The Comparative Performance Analysis of Data-intensive Applications for IBM Minsky and Newell Systems*

Ilya Afanasyev[1]

Lomonosov Moscow State University, Russia
afanasiev_ilya@icloud.com

Abstract. The main goal of this work is to evaluate the performance differences of various data-intensive applications for IBM Minsky and Newell systems. As an important example of data-intensive applications, several fundamental graph algorithms have been selected. According to already existing implementation approaches, efficient implementations of selected graph algorithms have been developed. The measured performance has been compared between two generations of IBM Power systems, demonstrating the 1.5-4 times better performance on the Newell system. In addition roofline models have been generated for both Power and NVidia implementations, which allowed to compare the efficiency of the developed implementations for both systems. Based on the results demonstrated in the paper we can conclude that with a well-optimised code for the Minsky system, it is relatively easy to obtain significant acceleration with high efficiency on the Newell system at least for the described class of graph algorithms.

Keywords: IBM Power · NVidia GPU · Graph algorithms · Data-intensive applications · Roofline model

1 Introduction

New generations of various architectures, systems and platforms appear almost every half a year in modern supercomputing. For many supercomputer users and software developers the question if it is worth to start using the newer generation of the platform they are working with is extremely relevant. This issue is no less relevant for management of supercomputer centres, who are usually responsible for hardware updates. Answering each of the following questions can be very important: what acceleration can be achieved by using the newer generation of the system? Is it possible to use newer generations of the system without changing program source code a lot? How much effort has one to spend to reach peak performance capabilities of the new platform?

* The results were obtained in Lomonosov Moscow State University with the financial support of the Russian Science Foundation (agreement N 14-11-00190).

Thus, it is very important to study differences between various implementations on various system generations. A group of data-intensive algorithms was very important and challenging for any computational platform: even when properly implemented, graph algorithms tend to stress target platform memory subsystem, demonstrating low performance, high latency and low data locality, paired with poor cache usage. Graph algorithms represent data-intensive applications extremely well, since they tend to have an enormous amount of random memory accesses, paired with low computational complexity. Moreover, graph processing is extremely relevant nowadays, since it includes such important real-world applications as web-graphs and social-networks processing.

IBM Power systems provide an important example of modern high-performance shared-memory platforms. IBM Power systems are developed in a close cooperation with NVidia company, what leads to their native integration with modern NVidia GPUs (for example using NVLink technology). Consequently, these systems demonstrate a very high performance on a various groups of problems, including computational algebra and deep-learning. However, data-intensive applications are much less studied, despite the fact that IBM platforms have all the necessary features to execute these applications efficiently. In this paper we are going to review the two latest generations of IBM Power systems: Power S822LC system with codename "Minsky" and Power S822LC system with codename "Newell". Using the developed efficient implementations of various fundamental graph algorithms, we are going to evaluate the performance and efficiency difference between these two generations of Power system..

2 Target Platforms

In this paper we review the two latest generations of IMB systems. The first one is Power S822LC system (with codename "Minsky"), which contains two Power8 CPUs of between eight and ten cores each (10 cores per socket in our configuration). Each Power core is optimised to run up to 8 threads per core, which results into 160 threads per system. The resulting frequency of 12-core Power8 processor is equal to 4.1 GHz, with sustainable peak performance equal to 395 GFLOPs. "Minsky" platform is equipped with up to four Nvidia Tesla P100 GPUs (2 in the configuration available for us). The P100 GPU is equipped with 3584 light-weighted cores each one with a frequency of 1.1 MHz, resulting into 9.3 TFLOPs single-precision performance. The P100 GPU is also equipped with 12 or 16 GB device memory with 549 or 732 GB/s peak bandwidth respectively. The main feature of Power8 processor is NVLink bus technology support, which allows to connect up to four GPU devices directly to the chip, thus leading to a 2.8 times faster communication between host and device compared to traditional PCI configurations.

The second generation we review in this paper is Power AC922 system (with codename "Newell"), which contains two Power9 CPUs of between 12 or 24 cores (16 cores per socket on our system). Each core is optimised to run up to 4 threads per core, which results into 128 threads per system. The resulting frequency of

16-core Power 8 processor is equal to 4 GHz, with sustainable peak performance equal to 592 GFLOPs. The platform may be also equipped with 2, 4 or 6 NVidia Tesla V100 GPS (4 GPUs in the configuration available for us).

