Inter-Warp Instruction Temporal Locality in Deep-Multithreaded GPUs

Ahmad Lashgar[1], Amirali Baniasadi[2], and Ahmad Khonsari[1,3]

[1] School of Electrical and Computer Engineering, University College of Engineering, University of Tehran, Tehran, Iran
[2] Electrical and Computer Engineering Department, University of Victoria, Victoria, British Columbia, Canada
[3] School of Computer Science, Institute for Research in Fundamental Sciences, Tehran, Iran

a.lashgar@ece.ut.ac.ir, amirali@ece.uvic.ca, ak@ipm.ir

Abstract. GPUs employ thousands of threads per core to achieve high throughput. These threads exhibit localities in control-flow, instruction and data addresses and values. In this study we investigate inter-warp instruction temporal locality and show that during short intervals a significant share of fetched instructions are fetched unnecessarily. This observation provides several opportunities to enhance GPUs. We discuss different possibilities and evaluate filter cache as a case study. Moreover, we investigate how variations in microarchitectural parameters impacts potential filter cache benefits in GPUs.

Keywords: Multi-threaded processors, Energy-efficient design, Pipeline front-end

1 Introduction

Innovative microarchitectural solutions have used locality to enhance processor performance in several ways (e.g., caches, branch predictors, etc). Many studies have investigated locality in CPUs. Locality in GPUs [3], however, has not received the same level of attention.

In this study we explore Inter-warp Instruction Temporal Locality (or simply ITL). ITL represents our observation that a small number of static instructions account for a significant portion of dynamic instructions fetched and decoded during short intervals and within the same stream multiprocessor. We investigate ITL among threads and show how GPUs come with ample ITL.

An important issue contributing to ITL in GPUs is deep multi-threading. GPUs achieve high throughput by employing and interleaving thousands of threads per core. Threads are grouped into coarser independent schedulable elements (called warps) to achieve both scheduling simplicity and SIMD efficiency. The warp scheduler issues instructions from different warps back-to-back filling the pipeline effectively. This pipeline organization amplifies ITL by fetching the same instruction for all warps during short intervals.

Moreover, we have observed that the chances of accessing a recently fetched instruction again are higher in GPUs compared to CPUs. For example, our evaluation shows the likelihood of fetching the same instruction within a 64-cycle period is 67% in CPUs. This grows to 82% in GPUs (see Section 5.1 for methodology).

Each generation of GPUs has superseded the precedent generation by increasing the number of executed warps (also referred to as multi-threading depth) and SIMD width. It is expected that deep multithreading will continue to serve an important role in performance growth in GPUs in upcoming years. Therefore, we expect the current trend in ITL in GPUs to continue in near future.

In this work, we list different opportunities to improve energy efficiency in GPUs by exploiting ITL. In particular and as a case study, we evaluate energy savings achievable under filter caches [12] in GPUs. As we show, employing a filter cache eliminates a significant share of the instruction cache accesses, improving the fetch engine's energy efficiency.

The rest of the paper is organized as follows. In Section 2 we study related works. In Section 3 we present the pipeline front-end of GPU microarchitectures and review ITL. We investigate our case study in Section 4. We present experimental setup and simulation results in Section 5. Finally, in Section 6 we offer concluding remarks.

2 Related Works

Locality studies (data or instruction) in GPUs have received less attention compared to CPUs. Collange et al [5] studied data locality in registers and found that register values often appear uniform across threads of a warp. They introduced dynamic mechanisms detecting up to 19% of the register values as uniform. Their mechanism can be exploited to reduce the number of accesses to the register file or reduce the size of vector register file. Gebhart et al [7] observed that the written registers are often read last within three instructions after the write. They introduce a register file cache to reduce the number of accesses to the conventional power-hungry register file. Moreover, to prevent heavy contention on register file cache, they introduce a two-level warp scheduler. At the first-level they maintain a few active warps. At the second-level other inactive warps are stored. Each warp at the first-level has a few registers in the cache. The scheduler issues instructions from these warps for as long as possible. Once a warp at the first-level stalls on the completion of a long latency instruction (due to operand dependency), the scheduler replaces it with a ready warp from the second-level. The mechanism effectively saves 36% of register file power, which translates to 3.8% of chip dynamic power.

