J. (2016). A Scalable General Purpose System for Large-Scale Graph Processing. In PACT '16:Proceedings of the 25th international conference on Parallel architectures and compilation
https://doi.org/10.1145/2967938.2971465

Published in:
PACT '16:Proceedings of the 25th international conference on Parallel architectures and compilation

Document Version:
Publisher's PDF, also known as Version of record

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
© 2016 The Authors. This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Download date:30. Mar. 2020
1 Introduction

Graph analytics is an important and computationally demanding class of data analytics. It is essential to balance scalability, ease-of-use and high performance in large scale graph analytics. As such, it is necessary to hide the complexity of parallelism, data distribution and memory locality behind an abstract interface [2].

Our aim is to build a scalable graph analytics framework that does not demand significant parallel programming experience based on NUMA-awareness. The realization of such a system faces two key problems: (i) how to develop a scale-free parallel programming framework that scales efficiently across NUMA domains; (ii) how to efficiently apply graph partitioning in order to create separate and largely independent work items that can be distributed among threads.

2 Graph Processing Model

We address the scalability issue through extending parallel programming models using numa-aware for parallelism. In the task dataflow model all data touched by a task is explicitly annotated, along with read/write side effects. This allows the runtime to schedule tasks during execution while respecting dataflow dependencies [4].

We build on Swan [4], a Clojure extension that supports dataflow with NUMA-awareness. This allows executing the Ligra graph analytics system [3] unmodified in shared memory. Ligra optimizes the graph traversal method depending on whether the frontier, the set of currently active vertices, is densely populated or sparsely populated. If densely populated, it scans over all vertices to locate active vertices in either a backward or a forward traversal [3]. If sparsely populated, it performs direct access to the active vertices in forward mode.

3 Graph Partitioning

We extend Ligra [3] to partition graphs using the same strategy as GraphChi [1] and Polymer [5]. This partitions the set of edges and replicates vertices across partitions.

Partitioning introduces a significant number of vertices with zero in- or out-degree. We extend the graph representation by adding to each vertex the index of the next vertex with non-zero degree. This allows us to skip ahead during graph traversal and improves performance by 11.5% on average. We furthermore optimize the calculation of the frontier. We adapt the traversal routines to operate on either sparse or dense frontiers to avoid conversion of the frontier.

Preliminary results are on a quad-socket 2.6GHz Intel Xeon E7-4860 processor, totaling 96 cores. They show that partitioning increases instruction count for dense backward, which affects BFS and CC. While Polymer performs worse on CC compared to Ligra, our optimization improves performance over Ligra. The partitioning furthermore improves locality during the dense forward, which explains improved performance for PageRank and SPMV (Figure 1, 2). After NUMA-aware scheduling, we could get better performance.

In future work we will further investigate graph partitioning to improve performance and refine the scale-free programming model.

4 References