Measuring GPU Power with the K20 Built-in Sensor

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

Ivan Zecena
Department of Computer Science
Texas State University
ivan_zcl@txstate.edu

Ziliang Zong
Department of Computer Science
Texas State University
ziliang@txstate.edu

ABSTRACT
GPU-accelerated programs are becoming increasingly common in HPC, personal computers, and even handheld devices, making it important to optimize their energy efficiency. However, accurately profiling the power consumption of GPU code is not straightforward. In fact, we have identified multiple anomalies when using the on-board power sensor of K20 GPUs. For example, we have found that doubling a kernel's runtime more than doubles its energy usage, that kernels consume energy after they have stopped executing, and that running two kernels in close temporal proximity inflates the energy consumption of the later kernel. Moreover, we have observed that the power sampling frequency varies greatly and that the GPU sensor only performs power readings once in a while. We present a methodology to accurately compute the instant power and the energy consumption despite these issues.

Categories and Subject Descriptors
C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors) – *Single-instruction-stream, multiple-data-stream processors (SIMD)*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids.

General Terms
Algorithms, Measurement, Performance, Design, Experimentation.

Keywords
Power measurement, energy measurement, GPU power sensor.

1. INTRODUCTION
The computing landscape has changed substantially since the introduction of accelerators, in particular compute GPUs [1]. For example, many of the world's top supercomputers now contain accelerators [2]. Every new GPU generation, including the Kepler architecture [3], improves the delivered performance and energy efficiency. As a result, the number of devices with GPUs and the number of GPU-accelerated applications will probably increase rapidly over the coming years. However, to be able to optimize the energy efficiency of GPU code, we first need to develop a good understanding of the energy consumption of such programs, which requires accurate power profiling.

GPGPU-7, March 01 2014, Salt Lake City, UT, USA

To make real-time power information available, the latest GPUs (*e.g.*, the Tesla K20) have on-board sensors for querying the power consumption at runtime. Being able to obtain accurate power data enables GPU programmers to evaluate and improve the energy efficiency of their code. However, the majority of published work in this area focuses on using models to estimate the power consumption [4] [5] [6] [7] [8], presumably because most GPUs do not yet include built-in sensors. To the best of our knowledge, there is no published work that comprehensively investigates how to measure the energy consumption of modern GPUs like the K20 using the built-in power sensor.

Such measurements are not as trivial as it might seem. In particular, the straightforward approach of sampling the power, computing the average, and multiplying by the runtime of the GPU code is likely to yield large errors and nonsensical results. For example, using this approach, doubling a GPU kernel's runtime results in much more than twice the amount of energy used. Similarly, a kernel's energy consumption appears to increase when another kernel is executed just before it.

The straightforward approach assumes that the power measurements closely track the GPU activity (*i.e.*, when kernel code is executing), that the sampling intervals are equal, that only application code causes GPU activity, *etc*. However, we show that the behavior of an actual GPU is more complex. In particular, this paper makes the following contributions.

- We discuss and explain a number of unexpected behaviors when measuring a GPU's power consumption.
- We make important observations that should be taken into account when working with K20 power samples.
- We present a methodology to accurately compute the true power and energy consumption using sensor data.
- We validate our methodology in multiple ways and test it on Kepler-based K20c, K20m, and K20x GPUs.
- We make our GPU energy-measurement tool, which implements this methodology, publicly available in open source at http://cs.txstate.edu/~burtscher/research/K20power/.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 describes our test bed. Section 4 analyzes potential problems with GPU power-sensor measurements. Section 5 presents our approach to 'correct' these measurements. Section 6 discusses validation results. Section 7 summarizes and draws conclusions.

2. RELATED WORK
The energy consumption of computer components can be obtained either directly or indirectly. Indirect measurements estimate the power consumption using a model that correlates power with hard-

ware performance counters or other events. This approach has already been used for CPUs before compute GPUs became available [6] [9]. Two widely-used CPU power models are Wattch [10] for single-core CPUs and McPAT [11] for multi-core CPUs. Research on developing power models for GPUs is still in an early stage. Most of the GPU power models that utilize performance counters rely on statistics to correlate power to performance [7] [8] [12] [13] [14] [15] [16] [17] [18]. Hong and Kim proposed an integrated power and performance prediction model to predict the optimal number of active GPU processors for a given application [5]. Song *et al.* proposed a GPU power model that is based on the training results from an artificial neural network [8]. Choi and Vuduc proposed the roofline model to estimate the GPU energy consumption [7]. Sohan *et al.* recently released GPUSimPow [12] [13], a framework for modeling GPU power consumption, and Leng *et al.* developed GPUWattch [14], a configurable GPU power model based on cycle-level simulations, but Kepler-based GPUs are not yet supported. The benefit of using models to estimate the power consumption is that it can be deployed with low cost and applied to short-running codes. However, the parameters of these models need to be retrained for new hardware. In addition, the power estimation may be relatively inaccurate. For example, GPUWattch differs by 9.9% to 13.4% from the power measured on actual hardware [14].

Direct measurement methods periodically collect samples of the current and voltage. The power is computed by multiplying the two values, and the total energy is calculated as the integral of the power over the execution time. For instance, using a simple power meter like WattsUP [19], we can collect the power readings of an entire node and calculate the total energy consumed by that node over a period of time. Note that WattsUP is easy to use but its maximum sampling frequency is rather low (1 Hz). To obtain the power consumption of an individual component, *e.g.*, a GPU, the base power of the node must be subtracted.

Oftentimes, coarse-grained power measurements are not sufficient to optimize the energy-efficiency of code with complicated characteristics. Therefore, several tools have been developed to provide fine-grained power consumption information. The best known such tool is probably PowerPack [20], which was developed at Virginia Tech for System G, the currently largest power-aware cluster [21]. PowerPack is able to measure the power consumption of individual components (*e.g.*, the CPU or DRAM) within a node. However, its hardware-software power profiling approach is fairly expensive, difficult to implement, and hard to scale. Another widely used tool is PowerMon [22] [23], which comprises a power monitoring card that plugs into the motherboard. Compared to PowerPack, PowerMon is cheaper and easier to implement because it only contains a single integrated circuit (no wiring or soldering required). However, it inherits the weakness of PowerPack to only be able to measure the power consumption of a component that is directly plugged into the power supply. Sandia National Laboratory is exploring component-level power measurement tools for deployment in large-scale systems [24][25][26]. Very recently, they have presented the PowerInsight tool [27], which can instrument accelerators that draw power from the PCI bus and external power supplies. Another notable trend is to directly integrate power sensors into accelerators. For example, compute GPUs such as the Tesla C2075 (Fermi architecture [1]) and the Tesla K20 (Kepler architecture [3]) include on-board power sensors that allow the direct measurement of the GPU's power draw.