Collagne et al. [4] stressed different NVIDIA GPUs by various workloads to reveal the effect of workload on GPU power. They evaluated power for different number of SIMD processors and reported energy per memory access and energy per ALU instruction. Hong and Kim [10] observed that the optimal performance/watt differs for various types of workloads, depending on the number of active cores. They introduced a performance-power model to predict the optimal number of cores for achieving the best performance/watt dynamically. They also reported the contribution of each architectural module to the dynamic power. They found that GPU's front-end (fetch, decode, and scheduling) accounts for 18% of GPU's dynamic power. Zhang et

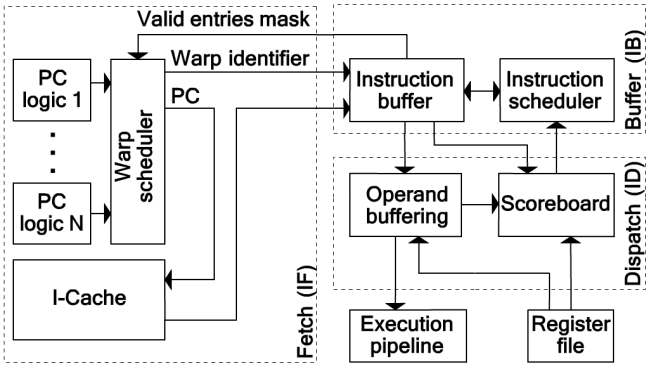

Fig. 1. The microarchitecture of SM's front-end.

al. [21] used a static regression-based model to analyze the performance and power of AMD/ATI GPUs. The analysis shows which units are the most critical to performance and power in GPUs. They found that the fetch engine accounts for 12% of the GPU power and is the 4th among the most important parameters contributing to GPU power. VLIW usage, ALU instructions, and global memory accesses are the other top important parameters. Kasichayanula [11] evaluated two schemes for measuring power in GPUs . In the hardware approach, he used external devices to measure GPU power. In the software approach, NVIDIA NVML library is employed to measure power using GPU performance counters.

3 Observation

3.1 Background

We assume a GPU-like SIMT (Single-Instruction Multiple-Thread) accelerator similar to NVIDIA Tesla. 16 8-wide stream multiprocessors (SM) are connected to six memory controllers through an on-chip crossbar interconnection network. Each SM maintains the context of 1024 concurrent threads. These threads are grouped into coarser independent-schedulable warps. Threads within a warp are executed in an SIMD lock-step manner.

Each SM employs a 24-stage SIMD pipeline for most instructions. In this work, we model the pipeline front-end according to NVIDIA's patents [14, 6, 15]. As shown in Figure 1, SM pipeline front-end consists of three stages: Instruction Fetch (IF), Instruction Buffer (IB), and Instruction Dispatch (ID). IF selects and fetches instructions from concurrent warps. IB buffers the instructions to resolve data dependencies. ID fetches the operands for dependency-free instructions and issues them to the execution pipeline. Below we review each stage in more details.

Instruction Fetch. IF uses per warp logic to determine the next warp instruction PC, followed by fetching the corresponding instruction through I-Cache. The baseline warp scheduler uses the round-robin policy among warps to select the next instruction from the instruction pool.

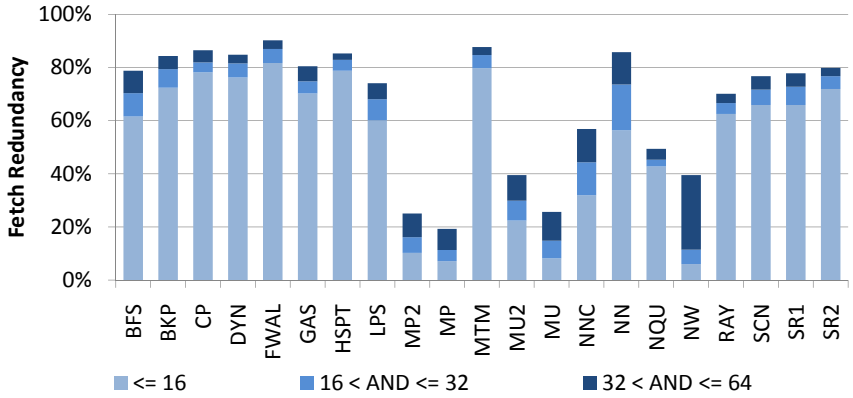

Fig. 2. Fetch redundancy among concurrent warps of SMs during 16, 32, and 64 recent fetches for different benchmarks.

