

To the Graduate Council:

I am submitting herewith a thesis written by Kiran Kumar Kasichayanula entitled “Power Aware Computing on GPUs.” I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Dr. Gregory D. Peterson, Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Robert J. Harrison

Dr. Shirley Moore

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

To the Graduate Council:

I am submitting herewith a thesis written by Kiran Kumar Kasichayanula entitled "Power Aware Computing on GPUs." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Dr. Gregory D. Peterson, Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Robert J. Harrison

Dr. Shirley Moore

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Power Aware Computing on GPUs

A Thesis Presented for

The Master of Science

Degree

The University of Tennessee, Knoxville

Kiran Kumar Kasichayanula

May 2012

© by Kiran Kumar Kasichayanula, 2012
All Rights Reserved.

To my beloved parents Sri. K. LakshmiNarayana and Smt. K. Manoranjani

Acknowledgements

At the outset, I am thankful to my advisors and mentors, Dr. Gregory D. Peterson, Dr. Shirley Moore, Dr. Dan Terpstra, and Dr. Robert Harrison for their guidance and support throughout the course of my education. This work is a result of their encouragement, timely ideas and constructive criticism. I would like to thank Dr. Stanimire Tomov and Dr. Piotr Luszczek for their insight on GPUs which gave me a complete overview of the fundamentals underlying this thesis. Many of the ideas and concepts that I learnt at ICL (Innovative Computing Laboratory) served me at critical points and stumbling blocks in this work. I am thankful to Dr. Jack Dongarra for his continued support throughout the my stay at ICL. Further, I am also thankful to my colleagues James Ralph, Heike Jagode, Sam Crawford, Dr. Vince Weaver, and others for their helpful suggestions. I would also like to thank Dr. Haihang You for his support, and the National Institute of Computational Science (NICS) for providing me with access to their resources.

This work has been partially supported by NSF grant CNS0910899 and by DOE SciDAC grant DE-SC0006733. Furthermore, I am thankful to NVIDIA for providing me with GPUs and insight into the NVIDIA Management Library, both of which were invaluable resources in my research.

I would also like to thank my friends Shanthan Reddy Mudhasani, Shanawaz Shaik, Revanth Kollipara, and Anirudh for their encouragement.

Winners don't make excuses!

Abstract

Energy and power density concerns in modern processors have led to significant computer architecture research efforts in power-aware and temperature-aware computing. With power dissipation becoming an increasingly vexing problem, power analysis of Graphical Processing Unit (GPU) and its components has become crucial for hardware and software system design. Here, we describe our technique for a coordinated measurement approach that combines real total power measurement and per-component power estimation. To identify power consumption accurately, we introduce the Activity-based Model for GPUs (AMG), from which we identify activity factors and power for microarchitectures on GPUs that will help in analyzing power tradeoffs of one component versus another using microbenchmarks. The key challenge addressed in this thesis is real-time power consumption, which can be accurately estimated using NVIDIA's Management Library (NVML) through Pthreads. We validated our model using Kill-A-Watt power meter and the results are accurate within 10%. The resulting Performance Application Programming Interface (PAPI) NVML component offers real-time total power measurements for GPUs. This thesis also compares a single NVIDIA C2075 GPU running MAGMA (Matrix Algebra on GPU and Multicore Architectures) kernels, to a 48 core AMD Istanbul CPU running LAPACK.

Contents

List of Tables	x
List of Figures	xi
1 Introduction	1
1.1 Nvidia Management Library	7
1.2 Primary Contributions of this work	8
2 Related Work	9
2.1 Summary	14
3 Power Measurement on Nvidia Fermi C2075	15
3.1 Fermi C2075 GPU	15
3.1.1 Memory	16
3.2 Measuring Power Consumption using external device	17
3.3 Power Measurement using NVML	18
3.4 PTX analysis	19
3.5 Frequency measurement of NVML Sensor	20
3.6 Summary	22
4 Activity-based Model for GPUs (AMG)	23
4.1 Idle Power	24
4.2 Run-time Power Consumption	24

4.2.1	Floating Point Operations	26
4.2.2	Shared Memory	27
4.2.3	Global Memory	29
4.3	Validation	30
4.4	Power and Temperature relationship	33
4.5	Summary	34
5	Power Analysis of MAGMA Kernels	36
5.1	BLAS 2 and BLAS 3 Kernels	37
5.1.1	Measuring time taken in MAGMA Kernels	39
5.2	MAGMA LU Factorization and Hessenberg	44
5.2.1	Reducing Noise in Power measurements	48
5.2.2	MAGMA Hessenberg	49
5.3	Predictions for MAGMA kernels for matrix of size 10K based on AMG	50
5.3.1	Power prediction for MAGMA DGEMM	51
5.3.2	Power prediction for MAGMA DGEMV	51
5.3.3	Power prediction for MAGMA DGETRF	52
5.3.4	Power prediction for MAGMA DGEHRD	52
5.3.5	Analysis of predictions for MAGMA kernels using AMG	53
5.4	Summary	53
6	Performance Application Programming Interface	55
6.1	PAPI NVML Component	56
6.2	Other Power Monitoring Components	57
6.2.1	PowerMon 2	57
6.2.2	Running Average Power Limit component	58
6.3	Summary	60
7	Conclusions	61
7.1	Future Work	62

Bibliography	63
Vita	69

List of Tables

4.1	Power consumption of various units	26
4.2	Base power for various units	26
4.3	Shared Memory	28
4.4	Matrix Matrix Multiply example	32
4.5	Matrix Matrix Multiply evaluated using AMG	32
5.1	Power consumption of MAGMA BLAS2 and BLAS3 kernels	43
5.2	GPU vs CPU	44
5.3	DGEMM calls in MAGMA LU	48
5.4	DGEMM and DGEMV calls in MAGMA Hessenberg for a matrix of size 10 K	50
5.5	Average power consumption for a matrix size 10 K	50

List of Figures

1.1	Multicore Era [22]	2
1.2	5 Decades of Linpack [22]	3
1.3	Performance of top 20 systems	4
1.4	Power of top 20 systems	5
1.5	Top 20 systems which have highest performance/power	6
3.1	Memory Layout	16
3.2	Validation using external power supply	18
3.3	Power sampling frequency 62.5 Hz	20
3.4	Power sampling frequency 200 Hz	21
3.5	Power sampling frequency 625 Hz	21
4.1	Average Power consumption of Floating point operations	27
4.2	Power consumed by shared memory without bank conflicts	28
4.3	Power consumed by global memory with coalesced memory accesses	29
4.4	Power consumed by global memory with noncoalesced memory access	30
4.5	Increase in power consumption with temperature for FLOPs benchmark	34
4.6	Increase in power consumption with Temperature for Global Memory benchmark	35
5.1	GPU GEMMs on Fermi architecture	38
5.2	Performance of single and double precision MAGMA BLAS 3 Kernels	38
5.3	Performance of single and double precision MAGMA BLAS 2 Kernels	39

5.4	Average Power consumption of single and double precision MAGMA BLAS3 Kernels	40
5.5	Average Power consumption of single and double precision MAGMA BLAS2 Kernels	41
5.6	Energy consumption of double precision MAGMA BLAS3 Kernels . .	41
5.7	Energy consumption of single precision MAGMA BLAS3 Kernels . .	42
5.8	Energy single precision MAGMA BLAS2 Kernels	42
5.9	Energy consumed by double precision MAGMA BLAS2 Kernels . . .	43
5.10	Dgetrf and Dgehrd comparison between GPU and CPU	45
5.11	Dgetrf	46
5.12	MAGMA Dgetrf power consumption for a 10k matrix	47
5.13	MAGMA Dgehrd power consumption for a 10k matrix	49
6.1	Graphical representation of RAPL	59

Chapter 1

Introduction

With power consumption and heat dissipation issues pushing multi-core CPUs to the limit the importance of a Graphical Processing Unit (GPU) cannot be emphasized enough. Figure 1.1 from [22] shows us a time line of the beginning of heat dissipation and cooling problems. One thing to note about this figure is that, while the single core era ends, the multi-core era begins. The two forms of scalability are strong scaling (Amdahls Law) and weak scaling (Gustafsons Law). Strong scaling is defined as how the solution time varies with the number of processors for a fixed total problem size. Weak scaling is defined as how the solution time varies with the number of processors for a problem size that scales per processor. These two laws are very important for machine processing speed. While the memory bandwidth and latency issues stall a CPU, a GPU may outperform a CPU in these aspects. For example the memory bandwidth for Nvidia C2075 is 144 GB/s. The growth rate of performance of GPUs has increased by at least a factor of 2 compared to that of CPU. A GPU can be used for parallel computing using stream processing [16]. Stream processing is related to Single Instruction Multiple Data (SIMD). GPU accelerated computing systems have drawn the attention of researchers because they have tremendous computational power and high memory bandwidth, and are inherently well suited for massively data parallel computation. In the November 2011 ranking, 39 of the Top 500 computer systems

utilized GPUs, up from 17 systems listed in the June 2011 ranking [7]. The Top 500 lists the fastest supercomputers in the world, and the performance benchmarks and statistics are of major interest to users and manufacturers. Every six months, in June and November, a new Top 500 list is released. The Top 500 list is compiled based on results from the LINPACK benchmark, which solves dense linear equations, and measures the number of floating point instructions per second (FLOPS) that the benchmarked machine can run. We show performance improvement achieved in 5 decades [7] in Figure 1.2. From 2001-2010, performance on LINPACK for the #1 system on the Top500 list grew at compound rates of 92% , which is astounding.

Coming along with this exciting computational capability, the power consumption of supercomputers has become a serious issue. For example, the average power

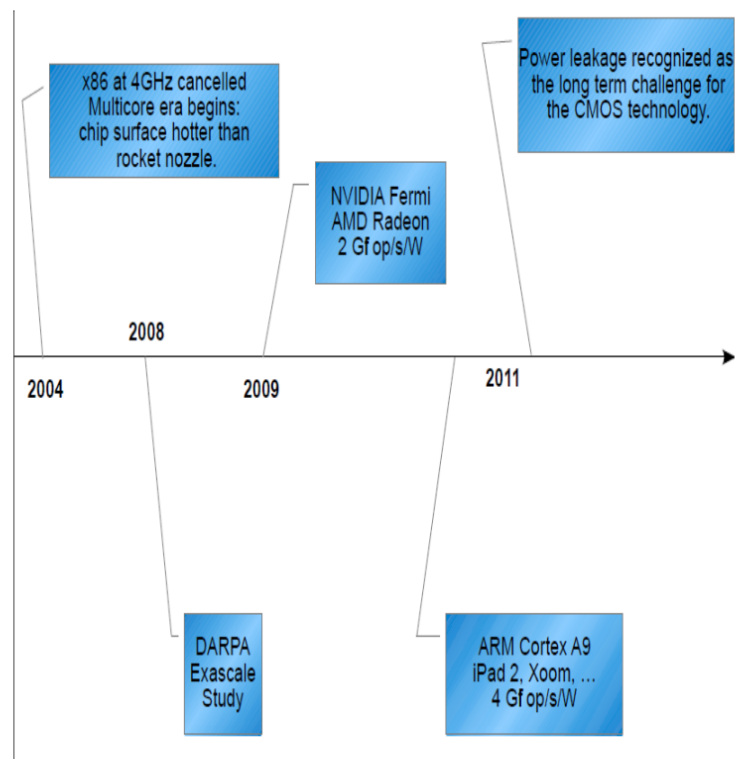


Figure 1.1: Multicore Era [22]

consumption of the TOP 10 supercomputing centers was 1.32 MW in 2008, and climbed to 3.2 MW in 2010, translating to a multi-million-dollar electric bills. Designers must employ aggressive techniques to keep the ballooning energy cost under control. The consequences of growing energy consumption are more complex cooling solutions and noisier fans. Cooling modern video cards is becoming much more difficult, especially when users are asking for quiet cooling solutions.

Figures 1.3 and 1.4 show us the performance and power consumption of the top 20 systems. The power consumption increases with performance, and engineers are now paying more attention to power consumption for new GPU designs. The cost to maintain such huge machines is expensive too, e.g. 12MW at \$0.10/kW-h is \$1200 an hour or about \$10.5 million per year.

Figure 1.5 shows us GFLOPS to power consumption [7]. For example, the latest number 1, i.e, the K supercomputer, has achieved maximum LINPACK performance of 10510.00 TFLOPS, but consumes 12.659 MW [7]. The Green 500 [35] provides a

5 Decades of the LINPACK: the LINPACK Law

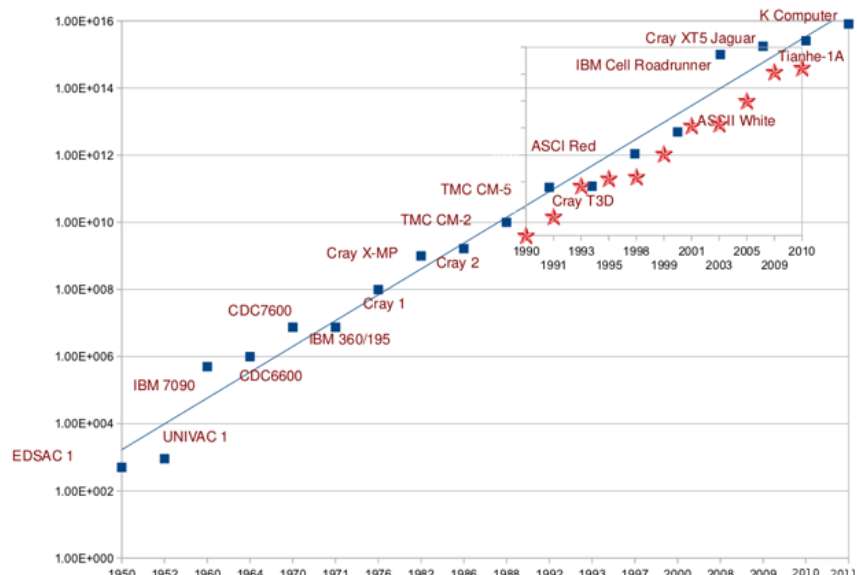


Figure 1.2: 5 Decades of Linpack [22]

ranking of the most energy efficient supercomputers in the world, in order to encourage users and vendors to be more energy efficient. This list tries to emphasize that energy efficiency is as important as performance. On the Green 500 list a supercomputer has to be at least a supercomputer on the Top 500 list.

With the announcement of the Titan [21], which promises to deliver 10-20 PFLOPS and the fact that 85% of its peak performance comes from GPUs, we simply cannot ignore the energy consumed by GPUs. The current Nvidia C2075 GPU consumes 220W Thermal Design Power (TDP) and delivers 515 GFLOPS theoretical peak of which 300 GFLOPS can be achieved for Double Precision General Purpose

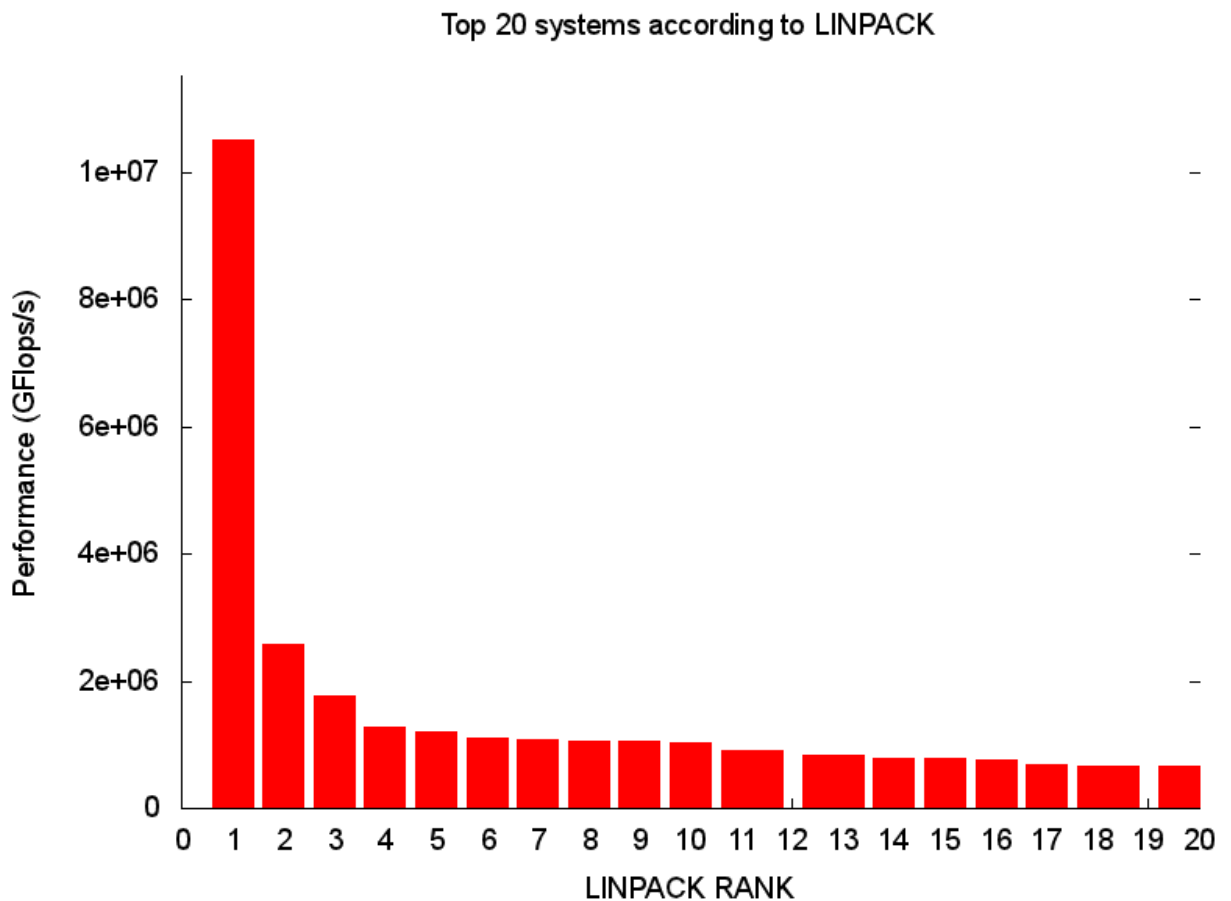


Figure 1.3: Performance of top 20 systems

Matrix Matrix Multiply (DGEMM) performance compared to the previous generation C1060 which consumes 200W but delivers only 78 GFLOPS of DGEMM performance.