Some of the observations we make in this paper for GPUs, including the tail energy, have also been reported for other devices such as the disk, WiFi, 3G, and GPS components of smartphones, which complicates the fine-grained energy measurement of mobile applications [28][29][30][31]. The inability to accurately capture the power consumption of very short-running kernels has also been identified in past CPU work and has motivated some of the aforementioned modeling approaches that are based on hardware performance counters.

3. BENCHMARKS, SYSTEM, AND ENERGY MEASUREMENT

3.1. Benchmark Description

We derived our test programs from two very different *n*-body implementations. Both algorithms simulate the gravity-induced motion of stars (a.k.a. bodies) in a star cluster for a user-selected number of time steps. The stars' positions and velocities are initialized according to the empirical Plummer model [32], which mimics the density distribution of globular clusters.

The first code, called NB, performs precise pair-wise force calculations. With n bodies, $O(n^2)$ interactions need to be considered. However, the same operations are performed for all n bodies, leading to a very regular implementation that maps well to GPUs. Moreover, the force calculations are independent, resulting in large amounts of parallelism. Our NB code reaches over 2 teraflops on a single K20c GPU and exceeds the performance of the *n*-body code in the CUDA 5.0 SDK.

The second code, called BH, uses the Barnes-Hut algorithm to approximately compute the forces [33]. It hierarchically partitions the volume around the n bodies into successively smaller cubes, called cells, until there is just one body per innermost cell. The resulting spatial hierarchy is recorded in an unbalanced octree. Each cell summarizes information about the bodies it contains. This hierarchy reduces the algorithm's complexity to $O(n \log n)$ because, for cells that are sufficiently far away, it suffices to perform only one force calculation with the cell instead of performing one calculation with each body inside the cell. We obtained the BH code from the LonestarGPU suite [34].

The NB code is relatively straightforward, has a high computational density, and only accesses main memory infrequently due to excellent caching in shared memory. In contrast, the BH code is quite complex, has a low computational density, performs mostly irregular pointer-chasing memory accesses, and consists of multiple different kernels. As a result, it 'only' reaches some 200 gigaflops. Nevertheless, because of its lower time complexity, it is about 33 times faster than the NB code when simulating one million stars. Both codes are over a factor of 30 faster than corresponding parallel OpenMP CPU code running on a modern Xeon multicore system.

3.2. GPU and Compiler Description

Our primary GPU is an NVIDIA Tesla K20c. It has 5 GB of global memory and 13 streaming multiprocessors (SMXs) with 2496 processing elements (PEs). We also used a second K20c GPU, a pair of Tesla K20m GPUs, and a pair of Tesla K20x GPUs for our study. The main difference between a K20c and a K20m is that the former is equipped with fans whereas the later needs to be passively cooled. The K20x, in contrast, has 6 GB of main memory and 14 SMXs with 2688 PEs. We compiled the CUDA codes with nvcc 5.0 and the '-O3 -arch=sm_35' flags. Depending on the experiment, we ran the programs with different numbers of stars and for one time step.

3.3. Energy measurement

We employ the GPU power sensors to obtain the energy consumption of the test kernels. We wrote our own tool to query the sensor via the NVIDIA Management Library (NVML) interface, which re-

turns the power readings in milliwatts [35]. For consistency, we always sample both the power and the memory usage and include a high-resolution timestamp with each sample even though we do not need all of this data for most of our experiments. Unless noted otherwise, all reported times are relative to when we started our tool and not relative to when the measured application or the first GPU kernel was launched.

4. MEASUREMENTS AND OBSERVATIONS

In this section, we highlight interesting aspects of power profiles obtained from GPU sensors and discuss some implications.

4.1. Power Lag and Distortion

Figure 1 displays the characteristic profile obtained when sampling the GPU power sensor before, during, and after the execution of a regular kernel. The kernel starts running at time t_1 and stops at time t_2. The power profile stems from a single invocation of the force-calculation kernel from the NB code. The profiles of other regular kernels look similar. Longer runtimes push the point t_2 to the right, extending the flat part of the power curve. Shorter runtimes push the point t_2 to the left, gradually eliminating the flat part (*cf.* Figure 2 below). The asymptotic level that the curve approaches is different for different kernels.

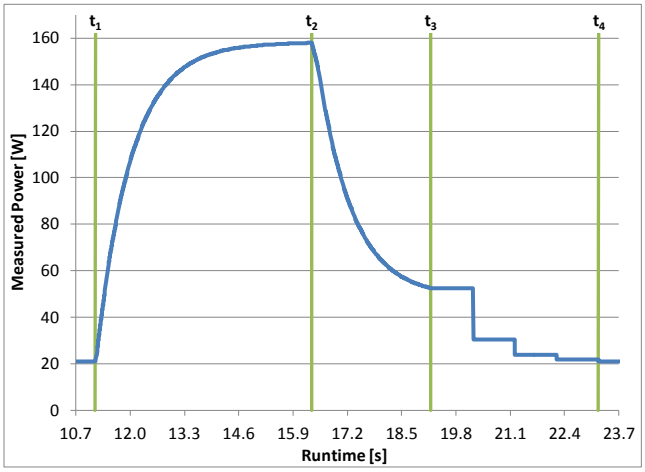

Figure 1: Typical shape of a GPU power profile when running a regular kernel (profile is from the NB force calculation kernel run once with 700,000 bodies)

The two most striking aspects of the power profile are that (1) the power consumption lags behind the kernel activity and that (2) the shape of the profile does not match the kernel activity. Note that the kernel runtime is 5.346 seconds, so these are macroscopic effects. The actual kernel activity almost instantaneously shoots up, stays at a constant level for the duration of the execution, and then almost instantaneously drops to zero, as the following experiment demonstrates. When reducing the kernel's workload to *X*%, the runtime also drops to *X*% (within a few milliseconds), indicating that the power level should stay constant during execution. Instead, the power profile for the first *X*% of the runtime is identical to the beginning of the profile in Figure 1. Similarly, the profile's shape after the kernel stops is also partially the same, as the dashed lines in Figure 2 show.

The power profile's lag and its dissimilarity from the actual kernel activity cause problems when integrating the power to obtain the energy consumption. For example, due to the gradual instead of instantaneous increase of the measured power, the integral from t_1 to t_2 will more than double when doubling the runtime of a kernel that performs the exact same computation twice. For instance, the energy calculated from the data in Figure 1 is 732 J, but the energy when running the same kernel twice as long is 1579 J, *i.e.*, 115 J or almost 8% more than double. This is counterintuitive. After all, the energy should also double or increase by a little less than a factor of two due to caching effects, only incurring certain overheads once instead of twice, *etc.*

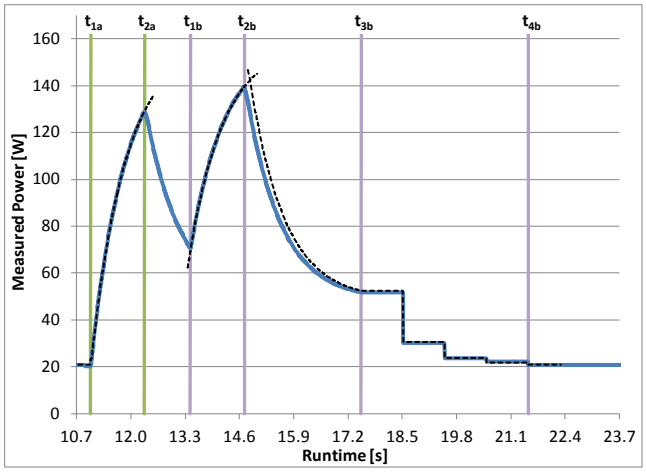

Figure 2: GPU power profile when running the NB force calculation kernel twice with a one-second delay between the two runs with 350,000 bodies (the dashed lines are three time-shifted partial copies of the profile from Figure 1)

Since the power is lagging behind the underlying kernel activity, it may be necessary to integrate beyond time t_2 (perhaps to time t_3 when the power curve changes to a stair shape or to time t_4 when the idle power is reached). Unfortunately, the end time for the integral is not obvious in the simple case shown in Figure 1 and is even less clear for more complex kernels or when consecutively invoking multiple kernels, which is common in GPU-accelerated applications. This latter case is particularly intriguing. If a second kernel is invoked before time t_4, that is, before the idle power is reached, its power profile will start at a higher level. The second kernel invocation at time t_{1b} in Figure 2 illustrates this case, where the subscript 'a' refers to the first invocation and the subscript 'b' to the second invocation. The dashed lines show that the power profiles of the first and second invocation precisely follow the shape of the longer running kernel from Figure 1 except for the period between t_{2b} and t_{3b} (*cf.* next subsection). However, the second invocation starts at a higher power level and reaches a higher maximum. Hence, the integrated energy of the second invocation is higher by an amount that is inversely related to the time between when the first kernel stops (t_{2a}) and when the second kernel starts (t_{1b}). For example, using the data from Figure 2, the energy consumption of the first invocation is 114 J whereas that of the second invocation is 147 J, a 29% increase, even though the same independent computations are performed on the same values.

4.2. Tail Energy

Figure 3 shows partial power profiles of six runs of the NB force calculation kernel. They are shifted such that the point t_2, at which the kernels stop running, is at 16.5 s for all of them. Different input sizes were used to produce different runtimes and thus different power levels when the kernels terminate.

Figure 3 illustrates that the power profile between t_2 and t_3 appears to be a function of the power level at time t_2. In contrast, the profiles between t_3 and t_4 are almost identical and independent of the kernel runtime or the reached power level. Moreover, a noticeable change in the shape of the profiles occurs at time t_3, where they transit from a curve to a step function. Clearly, integrating the power beyond time t_3 to obtain the energy consumption will simply add a constant amount of energy (about 130 J), making it doubtful whether this energy should be included. At best, it could be considered a fixed overhead that does not correlate with the kernel activity.

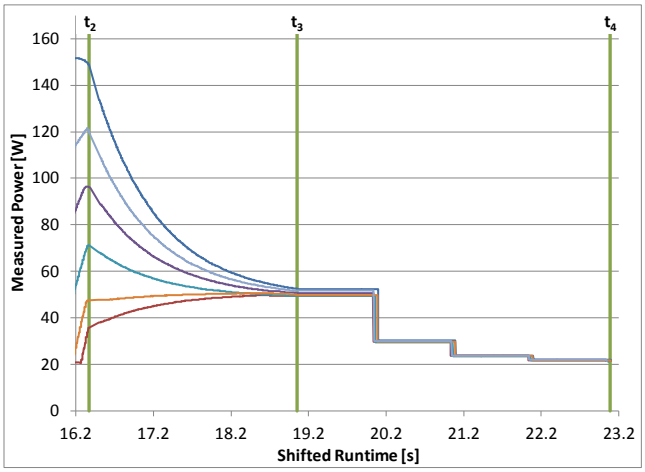

Figure 3: GPU power after running the NB force calculation kernel once with (from top to bottom) 500000, 320000, 250000, 190000, 124000 and 95000 bodies

Aside from the startling change in shape at time t_3, the most noteworthy aspect of these profiles is that they all settle at the same power level at time t_3, which is about 52.5 W. Note that this is true even if the power level at time t_2 is below 52.5 W, in which case the power draw actually *increases* after the computation has stopped. This counterintuitive behavior inflates the energy consumption of short-running kernels when integrating past t_2 to compensate for the power lag.

Since the delayed power consumption results in more 'missing' energy for short-running kernels, one could argue that it makes sense for the power draw to increase for a while after the kernel terminates. However, this is incorrect as the following experiment shows. With 95,000 bodies, the kernel runtime is 0.107 seconds, the maximum power is 35.7 watts, and the integrated energy from t_1 to t_2 is 3.25 J. Adding the first part (from t_2 to t_3) of the tail energy, which is 119 J, would result in a true power consumption of over 1.1 kW while the code is executing. This number far exceeds the GPU's maximum power rating. As on-board batteries and capacitors also cannot provide a kilowatt of power for a tenth of a second, integrating up to time t_3 or beyond to compensate for the power delay cannot be the correct approach. Hence, the energy consumption after time t_2, which we call 'tail energy', cannot be directly attributed to kernel activity.

4.3. Sampling Intervals

Figure 4 shows the elapsed time between consecutive power samples, *i.e.*, the sampling interval lengths, overlaid over the power profile from Figure 1. The cut-off spike at 16.6 s extends to 130 ms. These results are reproducible and were obtained when sampling the power at the maximum speed.

The interval lengths in Figure 4 exhibit several interesting features. First, they vary by almost a factor of 500, ranging from 0.266 ms to 130 ms. Clearly, the intervals cannot be assumed to be equal. Second, there are obvious spikes just before and a little after the kernel executes as well as whenever the power is stepped down during the t_3-t_4 phase. Third, the interval lengths fluctuate rapidly while the kernel is running and during the t_2-t_3 phase, *i.e.*, in both phases where the power profile is curved.

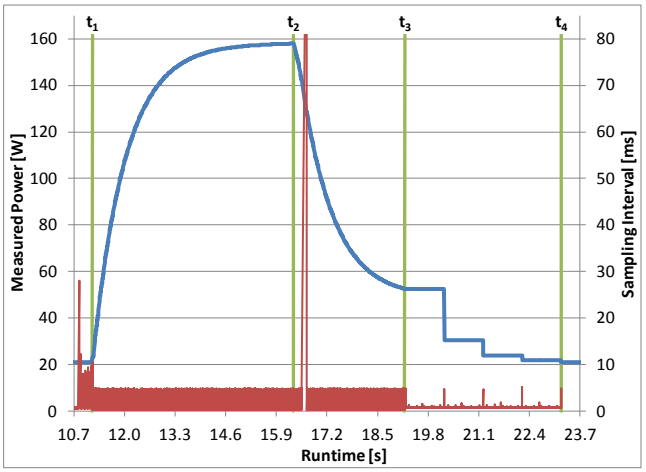

Figure 4: Sampling interval lengths (right axis, red line) and power profile (left axis, blue line) of the NB force calculation kernel run once with 700,000 bodies

Since the spikes in the t_3-t_4 phase coincide with the GPU stepping down to lower power levels, we assume that they are the result of the driver not responding as quickly during these periods. In other words, whenever the hardware or software is busy changing the GPU power level, requests for power samples are delayed. In fact, such activity also seems to be the reason for the largest spike. Note that this spike is not due to transferring data from the GPU to the CPU as the CUDA code measured in Figure 4 does not transfer any results back to the host. In fact, all of our experiments with transferring no, some, or a lot of data have never resulted in a significant change in power readings. Hence, we believe that the measured power domain does not include the PCI-express hardware. Additional experiments revealed that the large spike always occurs when the host code terminates. By also querying the amount of available memory on the GPU, we found that the spike coincides with the system-allocated memory being released. This again points to driver activity that delays the power sampling requests. Similarly, we surmise that the spikes before the GPU code starts running are due to driver activity to set up the kernel run. Again, memory consumption measurements corroborate this hypothesis. In particular, the largest spike before t_1 coincides with a multi-megabyte increase in GPU memory consumption that happens before the application allocates its own memory.

The reason for the rapid but consistent fluctuations of the interval lengths between t_1 and t_3 is also driver related. To better understand where these fluctuations are coming from, consider Figure 5, which shows a greatly magnified excerpt of Figure 4.

Figure 5 reveals a pattern that repeats throughout the t_1-t_3 phase. In particular, there are about 38 closely spaced and identical measurements, followed by a relatively large gap before the next set of closely spaced and identical measurements. The first set of power readings in Figure 5 are all at 109.325 W whereas the readings of the second set are all at 110.176 W. The sampling interval length

within these sets is about 0.283 ms. The gap between sets is around 4.7 ms. As the power measurements only change after a gap, we believe the gaps are caused by the GPU's sensor taking a new measurement, which is why it takes longer to respond. Hence, the closely spaced measurements appear to simply be repetitions of the latest measurement. With this in mind, we find the true maximum sampling frequency supported by the hardware to be about 66.7 Hz (15 ms sampling intervals).

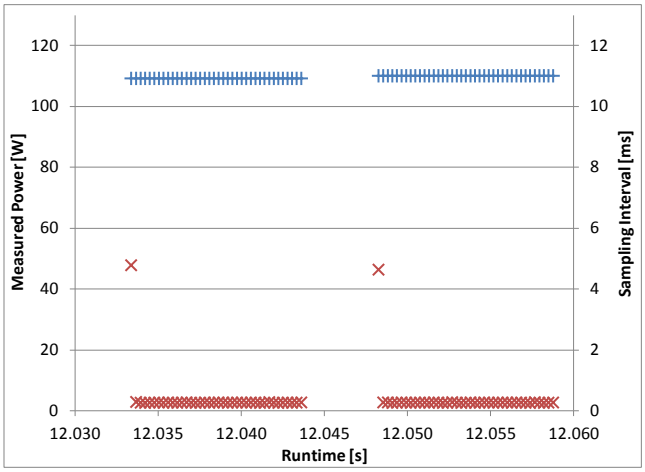

Figure 5: Zoomed in and re-scaled excerpt from Figure 4 showing the actual power samples ('+') and the corresponding interval lengths ('x', right axis)

This finding has interesting consequences. Assuming a well behaved power profile and that we are willing to accept an error of 5%, on the order of ten power samples are minimally needed to compute the energy. At 15 ms per sample, that amounts to a runtime of 150 ms, which exceeds the runtime of many kernels. In other words, it may be difficult to accurately measure the energy consumption of many real-world kernels with the built-in power sensor because their runtimes are too short.