IF resolves control dependences among warp instructions using the PC logic module. This module stalls the warp upon a pending branch. PC logic also determines the next active PC of the warp. IF takes one cycle to select the PC and fetch the corresponding instruction from I-Cache to the IB stage if I-Cache hits.

After an instruction is fetched, the warp scheduler sends it to an empty field in the instruction buffer in the next cycle. This empty field is reserved at fetch time and therefore the fetched instruction always finds an empty instruction buffer entry.

Instruction Buffer. Data dependency is resolved through scoreboarding. Instructions communicate only through the register file and operand forwarding is not supported. Once the scoreboard marks the instruction as ready, the scheduler can select the instruction to proceed. The instruction scheduler used at this stage uses round-robin policy.

Instruction Dispatch. ID buffers collect register operands from highly banked register files [15] and issue instructions to the pipeline's back-end as soon as all operands are buffered.

We assume a scoreboard structure similar to [6]. Each scoreboard entry is associated with a different warp and points to the registers having a pending value. This information is maintained at the register region granularity. Each register region is identified using a base (register ID) and an offset (number of register after the base). Each scoreboard entry can keep track of 6 register regions. Scoreboard stalls an instruction due to a RAW/WAR dependency if the input/output operands of the instruction belong to a register region in the warp's scoreboard entry. Note that a warp is stalled if all of its register regions are taken in the scoreboard. Therefore increasing the number of register regions reduces the probability of warp stalls. Our evaluation shows that six register regions per warp are enough to achieve maximum ILP under the evaluated workloads and the employed in-order pipeline. Once an instruction commits, the register region is released.

3.2 ITL

ITL stems from the fact that a few static instructions account for a significant portion of instructions fetched and decoded among different warps during short intervals and within the same warp scheduler. SMs often execute concurrent warps from the same kernel code. Moreover, the scheduler keeps the warps at the same pace to improve cache locality [13], further increasing ITL.

To provide better understanding we measure and report fetch redundancy as an indication of ITL. Fetch redundancy reports the percentage of instructions already fetched by other currently active warps recently. We measure fetch redundancy in different recency windows (16, 32, and 64). We present our findings in Figure 2.

Average fetch redundancy is 53%, 59%, and 67%for recency window sizes of 16, 32, and 64, respectively. Highly parallel benchmarks, which employ a high number of blocks per grid with a few branch divergences (e.g., CP, HSPT, and NN), show higher fetch redundancy. Other benchmarks, which have fewer concurrent warps (e.g., GAS, SR2, and NW), or more diverging branches (e.g., MP2, MP, MU2, MU, NQU), exhibit less fetch redundancy.

3.3 Exploiting ITL

In this section, we briefly go over possible power and performance benefits achievable using ITL in GPUs.

Fetch/decode bypassing. Loop buffering stores the decoded word of loop instructions in a dedicated buffer and skips fetch/decode as long as the thread proceeds inside the loop [9]. The challenge in CPUs is to find loop boundaries effectively and fitting the entire loop in the buffer. GPUs can take advantage of bypassing by storing the most recent decoded warp instructions and reusing them later by other warps.

Reducing the size of instruction buffer. We have observed that for ~42% of instruction fetches, the corresponding decoded instruction already exists in the instruction buffer (in the entries dedicated to other warps). The instruction fetch/decode process can be bypassed by reading the decoded word from the instruction buffer. A more efficient alternative is to share similar entries among warps to reduce instruction buffer size.

Reducing accesses to I-Cache. ITL can be used to filter accesses to I-Cache. This can be done by using a filter cache or row buffer [12]. Filter cache uses temporal locality to reduce cache miss rate by storing the most recent fetched instructions. Row buffer stores the last accessed I-Cache block (row) and serves fetch requests for blocks residing in the buffer. In the next section we evaluate energy saving opportunities provided by filter cache.

