The Energy Case for Graph Processing on Hybrid CPU and GPU Systems

Abdullah Gharaibeh, Elizeu Santos-Neto, Lauro Beltrão Costa, Matei Ripeanu
The University of British Columbia
{abdullah,elizeus,lauroc,matei}@ece.ubc.ca

ABSTRACT
This paper investigates the power, energy, and performance characteristics of large-scale graph processing on hybrid (i.e., CPU and GPU) single-node systems. Graph processing can be accelerated on hybrid systems by properly mapping the graph-layout to processing units, such that the algorithmic tasks exercise each of the units where they perform best. However, the GPUs have much higher Thermal Design Power (TDP), thus their impact on the overall energy consumption is unclear. Our evaluation using large real-world graphs and synthetic graphs as large as 1 billion vertices and 16 billion edges shows that a hybrid system is efficient in terms of both time-to-solution and energy.

Categories and Subject Descriptors
C.1.3 [**Processor Architectures**]: Other Architecture Styles – *Heterogeneous (hybrid) systems.* G.2.2 [**Discrete Mathematics**]: Graph Theory – *Graph Algorithms.*

General Terms
Measurement, Performance, Experimentation

Keywords
Graphics Processing Units, GPUs, Hybrid Systems, Graph Processing, Energy Efficiency, Energy Delay Product

1. INTRODUCTION
Efficient graph processing requires the whole graph to be present in memory. Large real graphs, however, can occupy gigabytes to terabytes of space; for example, a snapshot of the Twitter follower network has over 500 million vertices and 100 billion edges, and requires at least 0.5TB of memory. As a result, the most commonly adopted solution to cost-efficiently process massive scale graphs is to partition them and use shared-nothing cluster systems [21].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
IA^3 *'13*, November 17 2013, Denver, CO, USA
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2503-5/13/11…$15.00.
http://dx.doi.org/10.1145/2535753.2535755

Similar to Rowstron et al. [26], we start from the observation that, today, more efficient solutions are achievable: it is feasible to assemble *single-host* graph processing platforms that aggregate 100s of GB to TBs of DRAM and massive computing power [26, 28] all from commodity components and for a relatively low budget. Compared to clusters, single-node platforms are easier to program, and promise to offer better performance and energy efficiency for a large class of real-world graph problems. In fact such single-node graph processing platforms are currently being used in production: Twitter's 'Who To Follow' (WTF) service, which uses the follower network to recommend connections to users, is deployed on a single node [12].

At the same time, GPU-acceleration emerged as an appealing technique and has successfully been applied to regular (e.g., linear algebra) and irregular (e.g., sequence alignment [11]) processing problems, including graph processing [15]. In the context of graph processing, the key advantage GPUs bring is massive hardware multithreading: GPUs support orders of magnitude more in-flight memory requests while still performing useful work and thus masking memory access latency – the major performance hindrance for graph processing problems. Although current GPUs have limited memory, previous work demonstrate that large-scale graphs can still benefit from GPU acceleration by partitioning the graph to be processed concurrently on the CPU and the GPU [9, 10].

Although nothing prevents manufacturers from adding more memory to GPUs to solve this memory limitation, it is unclear how using GPUs affects power consumption. On the one hand, GPUs are known to have higher FLOP/watt rate than CPUs. However, graph processing workloads are memory bound, and hence do not benefit from this characteristic. Moreover, GPUs have high thermal design power (TDP) (~200W), typically double that of CPUs which may make an accelerated solution efficient in terms of time-to-solution but not in terms of energy. On the other hand, GPU-acceleration offers tangible performance benefits for workloads that fit their computational model. This allows a faster 'race-to-idle', enabling power savings that are sizeable for newer GPU models which are power-efficient in idle state (as low as 25W [24]).

This work builds on our previous work that demonstrates that GPUs can be effectively used to accelerate graph processing [9, 10]. Here we evaluate whether the performance benefits of the techniques we have proposed translate in the power and energy space as well. Concretely, we focus on the following high-level research questions:

- *Is it energy-efficient to partition the graph to be processed concurrently on a GPU and a CPU?*
- *Given a graph/algorithm workload and a fixed-power or energy budget, what is the (empirically determined) optimal balance between traditional and massively-parallel processors?*
- *What is the impact of increasing the graph scale on energy consumption and efficiency?*

Addressing these questions is important to inform the design of graph workload partitioning solutions that aim to optimally harness heterogeneous computing platforms. In the context of current hardware trends, as the cost of energy continues to increase relative to the cost of silicon, future systems will host a wealth of different processing units. In this hardware landscape, the key issue will become partitioning the workload and assigning the partitions to (possibly, a subset of) the existing processing elements where the workload can be executed most efficiently in terms of power, energy, or time.

Contributions. We use experiments to show that such hybrid systems bring advantages even today. Using large-scale real-world and synthetic workloads, we show that: First, GPU acceleration, i.e., adding new components, can contribute to better time-to-solution and can reduce the energy footprint for graph processing. Second, when maintaining the number of components constant, a hybrid (one CPU and one GPU) system is generally more energy efficient than a symmetric dual-CPU one while drawing similar power. Third, hybrid systems are attractive at scale, that is, both when increasing graph size and increasing the number of GPUs. Finally, to the best of our knowledge, this is the first work to evaluate graphs as large as 1 billion vertices and 16 billion edges on a single-node commodity machine.

