Comparing Platform Approaches to Common Graph Partitioning Algorithms

Gregory Siegfried, North Carolina Central University
gsiegfri@eagles.nccu.edu
Dr. Alade O. Tokuta, North Carolina Central University
atokuta@nccu.edu

Abstract:

In summary, there were six frameworks chosen for this project using the same set of graph processing kernels on one of three different platforms. Hardware was chosen to provide a single example used across all tests, although it might not have been the exact same computer between tests. Frog (FR)[1], GraphIt (GI)[2], Gunrock (GR)[3], Hygraph (HY)[4], Polymer (PM)[5], and Totem (TM)[6] are the provided frameworks for testing. Information regarding frameworks will be found under the platform used, while graphs and hardware can be found in their respective sections. Kernels used were breadth first search (BFS), connected components (CC), pagerank (PR), and single source shortest path (SSSP). Platforms used were CPU only, GPU only, and a hybrid approach depending on the graph and task.

I refer the reader to the results section for how everything played out in testing, but it appears that FR and GI performed better with most graphs and kernels. A noticeable runner up would be GR, and TM had a clean sweep with PR, but further testing will be required for validity. HY would not finish on any graph and technical support provided could not find a resolution. More information can be found reading further, as well as reading the research papers of the respective frameworks.

Introduction:

There are a few dozen or so graph partitioning frameworks to choose from. Most have compared themselves to the previous “best of ” group, and displayed their rankings accordingly. Usually with the newest framework performing the best. A reason for my performing these tests was to apply a standardized hardware set, a set of graphs, and different platforms to see if there are any advantages across a small spectrum of previous results. The small spectrum being the four kernels, or algorithms chosen that the frameworks have in common.

When I am referring to different platforms, I am speaking in regards to how each framework is using either the CPU, GPU/’s, or a hybrid approach. In this project, the frameworks chosen for each platform were the few that performed the best, and had comparable kernels. More information about the platforms can be found in the hardware section and their respective section in the paper.

The choice of graphs was another item to consider for this project, and a variety of available graphs (or sometimes can be referred to as maps) from an online repository were chosen for a variety of traits. In all there were sixteen candidates with chosen characteristics involving the number of connected components, the overall number of vertices and edges, to the largest in pure file size. More information on the graphs chosen can be found in the Graphs section as well as the Appendix.

Hardware used in this project was provided through XSEDE and the Pittsburgh Supercomputing Center (PSC)[7], and will help establish a baseline for current and future hardware capabilities. The hardware was chosen to keep the same set of hardware being used across all tests, and does not reflect the overall availability of the services at the PSC, or the capability of the frameworks presented.

Finally, the last part of the project was the choice of kernels being Breadth First Search (BFS), Connected Components (CC), Page Rank (PR), and Single Source Shortest Path (SSSP). Bellman-Ford algorithm was used in place of listed SSSP algorithms for any framework that differed. The four selected

kernels do not represent all of the options of each framework, but were most common among them all and provide a variety for consideration.

Hardware:

PSC has a list on their respective website detailing more information about hardware availability. The provided components for testing from that website are as follows:

CPUs: 2x Intel Haswell (E5-2695 v3) CPUs; 14 cores/CPU; 2 threads/core; 2.3 - 3.3 GHz
RAM: 128GB, DDR4-2133
Cache: 35MB SmartCache
Node-local Storage: 2 HDDs, 4TB each
Server: HPE Apollo 2000
GPUs: 2x NVIDIA Tesla K80; (1x K80 = 2x 12GB VRAM, 5GHz GDDR5) (equivalent of 4 gpu's)

The NVIDIA Tesla K80 was chosen in part due to its compatibility with more of the frameworks, but mostly for its use on the exact same CPU only configuration that is available from PSC Bridges. This would help ensure a standard across all three platforms.

CPU:

GraphIt (GI) - "a new domain specific language for graph computations that generates fast implementations for algorithms with different performance characteristics running on graphs with different sizes and structures"[2]. Compiled with Python 2.7 and g++ 5.3.0 with OpenMP as stated from the GitHub readme. The input file format used for GI was the EL and WEL. Results for GI provide each graph a total of ten iterations with each iteration providing an elapsed time. BFS and CC were a strength of GI, with it also doing well on the larger file sizes.

