Dynamic Warp Resizing: Analysis and Benefits in High-Performance SIMT

Ahmad Lashgar[1]
a.lashgar@ece.ut.ac.ir

Amirali Baniasadi[2]
amirali@ece.uvic.ca

Ahmad Khonsari[1 3]
ak@ipm.ir

[1]School of ECE
University of Tehran

[2]ECE Department
University of Victoria

[3]School of Computer Science
Institute for Research in
Fundamental Sciences

Abstract—**Modern GPUs synchronize threads grouped in warps. The number of threads included in each warp (or warp size) affects divergence, synchronization overhead, and the efficiency of memory access coalescing. Small warps reduce the performance penalty associated with branch and memory divergence at the expense of a reduction in memory coalescing. Large warps enhance memory coalescing significantly but also increase branch and memory divergence. Dynamic workload behavior, including branch/memory divergence and coalescing, is an important factor in determining the warp size returning best performance. Based on this observation, we propose Dynamic Warp Resizing (DWR). DWR outperforms static warp size decisions, up to 2.28X.**

Keywords- GPU architecture; Performance; Warp size; Memory access coalescing; Branch divergence;

I. INTRODUCTION

GPUs are still far behind their potential peak performance as they face two important challenges: branch and memory divergence [2]. One of the parameters strongly affecting the performance impact of such divergences is the number of threads in a warp or warp size. *Small warps*, i.e., warps as wide as SIMD width, reduce the likelihood of branch/memory divergence occurrence. On the other hand, small warps reduce memory coalescing, which can increase memory stalls. *Large warps* exploit potentially existing memory access localities among neighbor threads and coalesce them to a few off-core requests. On the negative side, large warps can increase serialization and the branch/memory divergence frequency.

In this paper we evaluate the effect of warp size on GPU performance and coalescing rate under general-purpose workloads. Accordingly, we propose *Dynamic Warp Resizing* or *DWR* to achieve performance benefits of both small and large warps. We also propose a realistic hardware implementation for DWR. More details regarding DWR including area overhead, alternative implementations, and evaluation under various microarchitectures (including those with different SIMD width, and L1 cache size) and more benchmarks is reported in our detailed technical report [3].

II. DYNAMIC WARP RESIZING

Overview. DWR is an adaptive microarchitectural solution, which varies warp size according to program behavior. Warp size is initially set to SIMD width (referred to as sub-warp) but can expand upon encountering specific program behaviors. This dynamic increase in warp size increases memory accesses coalescing (often absent from systems using small warps) and relies on using barrier synchronizers to synch and combine multiple sub-warps. DWR extends the ISA to implement this synchronization and warp scheduler to support warp combining.

Synchronization points. DWR groups and issues warps with different sizes; i) large warps for specific instructions, and ii) sub-warps for other instructions. Partner sub-warps are synchronized to build one large warp to execute the specific instructions. Specific instructions include a group of static low-level PTX instructions [4], which we refer to as Large-wArp-inTensive instructions or LATs. Non-LATs are always executed using sub-warps. LATs, on the other hand, are executed using large warps built from multiple sub-warps. We consider load/store instructions from/to global/local/param space as LATs [5]. DWR's warp scheduler combines multiple sub-warps into one large warp upon realizing that all partner sub-warps are ready to execute.

Synchronization realization. To guarantee that all partner sub-warps are ready to execute the associated LAT, we enforce a synchronization barrier just before the LAT. This synchronization can be realized by extending the ISA and hardware to support this inter-partner sub-warp synchronization barrier. Each LAT is transformed to two instructions: 1) LAT inter-partner sub-warp barrier, and 2) original LAT instruction. The new instruction operates similar to intra-thread-block synchronizer. Sub-warp Combiner (SCO) is used to construct large warps upon issuing an LAT. The sub-warp synchronizer sends a signal to SCO to identify sub-warps synchronized on an LAT. Sub-warps stay waiting until the synchronizer marks them as combine-ready. The combine-ready status indicates that all sub-warps have reached the LAT barrier and are ready to be combined and execute the associated LAT. SCO merges active masks of the combine-ready sub-warps, issuing one larger warp.

Avoiding unnecessary synchronization. In case partner sub-warps are diverged into different paths, synchronizing sub-warps is non-beneficial for coalescing as they execute different instructions. We employ a small ignore list table (referred to as ILT) to store program counters which have been detected as non-beneficial LATs. We avoid synchronization at these LATs.