4 Using ITL Case Study in GPUs: Filter Cache

Exploiting filter caches (FC) in current GPUs requires minor modifications to the baseline pipeline front-end (described in section 3.1). FC is probed using the next program counter (PC) to fetch. If the instruction look-up hits during FC tag check, the

Table 1. Front-end area, leakage power, read/write energy, and access delay measured by CACTI [16].

	Area (μm^2)	Leakage (mW)	Energy per R/W (pJ)	Delay (ps)
I-Cache tag	229	0.03	0.13	115.94
I-Cache data	18204	1.78	4.30	221.20
Instruction Buf.	2600	0.16	1.00	137.59
Scoreboard	6921	0.24	1.57	162.17
Operand Buf.	24173	0.53	4.16	174.05
FC tag (32-entry)	266	0.03	0.14	117.28
FC data (32-entry)	2229	0.11	0.81	161.76
FC tag (16-entry)	155	0.02	0.10	105.47
FC data (16-entry)	1337	0.05	0.57	143.38

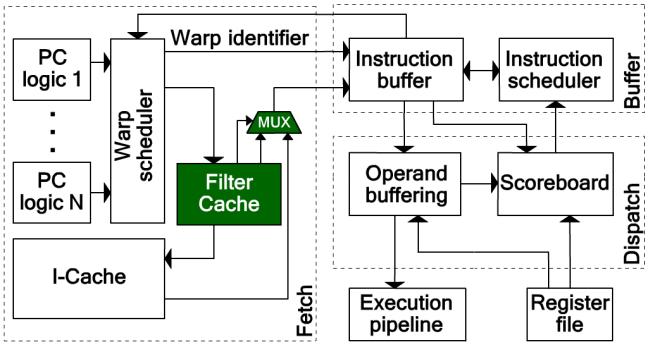

Fig. 3. Modifications made to the instruction fetch stage to implement an FC-enhanced SM.

following I-Cache access is prevented, else the instruction is fetched through the conventional path from I-Cache. Missing instructions update FC later.

Microarchitecture. Figure 3 shows an FC-enhanced front-end design. The modifications are highlighted over the baseline pipeline microarchitecture. FC is a cache-like structure consisting of two parts, i.e., FC check and FC fetch. FC check is similar to a tag array and compares the incoming PC tags against earlier stored tags. Upon a match, FC forwards the bypass signal to the fetch circuit and prevents instruction fetching. Bypass signal activates FC fetch and sends the instructions associated with the matched PC to the next stage through the multiplexer. Upon a mismatch, the front-end follows the conventional approach.

Timing. Using an FC imposes two delays: FC tag check and multiplexer. Upon FC check hit, fetch stage faces the following delay:

$$\text{(Warp scheduling)} + \text{(FC check)} + \text{(FC fetch)} + \text{(MUX)}$$

Upon FC mismatch, the delay is:

$$\text{(Warp scheduling)} + \text{(FC check)} + \text{(I-Cache)} + \text{(MUX)}$$

Table 2. Benchmark characteristics. CTA/SM indicates the maximum number of concurrent blocks per SM which is limited by both parallelism and occupancy.

