# Towards Green GPUs: Warp Size Impact Analysis

Ahmad Lashgar[1,3], Amirali Baniasadi[2], Ahmad Khonsari[1,3]
[1]School of ECE, University of Tehran
[2]ECE Department, University of Victoria
[3]School of Computer Science, Institute for Research in Fundamental Sciences
a.lashgar@ece.ut.ac.ir, amirali@ece.uvic.ca, ak@ipm.ir

***ABSTRACT*—Selecting the right GPU configuration can impact the overall design in many ways. One of the critical parameters in a GPU is warp size. Smaller warps come with branch divergence reduction while larger warps provide better memory coalescing.**

**In this work we are interested in two possible design choices and their impacts on GPUs: using small warps and investing in finding new solutions to enhance coalescing or using large warps and addressing branch divergence by employing effective control-flow solutions.**

**We also analyze warp size impact on memory coalescing and branch divergence and hence energy. We use our findings to study two machines: a GPU using small warps but equipped with excellent memory coalescing (SW+) and a GPU using large warps but employing an MIMD engine immune from control-flow costs (LW+). We conclude that SW+ provides better energy efficiency when compared to LW+.**

***Keywords*—GPU architecture, Warp size, SIMD efficiency, Branch divergence, Memory divergence, Energy efficiency.**

## I. INTRODUCTION

Conventional single-instruction multiple-threads (SIMT) accelerators execute thousands of threads simultaneously. In order to achieve high throughput with low cost, neighbor threads are bundled in warps and executed in lock-step. Operating at warp-level granularity keeps many threads at the same pace facilitating using common control-flow and memory access patterns. SIMD units are better utilized as a result of executing warps built using threads with the same PC. In addition, warping comes with the advantage of coalescing memory accesses of neighbor threads and reduces the number of off-core requests.

GPUs often do not reach their potential peak performance as the result of two challenges: branch and memory divergence [4, 10]. Both issues result in threads suffering from unnecessary waiting periods. This waiting harms performance and energy inefficiency as it leaves the core idle in the absence of ready threads.

Warp size is one of the parameters that impacts energy efficiency. Small warps, i.e., warps as wide as SIMD width, come with less frequent branch divergence occurrence. Reducing branch divergence frequency increases the number of active lanes hence improving SIMD efficiency. On the other hand, using a small size warp affects memory coalescing, which increases the number of memory stalls. This can result in redundant and energy consuming memory transaction. Using large warps leads to exploiting memory access localities among threads within the same warp and coalescing them to a few off-core requests. Bigger warp size can increase serialization and the branch divergence impact, which increases the number of idle cycles where static energy is wasted without contributing to performance.

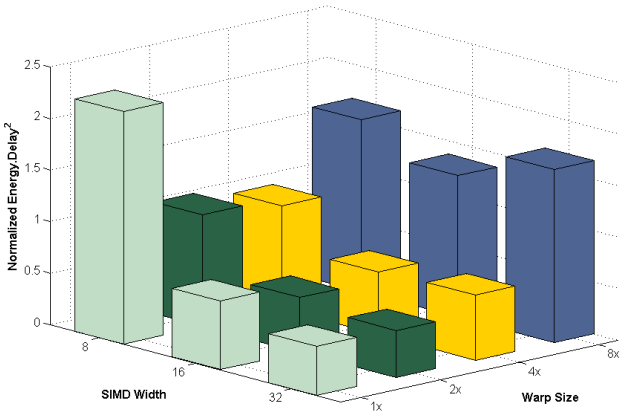



Fig. 1. Warp size impact on energy efficiency (Energy.Delay$^2$) for different SIMD widths, normalized to an 8-wide SIMD and 4x warp size.

Fig. 1 reports average energy efficiency (measured using Energy.Delay$^2$) for the benchmarks used in this study (see Section V for details) for GPUs using different warp sizes and SIMD widths. As reported under a fixed SIMD width, configuring employing warp sizes 2-4X larger than SIMD width achieves best average energy efficiency.

