The Name of the Title is Hope

Anonymous Author(s)

ABSTRACT

CCS CONCEPTS

• **General and reference** → **Evaluation**; **Measurement**; **Performance**; • **Computer systems organization** → *Special purpose systems.*

ACM Reference Format:

Anonymous Author(s). 2018. The Name of the Title is Hope. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

2 RELATED WORK

2.1 High Performance Computing

Dally, Keckler and Kirk's 2021 historical review of Graphic Processing Unit (GPU) development highlights the duality of the relationship between enabling hardware and the applications that use it. They explore the close relationship between Machine Learning applications and the GPUs. Their discussion suggests that while availablity of GPUs enabled more machine learning applications to be built, the reciprocal demand for higher performance machine learning models drives the development of improved GPUs. The salient question is whether hardware optimized for graph processing will have the same impact in the data analytics domain that GPUs had on scientific computing [6].

The lengthy 2023 report by Mutlu and their peers from the SAFARI research group is a response to the percieved limitations of the dominant processor-centric computing paradigm. The authors characterise the processing centric paradigm as one where large and dispersed data are moved to a central processing unit (or GPU etc) to be processed and returned to memory. They argue that the processor centric architecture is inefficent, and that rather than being compute bound, most modern applications are memory bound as a result. Their analysis finds that up to 62 percent of all power used by computers is just the action of moving data to and from memory. Their reponse is a processing in memory (PIM) paradigm. PIM is an old idea, but they argue is becoming viable with the emergence of new hardware technology like 3D chips. They define two methods of PIM: Processing Using Memory (PUM) and Processing Near Memory (PNM). PUM acknowledges that many of the simplest functions like adding scalar values to entire rows of memory, or initializing large blocks is a simple enough primitive option that it can be performed by the memory without having to be mapped to the CPU. PNM recognises that emerging 3d chips have small logic controllers that can be leveraged for simple decentralized processing actions. Their paper makes extensive references to Graph Processing. They identify that a driving cause of poor graph processing perfomance is the random memory accesses that results from sparse adjacency matrix representations. A second reason for poor performance is that the actual processing of items pulled out from memory is trival and completed quickly, exacerbating the effects of memory access latency. They design an architecture TESSERACT which as described appears to be a competitor to PiUMA. Regarding GPUs, they assert that they hide long latencies of memory accesses by interleaving arithmetic and logic operations. They present DAMOV, their framework for measuring memory-boundedness, identifying common culprits as cache misses, cache coherence traffic and long queueing latencies. They use intel V-TUNE for the profiling aspect of DAMOV, and then implement locality based clustering to characterize the spatial and temporal clustering features of applications under test. Overall this work motivates the need for specialized architectures to improve performance and proposed Processing-In-Memory as a paradigm to address the limitations of processor-centric models, including specifically for graph processing. They describe TESSEARCT as a possible competitor to PiUMA and DAMOV as a profiling and simulation suite [9].

2.2 System Benchmarking

A 2005 Study by Weinberg et. al. examines the measurement of spatial and temporal locality in high performance computing. They define spatial locality as memeory addresses which are located close to each other, and temporal locality as the same memory addresses being accessed repeatedly over time. They conduct their measurements by instrumenting their code using the MEMSIM platform, and use it to generate a spatial and temporal locality index with parameters L and K. L is a measurement of the stride size, and K is a measure of the randomness of access. They present these metrics despite earlier warnings in their own work about the reductive effects of generating an index to represent locality. Their analysis suggests that a lack of locality degrades the execution time of applications and so it is interesting that neither Graph500 nor the newer graph benchmark suites account for the loading or structuring of graphs in memory in their measurement. Chatacterising the shape of graphs in memory appears to be an under-explored dimension of the benchmarking space [11].

In 2009 Adhianto et. al. from Rice University released the initial build of the High Performance Computing Toolkit (HPCToolkit). The toolkit aims to profile the performance of code on high performance systems without needing to instrument the code, reducing the overheads imposed on the system under test. Their approach and toolkit exemplifies the measurement of perforance without instrumentation, demonstrating the feasibility of profiling code without intrinsicly degrading its performance [1].

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

A 2010 description of how Google implements system profiling across its data warehouses by Ren et. al. focuses on continuous monitoring rather than benchmarking. Their experience argues that sampling binaries during execution rather than fully instrumenting at compile time them is a superior approach that reduces memory usage and execution time. They sample events, which can include clock cycles, L1 and L2 cache misses and branch mispredictions. Their work provides a precedent for using profiling to compare different hadware implementations of the same application, supporting our evaluation of Graph Algorithms across CPU, GPU and PiUMA for the HIVE project. There appears to be a gap in defining what a standard 'profile' is for a graph algorithm. Detemining what a standard 'profile', and further determining a method to effectively vizualize memory accesses for graph applications by time and locality will be a prosperous avenue of further research [10].