Energy consumption concerns HPC systems, with systems requiring more and more computational power. With the increase in computational power the power consumed by the system tends to increase. With voltage scaling slowing down and leakage current increasing, the answers to this problem seem limited. Figure 1.5 shows the 50 most energy efficient systems according to the Green 500 [35]. With the evolution of GPUs systems become more energy efficient.

Though prior work has been done on power measurement of GPUs, [26, 23, 8], the real-time measurement of individual GPU components using a software approach, is

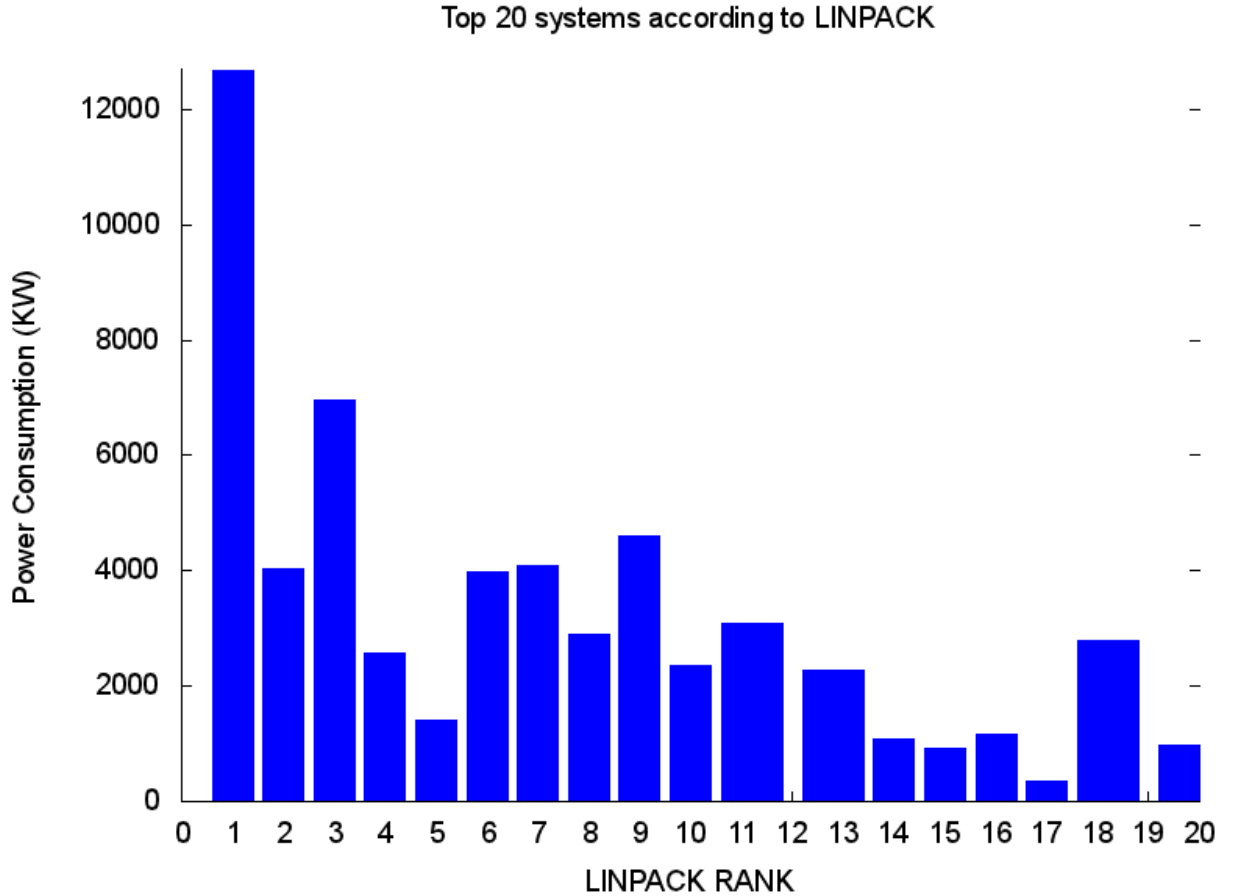


Figure 1.4: Power of top 20 systems

new. In this work, we develop our model to measure real-time power usage of micro-architectures running representative computational kernels through the use of NVML (Nvidia Management Library) [19].

We refer to estimating at this granularity as per-structure power estimation. Per-structure power estimation is useful for selectively enabling and disabling micro-architectural resources. As power-management becomes increasingly important, coarse-granularity power estimation is likely to become inadequate to manage and continuously reallocate power budgets for individual micro-architectural structures. To address the challenges of estimating per-structure power in hardware, we propose a new analytical model, called the Activity-based Model for GPUs (AMG), to estimate

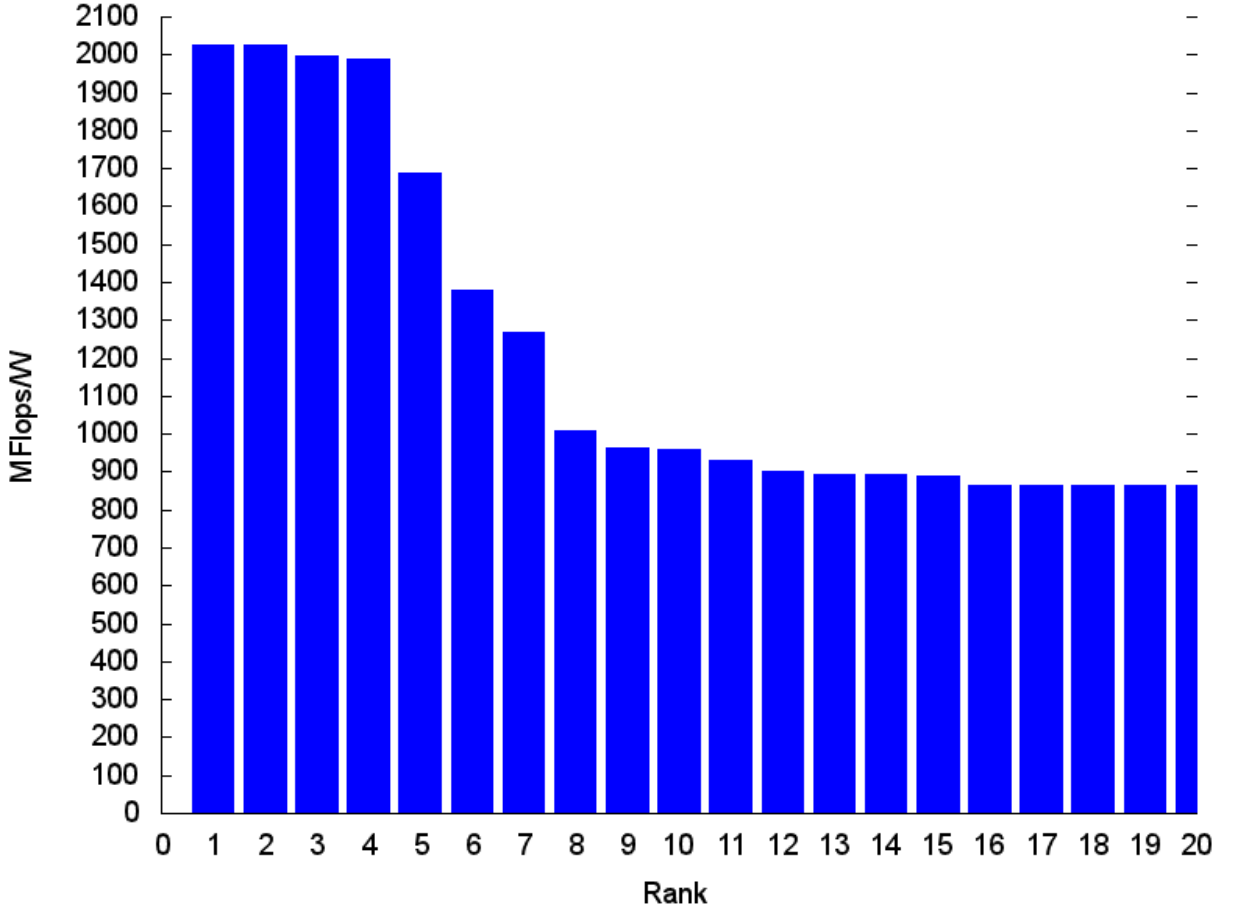


Figure 1.5: Top 20 systems which have highest performance/power

activity factors and power for micro-architectural structures on GPUs. This model does not rely on real-time current monitoring or simulating hundreds of utilization statistics similar to [27].

We maintain that only a few input statistics are sufficient to estimate per-structure dynamic power of a GPU because the myriad per-structure events are related to a small set of global parameters, such as load rate or the execution time of that unit. We use this key observation to drive the development of AMG. Using minimal input data, AMG’s linear-regression-based methodology can estimate activity for tens or hundreds of micro-structures. We first analyze the correlation of a variety of performance metrics. Then we monitor only the least correlated metrics and use the monitored metrics to extrapolate the metrics of interest. After we get all the desired metrics about the structure events, we apply a per-event energy model derived from a circuit model, to those structures to calculate the power consumption of each structure. We also show power vs temperature of several kernels.

Researchers in various fields have investigated the advantages of using GPUs as compared to CPUs, GPUs not only provide high performance, but they are also more energy efficient than CPUs [12]. High-end GPUs do consume more power than CPUs, but the GPU/CPU performance ratio is higher than the power consumption ratio, and thus a GPU can complete more computations per watt than a CPU. We demonstrate this fact by comparing an Nvidia TESLA C2075 GPU running MAGMA [29] with a 48 core AMD Istanbul CPU running LAPACK.

1.1 Nvidia Management Library

NVML is a C-based interface for monitoring and managing various states within Nvidia Tesla GPUs [19]. NVML has several functions that can measure characteristics of GPUs, such as device power, device temperature, unit power, unit temperature, and clock frequency. Using NVML, we measure power and temperature. We implemented a PAPI (Performance application Programming Interface) component that measures

power and temperature using NVML, thus allowing power and energy consumption measurements to be obtained from the familiar PAPI interface, with the capability of reading other, simultaneously obtained, PAPI metrics. More about the Fermi C2075 and NVML can be found in [chapter 3](#)

1.2 Primary Contributions of this work

- The AMG model for real-time measurement of power and energy consumption on GPUs;
- Per-component analysis of power consumption of different GPU components like floating point units, shared memory, and global memory;
- PAPI NVML component that offers a way to measure power and energy consumption in real-time;
- Energy consumption comparison of linear algebra routines between a GPU and a multi-core system.

Chapter 2

Related Work

Early work focussed on measurement of power dissipation using external devices such as clamp probes [26]. Use of probes to measure voltage and current is a very tedious and time consuming process as a probe requires direct connection to PCI-Express and auxiliary Power lines. A considerable amount of power is spent on data acquisition and control of measurement equipment. The use of markers is clearly explained and they have used previous marker positions to estimate the next marker position using the matching method by [26]. The performance and power relationship they have derived is

$$W = 72 + 1.02 * 10^{10} \rho \quad (2.1)$$

$$\rho = \begin{cases} 1 & \text{if the threads per block is a multiple of 16,} \\ 1 + 0.71 \times (16 - (\theta \bmod 16))/\theta & \text{otherwise} \end{cases} \quad (2.2)$$

where θ is the number of threads per block

The root mean square error of their approximation is 0.29 W [26].

Using hardware devices might be problematic especially since we need a separate

power supply and devices like Kill-A-Watt lack a method to log the data automatically. [11]

There has been other research on the power consumption of hybrid architectures such as CPU-GPU platforms, but our work emphasizes the GPU as an independent component. [23] used a LEAP-Server to monitor power of subcomponents of a system such as GPU with micro-scale capability. Their analysis was based on Low Power, Energy based Processing (LEAP2) which has a resource multiplexer that adds components through a set of peripherals and sensors, but the downside of this is we cannot have an accurate estimate of power consumption as, the host processor does not facilitate dedicated point-to-point connections. Their method requires hardware expertise of not only the LEAP-Server but also of the complete system.

Ischi et al. developed a hardware based counter model for power estimation of sub-components of a CPU [10]. A combination of P4 hardware performance events were used to estimate the power. The counter based run time for power monitoring is based on access rate heuristics, which can be used as weights to analyze power from run-time power. Although the model they developed is for a Pentium P4 CPU, the model itself fits well with the GPU architecture. Power consumption of 22 sub-components that include bus control, L1 cache, L2 cache, L1 branch prediction unit (BPU), L2 BPU, instruction TLB & fetch, memory order buffer, memory control, data TLB, integer execution, floating point execution, integer register file, floating point register file, instruction decoder, trace cache, microcode ROM, allocation, rename, instruction queue1, instruction queue2, schedule, and retirement logic, was derived using counter-based profiling. Breakdown of components was based on physical attributes rather than conceptual grouping. Power of each sub-component was derived using the formulae below:

$$\begin{aligned}
Power(C_i) &= AccessRate(C_i) \times ArchitecturalScaling(C_i) \\
&\times MaxPower(C_i) + NonGatedClockPower(C_i)
\end{aligned} \tag{2.3}$$

$$TotalPower = \sum_{i=1}^{22} Power(C_i) + IdlePower \tag{2.4}$$

The breakdown of sub-components shown by [10] is the basis of our component analysis. In our analysis of the GPU's power consumption we do not take non-gated clock power consumption as Nvidia Fermi C2075's is not a gated clock.

Hong et al. estimated the number of cores needed for optimal power and performance using GPGPUs [8]. The theory behind this is that when a memory bound application is executed, performance does not increase proportionally with the number of cores. Their conclusions show us that by not using all the cores we can save energy up to 22.09%. They have also estimated the power consumption of sub components using micro-benchmarks in such a way that the floating point benchmark has a high number of floating point operations.

Power consumption can be divided into two parts: dynamic power and static power:

$$Power = Dynamic_{power} + Static_{power}. \tag{2.5}$$

Power consumption of sub-component of GPU can be modelled:

$$\begin{aligned}
Power(C_i) &= AccessRate(C_i) \times ArchitecturalScaling(C_i) \\
&\times MaxPower(C_i) + NonGatedClockPower(C_i)
\end{aligned} \tag{2.6}$$

$$StaticPowerP_{static} = V_{cc} \times N \times K_{design} \times I_{leak} \quad (2.7)$$

$$RP_SMs = Max_SM \times \log_{10}(\alpha \times Active_SMs + \beta) \quad (2.8)$$

$$\alpha = (10 - \alpha)/NUM_SMs \quad and \quad \beta = 1.1$$

$$Max_SM = (Num_SM \times \sum_{i=1}^n SM_Component_i) \quad (2.9)$$

$$Runtime_power = (Max_SM + RP_Memory) \times \log_{10}(\alpha \times Active_SMs + \beta) \quad (2.10)$$

The maximum power derived from these micro-benchmarks is then multiplied by the access rate, and the run time power is derived. They show us that the memory operations from global memory are not only very time consuming but also power consuming. Work has also been conducted on the number of active SMs (Streaming Processors) vs power consumption on a GTX 280 GPU, which has 240 CUDA cores and 30 SMs. Temperature modeling using the RC model, [25], shows us the relation between power and temperature. Evaluations have been conducted between memory bound kernels and kernels which are not memory bound.