Exploiting large warps reduces the energy cost associated with thread scheduling dramatically simplifying the pipeline front-end logic. Moreover, memory access coalescing gain achieved by employing large warps prevents redundant energy consuming data transfers. On the other hand, large warps may increase the serialization imposed by branch/memory divergence consequently leaving the pipeline idle while on-core resources are leaking energy.

In this paper we extend our previous work [11] and study the impact of warp size on energy efficiency in GPUs. We begin with investigating GPUs with different warp sizes. We then study the effectiveness of two alternative solutions. The

**978-1-4799-0623-9/13/$31.00 ©2013 IEEE**

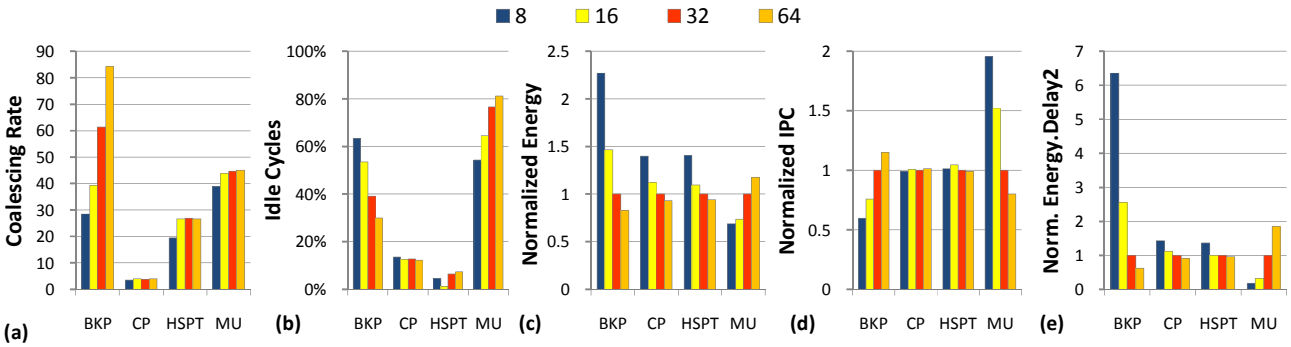



Fig. 2. (a) Coalescing rate, (b) Idle cycle share, (c) Energy, (d) Performance, and (e) Energy.Delay$^2$ under different warp sizes. Energy and IPC are normalized to a GPU using 32 threads per warp. We report part of the results here (complete results are presented in Section VI).

first solution uses large warps to reduce off-chip memory accesses by enhancing memory coalescing. At the same time, we use effective control-flow solutions to address the resulting increase in branch divergence. The second solution uses small warps and reduces the number of idle cycles by minimizing branch serialization. This approach takes extra steps to address memory stalls.

In this work we evaluate both approaches and estimate the resulting energy efficiency of both solutions. We show that starting with a small warp size, and then using dynamic memory divergence solutions is a better choice.

In summary we make the following contributions:

- We study the impact of warp size on different parameters including memory stalls, idle cycles, and energy.
- We introduce an effective approach to enhance energy efficiency in GPUs. We show that the using a static and simple approach to deal with branch divergence (using small warps) along with dynamic memory stall reductions solutions is an effective solution.
- We also investigate the alternative and show that using a static solution to enhance coalescing (i.e., using large warps) along with an ideal dynamic control-flow solution is outperformed by the first approach due to frequent synchronizations of a large number of threads.

The rest of the paper is organized as follows. In Section II we study background. In Section III we review warp size impact. In Section IV we present our machine models. In Section V we discuss methodology. Section VI reports experimental results. In Section VII we review related work. Finally, Section VIII offers concluding remarks.

## II. Background

In this study we focus on SIMT accelerators similar to NVIDIA Tesla architecture [14]. Stream Multiprocessors (SMs) are deeply multi-threaded processors sharing private non-coherent L1 caches among threads. On chip interconnection network (crossbar network in this study) is responsible for routing SM off-chip requests to corresponding memory controllers (MC) and delivering the MC respond to the SM.

Each SM keeps context (including register and shared memory) for 1024 threads. An SM has one thread scheduler which groups and issues warps on one SIMD group. Threads within a warp have one common program counter while control-flow divergence among threads is managed using re-convergence stack [4]. Diverged threads are executed serially until re-converging at the immediate post-dominator of branch. Re-convergence point is embedded in the binary and is extracted by the architecture during branch execution.