Abbr.	Name and Suite	Grid Size	Block Size	#Insn	CTA/SM
BFS	BFS Graph [2]	16x(8)	16x(512)	1.4M	1
BKP	Back Propagation [2]	2x(1,64)	2x(16,16)	2.9M	4
CP	Coulumb Poten. [19]	(8,32)	(16,8)	113M	8
DYN	Dyn_Proc [2]	13x(35)	13x(256)	64M	4
FWAL	Fast Wal. Trans. [18]	6x(32) 3x(16) (128)	7x(256) 3x(512)	11M	2, 4
GAS	Gaussian Elimin. [2]	48x(3,3)	48x(16,16)	9M	1
HSPT	Hotspot [2]	(43,43)	(16,16)	76M	2
LPS	Laplace 3D [1]	(4,25)	(32,4)	81M	6
MP2	MUMmer-GPU++ [8] big	(196)	(256)	139M	2
MP	MUMmer-GPU++ [8] small	(1)	(256)	0.3M	1
MTM	Matrix Multiply [18]	(5,8)	(16,16)	2.4M	4
MU2	MUMmer-GPU [2] big	(196)	(256)	75M	4
MU	MUMmer-GPU [2] small	(1)	(100)	0.2M	1
NNC	Nearest Neighbor [2]	4x(938)	4x(16)	5.9M	8
NN	Neural Network [1]	(6,28) (25,28) (100,28) (10,28)	(13,13) (5,5) 2x(1)	68M	5, 8
NQU	N-Queen [1]	(256)	(96)	1.2M	1
NW	Needleman-Wun. [2]	2x(1) … 2x(31) (32)	63x(16)	12M	2
RAY	Ray Tracing [1]	(16,32)	(16,8)	64M	3
SCN	Scan [18]	(64)	(256)	3.6M	4
SR1	Speckle Reducing [2] big	4x(8,8)	4x(16,16)	9.5M	2, 3
SR2	Speckle Reducing [2] small	4x(4,4)	4x(16,16)	2.4M	1

Our study shows that MUX delay is negligible compared to the rest and can be ignored. Table 1 reports the access latency of FC tag, FC data, I-Cache tag, and I-Cache data. IF delay for FC check hit/miss scenario is 0.28ns/0.45ns plus warp scheduling delay. The evaluated front-end runs under 1.3 GHz (0.77ns clock period). For warp scheduling delays below 0.32 ns (0.77 ns - 0.45 ns), FC-enhanced SM does not impose extra cycles. Under the pessimistic scenario where warp scheduling's delay exceeds 0.32 ns, IF should be pipelined into two stages. Under such circumstances, warp scheduling and FC check are done at the first stage. The second stage decides whether to fetch the instruction from FC or I-Cache. This design extends the pipeline depth by a cycle compared to the baseline resulting in a performance loss less than 1%.

Hardware Overhead. FC-enhanced SM requires two auxiliary structures. First, a multiplexer is used to select between two 64-bit instruction words. Second, additional storage is needed for the FC module to temporarily store a small number of instructions. Our simulations show that a 32-entry FC captures a significant share of

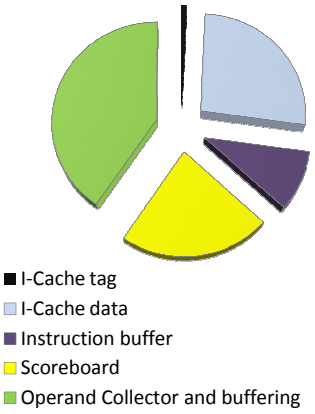

Fig. 4. SM's front-end energy breakdown.

Table 3. Baseline configurations for GPGPU-sim.

NoC	
#SMs : #Memory Ctrls	16 : 6
#SM Sharing a Network Interface	2
Clocking	
Core	1300 MHz
Interconnect	650 MHz
DRAM	800 MHz
Memory	
#Banks Per Memory Ctrls	8
DRAM Scheduling Policy	FCFS
SM	
Warp size : SIMD width	32 : 8
Thread/SM	1024
Register file : Shared memory	64KB : 16KB

fetch redundancy. The 32-entry FC imposes 4.7% area overhead compared to a conventional pipeline front-end.

5 Experiments

5.1 Methodology

We used GPGPU-sim v2.1.1b [1] to model the baseline architecture described in Section 3. We configured GPGPU-sim with the parameters shown in Table 3. We have extended GPGPU-sim to model the discussed FC-enhanced pipeline front-end. We extended the simulator to model 4KB 4-way 4-set [20] I-Cache per SM (32 8-byte instructions per line). On a cache miss, the associated warp is stalled for 300 cycles to access the cache block in global memory. Requests from different warps are merged through I-Cache per warp MSHRs. We used benchmarks from Rodinia benchmark suite [2], CUDA SDK 2.3 [18], Parboil [19], and the benchmarks distributed with GPGPU-sim. We also included the MUMmerGPU++ [8] third-party sequence alignment program. Table 3 shows benchmarks' characteristics.

We report both static and dynamic power. We use CACTI 6.5 [16] to estimate the power dissipation, area, and latency of an FC-enhanced SM compared to the baseline under 32nm technology. For small sized modules, like the operand collector, we scaled the number linearly to extrapolate the parameters. We extracted 6 samples of larger caches (keeping I/O bits, associativity, and other parameters the same) from CACTI to find the line's parameters.

5.2 Experimental Results

In this section, we first report the energy breakdown of the baseline architecture. Then we report the percentage of I-Cache accesses filtered by FC and the associated

Table 4. FC energy saving compared to the baseline as measured by CACTI [16]

	FC hit rate	Baseline I-Cache energy (nJ)	I-Cache + FC energy (nJ)	Front-end energy-saving using FC
BFS	97%	644.11	270.93	12%
BKP	96%	500.14	216.86	9%
CP	100%	16532.20	6616.15	7%
DYN	93%	9882.86	4435.14	8%
FWAL	96%	1730.16	740.77	8%
GAS	87%	1218.70	592.08	7%
HSPT	89%	12076.70	5676.64	8%
LPS	83%	14955.05	7625.97	8%
MP2	33%	67872.12	57354.82	2%
MP	30%	161.40	139.78	2%
MTM	95%	347.37	153.66	8%
MU2	49%	43099.85	31846.11	5%
MU	39%	100.32	81.10	4%
NNC	76%	1718.39	963.81	10%
NN	99%	132820.86	53780.92	19%
NQU	50%	217.81	159.97	5%
NW	70%	5627.29	3356.63	10%
RAY	76%	10520.69	5945.77	6%
SCN	97%	562.47	240.70	9%
SR1	87%	1430.76	699.11	8%
SR2	84%	368.94	189.63	7%

energy savings. Finally, we evaluate filter caches under various microarchitectural changes.

Energy Breakdown. Figure 4 presents the energy breakdown for the SM pipeline front-end for the evaluated workloads. The operand collector and the associated buffering are the most energy consuming parts accounting for 40% of dynamic power. I-Cache, which is the target of this case study, is second, accounting for 27%.

FC Hit Rate. Table 4 reports FC hit rate, which is equal to percentage of the I-Cache accesses filtered. FC check hit rate reaches a maximum of ~100%. Hit rate is above 60% for coherent control-flow compute-intensive workloads like CP, DYN, HSPT, LPS, and MTM. FC shows lower hit rate in control-flow intensive workloads with high branch divergence (e.g., MU2, and MP2). This is due to the fact that these benchmarks exhibit lower ITL as warps often follow different diverging paths. Hit rate is also low for workloads with limited warp-level parallelism (MU, NW, MP, SR2, and NQU). Lower number of concurrent warps reduces the chance of instruction reuse and consequently FC hit rate.

Energy Saving. Table 1 reports die area, leakage power, read/write energy per access, and access latency for modules with significant energy contribution in the pipeline front-end. We assume two ports per I-Cache and FC caches, one read port and one write port. Using a 32-entry FC cache, front-end static power increases by 5.0%. Assuming 16KB register file (7.46 mW leakage), 16KB shared data cache (5.80