The 2015 paper "Profiling a Warehouse-scale Computer" from Kanev et. al. provides insights into limitations and bottlenecks that computing at large scale experiences. While their work is not directly relevant to benchmarking graph algorithims, elements of their methodology are useful. For example, they conduct their profiling by randomly sampling from active machines, then levering the Linux Perf suite to collect data. They then tag the observations to link it to the code generating the observed behaviour and load the results into the Dremel database for analysis. They use the Top-Down profiling approach. The Top-Down approach uses the micro-operation queue to classify operations into one of four categories. The patterns of micro-operation occurences drive the characterization of system behaviour. They assert the canonical approach to determining instruction set size is to simulate and then look for the elbow point in the simulation where cache misses drop to zero, and offer an alternative method that samples the real system instead. They have two interesting findings relevant to the HIVE problemset. First, the main reason that they see back-end micro-operation slots being created is to serve data cache requirements. Second, 95 percent of the systems that they analyze use 31 percent or less of their memory bandwidth. Taken together, we can surmise that the size of the data being read from and written to memory is not the bottleneck, it is the latency in waiting for the accesses to occur. In graph processing, because the memory access patterns are not localized, and the processing operations are very simple the latency effect will be exacerbated. Finally, they identify that simultaneous multi-threading is a noted mechanism to improve overall performance assuming that there are a diverse cause of battlenecks. It is not clear whether the parralellizing of graph algorithms will improve or degrade memory latency in graph processing [7].

2.3 Problem Domains and Graph Algorithms

2.3.1 Community Detection. In 2008 Blondel et al characterize what in time becomes known as the Louvian Algorithm. The Louvian Algorithm is a heuristic algorithm to approximate community detection in graphs. It uses an interative 2-step algorithm to maximize the modularity score of communities in a hierarchical manner. The algorithm first assigns each node in the graph to a different community and calculates the improvement in modularity scores. Then it creates a new instance of the graph where changes to the communites increase the modulatiry score. The algorithm terminates when no changes to modularity score result from an iteration. They define community detection algorithms as belonging to one of three categories: (1) Divisive, (2) Agglomerative and (3) Optimization. The Louvian Algorithm is an agglomerative algorithm. They assert that their algorithm is bound by storage, not computation. Assuming that they mean memory when they write storage, it conforms to the expected behviour of graph platforms motivating the HIVE program. They observe that despite a minimal effect on the ending modularity scores, the starting node (and order of execition) has a varying effect on runtime. They were not able to determine why runtime is affected by start node. They assert a linear complexity for their algorithm, but do not present a formal proof, but show emprirically that they achieve superior modularity and runtime results compared to the other algprithms at the time. Runtime and Modularity appear to be suitable metrics for a community detection benchmark. Finally, they postulate further runtime improvements by introducing additional heuristics, like a 'good enough' threshold for modularity [3].

2.4 Graph Benchmarking

In 2010 Richard Murphy and his team from Sandia National Labs introduce the Graph500 dataset as a corollary to the Top500 dataset that tests floating point operations per second (FLOPS) on high performance computers. Graph500's inital goal focuses on creating benchmarks for the search, optimization and edge operations graph kernels (i.e. types of tasks). They specify the parameters to be used in RMAT to generate synthetic graphs, but are unclear on the actual metric they are benchmarking against. In context it appears to be temporal (i.e. time to complete an operation), which they note in their initial experiments is already limited at large scales. Graph 500 is a useful standard but relised on synthetically genrated data and focuses narrowly on a few problems, with simplistic metrics [8].

The 2015 Graphalytics Benchmark from Capotă et. al. aims to produce consistent reporting on graph workloads between all combinations of algorithms, datasets and platforms. It primarily targets distributed and paralell implementations. Their paper claims (but provides no substantive evidence or discussion of) support for evaluating algorithms run on GPUs and knowledge graph analytics. They assert that a good benchmark suite must balance using real-world datasets with designing specific problems to stress the known choke points. After highlighting that the use of real datasets is essential for credibility, they explain how they create their synthetic datasets. Specifically for graph workloads they identify (1) Excessive Network Utilization, (2) Large Memory Footprints, (3) Poor Access Locality and (4) Skewed Execution Intensity. They characterize their datasets by Vertex and Edge Count, Global and Average Cluster Coefficents and Assortativity (The assortativity coefficient is the Pearson correlation coefficient of degree between pairs of linked nodes). They also examine the defree distribution by goodness of fit to several known distributions. The Graphalytics suite supports five algos': (1) General Stats, (2) Breadth First Search, (3) Connected Components, (4) Community Detection, (5) Graph Evolution. They do not measure or report the time taken to extract,

transform and load the graph into memory, or any detail about how it is structured in memory. The Benchmark Suite measures execution time and traversed edges per second (calculated by dividing the execution time by the total number of edges - it is not clear if this will handle algoritms that revisit edges multiple times, and a more robust approach wi required). They assert that software engineering best practices should be applied to any benchmarking effort, and use static code analysis and formal change management to provide quality assurance for their system, justifying our use of TDD in approaching our solution [4].