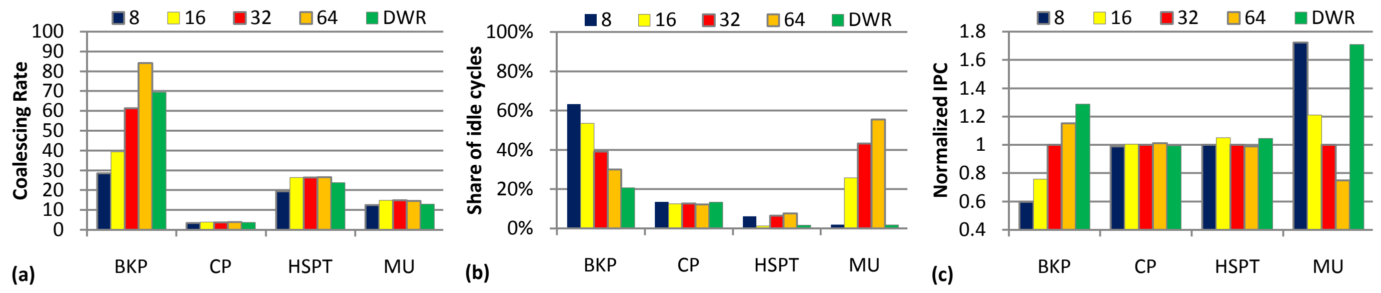

Figure 1. (a) Coalescing rate, (b) Idle cycle share and (c) Performance under different warp sizes and DWR. IPC is normalized to a GPU using 16 threads per warp.

III. Methodology

We modified GPGPU-sim [1] (version 2.1.1b) to model large warps (beyond 32 threads), memory coalescing similar to compute compatibility 2.0 devices [5], and DWR. For each machine (fixed warp size machines or DWR) we model memory access coalescing over the entire warp threads. Each SM is a 24-stage 8-wide pipelined processor exploiting 48KB L1 data cache (64-way, 12-set) and shares 16KB shared memory among 1024 threads. 16K 32-bit registers per SM are reserved for thread context. 16 SMs provide peak throughput of 332.8 GFLOPS. Six 64-bit wide memory partitions provide memory bandwidth of 76.8 GB/s at dual-data rate.

DWR configuration. We assume 32-entry (4-set, 8-way) per ILT per SM in this study. Smallest warp size is as wide as SIMD width and the largest warp size is 64 threads.

Benchmarks. In the interest of space we only report for a representative subset of four benchmarks [3].

IV. Results

In this section we report the number of idle cycles, memory access coalescing, and performance and compare DWR to fixed warp size machines.

Memory access coalescing. We measure memory access coalescing using the following equation:

$$Coalescing\ rate = \frac{Total\ memory\ insn.}{Total\ offchip\ request} \qquad (1)$$

Figure 1a compares coalescing rates for different warp sizes. As presented, increasing warp size improves coalescing rate. This increase starts to diminish for warp sizes beyond 32 for some benchmarks. Fixed 64 threads per warp provides the highest coalescing rate in most benchmarks.

DWR executes most instructions using 8 threads per warps to prevent unnecessary synchronizations. To maintain memory access coalescing of large warps, DWR synchronizes 8 sub-warps upon memory accesses. In benchmarks where memory accesses made by neighbor threads is coalescable, DWR provides far higher coalescing rate compared to an 8-thread per warp machine (e.g., BKP). Under DWR, MU loses coalescing rate considerably compared to fixed large warps. In this benchmark, a considerable portion of LATs is placed in the ILT. This coalescing loss, however, does not degrade performance as we show later.

Idle cycles. Figure 1b reports idle cycle frequency for different warp sizes. Idle cycles are cycles when the scheduler finds no ready warps in the pool. Small warps compensate branch/memory divergence by hiding idle cycles in MU. On the other hand, in BKP, small warps lose many coalescable memory accesses, increasing memory pressure. This pressure increases average core idle durations compared to larger warps in BKP. DWR reduces unnecessary synchronization of entire warp threads and interleaves sub-warps to hide latency. On average DWR reduces idle cycles by 26%, 12%, 17% and 25% compared to processors using fixed 8, 16, 32 and 64 threads per warp, respectively. As we show Figure 1b, DWR has the lowest average idle cycle share.

Performance. Figure 1c reports performance. Performance can improve if an increase in memory access coalescing outweighs synchronization overhead. Performance can suffer if the synchronization overhead associated with building large warps exceeds coalescing memory access gains. Performance improves in BKP with warp size. Performance is lost in MU as warp size increases. HSPT performs best under average warp sizes (16 threads). CP is less sensitive to warp size. In most benchmarks, DWR performs close to the best performing fixed warp size machine. This is due to the fact that DWR combines the benefits of small and large warps.

V. CONCLUSION

In this work we evaluated the performance of Tesla-like GPUs under different warp sizes. Small warps serve well for applications suffering from branch divergence. On the other hand, large warps are more suitable for memory-bounded workloads taking advantage of memory access coalescing. Based on these findings, we proposed DWR as a dynamic and adaptive solution aiming at achieving the benefits associated with both large and small warps. DWR outperforms fixed 8, 16, 32 and 64 threads per warp machine up to 2.16X, 1.7X, 1.71X and 2.28X, respectively.

References

[1] A. Bakhoda et al. Analyzing CUDA workloads using a detailed GPU simulator. IEEE International Symposium on Performance Analysis of Systems and Software, 2009.

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

[3] A. Lashgar, A. Baniasadi, and A. Khonsari. Dynamic Warp Resizing in High-Performance SIMT. arXiv:1208.2374v1

[4] NVIDIA Corp. PTX: Parallel Thread Execution ISA Version 2.3.

[5] NVIDIA Corp. CUDA C Programming Guide. Available: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf