Fine-grained Power Analysis of Emerging Graph Processing Workloads for Cloud Operations Management

Shuang Song, Xinnian Zheng, Andreas Gerstlauer, Lizy K. John
The University of Texas at Austin, Austin, TX, USA
{songshuang1990, xzheng1, gerstl, ljohn}@utexas.edu

***Abstract*—In modern cloud computing and analytics applications, large-scale data is often represented in the form of graphs. Many recent works have focused on understanding and improving performance of graph processing frameworks. Power consumption, which also serves as a key factor in the deployment and management of graph processing frameworks, has not been extensively studied. In this paper, we demonstrate the use of an online software power estimation tool that is capable of obtaining fine-grained power traces. By leveraging component-level power behavior, we show that static power consumption still constitutes a significant portion of the total power. Moreover, we illustrate the impact of various dynamic voltage and frequency scaling polices on these workloads, and observe that setting the computing node to its maximum frequency can achieve optimal performance and energy consumption. From our analysis on the impact of machine scale-up, we conclude that computing nodes with small number of computing threads consume more energy than the powerful ones. This observation can help cloud administrators on energy-efficient resource allocation.**

I. Introduction

The total amount of digital data stored in the world today is exceeding 4.4 zettabytes, and it is expected to increase ten-fold by the year 2020 [1]. As data volumes are increasing exponentially, graph has been proposed as a concise data structure for representing these massive amount of data. Operations and analytics on large graphs span multiple application domains such as online retail, social applications, and bioinformatics [2]. Over the years, graph processing has become one of the most important applications on modern cloud computing platforms. In order to select and manage computing resources for graph processing applications, we need to understand their underlying performance and power characteristics.

Optimizing graph processing can be achieved from both software and hardware perspective. On the software side, various graph processing frameworks focusing on programmability and performance have been proposed [3], [4], [5], [6], [7], [8], [9], where different programming models, compute engines, and methods of traversing are studied. On the hardware side, domain-specific accelerators dedicated towards graph processing has also been proposed [10], [11] to achieve high performance. These works mostly optimizes graph processing with respect to performance, where power efficiency is not typically considered. However, energy consumption of modern data center has become a critical issue for cloud service providers. In this paper, we characterize the power behavior of various graph processing applications and compared to all the prior works mentioned above, our main contributions are:

1) First, we demonstrate the use of an online component-level power monitoring tool for graph processing workloads on the modern server, which allows cloud service providers to measure the dynamic and static power of each individual core. We show that 33% of the total power still belongs to static power.
2) Secondly, we perform detailed studies on the effect of various dynamic voltage frequency scaling scheme. We show that on-demand power governor does not provide optimal performance and energy efficiency, whereas by simply setting operating frequency to the maximum achieves the best performance with an average of 5.8% energy reduction. This phenomenon indicates the need of better power governors, and validates that the "race to idle" concept [12] still holds for these emerging workloads.
3) Finally, we analyze the scale-up behavior in terms of both performance and energy cost. Our results show that graph processing workloads consume more energy as the total number of available computing threads decreases. This is particularly interesting, as modern cloud computing service providers, such as Amazon EC2 [13], always charge less for less powerful machines, which means they are making less profit on those machines due to the higher energy cost.

We expect such observations to be useful in managing the energy-efficiency and performance of cloud deployments. Potential actions include keeping inactive servers in power-gated mode and racing to finish for total energy savings. Similarly, charging less for less powerful servers may not be a smart policy from cloud service provider's energy cost perspective.

II. Online Digital Power Measurement

A. Tool Overview

In this paper, we employ WattWatcher [14] for online power measurement, which is a toolkit that integrates a number of Linux utilities and McPAT [15] power model with configurable system models and functional unit estimators. As shown in Figure 1, this toolkit can be split into three modules that are Controller, Collector, and Analyzer.

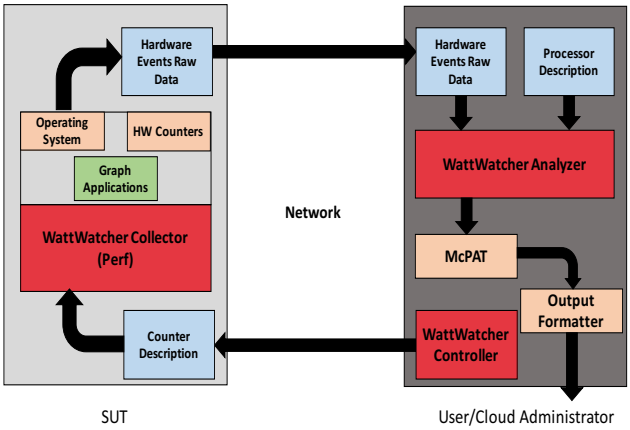


Fig. 1: Overview of WattWatcher Toolkit [14].

