ALUPower: Data Dependent Power Consumption in GPUs

Lucas, J., & Juurlink, B. (2016). ALUPower: Data Dependent Power Consumption in GPUs. In 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE. https://doi.org/10.1109/mascots.2016.21

ALUPower: Data Dependent Power Consumption in GPUs

Jan Lucas and Ben Juurlink
Technische Universität Berlin
Berlin, Germany
Email: {j.lucas,b.juurlink}@tu-berlin.de

***Abstract*—Existing architectural power models for GPUs count activities such as executing floating point or integer instructions, but do not consider the data values processed. While data value dependent power consumption can often be neglected when performing architectural simulations of high performance Out-of-Order (OoO) CPUs, we show that this approach is invalid for estimating the power consumption of GPUs. The throughput processing approach of GPUs reduces the amount of control logic and shifts the area and power budget towards functional units and register files. This makes accurate estimations of the power consumption of functional units even more crucial than in OoO CPUs. Using measurements from actual GPUs, we show that the processed data values influence the energy consumption of GPUs significantly. For example, the power consumption of one kernel varies between 155 and 257 Watt depending on the processed values. Existing architectural simulators are not able to model the influence of the data values on power consumption. RTL and gate level simulators usually consider data values in their power estimates but require detailed modeling of the employed units and are extremely slow. We first describe how the power consumption of GPU functional units can be measured and characterized using microbenchmarks. Then measurement results are presented and several opportunities for energy reduction by software developers or compilers are described. Finally, we demonstrate a simple and fast power macro model to estimate the power consumption of functional units and provide a significant improvement in accuracy compared to previously used constant energy per instruction models.**

I. Introduction

CMOS circuits dissipate dynamic power when they charge or discharge gates and wires. The power consumption of a circuit depends on how often each of the millions of gates and wires in the circuit change state. This activity of the circuit does not only depend on the circuit itself, but also on the data the circuit processes. RTL power simulators such as PowerMill [1] use test vectors to estimate activity factors for all signals.

Often, higher level simulators are required. Architectural power simulators are typically used in early design phases when circuit details are not yet known. Complex circuits consisting of thousands of gates are often abstracted into simple power macro models [2], [3]. Abstracting away individuals signals provides a tremendous reduction of simulation time and allows architectural power simulators to estimate power consumption for workloads that are out of the reach of circuit or RTL power simulators. Many of these high-level power modeling techniques do not model the data dependency of the power consumption [4]. To the best of our knowledge, no currently publicly available architectural simulator for GPUs takes the dependency of execution power consumption on actual data values processed into account. Modeling the energy consumption of the GPU datapath is important as it consumes significant parts of total energy consumption. Leng et al. [5] estimate that on average 44.9% of the GPU power is consumed by execution units, register files and pipelines. Our measurements reveal large changes of more than 100 W in total power consumption depending on the data values processed by the datapath. We executed a small test kernel on an NVidia GTX 580 GPU with three different input vectors. The results of this experiment are shown in Fig. 1. The test kernel reads input vectors from DRAM and then executes FMUL operations on these inputs. First, the power consumption of an all zeros test vector is measured. Executing the test kernel on this input keeps the floating point ALUs and most parts of the datapath idle. On this input the GPU consumes 155 Watt. Then we use random input vectors, where each thread works on different values and on average half of the floating point multipliers input bits flip each cycle, this results in an increase of 77 W. Finally we execute the test kernel with an input vector of all zeros and all ones, where all input bits switch each cycle. In this case the total power consumption is 102 W; 65.8% higher than with all zero values.

We observe that the energy consumption of the three kernel runs differs substantially, even though all three kernels execute exactly the same number of instructions, the execution order has not changed nor the bandwidth consumed. Only the data values processed are different. The activity factors used by simulators such as GPUWattch [5] or GPUSimPow [6] for all three kernel executions would be completely identical. As a result the predicted power consumption would also be identical. Therefore, not taking data values into account results in large prediction errors for the power consumption of this microbenchmark. Microarchitectural modifications such as different schedulers that change the order of execution might change the energy consumption of arithmetic units significantly, but these changes cannot be evaluated using current simulators.

By developing an accurate yet light-weight power model of real GPUs, we enable many new kinds of microarchitectural optimizations. Even techniques that might increase the runtime or the number of operations could be beneficial, if they can provide a significant reduction in the average number of bit flips in the arithmetic unit.

This paper is structured as follows. Section II provides an overview of related work. In the following Section III relevant architectural details of the Fermi GPU datapath are explained

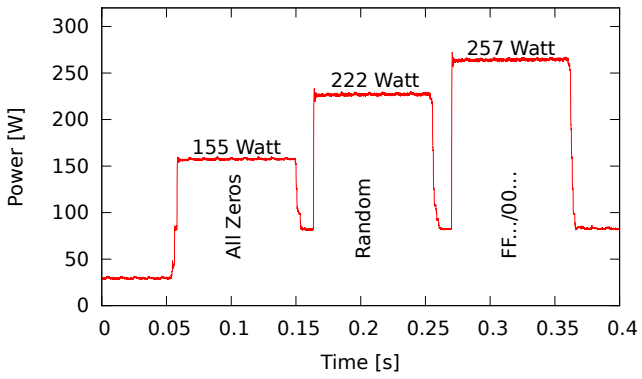


Fig. 1. Power Consumption of FMUL test kernel with three different input vectors on GTX580

as well as their influence on the design of our microbenchmarks. Section IV explains how the actual measurements were performed and how our test vectors were generated. Before looking at the details of ALU Power consumption on the Fermi GPU, in Section V-A we employ a portable CUDA microbenchmark to show that data values impose a strong influence on GPU power consumption, even on the cards using the latest Maxwell architecture [7]. Section V-B explains how the register file and data values influence the power consumption even if the input values supplied to the functional units are constant. The following Section V-C provides an overview of the results of our measurements. Section VI builds the ALUPower power model for GPU ALUs using the measured values and provides an initial discussion of its accuracy. In Section VII the accuracy of the power model is evaluated and compared to the constant energy per instruction models employed by current GPU architectural simulators. Section VIII concludes the paper.

II. Related Work

Many popular architectural power simulators for CPUs and GPUs ignore the data values processed by the datapath. McPAT's CPU core power model [8], for example, counts various register reads, uses of integer and floating point ALUs, integer multiplies, register renaming, but neither the exact instruction type nor any statistics about the processed data values are used to predict the power consumption.

GPUWattch [5] and GPUSimPow [6] are power simulators for GPUs. Both simulators are based on gpgpu-sim and extend it with a McPAT-based GPU power model. These architectural simulators count activity factors for various GPU units and use them to estimate the power consumption of the GPU. Counted activities are integer or floating point instructions, register reads and write, memory accesses, etc. None of the used activity factors measures how often datapath lines switch between 0 and 1.