Chen et al. showed us that in the previous generation GPUs there was no support for sensors to measure power, but with the evolution of the new Fermi architecture this has changed [4]. The older Fermi GPUs, like the C2050, have partially supported power measurement using power states P0-P15 where P0 is the power state when the GPU is running under full load and P15 is the idle state power consumption, while newer generations such as the C2075 have sensors which output power in watts. Chen et al. developed a GPU power consumption model based on a linear regression tree

[3], and random forest methods [2]. A regression is a statistical analysis assessing the relationship between two variables. Random forest uses various models to obtain performance, which is better than any individual method, and in this case consists of many decision trees, and returns the class that is the mode of classes output by individual trees. The most influential variables and several performance-sensitive architecture metrics were identified using the random forest model. Verification of their model was done using leave-one-out cross validation with an average percentage error of 7.77%.

Sheaffer et al. proposed the use of Qsilver to develop thermal management methods such as dynamic voltage scaling (DVS), clock gating, multiple clock domains, and temperature-aware floor plans [24]. They developed a tool for analysis of power and performance of graphics hardware and software. Chromium, which is used by Qsilver, can be used to maneuver graphics API commands on clusters of workstations [9]. The simulations which the authors developed run to evaluate different functional units which can be time-dependent. Qsilver can also be used to identify and understand performance bottlenecks and Sheaffer et al. also show us the use of dynamic voltage scaling for reducing considerable energy consumption.

The GPU architecture using micro benchmarks was explained [34]; we would like to use the same approach, but use it to characterize GPU power consumption. Branch divergence is a very important aspect which we would like to explore, and a detailed analysis is available [34]. The authors measured the execution time line for two concurrent warps in a block whose threads diverge 32 ways. This paper emphasizes some of the aspects of GPU architecture such as re-convergence, barrier synchronization in a single warp, and barrier synchronization across multiple warps. We can deduce a lot of cache characteristics using the technique they described and as long as the array fits in the cache, the latency remains the same. We used the same technique for our power analysis, which, as long as the GPU uses a single SM, the power should remain the same, and power scales with respect to the number of SMs used.

2.1 Summary

This chapter explains the related work. We have adapted parts of work from different papers such as equation 2.5 has been adopted from [10]. Our work is also unique in developing a model for the latest Fermi architecture and the model's power estimation is based on NVML-based sensors. The next chapter describes the Nvidia C2075 GPU architecture, validation using an external power monitoring device, and the Nvidia Management library.

Chapter 3

Power Measurement on Nvidia Fermi C2075

3.1 Fermi C2075 GPU

The Fermi C2075 GPU offers excellent solutions for high performance problems with 14SMs, 448 CUDA cores and 6GB of GDDR5 DRAM [17]. Each SM supports 32 CUDA cores and a fully pipelined Integer Arithmetic Logic unit (ALU), Floating Point Unit (FPU), and various levels of memory such as global memory, constant memory, texture memory, registers, shared memory, and local memory.

The release of the Fermi architecture shows the CUDA architecture is ever changing and evolving, and in each architectural generation there is major change in the way the GPU works. For example, execution of threads is different in the Fermi and Tesla architectures; on a Fermi architecture, groups of 16 cores execute warps; in Tesla 8 cores execute an instruction for a warp every four cycles. This makes the Fermi architecture a much better solution if all the threads execute the same code in a warp. The Fermi architecture can run several kernels simultaneously.

3.1.1 Memory

Global Memory

Global memory is used to allocate or copy data between the host and device (GPU). Bandwidth between host and device memory is very low compared to data transfer within the GPU, therefore communication between host and device should be minimized. There is an overhead per communication, so single large transfers are better than many small transfers. For example, in the floating point operation benchmark we use registers most of the time and write the result back to global memory.

Global memory is located in the main device memory, and data accesses from the SM to global memory are high latency (400-800 clock cycle) and low bandwidth (compared to on chip memory). The latency can be hidden to some extent if there

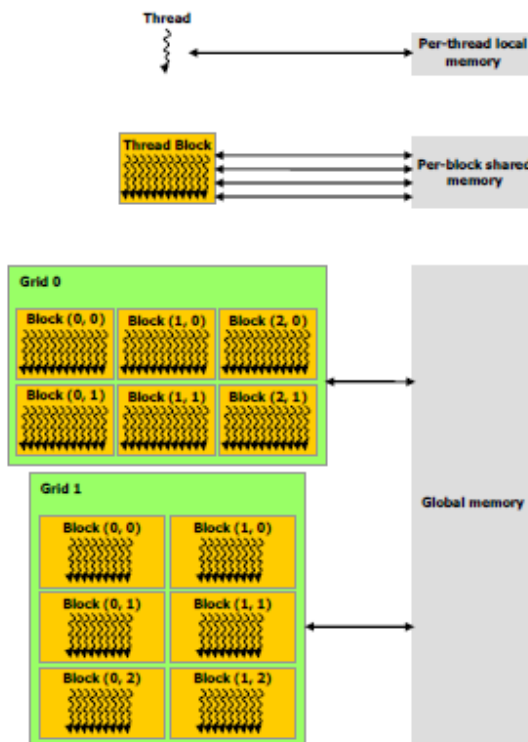


Figure 3.1: Memory Layout
[20]

are a large number of active threads. Access to global memory from the SM can be improved using coalescing. We use these rules to show power consumed by coalesced memory.

Registers

Registers are associated with each SM and give the fastest access. Registers can store scalars and built-in vector types. Arrays indexed by constant values known at compile time typically reside in registers. For our floating point benchmark we declare an array of constant values and make sure the size of the registers do not exceed 32 K since 32 K is the register space allocated per SM. Register spilling is very costly as it may result in data being placed in local memory rather than registers.

Shared Memory

Shared memory, which is software managed cache, is on chip memory which has high bandwidth and low latency. It can be used for thread cooperation as this memory is shared between all threads within a block. Shared memory is divided into successive equal sized banks, i.e. 32 x 32-bit for C2075, that can be accessed simultaneously. Shared memory can be as fast as the registers if bank conflicts are avoided. Multiple requests to the same bank result in serialization unless all threads read the same address.

3.2 Measuring Power Consumption using external device

We used Kill A Watt to validate our power model [11]. The Nvidia C2075 GPU is connected via PCI-Express to the main processor, but the power delivered through PCI-Express to the C2075 is not sufficient since PCI-Express can deliver only small amounts of power. We connected the C2075 to an external N110EF-00 power supply

so we can attach a power meter and validate our results. Figure 3.2 shows us the power management connections and gives a clear idea of how we validate our model.

3.3 Power Measurement using NVML

Nvidia Management Library (NVML) high level utility called `nvidia-smi` not only provides a way to measure power but also various other features like the ability to set ECC (Error Correction Code) to zero if it is not needed, or to monitor memory usage, among other things. For a full list of features available via `nvidia-smi` utility please refer to NVML manual [19]. NVML can be used to measure power when running the kernel but since `nvidia-smi` is a high level utility the rate of sampling power usage is very low and unless the kernel is running for a very long time we would not notice the change in power. NVML offers a lot of useful utilities for not only GPUs such as C2075 but also the Nvidia Tesla C2050 GPU where one would see power in states rather than in milliwatts. The `nvmlDeviceGetPowerUsage` function in the

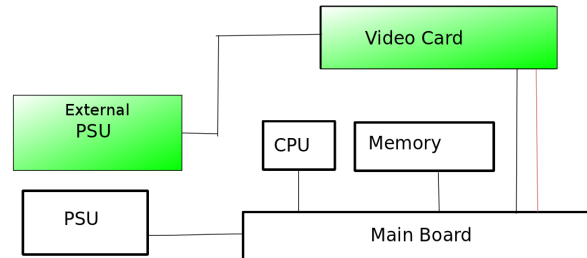


Figure 3.2: Validation using external power supply

NVML library retrieves the power usage reading for the device, in milliwatts. This is the power draw for the entire board, including GPU, memory, etc. The reading is accurate to within a range of +/- 5 watts error with milliwatt precision. It is only available if power management mode is supported.

We can also query for power management support using `nvmlDeviceGetPowerManagementMode`. For a C2050 GPU we would observe power states P0-P15 using the NVML function call `nvmlDeviceGetPowerState` where P0 is the power state when the GPU is running under full load and P15 is the power state when the GPU is completely idle for a long time. We can also retrieve temperature using the NVML high level utility or using the Nvidia Management Library's function call `nvmlDeviceGetTemperature`.

Kill-A-Watt is a power meter that we use to verify our results [11]. Kill-A-Watt allows us to check power usage when connected to an external power supply. Since we are measuring the power at the PSU (Power Supply Unit) level we have to take the efficiency of the the PSU into account. Our PSU which is a N110EF-00 has a loss of 11% of power.

3.4 PTX analysis

PTX stands for Parallel Thread eXecution, which is a pseudo-assembly code for GPUs [20]. PTX provides us with insight about how our code gets mapped into the CUDA architecture. It provides a machine independent ISA for C/C++.

We look at the PTX code to analyze the number of registers used, number of branches, global memory accesses, and floating point operations which is the key for our micro-benchmarks. This analysis is of particular interest to us because, if we wrote a micro-benchmark to test floating point operations, we would like to minimize data transfer and stress the floating point operations using registers. PTX allows us to analyze memory usage using `-ptxas-options=-v` and the output would be:

```

1 ptxas info: Compiling entry function '\_Z9logarithmPffii' for 'sm\_20'
2 ptxas info: Function properties for \_Z9logarithmPffii
3     4096 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
4 ptxas info: Used 23 registers, 52 bytes cmem[0], 16 bytes cmem[16]

```

Listing 3.1: PTX reporting memory usage

3.5 Frequency measurement of NVML Sensor

Frequency measurement is an important part of our analysis because if we measure power readings at a higher frequency than proposed, we observe the power reading repeating in a regular fashion. For example, if one calls NVML at 200 Hz frequency, one would observe 21 power measurement values repeating three times for our benchmark. To validate this, we have conducted a series of experiments where we measured Taylor series benchmark power at 200 Hz in Figure 3.4 and at 625 Hz in Figure 3.5 and each value repeats 3 and 6 times, respectively. So the maximum power measurement frequency is 62.5 Hz (Figure 3.3). These experiments show that over sampling the sensor will provide us with no additional information.

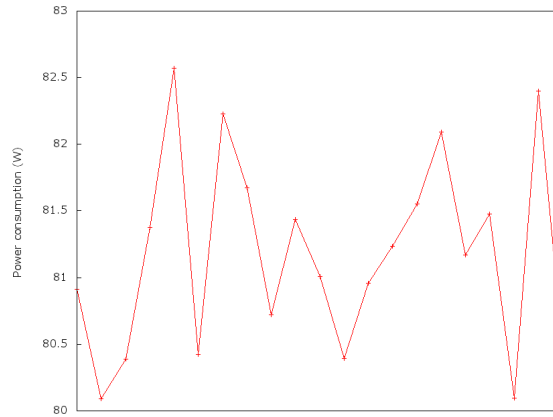


Figure 3.3: Power sampling frequency 62.5 Hz

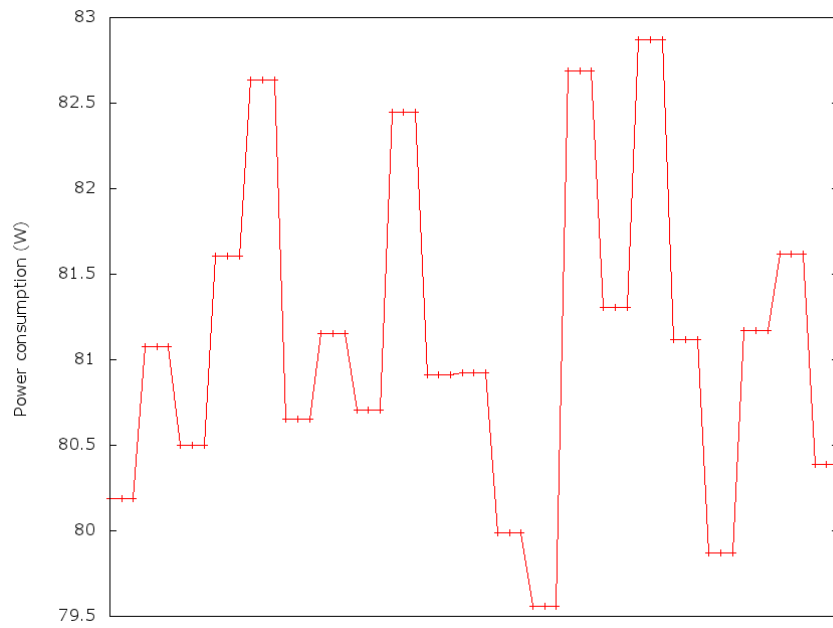


Figure 3.4: Power sampling frequency 200 Hz

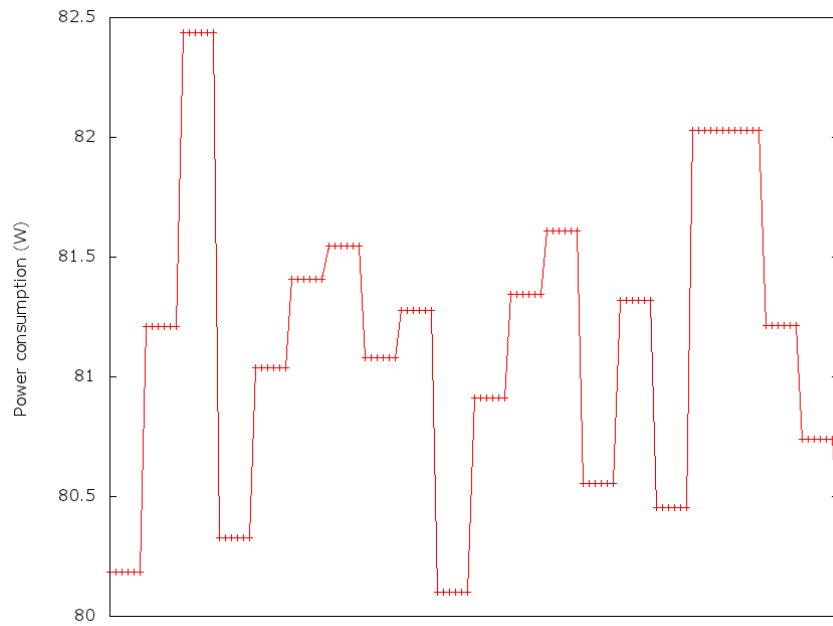


Figure 3.5: Power sampling frequency 625 Hz

3.6 Summary

This chapter explained the Nvidia C2075 architectural components such as registers, shared memory, global memory, and floating point units. The maximum frequency of the NVML power sensor is measured. Connections to Kill-A-Watt from an external power supply to validation our power measurements is also explained. PTX code is used to analyze the number of integer and floating point operations, shared and global memory accesses, and register accesses. The next chapter explains about our AMG model and each micro-benchmark in detail.

Chapter 4

Activity-based Model for GPUs (AMG)

A key challenge to effective runtime power management is to know the real-time power consumption. Although the power estimation for processors, memories, disks, and fans has been introduced, the power estimation technique of GPUs is relatively less addressed. However, runtime power estimation for individual micro-architectural structures on GPUs, such as caches and ALUs, would be useful for fine-grain management of package temperature and power requirements. We refer to estimating at this level as per-structure power estimation. Per-structure power estimation is useful for selectively enabling and disabling of micro-architectural resources.

To address the challenges of estimating per-structure power in hardware, we propose a new analytical model, called Activity-based Model for GPUs (AMG), to estimate activity factors and power for micro-architectural structures on GPUs. This model does not rely on real-time current monitoring or simulating hundreds of utilization statistics. We expect only a few input statistics are sufficient to estimate per-structure dynamic power of a GPU because the myriad per-structure events are related to a small set of global parameters, such as execution time and load rate. We use this key observation to drive the development of AMG. In spite of limited

input data, AMG’s linear-regression-based methodology can estimate activity for tens or hundreds of micro structures. We first analyze the correlation of a variety of performance metrics. Then we monitor only the least correlated metrics and use monitored metrics to extrapolate the concerned metrics. After we get all the concerned metrics about the structure events, we apply a per-event energy model to those structures to calculate the power of each structure. We believe that AMG makes a further step towards understanding and reducing the power of GPU systems through the usage of architecture level performance counters.

Power consumption can be divided into run-time power and idle power.

4.1 Idle Power

Idle power is the power consumed by the GPU when the GPU is turned on but no kernel is running. We measured the short idle power of a C2075 when the GPU is just turned on and does not do any work at 80W, which is also known as startup power for any kernel to be launched. When the GPU is in a long idle state, i.e., when the GPU is doing nothing for a long period of time, we measured the power consumed at 35W. The TDP (Thermal Design Power), as reported by NVIDIA, for the C2075 is 225W [18].

