Power Characteristics of Irregular GPGPU Programs

Jared Coplin
Department of Computer Science
Texas State University
coplin@txstate.edu

Martin Burtscher
Department of Computer Science
Texas State University
burtscher@txstate.edu

***Abstract*—This paper investigates the power profiles of irregular programs running on a K20 compute GPU and contrasts them with the profiles of regular programs. The paper further studies the effects on the power profile when changing the GPU's core and memory frequencies, using alternate implementations of the same algorithm, and varying the program input. Our results show that the power behavior of irregular applications often cannot be accurately captured by a single average. Rather, the entire profile, i.e., the power as a function of time, needs to be considered. In addition, lowering the frequency, employing alternate implementations, or using different inputs can drastically alter the power profile of irregular codes, meaning that measurements using one setting may not be representative of that program's power characteristics under a different setting.**

I. Introduction

GPU-based accelerators are widely used in high-performance computing and are quickly spreading in PCs and even handheld devices as they not only provide higher peak performance but also better energy efficiency than multicore CPUs. Nevertheless, large power consumption and the required cooling due to the resulting heat dissipation are major cost factors in HPC environments. To reach exascale computing, a 50-fold improvement in performance per watt is needed by some estimates [1]. Moreover, battery life is a key concern in all types of handhelds such as tablets and smartphones.

For these and other reasons, power-aware computing has become an important research area. While many hardware optimizations for reducing power have been proposed or are already in use, software techniques are lagging behind, particularly techniques that target accelerators like GPUs. However, to be able to optimize the power and energy efficiency of GPU code, we first need to develop a good understanding of the power consumption behavior of such programs.

To this end, we study the power profiles of GPU codes from different benchmark suites, in particular the profiles of irregular programs from the LonestarGPU suite. Irregular codes have data dependent behavior, making their power characteristics hard to predict statically and therefore especially interesting for our study. We also investigate how different GPU frequencies, different algorithm implementations, and different program inputs affect the power profiles.

By regular and irregular programs, we are referring to the behavior of the control flow and/or memory access patterns of the code. In regular code, control flow and memory references are not data dependent. Matrix-vector multiplication is a good example. Based only on the input size and the data-structure starting addresses, but without knowing any values of the input data, we can determine the dynamic behavior of the program on an in-order processor, i.e., the memory reference stream and the conditional branch decisions. In irregular code, the input values determine the program's runtime behavior, which therefore cannot be statically predicted and may change for different inputs. For instance, in a binary search tree, the values and the order in which they are processed affect the shape of the tree and the order in which it is built.

In most problem domains, irregular algorithms arise from the use of complex data structures such as trees and graphs. In general, irregular algorithms are more difficult to parallelize and more challenging to map to GPUs than regular programs. For example, in graph applications, memory-access patterns are usually data dependent since the connectivity of the graph and the values on nodes and edges may determine which nodes and edges are touched by a given computation. This information is usually not known at compile time and may change dynamically even after the input graph is available, leading to uncoalesced memory accesses and bank conflicts. Similarly, the control flow is usually irregular because branch decisions differ for nodes with different degrees or labels, leading to branch divergence and load imbalance. As a consequence, the power draw of irregular GPU applications can change over time in an unpredictable manner.

This paper makes the following main contributions.

1) It is the first paper to show power profiles and power characteristics from multiple irregular GPU programs.

2) It presents the first analysis of the power behavior of irregular GPU codes and how it differs from regular codes.

3) It reveals that the power profiles of irregular programs can be quite complex and can change significantly over time.

4) It demonstrates that different frequencies, inputs, and implementations can greatly alter the power profile.

The rest of this paper is organized as follows. Section II discusses related work. Section III provides an overview of the GPU we study. Section IV describes the evaluation methodology. Section V presents and analyzes the results. Section VI summarizes our findings and draws conclusions.

II. Related Work

We are not aware of other studies on the power behavior of irregular GPU programs. Only the power profile of the irregular Barnes Hut code has been described before [2], [20].