One exception to ignoring data values is the original Wattch power simulator [9]. Wattch contains a DYNAMIC_AF option in its source code that collects activity factors based on average population count of the processed values, but this is only used for some internal buses and memories but not for the ALU. In fact, the Wattch source code contains the comment: "FIXME: ALU power is a simple constant, it would be better to include bit AFs and have different numbers for different types of operations". The Wattch authors apparently recognized that this was a weak point in their simulator. For a CPU power simulator where control logic dominates the datapath, using such a simple model might still be acceptable, but accelerators such as GPUs try to keep the control logic small and simple and use large parts of their power and area budget for execution units and register files. In this paper we will show that for these accelerators more accurate power models are required.

Kim, Austin, Mudge and Grunwald [10] also recognize that architectural power simulators ignore values and memory addresses in their power estimation. They describe how an architectural simulator for CPUs that considers values could be build and developed a prototype based on SimpleScalar but did not validate their model.

Some related work exists for microprocessors. Sarta, Trifone and Ascia propose a data dependent power model for a simple DSP with a 2-stage pipeline [11]. They find that operands strongly influence the energy consumption and also employ linear least square fitting. Kerrison and Eder [12] model the energy consumption of a hardware multi-threaded microprocessor. They consider the overhead of switching from one instruction to another and the influence of data values on energy consumption.

III. Measuring GPU ALU Energy

Measuring the energy consumed by the GPU datapath requires knowledge about its structure and on how microbenchmarks can be designed that trigger specific test patterns at the functional units and register files without large unwanted and unpredictable side effects. This section focuses on the datapath details of the Fermi architecture. A general introduction into the architecture of NVidia GPUs can be found in [13], [14].

Fig. 2 shows a simple GPU datapath with register files and integer and floating point arithmetic units. Many GPUs hide the latency of memory access and functional units by switching execution between multiple warps [15]. They also use very wide SIMD units. Because of these design choices, GPUs require huge register files. The GTX 580 GPU employed in this paper contains register files with a total capacity of 2 MB. Conventional multi-ported register files enable high-performance CPUs to fetch many operands in a single cycle but consume large amounts of energy and die space even for register files of just few kBs. In order to enable large register files, NVidia GPUs split their register file into several banks of single-ported SRAM. [16] Single-ported SRAM is more area and energy effective than multi-ported RAM. Operand collectors connected via a routing network to the register bank fetch the operands in several cycles. If operands are distributed evenly over the banks, this organization can approach the performance of a conventional multi-ported register file. In our example we show four register file banks.

The datapath of the Fermi architecture is even more complex (Fig. 3). Each core (called streaming multiprocessor by NVidia) contains two warp schedulers. Each warp scheduler has its own register file. One of the warp schedulers executes all even warps, while the other warp scheduler executes all odd warps. Some execution units are exclusively connected to one

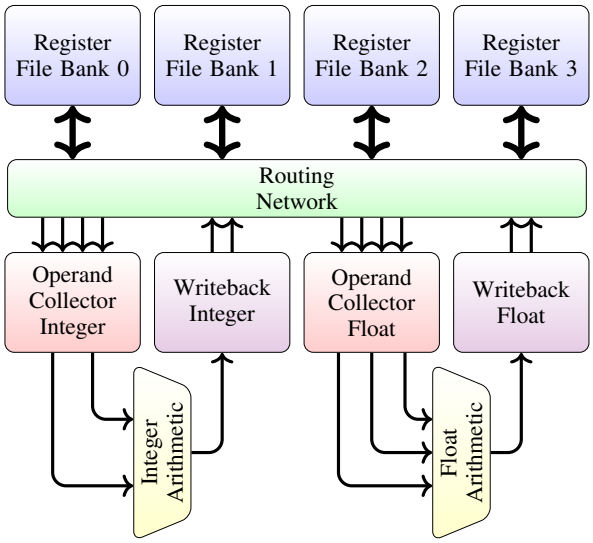


Fig. 2. GPU register file and datapath

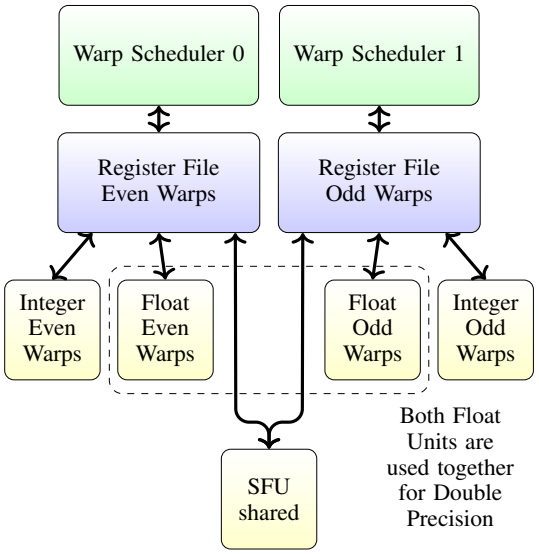


Fig. 3. Fermi datapath

warp scheduler, while others are accessible by both schedulers. Integer performance is half the floating point performance and the integer units are potentially shared as well or are just smaller units with lower throughput.

Unfortunately, we cannot build a power model by directly measuring the energy consumption of a single ALU while it performs a single instruction. Two main reasons prevent such an approach to GPU ALU power modeling. The first one is that the energy consumed in the execution of one instruction does not only depend on the instruction itself and its operands, but also depends on the previously executed instruction and previous operands. A meaningful model must therefore measure the power consumption of pairs of instructions and pairs of input operands. The second reason is that we cannot isolate the ALU from other components of the GPU and measure only the power consumed by a single ALU. Isolated measurements of the power consumption of a single ALU are only possible using detailed circuit simulations or special test chips manufactured for such purpose. Both would require access to a wealth of proprietary design files and other secret information only GPU designers have access to. Estimating GPU ALU energy consumption using self-developed RTL descriptions of such an ALU would allow using circuit and RTL level power tools and measuring the energy in isolation. The energy estimations produced by such a model could be much higher than commercial GPU ALUs because they lack the many person-years of manual optimization performed on them, or they could also underestimate the actual energy consumption because they would likely lack some functionality available in commercial GPU ALUs.

The energy consumption of the ALUs can be estimated, without being able to measure only the ALU itself, by measuring the GPU power consumption twice. While changing the input operands to the ALU and keeping all other activities of the GPU constant, the energy consumption of the ALU can be calculated. As each operation uses only a tiny amount of energy, for precise measurements we need to repeat the operation many times and also execute the operation on many identical ALUs in parallel. This way differences in energy consumption that are within the pJ range per operation can be measured using energy measurements in the mJ to J range.