Instructions from different warps are issued back-to-back into the deep 24-stage, 8-wide SIMD pipeline. If the warp pool has no ready warp, the pipeline front-end stays idle leading to back-end underutilization and leakage energy loss. Under such circumstances, all the warps are issued into the pipeline. However, there are ready threads that are inactive/waiting due to branch/memory divergence [16]. Inactive threads are those ready to execute different diverging paths while waiting threads are those waiting at the re-convergence point to get synchronized with other threads of the warp.

The global memory accesses of neighbor threads are coalesced to perform scatter/gather operations efficiently. We model a coalescing behavior similar to compute compatibility 2.0 [19]. Requests from neighbor threads accessing the same cache line are merged into one request. This can enhance energy efficiency significantly as it reduces the number of energy consuming memory accesses. Neighbor threads are aggregated over the entire warp. Consequently, memory accesses of a warp are coalesced into one or many cache line accesses. Each line is 64 bytes. Memory transaction granularity is the same as cache line size, which is one stride.

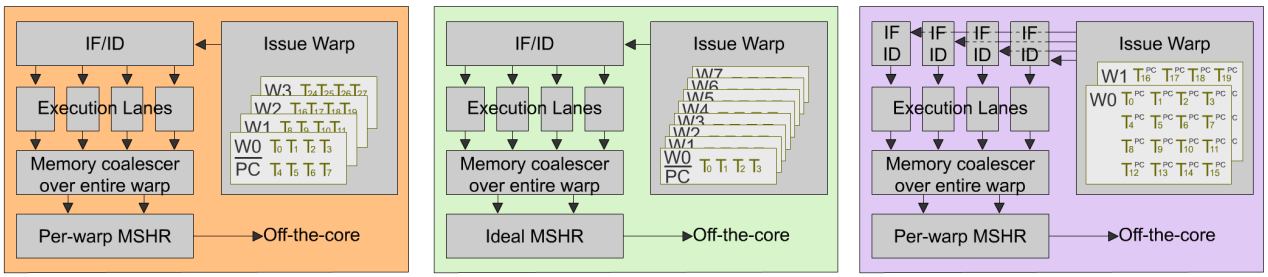
## III. Warp Size Impact

In this section we report how the impact of warp size on memory access coalescing, idle cycles, and energy efficiency. See Section V for methodology.

**Memory access coalescing.** Memory accesses made by threads within a warp are coalesced into fewer memory transactions to reduce bandwidth demand. We measure memory access coalescing using the following equation:

$$Coalescing\ rate = \frac{Total\ memory\ insn.}{Total\ off chip\ requests} \quad (1)$$

Fig. 2(a) reports coalescing under different warp sizes. As reported, increasing the warp size enhances coalescing.




(a) Baseline microarchitecture (b) SW+ microarchitecture (c) LW+ microarchitecture.

Fig. 3. Machine models compared in this study.

Larger warps increase the likeliness of memory accesses made to the same cache line residing in the same warp hence reducing the number of memory accesses. This effect starts to diminish for warp sizes beyond 32 threads for most benchmarks as coalescing width (16 words of 32-bit) becomes saturated. Accordingly, increasing warp size beyond a threshold, returns little coalescing gain.

**Idle cycles.** Idle cycles are cycles during which the scheduler lacks ready warps in the pool. Small warps improve latency hiding as they reduce thread synchronization. However, they can increase memory latency by reducing coalescing rate. Fig. 2(b) reports idle cycle frequency for GPUs using different warp sizes. Core idle cycles can be the result of branch/memory divergences which inactivate otherwise ready threads [16]. Small warps address divergence by hiding idle cycles (e.g. MU). For some benchmarks (e.g., BKP), small warps lose many coalescable memory accesses increasing memory pressure. This pressure increases average core idle periods compared to larger warps (e.g. BKP). We conclude that the effect of warp size on idle cycles depends on the wasted coalescing and obtained latency hiding.