Controller: Controller is the interface for any user (like cloud administer) to define the hostname and location of the System-Under-Test (SUT). As WattWatcher is a power estimator based on the hardware event activities, user needs to describe the microarchitectural features (cache layout, number of active CPUs, and operating frequency) and underlying event counters. The Controller uses these statistics to generate an XML file that represents the SUT. The Controller then stores the system configuration and proceeds to launch the Collector.

Collector: The Collector is operating in the SUT, which gathers the runtime information. Perf [16] is the tool that is used to collect fine-grained hardware activities, whose sampling frequency can be pre-defined by user. Perf will only probe the counters defined by user via Controller. For online mode, the Collector periodically outputs the data to the Analyzer for post power estimation.

Analyzer: This is the main module of the toolkit, as it is in charge of converting the raw data obtained from Collector to the power estimations. These dynamic raw data will be combined with system configurations defined in Controller to populate an input file for McPAT. For each captured sample, a corresponding power spread sheet will be generated for future analysis. As this process is done away from the SUT, there is no impact on the SUT's power behaviors.

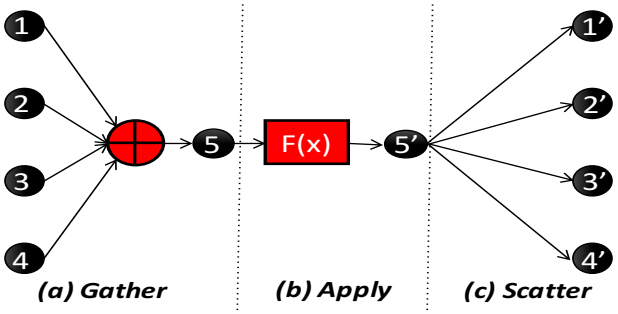


Fig. 2: GAS Computation Model.

B. Power Modeling

The tight correlation between power consumption and hardware counters has been discovered by many Researchers [17]. As we mentioned above, WattWatcher leverages this knowledge by using hardware event counters to estimate a large number of statistics for McPAT. Every microarchitecture needs a mapping file that defines the relationship between its performance counters and backend McPAT event counts. Some of them can be directly measured via available performance counters on the chip. However, others, like the number of reads/writes to an Reorder-buffer (ROB), requires some reasonable assumptions. The mappings in WattWatcher have been build for AMD Piledriver, Intel Haswell, and Intel Sandy Bridge microarchitectures. We add our own mapping function for Intel Ivy Bridge in this work.

C. Advantages Over Other Existing Methods

There are many different approaches have been purposed to either measure or estimate the processor's power cost. However, the WattWatcher we chose has many advantages that makes it suitable for the cloud computing management. Direct measurement via external analog probe can only accurately provide the power data at the coarse-granularity, which reflects entire power consumption of all the devices that draw the energy from the wall outlet. Finer-grained power watching requires destructive shunting of the circuits. Breaking down the power allocation at the core level is almost impossible. In addition, this type of measurement is really hard to expand for large scale systems. Wattwatcher can be easily scaled out, as the users just have to identify the hostnames and locations of the target SUTs. On-chip power/performance counters, such as Intel's Running Average Power Limit (RAPL) counters, provide estimations on the SUT. The overhead of this approach is very low, however, it only reports the power data at the processor or package level. WattWatcher can break down the power cost for each core, and monitor individual components' power behaviors. Offline Curve Fitting is highly used in academia that correlates the performance to the power cost. While this method is often effective, it does require extensive training and calibration. Bias training set or architecture changes will lead extremely inaccurate results. Moreover, it does need the reference power data before training occurs. However, WattWatcher does not suffer from any of these disadvantages.

III. Graph Processing

In this section, we would like to review the advanced techniques deployed in the modern graph processing frameworks, and discuss the state-of-the-art works in this field.

A. Graph Processing Frameworks

Many frameworks have been proposed to improve the performance of graph processing algorithms. GraphLab [3], GraphChi [4], and GraphMat [5] focus on the multithreading performance on a single computing node. Different from these single-node platforms, PowerGraph [6], Grappa [7], GraphX

[8], and PGX.D [9] target improvements of graph processing in a distributed system.