Polymer (PM) - "a NUMA-aware Graph-structured Analytics Framework written in C++"[5]. Non Uniform Memory Access(NUMA). Compiled and run with g++ version 5.3.0 with CILK+ as stated on the GitHub readme. Input file format for PM was in the form of an adjacency list. PM did run well on SSSP with the graphs that were about 1Gb in size.

GPU:

Frog (FR) - "asynchronous Graph Processing on GPU with Hybrid Coloring Model"[1]. They used a modified Pareto principle about coloring algorithms, also known as the 80-20 rule. Compiled with gcc 5.3.0 and cuda 8.0 as mentioned from their GitHub readme. I used the Process-Large files from my GitHub folder during compiling to allow the use of large file sizes. The framework recommends tuning the buffer size to the hardware that you are running, and for a default value I used 1024*1024 or 1Gb blocks. The input file format required a preprocessing step in the form of converting the matrix market file type into a binary version. FR performed the best when the file size was about 320Mb or less in the category of BFS and SSSP. Where FR performed the worst was with large file sizes, which could be attributed to my choice of buffer size. Another notable feature is that FR ran every test regardless of graph size.

Gunrock (GR) - "is to deliver the performance of customized, complex GPU hardwired graph primitives with a high level model programming model that allows programmers to quickly develop new graph primitives."[3]. Their chief problem was how to manage irregularity in work distribution. Compiled

with cuda 7.5 and gcc 4.8 as mentioned from my readme. The input format for GR is in the matrix market format which was how the files were downloaded. Results for GR were good overall, and possibly could have been improved if more recent versions of cuda or gcc were used during compilation. Also it should be mentioned that all parameters were set by default.

Hybrid:

HyGraph (HY) - "a novel graph-processing systems for hybrid platforms which delivers performance using CPU and GPU concurrently. It's core feature is a specialized data structure which enables dynamic scheduling of jobs onto both CPU and the GPU's, thus supersedes the need for static workload distribution, provides load balancing, and minimizes inter-process communication overhead by overlapping computation and communication."[4]. Compiled with cuda 8.0 and gcc 5.3.0 as well as the settings mentioned on their GitHub. Unfortunately neither version was successful in finishing any tests. Help was provided from the PSC Bridges help team and a solution was not found. Therefor HY is not found with any results. The input format for HY is the EL type, but can also use a binary version if you want to preprocess the file to further reduce loading time.

Totem (TM)- "a processing engine that provides a convenient environment to implement graph algorithms on hybrid platforms; we show that further significant performance gains can be extracted using partitioning strategies that aim to produce partitions that each matches the strengths of the processing element it is allocated to, and finally, we demonstrate the performance advantages of the hybrid system through comprehensive evaluation that uses real and synthetic scale-free workloads (as large as 16 billion edges), multiple graph algorithms that stress the system in various ways, and a variety of hardware configurations."[6]. Compiling TM with gcc 5.3 and cuda 8.0 as mentioned on their GitHub. The input file format needs to be in the *.totem format; which is a sorted EL with the number of nodes, edges, and whether the graph is directed or undirected, commented at the beginning of the file. The performance of TM was evident in the PR results, as no other framework was able to outperform it. More research will need to be done on why it was so efficient with that particular kernel.

Graphs:

I collected all the graphs used from the SuiteSparse Matrix Collection formerly the University of Florida Sparse Matrix Collection. All of the graphs were originally downloaded in the matrix market format, and then converted into the edge list format (EL). All of the weighted versions of the graphs used, started in the EL format and were converted to the weighted edge list (WEL) format with the use of Graph Algorithm Processing Benchmark Suite (GAPBS)[8]. From there the graphs were converted into the file type specified for processing with each framework resulting in a use of 576 Gb to store them all. One file (in bold) in particular was so large that it crashed a few of the programs and was not included in the results, but mentioned here for completeness.

The following table is the order of graphs used and some information about them. More information can be found in Appendix table 1 for the specific web address of the file, and table 2 for size comparisons of file types. The second column will show weather or not the graph is directed or undirected, as well as being symmetric or not. Considering, all of the graphs are square, the number of rows listed can also represent the number of columns. Next to that we have the number of nonzeros in the graph, which represents how much of the graph is filled with data and not zeros. We then have the number of strongly connected components listed, which varies based on where the information from the graph came from. The road maps and large objects comprise the smaller number of connected

components, while the web connections and journal citations are comprised of the large set of connected components. Lastly we have the file size in EL format to give as a comparison.