2. BACKGROUND

This section briefly discusses opportunities and challenges of graph processing on hybrid systems (§2.1), introduces TOTEM: the framework we developed that facilitates implementation of graph algorithms on hybrid systems (§2.2), and places this work in the context of our own past work (§2.3).

2.1 Graph Processing on Hybrid Systems

***The opportunities*:** GPUs not only have much higher memory bandwidth than traditional CPU processors, but also massive, hardware-supported multithreading that can mask memory access latency – by enabling orders of magnitude more in-flight memory requests.

Our own previous work demonstrates that partitioning graph workloads and processing them concurrently on both the CPU and the GPU offers tangible benefits [9]. Moreover, properly mapping the graph-layout and the algorithmic tasks between the CPU(s) and the GPU(s) enables exercising each of these computing units where they perform best: CPUs for fast sequential processing and GPUs for the bulk parallel processing.

The challenges: Large-scale graph processing poses two major challenges to hybrid systems: First, a *large memory footprint* since efficient graph processing requires the whole graph to be in memory. This is a major challenge for GPUs considering

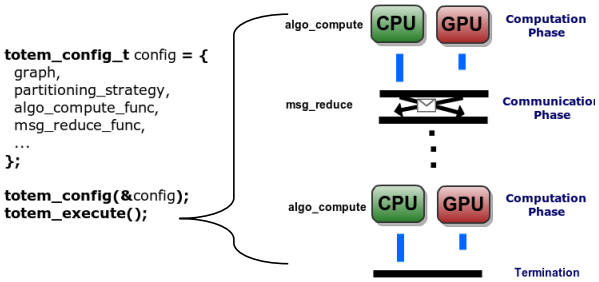

Figure 1: BSP model and its implementation in TOTEM [10]. The framework is customized for a specific algorithm by implementing callback functions; some are invoked on each partition concurrently (e.g., alg_compute_func during the computation phase).

that device memory is limited in space (roughly an order of magnitude less than the host). Second, an *irregular and data-dependent memory access pattern* which significantly reduces caching effectiveness on CPUs and increases thread divergence on GPUs. Additionally, most graph algorithms perform little computation, lowering the chance to hide memory access latency; thus, the major overhead for graph processing is generated by fetching the state of vertices (or edges) from memory, i.e. graph processing applications are memory bound.

Finally, it is unclear whether the advantage GPUs offer in terms of high processing rates can be preserved for irregular, memory-bound problems like graph processing as partitioning the graph over different memory spaces (the host and the accelerators) may lead to communication overheads over the PCI-E bus that render GPUs ineffective.

2.2 The Totem Framework

We have designed and implemented the TOTEM[1] framework [9] to enable developing graph algorithms for hybrid (CPU + GPU) platforms. TOTEM adopts a Bulk Synchronous Parallel (BSP) computation model [30]. In a nutshell, BSP processing performs in rounds of three phases executed in order (Figure 1): *computation* (CPUs and GPUs asynchronously process their local partitions), *communication* (processors exchange messages via boundary edges), and *synchronization* (guarantees the delivery the messages before the start of a new round). We note that since TOTEM is algorithm agnostic, better performance can likely be obtained using algorithm-specific implementations.

Figure 1 shows how BSP is used. The developer specifies a set of callback functions. TOTEM partitions the graph, transfers the partition to the GPU, and launches the computation according to the callbacks provided by the developer. Both processors, CPU and GPU, execute the user defined algo_compute_func on their own partitions concurrently (computation phase). Then, TOTEM uses msg_reduce_func to aggregate the messages sent to the same remote vertex, and bulk transfer messages to the destination processor (finishing a round/ BSP superstep). Invocations of algo_compute_func and msg_reduce_func continue for each round until the algorithm converges to termination.

[1] TOTEM is an open source project, the code can be found at: http://netsyslab.ece.ubc.ca

Using the BSP model offers two major advantages. First, it provides a simple framework to implement graph algorithms on distributed memory systems. Second, and more importantly, it allows circumventing the high-latency of the PCI-E bus by batching message transfers in the communication phase. For details on TOTEM we refer the reader to [9].

2.3 Relationship with Our Prior Work

We have used TOTEM to demonstrate the feasibility of graph processing on hybrid systems [9], and explored the effectiveness of various graph partitioning strategies for performance [10]. This work expands our investigation to include an energy evaluation.

Compared to past published work, this work also explores new optimizations (for both TOTEM and the various graph algorithms implemented on top of it), evaluates on newer CPU and GPU models, and uses real-world and significantly larger synthetic graphs.

We briefly present here three of the new optimizations done for this work that are generic for TOTEM: (To maintain a consistent 'storyline' focused on energy, we do not evaluate in detail these optimizations here, yet, for each of them, we present a situation which highlights its impact).

