# MEMPower: Data-Aware GPU Memory Power Model

Jan Lucas(✉) and Ben Juurlink

Embedded Systems Architecture, TU Berlin, Einsteinufer 17, 10587 Berlin, Germany
{j.lucas,b.juurlink}@tu-berlin.de
http://www.aes.tu-berlin.de/

**Abstract.** This paper presents the MEMPower power model. MEMPower is a detailed empirical power model for GPU memory access. It models the data dependent energy consumption as well as individual core specific differences. We explain how the model was calibrated using special micro benchmarks as well as a high-resolution power measurement testbed. A novel technique to identify the number of memory channels and the memory channel of a specific address is presented. Our results show significant differences in the access energy of specific GPU cores, while the access energy of the different memory channels from the same GPU cores is almost identical. MEMPower is able to model these differences and provide good predictions of the access energy for specific memory accesses.

**Keywords:** GPU · Memory · Power modeling · Data dependent power

## 1 Introduction

GPUs focus on applications with high computational requirements and substantial parallelism that are insensitive to latency [1]. Large caches are ineffective for GPUs due the execution of thousands of parallel threads [2]. These factors cause GPUs and many GPU applications to require memory interfaces that provide significantly higher DRAM bandwidth than what is required and provided for regular CPUs. GPUs usually achieve the high memory bandwidth by using special graphics DRAM memories with lower capacity but wider and faster interfaces, such as GDDR5. These high throughput memory interfaces consume a significant amount of power. Modeling their power consumption accurately is thus important for architectural GPU power simulators.

In our previous work, we have shown that data values influence the energy consumption of GPU ALU operation significantly [3]. While executing the same sequence of instructions the power consumption changed from 155 W to 257 W, when the processed data values were changed. In this work we demonstrate that energy cost of memory transaction also is influenced significantly by the data values written to the DRAM or read from the DRAM. MEMPower provides


M. Schoeberl et al. (Eds.): ARCS 2019, LNCS 11479, pp. 195–207, 2019.
https://doi.org/10.1007/978-3-030-18656-2_15

predictions that consider the data values used in transaction as well as the location of the transaction.

Most current discrete GPUs employ GDDR5 or GDDR5X memories [4,5]. Both employ pseudo open drain signaling (POD) [6]. In POD signaling, current flows when transmitting a zero, while no current flow happens when transmitting a one. To improve energy consumption as well as to limit the number of simultaneously switching outputs, both types of memories use data bus inversion (DBI) [7,8]. DBI encoding transmits data inverted, if that results in a lower energy consumption and uses an extra signal line to allow the receiver to reverse the inversion of the data, if required. POD signaling, together with DBI encoding, is a source of data dependent energy consumption of the memory interface.

CMOS circuits consume dynamic power when their internal circuit nodes are recharged to a different state. How much energy is consumed, depends on the load capacitance of this node and the voltages. Bus wires providing long on-chip distance routing are usually structures with high load capacitance. External off-chip interfaces, also contain large loads in their drivers, receivers, wires as well as parasitic package capacitances. How often each of the wires is recharged, depends on the data and the encoding of the data transmitted over the wire. The recharging of wires and other circuit nodes partly explains, why the energy cost of memory transaction depends on the transmitted data.

Memory transactions are generated within the GPU cores, also called streaming multiprocessors (SM). In the GTX580 GPU, the SMs are organized into graphics processor clusters (GPCs) [9]. Each GPC contains 4 SMs. The GTX580 uses a full GF100 die with all four 4 SMs activated in each of the 4 GPCs.

This paper is structured as follows: We present related work in Sect. 2. Section 3 describes our experimental setup including our microbenchmarks. The following Sect. 4 shows how latency measurements can be used to discover the mapping between memory addresses and memory channels. It also describes the properties of the mapping and insights gained from latency measurements. Section 5 introduces the design of the data dependent power model and evaluates the accuracy of the model. Section 6 concludes the paper.

## 2 Related Work

GPUWattch [10] and GPUSimPow [11] do not take data values and locations into account when predicting the energy cost of each memory transaction. MEMPower takes data values into account and thus bridges the gap between architectural simulators and slow but precise RTL power simulators.