Name	Directed / Symmetric	# of Rows	# of Nonzeros	# of Strongly Connected Components	File Size
Gh001 - Webbase-2001	D / Non-S	118,142,155	1,019,903,190	41,126,852	3.8Gb
Gh002 - Europe_Osm	UnD / Sym	50,912,018	108,109,320	1	1.89Gb
Gh003 - Sk-2005	**D / Non-S**	**50,636,154**	**1,949,412,601**	**8,815,057**	**24.5Gb**
Gh004 - It-2004	D / Non-S	41,291,594	1,150,725,436	6,753,961	19.9Gb
Gh005 - Uk-2005	D / Non-S	39,459,925	936,364,282	5,811,041	11Gb
Gh006 - Road_usa	UnD / Sym	23,947,347	57,708,624	1	.984Gb
Gh007 - Arabic-2005	D / Non-S	22,744,080	639,999,458	4,000,414	10.8Gb
Gh008 - Hugebubbles-00020	UnD / Sym	21,198,119	63,580,358	1	.538Gb
Gh009 - Rgg_n_2_24_s0	UnD / Sym	16,777,216	265,114,400	2	2.2Gb
Gh010 - Delaunay_n24	UnD / Sym	16,777,216	100,663,202	1	.839Gb
Gh011 - Hugetrace-00020	UnD / Sym	16,002,413	47,997,626	1	.398Gb
Gh012 - Hugetrace-00010	UnD / Sym	12,057,441	36,164,358	1	.292Gb
Gh013 - Wb-edu	D / Non-S	9,845,725	57,156,537	4,269,022	.903Gb
Gh014 - Ljournal-2008	D / Non-S	5,363,260	79,023,142	1,119,171	1.2Gb
Gh015 - Cage15	D / Non-S	5,154,859	99,199,551	1	1.4Gb
Gh016 - Soc-LiveJournal1	D / Non-S	4,847,571	68,993,773	971,232	1.0Gb

Results:

Results provided are the times returned from running tests on each graph, and in some cases averaged with other results if performed more than once. Any blank spaces occur when frameworks fail

to run a particular map. In the case of HY, it failed to complete every map. The failure most commonly reported dealt with exceeding memory allocation with CUDA. As you can see in appendix table 4 for example, failed for “kernel_process_block: too many resources requested for launch”. See also table 6 for error reporting example from running on slurm.

Time listed in milliseconds.

	BFS FROG	GRAPHIT	GUNROCK	HYGRAPH	POLYMER	TOTEM
gh001	179.42	24.21				497.81
gh002	10.27	1590.88	1751.53		2468.47	103011.25
gh004	4128.62	441.86				2840.35
gh005	10837.9	9.88	210.69			235.46
gh006	2.83	664.93	703.99		41	14005.84
gh007	1896.58	4.48	0.54			205.3
gh008	2.91	4.43	570.37		89.14	52
gh009	9.22	3.39	454		711.38	7470.56
gh010	4.16	3.77	320.36		469417.2	179.69
gh011	2.26	3.07	524.55		50.58	55.3
gh012	1.69	2.4	465.29		70.84	39.32
gh013	854.7	63.98	0.3			60.53
gh014	4.85	28.85	61.08		3512.68	72.92
gh015	319.9	94.64	68.69		346.26	52.47
gh016	81.29	26.03	60.07		4533.49	36.99

Time listed in milliseconds.

	CC FROG	GRAPHIT	GUNROCK	HYGRAPH	POLYMER	TOTEM
gh001	40,105.15	1,824.77				11,809.74
gh002	5,436.49	10,670.70	416.91		1,556.32	347,478.36
gh004	1,585.71	4,579.53				2,458.30
gh005	2,292.22	1,603.67				3,432.60
gh006	3,718.13	34,140.70	228.74		476.95	68,180.00
gh007	1,265.14	953.37	2,676.59			1,119.11
gh008	5,773.74	42.43	253.31		128.00	182.65
gh009	18,110.70	30.57	349.33		1,078.03	171.02
gh010	3,837.63	40.52	250.62		6,460.88	149.32
gh011	3,881.18	28.70	177.43		83.87	139.03
gh012	2,618.97	22.78	126.54		59.02	103.26
gh013	182.89	123.34	749.77			1,554.83
gh014	264.72	120.54	261.54		503.99	224.17
gh015	143.31	101.40	95.17		728.94	215.47
gh016	166.57	159.49	88.07		1,579.25	205.90

Time listed in milliseconds.

	PR FROG	GRAPHIT	GUNROCK	HYGRAPH	POLYMER	TOTEM
gh001	14,566.51	2,956.08				612.98
gh002	1,527.43	1,508.93	3,945.90		4,780.00	911.42
gh004	13,504.58	4,179.59				985.09
gh005	11,612.84	1,885.43				713.59
gh006	669.05	714.87	2,148.56		1,890.00	432.36
gh007	7,542.18	1,409.74	3,461.84		208,000.00	537.45
gh008	748.52	743.21	2,016.79		2,080.00	360.84
gh009	2,090.71	616.02	1,356.39		8,310.00	360.63
gh010	955.03	796.71	2,665.30		4,750.00	301.97
gh011	563.19	522.12	1,419.89		1,510.00	270.32
gh012	426.05	387.66	1,049.51		1,170.00	205.01
gh013	865.87	355.55	495.39			174.93
gh014	1,088.73	721.23	803.99		5,260.00	133.86
gh015	1,330.57	297.39	364.22		7,150.00	141.65
gh016	999.78	947.37			13,500.00	204.93

Time listed in milliseconds.

	SSSP FROG	GRAPHIT	GUNROCK	HYGRAPH	POLYMER	TOTEM
gh001	120.52	28.90	1.52		106.49	1,354.92
gh002	10.27	27,281.27	8,117.59		3.47	504,486.20
gh004	3,391.68	5,230.32				1,295.42
gh005	10,843.76	8.55	5,446.68		44.95	2,677.05
gh006	2.75	26,712.72	8,101.97		10.21	62,945.60
gh007	1,886.19	6.02	0.35		26.85	925.16
gh008	2.84	4.24	17,864.25		12.34	63.00
gh009	9.19	3.47	106,588.57		27.03	25,055.39
gh010	4.08	3.87	14,278.50		3.16	149.72
gh011	2.15	2.77	13,835.40		17.88	83.86
gh012	1.62	1.74	9,450.34		8.61	45.00
gh013	657.05	178.99	0.29		1.36	33.65
gh014	4.76	384.92	432.81		3.54	219.26
gh015	242.22	387.57	97.20		7.66	181.84
gh016	213.62	507.71	486.87		15.98	346.57

The final results are:

1 = Frog
2=Graphit
3=Gunrock
4=Hygraph
5=Polymer
6=Totem

	BFS	CC	PR	SSSP	Framework
gh001	2	2	6	3	2
gh002	1	3	6	5	
gh004	2	3	6	6	6
gh005	2	3	6	2	2
gh006	1	3	6	1	1
gh007	3	2	6	3	3
gh008	1	2	6	1	1
gh009	2	2	6	2	2
gh010	2	2	6	5	2
gh011	1	2	6	1	1
gh012	1	2	6	1	1
gh013	3	2	6	3	3
gh014	1	2	6	5	
gh015	6	3	6	5	6
gh016	2	3	6	5	
Top	1-2	2	6	5	
Runner Up	3	3		1	

In this last image with the results, the last column refers to the framework that showed the best results with a map, if one existed.

Conclusion:

Overall, I tried to do my best to keep some standard parameters so that testing would yield a more accurate comparison between frameworks and their respective kernels. Some frameworks were easier to work with in that regard, and some were not.

Finding a way to have a more accurate comparison proved challenging, with some frameworks needing a converted file format, while others performed the task at run time. Knowing that the timing has been left up to the program, could mean some results are skewed.

More research would need to be done to figure out exactly which framework was being precise. Further time spent researching the GABPS and Graph Algorithm Iron Law (GAIL)[9], would also be helpful in understanding the benchmarking process.

Also, all hard data and information can be found at my github account, where you can address any comments or questions about the project.

github.com/siegfriedgreg/ZoeBug/tree/master/Project GPA

Also, a special thanks to the help provided from the Pittsburgh Supercomputing Center, Extreme Science and Engineering Discovery Environment, and Dr. Scott Beamer. The learning was incredible!

References:

[1] Xuanhua Shi, Xuan Luo, Junling Liang, Peng Zhao, Sheng Di, Bingsheng He, Hai Jin, "Frog: Asynchronous Graph Processing on GPU with Hybrid Coloring Model", IEEE Transactions on Knowledge and Data Engineering, 30 (1): 29-42, 2018
doi: 10.1109/TKDE.2017.2745562

[2] Yunming Zhang, Mengjiao Yang, Riyadh Baghdadi, Shoaib Kamil, Julian Shun, and Saman Amarasinghe. 2018. GraphIt: A High-Performance Graph DSL.Proc. ACM Program. Lang.2, OOPSLA, Article 121 (November 2018),30 pages. https://doi.org/10.1145/3276491

[3] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D. Owens. Gunrock: A High-Performance Graph Processing Library on the GPU. In Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '16, pages 11:1–11:12, March 2016. Distinguished Paper. http://dx.doi.org/10.1145/2851141.2851145

[4] Heldens S., Varbanescu A. L., Iosup A., HyGraph: Fast Graph Processing on Hybrid CPU-GPU Platforms by Dynamic Load-Balancing, 2016. Doi: 10.1109/IA3.2016.16

[5] Kaiyuan Zhang, Rong Chen and Haibo Chen. Polymer: NUMA-aware Graph-structured Analytics. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'15), Bay Area, California, USA, February, 2015. http://dx.doi.org/10.1145/2688500.2688507

[6] A. Gharaibeh, L. Beltrão Costa, E. Santos-Neto, and M. Ripeanu. Efficient Large-Scale Graph Processing on Hybrid CPU and GPU Systems. In CoRR, 2013. https://arxiv.org/pdf/1312.3018.pdf

[7] This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. Specifically, it used the Bridges

system, which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC).

[8] The GAP Benchmark Suite, Scott Beamer, Krste Asanović, and David Patterson, arXiv:1508.03619 [cs.DC], 2015.

[9] Scott Beamer , Krste Asanović , David Patterson, GAIL: the graph algorithm iron law, Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms, p.1-4, November 15-15, 2015, Austin, Texas [doi>10.1145/2833179.2833187]

Appendix:

Table 1:

A list of the website links for all of the graphs used in this project. More information on graph structure and composition can be found there.

Gh001 - https://sparse.tamu.edu/LAW/webbase-2001
Gh002 - https://sparse.tamu.edu/DIMACS10/europe_osm
Gh003 - https://sparse.tamu.edu/LAW/sk-2005
Gh004 - https://sparse.tamu.edu/LAW/it-2004
Gh005 - https://sparse.tamu.edu/LAW/uk-2005
Gh006 - https://sparse.tamu.edu/DIMACS10/road_usa
Gh007 - https://sparse.tamu.edu/LAW/arabic-2005
Gh008 - https://sparse.tamu.edu/DIMACS10/hugebubbles-00020
Gh009 - https://sparse.tamu.edu/DIMACS10/rgg_n_2_24_s0
Gh010 - https://sparse.tamu.edu/DIMACS10/delaunay_n24
Gh011 - https://sparse.tamu.edu/DIMACS10/hugetrace-00020
Gh012 - https://sparse.tamu.edu/DIMACS10/hugetrace-00010
Gh013 - https://sparse.tamu.edu/Gleich/wb-edu
Gh014 - https://sparse.tamu.edu/LAW/ljournal-2008
Gh015 - https://sparse.tamu.edu/vanHeukelum/cage15
Gh016 - https://sparse.tamu.edu/SNAP/soc-LiveJournal1

Table 2:

All the used maps with their file size listed. The left column, from top to bottom, refers to: Adjacency List, Edge List, Matrix Market, Binary Matrix Market, Totem file type.

In Bytes.

	gh001	gh002	gh004	gh005	gh006	gh007	gh008
Adjacency	20,338,002,852	951,938,529	22,737,538,124	17,583,320,434	494,808,832	11,841,443,753	540,212,173
Edge List	3,850,992,348	1,897,981,740	19,978,659,994	11,082,967,111	984,970,306	10,893,830,336	538,897,997
MTX	18,497,838,955	949,001,304	20,241,588,318	16,451,551,195	492,496,838	10,893,833,669	538,907,930
MTX.BIN	8,631,794,160	636,085,372	9,370,969,884	7,648,753,976	326,623,904	5,210,972,004	339,113,928
TOTEM	3,848,394,327	1,897,981,785	19,978,660,038	11,082,916,303	984,970,350	10,893,830,077	538,898,020
MAX	20,338,002,852	1,897,981,785	22,737,538,124	17,583,320,434	984,970,350	11,841,443,753	540,212,173
AVG	11,033,404,528	1,266,597,746	18,461,483,272	12,769,901,804	656,774,046	9,946,781,968	499,206,010
MIN	3,848,394,327	636,085,372	9,370,969,884	7,648,753,976	326,623,904	5,210,972,004	339,113,928

	gh009	gh010	gh011	gh012	gh013	gh014	gh015	gh016
Adjacency	2,416,696,192	878,862,062	404,807,686	298,943,738	963,975,564	1,327,137,505	1,866,329,851	3,076,277,487
Edge List	2,210,494,279	839,303,889	398,650,785	292,150,178	903,100,945	1,233,480,013	1,475,669,040	1,003,771,408
MTX	2,210,504,057	839,313,191	398,660,712	292,160,105	903,101,884	1,233,483,873	2,627,728,707	1,011,609,587
MTX.BIN	1,127,566,484	469,761,692	256,000,176	192,887,216	496,635,216	653,638,196	814,215,864	571,340,488
TOTEM	2,210,494,243	839,303,893	398,650,808	292,150,201	902,331,898	1,233,479,841	1,475,669,081	1,003,760,104
MAX	2,416,696,192	878,862,062	404,807,686	298,943,738	963,975,564	1,327,137,505	2,627,728,707	3,076,277,487
AVG	2,035,151,051	773,308,945	371,354,033	273,658,288	833,829,101	1,136,243,886	1,651,922,509	1,333,351,815
MIN	1,127,566,484	469,761,692	256,000,176	192,887,216	496,635,216	653,638,196	814,215,864	571,340,488

Table 3:

A list of the vertice and edge counts.

	gh001	gh002	gh004	gh005	gh006
Vertices	118,142,155	50,912,018	41,291,594	39,459,925	23,947,347
Edges	1,019,903,190	54,054,660	1,150,725,436	936,364,282	28,854,312

	gh007	gh008	gh009	gh010	gh011
Vertices	22,744,080	21,198,119	16,777,216	16,777,216	16,002,413
Edges	639,999,458	31,790,179	132,557,200	50,331,601	23,998,813

	gh012	gh013	gh014	gh015	gh016
Vertices	12,057,441	9,845,725	5,363,260	5,154,859	4,847,571
Edges	18,082,179	57,156,537	79,023,142	99,199,551	68,993,773

Table 4:

An example file of a failed test run by Hygraph.

```
[   0.00000] reading from file: ../../maps/gh012.el
[   0.01926] reading graph from file '../../maps/gh012.el'
[   0.36429] read 1000000 lines (4.88%)
[   0.62222] read 2000000 lines (9.84%)
[   1.00307] read 3000000 lines (15.31%)
[   1.23965] read 4000000 lines (20.80%)
[   1.51216] read 5000000 lines (26.29%)
[   2.05847] read 6000000 lines (31.79%)
[   2.29932] read 7000000 lines (37.28%)
[   2.53274] read 8000000 lines (42.77%)
[   2.78185] read 9000000 lines (48.27%)
[   3.01726] read 10000000 lines (53.77%)
[   3.25151] read 11000000 lines (59.26%)
[   3.52037] read 12000000 lines (64.77%)
[   4.20647] read 13000000 lines (70.28%)
[   4.41124] read 14000000 lines (75.79%)
```

```
[   4.62373] read 15000000 lines (81.34%)
[   4.86971] read 16000000 lines (87.17%)
[   5.16710] read 17000000 lines (93.33%)
[   5.37370] read 18000000 lines (99.49%)
[   5.39692] done reading file, found 12057441 vertices and 18082179 edges
[   5.74177] executing algorithm: cc
[   6.47910] removed 0 edges
[  17.16377] initializing block-format graph, found 12057441 vertices, 36164358 edges, 1472 blocks
[  17.16389] copying vertex values
[  17.49627] copying edges
[  17.65846] sorting edges...
[  19.50830] finished sorting edges
[  19.50839] create blocks...
[  20.31676] found device Tesla K80 (compute capability 3.7, 13 SMs, 11.17 GB)
[  20.31686] initializing device
[  20.31689] allocating device memory
[  20.61296] finished initialization
[  20.61777] copying vertices state to device
terminate called after throwing an instance of 'hygraph::cuda_exception'
  what():  kernel_process_block: too many resources requested for launch
```

Table 5:

Hygraph run script for the BFS portion. Complete versions of the rest can be found on my github account.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p GPU
#SBATCH -t 04:00:00
#SBATCH --gres=gpu:k80:4
#SBATCH --mail-type=ALL

set -x

echo "Hygraph"

echo "BFS test"
./main ../../maps/gh001.el bfs &> bfs_gh001.txt
./main ../../maps/gh002.el bfs &> bfs_gh002.txt
./main ../../maps/gh004.el bfs &> bfs_gh004.txt
./main ../../maps/gh005.el bfs &> bfs_gh005.txt
./main ../../maps/gh006.el bfs &> bfs_gh006.txt
./main ../../maps/gh007.el bfs &> bfs_gh007.txt
./main ../../maps/gh008.el bfs &> bfs_gh008.txt
./main ../../maps/gh009.el bfs &> bfs_gh009.txt
./main ../../maps/gh010.el bfs &> bfs_gh010.txt
./main ../../maps/gh011.el bfs &> bfs_gh011.txt
./main ../../maps/gh012.el bfs &> bfs_gh012.txt
./main ../../maps/gh013.el bfs &> bfs_gh013.txt
./main ../../maps/gh014.el bfs &> bfs_gh014.txt
./main ../../maps/gh015.el bfs &> bfs_gh015.txt
```

```
./main ../../maps/gh016.el bfs &> bfs_gh016.txt
```

Table 6:

Hygraph slurm results for the BFS section of Table 5.

```
+ echo Hygraph
Hygraph
+ echo 'BFS test'
BFS test
+ ./main ../../maps/gh001.el bfs
/var/slurmd/job4234796/slurm_script: line 13:  3879 Aborted                 ./main ../../maps/gh001.el bfs
&>bfs_gh001.txt
+ ./main ../../maps/gh002.el bfs
/var/slurmd/job4234796/slurm_script: line 14:  4341 Aborted                 ./main ../../maps/gh002.el bfs
&>bfs_gh002.txt
+ ./main ../../maps/gh004.el bfs
/var/slurmd/job4234796/slurm_script: line 15:  4413 Aborted                 ./main ../../maps/gh004.el bfs
&>bfs_gh004.txt
+ ./main ../../maps/gh005.el bfs
/var/slurmd/job4234796/slurm_script: line 16:  4613 Aborted                 ./main ../../maps/gh005.el bfs
&>bfs_gh005.txt
+ ./main ../../maps/gh006.el bfs
/var/slurmd/job4234796/slurm_script: line 17:  5202 Aborted                 ./main ../../maps/gh006.el bfs
&>bfs_gh006.txt
+ ./main ../../maps/gh007.el bfs
/var/slurmd/job4234796/slurm_script: line 18:  5262 Aborted                 ./main ../../maps/gh007.el bfs
&>bfs_gh007.txt
+ ./main ../../maps/gh008.el bfs
/var/slurmd/job4234796/slurm_script: line 19:  5396 Aborted                 ./main ../../maps/gh008.el bfs
&>bfs_gh008.txt
+ ./main ../../maps/gh009.el bfs
/var/slurmd/job4234796/slurm_script: line 20:  5459 Aborted                 ./main ../../maps/gh009.el bfs
&>bfs_gh009.txt
+ ./main ../../maps/gh010.el bfs
/var/slurmd/job4234796/slurm_script: line 21:  5521 Aborted                 ./main ../../maps/gh010.el bfs
&>bfs_gh010.txt
+ ./main ../../maps/gh011.el bfs
/var/slurmd/job4234796/slurm_script: line 22:  5582 Aborted                 ./main ../../maps/gh011.el bfs
&>bfs_gh011.txt
+ ./main ../../maps/gh012.el bfs
/var/slurmd/job4234796/slurm_script: line 23:  5640 Aborted                 ./main ../../maps/gh012.el bfs
&>bfs_gh012.txt
+ ./main ../../maps/gh013.el bfs
/var/slurmd/job4234796/slurm_script: line 24:  5700 Aborted                 ./main ../../maps/gh013.el bfs
&>bfs_gh013.txt
+ ./main ../../maps/gh014.el bfs
/var/slurmd/job4234796/slurm_script: line 25:  5758 Aborted                 ./main ../../maps/gh014.el bfs
&>bfs_gh014.txt
+ ./main ../../maps/gh015.el bfs
/var/slurmd/job4234796/slurm_script: line 26:  5960 Aborted                 ./main ../../maps/gh015.el bfs
&>bfs_gh015.txt
```

```
+ ./main ../../maps/gh016.el bfs
/var/slurmd/job4234796/slurm_script: line 27:  6478 Aborted                 ./main ../../maps/gh016.el bfs
&>bfs_gh016.txt
```

Table 7:

Here is the bash script I wrote to sum up the results for GraphIt.

```
#!/bin/bash

#  Algorithms.
ALG=( 'bfs_' 'cc_' 'pr_' 'sssp_' )

#  Graphs: there are 15 (0-15) in total.
GRAPH=( 'gh001.txt' 'gh002.txt' 'gh004.txt' 'gh005.txt' 'gh006.txt' 'gh007.txt' 'gh008.txt' 'gh009.txt'
'gh010.txt' 'gh011.txt' 'gh012.txt' 'gh013.txt' 'gh014.txt' 'gh015.txt' 'gh016.txt' )

FOLDER=( 'Graphit_Results' )

ET="elapsed time:"
RD="rounds"
OutPut="graphit_results.txt"
dot="../"

# Handles the SSSP portion due to the different output of the others.
function do_SSSP() {
  if [ -e "$1" ]
        then
        local bit=0
        local Sum=0
        local Rct=0
        local Count=0

        while read it
        do
        if [ "$it" != "$ET" ] && [ "$it" != "$RD"  ]
        then
        if (( ((bit % 2)) == 0 ))
        then
        Sum=$(python -c "print $Sum+$it")
        else
        Rct=$(python -c "print $Rct+$it")
        fi
        bit=$((bit + 1))
        fi
        Count=$((Count + 1))
        done < $1

        Count=$((Count/4))
        Total=$(python -c "print $Sum/$Count")
```

```
		RCount=$(python -c "print $Rct/$Count")
		printf '%s\t%s\t%.6f---%d\n' "$2" "$1" "$Total" "$RCount" >> "$dot$OutPut"
		fi
}

# Handles the BFS,CC,PR portions due to similar outputs.
function do_OTHERS() {
  if [ -e "$1" ]
  then
		local Sum=0
		local Count=0

		while read it
		do
		if [ "$it" != "$ET" ]
		then
		Sum=$(python -c "print $Sum+$it")
		Count=$((Count + 1))
		fi
		done < $1

		Total=$(python -c "print $Sum/$Count")
		printf '%s\t%s\t%.6f\n' "$2" "$1" "$Total" >> "$dot$OutPut"
  fi
}

# Handles the switch between do_SSSP and the do_OTHERS
function get_File() {
  local Sum=0
  local Count=0
  local FILE="$1$2"

  if [ "$1" == "sssp_" ]
  then
		do_SSSP $FILE $3
  else
		do_OTHERS $FILE $3
  fi
}

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<	START		<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
echo "	Starting summary script"

  for i in "${ALG[@]}" # MAP
  do
		printf '\t%sstarting:\n' "$i" >> "$OutPut"
		for j in "${GRAPH[@]}" # BFS
		do
		for k in "${FOLDER[@]}" # LOCATION
		do
		cd $k
		( get_File "$i" "$j" "$k")
		cd ..
```

```
        done
        done
  done
#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<        FIN        <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
echo "   Finished summary script"

exit
```

Table 8:

Here is the output for the BFS portion of Table 7.

```
        bfs_starting:
Graphit_Results    bfs_gh001.txt    0.024210
Graphit_Results    bfs_gh002.txt    1.590884
Graphit_Results    bfs_gh004.txt    0.441864
Graphit_Results    bfs_gh005.txt    0.009887
Graphit_Results    bfs_gh006.txt    0.664931
Graphit_Results    bfs_gh007.txt    0.004481
Graphit_Results    bfs_gh008.txt    0.004431
Graphit_Results    bfs_gh009.txt    0.003390
Graphit_Results    bfs_gh010.txt    0.003778
Graphit_Results    bfs_gh011.txt    0.003076
Graphit_Results    bfs_gh012.txt    0.002403
Graphit_Results    bfs_gh013.txt    0.063982
Graphit_Results    bfs_gh014.txt    0.028851
Graphit_Results    bfs_gh015.txt    0.094646
Graphit_Results    bfs_gh016.txt    0.026031
```