Performance Impact of Memory Channels on Sparse and Irregular Algorithms

Oded Green[1,2], James Fox[2], Jeffrey Young[2], Jun Shirako[2], and David Bader[3]

[1]NVIDIA Corporation — [2]Georgia Institute of Technology — [3]New Jersey Institute of Technology

Abstract— **Graph processing is typically considered to be a memory-bound rather than compute-bound problem. One common line of thought is that more available memory bandwidth corresponds to better graph processing performance. However, in this work we demonstrate that the key factor in the utilization of the memory system for graph algorithms is not necessarily the raw bandwidth or even the latency of memory requests. Instead, we show that performance is proportional to the number of memory channels available to handle small data transfers with limited spatial locality.**

Using several widely used graph frameworks, including Gunrock (on the GPU) and GAPBS & Ligra (for CPUs), we evaluate key graph analytics kernels using two unique memory hierarchies, DDR-based and HBM/MCDRAM. Our results show that the differences in the peak bandwidths of several Pascal-generation GPU memory subsystems aren't reflected in the performance of various analytics. Furthermore, our experiments on CPU and Xeon Phi systems (see extended version [11]) demonstrate that the number of memory channels utilized can be a decisive factor in performance across several different applications. For CPU systems with smaller thread counts, the memory channels can be underutilized while systems with high thread counts can oversaturate the memory subsystem, which leads to limited performance. Finally, we model the potential performance improvements of adding more memory channels with narrower access widths than are found in current platforms (see [11]). We analyze performance trade-offs for the two most prominent types of memory accesses found in graph algorithms, streaming and random accesses.

I. Introduction

Graph processing is usually considered memory-bound due to the irregular and data-dependent nature of most graph problems, which leads to many irregular memory accesses. It is also commonly believed that these algorithms contain a mix of bandwidth- and latency-bound operations. For modern shared memory systems that handle these types of applications, there has been an explosion in new memory technologies as well as the amount of parallelism for computation and memory accesses. For example, a Power 9 system might have multiple processors in a single node, each with up to 24 cores and 96 threads. Similarly, Intel's Knights Landing (KNL) processor can have up to 68 cores and 272 threads, and the latest generation of NVIDIA's Volta GPUs can support up to 5120 threads on 80 streaming multi-processors. This increase in thread parallelism has been combined with faster and more complex memory technologies and standards in the last decade, including DDR4, DDR5 (expected to be released in 2020), GDDR5, Hybrid Memory Cube (HMC), and High Bandwidth Memory 2.0 (HBM2).

Graph algorithms are typically latency-bound if there is not enough parallelism to saturate the memory subsystem. However, the growth in parallelism for shared-memory systems brings into question whether graph algorithms are still primarily latency-bound. Getting peak or near-peak bandwidth of current memory subsystems requires highly parallel applications as well as good spatial and temporal memory reuse. The lack of spatial locality in sparse and irregular algorithms means that prefetched cache lines have poor data reuse and that the effective bandwidth is fairly low. The introduction of high-bandwidth memories like HBM and HMC have not yet closed this inefficiency gap for latency-sensitive accesses [14]. However, high-bandwidth memory has introduced a higher number of memory channels for stacked DRAMs, which can process a higher number of outstanding memory transactions. For this work, we define a **memory channel** as a logically grouped set of DRAM DIMMs for traditional memory systems (or a logical grouping of TSVs within a 3D stacked memory like HBM or HMC). Each channel is able to service and output data requests independently.

In this work, we analyze the performance and scalability of graph algorithms and their memory access characteristics as the number of threads is increased on a wide range of CPU and GPU systems. The fact that these systems have such a high thread count is crucial as the phenomena of over-saturating the memory subsystem is not visible for smaller thread counts (even on new CPU systems with tens of cores).

Extended Version of Paper

An extended version of our paper can be found at [11]. The extended version of this paper has additional benchmarks and performance analysis not covered in this shorter paper. Specifically, the extended version includes experiments on CPU systems, including Intel's KNL processor. The extended version also includes a performance model that looks at how a system might perform with additional but narrower memory channels. Specifically, this analysis focuses on systems where the bandwidth is not increased, but rather that bandwidth is split across more memory channels of narrower width.

Contributions

In this paper we challenge the commonly held notion that graph algorithms are either memory bandwidth- or latency-bound. Instead we show that the performance of these algorithms is dependent on the number of memory channels in

978-1-7281-5987-4/19/$31.00 ©2019 IEEE
DOI 10.1109/IA349570.2019.00016

the memory subsystem. Using a wide range of CPU and GPUs with a mix of DDR4, GDDR5, HBM2, and Multi-Channel DRAM (MCDRAM), we benchmark numerous analytics from highly optimized frameworks across a wide range of graphs with different properties. Contrary to past research that focused on analyzing only large thread counts, our analysis includes threads counts at different ranges, which allows us to find the point at which the memory subsystem is stressed.

Key findings: For the Intel KNL processor, which has both DDR4 and MCDRAM memories, MCDRAM starts to outperform DDR4 at around 64 threads. Neither DDR nor MCDRAM are saturated with less than 64 threads, and in this regime there is little performance difference due to the similar latencies of the memories.

For the NVIDIA GPU, we compare an HBM2-based GPU system with several GDDR5 and GDDR5X GPUs. While the GDDR5-based GPU's specifications suggest that peak available bandwidth has a large role to play, the HBM2 GPU outperforms them by over a factor of 2X even when the bandwidth relative to a GDDR5 device is less than 1.33X. Furthermore, there is little variance in performance across the GDDR5-based GPUs. We show that the performance is correlated to the number of memory channels available to the system - **a number that is rarely reported when conducting performance analysis**.

Lastly, we present a performance model that projects the performance impact of added, narrower memory channels. Our model does not focus on how such a subsystem should be created, but rather evaluates performance trade-offs from an application's point of view.

While the significance of memory channels has come up in architecture research and states that more channels will be better, to the best of our knowledge our paper is the first to show this dependency for sparse applications and to show that the performance of these applications is limited by the number of channels.

II. Related Work

Applications: Recently, Beamer *et al.* [1] showed that memory bandwidth is not fully utilized for several widely used graph kernels on a multi-threaded processor but that it is still possible to improve bandwidth utilization. We extend this analysis and show that for the same types of applications analyzed in Beamer *et al.* [1] it is quite likely that the fairly small number of thread counts (32 threads with 16 cores) was **not large enough** to saturate the memory sub-system, and this led to memory under-utilization. Xu et. al. [16] profiled a spectrum of graph applications on the GPU and found that long memory latencies as a result of high L1 cache miss rates tended to be the biggest performance bottleneck. Peng et. al. [13] analyzed the performance of different benchmarks for large threads on MCDRAM vs. DRAM on the KNL system. They concluded that sequential access, bandwidth-bound applications benefit from the MCDRAM but random access benchmarks are latency-bound and perform better on DRAM.

Architectural Approaches: Brunvand et. al. [2] note that the number of memory channels and clock rate determine peak memory bandwidth and correspond to the level of achievable concurrency while bank counts, access patterns, and the sophistication of memory controllers are other factors which determine the actual bandwidth achieved.

Generally speaking, data bandwidth is given by $B = W * F$, where W is data width and F is data frequency. Zhang et. al. [17] propose *Half-DRAM* as a way to toggle W to match B and to induce narrower rows, while preserving bandwidth and improving power consumption. Their implementation decouples rows into half-rows within banks, which reduces row activation power and enables doubling of memory-level parallelism. Similarly, the Emu Chick architecture [5] divides a DRAM channel into multiple "Narrow-Channel DIMMs" (NCDIMM) which allow for more fine-grained access for irregular applications. The Emu Chick would be an interesting comparison point for this study, but the Chick prototype currently is not able to run large-scale analytics applications.

Fine-grained DRAM or *FGDRAM*, as proposed by O'Connor et. al. [10], proposes the most novel approach to scaling memory bandwidth and reducing the energy contributions of memory accesses. This work rethinks the design of stacked memory by partitioning the DRAM die into smaller independent units with dedicated channels (64 channels versus 16 on current HBM) and reduced effective row size. Results on a GPU simulator show up to several factors of improvement on irregular benchmarks, due to increased concurrency for handling requests and higher row activation rates. Our characterization looks to evaluate whether this "fine-grained" approach with more channels would lead to better performance for real-world analytics test cases.

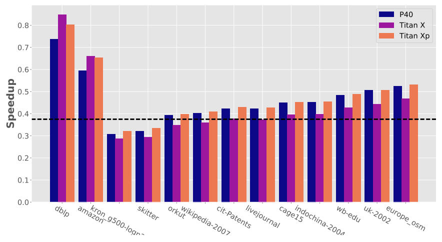
III. Results - GPU

System Configuration: The specifications of the GPUs used in our experiments can be found in Table I. The data transfer time between the GPU and GPU is not factored into runtime measurements, so we do not include the characteristics of the host systems and their CPU configurations. Experiments are primarily conducted on four different NVIDIA Pascal (same micro-architecture) GPUs. While the GPUs differ slightly in terms of core counts and clock frequencies (see Table I), these devices have roughly equivalent FLOP rates. Benchmarking is also run on a NVIDIA V100 GPU (Volta micro-architecture) with HBM2 memory.

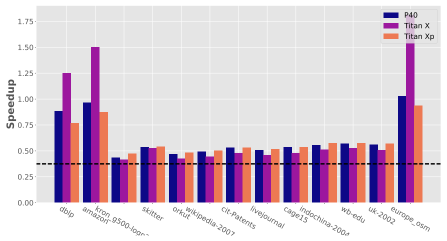
The Pascal-based GPUs have slightly different characteristics in terms of core counts, frequency, cache sizes, number of memory transactions per second, bandwidth, and **memory channels**. Specifically, these differences include the following: 1) Core counts vary by 7%, 2) clock speed varies by 26%, 3) the P100 also has a larger L2 cache size (4 MB vs 3 MB), and 4) maximum memory bandwidth, which varies by more than $2\times$. Cache size is not as impactful for the larger graph inputs as these are several times larger than any current GPU's L2 cache. Thus, we can guarantee that a majority of the memory accesses will need to access global memory.

TABLE I: GPU used in experiments. GPUs are primarily Pascal generation except for one V100 GPU from the Volta generation of GPUs. FLOP rate for the GPUs is for single point precision. All GPUs are PCI-E based.

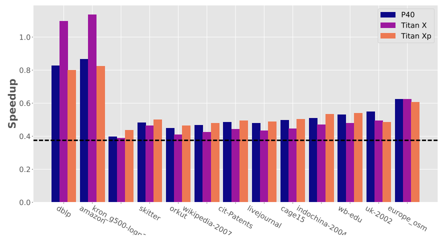
Architecture	Processor	Base Clock	SMs	Total SPs	L2 Size (MB)	DRAM Type	Size (GB)	Mem Clock (MT/s)	Bandwidth (GB/s)	Bus width (bits)	Memory Channels	Sin. Per. (TFLOP/s)	TDP (W)
GPU-CUDA	P100	1126	56	3854	4	HBM2	16 GB	1406	720	4096	32	9.5	250
GPU-CUDA	P40	1303	30	3840	3	GDDR5	24 GB	7200	345	384	12	10	250
GPU-CUDA	Titan Xp	1480	30	3840	3	GDDR5X	12 GB	11410	547.7	384	12	10.1	250
GPU-CUDA	Titan X	1417	28	3584	3	GDDR5X	12 GB	10000	480	384	12	11.3	250
GPU-CUDA	V100	1370	80	5120	4	HBM2	16 GB	1750	9000	4096	32	14	250



(a) PageRank



(b) Betweenness Centrality



(c) Shortest Path

Fig. 1: Speedup in comparison to a P100 GPU (only HBM2 based GPU). Execution times are normalized to the P100's time. The line $y = .375$ denotes the ratio in number of memory channels of the other GPUs to the P100.

The memory subsystems in the GPUs serves as the biggest distinction, and this varies between 345 GB/s on the P40 to the highest bandwidth of 720 GB/s on the P100. The NVIDIA P100 and V100 use High Bandwidth Memory (HBM2) and both have 32 memory channels serving 4096-bit total bus width. The remaining GPUS use GDDR5/GDDR5X memory and have 12 channels serving a 384-bit bus width. The number of memory channels for these processors is determined to the best of our ability using a recent NVIDIA presentation of high-bandwidth memory [12]. We also consulted per-channel width information reported in Micron Technology's GDDR5 and GDDR5X technical notes.

Gunrock Analysis on Pascal and Volta GPUs: We use the highly optimized Gunrock [15] framework to demonstrate that there is almost no difference for the three GDDR5 based GPUs despite there being up to a nearly 60% difference in bandwidth. This analysis and comparisons to the P100 and V100 GPUs are used to show that these common analytics kernels are not primarily *bandwidth-bound*.

