Optimization of Asynchronous Graph Processing on GPU with Hybrid Coloring Model

Xuanhua Shi¹, Junling Liang¹, Sheng Di², Bingsheng He³, Hai Jin¹, Lu Lu¹, Zhixiang Wang¹, Xuan Luo¹, Jianlong Zhong³

¹Services Computing Technology and System Lab/Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, China
²Argonne National Laboratory, USA
³School of Computer Engineering, Nanyang Technological University, Singapore
¹{xhshi, junlingliang, hjin, llu, wangzhx123, luoxuan}@hust.edu.cn, ²disheng222@gmail.com, ³{bshe, jzhong}@ntu.edu.sg

Abstract
Modern GPUs have been widely used to accelerate the graph processing for complicated computational problems regarding graph theory. Many parallel graph algorithms adopt the asynchronous computing model to accelerate the iterative convergence. Unfortunately, the consistent asynchronous computing requires locking or the atomic operations, leading to significant penalties/overheads when implemented on GPUs. To this end, coloring algorithm is adopted to separate the vertices with potential updating conflicts, guaranteeing the consistency/correctness of the parallel processing. We propose a light-weight asynchronous processing framework called Frog with a hybrid coloring model. We find that majority of vertices (about 80%) are colored with only a few colors, such that they can be read and updated in a very high degree of parallelism without violating the sequential consistency. Accordingly, our solution will separate the processing of the vertices based on the distribution of colors.

Categories and Subject Descriptors D.1.3 [Programming Techniques]: Concurrent Programming

General Terms Design, Algorithms, Performance

Keywords Graph Processing, Asynchronous Computing, GPGPU

1. Introduction
Modern GPGPUs are often used to accelerate the graph processing algorithms, not only because GPU has many more cores for parallel computing, but also due to much higher memory bandwidth and lower latency. Whereas, there are still some issues in the existing solutions.

• Most existing GPU-accelerated graph frameworks (such as Totem [1], Medusa [2], CuSha [3]) are designed based on the synchronous processing model - Bulk Synchronous Parallel (B-SP) model. Such a model, however, will introduce a huge cost in synchronization especially as the graph size grows significantly, because any message processing must be finished in the previous super-step before moving to the next one.
• In comparison to the synchronous model, there are some asynchronous models that have been proved more efficient in processing graphs, but they are not very suitable for parallel graph processing on GPU. In order to ensure the correct/consistent processing results in the parallel computations, many existing solutions (such as GraphLab [4] and GraphChi [5]) adopt fine-grained locking protocols or update most vertices sequentially for simplicity. Locking policy, however, is unsuitable for GPU-based parallel processing because of the huge cost of the locking operations on GPU.

In our work, we design a lock-free parallel graph processing method named Frog¹ with a graph coloring model. In this model, each pair of adjacent vertices with potential update conflicts will be colored differently. The vertices with the same colors are allowed to be processed simultaneously in parallel, also guaranteeing the sequential consistency of the parallel execution. We observe that a large majority of vertices (roughly 80%) are colored with only a small number of colors (about 20% or less). Based on such a finding, our solution will process the vertices based on their coloring distributions. In particular, we process the vertices with the same colors in a high degree of parallelism on GPU, and process the minority of the vertices that account for majority of colors in a separate super-step.

2. Design Overview
Many graph algorithms have been shown to converge faster when solved asynchronously. Synchronous computation incurs costly performance penalties since all vertices have been updated all the time and the runtime of each phase is determined by the slowest GPU thread. While having a faster convergence, we should ensure the serializability in our asynchronous approach: all parallel executions have an equivalent sequential execution to make sure the computation correctness.

A classic technique to achieve a serializable parallel execution of vertices in a graph is to construct a vertex coloring that assigns a color to each vertex such that no adjacent vertices share the same color. For a data graph with billions of vertices, hundreds of colors can be used to complete the graph coloring. We observe that a large majority of vertices are colored with only a small number of colors. Hence, we need a relaxed graph coloring algorithm to

¹The code can be found at https://github.com/AndrewStallman/Frog.git
Figure 1: Asynchronous execution based on the hybrid coloring

Table 1: Execution time on different algorithms, datasets and frameworks (in milliseconds)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Dataset</th>
<th>Amazon</th>
<th>DBLP</th>
<th>WikiTalk</th>
<th>LiveJournal</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Frog</td>
<td>4.968</td>
<td>3.380</td>
<td>3.532</td>
<td>7.482</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Totem</td>
<td>52.155</td>
<td>53.476</td>
<td>25.923</td>
<td>272.335</td>
<td>15.82X - 36.4X</td>
</tr>
<tr>
<td></td>
<td>CuSha</td>
<td>27.504</td>
<td>23.915</td>
<td>12.311</td>
<td>203.534</td>
<td>-3.9X - 27.2X</td>
</tr>
<tr>
<td>CC</td>
<td>Frog</td>
<td>11.762</td>
<td>10.748</td>
<td>12.986</td>
<td>213.720</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Totem</td>
<td>30.156</td>
<td>52.371</td>
<td>26.275</td>
<td>294.390</td>
<td>1.4X - 4.8X</td>
</tr>
<tr>
<td></td>
<td>CuSha</td>
<td>27.779</td>
<td>24.574</td>
<td>14.120</td>
<td>224.653</td>
<td>1.05X - 2.4X</td>
</tr>
</tbody>
</table>

Table 2: Properties of real-world graphs used in our experiments

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Amazon</th>
<th>DBLP</th>
<th>WikiTalk</th>
<th>LiveJournal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>735,322</td>
<td>986,286</td>
<td>2,394,385</td>
<td>4,847,571</td>
</tr>
<tr>
<td>Edges</td>
<td>3,158,012</td>
<td>6,707,236</td>
<td>5,021,410</td>
<td>68,475,391</td>
</tr>
</tbody>
</table>

Acknowledgments

This work is supported by the NSFC under grants No.61133008 and No. 61370104, Chinese Universities Scientific Fund under No. 2014TS008, the U.S. Department of Energy, Office of Science, under Contract DE-AC02-06CH11357, and Tencent. Bingsheng He and Jianlong Zhong are partly supported by a MoE AcRF Tier 2 Grant (MOE2012-T2-2-067) in Singapore.

References