Beamer et. al. from UC Berkeley introduce the GAP Graph Benchmark in their 2017 paper in response to their percieved shortcomings with the Graph500 benchmark. The authors assert that a benchmark suite should use real and diverse data wherever possible, and when synthetic data is used it must be standardized. They note among the many views of a benchmark suite, that GAP is suited to comparing the performance of hardware on common graph problems. They provide reference implemenatations which is tested across multiple compilers. While the GAP Benchmark paper makes reference to spatial locaility, they do not clearly explain how they mearure the effect of on execution time or traversed edges per second. They address variation in execution time caused by differing start nodes by repeatedly running workloads with different start points. The benchmark is designed to be wider-ranging than Graph500, and as a result has reference workloads for Breadth First Search, Single Source Shortest Path, Page Rank, Connected Components, Betweenness Centrality and Triangle Counting. While Connected components and betweenness centrality have some relation to community detection, and triangle counting is related to subgraph matching none are exact matches for the problem domain that HIVE is specifically pursuing. There is no mention at all of knowledge graph analytics. Given that overall system performance includes the loading and storage in memory of Graphs has it appears to be a gap that these parts of the process are not measured. An examination of how a graph is stored in memory effects its spatial locality during execution appars to be under-explored and so may be worth analyzing further [2].

2.5 Graph Dataset Generation

Chakrabarti, Zhan and Faloutsos' 2004 Recursive Matrix (R-MAT) remains a defacto standard in synthetic graph generation. R-MAT works by a simple mechanism, accepting parameters a, b, c and d, which are numbers between 0 and 1, that must add to 1. Each parameter reflects the probability that a node will be placed in a given quadrant of an adjacency matrix. The placement is done recursively, with each quadrant being subdivided into 4 regions until the base case is reached and the node is assigned. They briefly explain how one could estimate the values of a, b, c and d but it is not clear whether a tool that can analyse a graph and estimate the values of A, B C and D is available, or proven to work. If it does not, this would be a useful contribution. If it does, it will be worth examining their method to enable the generation of realistic datasets for the PiUMA experimentation [5].

REFERENCES

[1] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. 2010. HPCTOOLKIT: tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 685–701. https://doi.org/10.1002/cpe.1553 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1553

[2] Scott Beamer, Krste Asanović, and David Patterson. 2017. The GAP benchmark suite. *arXiv preprint arXiv:1508.03619* (2017). https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Beamer2017.pdf

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008. https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Blondel2008.pdf

[4] Mihai Capotă, Tim Hegeman, Alexandru Iosup, Arnau Prat-Pérez, Orri Erling, and Peter Boncz. 2015. Graphalytics: A big data benchmark for graph-processing platforms. In *Proceedings of the GRADES'15*. 1–6.

[5] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 442–446.

[6] William J Dally, Stephen W Keckler, and David B Kirk. 2021. Evolution of the graphics processing unit (GPU). *IEEE Micro* 41, 6 (2021), 42–51. https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Dally2021.pdf

[7] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 158–169. https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Kanev2015.pdf

[8] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. 2010. Introducing the graph 500. *Cray Users Group (CUG)* 19 (2010), 45–74. http://www.richardmurphy.net/archive/cug-may2010.pdf

[9] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2023. *A Modern Primer on Processing in Memory*. Springer Nature Singapore, Singapore, 171–243. https://doi.org/10.1007/978-981-16-7487-7_7

[10] Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Robert Hundt. 2010. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE micro* 30, 4 (2010), 65–79. https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Ren2010.pdf

[11] J. Weinberg, M.O. McCracken, E. Strohmaier, and A. Snavely. 2005. Quantifying Locality In The Memory Access Patterns of HPC Applications. In *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. 50–50. https://doi.org/10.1109/SC.2005.59

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

Benchmark Suite	Algorithm										Metrics			Imp		Arch		
	STATS	BFS	SSSP	PR	CC	BC	TC	CD	SGM	KGA	ET	TEPS	SLS	Seq	Par	CPU	GPU	DSA
Graph500 [2010][8]		X									X			X		X		
Graphalytics [2015][4]	X	X			X			X		?	X	X		X	X	X	?	
GAP [2017][2]		X	X	X	X	X	X				X			X		X		

Table 1: Summary of Graph Benchmark Datasets.
STAT = Statistics, BFS = Breadth First Search, SSSP = Single Source Shortest Path, PR = PageRank, CC = Connected Components, BC = Betweenness Centrality, TC = Triangle Counting, CD = Community Detection, SGM = Sub Graph Matching, KGA = Knowledge Graph Analytics, ET = Execution Time, TEPS = Travered Edges Per Second, SLS = Spatial Locality Score, Seq = Sequential, Par = Paralell/Distributed, CPU = Central Processing Unit, GPU = Graphics Processing Unit, DSA = Domain Specific Architecture