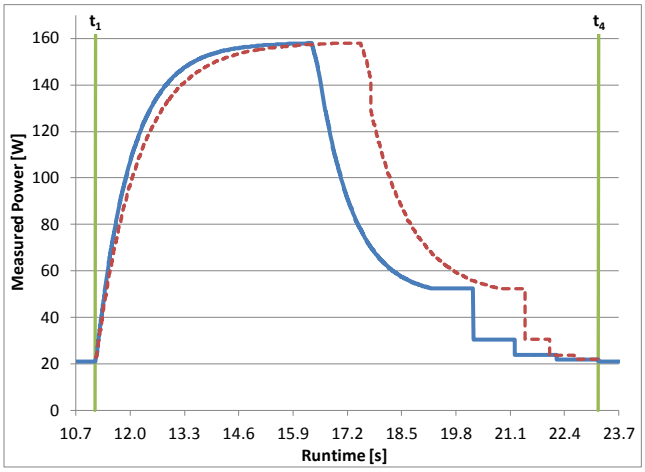

Figure 6: Power profile of the NB force calculation kernel run once with 700,000 bodies using actual (solid) and constant (dotted) sampling intervals

Another interesting point is that, due to the large fluctuations in the interval lengths, one cannot simply compute the average of the power readings and multiply it by the runtime to obtain the energy. Figure 6 illustrates this problem by showing both the power profile from Figure 1, which correctly accounts for the differences in interval lengths, as well as the same power readings but with fixed-size intervals between t_1 and t_4.

Obviously, the two profiles do not overlap. They also do not result in the same energy when integrated. The solid line in Figure 6 yields an energy consumption of 1066 J between times t_1 and t_4 whereas the dotted line results in an energy consumption of 1205 J over the same period, a 13% discrepancy.

5. PROPOSED APPROACH TO CORRECT THE POWER AND ENERGY MEASUREMENTS

Based on the observations described in the previous section, we have developed a methodology to accurately compute the true power and energy consumption of GPU kernels. The key insight is that the power sensor gradually approaches the true power level rather than doing so instantly. Since the 'curved' power readings between times t_1 and t_3 reminded us of capacitors charging and discharging, we tested whether the power profiles can be described by the same formulas. This turned out to work very well. We can only assume that this is the case because the power sensor hardware uses a capacitor of some sort.

Armed with this insight, it is straight-forward to determine the 'capacitance' of the power sensor by minimizing the error between the measured and the modeled data. For example, we used a single capacitor function to approximate the curve between times t_1 and t_2 in Figure 1 and determined the value of the capacitance that minimizes the sum of the differences between the measured values and the function values. As the capacitance is constant, it only needs to be established once for a given GPU. We found C = 833.333 s on all tested K20 GPUs. Computing the true instant power P_{true} then becomes a simple function of the slope of the power profile dP/dt and the measured power P_{meas} at time t_i. The following is the symmetric discrete solution where C is the GPU capacitance from above:

$$P_{true}(t_i) = P_{meas}(t_i) + C \times (P_{meas}(t_{i+1}) - P_{meas}(t_{i-1})) / (t_{i+1} - t_{i-1})$$

Based on this equation, we recommend the following approach for measuring and computing the true power of a GPU:

1. Sample at least at 66.7 Hz and include time stamps.
2. Remove consecutive samples of the same value that are no more than 4 ms apart as they do not constitute new data.
3. Compute the true power with the above equation, using neighboring power samples to approximate the slope.
4. Compute the true energy consumption by integrating the true power, using the time stamps, over all intervals where the power level is above the 'active idle' threshold of 52.5 W.

6. VALIDATION RESULTS

This section presents results obtained with our approach and validates them in multiple ways. First, we repeat some of the above studies and check whether our approach resolves the discovered anomalies. Then we apply our approach to other systems and GPUs. Finally, we investigate a complex GPU application. We sample the power at the maximum rate.

6.1. Single homogeneous kernel

Since the power curve rises at the same speed regardless of how long a kernel runs (*cf.* Section 4.1), and the behavior after a kernel stops is also largely independent of the kernel (*cf.* Section 4.2), the impact of these oddities on the total energy can be made insignificant by sufficiently increasing the runtime. After all, for very long runs, the

power curve is close to the asymptotic power level for most of the runtime. For homogeneous kernels, the energy can thus be computed with minimal error as the runtime times the asymptotic power. By scaling this energy appropriately, we obtain the expected true energy consumption of a shorter run. In other words, for very regular kernels, we hypothesize the energy consumption to be the runtime times the asymptotic power. Since the NB force calculation is such a regular kernel, we should be able to validate our approach on it.

Our approach yields the *corrected* power profile shown in Figure 7 for the NB force-calculation kernel. The dotted line represents the raw power measurements whereas the oscillating solid line shows the computed power draw between times t_1 and t_3. Since the profile is rectangular before t_1 and after t_3, the computed and actual power is the same in those regions.

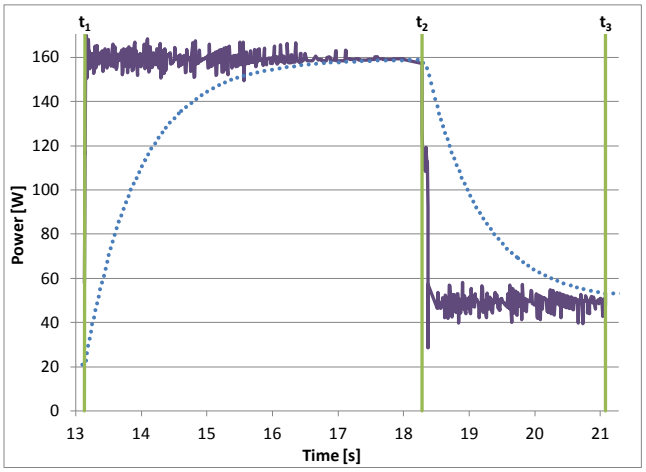

Figure 7: Corrected power profile (solid line) and raw profile (dotted line) when running the NB force calculation kernel once with 700,000 bodies

We observe that the computed power profile, at last, closely follows the GPU kernel activity. In particular, it almost instantly shoots up when the kernel starts, stays at a (more or less) constant level during execution, and almost instantly drops to the aforementioned 52.5 W after the kernel stops. Importantly, the power level during execution coincides with the asymptotic power between t_1 and t_2, which verifies the above hypothesis.

Note that the power level after the execution coincides with the asymptotic power between t_2 and t_3, which further validates our approach. This power level is identical to the 'active idle' level of 52.5 W at the beginning of the t_3 to t_4 phase (*cf.* Section 4.2). Together with the fact that the driver/hardware seems to be busy when stepping down the power level in the t_3 to t_4 phase, we can now say with confidence that the active-idle power is likely maintained by the GPU for a while in anticipation of a new kernel launch. If no kernel is launched within a given period, which seems to be around 3.5 seconds, the driver lowers the power step by step every second until reaching the idle power.