Experiment setup: To measure the impact of the memory channels on the performance of graphs on the GPUs we use both Gunrock, and we also report results for nvGRAPH, a linear-algebra based graph analytics library by NVIDIA. The graphs used in the experiments are taken from SNAP [8] and the Florida Matrix Market [4] repositories. For Gunrock, we measure the execution time (excluding data transfer) for the following analytics: Betweenness Centrality, Single-Source Shortest Path, and PageRank. Runtime results for Betweenness Centrality and Single-Source Shortest Path are averaged for the 200 top-degree vertices in the graph. Results for PageRank are averaged for 5 runs of each kernel, where each run has 50 fixed iterations. Using nvGRAPH we measure the execution time for Single-Source Shortest Path and PageRank using similar settings.

Gunrock Analysis: Fig. 1 depicts the performance of several graph algorithms using Gunrock. The abscissa represents the graph used in the experiment. The ordinate represents the normalized speedup execution with respect to the NVIDIA P100 PCI-E based GPU, the only NVIDIA Pascal GPU to have high-bandwidth memory.

For most algorithms and inputs, the P100 GPU outperforms the remaining GPUs by a significant factor - despite the other GPUs supporting a larger number of memory transactions per second. There are a few instances where the GDDR5-based GPUs outperform the P100. In most cases, this occurs for the smaller graphs where the data structure used by the analytics can mostly fit into the GPU's LLC. There are a very few cases where the larger graphs perform better on the GDDR5-based GPUs. As these experiments are not optimized to take advantage of a specific GPU, the difference in execution might be associated with load-balancing. These few cases require additional investigation.

For three of the benchmarks (PageRank, Betweenness Centrality, and Single-Source Shortest Path), the P100 outperforms the other benchmarks by about 2x-3x. Recall that the P100 has 32 memory channels whereas the other GPUs have 12. We have added a line at $y = \frac{12}{32} = 0.375$ (in all the subplots of Fig. 1) to indicate this ratio. Note that the bars of the GDDR5-based GPUs are fairly close to this curve. This is especially surprising given the fact that the Titan XP has almost 60% more bandwidth than the P40, while the difference in their execution time is clearly not 60%. Further, the Titan XP has 75% of the peak bandwidth fort the P100.

TABLE II: GPU Bandwidth Analysis With Synthetic Micro-benchmarks.

	P100	P40	Titan Xp	Titan X
Peak bandwidth (GB/s)	720	345	547.7	480
Coalesced read (GB/s) [3]	573	230	316	270
Coalesced write (GB/s) [3]	432	249	312	266
Random read (GB/s)	17.1	9	9	7.7
Random write (GB/s)	7.4	3.9	8.4	7.6

If these graph algorithms were solely bandwidth-bound, then we could expect the execution times to be correlated. Rather, it seems that, across all the GPUs, the ability to handle saturation of memory subsystems via additional channels is the dominant factor in the performance of the algorithms.

NVIDIA (VOLTA) V100 GPU: In the above analysis we primarily focus on the Pascal based GPUs as these share a lot of common traits. Experiments with the V100 GPU (not shown) show that the performance of the V100 is quite similar to the P100 GPU despite having an additional 30% cores and extra 25% bandwidth. This evaluation demonstrates that performance is constrained by the number of memory channels rather than by other factors.

nvGraph Analysis: We also run experiments using nvGRAPH [9], specifically for PageRank and Single-Source Shortest Path. NVGraph implements graph algorithms using linear algebra based operations (as discussed in the GraphBLAS standard [7]). For the sake of brevity, we only give a short summary of the results on for nvGRAPH and note that similar trends were seen when comparing performance for the P100 and P40. By using nvGraph, in addition to Gunrock, we show that the performance impacts of many small, irregular accesses is not dependent on the programming and load-balancing scheme that is use in Gunrock, and we demonstrate that this performance penalty for high-bandwidth, limited memory channel systems also can be found in sparse matrix-based implementations of graph analytics.