- *Improved load balancing between the CPU and the GPU for large graphs.* The GPU's limited memory space constrains the size of the offloaded partition. Current GPUs, which support at most 6GB of memory, can host at most 1.5Billion edges considering 4bytes edge identifiers (note that this estimate does not take into account the space needed for the vertices' state, hence this limit is even lower). This is a major challenge when targeting multi-billion scale graphs. To enable offloading a larger partition to the GPU, we allocate part of the state on host memory and map it into the GPU's address space. The tradeoff is an extra communication overhead over the high latency PCI-E bus. We reduce this overhead by taking the following measures: First, we avoid the high latency of the bus by restricting the use of mapped memory to allocate the part of the state that is (*i*) read-only, and (*ii*) can be accessed sequentially in batches; particularly, we used mapped memory to allocate the edges array since we assume static graphs. Second, we maximize transfer throughput by ensuring that the edges of a vertex are read in a coalesced manner when the vertex iterates over its neighbors. Finally, a side-effect of using mapped memory is that it naturally supports overlapped communication of a vertex's edge list with the computation of another vertex.

  This optimization speeds up the overall computation by up to 2x for hybrid configurations for the two largest workloads we use in our evaluation (i.e., RMAT29 and RMAT30, which we describe in more detail in the next section). This is because, for such large workloads, this optimization allows increasing the size of the offloaded partition from as little as 5% of the original graph (if we are to allocate the GPU partition state on device memory only) to up to 60%.

- *Improved load balance across GPU threads.* Early work on graph processing on GPUs employed parallelism across vertices [14]; however, this approach creates load-imbalance among threads and can lead to GPU underutilization since some vertices, in particular the high-degree ones, require more work than others. To address this problem, Hong et al. [15] propose to parallelize processing not only across vertices, but also across the edges of a vertex. Hong et al. do this by statically allocating a block of threads for each vertex to process its edges in parallel. Although this approach improves performance, it does not completely address the problem: the fact that threads were being statically allocated results in some vertices being assigned more threads than they require (e.g., vertices with degree less than the configured value), while others will be assigned less threads. This is especially an issue for scale-free graphs where the degree varies considerably across vertices. We address this problem by using a new feature introduced recently by CUDA: dynamic parallelism, which allows a GPU kernel to create work from within the GPU. We employ this feature to create threads dynamically based on vertex degree for each group of vertices with similar degree, and hence improving GPU utilization.

  This optimization speeds up the GPU computation by up to 80% for the Twitter workload (which is more unbalanced than the RMAT workloads). While this speedup does not translate to a performance gain for the whole computation (as the CPU takes longer), this optimization allows the GPU to run faster to idle, and hence reduces energy consumption.

- *Improved vertex access locality*. Although graph processing is known to have random access pattern, vertex placement in memory can still improve performance. In particular, placing vertices in memory close to their neighbors leads to important reduction in TLB cache misses on the CPU. We employ the techniques used for graph compression [4, 6] to determine the order by which the vertices are placed in memory for real-world graphs.

  This optimization speeds up the performance of CPU kernels (labeled 2S in plots) by up to 2x for both real-world workloads. As mentioned above, this was due to a major reduction in TLB misses, which we measured using hardware counters via the *OProfile*[2] profiling tool. This speedup is reflected on the performance of both, CPU only and, to a lesser extent, hybrid configurations.

3. EXPERIMENT SETUP

Workloads. We use real-world and synthetic graphs (Table 1). Note that the memory footprint of all workloads is larger than the memory space available on a single GPU.

R*eal-world graphs:* We use two of the largest real-world graphs publicly available: a snapshot of the Twitter network, and a crawl of about 100 million pages from the .uk domain.

S*ynthetic graphs:* Since the real-world graphs we have access to are still limited in scale, we also use large synthetic graphs

[2] The tool can be found at: oprofile.sourceforge.net

Table 1: Workload Characteristics

Workload	|V|	|E|
Twitter [19]	41M	1.5B
UK-Web [5]	105M	3.7B
RMAT27	128M	2.0B
RMAT28	256M	4.0B
RMAT29	512M	8.0B
RMAT30	1,024M	16.0B

generated in the same way as those for the Graph500 challenge[3]: Recursive MATrix (RMAT) process [7] with the following parameters: (A,B,C) = (0.57, 0.19, 0.19) and an average vertex degree of 16. We use the SNAP [29] network analysis library to generate the graphs.

Benchmarks. We evaluate two graph algorithms to stress the platform and TOTEM in different ways that are representative for the two ends of the computation-to-communication ratio spectrum. First, *Breadth-First Search* (BFS), a traversal-based algorithm that computes the shortest distance in an unweighted graph. Second, *PageRank* produces a ranking of the vertices as described in [25]. Note that *PageRank* has a higher compute-to-memory access ratio than *BFS* and, unlike BFS, its computational demands are stable for all computation rounds (in BFS only the 'frontier' vertices are involved in each round and the frontier size varies). A more detailed description of the algorithms and their implementations using TOTEM can be found in [10].