The V100 GPU of Volta architecture delivers considerably more performance: it is quipped with 5120 CUDA-cores and 640 Tensor-cores (special cores optimised for deep-learning problems), which results into 15.7 single-precision TFLOPs and 7.8 TFLOPs double-precision performance. Moreover, Volta architecture also adds many new features compared to Pascal architecture family: it introduces a new combined L1 data cache and shared memory subsystem, which significantly improves performance while also simplifying programming. Another important improvement is the increased memory bandwidth, where V100 provides 16GB HBM2 memory with 900 GB/sec peak memory bandwidth (1.5x delivered memory bandwidth versus Pascal GP100 and greater than 95% memory bandwidth efficiency running many workloads). Interconnect is also improved and based on the second generation of NVLink (version 2.0), which now supports CPU mastering and cache coherence capabilities with IBM Power 9 CPU-based servers.

In the memory subsystem Power 8 processors have 3 layers of cache memory: 64 KB of L1 data-cache, 512 KB of L2 cache and 8 MB of L3 cache per chip. Power 9 processors has smaller 32 KB L1 data-cache, but a very larger 120 MB L3 cache per chip.

3 Method

In order to comprehensively evaluate the performance differences between the reviewed systems on a class of data-intensive applications, we are going to discuss three fundamentally different types of implementations: CPU-only, GPU-only and hybrid. All those types of implementations are perfectly suitable for execution on target IBM power systems in different circumstances. For example, while processing large-scale graphs on GPUs in out-of-core mode (when graph can't be fully placed in device memory), it can be more beneficial to utilize multiple GPUs using unified memory technology, while for smaller graphs it can be more efficient to use only a single GPU to avoid unnecessary communications. In this paper we select 4 fundamental graph-processing problems and corresponding algorithms, which allow to solve those problems efficiently on both Power and NVidia platforms. Where possible, hybrid implementations are developed.

4 Problems, Algorithms and Input Data

Thus, we review 4 fundamental graph-processing problems: single source shortest paths (SSSP), breadth-first search (BFS), strongly connected components (SCC) and minimum spanning tree (MST) problems. The SSSP problem implies finding shortest paths in an undirected weighted graph from the selected source vertex to other vertices. The SCC problem implies partitioning a directed unweighted graph into disjoint sets of vertices, each one representing a strongly connected

component (a group of vertices where each vertex is reachable from another). The MST problem implies finding a subset of the edges that connects all the vertices together, without any cycles and with the minimum possible total edge weight. The BFS problem implies exploring the whole graph in the following manner: visiting all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level. In order to solve those problems efficiently on target architectures several fundamental algorithms have been selected. In the next paragraph, we provide the list of these algorithms together with a brief description of distinctive features and possible existing approaches to creating efficient implementations for both traditional CPUs and NVidia GPUs.

1. In oder to solve SSSP problem, Bellman-Ford [1, 2] is selected. From the computational point of view, the Bellman-Ford algorithm requires both floating point and integer arithmetics together with frequent indirect memory accesses.
2. In order to solve SCC problem, Forward-Backward algorithm with Trim step [3, 4] is selected. This algorithm requires only integer arithmetics with frequent indirected memory accesses, and has a more complex nested parallelism potential.
3. In order to solve BFS problem, Direction-Optimising algorithm is used [5, 6]. This algorithm has the smallest operation per byte ratio, with integer only arithmetics required. It also has the smallest ratio of computational complexity to the amount of transferred bytes through NVlink bus, since it has a linear-time complexity implementation.
4. Boruvka's algorithm is selected for solving MST problem [7, 8]. This algorithm requires usage of atomic operations, which may hurt the performance on certain architectures a lot.

According to the described implementation approaches, for each of the selected algorithms an efficient implementation has been developed for both Power and NVidia architectures, with hybrid generalisation where it is necessary and possible (for large-scale graph processing). We also used a variety of both synthetic and real word graphs as input data set. For synthetic graphs, RMAT [14] and SSCA-2 [15] graphs have been used. For real world graphs, we used various road, social network and web graphs, all available in the following dataset [13].