Many papers investigate Dynamic Voltage and Frequency Scaling (DVFS) on CPUs. For example, Kandalla et al. demonstrate the need to design software in a power-aware

978-1-4799-6177-1/14/$31.00 ©2014 IEEE

manner and to balance performance and power savings on a power-aware DVFS-capable cluster system [8]. Pan et al. also use DVFS and show that sometimes expending more energy does not result in a large performance benefit [16]. In addition to varying the frequency, Korthikanti and Agha explore how changing the number of active cores affects the energy consumption [9]. Freeh et al. perform a related study on a cluster that evaluates how different frequencies and numbers of compute nodes affect the power and performance of MPI programs [5]. There are also papers that highlight the lack of a standardized power measurement methodology for energy-efficient supercomputing [19] or talk about how ignoring power consumption as a design constraint in supercomputing will result in higher operational costs and diminished reliability [4].

Several publications propose and use analytical models to investigate power and energy aspects. For instance, Li and Martinez establish an analytical model for looking at parallel efficiency, granularity of parallelism, and voltage/frequency scaling [11]. Lorenz et al. explore compiler-generated SIMD operations and how they affect energy efficiency [13]. Some analytical models target GPUs. For example, Chen et al. institute a mechanism for evaluating and understanding the power consumption when running GPU applications [3]. Ma et al. use a statistical model to estimate the best GPU configuration to save power [14]. One simulation-based paper on thermal management for GPUs discusses methods for managing power through architecture manipulation [18].

There are several papers that measure energy consumption on actual GPU hardware. For instance, Gosh et al. explore some common HPC kernels running on a multi-GPU platform and compare their results against multi-core CPUs [7]. Ge et al. investigate the effect of DVFS on the same type of GPU that we are using [6]. A paper by Zecena et al. measures *n*-body codes running on different GPUs and CPUs [20].

All of these publications primarily or exclusively study regular codes. Moreover, most of them only consider the average or maximum power consumption. Our study is the first to evaluate the power profiles of multiple irregular GPU codes.

III. GPU Architecture

This section provides an overview of the architectural characteristics of the Kepler-based Tesla K20c GPU we use for our study. It includes an on-board power sensor that allows the direct measurement of the GPU's power draw.

The K20c consists of 13 streaming multiprocessors (SMs). Each SM contains 192 processing elements (PEs). Whereas each PE can run a thread of instructions, sets of 32 PEs are tightly coupled and must either execute the same instruction (operating on different data) in the same cycle or wait. This is tantamount to a SIMD instruction that conditionally operates on 32-element vectors. The corresponding sets of 32 coupled threads are called warps. Warps in which not all threads can execute the same instruction are subdivided by the hardware into sets of threads such that all threads in a set execute the same instruction. The individual sets are serially executed, which is called branch divergence, until they re-converge. Branch divergence hurts performance and energy efficiency.

The memory subsystem is also built for warp-based processing. If the threads in a warp simultaneously access words in main memory that lie in the same aligned 128-byte segment, the hardware merges the 32 reads or writes into one coalesced memory transaction, which is as fast as accessing a single word. Warps accessing multiple 128-byte segments result in correspondingly many individual memory transactions that are executed serially. Hence, uncoalesced accesses are slower and require more energy than coalesced accesses.

The PEs within an SM share a pool of threads called thread block or simply block, synchronization hardware, and a software-controlled data cache called shared memory. A warp can simultaneously access 32 words in shared memory as long as all words reside in different banks or all accesses within a bank request the same word. The SMs operate largely independently and can only communicate through global memory (main memory in DRAM). Shared memory accesses are much faster and more energy efficient than global memory accesses.

IV. Methodology

A. Benchmark programs

We study the irregular programs from the LonestarGPU benchmark suite [12] and contrast them with some regular programs from the Parboil suite [17]. We chose which codes, implementations, settings, and inputs to evaluate based on the resulting active runtime to obtain sufficiently many power samples. Table 1 lists the number of global kernels each program contains as well as the inputs we used.

Table 1: Programs, number of global kernels (#K), and inputs

Program	#K	Inputs
BFS	5	Roadmap of the entire USA (24M nodes, 58M edges)
BH	9	bodies-timesteps: 10k-10k and 100k-100
DMR	4	5M node mesh file
LBM	1	3000 timesteps
MST	7	Roadmap of the entire USA (24M nodes, 58M edges)
NSP	3	clauses-literals-literals/clause: 42k-10k-5
PTA	40	tshark, vim, and pine
SPMV	2	"large" benchmark input
SSSP	2	Roadmap of the entire USA (24M nodes, 58M edges)
TPACF	1	"large" benchmark input

1) LonestarGPU benchmarks

The LonestarGPU suite v2.0 is a collection of commonly used real-world applications that exhibit irregular behavior.

a. Barnes-Hut *n*-body Simulation (BH): An algorithm that quickly approximates the forces in a system of bodies in lieu of performing precise force calculations.

b. Breadth First Search (BFS): Computes the distance of each node from a source node in an unweighted graph using a topology-driven approach. In addition to the standard BFS implementation (topology-driven, one node per thread), we also study the atomic variation (topology driven, one node per

thread that uses atomics), the wla variation (one flag per node, one node per thread), the wlw variation (data driven, one node per thread) and the wlc variation (data-driven, one edge-per-thread version using Merrill's strategy [15]).

c. Delaunay Mesh Refinement (DMR): This implementation of the algorithm described by Kulkarni et al. [10] produces a guaranteed quality 2D Delaunay mesh, which is a Delaunay triangulation with the additional constraint that no angle in the mesh be less than 30 degrees.

d. Minimum Spanning Tree (MST): This benchmark computes a minimum spanning tree in a weighted undirected graph using Boruvka's algorithm and is implemented by successive edge relaxations of the minimum weight edges.

e. Points-to Analysis (PTA): Given a set of points-to constraints, this code computes the points-to information for each pointer in a flow-insensitive, context-insensitive manner implemented in a topology-driven way.

f. Single-Source Shortest Paths (SSSP): Computes the shortest path from a source node to all nodes in a directed graph with non-negative edge weights by using a modified Bellman-Ford algorithm.

g. Survey Propagation (NSP): A heuristic SAT-solver based on Bayesian inference. The algorithm represents the Boolean formula as a factor graph, which is a bipartite graph with variables on one side and clauses on the other.

2) Parboil benchmarks

Parboil is a set of applications used to study the performance of throughput-computing architectures and compilers. We investigate the following regular codes from this suite.

a. Lattice-Boltzmann Method Fluid Dynamics (LBM): A fluid dynamics simulation of an enclosed, lid-driven cavity using the Lattice-Boltzmann Method.

b. Sparse Matrix Vector (SPMV): An implementation of sparse matrix vector multiplication. The input file is a compressed matrix in Jagged Diagonal Storage (JDS) sparse matrix format. The output is a vector.

c. Two-Point Angular Correlation Function (TPACF): Used to statistically analyze the spatial distribution of observed astronomical bodies.

B. Evaluation test bed

We measured the GPU-kernel active runtime and energy consumption with the K20Power tool [2]. Our Tesla K20c GPU has 5 GB of global memory and 13 streaming multiprocessors with a total of 2,496 processing elements. It supports six clock frequency settings, of which we evaluate three: the "default" configuration, which uses a 705 MHz core speed and a 2.6 GHz memory speed, the "614" configuration, which uses a 614 MHz core speed and a 2.6 GHz memory speed, i.e., the slowest available compute speed at the normal memory speed, and the "324" configuration, which uses a 324 MHz core and memory speed, i.e., the slowest available frequency. Thus, our configurations are 1) default, 2) 614, and 3) 324.

We performed each experiment three times and report the results in form of power profiles. We only show one of the three profiles as we found the variability between them to be small. For example, the energy consumption varied by 1.9% on average between the three experiments. All codes were compiled using the default configurations set by their authors.

C. Active runtime

Throughout this paper, we refer to the "active runtime", which is not the total application runtime but rather the time during which the GPU is actively computing. The K20Power tool defines this as the amount of time the GPU is drawing power above the idle level. Figure 1 illustrates this.