4.2 Run-time Power Consumption

We measure the run-time power of a kernel with the NVML library by running the kernel on a thread and NVML on another thread using Pthreads. We have chosen Pthreads because we would like to reduce overhead, and the only communication we would like to have with the main thread is a flag variable and variable to store power readings that are set to be volatile. The thread that is running NVML stops when the flag is reset, which is when the GPU kernel stops executing. For our run-time power consumption measurements of different micro-architectures, such as floating point,

shared memory, and global memory, we have designed micro-benchmarks such as memory copy with coalesced memory and with noncoalesced memory. For the floating point benchmark derived from a Taylor series, we run 1 million operations with measure power of 14 blocks and each block running 1024 threads. We used enough threads to cover the arithmetic latency of the SMs (Streaming Multiprocessors), which means that on a Compute Capability 2.0 GPU, we need about 10 warps (groups of 32 threads) per SM. So that means, for example, on a Fermi C2075 GPU with 14 SMs, we would need at least 4480 threads, divided into at least 14 blocks. The way to manage the number of active SMs is changing the number of active blocks [8]. We use 14 blocks to run each benchmark since the C2075 has 14 SMs that run simultaneously. If more than 14 blocks are assigned, the next blocks waits for one of the blocks to finish working and then starts working. Energy consumption varies with the number of SMs because of the low activity factors, as idle SMs do not consume as much energy as active SMs.

A Unit is defined as an architectural component such as a floating point unit (FPU), shared memory, or global memory. We construct our model as

$$\text{Total power consumption} = \text{Idle Power} + \text{Runtime Power} \quad (4.1)$$

$$\text{Runtime Power} = \sum_{i=1}^e (N_{SM} \times P_{u,i} \times U_{u,i}) + B_{u,i} \times U_{u,i} \quad (4.2)$$

N_{SM} Number of units

$P_{u,i}$ Power consumption of active unit

e number of architectural component types

$B_{u,i}$ Base power of unit

$U_{u,i}$ Utilization

Table 4.1: Power consumption of various units

$P_{u,i}$ Values for different Units	Value
Floating Point Unit	2.2
Shared Memory	1
Global Memory	3.0

Table 4.2: Base power for various units

$B_{u,i}$ Values for different Units	Value
Floating Point Unit	6
Shared Memory	3.85
Global Memory	10

4.2.1 Floating Point Operations

The intent of this benchmark is to create kernels that use the floating point units, but with little or no other parts of SPs used. The benchmark scales from 1 to 14 SMs, with each of their floating point ALUs heavily used. The power contribution of floating point units is to fit a line parameterized by the number of SMs that are busy. We designed our floating point benchmark based on a Taylor series in such a way that each thread computes the Taylor series of an element. We iterate each calculation 16000 times to make the kernel run long enough so that we can get stable power readings. During this process of measuring floating point instructions, we only use registers for storage. The memory usage reported by `cudaMemGetInfo` is found to be 80 MB mainly because that is the memory that is set apart by the compiler for the GPU usage. We expect the memory usage to be much lower than that since most of the variables are reused in an iterative way by each thread. We use `cublasSasum` to add the thread's results together so that the compiler will not be able to optimize.

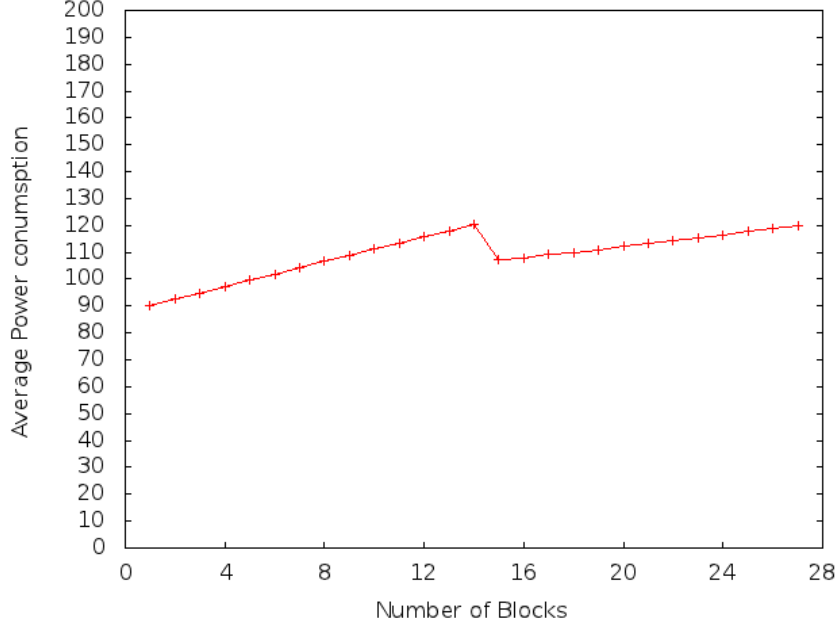


Figure 4.1: Average Power consumption of Floating point operations

4.2.2 Shared Memory

We use `__shared__` to allocate shared memory as explained in the CUDA manual [17]. We wrote micro-benchmarks for shared memory with and without bank conflicts. We use the `cudaFuncSetCacheConfig` function to increase the shared memory size from 16K to 48K. This allows us to estimate average power consumed by the kernel when we use the shared memory completely. The default is 48 KB for shared cache and 16 KB for L1. Shared memory is divided into 32 banks and each bank holds a 32-bit value (integer or float), so we write micro-benchmarks to exhibit the energy difference between shared memory with bank conflicts and without bank conflicts. Shared memory without bank conflicts is designed to have regular access patterns. A set of micro-benchmarks is designed to analyze shared memory.

The advantages of using shared memory over global memory are many:

1. Cooperation between threads.
2. Much faster than global memory.

3. If one thread loads data it can be used all the threads.
4. The amount of shared memory is configurable via the `cudaFuncSetCacheConfig` function.

Table 4.3: Shared Memory

Case1	No bank conflicts
Case2	two bank conflicts
Case3	four bank conflicts
Case4	eight bank conflicts
Case5	sixteen bank conflicts

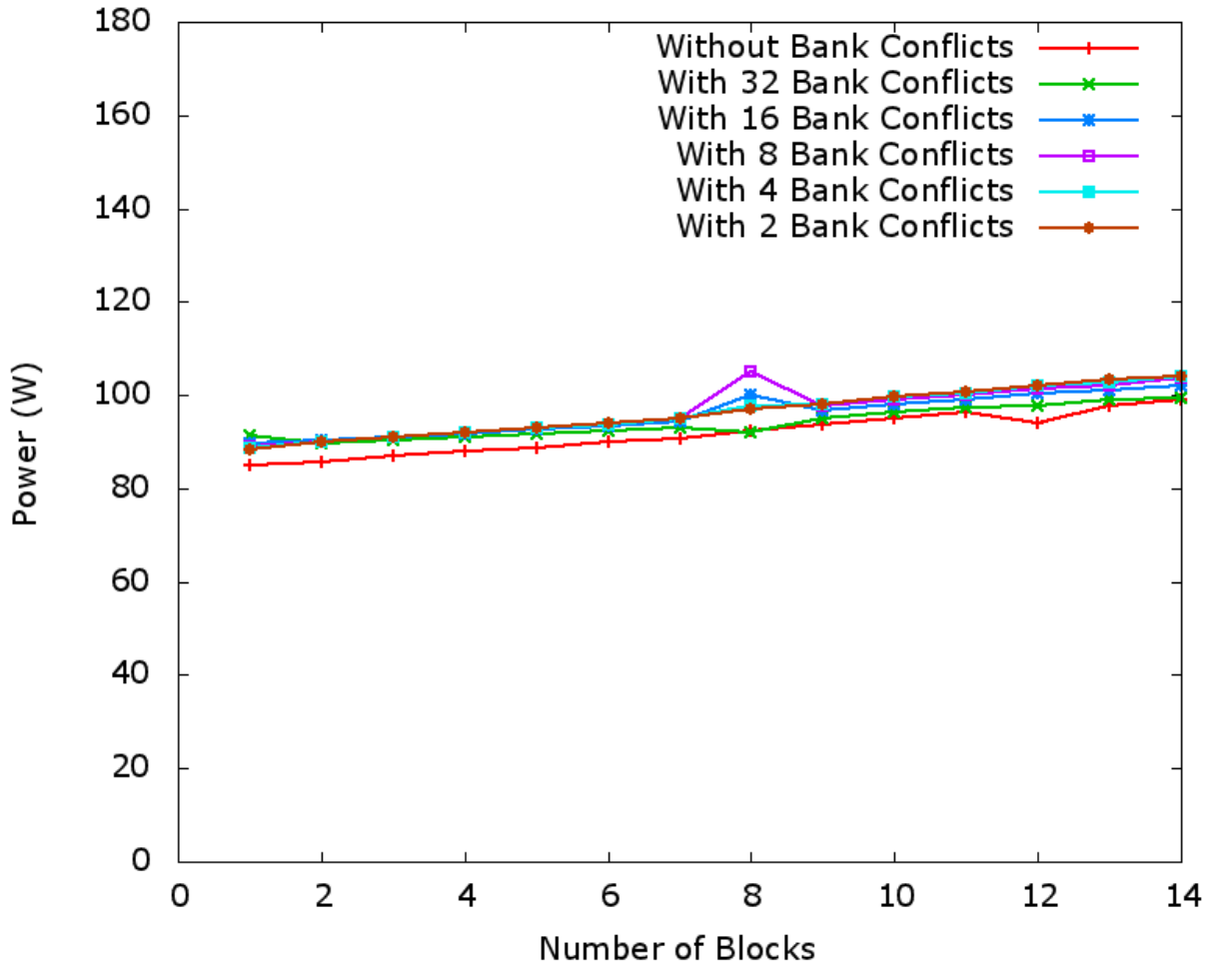


Figure 4.2: Power consumed by shared memory without bank conflicts

4.2.3 Global Memory

Global memory space is the largest memory available on a GPU. For example, on NVIDIA C2075 there are 6 GB of GDDR5, which is global memory implemented with Dynamic Random Access Memory (DRAM). The latency of global memory is on the order of hundreds of cycles, and the bandwidth is also very limited. By looking at the PTX code we can actually identify the global memory accesses.

Coalesced Memory

Since access to global memory is via 32, 64, or 128 byte accesses, we design our benchmark in such a way each thread can access it in a regular pattern of 128 bytes. Coalesced memory accesses are very important for instruction throughput. The local and global variables use global memory. If we declare an array of large size without using shared memory, it resides in global memory and accesses of strides of 128 are actually better than an irregular pattern. Figure 4.3 shows power consumption of coalesced memory.

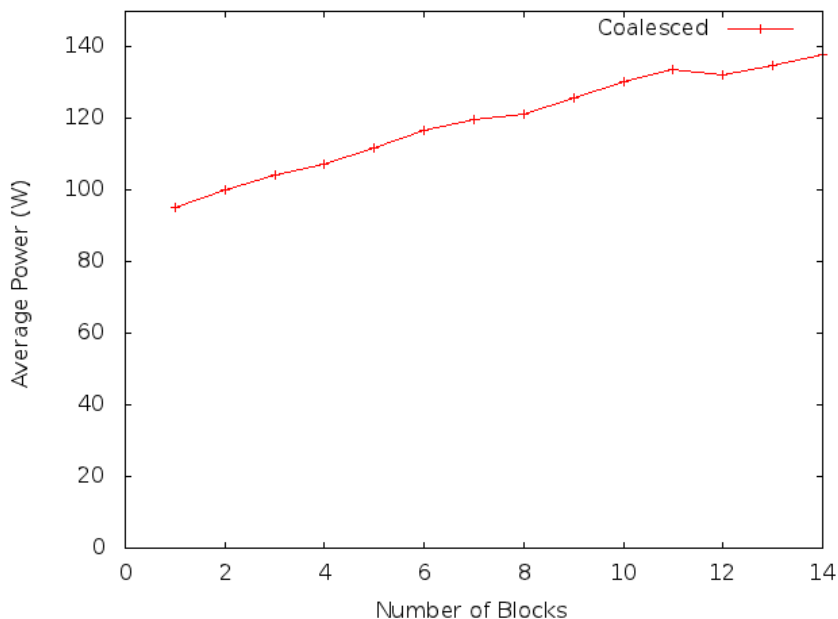


Figure 4.3: Power consumed by global memory with coalesced memory accesses

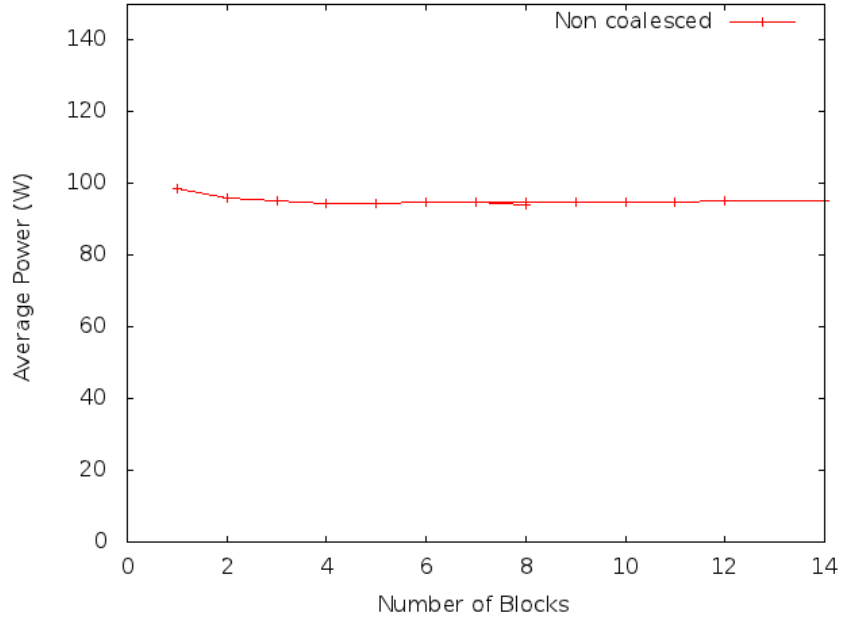


Figure 4.4: Power consumed by global memory with noncoalesced memory access

Noncoalesced Memory

If memory accesses to global memory which are not regular patterns to global memory are called noncoalesced accesses. Our results show that the noncoalesced memory consumes at least twice the energy consumed for 16k writes and 16k reads compared to coalesced memory accesses. Figure 4.4 shows power consumption of noncoalesced memory accesses. The noncoalesced memory accesses are at least 4 times slower than coalesced memory accesses which results in huge energy consumption.

4.3 Validation

The first step in validating our model is to plot the predicted values. To validate our model we use two versions of matrix matrix multiply from the CUDA SDK, i.e. one which uses shared memory and one which does not use shared memory. For matrix multiply which does not use shared memory the number of global memory reads is N^2 and writes for the kernel is N since we consider two matrices of $N * N$. The average

power consumed by this kernel of size 14K is 130 W. We choose matrix of size 14K because C2075 GPU has 14 SM and each SM has 1024 threads so for a matrix of size 14K all the threads in each SM are working. The average power consumed by the kernel that uses shared memory is 120 W since the number of reads and writes to global memory decrease by a large factor.

```

1 for (e = 0; e < A.width; ++e)
2 {
3     Cvalue += A.elements[row * A.width + e]
4     * B.elements[e * B.width + col];
5 }
6 C.elements[row * C.width + col] = Cvalue;

```

Listing 4.1: Naive Matrix Matrix multiply

The naive implementation shown above does not use shared memory. As a result is a performance penalty and power consumption also increases.

```

1 __shared__ float Mds[TILE_WIDTH][TILE_WIDTH];
2 __shared__ float Nds[TILE_WIDTH][TILE_WIDTH];
3 int bx=blockIdx.x, by=blockIdx.y, tx=threadIdx.x, ty= threadIdx.y;
4 int Row = by * TILE_WIDTH + ty;
5 int Col = bx * TILE_WIDTH + tx;
6 float Pvalue = 0;
7 for (int k = 0; k < Width/TILE_WIDTH; ++k)
8 {
9     Mds[ty][tx]=Md[Row*Width + (k*TILE_WIDTH + tx)];
10    Nds[ty][tx]=Nd[(k*TILE_WIDTH + ty) * Width + Col];
11    __syncthreads();
12    for(int k = 0; k < TILE_WIDTH; ++k )
13        Pvalue+=Mds[ty][k]*Nds[k][tx];
14    __syncthreads();
15 }
16 Pd[Row* Width + Col] = Pvalue;

```

Listing 4.2: Matrix Matrix multiply with shared memory

This implementation uses shared memory, the surprising fact is that the kernel with shared memory is faster as well as consumes less power so we can conclude that that global memory consumes a lot of power.

Predicting power using AMG is an important step since previous generation GPUs such as Nvidia C2050 do not fully support NVML. To predict power we need the execution time of each unit

We split the matrix matrix multiply so that we can tease out computation and memory. We write a CUDA kernel which performs the same number of floating point operations as the matrix multiply and measure the time taken and average power consumed by the kernel and we follow the same rule for memory operations.

