Forschungsprojekt

Simon König (3344789) st156571@stud.uni-stuttgart.de
Leon Matzner (3315161) @stud.uni-stuttgart.de
Felix Rollbühler (3310069) st154960@stud.uni-stuttgart.de
Jakob Schmid (3341630) @stud.uni-stuttgart.de

***Abstract*—In this paper we will analyze and compare the graph frameworks Galois, Ligra, Polymer, Gemini and Giraph in their parformance. All the frameworks will be tested in shared memory and Galois, Gemini and Giraph are tested on a distributed cluster as well. Furthermore we will give some insight on the complexity of writing custom applications based on these frameworks.**

***Index Terms*—graphs, distributed computing, Galois, Ligra, Polymer, Giraph, Gluon**

I. Introduction

In recent years graph sizes have increased significantly, thus performance and memory efficiency of the graph analysis applications is now more important than ever.

This paper makes the following contributions:

- Comparison of some of the most widely used graph based calculation frameworks
-
-
-

II. Overview of the framworks

A. Galois and Gluon

Galois[1] is a general purpose library designed for parallel programming. The Galois system supports fine grain tasks, allows for autonomous, speculative execution of these tasks and grants control over the task scheduling policies to the application. It also simplifies the implementation of parallel applications by providing an implicitly parallel unordered-set iterators.

For graph analytics purposes a topology aware work stealing scheduler, a priority scheduler and a library of scalable data structures have been implemented. Galois includes applications for many graph analytics problems, among these are single-source shorthest-paths (sssp), breath-first-search (bfs) and pagerank. For most of these applications Galois offers several different algorithms to perform these analytics problems and many setting options like the amount of threads used or policies for splitting the graph. All of these applications can be executed in shared memory systems and, due to the Gluon integration, with a few modifications in a distributed environment.

Gluon[2] is a framework written by the Galois team as a middleware to write graph analysis applications for distributed systems. It reduces the communication overhead needed in distributed environments by exploiting structural and temporal invariants.

The code of Gluon is embedded in Galois. It is possible to integrate Gluon in other frameworks too, which the Galois team showed in their paper[2].

B. Ligra

C. Polymer

Polymer is very similar to Ligra, in fact Polymer inherits the programming interface from Ligra[3].

Polymer aims to minimize both random and remote memory accesses by implementing NUMA- and graph-aware data layout and memory access strategies. Specifically, Polymer co-locates graph data and the computation within NUMA-nodes to reduce remote memory accesses. For example, Polymer eliminates remote accesses by letting threads allocate memory in their local memory node for graph topology data like vertices and edges that are only accessed by one thread. Application-defined data with static memory locations which gets dynamically updated during computation is allocated with virtual addresses that make for a seamless cross-node data access. Other mutable runtime states (e.g. active vertices) might be dynamically allocated in each iteration. This data is allocated in a distributed way but only accessed through a global lookup table.

D. Gemini

Gemini[6] is a framework for parallel graph processing. In comparrison to the other NUMA-aware frameworks we discuss here Gemini does's not support shared memory parallel processing, it is completely message based using MPI. It seems to be very lightweight and has only the very basically needed functionallity implemented, which leeds to clear source code and small binarys. A basic API hides data and computation distribution details from the user writing applications. It comes with five algorithms already implemented included the three we are testing in this paper. While setting it up and even while benchmarking we had various problems with bugs in the source code, like non zero terminated strings and missing return statements. While debugging we forked the origional repository and made our changes there. DODO link repository.

E. Giraph

Giraph

III. AND

A. An overview of some graph formats

A rather big portion of our time was invested in figuring out which graph framework requires which graph formats. We thus decided to give an overview over all the formats we encountered, with explanation on how they represent the graph.

Additionally, to make life in the future a little bit easier, we wrote multiple tools to convert graphs acquired from Snap or Konect to the required formats. Additional information on this is available in the section Supplementary Data at the end.

1) AdjacencyList: The AdjacencyList and WeightedAdjacencyList formats[4] are used by Ligra and Polymer. They represent the directed edges of a graph as a number of offsets that point to a set of target nodes in the file. First the file contains the number of vertices n and edges m, followed by an offset for each vertex. This offset specifies at what point in the following list of numbers the information for a node begins. Lastly the file format contains a list of target nodes. The numbers are all separated by newlines.

$$
\begin{aligned}
&n\\
&m\\
&o_1\\
&o_2\\
&\vdots\\
&o_n\\
&t_1\\
&t_2\\
&\vdots\\
&t_m
\end{aligned}
$$

The offsets $o_i = k$ and $o_{i+1} = k + j$ mean that vertex i has j outgoing edges, these edges are

$$(i, t_k), (i, t_{k+1}), \ldots, (i, t_{k+j-1})$$

For the WeightedAdjacencyList format, the weights are just appended to the end of the file in the same order as the edges.