**Energy.** Warp size variations impact both static and dynamic energy (assuming fixed threads per SM). Specifically, warp size variations impact the sizes of warp scheduler, instruction buffer, and scoreboard unit and therefore static power in an SM. Warp size variations impact the dynamic power of an SM by changing memory, warp scheduler, fetch, and decode units access rates. Fig. 2(c) reports energy consumption under different warp sizes. In most cases, GPUs using large warps are less energy consuming compared to those using small warps. Under large warps, energy can be saved if the reduction in the energy invested in warp scheduling and memory accesses exceeds the leakage increase imposed by branch divergence serialization (e.g. BKP, CP, HSPT). Energy consumption can increase if the leakage overhead associated with large warps outweighs warp scheduling and memory access reduction gains (e.g. MU).

**Performance.** Using larger warp sizes may have opposing impacts on performance. Performance can improve if the enhanced memory access coalescing outweighs the synchronization overhead imposed by large warps. Performance can suffer if the synchronization overhead associated with large warps is higher than the coalescing memory access gain. Fig. 2(d) reports performance for different warp sizes. As reported, in most workloads presented here, warp size has significant impact on performance.

**Energy.Delay$^2$.** Fig. 2(e) reports Energy.Delay$^2$ as an indication of the overall energy efficiency. The numbers follow the energy report in 2(c) where large warps provide lower Energy.Delay$^2$ in most cases compared to small warps.

Considering the branch/memory divergence challenges and memory access coalescing gains associated with warping, there are two approaches to enhancing the energy efficiency of future GPUs. Possible solutions to enhance energy efficiency are starting with small or large warp sizes and then invest in compensating the negative aspects with aggressive solutions. In the remainder of this work, we study two machine models and investigate them further.

## IV. MACHINE MODELS

In our previous work [11], we introduced two machine models to investigate warp size impact on performance. Fig. 3 presents the machines and the baseline. The first model represents a coalescing-enhanced small warp machine (SW+). SW+ uses small warps but is equipped with ideal coalescing. Intuitively we study SW+ to measure the energy efficiency potential in building small warp machines. The second machine is a control-flow-enhanced large warp machine (LW+). We use LW+ to estimate the energy efficiency improvement possible for a processor using a large warp size but equipped with an ideal control-flow solution. In this section, we discuss the energy aspect of these machines.

### A. SW+

SW+ suffers from the same energy-consumption disadvantages as the small warps machine (higher the number of schedulable elements). Moreover, ideal coalescing hardware module in SW+ consumes additional energy compared to small warps. Energy overhead of this machine is mainly due to higher number of accesses to pipeline front-end; warp scheduler, fetch, and decode stages. We take into account these overheads in our evaluations.

The motivation behind investigating SW+ is to study if investing in a small warp size machine to reduce memory access energy consumption leads to an overall energy reduction.

TABLE 1. BENCHMARKS CHARACTERISTICS.

| Name | Grid Size | Block Size | #Insn |
|---|---|---|---|
| **BFS:** BFS Graph [3] | 16x(8,1,1) | 16x(512,1) | 1.4M |
| **BKP:** Back Propagation [3] | 2x(1,64,1) | 2x(16,16) | 2.9M |
| **CP:** Distance-Cutoff Coulomb Potential [1] | (8,32,1) | (16,8,1) | 113M |
| **GAS:** Gaussian Elimination [3] | 48x(3,3,1) | 48x(16,16) | 8.8M |
| **HSPT:** Hotspot [3] | (43,43,1) | (16,16,1) | 76.2M |
| **LPS:** Laplace equation on regular 3D grid [1] | (4,25) | (32,4) | 81.7M |
| **MP:** MUMmer-GPU++ [6] | (1,1,1) | (256,1,1) | 0.3M |
| **MU:** MUMmer-GPU [1] | (1,1,1) | (100,1,1) | 0.15M |
| **NN:** Neural Network [1] | (6,28)<br>(50,28)<br>(100,28)<br>(10,28) | (13,13)<br>(5,5)<br>2x(1,1) | 68.1M |
| **NNC:** Nearest Neighbor [3] | 4x(938,1,1) | 4x(16,1,1) | 5.9M |
| **NQU:** N-Queen [1] | (256,1,1) | (96,1,1) | 1.2M |
| **RAY:** Ray-tracing [1] | (16,32) | (16,8) | 64.9M |
| **SC:** Scan[18] | (64,1,1) | (256,1,1) | 3.6M |
| **SR1:** SRAD [3] (large dataset) | 3x(8,8,1) | 3x(16,16) | 9.1M |
| **SR2:** SRAD [3] (small dataset) | 4x(4,4,1) | 4x(16,16) | 2.4M |