Wattch [12] collects some activity factors related to data for some memories and busses but does not model high performance GPUs and graphics DRAM.

Wong et al. used microbenchmarking to reveal various latency and cache characteristics of the GT200 [13], but do not consider energy and memory channel mapping. Mei and Chu used microbenchmarks to analyze the structure of the caches, shared memory as well as latency and throughput of the DRAM in more recent NVidia GPUs [14].

**Table 1.** GPU configuration in experimental evaluation.

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| GPU cores (SMs) | 16 | Integer units/core | 16 |
| GPCs | 4 | Float units/core | 32 |
| Core clock | 1.5 Ghz | Memory clock | 2 Ghz |
| CUDA | 6.5 | Driver | 343.36 |

## 3 Experimental Setup

For our experiments, we used an NVidia GTX580 GPU with a full GF100 chip using the Fermi architecture [9]. A short overview of its parameters is provided in Table 1. This GPU was selected for two main reasons: 1. GPGPU-Sim currently does not support more recent GPU architectures. Energy was measured using a GPU power measurement testbed that has been described in a previous work [11]. 2. Our previous work resulted in a data-dependent power model for the ALUs of this GPU [3]. This work adds the missing memory power model to enable the creation of architectural power model of the GTX580 GPU, that includes both ALU and memory data dependent power.

In order to measure the power consumption of memory transactions we developed custom microbenchmarks. These microbenchmarks execute the tested memory transaction millions of times. This allows us to measure the small energy used per transaction. In order to measure only the data dependent energy of each transaction we measure every transaction twice: Once with the test vector and once with a baseline vector of all ones. Then the energy consumed by the baseline vector is subtracted to calculate the energy difference caused by the specific test vector. Both measurements are performed at nearly the same time to ensure that the GPU temperature stays approximately constant in both measurements to avoid errors. Without this step GPU temperature variations could result in different amounts of static (leakage) power.

The microbenchmarks use inline PTX assembler to generate special load and store instructions that mostly bypass the L2 cache (`ld.global.cv.u32` and `st.wt.u32`). Even with these instructions, using the nvprof profiler, we detected that multiple accesses to the same address, issued at nearly the same time, are still combined at the DRAM. Our microbenchmark was then redesigned to avoid this issue by making sure that the different SMs are not generating accesses to the same location at nearly the same time. The profiler was used to verify that our microbenchmark generates the expected number of memory transactions. Each measurement was performed 128 times and averaged. The order of the measurements was randomized.

## 4 Memory Layout

According to NVIDIA the GTX580 features 6 different memory channels [9]. CUDA allows us to allocate space in the GDDR5 but does not provide any control over which memory channels are used for the allocation. We suspected that the different memory channels might have different properties in terms of energy consumption due to different PCB layout of the memory channels as well as internal layout GF100 differences. To use all available memory bandwidth, allocations are typically spread over all memory channels, so that all the capacity can be used and all memory bandwidth can be utilized. However, when we want to measure a specific memory channel we need to identify where a specific memory location is actually allocated. As no public API is available to query that information, we hypothesized that the differences in physical distance between the GPU cores and the memory channels would also result in slightly different latencies when accessing the memory. CUDA offers a special `%smid` register that can be used to identify the SM executing the code and a `%clock` register that allows very fine-grained time measurements. We used these two features to measure the memory latency of reading from each location from each SM. We measure the latency of each location 32 times and averaged our measurements to reduce measurement noise. For each location, this results in a 16 element latency vector, where each element of the vector shows the average memory read latency from that SM to the memory location. We detected that the latency to the same memory location is indeed different from different SMs and different memory locations show different latency patterns. We noticed that the latency pattern stays constant for 256 consecutive naturally aligned bytes. This means the granularity of the mapping from addresses to memory channels is 256 bytes, and we only need to perform our latency measurements once for each 256 byte block to identify the location of the whole block.