In this paper, we deploy the PowerGraph framework for demonstration. First, let us briefly review some highlights of this framework.

Computation Model: Same as the other state-of-the-art graph processing frameworks, PowerGraph deploys the vertex computation model to express graph algorithms. The computation model consist of three phases, which are gather, apply, and scatter (GAS), as shown in the Figure 2. Logically, each vertex needs to iterate through these three steps of a program independently of each other with barrier to enforce the correctness and synchronization. During the gather phase, the graph engine performs a map/reduce operation on the edges and adjacent vertices of vertex v. The results from reduction will go to the apply stage. The current information in v and reduced result will be merged to compute a new value. Finally, this new value will be scattered out to the adjacent vertices and used in the next GAS iteration.

Graph Engine: To iterate through the GAS phases, PowerGraph employs two different computing engines, which are synchronous and asynchronous engines. The synchronous engine guarantees the steps of the vertex-centric algorithms via using strict synchronization barriers. Alternatively, the asynchronous engine are more flexible, as it allows vertices to run out of synchronization. However, it does deploy the fine-grained locking to maintain data consistency. Intuitively, these two engines will result in different level of performance boost/loss. Xie et al. [18] makes a comprehensive performance comparison on these two, and illustrates the performance of the two engines varies significantly with different graph algorithms, input graphs, and many other factors. **Coloring** is the application that uses the asynchronous engine in our experiment.

B. Architectural Improvements

Besides the platform-level work mentioned above, many researchers also argue for the optimizations on the hardware level for graph processing applications. Except those single-node graph platforms, such as GraphChi, almost all the distributed ones keep the graph data in the memory to avoid the disk I/O overhead during execution. Ahn et al. [10] purposes to alleviate conventional concept of processing-in-memory (PIM) to design a programmable PIM accelerator that can achieve memory-capacity-proportional performance for large scale graph processing. Graphicionado [11] graph processing accelerator exploits both data structure-centric datapath specialization and memory subsystem specialization, which improves the inefficiencies of general-purpose CPUs. However, none of these work deep dive into the power consumptions of current server-level CPUs. Our work presents the dynamic power behaviors of several popular graph analysis applications on modern server CPUs at a fine-granularity. This can provide insights for both hardware designers to optimize the current CPUs' power efficiency and for cloud administers to better manage/control the resources for graph processing applications.

IV. Experiment Setup

This section introduces our experimental setup, including the machine configurations, data sets, and graph applications used in the evaluation. Our experiments are performed on an Intel Ivy Bridge E5-2430 v2 processor with 12 computing cores (six physical cores with multithreading support), 64 GB DRAM and 1TB hard drive. In order to study the scale-up behaviors, we manually enable and disable compute cores to form six different configurations with on-demand power governor. To analyze the effects of different dynamic voltage and frequency policies, we perform the studies of four governors, which are on-demand, performance, powersave, and userspace. Different policies are executed on using all 12-cores. Various graph datasets used in our experiments are shown in Table I. Their total memory usage varies from 40 Megabytes to over 1 Gigabyte with diverse edge density. The edge density factor specifies the underlying sparseness of the graph.

TABLE I: Real world graphs [19].

Name	Vertices	Edges	Footprint	Edge Density Factor
amazon	403,394	3,387,388	46MB	2.004
citation	3,774,768	16,518,948	268MB	2.169
social_network	4,847,571	68,993,773	1.1GB	1.950
wiki	2,394,385	5,021,410	64MB	2.478

We selected four popular graph applications from various machine learning and data mining (MLDM) applications. Those applications are briefly described as follows:

Pagerank: The Pagerank algorithm [20] is a method to measure the importance of web pages based on their link connected. Its main use is to compute a ranking for every website in the world. This algorithm is defined as:

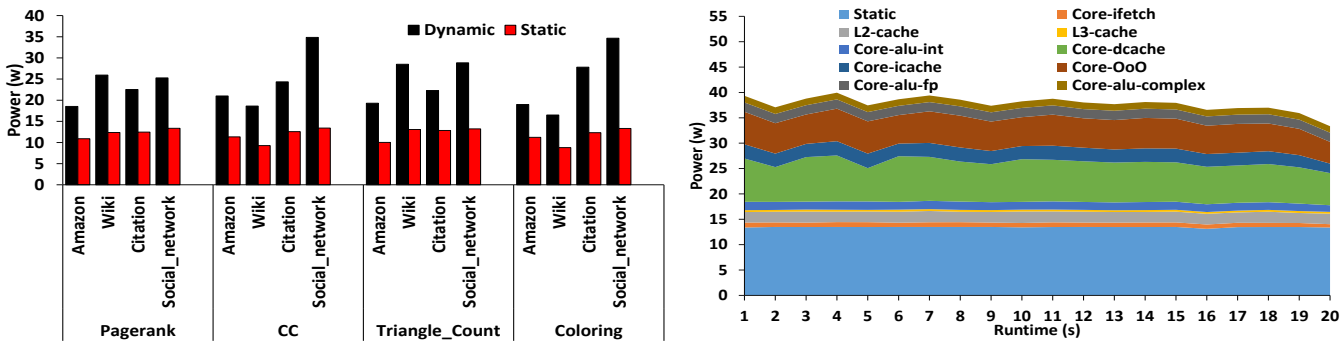
$$PR(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}. \quad (1)$$

Here, d is the damping factor and N is the total number of pages. B_u is the set of pages. $L(v)$ represents the number of outbound links on page v.

Coloring: The Coloring application is a special case of graph labeling. It attempts to color the vertices with different colors such that no two connected vertices share the same color. In PowerGraph, this application is implemented to color directed graphs, and count the total number of colors in use.

Connected Component (CC): The Connected Component algorithm is designed to count fully connected subgraphs in which any two vertices are connected by a path. The algorithm counts connected components in a given graph, as well as the number of vertices and edges in each connected component.

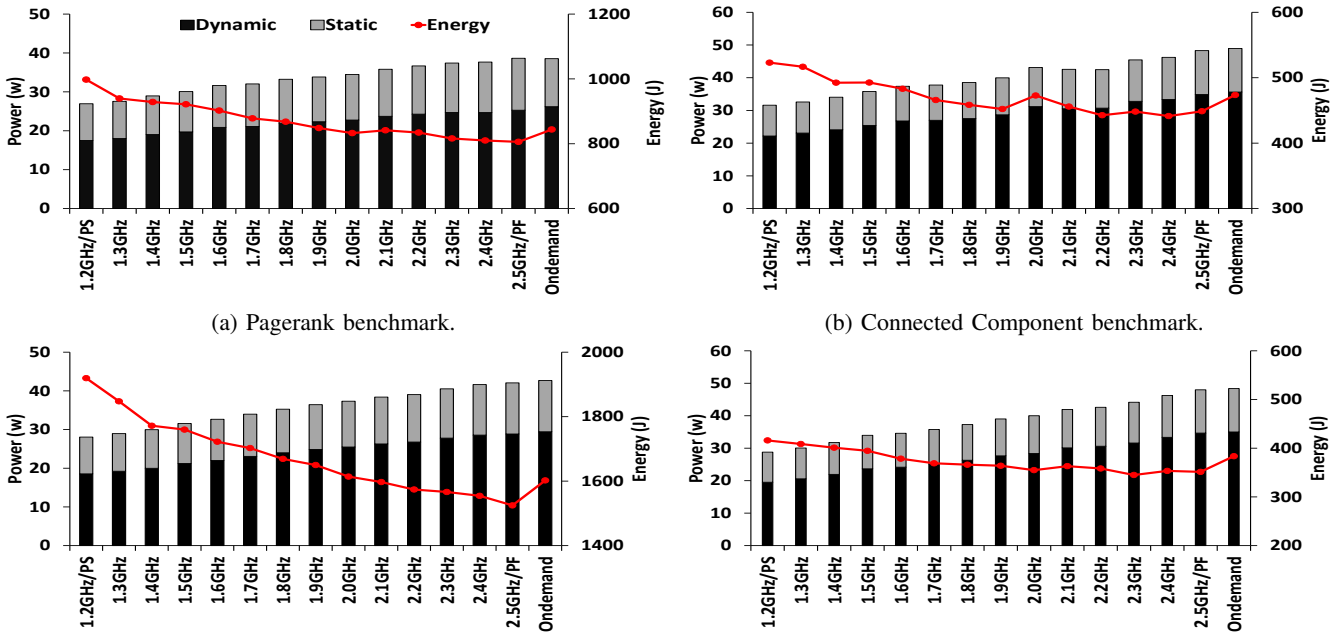
Triangle Count (TC): Each graph triangle is a complete subgraph formed by three vertices. The Triangle Count application counts the total number of triangles in a given graph, as well as the number of triangles for each vertex. The number of triangles of a vertex indicates the graph



(a) Dynamic power vs. static power.

(b) Online power profiling.

Fig. 3: Dynamic/Static power comparison and online power monitoring for pagerank benchmark with social network graph



(a) Pagerank benchmark.

(b) Connected Component benchmark.

(c) Triangle Count benchmark.

(d) Coloring benchmark.

Fig. 4: Performance and power consumption comparison among various operating frequencies and power governors.

connectivity around that vertex. The application implemented in PowerGraph maintains a list of neighbors for each vertex in a hash set. It counts the number of intersections of vertex u's and vertex v's neighbor sets for every edge (u,v).

V. Evaluation

A. Dynamic vs. Static Power

In this section, we demonstrate the results of using WattWatcher for emerging graph processing workloads. As we mentioned before, WattWatcher has the capability of being allocated to any individual or multiple SUTs that cloud administrator wants to monitor. The results shown in this section are captured from the SUT. As illustrated in Figure 3, the dynamic and static power consumption can be measured via WattWatcher, where we observe that the static power in the modern server processor still contributes 33% of the total power on average. Therefore, the processor node would require circuit-level techniques such as power-gating in order to minimize static energy consumption. In addition, we demonstrate that different graphs data can cause the same graph application to consume different amount of dynamic power. The relationship between dynamic power depends upon the characteristics of the underlying graph, such as the average edges per vertices and edge distribution. In Figure 3b, we see the component-level dynamic power breakdown of pagerank, where out-of-order execution and L1-D cache consume 23.9% and 32.2% of the total dynamic power respectively. Server architects can optimize these two components' energy consumption to minimize the total energy cost of the processor.

B. Governor Settings

This section compares the energy and power consumption for various operating frequencies and power governors. The performance governor sets the machine to its highest operating

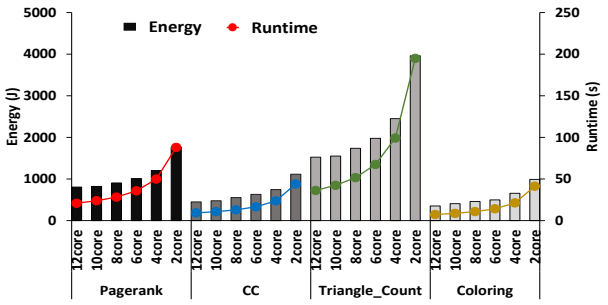


Fig. 5: Processor scale-up study for pagerank benchmark with social network graph.

frequency, whereas the powersave governor sets the frequency to the lowest. The Userspace governor allows the user/cloud administrator to manually controls the operating frequency. The Ondemand power governor is designed to save both dynamic and static power by setting the processor operating frequency based on the current utilization. During the low utilization phase, it will reduce the voltage and frequency and vice versa. The negative effect of this dynamic voltage and frequency scaling method is caused by the post-reactions. For example, when the frequency/voltage are just reduced to the lower level and the processor's utilization goes up, then the frequency and voltage has to be increased for the next period. This false reaction results in performance loss. In order to comprehending our experiments, we employ both synchronous and asynchronous graph engines. Pagerank, CC, and Triangle Count are operating on the synchronous engine, while Coloring is executed by asynchronous engine. However, the power/energy reactions of synchronous/asynchronous engines are very similar with different power governors and frequency settings. As shown in Figure 4, the minimal energy consumptions of all four application locates around the maximum operating frequency bar (the performance governor). Setting the machine to operate at its highest frequency point can save 5.8% energy cost on average and simplify the complexity of cloud management. Therefore, we encourage the cloud providers to follow the "race-to-idle" concept when servicing graph processing applications.

C. Scale-up Analysis

As modern data centers trending towards a more heterogeneous composition of compute resources, selecting suitable machines for emerging workloads, such as graph processing applications, becomes a critical problem. In this section, we demonstrate the performance and energy costs of machines with various amount of compute cores. Similar to our experiment, the Amazon AWS machines in the same category have same type of processor with different amount of virtual cpus. To mimic a heterogeneous compute environment, we manually turn on and off cores. As illustrated in Figure 5, the average power is monotonically decreasing with respect to the number of compute cores. However, the minimal energy consumption is achieved at the machine with the 12-core. This phenomenon is caused by significant performance degradation of the less powerful machine. Interestingly, the modern cloud service providers, such as Amazon EC2 [13], charges less for less powerful compute nodes. In fact, this leads to an increase in the energy budget from the service provider's perspective.