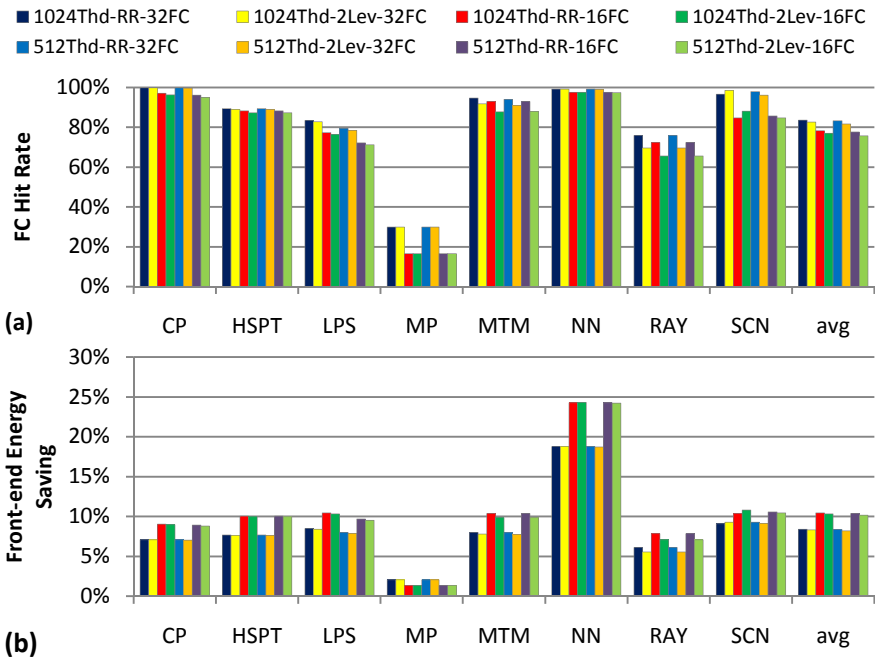

Fig. 5. Sensitivity for a) FC hit rate and b) front-end energy reduction to the multithreading depth (1024 and 512), warp scheduling policy (RR for round-robin and 2Lev for two-level scheduler), and FC size (32 and 16) for different benchmarks.

mW leakage), 5KB texture cache (1.93 mW leakage), and 2KB L1 constant cache (0.82 mW leakage) per SM, FC imposes less than 0.7% leakage power per SM.

In Table 4, we report the dynamic energy consumption of the baseline I-Cache compared to the FC-enhanced design (I-Cache + FC). As reported, FC can reduce I-Cache energy from 13% (MP) to 60% (CP). This translates to from 2% (MP benchmark) to 19% (NN benchmark) of the overall pipeline front-end dynamic energy reduction. Assuming an 18% overall energy share for the front-end [10], an FC-enhanced GPU saves up to 3.4% of the dynamic energy.

Sensitivity Analysis. In this section we report FC hit rate and energy reduction under variations in multithreading depth, FC size, and warp scheduler. We evaluate filter caches for 512 threads per SM, 16-entry FC, and a two-level warp scheduler [17] as an alternative to our baseline 32-entry FC, 1024 threads per SM, and round-robin warp scheduler.

Employing the two-level scheduler improves memory latency hiding. Two-level scheduler divides the warps into multiple fetch groups and gives the highest priority to the warps belonging to the fetch group of the last issued warp. This mechanism can hide the memory accesses made in one fetch group using the computations of other fetch groups. We choose 8 warps per fetch group to achieve maximum latency hiding.

Figure 5 reports FC hit rate and front-end energy reduction for different combinations: multithreading depth (1024, and 512), FC sizes (32, and 16), and warp

schedulers (RR, and 2Lev). In the interest of space we report average and a few representative benchmarks.

Multithreading Depth. Reducing multithreading depth is expected to reduce ITL. CP, LPS, and SCN are among the benchmarks that have enough parallelism to run more than 512 threads per SM. As reported, reducing multithreading depth to 512 threads reduces FC hit rate up to 6% (in LPS) compared to 1024 threads per SM but has minor impact on energy reduction.

Warp Scheduler. We expect to see lower FC hit rate under 2Lev compared to RR under fixed FC size and multithreading depth. This is because 2Lev keeps the warps of different fetch groups at different paces reducing the ITL. As reported, 2Lev often impacts FC hit rate but insignificantly.

FC Size. Among the parameters studied here, FC size has the highest impact. Lower FC size reduces FC hit rate. However, smaller FC comes with lower energy overhead. As reported, a 16-entry FC has a lower FC hit rate (on average of 9% to 10% percent) compared to a 32-entry. However, since the 16-entry consumes less energy, it still shows higher energy reduction.

6 Conclusion

In this study we showed that there is high temporal instruction locality in GPU microarchitectures for general-purpose computations. Concurrent warps fetch and decode the same instruction frequently providing an opportunity to design a more efficient pipeline front-end. We suggested different possibilities to exploit this locality to improve performance, energy, and area. We investigated filter cache as a case study. We found that a simple direct-map filter cache per SM can eliminate 30% to ~100% of I-Cache requests reducing pipeline front-end energy up to 19%. We have evaluated our results under various microarchitectural changes including multithreading depth, warp scheduling, and filter cache size.