As the memory latency is not completely deterministic but changes slightly, e.g. due to background framebuffer accesses running in parallel to the measurement, all the latency vectors are slightly different. We solved this issue using k-means clustering [15]. We initially tried to map our latency vectors into six clusters corresponding to the six memory controllers listed in NVIDIA's descriptions of the GF100 [9]. This, however, failed to provide a plausible mapping of the memory locations, but mapping the latency vectors into twelve clusters was successful.

When we assume twelve clusters, all latency vectors are located close to one of the twelve centroids and the second closest centroid is much farther away. The number of points that gets assigned to each cluster is also approximately equal. When we access only locations mapped to one centroid, we achieve approximately 1/12 of the bandwidth achieved, when all locations from all channels are used. This pattern also continues if we selected larger subsets of the centroids, e.g. selecting locations from two clusters results in 1/6 of the bandwidth. The nvprof profiler also provides additional hints that the identified mapping is correct: Many DRAM counters are provided twice, one counter for something called subpartition 0 and another counter for subpartition 1. If we access only locations

from a single cluster, we notice that only one of these two performance counters is incremented significantly, while the other counter stays very close to zero. This indicates all locations in each of the clusters are part of the same subpartition.

Lopes et al. list six L2 Cache banks with two slices each for GTX580 [16]. The GTX580 has a 384-bit wide memory interface. Six 64-bit wide channels together with the 8n prefetch of GDDR5 would result in a fetch-granularity of 64 bytes per burst. Memory access patterns that only access 32 consecutive bytes and do not touch the next 32 bytes would always overfetch 32 bytes per transaction and would result in an effective bandwidth of less than half the peak bandwidth. However, our experiments showed better than expected performance for 32 byte fetches. An additional hint at 32 byte transaction is also provided by the NVIDIA profiler, where many DRAM related performance counters are incremented by one per 32 bytes. This indicates that the GTX580 can fetch 32 bytes at a time, which is consistent with twelve 32-bit channels. From these findings, we estimate that the GTX580 uses six memory controllers with two subpartitions in each controller and one 32-bit wide channel per subpartition.

As twelve is not a power of two, the GTX580 cannot simply use a few address bits to select the memory channel. Round-robin mapping of addresses to memory channels is conceptually simple but would require a division of the addresses by twelve.

Figure 1 provides a graphical representation of the recovered memory mapping of 1 MB block of memory. Each pixel represents a 256 byte block, each of the 64 lines represents $64 \times 256\,\mathrm{B} = 16\,\mathrm{kB}$. The memory mapping seems to be

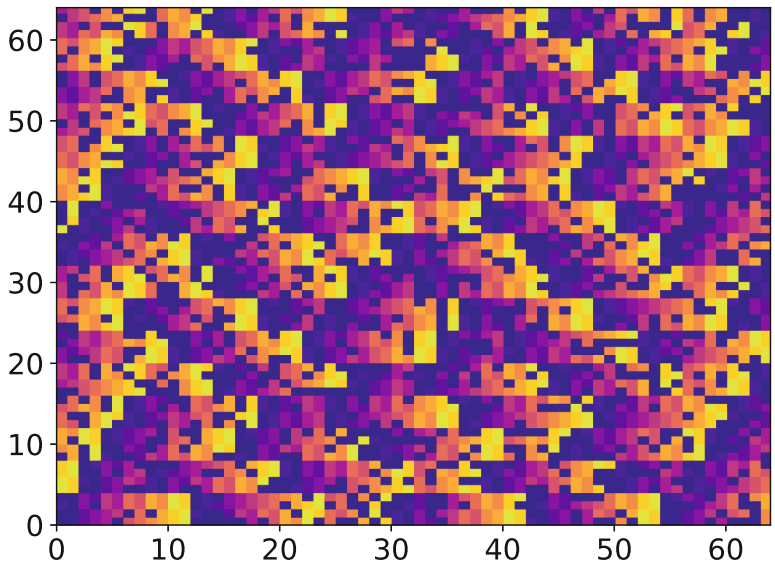


**Fig. 1.** 1 MB memory block with recovered memory channel mapping, each pixel is equivalent to a 256 byte block

structured, but does not use any simple round robin scheme. With this mapping twelve consecutive 256B blocks, on average, use 10.6 different memory channels. A simple round robin scheme would likely result in some applications having biased memory transaction patterns that favor some memory channels over others, which would result in a performance reduction. The mapping is likely the output of a simple hash function, that makes it unlikely for applications to use a biased memory access patterns by chance. Sell describes a similar scheme used by Xbox One X Scorpio Engine [17].

We also analyzed the latency vectors (Table 2) to reveal more information about the internal structure of the GPU. We first notice that all SMs in the same GPC have nearly the same latency pattern for the memory channels. The first SM in each GPC seems to have the lowest latency. The other SMs are approximately 2, 6 and 8 cycles slower. This additional latency within the GPC does not depend on the memory channels addressed. It is also identical for all four GPCs. This indicates an identical layout of all four GPCs and a shared connection of all SMs of a GPC to the main interconnect. The latency of four memory channels is lowest at GPC1. This is also true for GPC2 and GPC3. There are no memory channels where GPC0 provides the lowest latency. We suspect that is the result of a layout such as shown in Fig. 2. This also matches well with the PCB layout of a GTX580 where DRAM chips are located on 3 of the four sides of the GF100 and the PCIe interface can be found at the bottom.

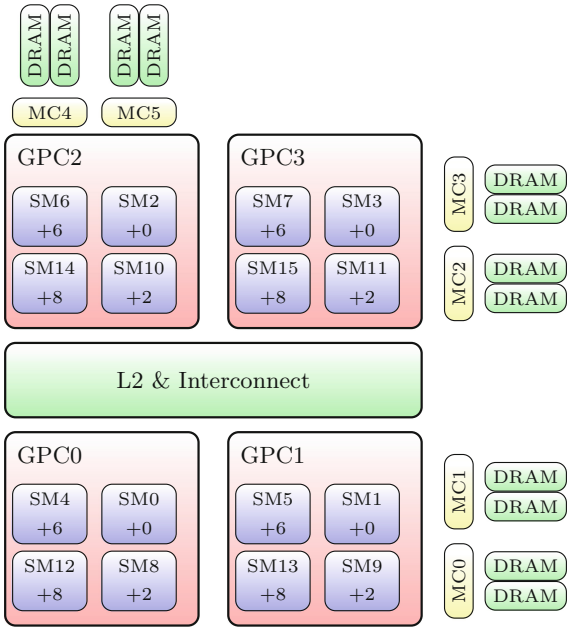


**Fig. 2.** GF100 organization

**Table 2.** DRAM latency.

| GPC0 | GPC1 | GPC2 | GPC3 |
|---|---|---|---|
| Added latency vs. GPC1 | | | |
| 3.6 | - | 3.6 | 7.5 |
| 3.9 | - | 3.8 | 7.5 |
| 7.4 | - | 3.7 | 11.2 |
| 11.3 | - | 5.7 | 15.1 |
| Added latency vs. GPC2 | | | |
| 3.6 | 3.8 | - | 0.0 |
| 3.8 | 3.8 | - | 0.1 |
| 9.4 | 3.8 | - | 5.8 |
| 11.2 | 2.0 | - | 7.6 |
| Added latency vs. GPC3 | | | |
| 4.0 | 7.6 | 4.0 | - |
| 3.9 | 7.7 | 3.9 | - |
| 3.9 | 9.5 | 5.9 | - |
| 3.8 | 9.6 | 5.7 | - |

## 5 Data-Dependent Energy Consumption

As already described in the introduction, we expect two main reasons for data dependent energy consumption: 1. Special signaling lines such as the GDDR5 DQ lines with additional energy consumption at a certain signal level. 2. State changes of wires and other circuit nodes. Our model allows a fast and simple evaluation, for this reason, we selected a simple linear model. Every memory transaction is mapped to a small vector that describes the relevant properties of the block. A dot product of this vector with a coefficient vector results in the estimated energy consumption for this transaction. The coefficient vector is calculated in a calibration process.

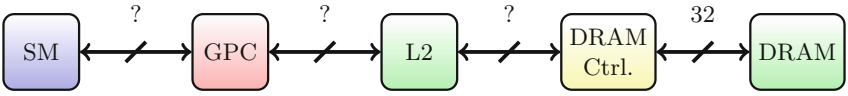


**Fig. 3.** Memory datapath

The following properties of the block are used to estimate the energy consumption. We model signal level related energy consumption by including the population count of the block. The population count is the number of set bits. We also need to estimate the amount of recharging of internal wires and circuitry caused by the transaction. Memory transactions travel through several units and various connections until they finally reach the DRAM. A simplified diagram is shown in Fig. 3. We know that the transaction travels through a 32-bit wide interface between DRAM and memory controller. Unless a reordering of bits is performed, we know which bits will be transmitted through the same wire and could cause switching activity on these wires, e.g: bits $0, 32, 64, ...$ are transmitted on the same DQ line, bits $1, 33, 65, ...$ are transmitted on the next DQ line, etc. While we know the width of the DRAM interface itself, the width of the various internal interconnections is unknown. We assume the internal link width are powers of two and are at least byte wide. The coefficients for all potential link sizes are first added to the model. During the calibration of the model, the best subset of coefficients is selected, and we indirectly gain knowledge about the internal interconnections. Because GDDR5 memory can use DBI encoded data, an extra version of each of the previously described coefficients is added to our model. This second version assumes DBI encoded data.

A synthetic set of test vectors was generated to calibrate the model. The calibration test vectors are designed to span a wide range of combinations in terms of toggles at various positions and in terms of population count. We measured the real energy consumption of our test vectors. Initially, the model uses a larger number of coefficients and some of these likely have no corresponding hardware structure in the GPU. This causes a significant risk of overfitting the coefficients to our calibration measurements. We avoid this issue by using LASSO regression as an alternative to regular least square fit [18]. Instead of fitting the

calibration data as closely as possible LASSO also tries to reduce the number of used coefficients and reduces their size. The hyperparameter $\alpha$ controls the trade off between number and size of the coefficients and prediction error with the calibration set.

In addition to the set of calibration vectors, we generated another set of test vectors to validate our model. The validation vectors are generated to mimic real application data. The vectors use various integer and floating-point data types, a mixture of random distributions with different parameters was used to generate realistic data. Real application data is often also highly correlated, some test vectors used a Gaussian process to provide correlated data.

Figure 4 shows the prediction error at various values of $\alpha$. $\alpha = 0.007$ results in the smallest error in the validation set for store transaction. Smaller values of $\alpha$ overfit the calibration set, while larger values discard important coefficients. Table 3 shows the coefficients, it should be noted that the coefficients were calculated per 512 bitflips for numerical reasons. None of the DBI coefficients are used, which indicates that the GPU is not using DBI encoding for stores. The largest coefficient corresponds to a 32 byte wide link. Coefficients for 4 and 8

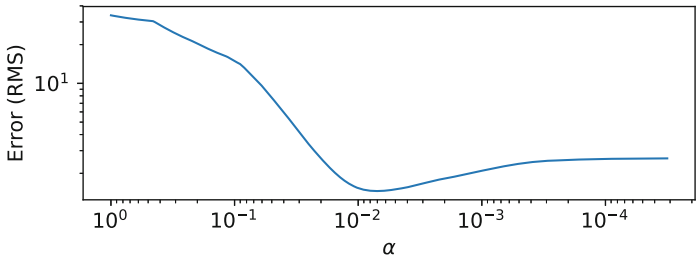


**Fig. 4.** Store access prediction accuracy vs. $\alpha$

**Table 3.** 128B transaction coefficients

**Store**

| Coefficient | DBI | Value (nJ) |
|---|---|---|
| Const | No | 7.631 |
| Pop Cnt. | No | -3.060 |
| Pop Cnt. | Yes | -0.551 |
| Toggle 1 | No | 0.031 |
| Toggle 1 | Yes | 0.036 |
| Toggle 2 | No | 0.013 |
| Toggle 2 | Yes | 0.025 |
| Toggle 4 | No | 0.933 |
| Toggle 4 | Yes | 0.084 |
| Toggle 8 | No | 0.810 |
| Toggle 8 | Yes | -0.035 |
| Toggle 16 | No | 2.276 |
| Toggle 16 | Yes | 0.042 |
| Toggle 32 | No | 9.354 |
| Toggle 32 | Yes | 0.156 |
| Toggle 64 | No | 5.169 |
| Toggle 64 | Yes | 0.132 |

**Load**

| Coefficient | DBI | Value (nJ) |
|---|---|---|
| Const | No | 9.001 |
| Pop Cnt. | No | -3.905 |
| Pop Cnt. | Yes | -0.491 |
| Toggle 1 | No | 0.009 |
| Toggle 1 | Yes | -0.005 |
| Toggle 2 | No | 0.011 |
| Toggle 2 | Yes | -0.018 |
| Toggle 4 | No | 1.676 |
| Toggle 4 | Yes | 0.000 |
| Toggle 8 | No | 0.435 |
| Toggle 8 | Yes | -0.004 |
| Toggle 16 | No | 1.021 |
| Toggle 16 | Yes | 0.000 |
| Toggle 32 | No | 7.446 |
| Toggle 32 | Yes | 0.020 |
| Toggle 64 | No | 9.919 |
| Toggle 64 | Yes | 1.872 |

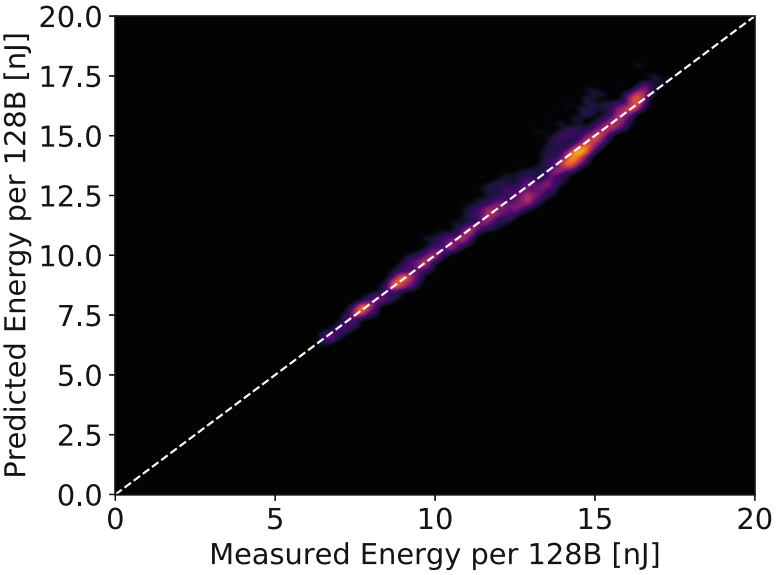


**Fig. 5.** MEMPower energy prediction for store access

byte wide links are small. Narrow 1 or 2 byte wide links are not employed. The large coefficient for a 64 byte wide link could be linked to SM internal power consumption, as the SMs use 16 wide SIMD units with 32-bits per unit.

The heatmap in Fig. 5 shows the prediction accuracy of our model for 128 byte store transactions. If the model would offer perfect prediction all points would be on the dashed white line. However, all our predictions are very close to the line which indicate a great prediction accuracy. Our RMS error is 0.39 nJ and the relative error is just 3.1%. Smaller transactions use different coefficients, results are not shown here because of the limited space. But one interesting result is that register values from disabled threads influence the energy consumption. Likely these register values are still transmitted through parts of the interconnect but marked as inactive. Taking data values into account instead of assuming a constant average energy per transaction improves the prediction error from an average error of 1.7 nJ to a error of just 0.39 nJ.

Figure 6 shows the prediction accuracy of our load model. In general, the model achieves a good prediction accuracy of 9.1% but tends to underestimate the energy required for cheaper transactions. Our load kernel achieves a significantly lower bandwidth than the store kernel as it will not send the next load transaction before the last transaction returned, while stores will be pipelined. The lower bandwidth results in a reduced signal to noise ratio of the measurements. The load coefficients printed in Table 3 indicate that load transaction are employing DBI encoding. Error improves from 2.3 nJ to 1.43 nJ.

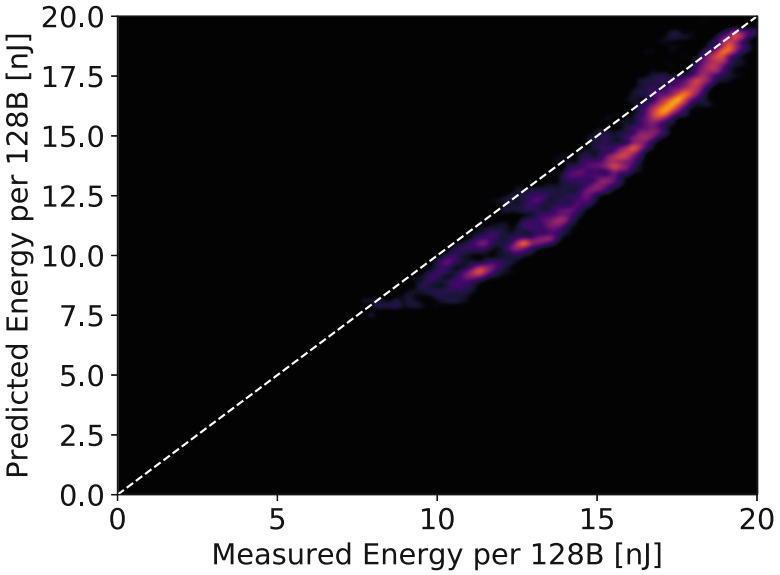


**Fig. 6.** MEMPower energy prediction for read access

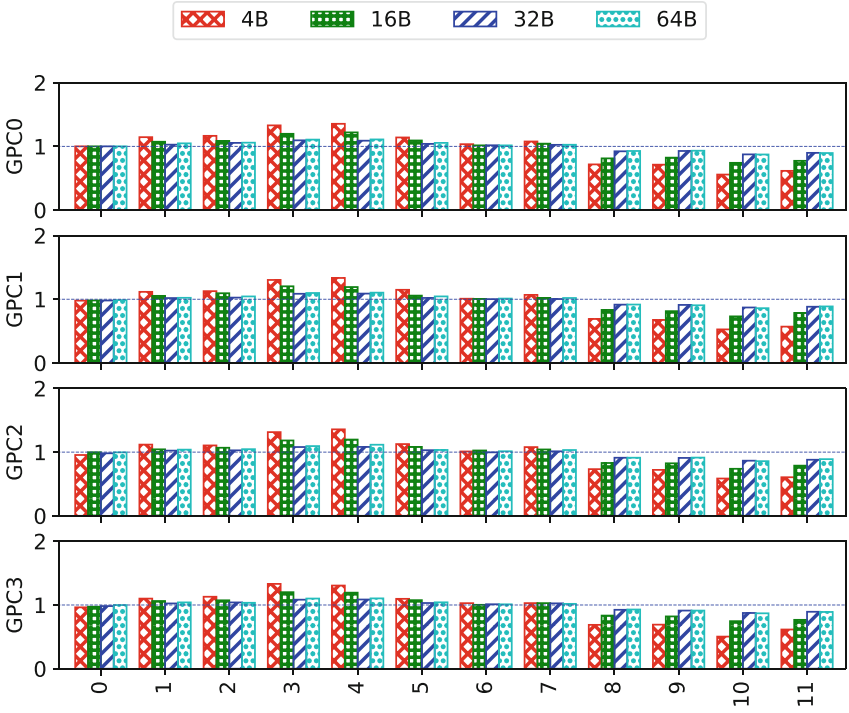


**Fig. 7.** Normalized memory channel energy consumption

We combined the microbenchmarks with the memory channel identification technique from Sect. 4 to check for energy differences between different memory channels and SMs. We tested the first SM from each GPC and used simplified test vectors to check for changes of our most important coefficients. The normalized results are shown in Fig. 7. We detected only small differences between the different SMs, however, the blue coefficient for switching activity on a 4 byte wide bus shows a large variance between different memory channels. Memory transactions to channels 8 to 11 are significantly cheaper than memory transactions on Channels 0 to 3 and 5 to 7. Memory transactions on Channels 3 and 4 are more expensive. As these results are consistent for all four GPCs, these differences are likely the result of slightly different PCB layout of the different memory channels instead of chip internal routing.

## 6 Conclusion

In this paper, we have presented the MEMPower power model for GPU memory transactions. Our contributions can be summarized as follows:

- We presented a novel technique to identify in which memory channel a specific memory address is located.
- Our microbenchmarks uncovered previously unknown architectural details of GF100-based GPUs.
- We show that memory channels are not completely identical, but differ in latency and energy consumption.
- The MEMPower model improves the energy predictions accuracy by on average 37.8% for loads compared to non-data dependent models and provides a 77.1% improvement on our validation set for stores.

At peak bandwidth data dependent changes to energy can influence the total power consumption of the GTX580 GPU by more than 25 W or around 10% of the total power. Future Work includes software and hardware techniques to reduce the energy consumption. Common but expensive data patterns could be recoded to patterns with reduced energy consumption. As memory transactions are significantly more expensive than simple ALU operations, even software solutions could be beneficial. Programmer control over data allocation could allow rarely used data to be placed in memory channels with costlier memory access and often used data in memory channels with reduced energy consumption.

**Acknowledgements.** This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 688759 (Project LPGPU2). A prior version of this work is part of the first author's defended, but currently unpublished, doctoral thesis.

## References


1. Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C.: GPU computing. Proc. IEEE **96**(5), 879–899 (2008)
2. Fatahalian, K., Houston, M.: A closer look at GPUs. Commun. ACM **51**(10), 50–57 (2008)
3. Lucas, J., Juurlink, B.: ALUPower: data dependent power consumption in GPUs. In: IEEE Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS) (2016)
4. JEDEC Standard: Graphics Double Data Rate (GDDR5) SGRAM Standard. JESD212C, February 2016
5. JEDEC Standard: Graphics Double Data Rate (GDDR5X) SGRAM Standard. JESD232A, August 2016
6. JEDEC Standard: POD15 - 1.5 V Pseudo Open Drain I/O. JESD8-20A (2009)
7. Hollis, T.M.: Data bus inversion in high-speed memory applications. IEEE Trans. Circuits Syst. II: Express Briefs **56**, 300–304 (2009)
8. Kim, J.H., et al.: Performance impact of simultaneous switching output noise on graphic memory systems. In: Electrical Performance of Electronic Packaging. IEEE (2007)
9. Wittenbrink, C.M., Kilgariff, E., Prabhu, A.: Fermi GF100 GPU architecture. IEEE Micro **31**(2), 50–59 (2011)
10. Leng, J., et al.: GPUWattch: enabling energy optimizations in GPGPUs. In: Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA) (2013)
11. Lucas, J., Lal, S., Andersch, M., Alvarez-Mesa, M., Juurlink, B.: How a single chip causes massive power bills GPUSimPow: a GPGPU power simulator. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (2013)
12. Brooks, D., Tiwari, V., Martonosi, M.: Wattch: a framework for architectural-level power analysis and optimizations. In: Proceedings of the International Symposium on Computer Architecture, ISCA. ACM (2000)
13. Wong, H., Papadopoulou, M.M., Sadooghi-Alvandi, M., Moshovos, A.: Demystifying GPU microarchitecture through microbenchmarking. In: 2010 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 235–246. IEEE (2010)
14. Mei, X., Chu, X.: Dissecting GPU memory hierarchy through microbenchmarking. IEEE Trans. Parallel Distrib. Syst. **28**(1), 72–86 (2017)
15. Hartigan, J.A., Wong, M.A.: Algorithm AS 136: a k-means clustering algorithm. J. R. Stat. Soc. Ser. C (Appl. Stat.) **28**(1), 100–108 (1979)
16. Lopes, A., Pratas, F., Sousa, L., Ilic, A.: Exploring GPU performance, power and energy-efficiency bounds with cache-aware roofline modeling. In: International Symposium on Performance Analysis of Systems and Software (ISPASS) (2017)
17. Sell, J.: The Xbox One X Scorpio engine. IEEE Micro **38**(2), 53–60 (2018)
18. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B (Methodol.) **58**, 267–288 (1996)