Synthetic Benchmark Analysis: Table II depicts the performance of the GPUs running two synthetic benchmarks: 1) the SHOC benchmark [3], which tests the GPUs bandwidth capabilities for regular memory access patterns and 2) random memory access benchmarks that measure random reads and writes in an array. The latter is our own implementation, which performs reads and writes to addresses determined by an efficient FNV-1 [6] hash. Bandwidth reported for random reads and writes is *effective bandwidth*, or simply the total number of memory accesses (scaled by data type size) divided by execution time. The coalesced read/write benchmarks show relative performance consistent with peak bandwidth differences across the different GPUs. However, our benchmarks suggest that the P100 is roughly twice as effective as other GPUs on random reads. For random writes, the P100 achieves comparable bandwidth to the Titan Xp and Titan X, a ratio that is 2× more than the P40.

IV. Conclusions

In this paper, we have demonstrated that the performance of graph algorithms depends not just on latency or bandwidth but instead is correlated with the number of memory channels available in the memory subsystem. Our experiments show that as a larger number of threads are launched on both Intel's KNL processor and NVIDIA GPUs, the high bandwidth memory system outperforms DRAM by a factor that is closely tied to the ratio of memory channels in their respective memory subsystems. This scalability for high-bandwidth memory systems is almost twice is high as that of the DRAM-based systems, even when controlling for bandwidth and other factors. Additionally (see Extended version [11]), by using gap analysis and a simple performance model, we project that systems with narrower but more numerous channels will provide additional performance benefits for continued performance scaling up to maximum thread resources. This finding is especially important as new processors and accelerators with larger numbers of cores are deployed, and the gap between threads and memory channels continues to grow.

References

[1] S. Beamer, K. Asanovic, and D. Patterson, "Locality Exists in Graph Processing: Workload Characterization on an Ivy Bridge Server," in *IEEE Int'l Symp. on Workload Characterization*, 2015.

[2] E. Brunvand, N. Chatterjee, and D. Kopta, "Why graphics programmers need to know about dram," in *ACM SIGGRAPH 2014 Courses*. ACM, 2014, p. 21.

[3] A. Danalis, G. Marin *et al.*, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.

[4] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," http://www.cise.ufl.edu/research/sparse/matrices, 2011.

[5] T. Dysart, P. Kogge *et al.*, "Highly scalable near memory processing with migrating threads on the emu system architecture," in *Proceedings of the Sixth Workshop on Irregular Applications: Architectures and Algorithms*, ser. IA$\hat{3}$ '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 2–9. [Online]. Available: https://doi.org/10.1109/IA3.2016.7

[6] G. Fowler, "Fowler/noll/vo (fnv) hash," *ONLINE http://isthe.com/chongo/tech/comp/fnv*, 1991.

[7] J. Kepner, P. Aaltonen *et al.*, "Mathematical foundations of the GraphBLAS," in *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*. IEEE, 2016, pp. 1–9.

[8] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford Large Network Dataset Collection," Jun. 2014.

[9] NVIDIA, "nvGraph," 2016.

[10] M. O'Connor, N. Chatterjee *et al.*, "Fine-grained dram: energy-efficient dram for extreme bandwidth systems," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 41–54.

[11] Oded Green and James Fox and Jeffrey Young and Jun Shirako and David Bader, "Performance Impact of Memory Channels on Sparse and Irregular Algorithms," *https://arxiv.org/abs/1910.03679*, 2019.

[12] M. OConnor, "Highlights of the high-bandwidth memory (hbm) standard."

[13] I. B. Peng, R. Gioiosa *et al.*, "Exploring the performance benefit of hybrid memory system on hpc environments," in *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*. IEEE, 2017, pp. 683–692.

[14] M. Radulovic, D. Zivanovic *et al.*, "Another trip to the wall: How much will stacked dram benefit hpc?" in *Proceedings of the 2015 International Symposium on Memory Systems*, ser. MEMSYS '15. New York, NY, USA: ACM, 2015, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/2818950.2818955

[15] Y. Wang, Y. Pan *et al.*, "Gunrock: Gpu graph analytics," *arXiv preprint arXiv:1701.01170*, 2017.

[16] Q. Xu, H. Jeon, and M. Annavaram, "Graph processing on gpus: Where are the bottlenecks?" in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 140–149.

[17] T. Zhang, K. Chen *et al.*, "Half-dram: a high-bandwidth and low-power dram architecture from the rethinking of fine-grained activation," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*. IEEE, 2014, pp. 349–360.