2) EdgeList: The EdgeList format is probably the easiest to understand and is one of the most commonly used in the online graph repositories. The directed eges $(s_1, t_1), (s_2, t_2), \ldots$ or $(s_1, t_1, w_1), (s_2, t_2, w_2), \ldots$ are represented in the following way.

$$
\begin{aligned}
&s_1 \quad t_1 \quad w_1\\
&s_2 \quad t_2 \quad w_m\\
&\quad\vdots\\
&s_m \quad t_m \quad w_m
\end{aligned}
$$

The weights are optional, everything is ASCI encoded and the inline delimiter is a variable amount of any whitespace.

3) Binary EdgeList: The binary EdgeList format is used by Gemini. Finding information on this format required reverse engineering of the Gemini code.

We found that Gemini requires the following input format

$$s_1 t_1 w_1 s_2 t_2 w_2 \ldots$$

where s_i, t_i have `uint32` data type and the optional weights are `float32`. Gemini will derive the number of edges from the file size, so there is no file header or anything similar allowed.

4) Giraph's numerous I/O formats:

IV. TESTING METHODS

A. Hardware and Software

For testing all systems we used 5 machines with 96 CPU cores each (48 physical) and 256 GB of RAM. One of those machines only had 128 GB of RAM, this one was only used as part of the calculation cluster for the distributed systems. All servers were running Ubuntu 19.x. Setup of each framework was performed according to our provided installation guides.

B. Benchmark setup

We measured the total runtime of each process as well as the actual calculation times. The console output of each frameworks was used as an indicator for when loading the graph data was finished. This proved to be more difficult than expected in some cases.

Each test case consisting of graph, framework and in some cases even the algorithm was run 10 times, allowing us to smooth slight variations in the measured times. This also helps to reduce the error we introduced by measuring time through the console log.

All benchmarks were initiated by our benchmark script that is available in our repository.

We will compare both the computation times as well es the execution times. This way we compare how much setup time each framework has.

Hier sollten wir auch evtl Vergleichsgrundlagen für Aufsetzen und Apps schreiben einführen, wenn wir das mit aufnehmen.

V. RESULTS

A. Komplexität Aufsetzen

Systeme wie distr. Giraph bei denen man erstmal ewig suchen muss bis man überhaupt einen Guide findet sollten hier schlechter abschneiden. *vorausgesetzt, wir nehmen das hier überhaupt mit auf..*

B. Komplexität eigene Apps schreiben

Schwierig objektiv zu vergleichen.

C. Pure Performance-Ergebnisse

D. Ergebnisse Calc time

der offensichtliche, wichtige Vergleich

E. Ergebnisse exec time

hier erhoffe ich mir einen Vergleich der Ladezeiten und erwarte, dass Systeme wie Giraph, die erstmal auf irgendwas warten schlecht abschneiden. Aber vielleicht ist auch die setup time bei gleichen frameworks zwischen verteilt und shared memory ganz interessant zu vergleichen.

VI. Discussion

VII. Conclusion

The conclusion goes here.

Acknowledgment

We are using the graph frameworks Galois [1], Ligra [5], Polymer [3], Gemini [6] as well as Apache Giraph [7].

Also we use Gluon [2] for the distributed setups.

Gemini [6]

Supplementary Data

We have written a number of conversion tools and installation guides to help users or develeopers with the use of the tested frameworks.

Everything can be retrieved on our GitHub repository http://www.github.com/serengti/Forschungsprojekt.

For each Framework, there is a

The graphs are downloaded from [8].

References

[1] D. Nguyen, A. Lenharth, and K. Pingali, "A lightweight infrastructure for graph analytics," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 456–471. [Online]. Available: https://doi.org/10.1145/2517349.2522739

[2] R. Dathathri, G. Gill, L. Hoang, H.-V. Dang, A. Brooks, N. Dryden, M. Snir, and K. Pingali, "Gluon: A communication-optimizing substrate for distributed heterogeneous graph analytics," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 752–768. [Online]. Available: https://doi.org/10.1145/3192366.3192404

[3] K. Zhang, R. Chen, and H. Chen, "Numa-aware graph-structured analytics," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 183–193. [Online]. Available: https://doi.org/10.1145/2688500.2688507

[4] J. Shun, G. Blelloch, J. Fineman, P. Gibbons, A. Kyrola, K. Tangwonsan, and H. V. Simhadri. (2020, Jun.) Problem Based Benchmark Suite. graphIO.html. [Online]. Available: http://www.cs.cmu.edu/~pbbs/benchmarks/

[5] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 135–146. [Online]. Available: https://doi.org/10.1145/2442516.2442530

[6] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 301–316. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/zhu

[7] A. S. Foundation. (2020, Jun.) Apache Giraph. [Online]. Available: https://giraph.apache.org

[8] J. Kunegis, "Konect: the koblenz network collection," 05 2013, pp. 1343–1350.