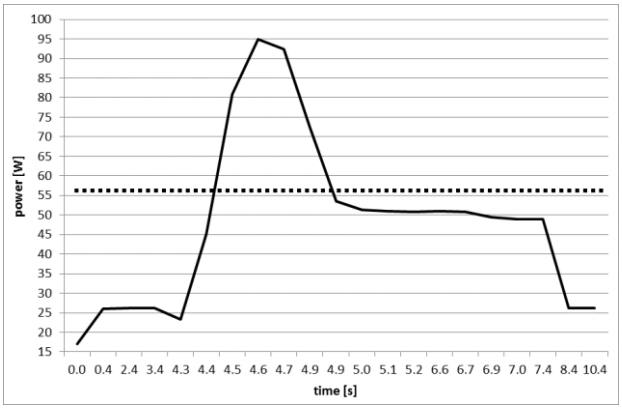

Figure 1: Sample power profile

Because of how the GPU draws power and how the built-in power sensor samples, only readings above a certain threshold (the dashed line at 55 W in this case) reflect when the GPU is actually executing the program [2]. Measurements below the threshold are either the idle power (less than about 26 W) or the "tail power" due to the driver keeping the GPU active for a while before powering it down. Using the active runtime ignores any execution time that may take place on the CPU. The power threshold is dynamically adjusted to maximize accuracy for different GPU settings.

V. Experimental Results

A. Regular vs. irregular GPU codes

1) Idealized profile

Figure 2 illustrates the shape of an idealized power profile from a regular kernel. At point 1, the GPU receives work and begins executing, which increases the power draw. The power remains stable throughout execution (2). All cores finish at point 3, thus returning the GPU to its idle power (about 17 W in the example), which completes the idealized rectangular profile. This profile's shape can be fully captured with just two values: the active runtime and the average power draw.

2) Regular codes

Figure 3 shows the actual power profiles of the three regular programs we study. Even though the power does not go up and down instantaneously and the active power is not quite constant, the profiles basically follow the idealized rectangular shape. In particular, the power ramps up quickly when the

code is launched, stays level during execution, and then drops off. Note that the power peaks at different levels depending on the program that is run and how much it exercises the GPU. It looks like TPACF gradually runs out of work towards the end of its execution, presumably due to load imbalance, and SPMV exhibits a small amount of irregularity due to variations in the sparseness of its input.

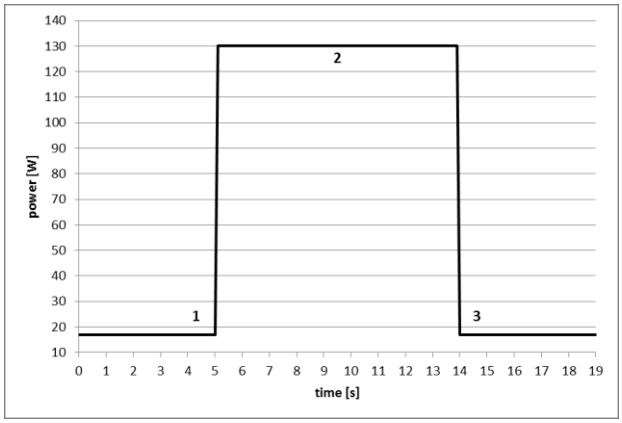

Figure 2: Idealized power profile

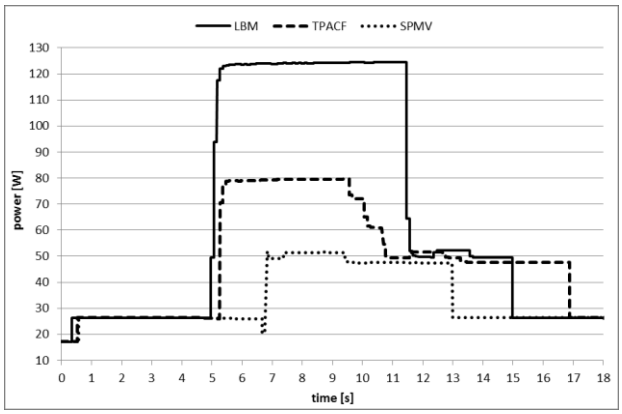

Figure 3: Power profiles of three regular codes

As previously reported [2], once a kernel finishes running, the power does not return to idle immediately. Rather, the GPU driver steps down the power in a delayed manner as it first waits for a while in case another kernel is launched. Note that the GPU samples the power less frequently at lower power levels. Since the power sensor's primary purpose is to prevent the GPU from damaging itself by drawing too much power, we surmise that the driver automatically reduces the sampling frequency when the power draw is low as there is little chance of damage to the GPU.