Table 4.4: Matrix Matrix Multiply example

Kernel	Run time Power(W)	Processing Time (sec)
Matrix Matrix Multiply	50	111
Floating point	37	45
Memory	45	47

Floating Point Unit

Time taken by floating point unit = 45 seconds

Number of SMs used $M = 14$

Power consumed/SM by active Unit = 2.2

Global memory

Time taken by floating point unit = 47 seconds

Number of SMs used $M = 14$

Power consumed/SM by active Unit = 3

Table 4.5: Matrix Matrix Multiply evaluated using AMG

Parameter	FPU	Global
M	14	14
$P_{u,i}$	2.2	3
$B_{u,i}$	6	10
$U_{u,i}$	0.405	0.423

$$\text{Run time power} = (((14 \times 2.2 \times 0.405) + 6 \times 0.405) + \quad (4.3)$$

$$((14 \times 3 \times 0.423) + 10 \times 0.423)) \quad (4.4)$$

$$\text{Runtime Power} = 36.9 \text{ W} \quad \text{Idle Power} = 80 \text{ W}$$

$$\text{Total Power for Matrix Multiply} = 116.9 \text{ W}$$

$$\%Error = \frac{|ActualValue - PredictedValue|}{ActualValue} * 100 \quad (4.5)$$

$$\%Error = \frac{13.91}{130} * 100 \quad (4.6)$$

$$\%Error = 10.7\%$$

Using matrix matrix multiply we have shown that our model predicts power consumption if we know the execution rates. The execution time of each individual components do not add to the total execution time, so that is the primary reason for the error. If we could obtain more precise execution rates of each individual units we might be able to obtain higher accuracy.

4.4 Power and Temperature relationship

The power consumption is a very critical parameter of contemporary integrated circuits. It is obvious that a circuit should consume as little power as possible and ought to work with maximum speed and efficiency. However, power parameters are dependent on temperature, which can change with power dissipated in the circuit. In a GPU there are several SMs working simultaneously which further increases power. With rising temperature, power consumption becomes higher, too. The principal reason for that behavior is the increased amount of leakage current with higher temperatures, and the negative temperature coefficient of the transistors [5].

Figures 4.5 and 4.6 show the increase in power consumption when Taylor series and memory copy benchmarks are executed at various startup temperatures. To figure out the temperature influence on power, a kernel is executed applies a workload to the GPU in order to raise the temperature of the GPU to a certain value. The power consumption of FLOPs benchmark increases by 4 W when startup temperature increase from 50 C to 80 C. Because the NVML power measurements are only accurate to within 5 W, we don't consider temperature hereafter. The temperature influence on power consumption is only 3-4% for the current generation GPU. Thermal slowdown occurs at 90 C and thermal shutdown at 100 C.

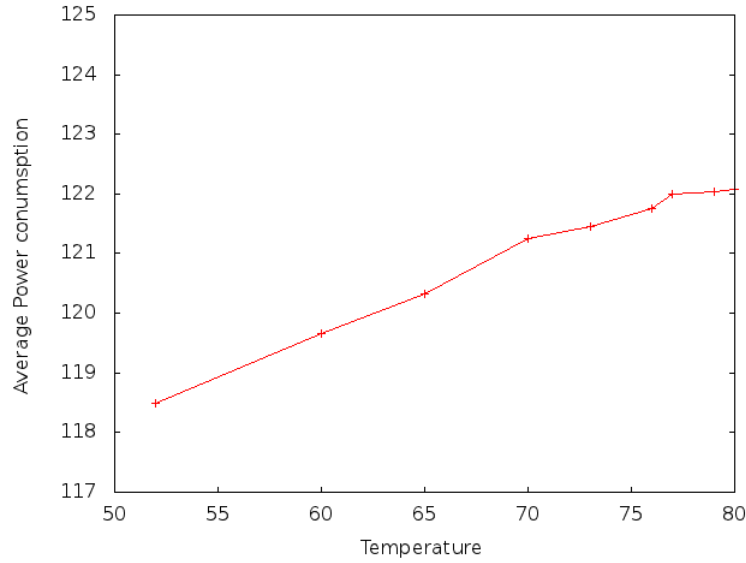


Figure 4.5: Increase in power consumption with temperature for FLOPs benchmark

4.5 Summary

In this chapter we present our AMG model which can be used to predict power consumption. We explain micro-benchmark for each architectural unit such as floating point unit, shared memory, global memory. We have validated our model using matrix-matrix multiply and the error is only 10.07%. We have also analyzed power and temperature relationship and found that temperature influence on power is only

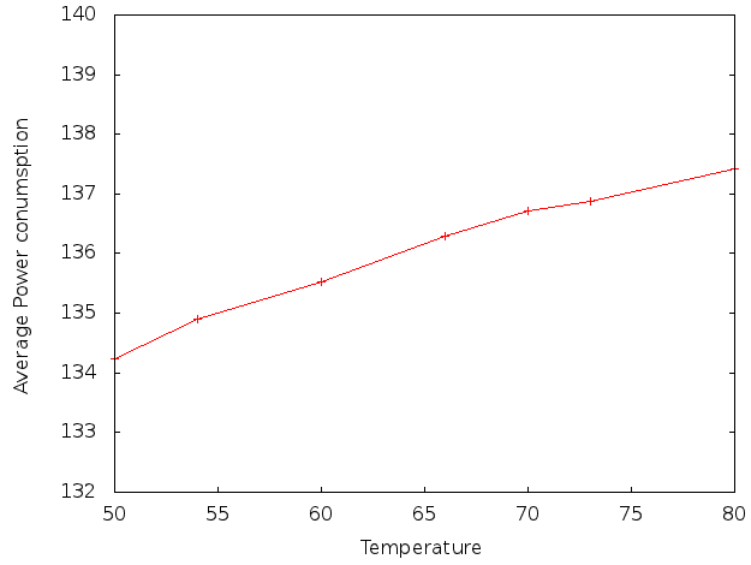


Figure 4.6: Increase in power consumption with Temperature for Global Memory benchmark

3-4%. The next chapter discusses MAGMA BLAS2 kernels, BLAS3 kernels, LU factorization, and Hessenberg and we estimate power for these kernels based on our model.

Chapter 5

Power Analysis of MAGMA Kernels

In the 1970's, vector machines were introduced; to take advantage of the vector processors, matrix vector operations were introduced. Next came the RISC processors, and to take advantage of memory heirarcy, cache optimized matrix matrix operations were introduced. Improvement in performance in current GPUs were as a result of adding different levels of cache and improvements in memory bandwidth compared to the previous generation of GPU [6]. Block operations can be optimized for each architecture to account for its memory hierarchy, and therefore provide a transportable way to achieve high efficiency on diverse modern machines.

The energy consumption of linear algebra kernels is of vital importance, as these kernels are widely used, so we measured some of the MAGMA kernels that are both memory and computationally intensive. We analyzed the real time power consumption of two fundamental linear algebra algorithms; the LU factorization (MAGMA_Dgetrf) for solving dense linear systems of equations, and the upper Hessenberg reduction (magma_Dgehrd) for solving the general eigenvalue problem. Results show that the MAGMA implementations of these algorithms achieve astounding energy efficiency. We have demonstrated that, depending on hardware

and software configuration, MAGMA uses as little as 1/50th the energy of traditional multicore CPUs. Shown below are the performance charts for the two algorithms along with the real-time power consumption traces. The MAGMA LU factorization is a compute bound algorithm (expressed in terms of GEMMs), and the MAGMA Hessenberg reduction is memory bound (expressed in terms of GEMVs and GEMMS, respectively 20% and 80% of the flops). The real-time power consumption for these kernels (GEMM and GEMV) is also measured, and power consumed by the MAGMA DGEMM and SGEMM algorithm are found to be 180 W and 180 W and for DGEMV power consumption is 135 W and SGEMV is 135 W.

The advantages of using the Fermi architecture vs. older generation GPUs are numerous. For example, the LU factorization takes full advantage of the increased shared memory, number of registers, number of CUDA cores in a multiprocessor, and some other changes that were also introduced. For example, register blocking was introduced especially for Fermi architecture to take advantage of memory hierarchy. Figure 5.1, according to [13], shows the computation is divided into $N_T = N_{TX} * N_{TY}$ so that each sub-matrix can load in shared memory. The MAGMA GEMM takes advantage of coalesced memory to write final results from registers to global memory.

5.1 BLAS 2 and BLAS 3 Kernels

The MAGMA kernels utilize CPU and GPU for the computations. The measuring frequency is 125 KHz which is twice the maximum frequency. The impact on CPU computations while spawning Pthreads to measure power using NVML is small as frequency is not very high. Figure 5.2 shows the performance of SGEMM and DGEMM. The DGEMM performance for a matrix of size 9K is 296.11 GFLOPS, which is 58% of the theoretical peak, and the performance of SGEMM for a size 9K matrix is 632 GFLOPS.

The performance of SGEMV and DGEMV is considerably less compared to SGEMM and DGEMM because the BLAS 2 kernels do not utilize the GPU as

efficiently as BLAS 3 kernels. Figure 5.3 shows the performance numbers for a 9K matrix is 60 GFLOPS for SGEMV and for DGEMV the performance is 30.47 GPLOPS.

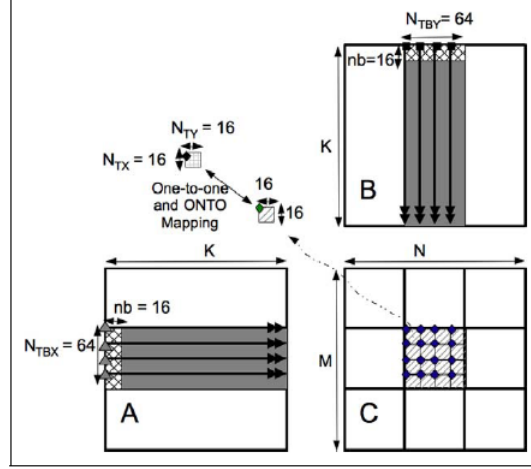


Figure 5.1: GPU GEMMs on Fermi architecture
[13]

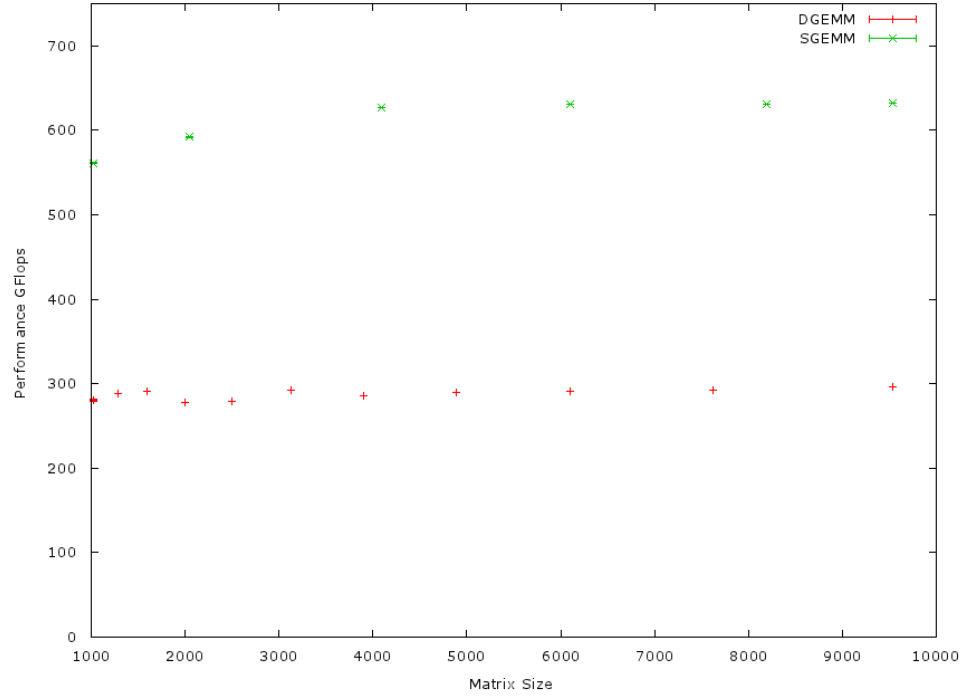


Figure 5.2: Performance of single and double precision MAGMA BLAS 3 Kernels

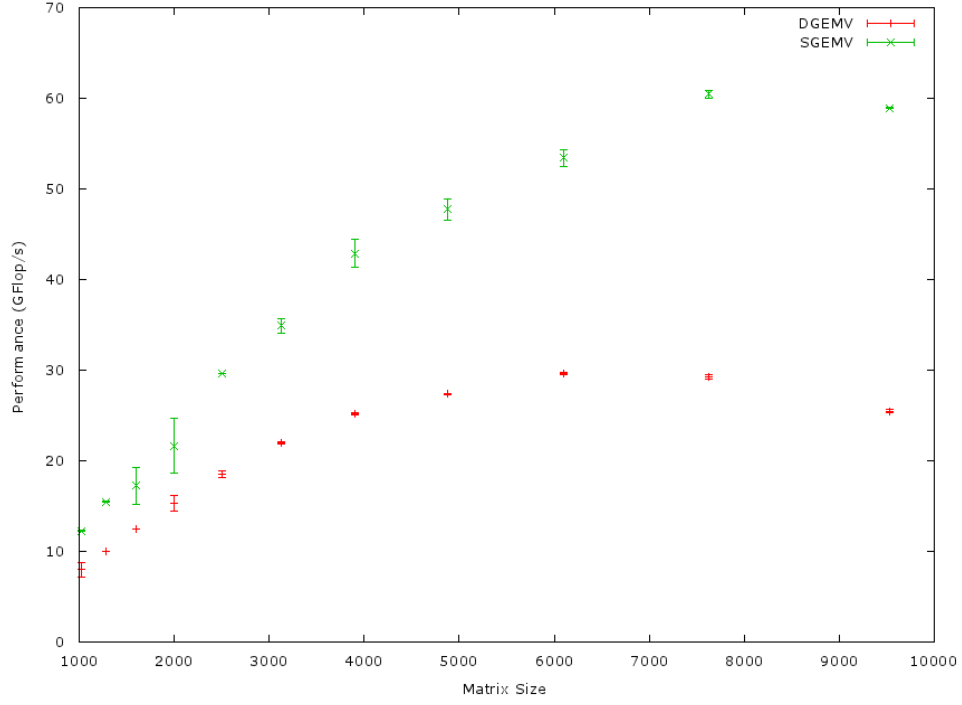


Figure 5.3: Performance of single and double precision MAGMA BLAS 2 Kernels

Figure 5.4 shows the power consumed by single and double precision MAGMA GEMMs. The same amount of power is consumed by both the BLAS 3 kernels because an SP can issue two single precision instructions or one double precision every two clocks, but energy varies since single precision MAGMA GEMMs and GEMVs are twice as fast as double precision.

Figure 5.4 shows us the performance and power consumption of MAGMA DGEMM and MAGMA SGEMM with NVML. The GFLOPS per Watt for a matrix of size 10112 is 1.49; this proves that the GPU not only has better performance, but also saves on energy.

5.1.1 Measuring time taken in MAGMA Kernels

The function `get_current_time()` calls `gettimeofday()`, so the resolution is a microsecond. Before calling the `gettimeofday()` there is a call to `cudaThreadSynchronize()` to make sure previous GPU tasks have completed. Thus one can measure the time of

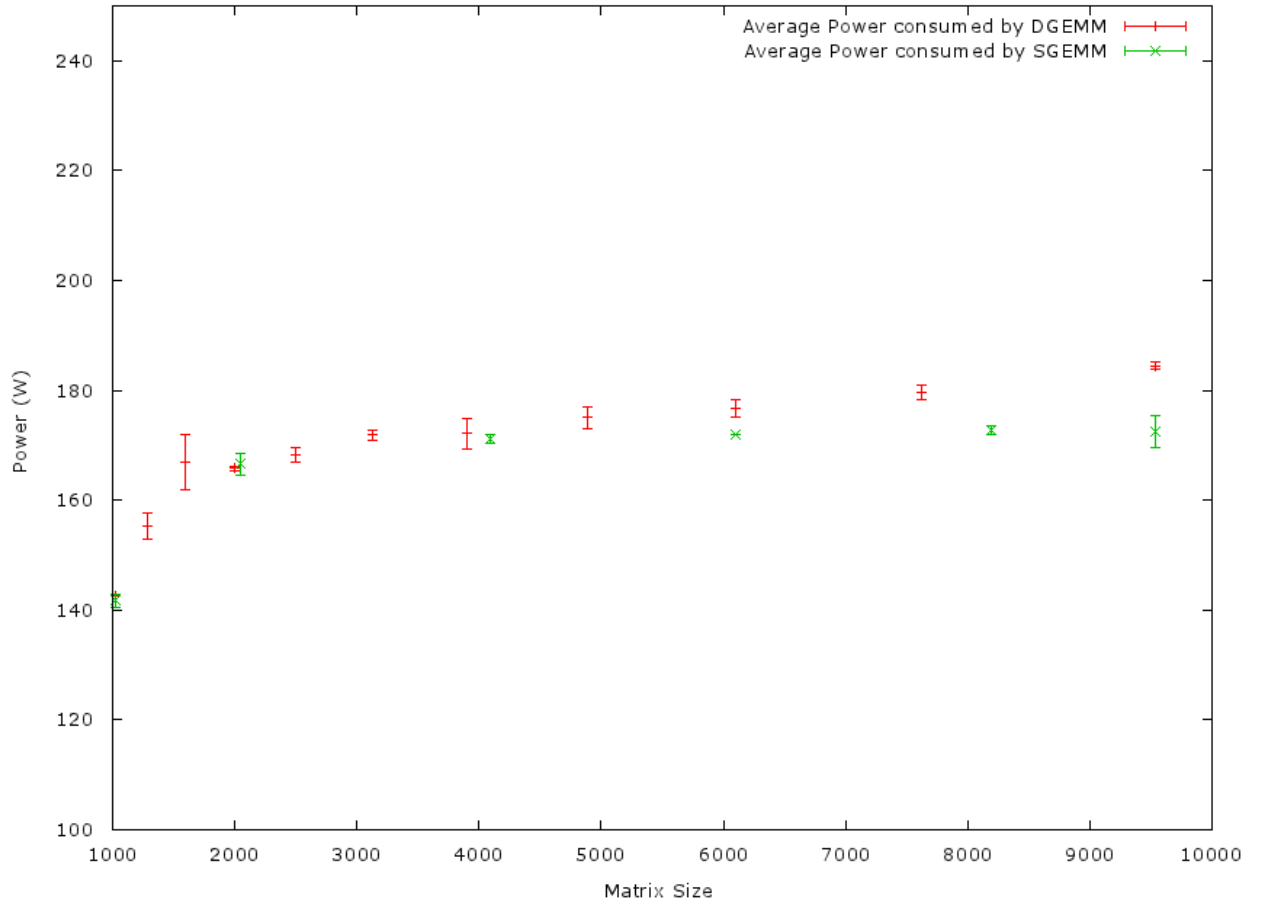


Figure 5.4: Average Power consumption of single and double precision MAGMA BLAS3 Kernels

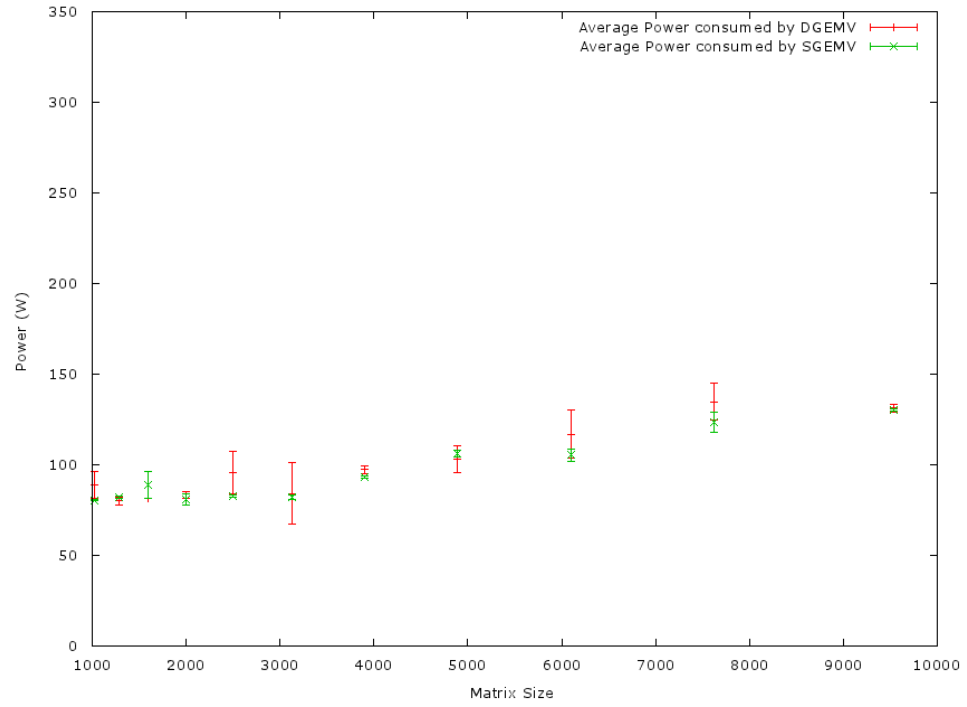


Figure 5.5: Average Power consumption of single and double precision MAGMA BLAS2 Kernels

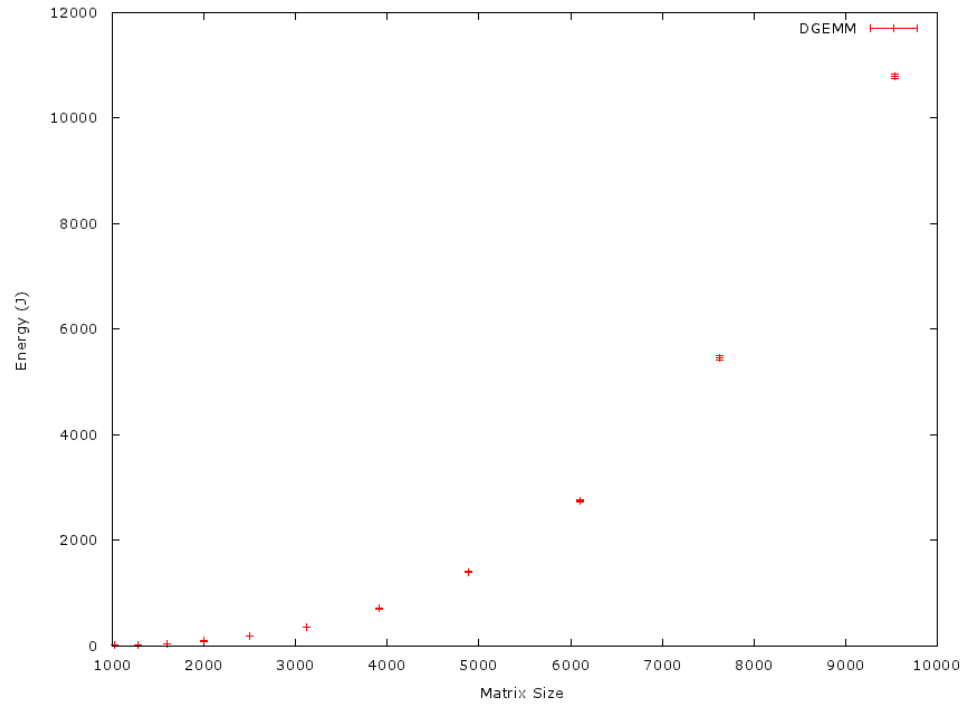


Figure 5.6: Energy consumption of double precision MAGMA BLAS3 Kernels

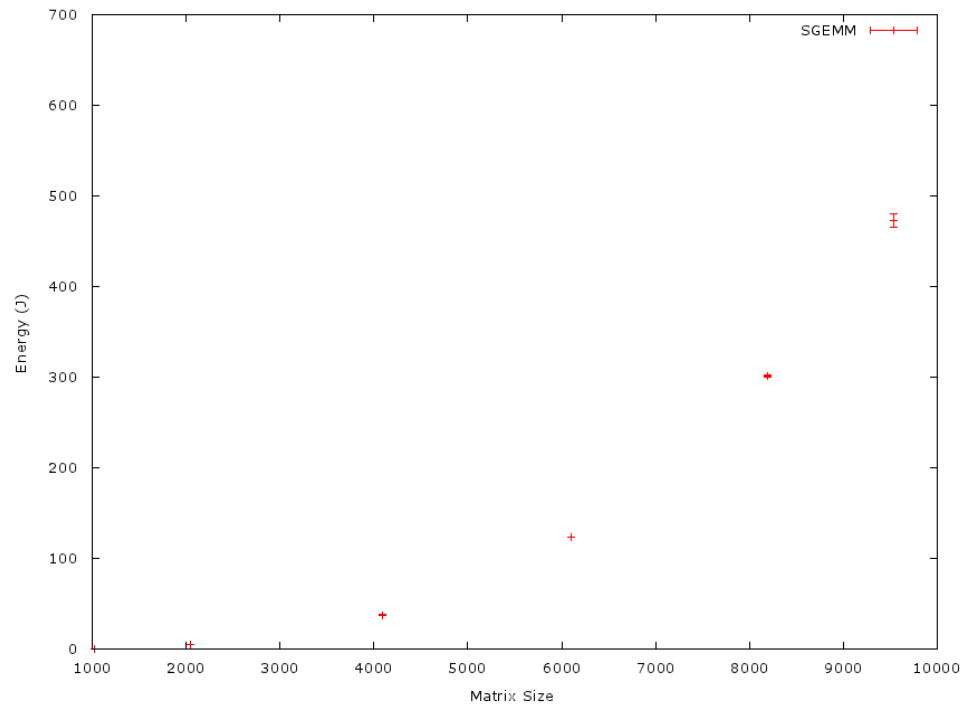


Figure 5.7: Energy consumption of single precision MAGMA BLAS3 Kernels

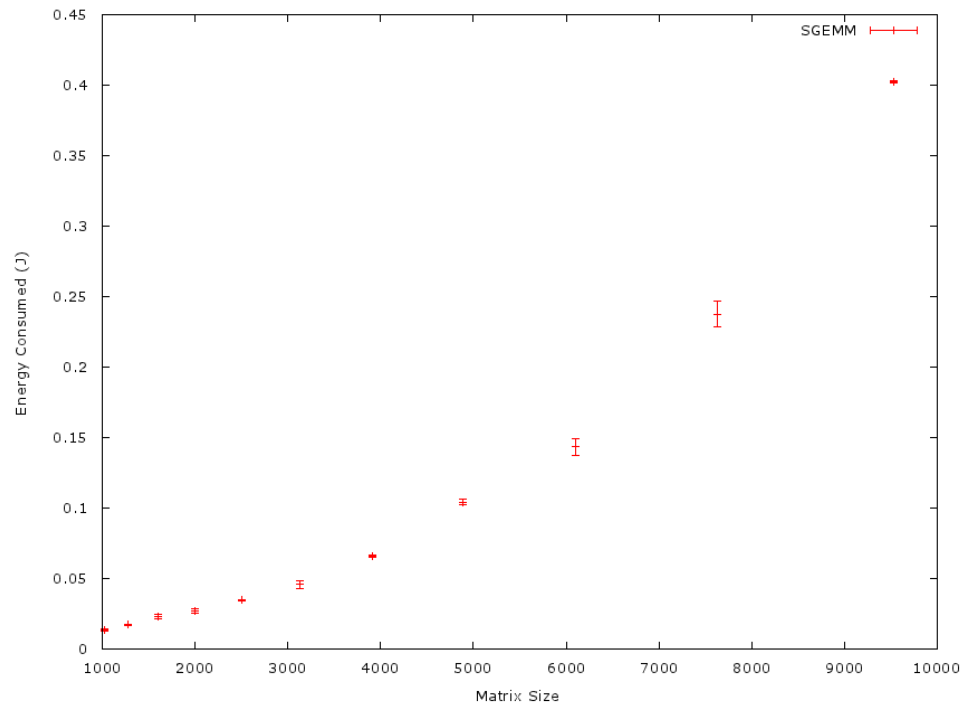


Figure 5.8: Energy single precision MAGMA BLAS2 Kernels

a particular GPU kernel by surrounding it by calls to `get_current_time()`. If between two `get_current_time()` calls there are functions transferring data, the time measure will include the time for the transfer. We measure the time for DGEMM on the GPU, i.e., we assume the data and the result will be in the GPU memory.

Table 5.1: Power consumption of MAGMA BLAS2 and BLAS3 kernels

MAGMA Kernel	Average power consumed (W) of matrix size 8K
DGEMM	180
SGEMM	180
DGEMV	135
SGEMV	135

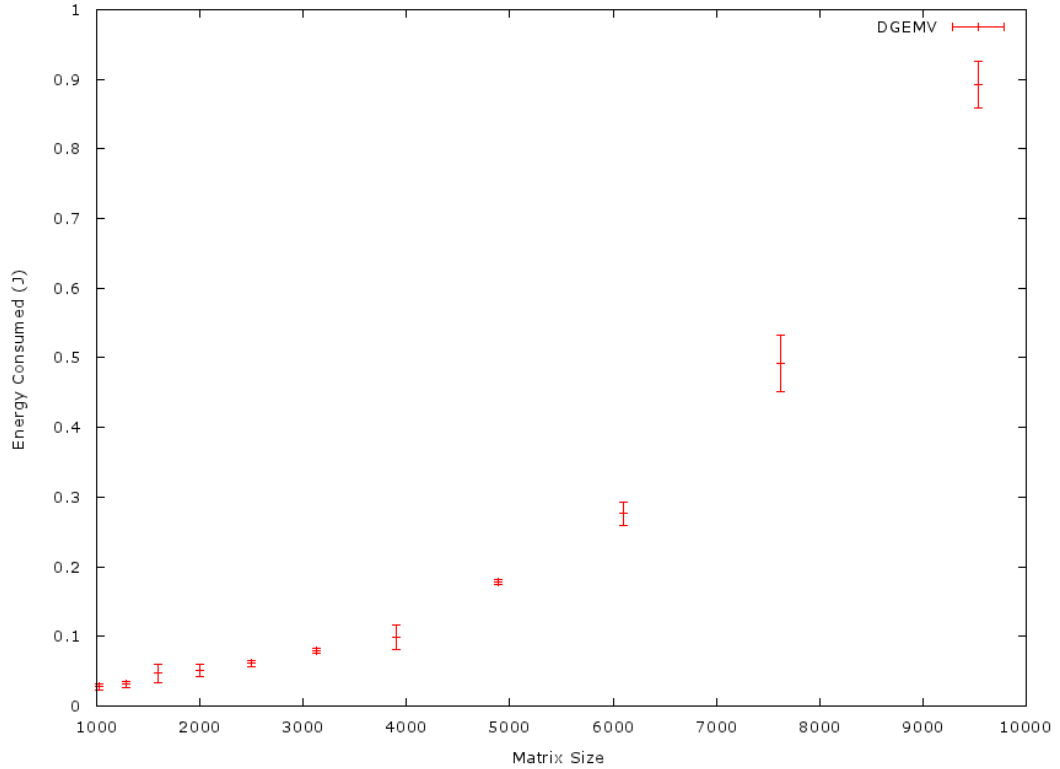


Figure 5.9: Energy consumed by double precision MAGMA BLAS2 Kernels

5.2 MAGMA LU Factorization and Hessenberg

Let A be a square matrix, then A can be decomposed into LU , where L is the lower triangular and U is the upper triangular matrix.

$$A = LU$$

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{21} \\ U_{21} & U_{22} \end{pmatrix} \quad (5.1)$$

And LU factorization with partial pivoting is of the form

$$PA = LU$$

$$P \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{21} \\ U_{21} & U_{22} \end{pmatrix} \quad (5.2)$$

where P is a permutation matrix, L is a lower triangular matrix and U is an upper triangular matrix

Table 5.2: GPU vs CPU

Device Name	Fermi C2075 GPU	AMD Istanbul (8 socket * 6 core (48 core) @ 2.8 GHz)
DP Peak	515 + 40 GFLOPS	538 GFLOPS
System Cost	\$3000	\$10000
Power	220 W	1022 W

The MAGMA LU factorization uses a hybridization methodology to split the computation between the CPU host and GPU. The splitting aims to match the LU's algorithmic requirements to the architectural strengths of the GPU and the CPU. In

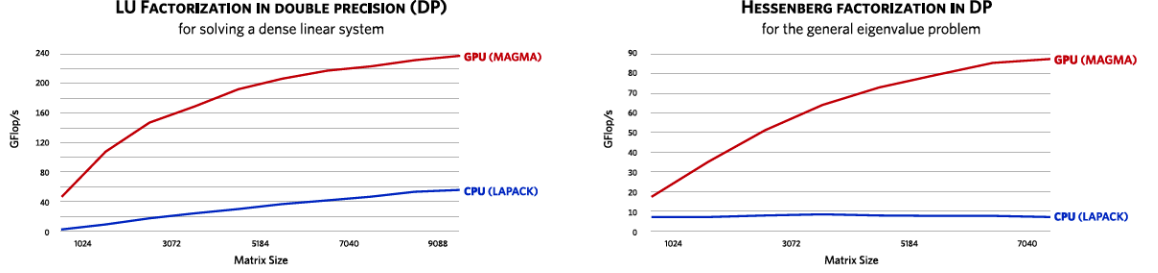


Figure 5.10: Dgetrf and Dgehrd comparison between GPU and CPU

the case of LU, this translates into having all matrix-matrix (GEMM) multiplication done on the GPU, and the panel factorizations on CPU. The design of the algorithm allows for big enough matrices to totally overlap the CPU work with the large matrix-matrix multiplications on the GPU. As a result, the performance of the MAGMA LU algorithm runs at the speed of performing GEMMs on the GPU. Our experiments have shown that the use of MAGMA GEMM operations on GPU completely utilize it thus maximize power consumption, which, combined with the description above, explains why the hybrid LU factorization also maximizes the GPU power consumption which reduces time taken so the overall energy consumption is minimized. Figure 5.10 shows us the theoretical performance of MAGMA DGETRF, which is an astounding 515 GFlops/s. Figure 5.12 shows power consumption analysis of DGETRF which is actually close to DGEMM power consumption since 100% of the flops are from DGEMM.