Measuring the GPU power consumption requires kernels that mostly execute a specific pair of instructions with a specific pair of inputs and nothing else. Writing such a code in a high level language such as CUDA or OpenCL would cause the optimizer in the compiler to spot that the code executes many redundant operations and eliminate most of the test code. Even if the test code is written using an intermediate language such as SPIR, HSAIL or PTX, optimizers in the driver can still remove parts of the test code or reorder instructions. Both would prevent valid measurements of the GPU power consumption. For this reason the test code needs to be written in the actual ISA of the GPU and should run without changes on the GPU. Unfortunately most GPU manufacturer do not support that developers write code directly in the actual ISA of their GPUs. NVidia provides a disassembler for the ISA, but does not provide an assembler nor a description of the instruction set. The disassembler allowed Yunqing to reverse engineer an assembler for Fermi called asfermi [17]. Not all instructions are supported yet, but enough instructions are supported to test the most important functional units: integer arithmetic, logic and basic floating point operations such as FADD and FMUL. Our test code first loads the test vectors from DRAM into registers and then executes instruction pairs in a loop with aggressive unrolling. Unrolling ensures that the GPU executes mostly our test instruction pair and that the energy and time spent on executing loop handling code is negligible.

Fig. 4 shows a short version of our microbenchmark. First we load the test vectors into registers and then we execute a long unrolled loop. Only one set of registers is used. Fermi GPUs use 32 threads per warp, but use only 16 wide functional units to execute each warp. First, the first half of the warp is executed, then in a second cycle the second half of the threads is executed. This is illustrated in Fig. 5, but for brevity, only 3 of the 16 units are shown. By loading different test vectors into the first and the second half of the warp, we can stimulate

```
(...) // calculate test vector address in R2
// load test vectors into registers
LD.E R4,[R2+0x00]
LD.E R5,[R2+0x04]
(...) // load loop counter in R2
!Label loop
FMUL R4,R5,R6
(...) // 63x more FMUL
IADD R2,R2,-1
ISETP.EQ P0, P7, R2, RZ
@!P0 BRA !loop
EXIT
```

Fig. 4. Microbenchmark Code written in Fermi Assembly

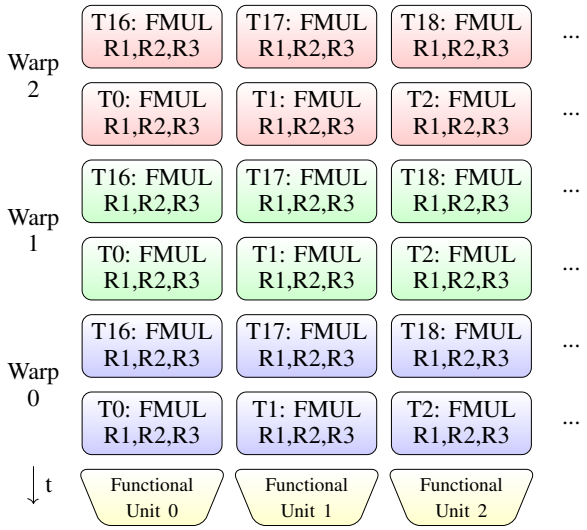


Fig. 5. Execution of Warps on GPU Functional Units

the functional unit with different test vector pairs and measure how much energy is consumed due to the transition. As the two halfs of the warp are always executed together we could even execute multiple identical warps at the same time and would still get the same transitions, no matter how the warp scheduler schedules warps. We, however, used only one active warp at a time to avoid additional interference at the register file.

IV. Experimental Methodology

We executed our test code on an NVidia Geforce GTX 580 card based on NVidia's Fermi architecture. A short overview of its parameters is provided in Table I. By using a Fermi architecture card the test code could be written using asfermi. To measure the energy consumption, a power measurement test-bed similar to the ones used in GPUSimPow [6] and GPUWattch [5] was used. The NVidia CUDA command line profiler was used to gather kernel start and end times. Before the main test code was executed, a few test kernels were executed to generate a known series of power consumption spikes. These spikes were used to calculate the offset between the profiler clock and the sample clock. The power samples and the start and stop times from the profiler were then used to calculate the energy consumption of each executed kernel.

TABLE I. GPU configuration in experimental evaluation.

Parameter	Value	Parameter	Value
GPU cores (SMs)	16	Integer units / core	16
Warp Scheduler / Core	2	Float units / core	32
Core clock	1.5 Ghz	Memory clock	2 Ghz

As we want to quantify the dynamic power of executing instruction with different inputs, for every input vector we executed our measurement kernel twice: Once with the test vector and once with a baseline vector of all zeros. We assume that this all-zeros test vector triggers the minimum number of signal transitions and allows us to measure static power and the power used for fetching and scheduling the instructions. In both cases we execute the same number of instructions in the same order. DRAM memory transactions triggered by our kernel are equal in both runs and are insignificant as our code runs from cache and only a few kilobytes of code and test vectors are loaded from DRAM on each kernel execution. Even though our baseline vector is the same for all measurements, we execute it again for every new test vector. This is required because the leakage power consumption depends on the temperature of the GPU. By executing both the test vector and the baseline test vector at nearly the same time and each only for a few milliseconds, the large thermal inertia of the GPU ensures that the temperature of GPU and thus the temperature dependent leakage power is almost constant in both measurements. All differences in energy consumption between the two kernel executions should therefore be due to the different input vectors.

We tested our measurement equipment and microbenchmarks by measuring the energy consumption of 10 values 100 times each for all configurations. A high repeatably of the measurements was observed and we found an average standard derivation of only 0.9 pJ for repeated measurements of the same data point. Even this small measurement error will, however, add to our prediction error, as we cannot predict the noise.

A simple metric to characterize the value pairs in our test vector set is the Hamming distance; the number of bits that differ between two words. Initially we tried using random test vectors but discovered that almost all random test vectors had medium Hamming distances and vectors with low or high Hamming distance were rare. We then improved our test vector generation. The Hamming distance of 32-bit values is between 0 and 32. With two inputs and one output test vectors, $33 \times 33 \times 33$ combinations of Hamming distances could exists. We tried to find 10 samples for each combination. As the output Hamming distance depends on the inputs for some combinations no test vectors or only a smaller number of test vectors could be found. For most instructions we started with an initial set of around 220.000 test vectors. This test vector set turned out to be too large, since it resulted in measurement durations of several days without noticeable improvement in accuracy compared to smaller sets. We then randomly selected two disjoint sets of 50.000 values each from our initial set of test vectors. One set was used for fitting the coefficients, and the other set for initial validation.