3) Irregular codes

Figure 4 shows the power profiles of the seven irregular LonestarGPU programs. Interestingly, the profiles of BFS and SSSP (the two dashed profiles in the top panel) are similar to the profiles of the regular codes shown in the previous subsection. This is because both implementations are topology driven, meaning that all vertices are visited in each iteration, regardless of whether there is new work to be done or not. A topology-driven approach is easy to implement as it essentially "regularizes" the code but may result in many useless computations being executed. Hence, BFS and SSSP behave like regular codes and have corresponding power profiles. Though harder to see (lower panel), NSP belongs to the same category as it also processes all vertices in each iteration.

In contrast, the profiles of DMR, MST, and PTA exhibit many spikes, which reflect the irregular nature of these programs. Due to dynamically changing data dependencies and parallelism amounts, their power draw fluctuates widely and rapidly. Clearly, the profiles of these three codes are very different from those of regular programs. Moreover, they are different from each other's profiles, highlighting that there is no such thing as a standard profile for irregular codes.

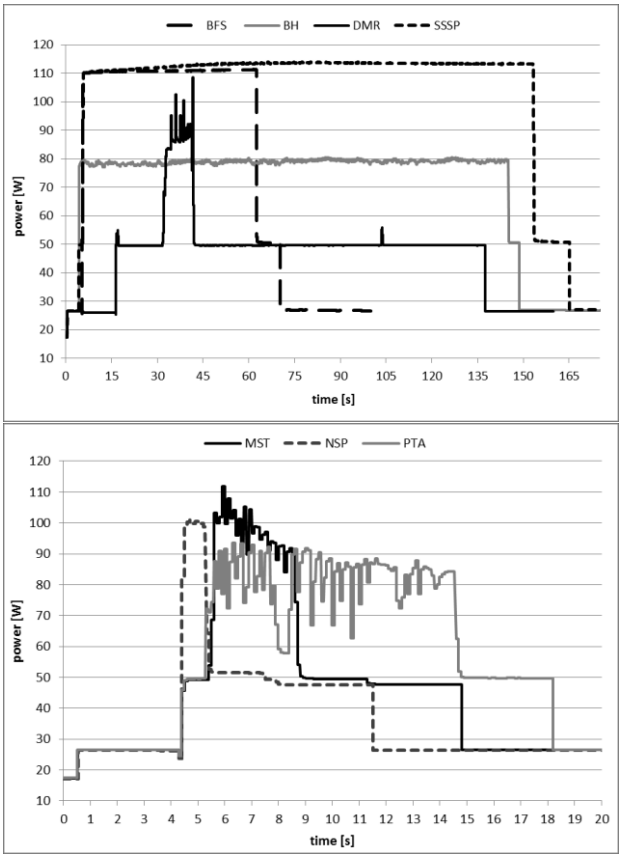

Figure 4: Power profiles of seven irregular codes (split over two panels to improve readability)

BH is more subtle in its irregularity. Its profile appears relatively regular except the active power wobbles constantly. In each time step, this program calls a series of kernels with different degrees of irregularity. Since the BH profile shown in Figure 4 was obtained with 10,000 time steps (and 10,000 bodies), the true irregularity is largely masked by the short runtimes of each kernel. Figure 5 shows the profile of the same BH code but with 100,000 bodies and only 100 time steps, which makes the repeated invocation of the different kernels much more evident. The power draw fluctuates by about 15 W while the program is executing. However, the irregularity within each kernel is still not visible. Note that the

active power is over 100 W in most cases in Figure 5 whereas it only reaches about 80 W in Figure 4. This is because running BH with 10,000 bodies does not fully load the GPU and the 10,000 time steps result in a large number of brief pauses between kernel calls, which lowers the GPU power draw.