Assuming this is correct, it is now clear that the power should be integrated from time t_1 to time t_2 to obtain the kernel's energy consumption. Any energy consumption by the GPU before or after the kernel execution is due to idling (at different power levels) and is a function of time but independent of the kernel.

The corrected power profile in Figure 7 is not quite rectangular but exhibits wobbles. These are errors in the finite-precision measurements of the power sensor (whose resolution is about 10mW), which are magnified by the slope of the curve. This is why the wobbles are larger where the curve is steeper. Fortunately, these errors tend to average out, resulting in accurate energy integrals. The wobbles can easily be smoothened by using a higher-order formula to calculate the slope, *i.e.*, by using more than two neighboring power samples.

6.2. Revisiting the Anomalies

In Section 4.1, we noted two anomalies. The first was that doubling the kernel runtime resulted in more than twice the energy consumption. Looking at the corrected power profile from Figure 7, we now find that doubling the runtime doubles the energy, as we would expect. Hence, the first anomaly is resolved, which further validates our approach.

The second anomaly was that running the same kernel twice in close temporal proximity inflates the energy consumption of the second invocation. Figure 8 shows that our approach also resolves this anomaly as the two corrected profiles are now at the same level (and the power level between the kernel runs is at the active-idle level). In other words, the computed power profile of a kernel is unaffected by prior kernel runs, which is an important advantage of our approach. This means that we do not have to delay kernel runs until the GPU reaches its idle power level before we can measure the energy consumption of the next kernel.

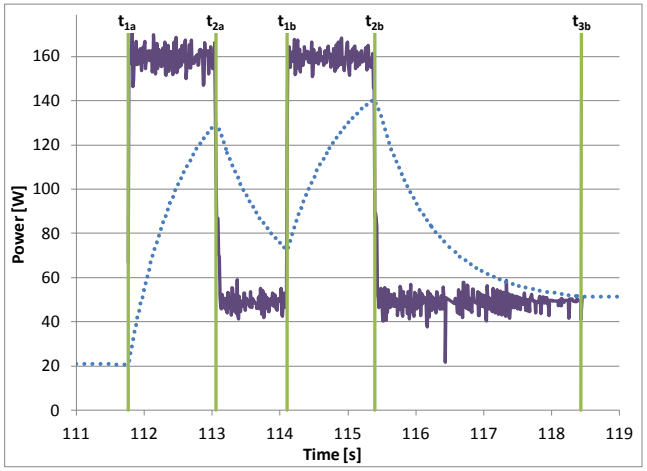

Figure 8: Corrected power profile when running the NB force calculation kernel twice with a 1-second delay between the two runs with 350,000 bodies

6.3. Other GPUs

To ensure that the observed behavior of our GPU's power sensor is not due to a hardware (or software) problem, we repeated the experiment shown in Figure 7 on other systems with different GPUs and different driver versions.

Figure 9 shows the obtained profile on another system that also houses a K20c GPU, *i.e.*, the same type of GPU that we have used for all of our measurements up to this point. Clearly, the power sensor in the second GPU behaves nearly identically to the sensor in our primary GPU, including all the described oddities. The only notable difference is that all of the measurements are a few watts higher on the second GPU. The difference is, however, within the 5 W absolute accuracy of the sensor.

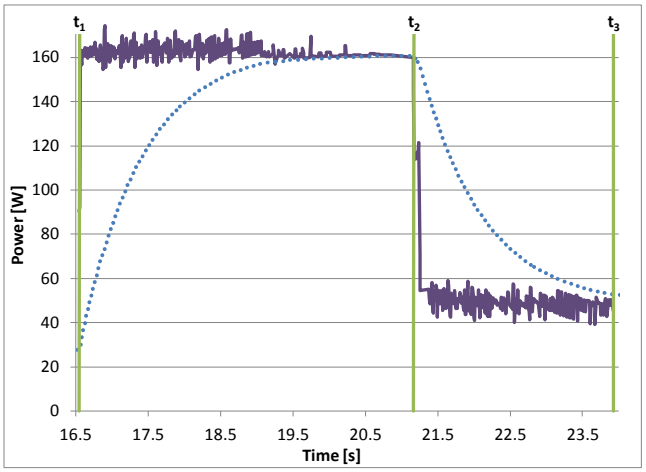

Figure 9: Corrected and raw power profiles when running the NB force calculation kernel once with 700,000 bodies on another K20c GPU

Figure 10 displays the power profile measured on a K20m GPU. We obtained a very similar profile on a second K20m and therefore only show one of them. Again, the general pattern remains the same. However, this GPU runs the kernel at a noticeably higher asymptotic power level of about 168 W.

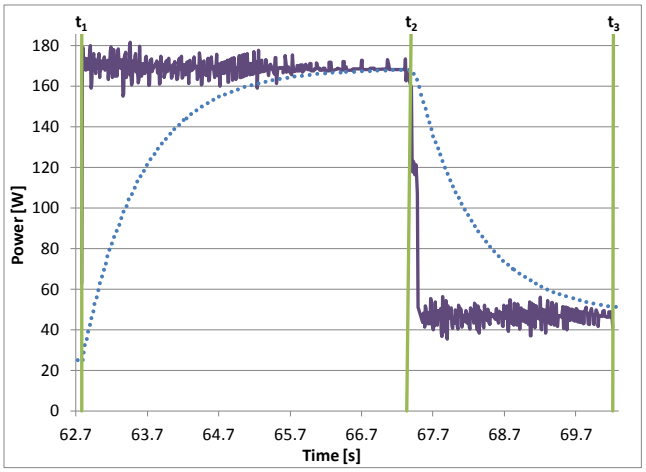

Figure 10: Corrected and raw power profiles when running the NB force calculation kernel once with 700,000 bodies on a K20m GPU

Figure 11 displays the power profile measured on a K20x GPU. Again, a second K20x delivered an almost identical profile. Surprisingly, the GPU power sensor on the K20x behaves very differently. In particular, it seems to measure the true instant power. This result verifies that our approach accurately corrects the power profiles on K20c and K20m GPUs. Moreover, it also works on a K20x. After all, integrating the unnecessarily corrected power profile yields nearly the same energy as integrating the raw measurements since the spikes contribute almost nothing to the integral and tend to average out. For the data shown in Figure 11, integrating the raw power values from t_1 to t_2 results in 680 J whereas integrating the 'corrected' power results in 687 J, merely a 1% difference. Most of this discrepancy is due to the relatively wide cut-off spike just after t_1. Note that the large drop after time t_2 is due to the by far longest non-response we have measured on any GPU. In this case, it takes the driver nearly 600 ms to return a single power measurement.

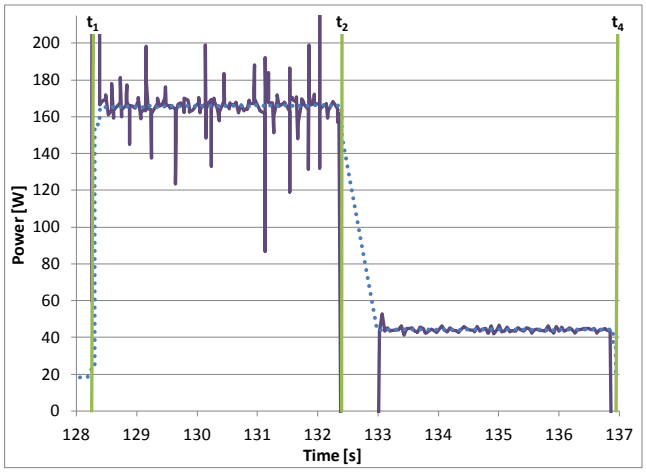

Figure 11: Corrected and raw power profiles when running the NB force calculation kernel once with 700,000 bodies on a K20x GPU

6.4. Complex GPU Application

Figure 12 displays the raw power measurements as well as the corrected power profile for the complex Barnes-Hut (BH) *n*-body code. It contains several distinct kernels, most of which operate on an unbalanced octree data structure. The source code is instrumented to measure the runtime of each kernel. We ran the BH code with 22 million bodies (and one time step) to obtain a sufficient number of power samples for more than just the dominant kernel. Nevertheless, some of the kernels execute in under 2 ms and are therefore not visible in the power profile. The figure extends from the start of the first kernel to the end of the last kernel.

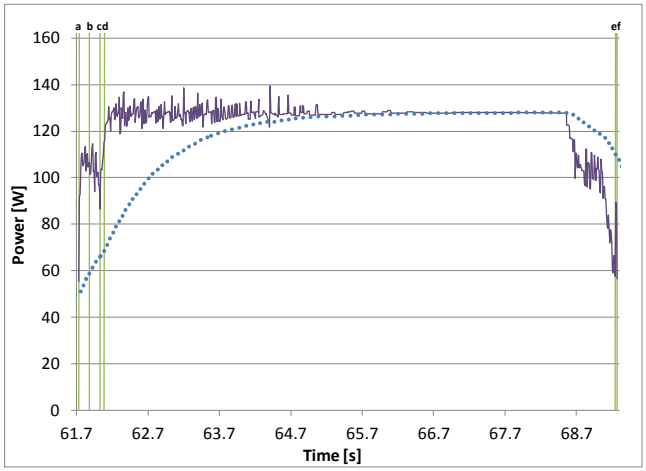

Figure 12: Raw power measurements and corrected power profile when running the full BH application for one time step with 22 million bodies

There are several interesting observations. First, the asymptotic power level is substantially lower for BH than for NB. Second, the raw power data follows the previously observed general shape but exhibits slight deviations. Third, the power starts dropping before the end of the computation.

The BH power tops out at 128 W (compared to 158 W for the NB experiments). This lower level is reasonable as the BH code is irregular and therefore utilizes the GPU hardware less effectively. In particular, the streaming multiprocessors spend more time stalling

(idling) when running the BH code, which reduces the power draw. Our corrected power profile reflects this nicely. All of the kernels running in the a-d phase are highly irregular and, hence, reach a lower power level than the main kernel running in the d-e phase, which is implemented in a warp-centric way and therefore partially regularized.

Note that these differences are almost undetectable and hard to interpret in the raw power data but quite obvious in the corrected profile. Clearly, the slight deviations from a 'smooth' profile are actually quite important. In fact, we now see that the kernel running in the a-b phase, which builds the octree top down, results in a similar power draw as the kernel running in the b-c phase, which performs a bottom-up traversal of the tree. Both of these kernels run for about 150 ms. In contrast, the kernel running for 50 ms in the c-d phase, which sorts the data, spends a lot of time waiting, which is why there is a noticeable drop in the power draw. Interestingly, there is a big spike in the very short e-f phase, which is due to a regular kernel. Even though its runtime is only 8 ms, which corresponds to only half a power sample interval, we can see it in the corrected profile.

The dominant force-calculation kernel, which performs many different partial top-down traversals of the octree, draws a mostly constant amount of power except towards the end, where the power drops off. This drop-off is due to load imbalance, which results in some SMXs running out of work prematurely. This imbalance demonstrates that our approach computes the instant power rather than simply predicting the stable power.

7. SUMMARY AND FUTURE WORK

To enable programmers to study and reduce the energy consumption of their GPU codes, accurate energy measurements are needed. However, the simple approach of sampling the power, computing the average, and multiplying by the runtime can result in large errors when using the K20's built-in power sensor. This is due to a number of complications. For example, our study shows that 1) the power profile is distorted with respect to the kernel activity, 2) the measured power lags behind the kernel activity, 3) running multiple kernels one after another inflates the power draw of the later kernels, 4) after a short-running kernel, the power draw can even increase for a while, 5) integrating the power to a discernable time after a kernel stops does not correctly compensate for the power lag, 6) the sampling interval lengths vary greatly, 7) the GPU sensor only performs power measurements once in a while, 8) the true sampling rate may be too low to accurately measure short-running kernels, and 9) the PCI-bus activity is not included in the sensor's measurements.

This paper proposes and evaluates a power- and energy-measurement methodology that is accurate even in the presence of the above problems. It computes the true instant power consumption from the measured power samples, accounts for variations in the sampling frequency, and correctly identifies when kernels are running and when the GPU is in active-idle mode waiting for the next kernel launch. Furthermore, it is insensitive to earlier activities on the same GPU and works on different types of K20 GPUs. We also provide guidelines on how long the kernel runtime should minimally be to obtain accurate energy results. We validated our methodology using multiple systems, GPU types, scenarios, and CUDA programs.

In future work, we plan to use our methodology to investigate the power and energy consumption of a wide range of GPU applications and the energy efficiency of different code optimizations. Furthermore, we want to add smoothing and support for multi-GPU nodes to our tool. Finally, we intend to automatically combine the results from multiple compute nodes to provide aggregate energy numbers in cluster environments.

8. ACKNOWLEDGMENTS

The work reported in this paper is supported by the U.S. National Science Foundation under Grants DUE-1141022, CNS-1217231, CNS-1305359, a grant from the Texas State University Research Enhancement Program as well as grants and equipment donations from NVIDIA Corporation.

9. REFERENCES

[1] Fermi Compute Architecture Whitepaper, NVIDIA.

[2] http://top500.org/lists/2012/11/

[3] Kepler Compute Architecture Whitepaper, NVIDIA.

[4] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical Power Modeling of GPU Kernels Using Performance Counters", in Proceedings of the Green Computing Conference, 2010.

[5] S. Hong and H. Kim, "An Integrated GPU Power and Performance Model", in Proceedings of the 37th International Symposium on Computer architecture, 2010.

[6] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data", in Proceedings of the 36th IEEE/ACM International Symposium on Microarchitecture, 2003.

[7] J. W. Choi and R. Vuduc, "A Roofline Model of Energy", Technical Report GT-CSE-2012-01, Georgia Tech, 2012.

[8] S. W Song, C.Y. Su, B. Rountree, and K. W. Cameron, "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures", in Proceedings of the IEEE International Parallel and Distributed Processing Symposium, 2013.

[9] G. Contreras and M. Martonosi, "Power Prediction for Intel Xscale Processors Using Performance Monitoring Unit Events", ISLPED 2005.

[10] D. Brooks *et al.*, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", in Proceedings of the ACM/IEEE International Symposium on Computer Architecture, 2000.

[11] S. Li *et al.*, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", in Proceedings of the IEEE/ACM International Symposium on Microarchitecture, 2009.

[12] S. Lal, J. Lucas, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "A Framework for Modeling GPUs Power Consumption", LPGPU Workshop on Power-Efficient GPU and Many-core Computing in conjunction with the HiPEAC Conference, 2013.

[13] J. Lucas, S. Lal, M. Andersch, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a Single Chip Causes Massive Power Bills GPUSimPow: A GPGPU Power Simulator", in Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2013.

[14] J. Leng, S. Gilani, A. El-Shafiey, T. Hetherington, N. Sung Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimization in GPGPUs", in Proceedings of the International Symposium on Computer Architecture, 2013.

[15] Jianmin Chen *et al.*, "Statistical GPU Power Analysis Using Tree-based Methods", in Proceedings of the Green Computing Conference, 2011.

[16] X. Ma *et al.*, "Statistical Power Consumption Analysis and Modeling for GPU-based Computing", in Proceeding of ACM SOSP Workshop on Power Aware Computing and Systems. 2009.

[17] S. Ghosh, S. Chandrasekaran, and B. M. Chapman, "Statistical Power and Energy Modeling of Multi-GPU Kernels", http://www2.cs.uh.edu/ hpctools/pub/SC12-Poster.pdf.

[18] S. Ghosh, S. Chandrasekaran, and B. M. Chapman, "Power and Energy Prediction of Multi-GPU Kernels using Non-linear Regression", GTC 2013 Poster.

[19] https://www.wattsupmeters.com

[20] R. Ge, X. Z. Feng, S.W. Song, H.C. Chang, D. Li, and K.W. Cameron, “Powerpack: Energy Profiling and Analysis of High-Performance Systems and Applications”, IEEE Transactions on Parallel and Distributed Systems, 21(5):658-671, May 2010.

[21] http://www.cs.vt.edu/node/4666

[22] D. Bedard, M. Lim, R. Fowler, and A. Porterfield, “Powermon: Fine-Grained and Integrated Power Monitoring for Commodity Computer Systems”, in Proceedings of the IEEE Southeastcon Conference, 2010.

[23] Bedard, M. Lim, R. Fowler, and A. Porterfield, “PowerMon 2: Fine-grained, Integrated Power Measurement”, RENCI Technical Report TR-09-04, Renaissance Computing Institute, 2009.

[24] J. H. Laros III, K. T. Pedretti, S. M. Kelly, J. P. Vandyke, K. B. Ferreira, C. T. Vaughan, and M. Swan, “Topics on measuring real power usage on high performance computing platforms,” in Proceedings of the IEEE International Conference on Cluster Computing, September 2009.

[25] J. H. Laros III, K. T. Pedretti, S. M. Kelly, W. Shu, and C. T. Vaughan, “Energy based performance tuning for large scale high performance computing systems,” in Proceedings of the 20th High Performance Computing Symposium, March 2012.

[26] J. H. Laros III, K. T. Pedretti, S. M. Kelly,W. Shu, K. Ferreira, J. Van Dyke, and C. T. Vaughan, “Energy-Efficient High Performance Computing - Measurement and Tuning“, Springer, ISBN 978-1-4471-4492-2, 2012.

[27] J. H. Laros III, P. Pokorny, and D. Debonis, “PowerInsight - A Commodity Power Measurement Capability”, in Proceedings of the 4th International Green Computing Conference, June 2013.

[28] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, “Fine-grained Power Modeling for Smartphones using System Call Tracing,” in Proceedings of the Sixth Conference of Computer Systems, 2011.

[29] A. Pathak, Y. C. Hu, M. Zhang, “Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof”, in Proceedings of the Seventh ACM European Conference of Computer Systems, 2012.

[30] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, “Energy Consumption in Mobile Phones: a Measurement Study and implications for network applications,” in Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, 2009.

[31] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao, and L. Yang, “Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones,” in Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, 2010.

[32] H. C. Plummer, “On the Problem of Distribution in Globular Star Clusters”, Mon. Not. R. Astron. Soc., 71:460, 1911.

[33] J. Barnes and P. Hut, “A Hierarchical O(n log n) Force-calculation Algorithm”, Nature, 324(4):446-449, 1986.

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

[35] NVIDIA, “Nvidia management library”, 2011. Available: http://developer.nvidia.com/nvidia-management-library-nvml