Measuring Power. We measure power at the outlet using a WattsUP meter which collects samples at one second intervals [31]. To get a representative measurement of the energy consumption, we run each experiment for 10 minutes (e.g., repeating BFS searches). For accuracy, CPU-only experiments are conducted after removing the GPUs from the machine (as the GPU draws power from the PCI-E bus as well).

Testbed Characteristics. We use a host with state-of-the-art CPU and GPU models (Table 2). To characterize the power consumption of the machine we use simple compute and memory intensive kernels. Figure 2 shows the power consumption at idle, then when stressing one and both CPUs (listed as 1S and 2S in the plot), then the memory, and then each of the two GPUs. The high idle power consumption, which includes idle power of CPUs and RAM only, is mainly caused by the sizeable amount of available RAM (256GB). Note that the two CPUs consume less power than the DRAM, and less than one GPU. We highlight two points: (*i*) at peak load a significant share of the power is consumed by DRAM, and (*ii*) when loaded, GPUs consume significant power compared to other system components.

Table 2: Machine characteristics: two Xeon 2560 processors and two Tesla Kepler K20 GPUs (connected via PCI-E 2.0 bus, which has a measured bandwidth of 5 GB/s). Total amount of host memory is 256GB.

Characteristic	Sandy-Bridge (Xeon 2650)	Kepler (K20)
Proc. Count	2	2
Cores / Proc.	8	13
Hardware Threads / Core	2	192
Frequency / Core (MHz)	2000	705
LLC / Proc. (MB)	20	2
Main Memory / Proc. (GB)	128	5
Memory Bandwidth / Proc. (GB/s)	52	208
TDP / Proc. (Watts)	95	225

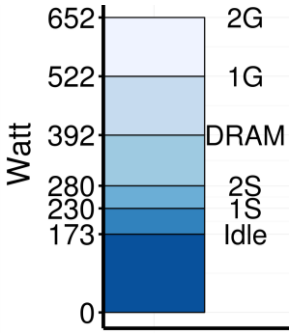

Figure 2: Power characterization of the testbed. The characterization is obtained by incrementally stressing the different components of the system. Note that the GPUs are removed from the system when characterizing only the host components. Finally, "Idle" measures the idle power of the system without the GPUs.

Metrics. To address the research questions, we use four performance and energy metrics: *i)* raw processing rate measured in Traversed Edges Per Second (TEPS)[4] (§4.1), *ii)* power consumption in Watts (§4.2), *iii)* power-normalized processing rate in TEPS/Watt similar to GreenGraph500 (§4.3), and, *iv)* the energy-delay product, a metric biased for low-time-to-solution while taking into account the energy cost (§4.4).

4. EXPERIMENTAL RESULTS

This section uses the metrics above to compare the characteristics of four hardware configurations for all workload/benchmark combinations. We start (§4.1) by focusing on the time-to-solution aspect: first, by presenting the performance of a traditional configuration that uses two processor sockets (labeled as *2S* in the plots) to a configuration that replaces one of the processors to a GPU (*1S1G*). We then add data for configurations that use both processors and add one GPU (*2S1G*) or two GPUs (*2S2G*).

The experiments presented in §4.1 support our claim that, from a time-to-solution perspective, GPU acceleration is an appealing solution for graph processing. The remaining subsections explore the power (§4.2), and energy aspects (§4.3 and §4.4).

[3] The RMAT graphs are described by the log base 2 of the number of vertices (e.g., RMAT30 graph has 2^{30} vertices). Unlike in the Graph500 challenge, the graphs are directed.

[4] TEPS became the accepted performance metric for BFS (see Graph500 benchmark). The corresponding TEPS for PageRank is computed by dividing the number of edges in the graph by the mean time per PageRank iteration (since, in each iteration, each vertex accesses the state of all its neighbors).

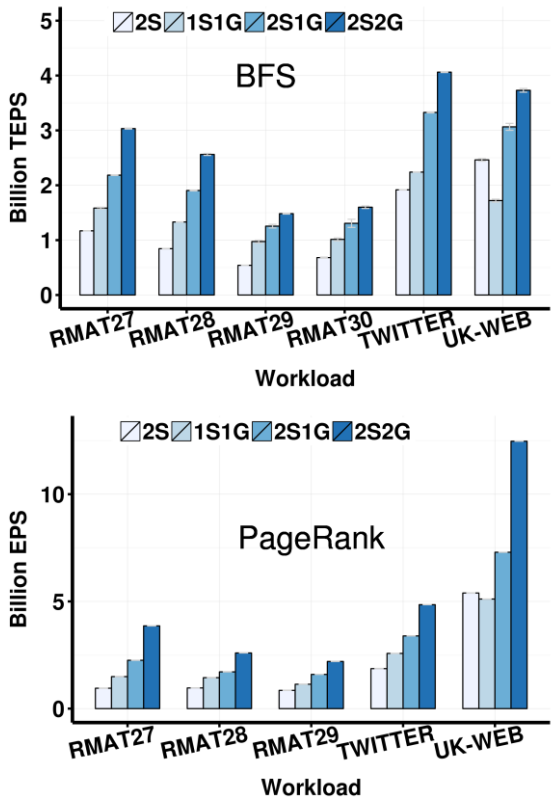

Figure 3: *BFS* (top) and *PageRank* (bottom) processing rates (the higher the better) for different workloads and hardware configurations. The hardware configurations are presented in the following format: *xS yG*, where *x* is the number of CPU sockets (processors) used, while *y* represents the number of GPUs. Experiments for the configuration with a single socket (i.e., *1S1G*) were performed by binding the CPU threads to the cores of a single socket. For PageRank, experiments with RMAT30 workload are not shown as it does not fit in memory (the state required by PageRank is larger than that for BFS).