### B. LW+

LW+ exploits the same warp-scheduling energy efficiency advantages of large warps. However, the MIMD structure behind the LW+ demands multiple concurrent fetch/decodes per warp. Accordingly, fetch/decode is replicated for each lane increasing the leakage power. In our evaluations we assume one fetch/decode unit per lane and report the associated energy overhead.

Previous studies [2, 4, 5, 16, and 17] have suggested solutions to reduce the impact of branch divergence. LW+ is a many-instruction multiple-threads (MIMT) architecture, which is an aggressive extension of dual-instruction multiple threads (DIMT) architecture exploited by SBI [2] architecture.

## V. METHODOLOGY

### A. PERFORMANCE EVALUATION

We modified GPGPU-sim [1] (version 2.1.1b) to support warps larger than 32-thread and model memory coalescing similar to compute compatibility 2.0 devices [19]. We configured GPGPU-sim according to parameters shown in Table 2. 16 SMs provide peak throughput of 332.8 GFLOPs. Six 64-bit wide memory partitions provide memory bandwidth of 76.8 Gbytes/s at dual-data rate. We use an 8-wide SIMD configuration.

### B. ENERGY EVALUATION

We report our results in the 32 nm manufacturing technology. We report Energy.Delay$^2$ as an indication of energy efficiency. In order to evaluate the power consumption of GPGPU-like microarchitecture we used a methodology similar to GPU-PowerSim [8] and GPUSimPow [7]. We estimate the power of SMs and MCs using McPat [13] and their interconnection network using Orion 2.0 [9] as follows.

TABLE 2. BASELINE CONFIGURATIONS FOR GPGPU-SIM.

| | |
|---|---|
| **NoC** | |
| #SMs / #memory controllers | 16 / 6 |
| Number of SM Sharing an Network Interface | 2 |
| **SM** | |
| #thread per SM / SIMD width | 1024 / 32 |
| Maximum allowed CTA per SM | 8 |
| Shared Memory/Register File size | 16KB/64KB |
| Warp Size | 8 / 16 / 32 / 64 |
| L1 Data/Texture/Constant cache | 64KB : 16KB : 16KB |
| **Clocking** | |
| Core / Interconnect / DRAM | 1300 / 650 / 800 MHz |
| **Memory** | |
| banks per memory ctrl : DRAM Scheduling Policy | 8 : FCFS |

We retrieve the performance counters and module activities using GPGPU-sim. Subsequently these performance counters and the configurations listed in Table 2 are given to McPat to estimate the static/dynamic power of six MCs and 16 in-order SIMD multithreaded SMs.

We use Orion 2.0 to estimate the power consumption of the interconnection network among SMs and MCs. We assume two crossbars modeling the interconnection network; 1) SMs to MCs, and 2) MCs to SMs.

### C. WORKLOADS

We include benchmarks from GPGPU-sim [1], Rodinia [3], and CUDA SDK 2.3 [18]. We also included MUMmerGPU++ [6] third-party sequence alignment program. Our benchmark set exhibits different behaviors: memory bounded, computational intensive, high/low branch divergence occurrence, and large/small number of concurrent threads per SM. Table 1 lists specifications of these benchmarks.

## VI. RESULTS

In this section, we evaluate SW+, LW+ and processors using different warps sizes. In Section VI.A we present memory access coalescing. Idle cycle is reported in VI.B. In Section VI.C we report performance. Finally, in Section VI.D we report energy and Energy.Delay$^2$.

### A. MEMORY ACCESS COALESCING

Fig. 4 reports the coalescing rate. As reported, SW+ provides the best coalescing rate in most benchmarks. SW+ coalesces memory accesses among all threads of an SM to achieve this. Widening coalescing to merge accesses from all threads can improve coalescing rate by 68% and 40% compared to coalescing width of 32 threads and 64 threads, respectively.

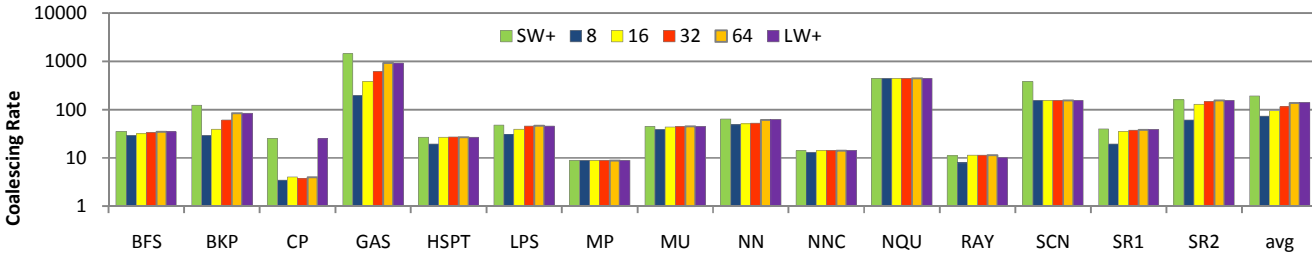



Fig. 4. Coalescing rate for SW+, LW+ and processors using different warp sizes.

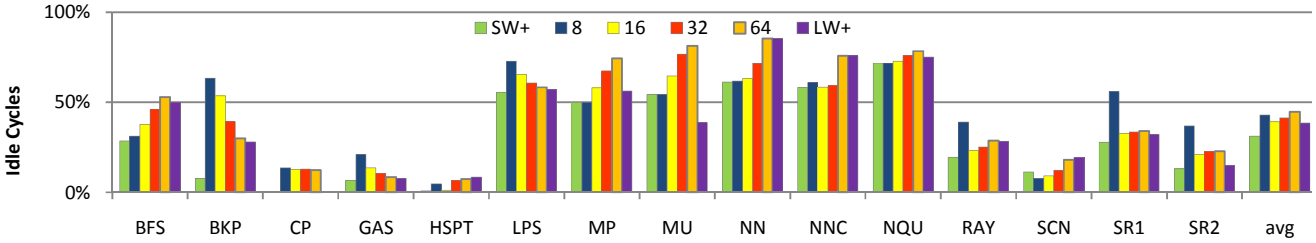



Fig. 5. Idle cycle share for SW+, LW+ and processors using different warp sizes.

On average, 64-thread per warp machine does better than LW+. This is due to the fact that LW+'s MIMT execution does not keep threads at the same pace to coalesce their accesses. In some cases (e.g., MP and MU) splitting the warp upon divergence prevents merging memory requests. Under such circumstances, redundant memory accesses degrade coalescing rate. However, as we show in VI.D, this does not translate to energy loss since the memory subsystem is not under-pressure in these workloads (MU and MP).

### B. Idle Cycles

As discussed in Section II, small warps reduce the number of idle cycles by reducing unnecessary waiting due to branch/memory divergence. This benefits of idle cycle saving is partially lost since small warps lose memory access coalescing, pressuring the memory subsystem. SW+ compensates this drawback by exploiting ideal coalescing. As shown in Fig. 5, SW+ shows lowest idle cycle share in most workloads. On average, using short warps combined with ideal coalescing (SW+), reduces idle cycles by 12%, 8% and 10% compared to processors using 8, 16 and 32 threads per warp, respectively.

Our analysis shows synchronizing a large number of threads at every instruction can significantly increase the number of idle cycles in LW+.

### C. Performance

Fig. 6 reports performance for SW+, LW+ and processors using different warp sizes. SW+ outperforms other machines in most benchmarks. On average, SW+ outperforms LW+, and machines using 8, 16 and 32 threads per warp by 7%, 18%, 15% and 25%, respectively.

