OpenCL-Based High-Performance 3D Stencil Computation on FPGAs

Extended Abstract

Hamid Reza Zohouri*, Artur Podobas*, Naoya Maruyama†, and Satoshi Matsuoka*
*Tokyo Institute of Technology, †RIKEN Advanced Institute for Computational Science
Email: *{zohouri.h.aa@m, podobas.a.aa@m, matsu@is}.titech.ac.jp, †nmaruyama@riken.jp

KEYWORDS

FPGA, OpenCL, Stencil Computation

1 INTRODUCTION

In recent years, power usage and efficiency has become a major concern in high-performance computing (HPC). FPGAs, due to their low power consumption and architectural flexibility, are slowly finding their way in HPC. With the recent advancements in High-Level Synthesis (HLS), especially availability of OpenCL from FPGA manufacturers, these devices can now be considered as a viable alternative for common HPC accelerators like GPUs.

Stencils are one of the important computation patterns in HPC, used in solving PDEs and many different types of scientific simulations. In this work we show that for 3D stencil computation, due to architectural advantage of FPGAs, apart from superior power efficiency, we can also achieve competitive performance compared to a highly-optimized implementation on high-end GPUs.

2 RELATED WORK

Spatial and temporal blocking are widely used in stencil computation to reduce the memory bandwidth requirements. In [1], the authors present combined 3.5D spatial blocking and temporal blocking to minimize memory requirement for every stencil. In [2], a highly-optimized implementation of a 7-point 3D stencil is presented with 3.5D blocking and multiple different optimizations, and performance is reported on two generations of NVIDIA GPUs. We use the same stencil for our evaluation, and compare against the same highly-optimized GPU code.

Among recent implementations of stencil computation using HLS on FPGAs, [3] discusses multiple 2D, and one 3D stencil implementation using Intel FPGA SDK for OpenCL, and [4] presents an implementation of a Tsunami simulation with two 2D stencil passes. Both of these implementations take advantage of temporal blocking, but do not use spatial blocking and hence, cannot be used for large input sizes in which one dimension (for 2D) or one plane (for 3D) of the input does not fit in the FPGA on-chip memory. We avoid this unreasonable restriction by using spatial blocking. In [5], the authors present a framework for stencil computation targeting Xilinx's OpenCL SDK (SDAccel) with both spatial and temporal blocking. Since this work uses a multi-threaded implementation, the shift register-based optimization [6], which is a major advantage of using FPGAs for stencil computation compared to other architectures, cannot be used. Apart from this, they do not use 3.5D blocking and hence, miss a lot of room for optimization. Our implementation uses both techniques and multiple other advanced optimizations to maximize performance.

3 IMPLEMENTATION

We use 2.5D spatial blocking combined with temporal blocking as described in [1]. We use the single work-item programming model in Intel FPGA SDK for OpenCL. Using this model on FPGAs provides us with two major advantages over implementing the same algorithm on CPUs and GPUs:

- We can use shift registers as on-chip buffers [6], which allows us to significantly reduce on-chip storage size compared to other architectures.
- Lack of threads, barriers, and fixed warps in a deep-pipelined FPGA design eliminate thread divergence and the need for warp specialization [2].

These FPGA-specific advantages allow us to achieve better scaling with temporal blocking on these devices compared to GPUs. A top-down view of our overlapped blocking (tiling) technique is shown in Fig. 1 (*XY* plane). Computation is "streamed" in the *Z* dimension (not depicted here).

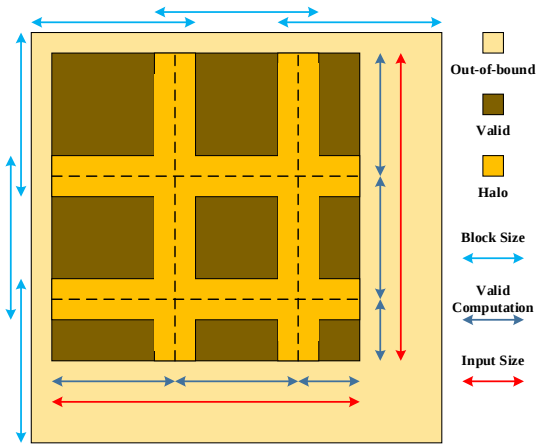

Figure 1: Top-down view of our blocking technique

SC'17, November 2017, Denver, Colorado USA

Our implementation avoids input size restrictions which are unacceptable in HPC; specifically, input dimensions neither need to be a multiple of block size, nor do they need to be smaller than a fixed value, as long as the input fits in FPGA off-chip memory.

Our design consists of a *read*, *write* and *compute* kernel. The *compute* kernel is defined as the advanced *autorun* kernel type and is replicated by the degree of temporal parallelism. Data is read from off-chip memory and streamed through the replicated *compute* modules and finally written back to off-chip memory by the *write* kernel. FPGA on-chip channels are used to connect these modules. Every replica of the *compute* kernel works on the same spatial block of a different time step, starting from the top-left one. All *compute* kernel replicas work on consecutive time steps in parallel, which each being one plane behind the previous one.

4 ADVANCED OPTIMIZATIONS

4.1 Loop Collapse

Since multiple nested loops are required to iterate over block and dimension variables, a lot of area is wasted on keeping the "state" of variables between these loops. To avoid this area overhead, we merge all these loops into one single *while* loop.

4.2 Exit Condition Optimization

The loop exit condition for stencil computation is determined by sequentially updating the index in every dimension and comparing it with the input size in that dimension. This creates a long critical path and reduces operating frequency (<200 MHz). We use a global index which is incremented once per loop iteration, and compare it with the loop iteration count to simplify exit condition. This shortens the critical path and increases the operating frequency to ~300 MHz.

4.3 Padding

On the FPGA, memory accesses that are not 512-bit aligned are split into two accesses. Due to starting from a negative address, and block overlapping in our design, many accesses will be non-aligned and memory performance will suffer. We pad the buffers in FPGA off-chip memory, which allows full alignment if the degree of temporal parallelism is a multiple of 4, and improves alignment for other values.

5 METHODOLOGY

We target the Terasic DE5-Net (Stratix V 5SGXA7), and Nallatech 385A boards (Arria 10 GX 1150). Block size, loop unroll factor and degree of temporal parallelism are tuned for both FPGAs to achieve the best performance. Power usage for the Stratix V board is estimated using software, and 2.34 Watts is added for its memory module. For the Arria 10 board we use the on-board power sensor. We compare our results with the highly-optimized implementation from [2] on the highest-end Kepler, Maxwell and Pascal NVIDIA GPUs (with ECC disabled). We measure power on the GPUs using NVIDIA's NVML library.

For all cases we report computation throughput and power efficiency for a cubic input with a dimension size that achieves best performance on that particular device; for FPGAs, this value is a multiple of *block_size – (2 * halo_size)*, (minimum out-of-bound computation) and for GPUs is a multiple of 512 (maximum SM utilization). We use 1000 iterations so that kernel run time is over 2 seconds in every case. We disregard host/device transfer time for this evaluation.

6 RESULTS

Fig. 2 shows measured throughput on the evaluated hardware, alongside with their peak memory bandwidth and power efficiency. Our evaluated stencil is memory-bound on all the GPUs, but despite using temporal blocking, it is not possible to reach the peak memory bandwidth of these devices. This confirms the limited effectiveness of temporal blocking on GPUs. However, due to the architectural advantage of FPGAs for stencil computation, temporal blocking achieves good scaling and allows us to achieve multiple times higher effective computation throughput than the peak memory bandwidth of these devices. Because of this, despite much lower peak memory bandwidth, the Arria 10 FPGA achieves better performance than the Tesla K40c GPU, and higher power efficiency compared to GTX 980 Ti, and close to that of the state-of-the-art Tesla P100.

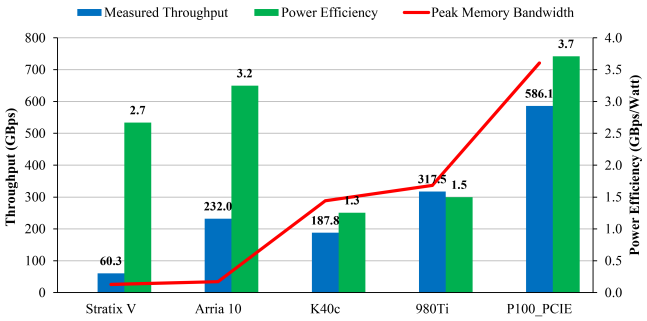

Figure 2: **Throughput, peak bandwidth and power efficiency**

7 CONCLUSION

In this work we showed that due to the architectural advantage of FPGAs, it is possible to achieve better scaling with temporal blocking for stencil computation compared to GPUs. Also unlike previous work on FPGAs, we employ spatial blocking to avoid putting artificial limits on input size. These allow us to achieve comparable performance to that of modern GPUs for large input sizes, despite having much lower memory bandwidth.

With addition of HBM to the upcoming Stratix 10 FPGAs, we expect a 2x speed-up from temporal blocking to be enough to achieve better performance compared to same-generation GPUs, even for higher-order stencils.

ACKNOWLEDGEMNT

This work was supported by MEXT, and JST CREST under Grant Number JPMJCR1303, and performed under the auspices of the Real-world Big-Data Computation Open Innovation Laboratory, Japan.

REFERENCES

[1] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey. 2010. 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*. IEEE Computer Society, Washington, DC, USA, 1-13.

[2] Naoya Maruyama, Takayuki Aoki. 2014. Optimizing stencil computations for NVIDIA Kepler GPUs. In *Proceedings of the 1st Int'l Workshop on High-Performance Stencil Comp. (HiStencils '14)*. Vienna, Austria, 89-95.

[3] Hasitha Muthumala Waidyasooriya, Yasuhiro Takei, Shunsuke Tatsumi, and Masanori Hariyama. 2017. OpenCL-Based FPGA-Platform for Stencil Computation and Its Optimization Methodology. *IEEE Trans. Parallel Distrib. Syst.* 28, 5 (May 2017), 1390-1402.

[4] Kohei Nagasu, Kentaro Sano, Fumiya Kono, and Naohito Nakasato. 2017. FPGA-based tsunami simulation: Performance comparison with GPUs, and roofline model for scalability analysis. *J. of Parallel and Distrib. Comput.* 106 (Aug. 2017), 153-169.

[5] Shuo Wang and Yun Liang. 2017. A Comprehensive Framework for Synthesizing Stencil Algorithms on FPGAs using OpenCL Model. In *Proceedings of the 54th Annual Design Automation Conference 2017* (DAC '17). ACM, New York, NY, USA, Article 28, 6 pages.

[6] Intel PSG. 2017. Finite Difference Computation (3D) Design Example. (May 2017). Retrieved July 22, 2017 from https://www.altera.com/support/support-resources/design-examples/design-software/opencl/fdtd-3d.html