5 Implementations Details

In order to compare the performance results between two generations of IBM Power Systems, first we had to develop efficient implementations for Power 8 & Pascal architectures. To achieve efficient parallelisation and vectorisation inside a single Power socket, several OpenMP have been inserted. The #pragma parallel for OpenMP directive has been used with the schedule(guided,1024) OpenMP clause, in order to reduce overheads of synchronisations between threads. Moreover, for Power implementations, vectorisation has been used, which was achieved with -O3 -qsimd=auto -qhot -qarch=pwr8 -qtune=pwr8 -qpdf2 compiler flags.

For Volta and Pascal operations, various optimisations including texture cache usage have been also used.

Moreover, for both Power and NVidia architectures the following sorting strategy have been used in order to improve data locality: input graph edges have been sorted in a way that edges, stored in adjacent memory cells, start pointing to adjacent cells in reachability(or distances) arrays. This reordering results into gather and scatter vector operations being much more efficient for adjacent edges (since information is gathered from adjacent cells of memory in distances or reachability array). It is also possible to remove loops and multiple arcs during this pre-processing stage. The proposed optimisation of storage formats provided almost up to a 10x performance improvement compared to the implementation with randomly sorted edges of the input graph.

As a result of the described optimisations, the developed implementations demonstrated high performance on Power 8 & Pascal architectures. In order to optimise the developed implementations for Power 9 & Volta architectures, minor changes had to be made: most importantly, graph edges reordering had to be changed in accordance to cache sizes of the new platforms.

6 Performance Evaluation

The achieved acceleration between Minsky and Newell systems is demonstrated on Fig. 1. The acceleration is calculated as the ratio of execution times on the same input graphs, measured in 3 different modes: Power 9 to Power 8 acceleration, and V100 to P100 acceleration with and without data copy time included in time measurement. When data copy time is not included, the input graphs are supposed to be already placed inside device memory. While when data copy time is included, the input graph have to be copied from host into device memory through NVLink, and the time required for these copies is included into measured execution time.

From Fig. 1 it is possible to highlight different trends.

1. Power 9 implementations demonstrate higher performance compared to Power 8 ones due to larger L3 cache on the small and medium graphs, when it is possible to fully place the arrays with indirect accesses into L3 cache.
2. Hybrid implementations for large graphs (L-RMAT) demonstrate better performance on Newell system due to higher bandwidth of NVLink 2.0 used as interconnect.
3. On a very small graphs (for example small scale RMAT graphs, with a size less then 50 MB) V100 implementations demonstrate even lower performance compared to P100 implementations, due to the lack of necessary amount of parallelism.

7 Efficiency evaluation

For the efficiency analysis of the developed implementations, the roofline model [9] is used. The Roofline model is a visual performance model used to provide per-

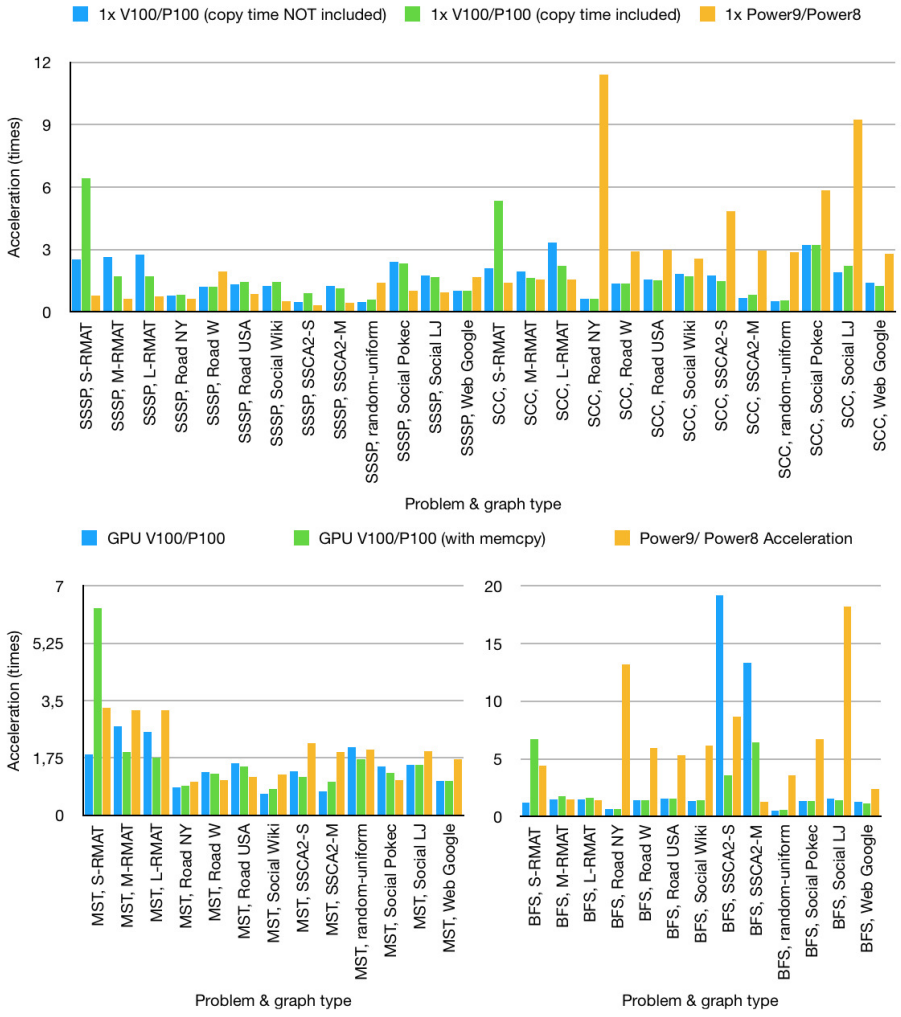

Fig. 1. The achieved acceleration between Minsky and Newell systems for various graph problems and different input graphs

formance estimates of a given compute-bound or memory-bound kernels or applications running on both multi-core and accelerator architectures, by showing hardware limitations based on peak performance and peak memory bandwidth. The roofline model provides insights into the system performance capabilities, and moreover shows inherent hardware limitations, and potential benefit and priority of optimisations of target applications. In this work the roofline model is used to evaluate the efficiency of both CPU-only and GPU-only implementations compared to peak performance capabilities of target system.

There is also an extension of roofline model called cache-aware roofline model [10] - a novel approach to provide a more insightful performance modelling of modern architectures by introducing cache-awareness, thus significantly improving the guidelines for application optimisation.

In order to create a cache-aware roofline model for both Power and NVidia architectures, peak integer performance, peak memory bandwidth and peak bandwidths of caches of all levels have to be benchmarked. The benchmarking of Power systems can be obtained using [11] benchmark, while peak metrics for NVidia architectures can be obtained using [12]. The collected benchmark results are shown in table 1.

Table 1. Peak performance/bandwidth characteristics for various components of IBM Power systems

Metric name	Power 8	Power 9	P100 GPU	V100 GPU
Peak integer performance	76 GIOP/s	81 GIOP/s	2242 GIOP/s	3631 GIOP/s
L1 cache / shared memory bandwidth	548 GB/s	621 GB/s	8502 GB/s	14917 GB/s
L2 cache bandwidth	235 GB/s	249 GB/s	1757 GB/s	2795 GB/s
L3 cache bandwidth	-	-	-	-
Texture cache bandwidth	-	-	2629 GB/s	3932 GB/s
Memory bandwidth	97 GB/s	103 GB/s	628 GB/s	898 GB/s

In addition to benchmarking peak performance and bandwidths, for each application or kernel of interest the number of integer operations and operation per byte ratio has to be collected. For obtaining those values, on NVidia platform nvprof is used for calculating both bytes requested and integer instructions operation. The amount of bytes requested for Power processors can be obtained using Linux Perf or PAPI utilities, while measuring some specific memory-related hardware events. However, it is quite problematic to calculate the amount of integer operations on modern central processors, since both modern IBM and Intel processors lack hardware events responsible for the amount of executed integer or floating-point. As a result, we had to directly calculate an approximate amount of operations from the source code of the program. We understand that it not a very reliable method, since in complex algorithms (such as Boruvka's algorithm) it can be very problematic to estimate the amount of

integer instructions calculated. So, the development of a more general method for creating roofline models is one of the priority areas for our further research.

Based on the data provided in table 1, roofline models can be obtained both P100/V100 GPUs and P8/P9 processors for the 3 reviewed graph-processing problems (MST problems is currently excluded due to the difficulties described above). The obtained sample roofline model for Pascal GPU architecture for several problems and various input graphs are demonstrated on Fig. 2. The provided roofline indicates that the investigated implementations are in general memory-bound, while being bottlenecked somewhere in between L1/shared memory and texture cache roofs, which is equal to 40-90% of peak performance.

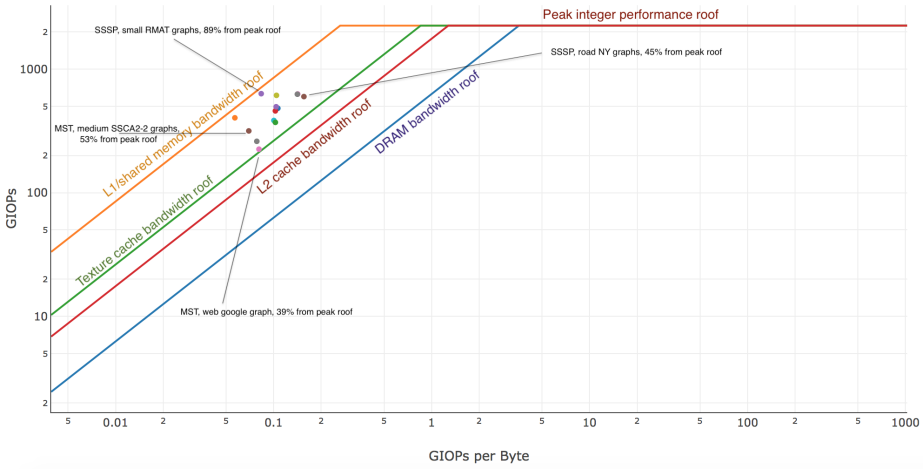

Fig. 2. An example of the cache-aware roofline model for SSSP and MST operations, P100 GPU

Data from the generated rooflines can be used for comparing the achieved efficiency of various implementations and input graphs between V100/P100 and Power 8/Power 9 target platforms. This comparison is presented on Fig. 3. Fig. 3 indicates that in average the efficiency between the two reviewed platforms remains relatively unchanged for a given class of algorithms even in an absence of platform-specific optimisations. This leads us to the conclusion that many graph algorithms can be effectively ported from Minsky to Newell systems without significant efforts required for algorithms and code modifications.

8 Conclusion

In this paper, a comparative analysis of the acceleration and the overall efficiency for several implementations of fundamental graph-processing algorithms

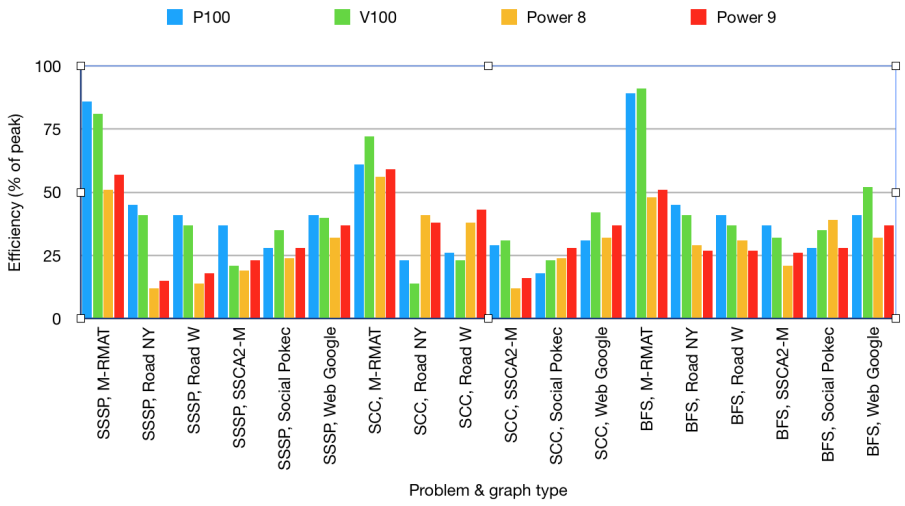

Fig. 3. Efficiency comparison for various problems and input graphs

has been provided for IBM Minsky to Newell systems. On various type of input graphs the developed implementations for Newell system demonstrate up to 6 times acceleration compared to previous generation of GPU architecture (pascal), while up 12x acceleration compared to previous generation of Power architectures (P8).