VI. Conclusion

Graph Processing applications are emerging as an extremely important class of workloads during the era of big data. Characterizing the power behavior of representative graph processing applications can signicantly help cloud service providers to select and manage the existing computing resources. In this paper, we demonstrate the use of an online fine-grained power watching toolkit for graph processing applications. The toolkit can allow cloud operators to probe the power consumptions of underlying hardware dynamically. In addition, this toolkit can be easily distributed in the large-scale data center. Besides this, we perform a comprehensive analysis on the performance and energy cost of various power governors. Compared to the ondemand governor, operating at machine's highest frequency (performance governor) can reduce an average of 5.8% energy. Lastly, we study the scale-up effects to help cloud administrators select the optimal computing resource. Based on our results, we observe that machines with higher computing slots can finish the task signicantly faster and achieve minimal energy consumption, although the average power cost is higher.

VII. Acknowledgments

This work was partially supported by Semiconductor Research Corporation Task ID 2504, and National Science Foundation grant CCF-1337393. The authors would also like to thank the Texas Advanced Computing Center (TACC) and Amazon for their donation of the EC2 computing resources used in this work. Any opinions, findings, conclusions, or recommendations are those of the authors and do not necessarily reflect the views of the National Science Foundation or the other funding agencies.

References

[1] C. Baru, M. Bhandarkar, R. Nambiar, *et al.*, "Setting the direction for big data benchmark standards," in *Selected Topics in Performance Evaluation and Benchmarking*, pp. 197–208, Springer, 2013.

[2] K. Ammar and M. T. Özsu, "Wgb: Towards a universal graph benchmark," in *Advancing Big Data Benchmarks*, pp. 58–72, Springer, 2014.

[3] Y. Low, J. E. Gonzalez, A. Kyrola, *et al.*, "Graphlab: A new framework for parallel machine learning," *UAI*, pp. 340–349, 2010.

[4] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc," in *Conference on Operating Systems Design and Implementation (OSDI)*, pp. 31–46, USENIX Association, 2012.

[5] N. Sundaram, N. Satish, M. M. A. Patwary, S. R. Dulloor, M. J. Anderson, S. G. Vadlamudi, D. Das, and P. Dubey, "Graphmat: High performance graph analytics made productive," *Proc. VLDB Endow.*, 2015.

[6] J. E. Gonzalez, Y. Low, H. Gu, *et al.*, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 17–30, USENIX Association, 2012.

[7] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin, "Latency-tolerant software distributed shared memory," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, USENIX Association, 2015.

[8] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, USENIX Association, 2014.

[9] S. Hong, S. Depner, T. Manhardt, *et al.*, "Pgx.d: A fast distributed graph processing engine," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 58:1–58:12, ACM, 2015.

[10] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, ACM, 2015.

[11] H. Tae Jun, W. Lisa, S. Narayanan, S. Nadathur, and M. Margaret, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *Proceedings of the 49th International Symposium on Microarchitecture*, MICRO-49, ACM, 2016.

[12] S. Albers and A. Antoniadis, "Race to idle: New algorithms for speed scaling with a sleep state," *ACM Trans. Algorithms*, 2014.

[13] "Amazon EC2." http://aws.amazon.com/ec2. Accessed: 04-16-2015.

[14] M. LeBeane, J. H. Ryoo, R. Panda, and L. K. John, "Watt watcher: Fine-grained power estimation for emerging workloads," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2015 27th International Symposium on*, IEEE, 2015.

[15] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pp. 469–480, ACM, 2009.

[16] "perf: Linux profiling with performance counters." https://perf.wiki.kernel.org/.

[17] W. L. Bircher and L. K. John, "Complete system power estimation using processor performance events," *IEEE Trans. Comput.*, 2012.

[18] C. Xie, R. Chen, H. Guan, B. Zang, and H. Chen, "Sync or async: Time to fuse for distributed graph-parallel computation," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, ACM, 2015.

[19] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data. Accessed: 04-16-2015.

[20] L. Page, S. Brin, R. Motwani, *et al.*, "The pagerank citation ranking: Bringing order to the web.," Technical Report 1999-66, Stanford InfoLab, 1999.