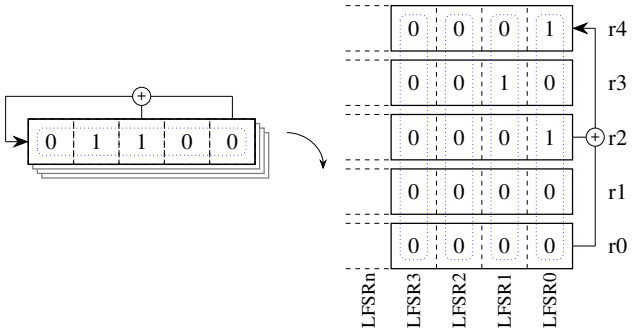


Fig. 6. Bit-sliced Linear Feedback Shift Register

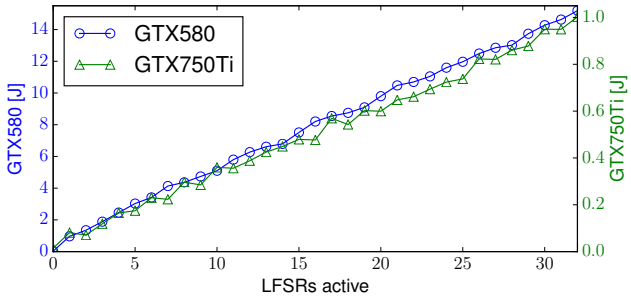


Fig. 7. ALU energy consumption of GTX580 and GTX750Ti depending on the number of active LFSRs

V. Experimental Results

In this section we first perform an experiment to validate that data dependent energy consumption is not exclusive to Fermi GPUs. Then we perform more detailed measurements on Fermi GPUs, at first results obtained with static input values and then we present measurement results with changing test vectors.

A. Data-Dependent Power Consumption on Fermi and Maxwell

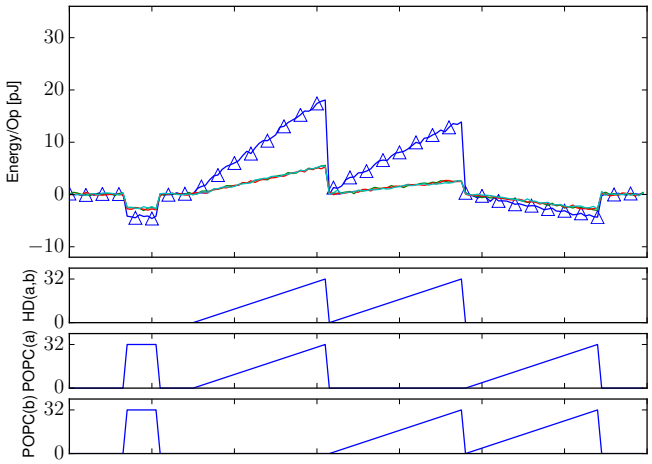
As explained in the last section, the assembler available for Fermi allows us to perform detailed measurements of Fermi GPUs, but we wanted to verify that our hypothesis, i.e. data values strongly influence the power consumption of GPUs, is also true on more recent GPUs. For this reason we designed a special microbenchmark that can be written in a high level language such as CUDA or OpenCL. The microbenchmark does not allow the detailed measurements required to build an ALU power model, but measuring its power consumption substantiates our hypothesis. The microbenchmark evaluates 32 linear feedback shift registers (LFSR) in parallel using a bit-sliced implementation. Fig. 6 shows a 5-bit LFSR and the bit-sliced implementation. In this implementation the state of each LFSR is not stored in a single register, but instead the state of LFSR 0 is stored in the bits at position 0 of registers r0 to r4, the state of LFSR 1 is stored in the bits at position 1 of the same registers, and so on. Shifting instructions are not used but instead the mapping between logical LSFR bits to physical registers rotates each cycle to account for the shifting. After unrolling this results in a long chain of xor instructions in a loop.

LFSRs are often used as pseudo-random generators, but have a special property that we exploit for our microbenchmark: An LFSRs with an initial state of all zeros stays in that state constantly, while any other initial state generates a pseudo random sequence. Controlling the initial state of the LFSRs allows us to adjust the number of bits that flip during the execution of this microbenchmark. If all our 32 LFSRs are loaded with all zeros, the power consumption should be low as the xor input will be constantly zero. If all LFSRs are loaded with a non-zero initial state the power consumption should be higher as the inputs will change. In Fig. 6 LFSRs 2 and 3 are loaded with all zero bits and stay in the locked up state, while the state of LFSR 0 and 1 is changing in every cycle. As this code is written in pure CUDA we can execute it on the GTX580, but can also use a GTX750Ti card that employs NVidia's latest Maxwell architecture.

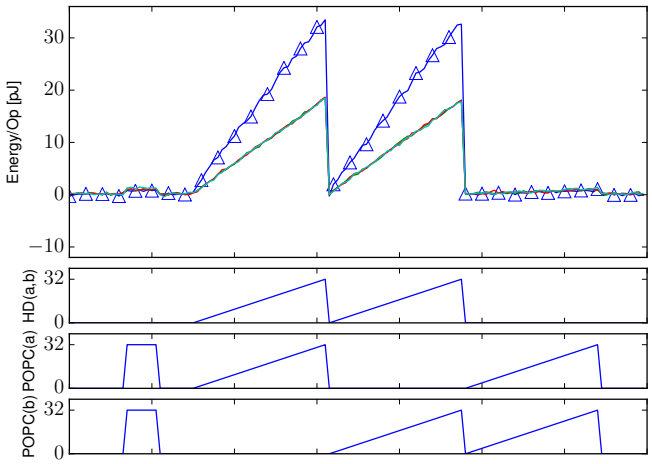
Fig. 7 shows the results of this experiment. The energy consumption displays an almost completely linear relationship with the number of active LFSRs on both cards. The total energy consumption of GPU is not shown in the diagram but in both cards the change between 0 LFSRs active and all 32 LFSRs active amounts to around 30% of average total energy consumption (GTX580: 28.8%, GTX750Ti: 31.2%). We scaled the working set size with the number of cores: the smaller GTX750Ti card only executes $5/16$ the work of the bigger GTX580 card, but the GTX750Ti card with its Maxwell architecture is still almost 5 times as energy efficient.

B. Impact of Register File

We initially assumed that, if we execute the same instructions over and over again using constant inputs, the signals and gates of the GPU datapath should stay in a constant state and their power consumption should not change depending on which constant inputs are used. The measurements revealed, however, that this assumption is not true and that the energy consumption of the GPU datapath depends on the constants used and also on the warp that is executing the instruction. The results of these measurements, for even and odd warps, are shown in Fig. 8a and 8b. In this experiment we measure the energy consumption of the GPU with various constant inputs relative to the energy consumption with all zeros as input. Negative values indicate that the operation with these inputs uses less energy than the same operation with all zeros as input. We picked test vectors with different Hamming distances between the operands and with different number of set bits in the operand, also known as population count (POPC). In Fig. 8, below the energy measurements, we show the properties of the test vectors: on the top the Hamming distance between the two inputs, followed by the population count of the first operand (called a) and the second operand (called b). There are also large difference depending on the registers used to store the input values. As explained earlier NVidia GPUs use banked registers and we found that the power consumption is higher if the register number of registers a and b modulo 4 matches. This is shown in the triangle marked line with higher energy consumption. The three unmarked lines show experiments where the test vectors are fetched using different register banks. This makes it likely that 4 register banks are used and the higher power consumption happens when both



(a) Even Warps



(b) Odd Warps

Fig. 8. Register file energy consumption for LOP.AND

operands need to be fetched sequentially from the same bank. On the other hand if different banks are used each bank can stay in a constant state. This presents a power optimization opportunity for the compiler: During register allocation the compiler should avoid fetching values with large Hamming distances from the same register bank.