4.1 Raw Performance

Figure 3 shows BFS (*top*) and PageRank (*bottom*) processing rates as averages over 64 runs. Error bars present the 95% confidence interval, in most cases too narrow to be visible.

First, it is important to stress that the performance for the CPU-only configuration (2S – for two processor sockets used) is comparable to the best reported numbers on similar CPU models [16, 28]. This increases our confidence that we compare to a meaningful baseline.

Second, the figures show that: rather than employing a second CPU, it is more performance efficient to use a GPU if available. Apart for the UK-WEB graph, the hybrid (1S1G – one socket and one GPU) system performs faster than the dual-socket system (2S). This is important considering the fact that the graphs are large, and the GPU has only limited device memory, which constrains the size of the partition offloaded (e.g., in the case of the UK-WEB graph, only 25% of the total number of edges and vertices of the graph fit in the memory of a single GPU).

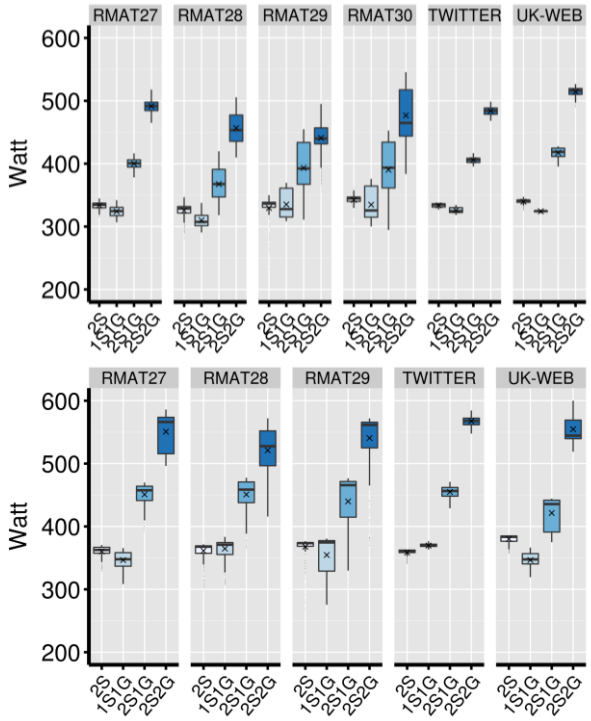

Figure 4: Power consumption (the lower the better) for *BFS* (top) and *PageRank* (bottom). The upper and lower "hinges" of the boxplot correspond to the first and third quartiles. The middle line corresponds to the median. The whiskers extend from the lowest data point within 1.5 IQR of the lower quartile, to the highest data point within 1.5 IQR of the upper quartile (IQR is the Interquartile Range, which is the distance between the first and third quartiles). The mean is shown as a cross. Note the y-axis starts at 200W.

Finally, the figures uniformly show the ability of the hybrid system to harness the extra processing elements for both benchmarks. Most notably, when adding a second GPU, a larger portion of the graph can be offloaded, and hence better performance is obtained.

4.2 Power Consumption

We next turn our attention to power consumption with two key goals in mind: firstly, we aim to understand the degree to which additional processing elements lead to additional power consumption (and how this relates to their TDP rating), and secondly, we aim to characterize the variability in power drawn during processing.

Figure 4 shows the system power consumption under different benchmark/workload combinations. To better illustrate the variation in power consumption during execution, the data is presented as boxplots.

The main differentiating factor in terms of power consumption is the hardware configuration (i.e., the number and type of processing elements used). A second factor is the workload (i.e., PageRank draws more power than BFS as the former not only stresses the memory, but also the FPU pipeline). Finally, we note that there is no major power difference across workloads

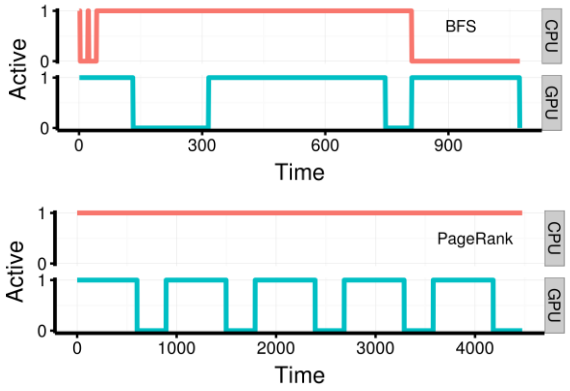

Figure 5: CPU/GPU active/idle state while processing an RMAT27 graph on a 2S1G setup (time is in milliseconds) for BFS (*top*) and PageRank (*bottom*). For BFS, the 'frontier' evolves in unpredictable ways, which results in having a processing element active in specific rounds and not in others. For PageRank, the GPU finishes execution before the CPU in each execution round.

for the same hardware configuration and benchmark combination.