LW+ synchronizes all threads in a warp at every instruction. This synchronization is costly for memory instructions with memory divergence. Even MIMD cores cannot compensate this synchronization overhead. Therefore a big part of MIMD's gain is lost due to unnecessary waitings. On average, LW+ outperforms processors using 8, 16, 32 and 64 threads per warp by 11%, 8%, 17% and 30%, respectively.

### D. Energy and Energy.Delay$^2$

Fig. 7 reports energy consumption for SW+, LW+, and different warp sizes. SW+ is the second energy hungry machine after the 8-thread per warp machine. On average, the energy consumption of SW+ and LW+ machine are 30% and 9% higher than the 32-thread per warp machine, respectively.

Fig. 8 reports energy efficiency using the Energy.Delay$^2$ metric. On average, SW+ machine is 62% more energy efficient than LW+ and 136%, 13%, 4%, and 74% more energy efficient than 8-, 16-, 32-, and 64-thread per warp machines.

## VII. Related Works

Mahersi et al. [15] evaluate the impact of warp size on SIMD efficiency under perfect memory (zero latency). They found that lower warp size returns higher SIMD efficiency. Lashgar et al. [12] introduced DWR to resize warp size dynamically to enhance performance. We have extended our previous work in [11] and have evaluated energy-efficiency of GPUs under different warp sizes.

## VIII. Conclusion

Building energy efficient GPUs requires addressing both memory and branch divergence.

Finding the right microarchitectural configuration of a GPU is critical in achieving high energy efficiency. Such static decisions, however, influence the dynamic solutions a system requires to deal with runtime challenges. Choosing the right warp size is one example. Large warp sizes reduce memory access frequency and energy consumption but can increase leakage energy resulting from frequently occurring branch/memory divergences. An alternative approach is to deal with control-flow first by using small warps and then

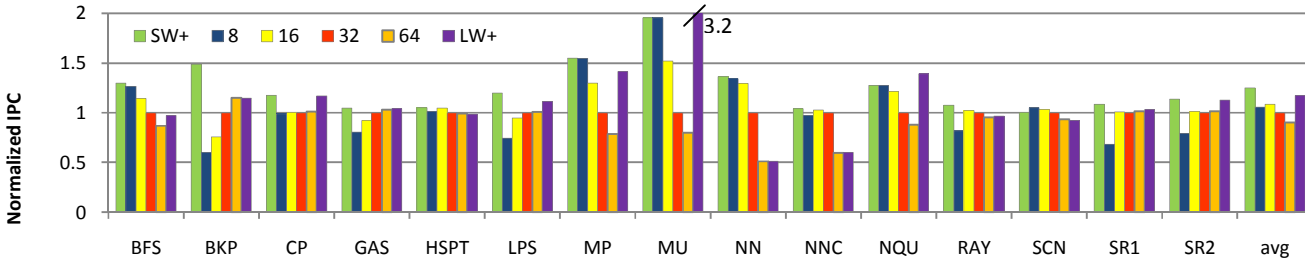



Fig. 6. Performance for SW+ and LW+ and processors using different warp sizes.

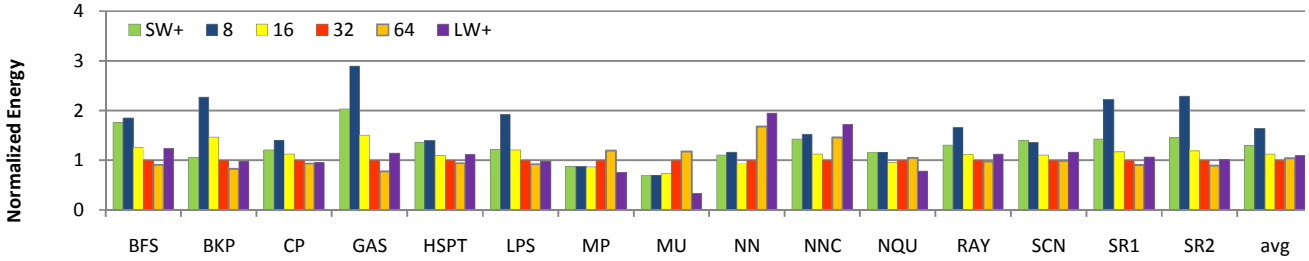



Fig. 7. Energy for SW+ and LW+ and processors using different warp sizes.