Even if there are no differences between the first and second operand, a difference in power consumption exists depending on how many bits are set. In even warps some input vectors use less power than our all zero base line. This can happen, if some wires or gates are charged to 1 first and need to be recharged only if a 0 is transmitted but can stay constant otherwise. The energy consume by the four different register banks is almost identical in our measurements. The same effects but smaller and with opposite sign happen in the odd warps. We also notice that in even warps set bits on input port b consumed less energy than those on port a. For many instructions input ports a and b can be swapped. Using this information the compiler could swap the input ports based on input statistics for a small power reduction.

C. Energy Consumption per Instruction

After these initial experiments we measured the energy for each instruction of with pairs of test vectors. These results are employed in the next section to develop a model and evaluate the power model, but first we look at the average power consumption of each instruction and how much their energy consumption differs with different input values.

The average energy consumption of AND, OR, XOR and IADD is very similar with 60.9, 60.9, 60.7 and 61.5 pJ, respectively. 95% of the AND instructions use between 28.3 and 91.5 pJ. OR and IADD instructions share a similar upper end with 97.5% of the instruction below 91.4 and 94.4 pJ, respectively. Upper bound for XOR is slightly lower at 81.3 pJ. Lower bounds are similar for OR, XOR and IADD with 28.8 and 30.6, 29.6 pJ, respectively. FADD on average uses 94.0 pJ while the average for FMUL is 95.3 pJ. 95% of the FADD instructions use between 36.8 to 135.0 pJ, while FMUL instructions consume between 49.3 to 131.5 pJ. IMUL is the instruction with the highest average power consumption (165.6 pJ) as well as the largest interval (76.7 to 218.3 pJ).

VI. ALU Energy Model

Based on our previous findings we selected 6 parameters, one coefficient for every parameter, and one constant offset. The following equation describes how energy estimates are predicted by our model:

$$\begin{aligned} E(a_0,b_0,a_1,b_1) = c_0 + c_1 HD(a_0,a_1) + c_2 HD(b_0,b_1) \\ + c_3 HD(o_0,o_1) + c_4 HD(a_0,b_0) + c_5 HD(a_1,b_1) \\ + c_6(POPC(a_0)+POPC(a_1)+POPC(b_0)+POPC(b_1)) \end{aligned} \quad (1)$$

In this equation a_0 and b_0 are the inputs of the first executed instruction and a_1 and b_1 are the inputs of the second executed instruction. o_0 is the output of the first instruction and o_1 is the result of the second instruction. Four parameters are Hamming distances (HD) between the input operands. Parameters $HD(a_0,a_1)$ and $HD(b_0,b_1)$ represent changes to the input wires. These parameter reflect the energy consumed by the wiring to the functional unit. The parameters $HD(a_0,b_0)$ and $HD(a_1,b_1)$ are designed to catch interference between two operands from different input ports of the functional unit. This either happens in the internal logic of the functional unit or in the register file. In addition our model uses the Hamming distance of the two different outputs ($HD(o_0,o_1)$) and based on our findings from the register file also one parameter for the population count of the inputs. For this parameter we do not differentiate between the inputs and add together the population counts (POPC) of all four input values. The coefficients c_0 to c_6 depend on the instruction type, even or odd warp, and the register banks used by input registers.

One limitation of this model is that it does not model state changes within functional units due to different operations. For example, OR and AND will likely be executed by the same functional unit. Even if their inputs and the output does not change, changing the executed operation will likely change parts of the internal state, which will consume additional energy. Modeling the energy consumption of these internal

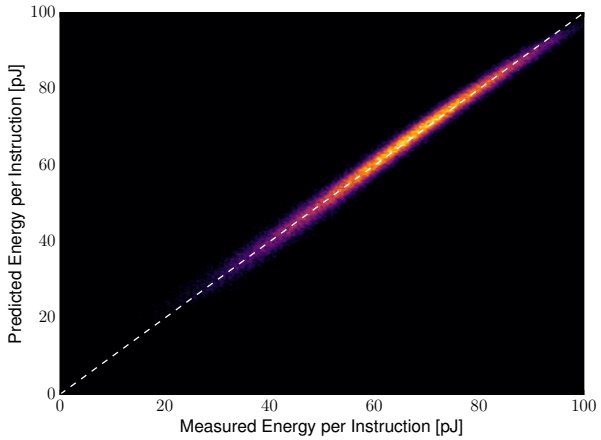


Fig. 9. Predicted Energy vs. Actual Energy for LOP.AND (Even Warps)

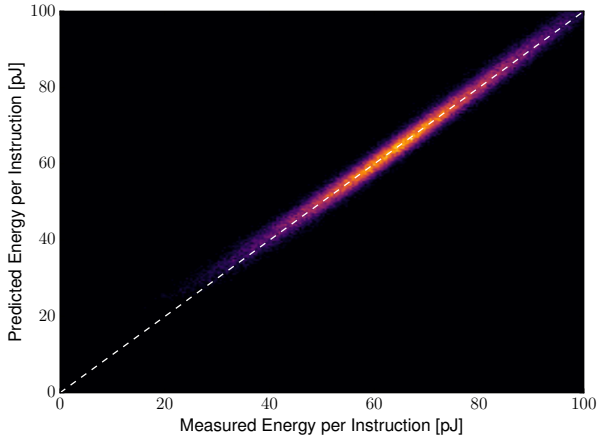


Fig. 10. Predicted Energy vs. Actual Energy for IADD (Even Warps)

state changes is considered future work. ALUPower estimates energy consumed. To estimate power the architectural simulator must accumulate the energy estimates and divide the accumulated energy by the collection period. Because ALUPower is a linear model, the energy does not have to be evaluated each time an instruction is executed, but the input parameters can be accumulated and the energy can be calculated at the end of each evaluation period. Hamming distances can be calculated very quickly, especially when hardware population count instructions are available such as the x86 popcnt instruction.

We used linear minimal least squares fitting on our measured data points to determine the values of the coefficients c_0 to c_6. One set of coefficients was calculated for each instruction and for each of the four different combinations of even or odd warp and input registers from the same bank or from different banks. To evaluate our model we measured the energy consumption of another set of test vectors disjoint from our initial set. Then our coefficients were used to predict the energy consumption of these new test vectors. After these steps, for each test vector the actual as well as the predicted energy consumption is available. Heat maps are used to show how many samples we found for each combination of predicted and

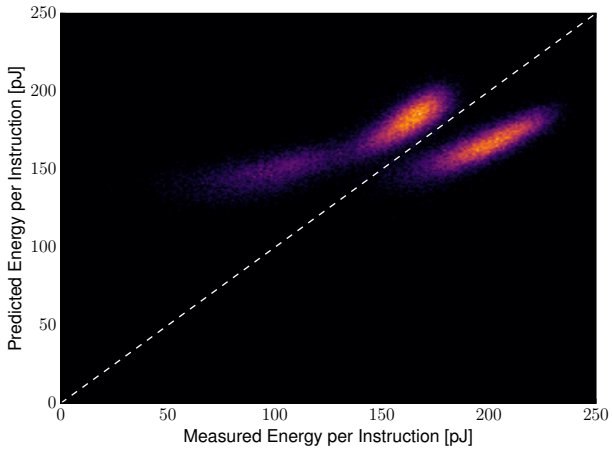


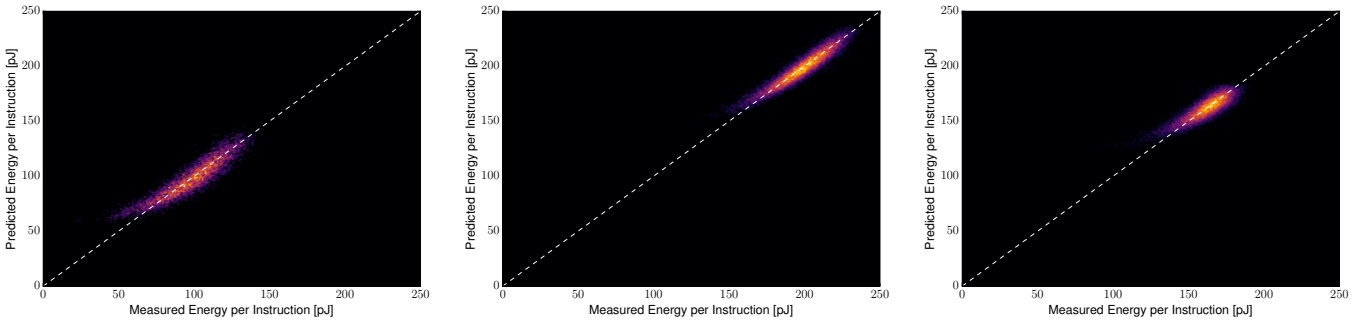
Fig. 11. Predicted Energy verus Actual Energy for IMUL (Even Warps)

TABLE II. IMUL TEST VECTORS FROM EACH CATEGORY

0 sign bit flips		**1 sign bit flips**		**2 sign bit flips**	
Op 1	***Op 2***	***Op 1***	***Op 2***	***Op 1***	***Op 2***
3×5	2×1	3×5	-2×1	3×-5	-2×1
-2×-8	-1×-4	-2×8	-1×-4	-2×-8	-1×-4
1×-2	7×-3	1×-2	7×3	1×-2	-7×3

measured energy. The heat-map for the LOP.AND instruction is shown in Fig. 9. In a perfect model together with perfect noiseless measurements, all points would fall on the diagonal dashed white line. For samples that are above the diagonal the predicted energy is higher than measured energy, while for samples below the diagonal the predicted energy is lower than the actually measured energy. Samples that are further from the diagonal have a larger prediction error. The prediction error stems from different source: limitations of the power model, but also measurement noise. Samples in the center of the heat-map are more common because test vectors with few or many bit flips are relatively rare compared to test vectors with average bit flip statistics. If measurement noise and the test vector distribution is taken into account, the model's predictions of the AND instruction are very close to optimal. As discussed in Section IV, even with a perfect model, noise would cause an average prediction error of 0.9 pJ and our prediction error for AND is just slightly larger at 1.3 pJ. Heat-maps for the two other logic operations OR and XOR show very similar results and are therefore not shown. Fig. 10 shows the prediction heat-map for the IADD instruction and again the predictions are very accurate, even though IADD is more complex than simple logic operations. No internal circuit details such as the state of the carry chain are modeled, but the energy model still performs very well.

Fig. 11 shows the results of our initial model for the IMUL instruction. Here we notice a different picture: Instead of a line close to the diagonal three clouds of samples can be observed, two of them above the diagonal, one below. We extracted some test vectors from each cloud and searched for a property that can be used to classify the test vectors. The important difference between the samples are the sign bits of their test vectors. We classified each IMUL test vector into three categories and fitted different coefficients for each category. Table II shows example test vectors to



(a) No sign bit flips (b) One sign bit flips (c) Both sign bits flips

Fig. 12. Predicted Energy vs. Actual Energy for IMUL for each category (Even Warps)

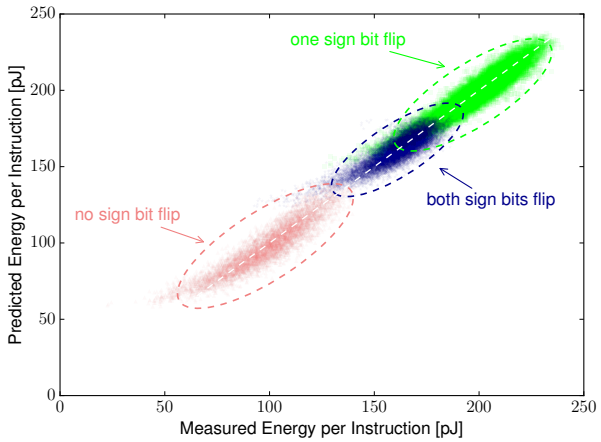


Fig. 13. Predicted Energy vs. Actual Energy for IMUL using classification (Even Warps)

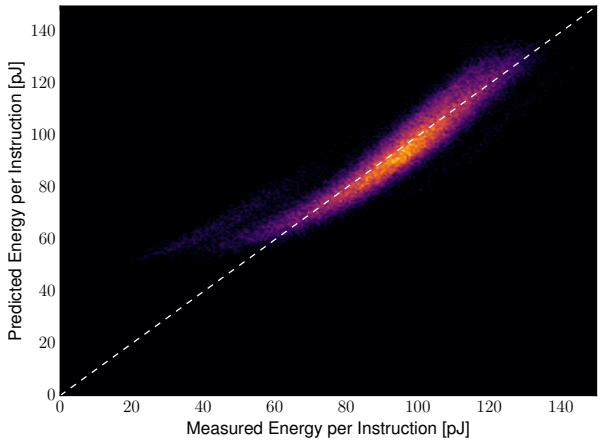


Fig. 14. Predicted Energy vs. Actual Energy for FMUL (Even Warps)