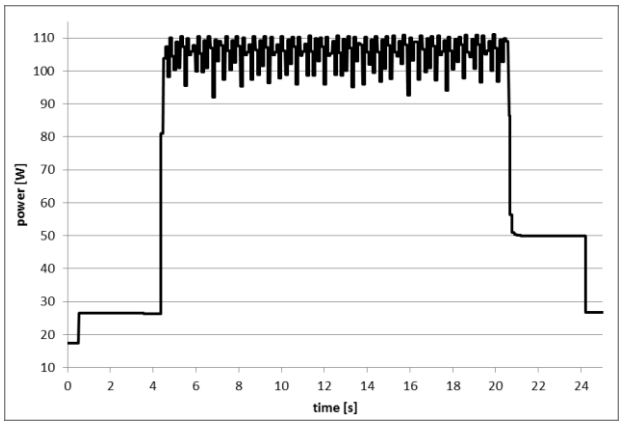

Figure 5: Power profile of BH with 100k bodies and 100 time steps

4) Comparing idealized, regular, and irregular profiles

Figure 6 shows the idealized power profile overlaid with a regular (LBM) and an irregular (PTA) profile of roughly the same active time. While the idealized and regular profiles have a similar shape, the irregular power profile clearly does not. Whether because of load imbalance, irregular memory access patterns, or unpredictable control flow, the irregular code's power draw fluctuates wildly. The active power of the PTA code averages 82 W but reaches as high as 93 W and as low as 57 W, which is a variation of over 60%. This is why the power behavior of irregular programs cannot accurately be captured by averages. Instead, the entire profile, i.e., the power as a function of time, needs to be considered.

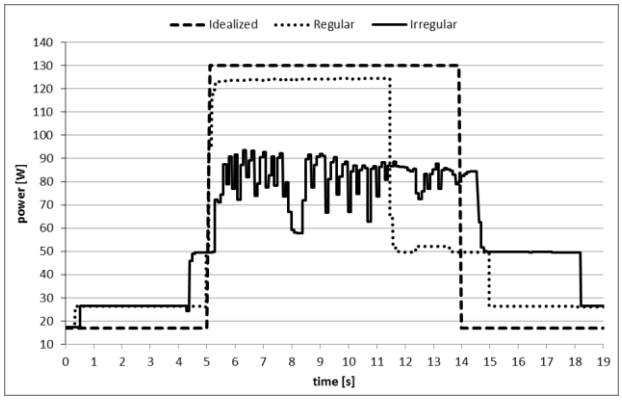

Figure 6: Comparison of idealized, regular, and irregular profile

B. Changing implementations, clock frequencies, and inputs

1) Changing the implementation

Changing the implementation of certain irregular algorithms can have a profound impact on both the active runtime and the power draw. Figure 7 illustrates this on the example of BFS, for which the LonestarGPU suite includes multiple versions. We profiled each version on the same input.

As discussed before, the default BFS implementation is topology-driven and therefore quite regular in its behavior but inefficient. BFS-WLA, the next faster version, is still topology-driven but includes an optimization to skip vertices where no re-computation is needed. This results in a shorter runtime but also substantial load imbalance, which is why the active power is so low. BFS-ATOMIC uses a different optimizations strategy involving atomic operations, making it even faster but resulting in a somewhat irregular power profile. The remaining implementations are data-driven and therefore work efficient, i.e., they do not perform unnecessary computations. This is why they are much faster. Both BFS-WLW's and the improved BFS-WLC's profile exhibits irregular wobbles, but they are too short to be visible in Figure 7. Clearly, the shape of the power profile is not necessarily similar for different implementations of the same algorithm.