In order to evaluate the efficiency of the developed implementations, cache-aware roofline model has beed utilised for a significant part of the algorithms and input data. Analysing the absence of changes in roofline efficiency between the reviewed platforms it is possible to emphasise that the transition from Minsky to Newell architecture does not require complex modifications of the source code (at least for the reviewed algorithms and implementations), and can performed relatively easily for a variety of applications, while obtaining significant acceleration.

The future work includes more detailed performance analysis using other performance models (which will allow to cover hybrid implementations), finding the solution of the described difficulties with the construction of CPU cache-aware roofline mode, while extending the sets of the reviewed problems, algorithms and input data-set.

References

1. Nepomniaschaya, A. S. (2001, September). An associative version of the Bellman-Ford al- gorithm for finding the shortest paths in directed graphs. In International Conference on Parallel Computing Technologies (pp. 285-292). Springer, Berlin, Heidelberg.

2. Jeong, I. K., Uddin, J., Kang, M., Kim, C. H., Kim, J. M. (2014). Accelerating a Bellman?Ford Routing Algorithm Using GPU. In Frontier and Innovation in Future Computing and Communications (pp. 153-160). Springer, Dordrecht.
3. Fleischer, L. K., Hendrickson, B., Pnar, A. (2000, May). On identifying strongly connected components in parallel. In International Parallel and Distributed Processing Symposium (pp. 505-511). Springer, Berlin, Heidelberg.
4. Barnat, J., Bauch, P., Brim, L., Ceska, M. (2011, May). Computing strongly connected components in parallel on CUDA. In 2011 IEEE International Parallel & Distributed Processing Symposium (pp. 544-555). IEEE.
5. Beamer, S., Asanovi?, K., Patterson, D. (2013). Direction-optimizing breadth-first search. Scientific Programming, 21(3-4), 137-148.
6. Zou, D., Dou, Y., Wang, Q., Xu, J., Li, B. (2013, November). Direction-Optimizing Breadth-First Search on CPU-GPU Heterogeneous Platforms. In HPCC/EUC (pp. 1064-1069).
7. Cong, G., Bader, D. A. (2007, August). Techniques for designing efficient parallel graph algorithms for SMPs and multicore processors. In International Symposium on Parallel and Distributed Processing and Applications (pp. 137-147). Springer, Berlin, Heidelberg.
8. Vineet, V., Harish, P., Patidar, S., Narayanan, P. J. (2009, August). Fast minimum spanning tree for large graphs on the GPU. In Proceedings of the Conference on High Performance Graphics 2009 (pp. 167-171). ACM.
9. G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato and M. Püschel, "Applying the roofline model," 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, 2014, pp. 76-85. doi: 10.1109/ISPASS.2014.6844463
10. A. Ilic, F. Pratas and L. Sousa, "Cache-aware Roofline model: Upgrading the loft," in IEEE Computer Architecture Letters, vol. 13, no. 1, pp. 21-24, 21 Jan.-June 2014. doi: 10.1109/L-CA.2013.6
11. Empirical Roofline Tool, https://bitbucket.org/berkeleylab/cs-roofline-toolkit. Last accessed 4 Sep 2018
12. Konstantinidis, E.; Cotronis, Y., "A quantitative performance evaluation of fast on-chip memories of GPUs", 24th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Heraklion, Crete, Greece, pp. 448-455, 2016. doi: 10.1109/PDP.2016.56
13. Stanford Large Network Dataset Collection - SNAP: Stanford, https://snap.stanford.edu/data/. Last accessed 3 Sep 2018
14. Chakrabarti, D., Zhan, Y., Faloutsos, C. (2004, April). R-MAT: A recursive model for graph mining. In Proceedings of the 2004 SIAM International Conference on Data Mining (pp. 442-446). Society for Industrial and Applied Mathematics.
15. Bader, D. A., Madduri, K. (2006). Gtgraph: A synthetic graph generator suite. Atlanta, GA, February.