We observe that, although the hybrid 1S1G configuration has higher TDP rating, it draws comparable, sometimes lower, power to the symmetric configuration 2S, which has the same number of processing elements. Also, while adding GPUs to the 2S configuration increases power drawn, the increase is well below the TDP of the GPU. Adding a GPU increases power by ~100W, which is ~50% of the GPU's TDP. The reason is that the GPUs finish processing their partition first and go in an energy-efficient idle state that consumes only a fraction of their peak power (25W) (see Figure 5).

The hybrid configurations generate more variation in power consumption than processing on the CPU only. This is because, for some workloads, the computation is unbalanced between the CPU and the GPU(s). There are two reasons for this unbalance: First, GPUs do not have enough memory to hold a large enough partition that would balance the work for some workloads. Second, for BFS, the load varies across iterations. The time-series that presents the active/idle states for each benchmark shed more light on this effect (Figure 5).

4.3 Power-normalized Processing Rates

To estimate the energy efficiency of different configurations, Figure 6 shows the power-normalized performance for BFS and PageRank respectively (i.e., raw performance divided by drawn average power). Note that, for each workload, the plots can also be viewed as a comparison of raw energy consumed to process the graph.

First, we compare the power-normalized performance of configurations with two processing elements. Apart from the BFS benchmark on UK-WEB graph, a hybrid 1S1G system improves *both* raw performance and power-normalized performance compared to the symmetric 2S system. In the best case, *the hybrid system achieves 1.9x higher efficiency* for the power-normalized performance metric.

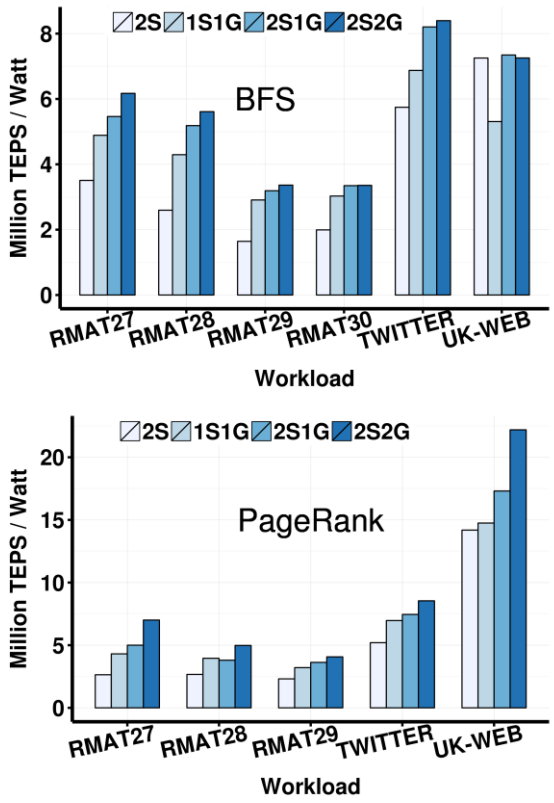

Figure 6: Power-normalized processing rate (the higher the better) for *BFS* (top) and *PageRank* (bottom).

Second, adding more GPUs improves power-normalized performance as the gain in raw performance is higher than the increase in power consumption. *Importantly, these gains are preserved when processing larger graphs and for all executions of PageRank* and most workloads for BFS (§6 discusses the exceptions).

4.4 Energy Delay Product

Finally, using a different energy-oriented metric, the energy-delay product (EDP), would not only support the same qualitative observations, but the relative advantage of the hybrid solution is even higher. Figure 7 presents the results of this experiment *normalized* to the 2S configuration to make the plot readable.

5. RELATED WORK

There is no shortage of work on optimizing graph algorithms on either the CPU or the GPU alone [2, 8, 16, 22]. This past work is complementary to our approach since we use some of these techniques as building blocks in our hybrid implementations.

Previous work on evaluating power efficiency focused on either the CPU or the GPU independently [1, 17, 23, 27], and mainly targeted regular applications that are friendly to GPU's SIMT computation model (e.g., dense linear algebra problems). This work focuses on system-level evaluation of an application that harnesses both processing elements. Moreover,