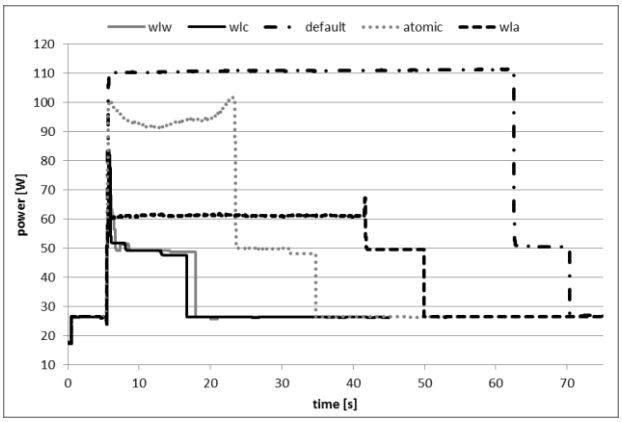

Figure 7: Power profile of different implementations of BFS

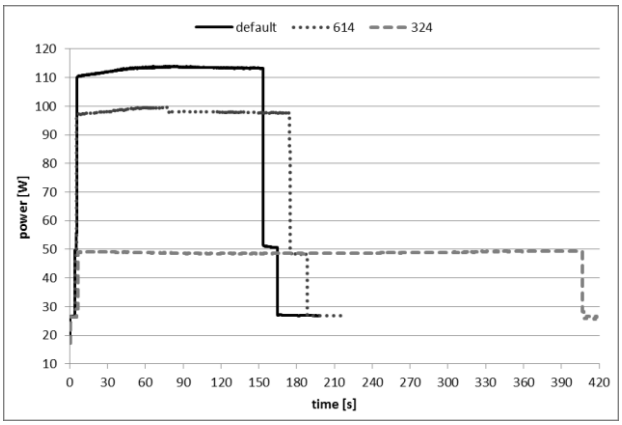

Figure 8: Power profile of SSSP for different clock frequencies

2) Changing the clock frequency

Figure 8 shows the effects on SSSP's power profile when the GPU frequency is changed. The 614 configuration has a core speed that is about 15% slower than the default, which results in a reduction of the power draw by a little over 10 W and an increase in the active runtime by almost 25 seconds. Hence, the total energy consumption remains about the same but the power draw is markedly lower.

The 324 configuration has a core speed that is less than half

that of the default, and its power draw is also a little less than half. Since the runtime is about 2.5 times higher than the default, the energy consumption goes up. Note that the 324 configuration has a memory speed that is eight times slower than the default, so the more memory-bound a program is the greater the increase in runtime will be. It should also be noted that on low-frequency settings, the power rarely reaches a sufficient level for the GPU to switch from the 1 Hz to the 10 Hz sampling rate, so obtaining enough power samples to study the power profiles at such low frequencies can be difficult.

3) Changing the input

Figure 9 shows how using different inputs affects the power profile of the irregular PTA code. In fact, each tested input yields a rather distinct profile. For example, pine results in an over 15 W higher power draw on average than vim, and tshark yields a profile in which the power ramps up after an initial spike whereas the other two inputs result in more spikes and a slight decrease in power consumption over time. Since a change in input can have such a drastic effect on the power profile of irregular programs, it may not be possible to use information from one type of input to accurately characterize the behavior for a different input. In fact, the early behavior of a running program may not even be indicative of its later behavior during the same program run, as tshark illustrates.

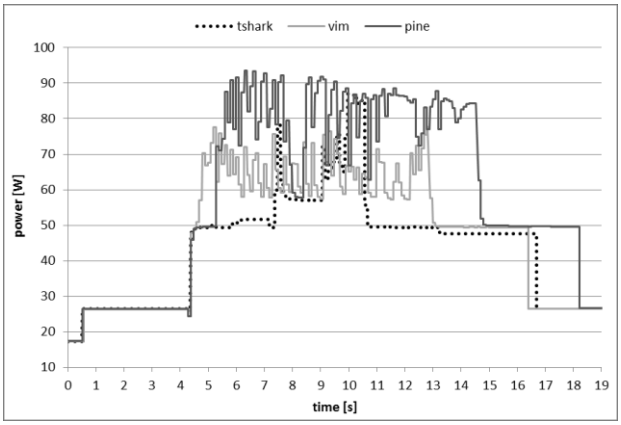

Figure 9: Power profile of PTA for different program inputs

VI. Summary and Conclusions

This paper studies the dynamic power characteristics of three regular and seven irregular GPGPU programs. We find that the power profiles of regular codes are typically similar to the idealized "rectangular" power profile but the profiles of irregular codes often are not. In fact, there is no such thing as a standard power profile for irregular codes. Rather, each program potentially yields a dissimilarly shaped profile. Moreover, changing the implementation or using a different input can drastically alter the power profile. In fact, even for a fixed implementation and input, the early execution behavior of an irregular code may not be indicative of the later behavior due to dynamically changing data dependencies and parallelism amounts. Hence, the power behavior of irregular programs cannot accurately be captured by averages. Instead, the power as a function of time must be considered and may have to be re-evaluated for each input and after each code modification.