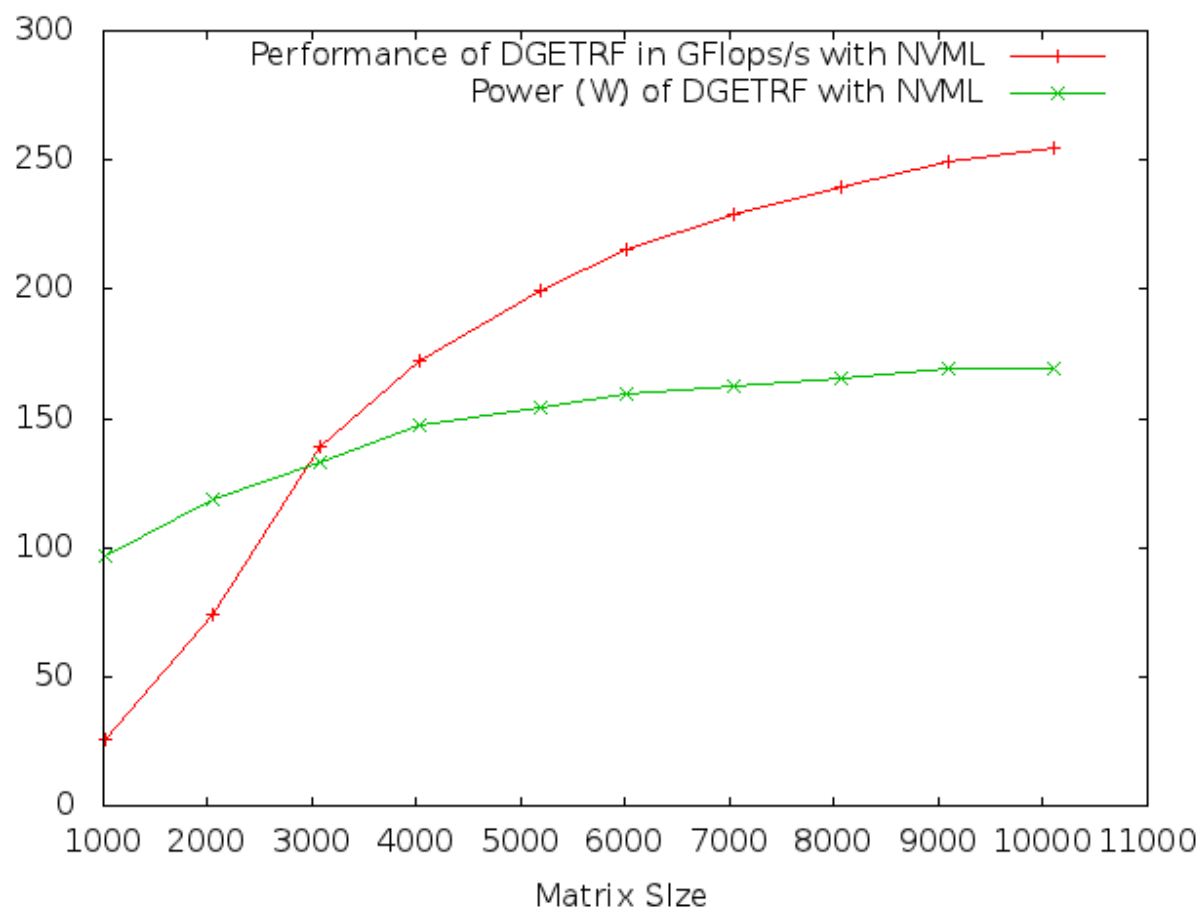


Figure 5.11: Dgetrf

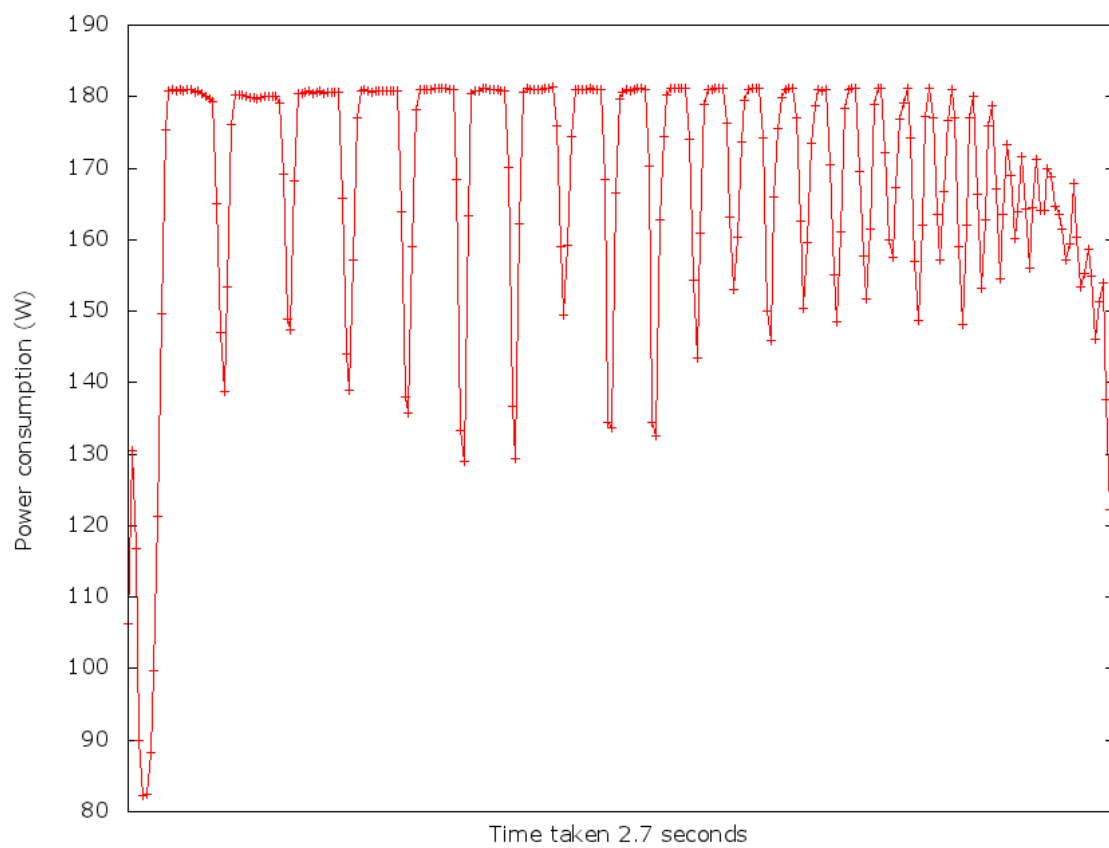


Figure 5.12: MAGMA Dgetrf power consumption for a 10k matrix

5.2.1 Reducing Noise in Power measurements

For Figure 5.12, a signal averaging technique is used over 24 sets of data. The skew is very little as we start the thread, which is running NVML just before the MAGMA kernel call using a volatile variable. Signal averaging is a signal processing technique applied in the time domain, intended to increase the strength of a signal relative to noise that is obscuring it [33]. By averaging a set of replicate measurements, the signal-to-noise ratio, S/N, will be increased, ideally in proportion to the square root of the number of measurements.

We chose our bandwidth according to Nyquist theorem which states that a bandlimited analog signal can be perfectly reconstructed from an infinite sequence of samples if the sampling rate exceeds $2B$ samples per second, where B is the highest frequency of the original signal [32]. The measuring frequency is 125 Hz because the maximum frequency is 62.5 Hz.

The other technique that was used to reduce noise was to measure the number of DGEMM and DGEMV calls from the algorithm to measurements but NVML has a very low frequency of 62.5 Hz compared to the clock rate of NVIDIA C2075 GPU which is 1.15 GHz so cannot observe all the peaks.

According to the MAGMA DGETRF algorithm

$$\forall panel = \begin{cases} 1 \text{ small DGEMM} \\ 1 \text{ large DGEMM} \end{cases}$$

Table 5.3: DGEMM calls in MAGMA LU

Source	Number of DGEMMs
Algorithm	78
NVML	75

Table 5.3 shows the number of DGEMM calls MAGMA DGETRF makes. From the MAGMA algorithm, we can infer that there are 39 small DGEMMs and 39 large

DGEMMs. For measurement we would consider a spike greater than 175 W as a DGEMM call, since power consumed by a DGEMM kernel is 180 W.

5.2.2 MAGMA Hessenberg

The Hessenberg reduction algorithm is of the form $Q^T A Q = H$. In contrast to the LU factorization, the other algorithm that we have studied, namely the Hessenberg Reduction, cannot be entirely expressed in terms of GEMMs. Proper task splitting on hybrid architectures using the Hessenberg principle, have been known to give enormous performance benefits [30]. According to [30] the operation count to reduce an N by N matrix is $(10/3)n^3$ and this makes Hessenberg reduction a suitable candidate for acceleration. But 20% of the total flops of the algorithm, which take 70% of time, are in level 2 BLAS. This makes the algorithm memory bound and we observe it for an energy consumption that is close to the power consumption for matrix-vector operations. Figure 5.13 shows us the power consumption for MAGMA DGEHRD which is also signal averaged over 24 data samples.

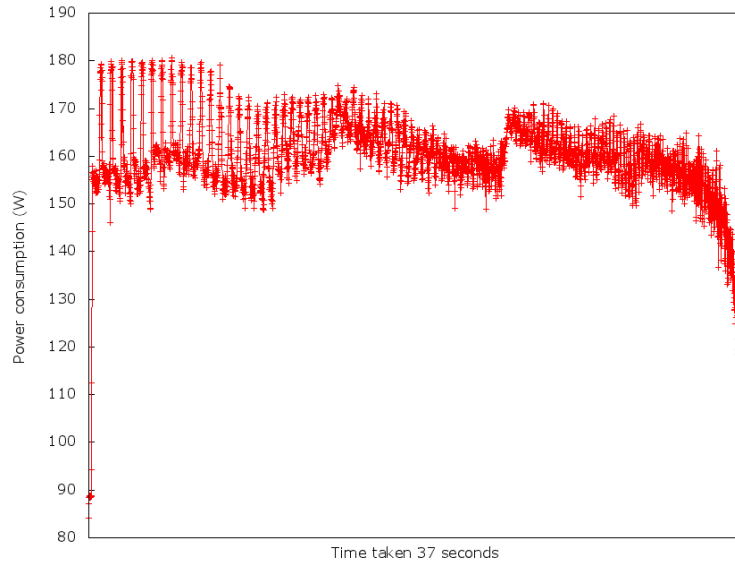


Figure 5.13: MAGMA Dgehrd power consumption for a 10k matrix

According to the MAGMA DGETRF algorithm

$$\forall panel = \begin{cases} 64 \text{ small DGEMV} \\ 6 \text{ large DGEMM} \end{cases}$$

Table 5.4: DGEMM and DGEMV calls in MAGMA Hessenberg for a matrix of size 10 K

Source	Number of DGEMM calls	Number of DGEMV calls
Algorithm	936	9984
NVML	367	1457

Table 5.4 shows the number of DGEMM and DGEMV calls in MAGMA Hessenberg. There is a large difference between the number of DGEMM and DGEMV calls measured and observed because the frequency of the NVML sensor is on the order of 62.5 Hz whereas the clock frequency of the GPU is of the order of 1.15 GHz.

Table 5.5: Average power consumption for a matrix size 10 K

MAGMA Kernel Name	Average power consumed (W) of matrix size 8K
DGETRF	165
DGEHRD	150

5.3 Predictions for MAGMA kernels for matrix of size 10K based on AMG

From our AMG model 4.1

$$\text{Total power consumption} = \text{Idle Power} + \text{Runtime Power} \quad (5.3)$$

$$\text{Runtime Power} = \sum_{i=1}^e (N_{SM} \times P_{u,i} \times U_{u,i}) + B_{u,i} \times U_{u,i} \quad (5.4)$$

5.3.1 Power prediction for MAGMA DGEMM

MAGMA Double precision General Matrix Matrix multiply uses the GPU completely. Power is predicted using our AMG model. Even though MAGMA kernels do a very good job of hiding data latencies with computations utilization rates for shared memory and global memory is 100%.

$$\text{Power consumed by floating point unit} = 14 * 2.2 \times 0.58 + 6 \times 0.58 = 17.864 \text{ W}$$

$$\text{Power consumed by shared memory} = 14 \times 1 \times 1 + 3 \times 1 = 17 \text{ W}$$

$$\text{Power consumed by global memory} = 14 \times 3 \times 1 + 10 \times 1 = 52 \text{ W}$$

$$\text{Total runtime power} = 86.864 \text{ W}$$

$$\text{Idle power} = 80$$

$$\text{Total power Predicted} = 166.864 \text{ W}$$

$$\text{Total power measured} = 180 \text{ W}$$

$$\text{error} = 7.3\%$$

MAGMA DGEMM achieves 58% of the theoretical peak that is the reason utilization for floating point is .58.

5.3.2 Power prediction for MAGMA DGEMV

MAGMA Double precision General Matrix Vector does not utilize the GPU fully as the matrix-vector operations get stalled by memory since the memory reads and writes to global memory are not as fast as floating point operations on shared memory or registers.

$$\text{Power consumed by floating point unit} = 14 \times 2.2 \times 0.4 + 6 \times 0.4 = 14.72$$

$$\text{Power consumed by shared memory} = 1 \times 14 \times 0.2 + 3 \times 0.2 = 3.4 \text{ W}$$

$$\text{Power consumed by Global memory} = 3 \times 14 \times 0.8 + 10 \times 0.8 = 41.6 \text{ W}$$

$$\text{Total runtime power} = 63.4 \text{ W}$$

$$\text{Idle power} = 80 \text{ W}$$

$$\text{Total power Predicted} = 143.4 \text{ W}$$

$$\text{Total power measured} = 135 \text{ W}$$

$$\text{error} = 6.2\%$$

The total power predicted is close to the measured power consumption. One of the difficulties was finding the execution time of each unit. Floating point units are busy only half of the time as memory needs to be fetched. Shared memory is used for $1/5^{th}$ of the time and global memory is used the rest of the time, that is the reason performance of DGEMV is only 30 GFLOPS; it is a memory bound kernel.

5.3.3 Power prediction for MAGMA DGETRF

MAGMA LU factorization's power consumption is close to DGEMM power consumption because 100% of flops in MAGMA LU are from MAGMA DGEMM.

$$\text{Power consumed by floating point unit } 14 * 2.2 \times .48 + 6 \times .48 = 17.64$$

$$\text{Power consumed by shared memory } 1 \times 14 \times 1 + 3 \times 1 = 17$$

$$\text{Power consumed by global memory } 3 \times 14 \times 0.6 + 10 \times 0.6 = 31.2$$

$$\text{Total runtime power} = 65.2$$

$$\text{Idle power} = 80$$

$$\text{Total power predicted} = 144.64 \text{ W}$$

$$\text{Total power measured} = 165 \text{ W}$$

$$\text{error} = 12\%$$

The MAGMA DGETRF achieves 48% of the peak that is why utilization for floating point unit is .48 and all the available shared memory is used.

5.3.4 Power prediction for MAGMA DGEHRD

MAGMA Hessenberg's power consumption is close to MAGMA DGEMV power consumption as 20% of the total flops of the algorithm, which take 70% of time, are in level 2 BLAS.

$$\text{Power consumed by floating point unit } 14 \times 2.2 \times 0.5 + 6 \times 0.5 = 18.4$$

$$\text{Power consumed by shared memory } 1 \times 14 \times 0.2 + 3 \times 0.2 = 3.4 \text{ W}$$

$$\text{Power consumed by Global memory } 3 \times 14 \times 0.8 + 10 \times 0.8 = 41.6 \text{ W}$$

Total runtime power = 63.4 W
Idle power = 80 W
Total power Predicted = 143.4 W
Total power measured = 150 W
error = 4.6%

5.3.5 Analysis of predictions for MAGMA kernels using AMG

Texture memory is used heavily for MAGMA kernels, and since texture cache was not included in our model, the total power prediction is affected by this. We also find that it fails to yield accurate estimates for kernels with constant cache accesses because of the lack of constant memory model for monitoring constant memory accesses, resulting in significant underestimation for such kernels. The NVML sensor readings are accurate with ± 5 W so we can safely say that our model predicts power accurately.

5.4 Summary

In this chapter we have analyzed power consumption of MAGMA BLAS 2, BLAS3, LU, and Hessenberg kernels. BLAS 3 kernels utilize the GPU completely, that is the reason we observe power consumption of power consumption of these kernels close to the maximum. BLAS 2 kernels are memory bound kernels, i.e. they are limited by memory latencies which are large compared to the fast floating point operations within registers or shared memory. GPU running MAGMA is compared with a CPU running LAPACK and GPU uses as little as 1/50 of energy. Predictions for MAGMA kernels power consumption using AMG are also noted. The maximum error is found to be 12% mainly because we do not have a model for texture cache and texture

cache is heavily used in MAGMA kernels. In the next chapter we are going to discuss power measurements through PAPI.