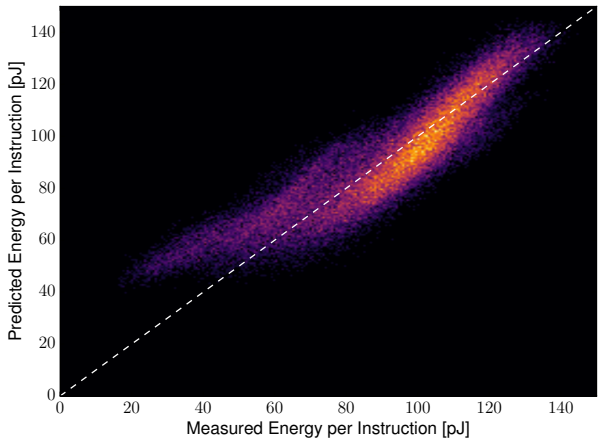


Fig. 15. Predicted Energy vs. Actual Energy for FADD (Even Warps)

clarify the categories: If all sign bits stay constant this results in the lowest power consumption. The results of our prediction for this category is shown in Fig. 12a. The highest power consumption appears if only one of the sign bits flip (Fig. 12b). Very likely the reason for the behaviour is the flipped sign of the output. If both input sign bits flip, then the output sign does not change. In our measurements this results in a slightly lower power consumption than if only one sign bit flips as shown in Fig. 12c. This property has been integrated in our model by using three different sets of coefficients for the IMUL instruction. Fig. 13 shows the results of the integrated model. A scatter plot instead of a heat map is used to also display the classification of each data point. Most points are close to the diagonal now, albeit not as close as the points for the simpler logic and IADD instructions.

Results for the floating point multiply and add operation are shown in Fig. 14 and 15. For these two operations our model is not as accurate as for integer and logic instructions but much better than our first version of the IMUL model. The heat maps seem to indicate that there might be a property of the test vectors that influences the energy consumption that is not incorporated in our model. We tried to improve the accuracy of the model for floating point instructions by considering the exponent values (for example, if the exponents of the FADD differ a lot, the output will be close to one of the inputs), but this did not improve the prediction accuracy. In order to provide an even more accurate energy model of the floating point unit, additional insight into the design of this specific execution unit is required.

TABLE III. COEFFICIENTS FOR EVEN AND ODD WARPS, SAME REGISTER FILE BANK, COEFFICIENTS IN pJ

Coefficient	c_0		c_1		c_2		c_3		c_4		c_5		c_6		***Average Energy***	
multiply by	1		$HD(a_0, a_1)$		$HD(b_0, b_1)$		$HD(o_0, o_1)$		$HD(a_0, b_0)$		$HD(a_1, b_1)$		$POPC$		-	
Warp	***Odd***	***Even***	***Odd***	***Even***	***Odd***	***Even***	***Odd***	***Even***	***Odd***	***Even***	***Odd***	***Even***	***Odd***	***Even***	***Odd***	***Even***
LOP.AND	17.97	14.64	0.82	0.63	0.93	0.95	1.00	0.99	0.06	0.06	0.52	0.12	-0.09	-0.01	*64.10*	*56.34*
LOP.OR	5.18	4.54	0.82	0.63	0.93	0.95	1.00	0.98	0.07	0.07	0.52	0.11	0.11	0.15	*64.16*	*56.27*
LOP.XOR	12.32	10.38	0.87	0.68	0.99	0.99	0.72	0.76	0.10	0.09	0.54	0.13	0.01	0.07	*64.09*	*56.68*
IADD	13.97	11.65	0.87	0.67	0.98	0.99	0.71	0.74	0.06	0.06	0.51	0.11	0.01	0.07	*65.02*	*57.53*
IMUL$_{\text{no sign}}$	43.09	43.45	1.97	1.77	3.32	3.33	0.38	0.38	0.15	0.08	0.57	0.11	0.06	0.12	*169.16*	*161.61*
IMUL$_{\text{one sign}}$	135.22	134.20	1.57	1.36	1.97	1.98	0.13	0.14	0.14	0.11	0.59	0.16	0.00	0.06	*169.16*	*161.61*
IMUL$_{\text{both sign}}$	120.55	117.75	1.05	0.85	1.00	1.01	-0.05	-0.05	-0.37	-0.34	0.08	-0.29	0.05	0.11	*169.16*	*161.61*
FMUL	43.42	42.46	1.15	1.07	1.28	1.35	0.18	0.17	0.10	0.08	0.58	0.17	-0.03	0.14	*94.07*	*96.49*
FADD	45.94	45.80	1.52	1.56	1.41	1.39	0.31	0.31	0.01	-0.01	0.42	0.04	-0.17	-0.04	*92.73*	*94.67*

The coefficients used in our model and the average energy per instruction are shown in Table III. Coefficients for odd and even warps are different as odd and even warps are managed by different schedulers and layout differences cause differences in energy consumption. For reasons of space only the coefficients for input registers from the same bank are shown. The coefficients for input registers from different register banks are very similar. The main difference to the coefficients shown here are smaller coefficients for $HD(a_1, b_1)$, which is due to reduced interference of the operands in the register file, as discussed in Section V-B.

VII. ACCURACY

After developing the energy model and performing an initial visual evaluation, we calculate the root mean square error (RMS error) for every instruction and compare it to the previously used constant energy models. These models do not consider data values but assume a constant energy consumption for each instruction. For these models we assume that every instruction uses the average amount of energy for that instruction type, as this minimizes the average error of a constant energy model. Fig. 16 shows the root mean square error for the constant-energy and ALUPower energy models. Our data-dependent ALUPower model is significantly more accurate for all instructions. The geometric mean of the RMS error is 85.6% smaller than the error of current architectural power models for GPUs. The largest accuracy improvement is for AND instructions, where ALUPower provides 91.9% more accurate predictions. Even for the FADD instruction our energy model is 60.5% more accurate. The constant energy model has different coefficients for different instruction types and odd/even warps, while GPUWattch and gpusimpow [5], [6] only differentiate between integer and floating point instruction. These even simpler models would perform even worse than the constant energy model employed here.

Test vectors that consume large amounts of energy in the real GPU ALU should also consume large amounts in the model, i.e. measurements and predictions should be strongly correlated. Fig. 17 shows the Pearson correlation coefficient of measurements and our predictions. A correlation coefficient of 0 implies no correlation, while 1.0 would be perfect correlation. Since the constant energy model does not consider the data values at all, its predictions are not correlated at all to actual power consumption and its correlation coefficient is 0 for all instructions. The ALUPower energy model exhibits a high average correlation of 0.976 between the actual energy consumption and the prediction. The OR instruction shows the highest correlation of 0.996, while the FADD and FMUL instructions show a correlation of 0.917 and 0.948, respectively. The predictions for IMUL have a correlation of 0.987.