Acknowledgments

The work reported in this paper is supported by the U.S. National Science Foundation under Grants DUE-1141022, CNS-1217231, CNS-1406304, and CCF-1438963, a REP grant from Texas State University as well as grants and equipment donations from NVIDIA Corporation.

References

[1] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick. "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems." Peter Kogge, 2008.

[2] M. Burtscher, I. Zecena, and Z. Zong. "Measuring GPU Power with the K20 Built-in Sensor." *Seventh Workshop on General Purpose Processing on Graphics Processing Units*. March 2014.

[3] J. Chen, B. Li, Y. Zhang, L. Peng, and J. Peir. "Statistical GPU power analysis using tree-based methods." *2011 International Green Computing Conference and Workshops*. July 2011.

[4] W. Feng, X. Feng, and R. Ge. "Green Supercomputing Comes of Age." *IT Professional*. February 2008.

[5] V. Freeh, F. Pan, N. Kappiah, D. Lowenthal, and R. Springer. "Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster." *19th IEEE Int'l Par. and Distributed Processing Symp*. 2005.

[6] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong. "Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU." *2nd Workshop on Power-aware Algorithms, Systems, and Arch*. 2013.

[7] S. Ghosh, S. Chandrasekaran, and B. Chapman. "Energy Analysis of Parallel Scientific Kernels on Multiple GPUs." *2012 Symposium on Application Accelerators in High Performance Computing.* July 2012.

[8] K. Kandalla, E.P. Mancini, S. Sur, and D.K. Panda. "Designing Power-Aware Collective Communication Algorithms for InfiniBand Clusters." *39th International Conference on Parallel Processing*. 2010.

[9] V. Korthikanti and G. Agha. "Towards optimizing energy costs of algorithms for shared memory architectures." *22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures.* June 2010.

[10] Milind Kulkarni, Keshav Pingali, Bruce Walter, Ganesh Ramanarayanan, Kavita Bala and L. Paul Chew. "Optimistic Parallelism Requires Abstractions." *ACM Conference on Programming Languages Design and Implementation,* 211-222, June 2007.

[11] J. Li and J. Martínez. "Power-performance considerations of parallel computing on chip multiprocessors." *ACM Transactions on Architecture and Code Optimization*. December 2005

[12] LonestarGPU: http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu

[13] M. Lorenz, P. Marwedel, T. Dräger, G. Fettweis, and R. Leupers. "Compiler based exploration of DSP energy savings by SIMD operations." *Asia and South Pacific Design Automation Conference*. 2004.

[14] X. Ma, M. Rincon, and Z. Deng. "Improving Energy Efficiency of GPU based General-Purpose Scientific Computing through Automated Selection of near Optimal Configurations." *TR UH-CS.* August 2011.

[15] Duane Merrill, Michael Garland, and Andrew Grimshaw. "Scalable GPU graph traversal." *17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 117-128. 2012.

[16] F. Pan, V. Freeh, and D.M. Smith. "Exploring the energy-time tradeoff in high-performance computing." *Par. and Distr. Proc. Symp.* 2005.

[17] Parboil: http://impact.crhc.illinois.edu/Parboil/parboil.aspx

[18] J. Sheaffer, K. Skadron, and D.P. Luebke. "Studying Thermal Management for Graphics-Processor Architectures." *IEEE International Symposium on Performance Analysis of Systems and Software.* March 2005.

[19] B. Subramaniam and W. Feng. "Understanding Power Measurement Implications in the Green500 List." *Green Computing and Communications, 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical, and Social Computing*. December 2010.

[20] I. Zecena, M. Burtscher, J. Tongdan, and Z. Ziliang. "Evaluating the performance and energy efficiency of n-body codes on multi-core CPUs and GPUs." *2013 IEEE 32nd Int'l Perf. Comp. and Comm. Conf.* 2013.