Chapter 6

Performance Application Programming Interface

PAPI is an acronym for Performance Application Programming Interface. The PAPI Project is being developed at the University of Tennessee's Innovative Computing Laboratory. This project was created to design, standardize, and implement a portable and efficient API (Application Programming Interface) to access the hardware performance counters found on most CPUs and other components such as GPUs, sensors, and others.

Many microprocessors such as Intel's Sandy Bridge, Nvidia C2075 offer performance counter support. We develop the API so that application programmers can improve their performance based on the information provided by counters.

Some of the important features of PAPI are:

- Platform independent comparison and analysis.
- Presents standard definitions for cross platform development.
- Standardize API among users, vendors.
- Open software, user support, well documented and structured.

6.1 PAPI NVML Component

The PAPI Nvidia Management Library Component offers to provide an interface to measure power and temperature on devices that support offer to support the NVM library fully. Power consumption is instantaneous so we report power from Nvidia cards at that instance.

```
1 PAPI_event_name_to_code( "NVML.TeslaC2075.Device0.  
    Device_Get_Power_Usage", &(Events[0]) )  
2 PAPI_event_name_to_code( "NVML.TeslaC2075.Device0.Device_Get_Temperature  
    ", &(Events[1]) )  
3 PAPI_create_eventset(&EventSet)  
4 PAPI_add_events( EventSet, ( int * ) Events, NUMEVENTS )  
5 PAPI_start(EventSet)  
6 invoke the kernel  
7 PAPI_stop( EventSet, values )  
8 printf("Power usage is %lld \n", values[0] )  
9 printf("Temperature is %lld \n", values[1] )
```

Listing 6.1: PAPI source code

Listing 6.1 shows the the name of the event. The name of the event is split into different parts that are defined as name of component, name of device, device number, and capability of event. `PAPI_event_name_to_code` converts the name of the event into a PAPI event code. Since multiple events are added we have to create an event set. `PAPI_start` starts PAPI counters; `PAPI_stop` would stop the counting.

```

1 PAPI_VERSION      :      4      2      1
2 Name NVML.TeslaC2075.Device0.Device_Get_Temperature — Code: 44000001
3 Name NVML.TeslaC2075.Device0.Device_Get_Power_Usage — Code: 44000000
4 END: Hello World!
5      51      —> NVML.TeslaC2075.Device0.Device_Get_Temperature
6     32365    —> NVML.TeslaC2075.Device0.Device_Get_Power_Usage

```

Listing 6.2: PAPI output

6.2 Other Power Monitoring Components

6.2.1 PowerMon 2

A component is being developed for PowerMon 2 [1]. PowerMon is a device that can be inserted between a system's power supply and motherboard to monitor the current and voltage of all 8 rails. Devices such as Kill-A-Watt have a very low frequency of measurement, on the order of once a second, and since power is instantaneous, measuring power with such a low frequency device inhibits the observation of fine changes in power. PowerMon offers us a chance to observe changes in power as it has a very high frequency of 1024 Hz. Use of PowerMon has many advantages over other measurement tools. PowerMon can be connected to host systems via USB which can monitor power for the device. A microcontroller is used so that various tasks performed by PowerMon such as scheduling, logging, and timestamping do not add overhead to the system. The AMD1191 digital power monitor IC on each power rail is used to detect voltage and current. The software used in PowerMon is a modified version of Till Harbaums i2c-tiny-usb [28]. PowerMon2 is capable of reading 3000 readings per second.

The advantages of using PowerMon are

1. GPU power consumption can be measured

2. Isolation of various components' power consumption
3. Finer measurement since the frequency is 1024 Hz
4. Timestamping and log based analysis
5. Compact
6. Self monitoring device
7. Synchronization on separate targets

6.2.2 Running Average Power Limit component

PAPI supports the Running Average Power Limit component (RAPL) on Sandy-Bridge by using the x86-msr driver [31]. RAPL provides a standard interface for limiting power used by memory. Some of the essential parameters of RAPL are power limit and time window. RAPL considers energy credits rather than setting instantaneous limits. Once the limits are established by RAPL, it uses control mechanisms to maintain that limit.

```

1
2 Trying all RAPL events
3 Found rapl component at cid 2
4
5 Starting measurements...
6
7 Doing a naive 1024x1024 MMM...
8 Matrix multiply sum: s=1016404871450364.375000
9
10 Stopping measurements, took 4.110s, gathering results...
11
12 Energy measurements:
13 PACKAGE_ENERGY:PACKAGE0    176.450363J    (Average Power 42.9W)
14 PACKAGE_ENERGY:PACKAGE1     75.812454J    (Average Power 18.4W)
15 DRAMENERGY:PACKAGE0         11.899246J    (Average Power 2.9W)

```

```

16 DRAM.ENERGY:PACKAGE1      8.341141J    ( Average Power  2.0W)
17 PP0.ENERGY:PACKAGE0      118.029236J   ( Average Power 28.7W)
18 PP0.ENERGY:PACKAGE1      16.759064J   ( Average Power  4.1W)
19
20 Fixed values :
21 THERMAL_SPEC:PACKAGE0     135.000W
22 THERMAL_SPEC:PACKAGE1     135.000W
23 MINIMUM_POWER:PACKAGE0    51.000W
24 MINIMUM_POWER:PACKAGE1    51.000W
25 MAXIMUM_POWER:PACKAGE0    215.000W
26 MAXIMUM_POWER:PACKAGE1    215.000W
27 MAXIMUM_TIME_WINDOW:PACKAGE0 0.046 s
28 MAXIMUM_TIME_WINDOW:PACKAGE1 0.046 s
29 rapl_basic.c                PASSED

```

Listing 6.3: RAPL output

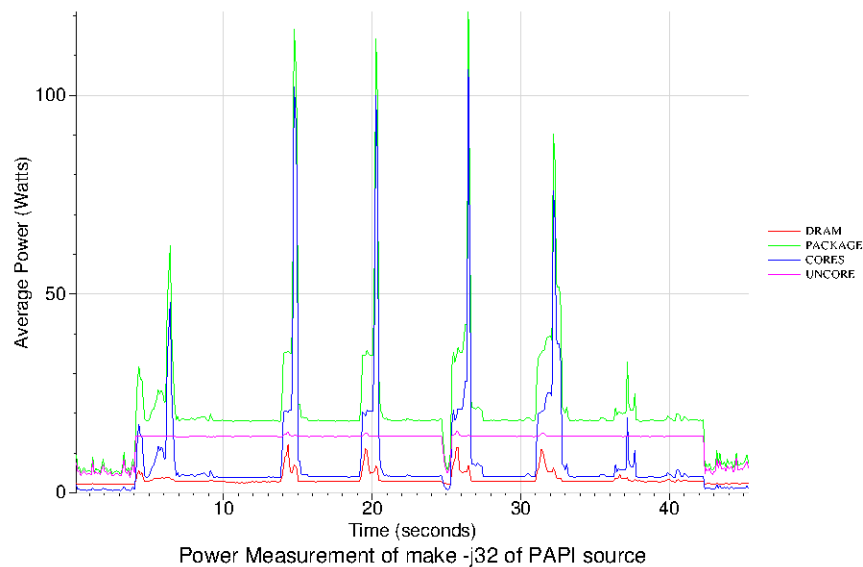


Figure 6.1: Graphical representation of RAPL

Listing 6.3 shows the output from the graphical tool [31], that needs to be compiled against PAPI and built with “`--with-components = rapl`”

6.3 Summary

In this chapter we have presented PAPI components to measure power on various devices. PAPI NVML component measures power and temperature on GPUs which support NVML fully. In the next chapter we are going to present a comprehensive analysis of our work.

Chapter 7

Conclusions

There is a huge potential of research in this field of energy-aware high performance computing with GPUs. The power consumed by the Japanese K computer is 12 MW. To address the challenges of estimating per-structure power in hardware, we proposed a new analytical model, called Activity-based Model for GPUs (AMG), to estimate activity factors and power for micro-architectural structures on GPUs. The power model using AMG predicts the power consumption and the execution time with an average of 3% error for the evaluated GPGPU kernels. Live measurements using Nvidia Management Library (NVML) is of particular interest to users, so we have measured power on Nvidia C2075 GPU using NVML. We have also analyzed power consumption of various MAGMA kernels, level 2 and level 3 BLAS kernels. We have also analyzed power consumption of arithmetic intensive MAGMA LU factorization and a memory bound MAGMA Hessenberg kernel.

This research differs from previous power estimation work in several aspects. Our model targets towards the new Fermi with fine micro-architectural details and highly variable power behavior. Our power measurement technique is non-disruptive, and the AMG-based implementation is highly-portable. The component breakdowns we produce are based on physical entities. As a result, these component breakdowns can offer a foundation for future thermal modeling research. The fact that detailed power

data can be collected in real-time is also important for thermal research, since the large thermal time constants mandate long simulation runs. Using NVML, power can be measured on Nvidia GPUs.

There are several key contributions of this work. The measurement technique itself is portable, and can offer a viable alternative to many of the power simulations currently guiding research evaluations. The component breakdowns offer sufficient detail to be useful on their own, and their properties as a power signature for power aware phase analysis seem to be even more promising. In conclusion, this work offers both a measurement technique, as well as a characterization of a GPU's various components. We feel it offers a promising alternative to purely estimation-based power research. We have developed a PAPI NVML component so that users can access power and temperature measurements through more familiar PAPI.

7.1 Future Work

Our model can also be used by compilers or programmers to optimize program configurations as we have demonstrated in the thesis. In our future work, we will build a multi CPU_GPU model, that will give us a complete picture of power consumption for a system like Keenland [14], which has 120 nodes with 240 CPUs and 750 GPUs.

Power consumption for older GPUs is reported in Power states (P0-P15), where P0 is the power state under maximum load, and P15 is idle power consumption. We would like to deduce power numbers in Watts for those states so that users can understand the meaning of power states. With ARM-based CPU/GPU hybrid systems being deployed to reduce energy consumption, issues for modelling such hybrid systems will be of special interest to us. The Barcelona Supercomputing Center is developing a new ARM based machine to achieve 4 to 10 times the energy efficiency of today's supercomputers [15].

Bibliography

Bibliography

- [1] Bedard, D. and Min Yeol Lim and Fowler, R. and Porterfield, A. PowerMon: Fine-grained and Integrated Power Monitoring for Commodity Computer Systems. In *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, pages 479–484, March 2010. [57](#)
- [2] Leo Breiman. Random Forests. *Mach. Learn.*, 45:5–32, October 2001. [13](#)
- [3] Breiman, Leo and Friedman, Jerome H. and Olshen, Richard A. and Stone, Charles J. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984. [13](#)
- [4] Jianmin Chen, Bin Li, Ying Zhang, Lu Peng, and Jih Peir. Statistical GPU Power Analysis using Tree-based Methods. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–6, July 2011. [12](#)
- [5] Farzan Fallah and Massoud Pedram. Standby and Active Leakage Current Control and Minimization in CMOS VLSI Circuits. *IEICE Transactions*, 88-C(4):509–519, 2005. [33](#)
- [6] Fatahalian, K. and Sugerman, J. and Hanrahan, P. Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '04, pages 133–137, New York, NY, USA, 2004. ACM. [36](#)

- [7] Jack Dongarra Hans Meuer, Erich Strohmaier. TOP500 Supercomputer Site. <http://www.top500.org>, 2012. 2, 3
- [8] Sunpyo Hong and Hyesoon Kim. An integrated GPU Power and Performance Model. *SIGARCH Comput. Archit. News*, 38(3):280–289, June 2010. 5, 11, 25
- [9] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 693–702, New York, NY, USA, 2002. ACM. 13
- [10] Canturk Isci and Margaret Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society. 10, 11, 14
- [11] Kill A Watt. KILL A WATT P3. <http://www.p3international.com/products/special/P4400/P4400-CE.html>. 10, 17, 19
- [12] Dong Li, Surendra Byna, and Srimat Chakradhar. Energy-Aware Workload Consolidation on GPU. *Parallel Processing WorkshopS*, 0:389–398, 2011. 7
- [13] Nath, Rajib and Tomov, Stanimire and Dongarra, Jack. An Improved Magma Gemm For Fermi Graphics Processing Units. *Int. J. High Perform. Comput. Appl.*, 24(4):511–515, November 2010. 37, 38
- [14] NICS. Keeneland Supercomputer. <http://keeneland.gatech.edu/>, 2012. 62
- [15] NVIDIA. ARM-based Supercomputer. <http://blogs.nvidia.com/2011/11/worlds-first-arm-based-supercomputer-to-launch-in-barcelona/>, 2012. 62

- [16] NVIDIA Corporation. GPGPU. <http://www.nvidia.com/page/home.html>, 2012. 1
- [17] NVIDIA Corporation. NVIDIA CUDA C Programming Guide. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf, 2012. 15, 27
- [18] NVIDIA Corporation. NVIDIA TESLA C2075 COMPANION PROCESSOR. <http://www.nvidia.com/docs/IO/43395/NV-DS-Tesla-C2075.pdf>, 2012. 24
- [19] NVIDIA Corporation. NVML API Reference. <http://developer.download.nvidia.com/compute/DevZone/NVML/doxygen/index.html>, 2012. 6, 7, 18
- [20] NVIDIA Corporation. PTX manual. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/ptx_isa_3.0.pdf, 2012. 16, 19
- [21] ORNL. Titan Super Computer. <http://www.olcf.ornl.gov/titan/titan-overview/>, 2012. 4
- [22] Piotr Luszczek. Energy Fotprint for LINPACK Benchmark from Supercomputers to Tablet Devices. In *Centre Euopen de Calcul Atomique et Molculaire*, pages 1–6, July 2011. xi, 1, 2, 3
- [23] Mahsan Rofouei, Thanos Stathopoulos, Sebi Ryffel, William Kaiser, and Majid Sarrafzadeh. Energy-Aware High Performance Computing with Graphic Processing Units. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower’08, pages 11–11, Berkeley, CA, USA, 2008. USENIX Association. 5, 10
- [24] J. W. Sheaffer, D. Luebke, and K. Skadron. A Flexible Simulation Framework for Graphics Architectures. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS ’04, pages 85–94, New York, NY, USA, 2004. ACM. 13

- [25] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-Aware Microarchitecture: Modeling and Implementation. *ACM Trans. Archit. Code Optim.*, 1:94–125, March 2004. 12
- [26] Reiji Suda and Da Qi Ren. Accurate Measurements and Precise Modeling of Power Dissipation of CUDA Kernels toward Power Optimized High Performance CPU-GPU Computing. In *Proceedings of the 2009 International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT '09, pages 432–438, Washington, DC, USA, 2009. IEEE Computer Society. 5, 9
- [27] Texas Instruments. OMAP-L138 Power Consumption Summary. http://processors.wiki.ti.com/index.php/OMAP-L138_Power_Consumption_Summary#Activity-Based_Models, 2012. 7
- [28] Till Harbaum. i2c-tiny-usb. http://www.harbaum.org/till/i2c_tiny_usb/index.shtml, 2012. 57
- [29] Stanimire Tomov. Matrix Algebra on GPU and Multicore Architectures. <http://icl.cs.utk.edu/magma/>, 2012. 7
- [30] Tomov, Stanimire and Nath, Rajib and Dongarra, Jack. Accelerating the Reduction to Upper Hessenberg, Tridiagonal, and Bidiagonal forms through Hybrid GPU-based Computing. *Parallel Comput.*, 36:645–654, December 2010. 49
- [31] Vince Weaver. RAPL PAPI. <http://web.eecs.utk.edu/~vweaver1/projects/rapl/>, 2012. 58, 59
- [32] Wikipedia. Nyquist theorem. http://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem, 2012. 48
- [33] Wikipedia. signal Averaging. http://en.wikipedia.org/wiki/Signal_averaging, 2012. 48

- [34] Wong, Henry and Papadopoulou, Misl-Myrto and Sadooghi-Alvandi, Maryam and Moshovos, Andreas. Demystifying GPU Microarchitecture through Microbenchmarking. pages 235–246, March 2010. [13](#)
- [35] Kirk Cameron Wu-chun Feng. GREEN500 Supercomputer Site. <http://www.green500.org>, 2012. [3](#), [5](#)

Vita

Kiran Kumar Kasichayanula was born in Guntur, Andhra Pradesh and raised in Hyderabad, a city located in the southern part of India. He graduated from St. John's School at Gannavaram and then moved to Hyderabad, the capital city of the Andhra Pradesh and enrolled in the Electronics and Communications Engineering undergraduate program of the Jawaharlal Nehru Technological University, Hyderabad. He received his undergraduate degree in Electronics and Communications Engineering in year 2008. In 2008, he enrolled in the Master of Science program in Computer Engineering in the University of Tennessee Knoxville. He received his Master of Science degree in Computer Engineering in 2012. Between years 2009 and 2010, he worked as an Graduate Assistant in the Office of Information Technology at UTK. Between 2010 and 2012, he worked as a Graduate Research Assistant in Innovative Computing Laboratory UTK. His current position is computer programmer in Multicoreware.