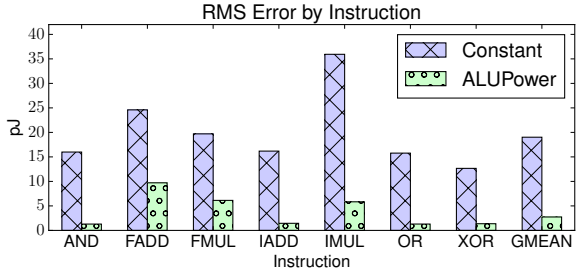


Fig. 16. RMS Error of Constant-Energy and ALUPower

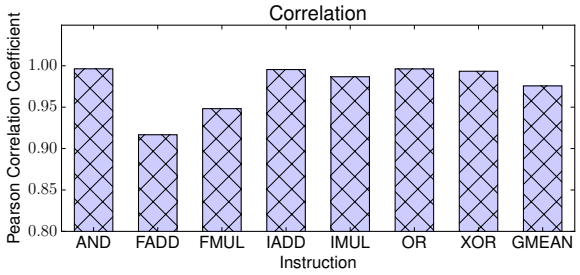


Fig. 17. Pearson Correlation Coefficient

The error results so far were calculated with test vectors with similar statistical properties to the test vectors used to perform the linear least square fitting of the model. The high accuracy on these test vectors could have been the result of overfitting. To validate that our model works well even with different test vector sets, we executed several Rodinia [18] GPU benchmarks and benchmarks included with the simulator on a modified gpgpu-sim[13]. For every tested instruction, gpgpu-sim was modified to extract value pairs used in thread 0 of each block and write them to a file. We randomly picked 1000 disjoint test vectors for each instruction, measured their power consumption on the actual GPU and compared them to the predictions. The results of this experiment are shown in Fig. 18. On this set of test vectors ALUPower has an average RMS error of 7.8 pJ, instead of 2.7 pJ on the generated test vectors. However, the error of the constant-energy model has increased from 19.0 pJ to 31.8 pJ. So the RMS error of the constant energy model increases even more than that of ALUPower. In this data set small unsigned integer values are much more common than in the generated data sets. For

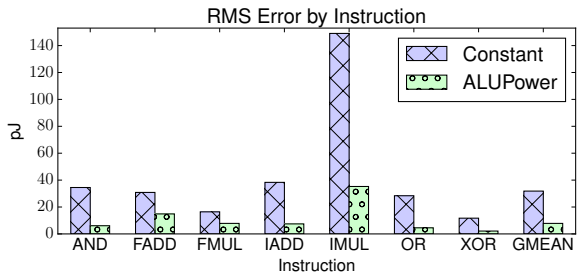


Fig. 18. RMS Error of Constant-Energy and ALUPower (Validation Dataset)

this reason, especially the constant energy model for IMUL predicts significantly worse.

VIII. Conclusions

The design and results of ALUPower, an energy model for GPU ALUs, were presented. The main contributions can be summarized as follows:

- We developed the ALUPower energy macro model for GPU ALUs based on measurements of commercial high performance GPUs.
- ALUPower improves the prediction accuracy by 85.6% over previous models and exhibits an average correlation of 0.976 to measured results on real GPUs.
- We demonstrated the large ($\geq 30\%$) influence of data values on the energy consumption on both Fermi and Maxwell GPUs.
- We identified several potential energy optimizations for code running on GPUs, such as optimized register allocation or swapping operands to reduce energy consumption.
- ALUPower enables the development of new architectural optimizations to GPUs and similar architectures.

In the future, we intend to integrate the ALU energy model into GPU architectural simulators such as GPUWattch. This will enable the development of additional optimizations of the GPU architecture that cannot be evaluated properly using current GPU simulators, such as special warp or register fetch schedulers that are aware of values and try to execute instructions with similar inputs consecutively to reduce the power consumption. ALUPower can also be useful for modeling different GPU architectures as its coefficients can be scaled based on process node and voltage. Our LSFR benchmark can be used to calculate the scaling factor or, if an assembler is available, the microbenchmarks can be ported and new coefficients can be determined. DVFS and boost clocking schemes also benefit from more accurate energy predictions using hardware counters for input statistics such as average Hamming distances. Without a data dependent power model such as ALUPower, GPU architects aiming at reducing GPU ALU energy consumption are limited to techniques that reduce the number of executed instructions. ALUPower enables optimizations that reduce the energy consumption by reordering instructions to execute instructions with similar values consecutively on the same functional unit. It also enables fair evaluation of techniques such as new warp schedulers that reorder instructions for different reasons, as reordering instructions can sometimes increase the energy consumption. Conventional power models leave GPU architects oblivious, while ALUPower makes them aware of these effects.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688759 (Project LPGPU2).

References

[1] C. X. Huang, B. Zhang, A.-C. Deng, and B. Swirski, "The Design and Implementation of PowerMill," in *Proc. Int. Symp. on Low Power Design, ISPLED*. ACM, 1995.

[2] S. Gupta and F. N. Najm, "Power Macromodeling for High Level Power Estimation," in *Proc. Design Automation Conference, DAC*. ACM, 1997.

[3] F. Klein, G. Araujo, R. Azevedo, R. Leao, and L. C. Dos Santos, "On the Limitations of Power Macromodeling Techniques," in *Proc. Symp. on VLSI, ISVLSI*. IEEE, 2007.

[4] E. Macii, M. Pedram, and F. Somenzi, "High-level Power Modeling, Estimation, and Optimization," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, vol. 17, no. 11, 1998.

[5] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *Proc. Int. Symp. on Computer Architecture, ISCA*. ACM, 2013.

[6] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a Single Chip Causes Massive Power Bills GPUSimPow: A GPGPU Power Simulator," in *Proc. Int. Symp. on Performance Analysis of Systems and Software, ISPASS*. IEEE, 2013.

[7] "NVIDIA GeForce GTX 750 Ti Whitepaper," http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf.

[8] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Proc. Int. Symp. on Microarchitecture, MICRO*, 2009.

[9] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proc. Int. Symp. on Computer Architecture, ISCA*. ACM, 2000.

[10] N. S. Kim, T. Austin, T. Mudge, and D. Grunwald, "Challenges for Architectural Level Power Modeling," in *Power aware computing*, 2002.

[11] D. Sarta, D. Trifone, and G. Ascia, "A Data Dependent Approach to Instruction Level Power Estimation," in *Low-Power Design, 1999. Proc. Alessandro Volta Memorial Workshop on*. IEEE, Mar 1999.

[12] S. Kerrison and K. Eder, "Energy Modeling of Software for a Hardware Multithreaded Embedded Microprocessor," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 3, Apr. 2015.

[13] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *Proc. Int. Symp. on Performance Analysis of Systems and Software, ISPASS*, 2009.

[14] N. Leischner, V. Osipov, and P. Sanders, "Fermi Architecture White Paper," http://www.nvidia.de/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2009.

[15] J. Nickolls and W. J. Dally, "The GPU Computing Era," *IEEE Micro*, vol. 30, no. 2, March 2010.

[16] J. E. Lindholm, M. Y. Siu, S. S. Moy, L. S., and J. Nickolls, "Simulating Multiported Memories Using Lower Port Count Memories," 2004, patent US 7339592.

[17] H. Yunqing, "asfermi," https://github.com/hyqneuron/asfermi.

[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proc. Int. Symp. on Workload Characterization, IISWC*. IEEE, 2009.