7 Acknowledgement

The authors like to thank anonymous reviewers. This work was partially supported by School of Computer Science at Institute for Research in Fundamental Sciences (IPM).

References

1. A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, T.M. Aamodt. “Analyzing CUDA workloads using a detailed GPU simulator.” In Proc. of ISPASS 2009. 163 - 174.
2. S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, L. Sang-Ha, K. Skadron. "Rodinia: A benchmark suite for heterogeneous computing." In Proc. of IEEE International Symposium on Workload Characterization (IISWC). 2009. 44 – 54.
3. S. Collagne. “Exploiting all forms of parallel locality in many-thread architectures.” ALF research group seminar, IRISA. Rennes, December 21, 2011.

4. S. Collange, D. Defour, A. Tisserand. “Power Consumption of GPUs from a Software Perspective.” Proc. of the 9th International Conference on Computational Science (ICCS) 2009. pp 914-923.
5. S. Collange, D. Defour, Y. Zhang. "Dynamic detection of uniform and affine vectors in GPGPU computations." In Proc. of Euro-Par 2009. 46-55.
6. B. W. Coon, P. C. Mills, S. F. Oberman, M. Y. Siu. “Tracking register usage during multithreaded processing using a scoreboard.” United States Patent, Patent number: 7434032.
7. M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, K. Skadron. “Energy-efficient mechanisms for managing thread context in throughput processors.” In Proc. of the 38th annual international symposium on Computer architecture (ISCA). 2011. 235-246.
8. A. Gharaibeh, M. Ripeanu. “Size Matters: Space/Time Tradeoffs to Improve GPGPU Applications Performance.” In Proc. of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. 2010. 1-12.
9. M. Hiraki, R. S. Bajwa, H. Kojima, D. J. Gorny, K. Nitta, A. Shri. “Stage-skip pipeline: a low power processor architecture using a decoded instruction buffer.” International Symposium on Low Power Electronics and Design. 1996. 353-358.
10. S. Hong, H. Kim. “An Integrated GPU Power and Performance Model." In Proc. of ISCA 2010. 280-289.
11. K. K. Kasichayanula. Power Aware Computing on GPUs. Master Thesis Dissertation, University of Tennessee, Knoxville, May 2012.
12. J. Kin, M. Gupta, W. H. Mangione-Smith. "The filter cache: an energy efficient memory structure." In Proc. of MICRO 1997. 184-193.
13. J. E. Lindholm, B. W. Coon, J. Wierzbicki, R. J. Stoll, S. F. Oberman. Credit-Based Streaming Multiprocessor Warp Scheduling. United States Patent, application number: 12/885,299
14. J. E. Lindholm, B. W. Coon, S. S. Moy. Across-thread out-of-order instruction dispatch in a multithreaded microprocessor. *United States Patent, Patent number: 7676657.*
15. S. Liu, J. E. Lindholm, M. Y. Siu, B. W. Coon, S. F. Oberman. Operand collector architecture. United States Patent, Patent number: 7834881.
16. N. Muralimanohar, R. Balasubramonian, N. Jouppi. “Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0.” In Proc. of MICRO 2007. 3-14.
17. V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, Y. N. Patt. "Improving GPU performance via large warps and two-level warp scheduling." In Proc. Of MICRO 2011. 308-317
18. NVIDIA Corp. NVIDIA CUDA SDK 2.3.
19. J. A. Stratton, C. Rodrigues, I. J. Sung, N. Obeid, L. W. Chang, N. Anssari, G. D. Liu, W. M. W. Hwu. "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing." IMPACT Technical Report, 2012.
20. H. Wong, M. M. Papadopoulou, M. Sadooghi-Alvandi, A. Moshovos. "Demystifying GPU microarchitecture through microbenchmarking." In Proc. of ISPASS 2010. 235 - 246.
21. Y. Zhang, Y. Hu, B. Li, L. Peng. Performance and Power Analysis of ATI GPU: A Statistical Approach. 6th IEEE International Conference on Networking, Architecture and Storage (NAS), 2011. pp 149 - 158.