DART: A GPU architecture exploiting temporal SIMD for divergent workloads

Jan Lucas*[1], Sohan Lal*[1],
Mauricio Alvarez Mesa*[1] Ahmed
Elhossini*[1], Ben Juurlink*[1]

* *AES, Technische Universität Berlin, Sekretariat EN 12, Einsteinufer 17, 10587 Berlin, Germany*

ABSTRACT

GPUs can offer very high computational power and high energy efficiency for some applications. Their programming model gives the illusion of independent threads while the GPU internally packs together threads into groups often called warps and executes them on a SIMD unit. This improves power and area efficiency by avoiding the need to fetch, decode and schedule an instruction once per executed operation. Instead fetching, decoding and scheduling of instructions can be done on a per warp level with each scheduled instruction executing in multiple threads. It is normally implemented using spatial SIMD units that execute each SIMD bundle on a fixed number of physical functional units (FU), each supplied with different data, but the same control vector. If not all threads share the same control flow spatial SIMD units can only be utilized partially. In this paper we present a GPU with temporal SIMD (TSIMD), where each functional unit uses its control vector value for multiple cycles to execute a single instruction on a flexible number of data values. TSIMD allows to adjust the group size at runtime and skip unused SIMD slots, thus prevents wasting FU cycles when not all threads within warp are on the same control flow. During the time a TSIMD unit is active, the frontend is free to schedule new instructions to a different TSIMD unit, thus being able to reach the same high peak performance as conventional, spatial SIMD GPUs. In this paper we explain DART, a GPU architecture exploiting TSIMD and the differences to a conventional GPU and show simulation results, for synthetic microbenchmarks, using a modified version of GPGPU-Sim that is able to simulate DART GPUs with TSIMD.

KEYWORDS: Temporal SIMD; GPU; ACACES; poster session

1 Introduction and Related Work

During the last years GPUs have evolved from 3D graphic accelerators to devices able to do general purpose computing at very high throughputs at a high energy and area efficiency. GPUs capable of general purpose GPU computing (GPGPUs) are now become common

[1]E-mail: {j.lucas,sohan.lal,alvarez,elhossini,juurlink}@aes.tu-berlin.de

even in SoCs intended for mobile devices. They are programmed using a single program multiple data (SPMD) programming model. While this programming model gives the programmer the illusion of thousands of independent threads, the GPU internally groups together threads into groups (called warps in NVIDIA termionology) and executes them on a SIMD unit. But at the same time this imposes the requirement of running all threads within one warp in lockstep and executing all branch directions and masking out all threads that do not follow the currently executing direction. During the execution the masked out functional units (FUs) can not be used.

The grouping of threads into warps allows the GPU to fetch instruction once per warp instead of once per thread. This SIMD execution provides high efficiency and high peak performance to GPUs but at the same time it also makes execution of code with divergent branches less efficient. Efficient execution of branch divergent code in GPUs has been an active research area.[BCD12][NSL$^+$11] However these techniques only work well with specific branch patterns. Temporal SIMD is a new and potentially more general technique for more efficient execution of SPMD code. Temporal SIMD was first mentioned by Keckler at al.[KDK$^+$11] without providing any details of the techniques. At roughly the same time NVidia filled a patent[Kra11] on the technique. This patent was recently published by the US patent office and provides a more detailed description of the technique. However, the performance characteristics and architectural changes required for TSIMD have not been analysed in detail. This paper describes DART, a GPU architecture build around TSIMD execution units and enables a first look at the performance characteristics of such an architecture. DART stands for Decoupled ARchitecture for TSIMT. In DART the execution units and register files together with some additional logic form lanes that execute the instructions of the threads in a self-managed way, decoupling the execution of instruction from the fetching and decoding. With this decoupling it becomes possible to adjust the performance of the fetch and decode frontend and the execution lanes to the application and the level of divergence present in the application.

2 Proposed Architecture and Experimental Setup

Figure 1 shows the basic idea of the modeled architecture. The execution is handled by lanes formed from a register file, an execution unit and an instruction register. The instruction register stores the control vector at the execution unit and automatically cycles through all active threads within the warp. After all threads of the instruction have been sent to the execution unit the next instruction can be received from the frontend. However instructions are still fetched, scheduled and decoded in the front end on a per-warp level. While one of these lanes is busy with the self-managed execution of the SIMD instruction, the frontend is able to issue instructions to one of the other execution units. If all threads in all warps are active, the frontend needs to supply one instruction per 32 executed operations. If a smaller number of threads in the warp is active, due to divergence, the lane will sooner be able to accept a new instruction from the frontend. DARTs capability to handle divergent workloads stems from the flexiblity of the TSIMD lanes and the more powerful frontends ability to supply instructions at a higher rate if needed.

We modified GPGPU-Sim 3.1.0[BYF$^+$09] to simulate a DART GPU. As a baseline we configured a GPU with a single core with NVidia GT200 based parameters. GT200 based GPUs

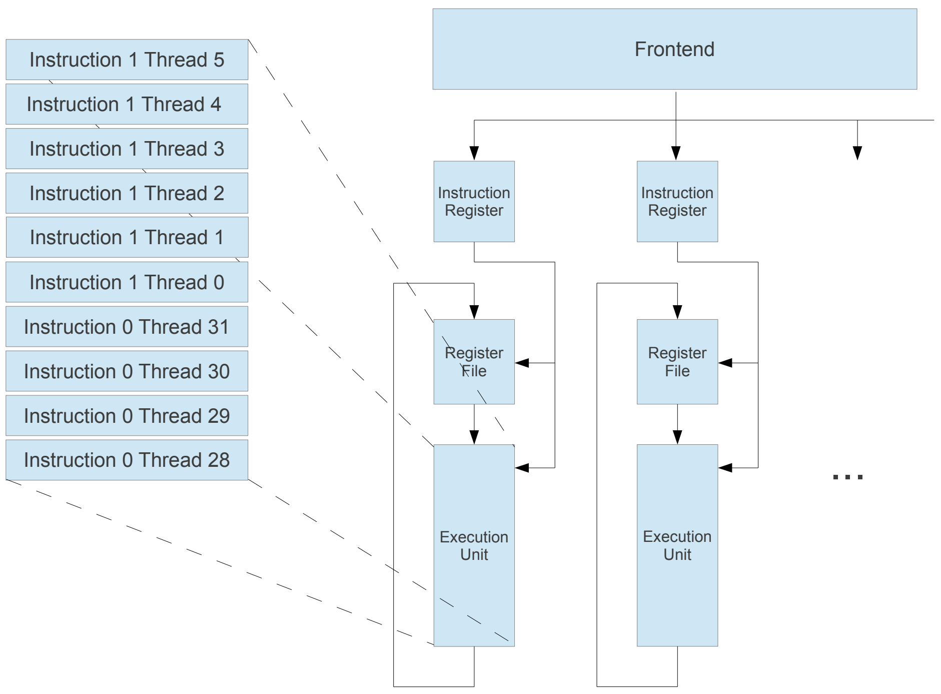

Figure 1: Diagram that shows the basic architectural idea of a TSIMD GPU core with multiple execution units

use SIMD units with 8 execution units. Execution of a warp is always done over 4 cycles, irrespective of the number of enabled threads per warp. The GT200 frontend can schedule new instructions every 4th cycle. The DART frontend is able schedule one new instruction every cycle if needed. It is thus capable of a up to 4x higher instruction fetch and decode throughput. The parameters of our simulated TSIMD are also based on a single GT200 core, but this time with 8 TSIMD lanes. Theoretical peak IPC for both configurations is 8.0.

3 Results

We developed a microbenchmark for testing the performance at different levels of divergence. To test different levels of divergence we can configure this microbenchmarks control flow to branch into 1 to 32 divergent flows per warp. Figure 2 shows the result of our microbenchmark runs. With all threads on the same control flow conventional GPUs and DART are showing similar performance. But even with just two to three different control streams conventional GPU performance shows a steep drop while DART is still able to archive almost peak IPC. The DART GPU is able to main a approximately 4x advantage over conventional GPUs on highly divergent workloads, which reflects increased frontend performance.

4 Conclusion

DART is an interesting and promising architecture for future GPUs. The ratio of instruction fetch and decode to execution units in DART GPUs can be adjusted to provide near

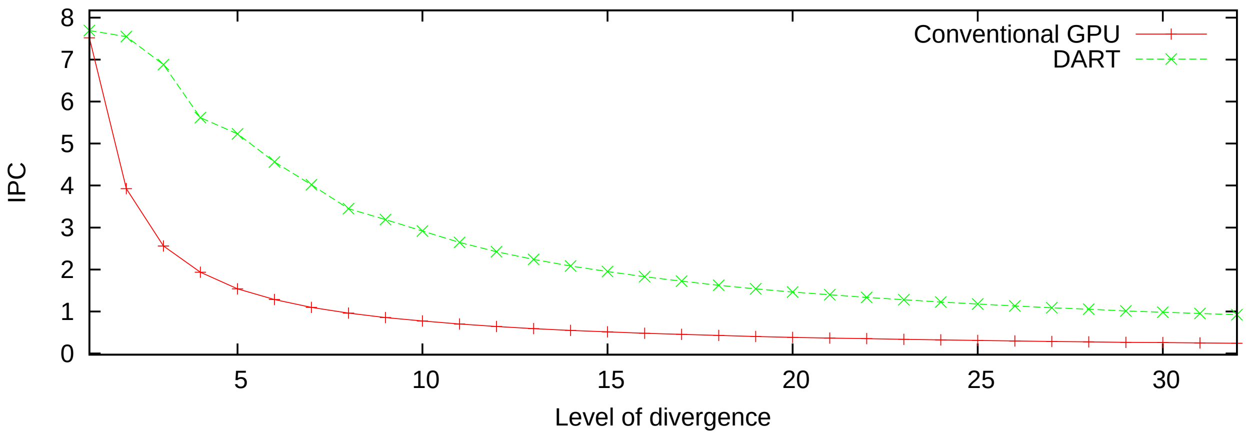

Figure 2: DART IPC vs. Conventional GPU IPC for different levels of divergence

optimal performance with divergent workloads. However additional research is needed to fully understand its performance characteristics. It might enable the effective use of GPUs for applications with a less regular control flow.

5 Acknowledgements

The research leading to these results received funding from the EC's 7th Framework Programme, LPGPU project, grant agreement n° 288653.

References

[BCD12] N. Brunie, S. Collange, and G. Diamos. Simultaneous Branch and Warp Interweaving for Sustained GPU Performance. In *Proc. of 39th Annual Int. Symp. on Computer Architecture*, pages 49–60, 2012.

[BYF+09] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, and T.M. Aamodt. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *Proc. of the IEEE Int. Symp. on Performance Analysis of Systems and Software*, 2009.

[KDK+11] S.W. Keckler, W.J. Dally, B. Khailany, M. Garland, and D. Glasco. GPUs and the Future of Parallel Computing. *Micro, IEEE*, 31(5):7–17, 2011.

[Kra11] R. M. Krashinsky. Temporal SIMT Execution Optimization, 2011. Patent, US 20130042090 A1.

[NSL+11] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt. Improving GPU Performance via Large Warps and Two-level Warp Scheduling. In *Proc. of the 44th Annual IEEE/ACM Int. Symp. on Microarchitecture*, pages 308–317, 2011.