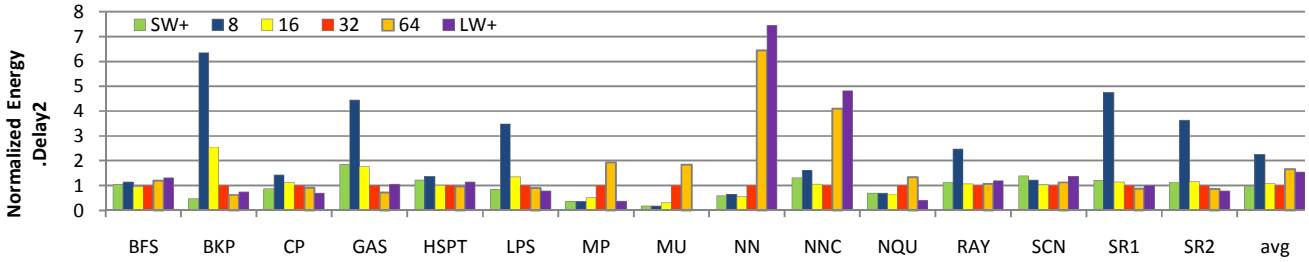



Fig. 8. Energy efficiency for SW+ and LW+ and processors using different warp sizes.

investigating dynamic solutions to address memory coalescing.

We study both approaches and conclude that the latter approach comes with better energy efficiency returns for the benchmarks and configurations used in this work.

## REFERENCES


[1] A. Bakhoda et al. Analyzing CUDA workloads using a detailed GPU simulator. ISPASS 2009.

[2] N. Brunie et al. Simultaneous Branch and Warp Interweaving for Sustained GPU Performance. ISCA 2012.

[3] S. Che, et al. Rodinia: A benchmark suite for heterogeneous computing. IISWC 2009.

[4] W. W. L. Fung et al. Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow. MICRO 2007.

[5] W. W. L. Fung and Tor M. Aamodt. Thread Block Compaction for Efficient SIMT Control Flow. HPCA 2011.

[6] A. Gharaibeh and M. Ripeanu. Size Matters: Space/Time Tradeoffs to Improve GPGPU Applications Performance. SC 2010.

[7] GPUSimPow. Available: http://www.aes.tu-berlin.de/menue/forschung/projekte/gpusimpow_simulator/

[8] GPU-PowerSim. Available: http://www.ideal.ece.ufl.edu/main.php?action=gpu-powersim

[9] A. B. Kahng, B. Li, L. S. Peh, K. Samadi. ORION 2.0: A power-area simulator for interconnection networks. IEEE TVLSI Systems, Volume 20, Issue 1, pp 191-196.

[10] A. Lashgar and A. Baniasadi. Performance in GPU Architectures: Potentials and Distances. 9th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD 2011).

[11] A. Lashgar, et al. Warp Size Impact in GPUs: Large or Small?. GPGPU6, March 16, 2013.

[12] A. Lashgar, et al. Dynamic Warp Resizing: Analysis and Benefits in High-Performance SIMT. ICCD 2012 Poster Session. October 1, 2012.

[13] S. Li, et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. MICRO 2009.

[14] E. Lindholm et al. NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro, March-April 2008, Volume 28, Issue 2, pp 39-55.

[15] A. Mahesri, et al. Tradeoffs in designing accelerator architectures for visual computing. MICRO 2008.

[16] J. Meng et al. Dynamic warp subdivision for integrated branch and memory divergence tolerance. ISCA 2010.

[17] V. Narasiman et al. Improving GPU Performance via Large Warps and Two-Level Warp Scheduling. MICRO 2011.

[18] NVIDIA Corp. CUDA SDK 2.3. Available: http://developer.nvidia.com/cuda-toolkit-23-downloads

[19] NVIDIA Crop. CUDA C Programming Guide. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html