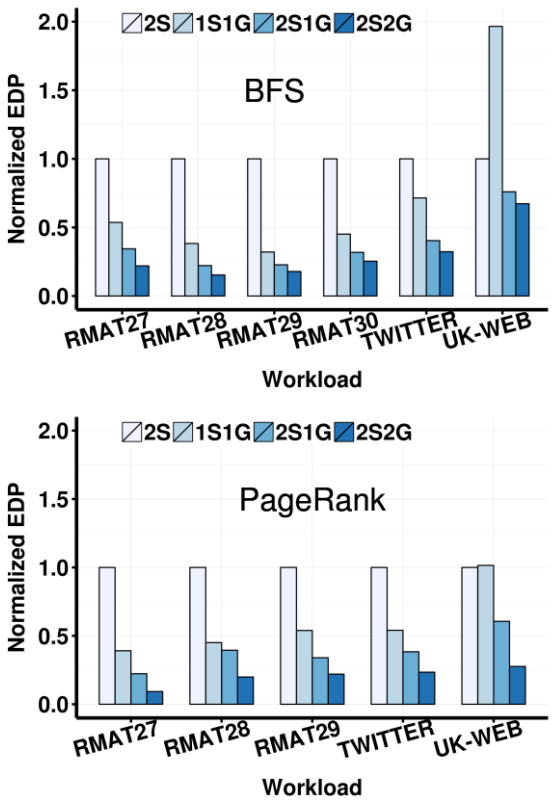

Figure 7: Normalized energy-delay product (the lower the better). The baseline is the configuration that uses both processors (2S). BFS (*top*) and PageRank (*bottom*).

the application domain we target, graph processing, exhibits irregular parallelism, where the power and performance benefits of GPU acceleration are not intuitive.

Finally, the Graph500 committee has recently introduced a new challenge, GreenGraph500[5], which ranks solutions based on TEPS/watt. The first rank (June, 2013) was dominated by small graphs processed on single-node machines. Our submission was ranked, and it processed the largest graph (scale-28) in its category ('Small Data' category). This paper includes additional optimizations (the results we report here are not directly comparable as we use directed graphs unlike the benchmark which uses undirected graphs).

6. SUMMARY AND DISCUSSION

Summary. This work demonstrates that GPU-acceleration improves both time-to-solution as well as energy consumption for large-scale graph processing, and that this improvement scales when increasing the graph size and adding more GPUs. Further, although the GPUs we use have one order of magnitude less memory, our experience shows that a hybrid (one CPU and one GPU) system can be more power efficient than a dual-CPU symmetric one.

The rest of this section discusses few related questions.

Why not use DVFS to lower energy footprint? On the CPU side, a recent analysis [27] shows that, for recent Intel processors (e.g., Intel's Sandy Bridge), both memory latency and bandwidth strongly depend on processor frequency (we confirmed this result on our platform). This limits the opportunity to use DVFS to save energy on the CPU side. On the GPU side, however, recent GPU models support setting different frequencies for the memory and the compute cores. Previous work [1, 17] shows that energy consumption can be reduced by lowering the core frequency for memory-intensive kernels. This is a direction we aim to investigate to lower the power drawn by the GPUs.

What other factors that could improve the performance/watt ratio of hybrid systems? On the algorithmic side, we aim to investigate the feasibility of graph partitioning techniques that target to minimize energy consumption. On the hardware side, we believe that having more memory on the GPU would significantly improve performance as a larger partition can be offloaded. Also, we aim to investigate the use of low-voltage DRAM, which reduces the power drawn by the large memory space. Finally, high-bandwidth, low-power SSDs are now available (e.g., Intel's 900 family, supports 1GB/s sequential read and draws as little as 25W); such storage can be used to offload part of the read-mostly graph state (e.g., the graph data structure), and hence reduce power drawn by memory.

Do all graphs look like UK-WEB and thus make our points invalid for BFS? No. The UK-WEB workload has a long diameter. It has two orders of magnitude longer than the other workloads, e.g. social networks, although the average path length stayed about the same. This topology leads to a large number of BSP rounds and stresses a limitation in the relatively simple design of TOTEM's communication layer. In particular, TOTEM communicates the entire 'ghost-zone' at the frontier between the CPU and the GPU, which results in extra communication over the PCI-E 2.0 bus. This limitation manifests only for BFS because of its minimal computation per vertex compared to other graph algorithms (e.g, PageRank and Betweenness Centrality), which is not enough to hide this overhead. This limitation can be addressed in two non-exclusive ways: (*i*) improving the design of TOTEM's communication layer by keeping track of changes and communicating only these ones, and (*ii*) connecting the GPU via a PCI-E 3.0 bus, which offers double the bandwidth. Note that other important large-scale networks, such as social networks [18, 19] and the Web itself [3], are known to have a shorter diameter.

Here is my graph, what platform offers the best tradeoff between acquisition cost, energy and performance? Our experience supports recommending the following simple decision process: If the graph is small and fits GPU memory, we recommend processing it on GPU only (a single GPU draws power comparable to a dual-socket CPU, but it is at least 2x faster). For larger graphs, we recommend boosting the host's memory, adding GPUs and using TOTEM to implement algorithms on such a hybrid setup. Finally, for massive many-billion vertices graphs, if energy is the main concern, we speculate that a single-node solution along the lines of GraphChi [20], which processes the graph from SSDs, will be most advantageous. If time-to-solution is the primary concern then we conjecture that

[5] http://green.graph500.org/

a cluster composed of as few *fat* nodes as possible, where each node is provisioned with as much memory and GPUs as possible, will be the most efficient setup (compared to a cluster of many low-end commodity nodes as used today).

7. REFERENCES

[1] Abe, Y., Sasaki, H., Peres, M., Inoue, K., Murakami, K. and Kato, S. Power and performance analysis of GPU-accelerated systems. *HotPower '12*.

[2] Agarwal, V., Petrini, F., Pasetto, D. and Bader, D.A. Scalable Graph Exploration on Multicore Processors. *SC'10*.

[3] Albert, R., Jeong, H. and Barabasi, A.-L. 1999. Internet: Diameter of the World-Wide Web. *Nature*. 401, 6749 (Sep. 1999), 130–131.

[4] Boldi, P., Rosa, M., Santini, M. and Vigna, S. Layered Label Propagation: A Multiresolution Coordinate-Free Ordering for Compressing Social Networks. *WWW '11*.

[5] Boldi, P., Santini, M. and Vigna, S. A Large Time-Aware Web Graph. *ACM SIGIR '08*.

[6] Boldi, P. and Vigna, S. The Webgraph Framework I. *WWW '04*.

[7] Chakrabarti, D., Zhan, Y. and Faloutsos, C. R-MAT : A Recursive Model for Graph Mining. *SDM '04*.

[8] Chhugani, J., Satish, N., Kim, C., Sewall, J. and Dubey, P. Fast and Efficient Graph Traversal Algorithm for CPUs: Maximizing Single-Node Efficiency. *IPDPS '12*.

[9] Gharaibeh, A., Beltrão Costa, L., Santos-Neto, E. and Ripeanu, M. A Yoke of Oxen and a Thousand Chickens for Heavy Lifting Graph Processing. *PACT '12*.

[10] Gharaibeh, A., Costa, L.B., Santos-Neto, E. and Ripeanu, M. On Graphs, GPUs, and Blind Dating: A Workload to Processor Matchmaking Quest. *IPDPS '13*.

[11] Gharaibeh, A. and Ripeanu, M. Size Matters: Space/Time Tradeoffs to Improve GPGPU Applications Performance. *SuperComputing '10*.

[12] Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D. and Zadeh, R. WTF: The Who to Follow Service at Twitter. *WWW '13*.

[13] Han, W.-S., Lee, S., Park, K., Lee, J.-H., Kim, M.-S., Kim, J. and Yu, H. TurboGraph: A Fast Parallel Graph Engine Handling Billion-Scale Graphs in a Single PC. *KDD '13*.

[14] Harish, P., Narayanan, P., Aluru, S., Parashar, M., Badrinath, R. and Prasanna, V. Accelerating Large Graph Algorithms on the GPU Using CUDA. *HiPC '07*.

[15] Hong, S., Kim, S.K., Oguntebi, T. and Olukotun, K. Accelerating CUDA Graph Algorithms at Maximum Warp. *PPoPP '11*.

[16] Hong, S., Oguntebi, T. and Olukotun, K. Efficient Parallel Graph Exploration on Multi-Core CPU and GPU. *PACT '11*.

[17] Jiao, Y., Lin, H., Balaji, P. and Feng, W. Power and Performance Characterization of Computational Kernels on the GPU. *GREENCOM '10*.

[18] Kumar, R., Novak, J. and Tomkins, A. Structure and Evolution of Online Social Networks. *KDD '06*.

[19] Kwak, H., Lee, C., Park, H. and Moon, S. What is Twitter, a social network or a news media? *WWW '10*.

[20] Kyrola, A., Blelloch, G. and Guestrin, C. GraphChi: Large-Scale Graph Computation on Just a PC. *OSDI '12*.

[21] Malewicz, G., Austern, M.H., Bik, A.J.., Dehnert, J.C., Horn, I., Leiser, N. and Czajkowski, G. Pregel: A System for Large-Scale Graph Processing. *SIGMOD '10*.

[22] Merrill, D., Michael, G. and Grimshaw, A. Scalable GPU Graph Traversal. *PPoPP'12*.

[23] Molka, D., Hackenberg, D., Schone, R. and Muller, M.S. Characterizing the Energy Consumption of Data Transfers and Arithmetic Operations on x86−64 Processors. *ICGC '10*.

[24] NVIDIA 2013. TESLA K20 GPU active accelerator board specification.

[25] Page, L., Brin, S., Motwani, R. and Winograd, T. The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab '99.

[26] Rowstron, A., Narayanan, D., Donnelly, A., O'Shea, G. and Douglas, A. Nobody ever got fired for using Hadoop on a cluster. *HotCDP '12*.

[27] Schöne, R., Hackenberg, D. and Molka, D. Memory Performance at Reduced CPU Clock Speeds: An Analysis of Current x86 Processors. *HotPower '12*.

[28] Shun, J. and Blelloch, G.E. Ligra: A Lightweight Graph Processing Framework for Shared Memory. *PPoPP '13*.

[29] Stanford Network Analysis Project, http://snap.stanford.edu: .

[30] Valiant, L.G. A Bridging Model for Parallel Computation. *Communications of the ACM '90*.

[31] WattsUP Meter, http://www.wattsupmeters.com: .