ANL/ALCF-17/2

Evaluation of the Single-precision Floating-point Vector Add Kernel Using the Intel FPGA SDK for OpenCL

Argonne Leadership Computing Facility

ANL/ALCF-17/2

Evaluation of the Single-precision Floating-point Vector Add Kernel Using the Intel FPGA SDK for OpenCL

prepared by
Zheming Jin, Kazutomo Yoshii, Hal Finkel, Franck Cappello

Argonne Leadership Computing Facility, Argonne National Laboratory

April 20, 2017

Evaluation of the Single-precision Floating-point Vector Add Kernel Using the Intel FPGA SDK for OpenCL

Zheming Jin* Kazutomo Yoshii Hal Finkel Franck Cappello

ABSTRACT

Open Computing Language (OpenCL) is a high-level language that enables software programmers to explore Field Programmable Gate Arrays (FPGAs) for application acceleration. The Intel FPGA software development kit (SDK) for OpenCL allows a user to specify applications at a high level and explore the performance of low-level hardware acceleration.

In this report, we present the FPGA performance and power consumption results of the single-precision floating-point vector add OpenCL kernel using the Intel FPGA SDK for OpenCL on the Nallatech 385A FPGA board. The board features an Arria 10 FPGA. We evaluate the FPGA implementations using the compute unit duplication and kernel vectorization optimization techniques. On the Nallatech 385A FPGA board, the maximum compute kernel bandwidth we achieve is 25.8 GB/s, approximately 76% of the peak memory bandwidth. The power consumption of the FPGA device when running the kernels ranges from 29W to 42W.

1. INTRODUCTION

The OpenCL standard is an open programming model for accelerating algorithms on heterogeneous computing system. OpenCL extends the C-based programming language for developing portable codes on different platforms such as CPU, GPU, DSP and FPGA. The Intel FPGA SDK for OpenCL is a suite of tools that allows developers to abstract away the complex FPGA-based development flow for a high-level software development flow. Users can focus on the design of hardware-accelerated kernel functions in OpenCL and then direct the tools to generate the low-level FPGA implementations.

Vendors offer board support package (BSP) variants to support different applications on their boards. The BSP leverages the on-board and low-level resources on FPGAs to allow users to quickly develop applications without manually building the hardware basic blocks. Users can focus on the algorithm implementation on FPGAs rather than the physical implementation at the board level.

With the SDK and BSP tools, users can evaluate the performance of an FPGA implementation of a kernel within one day though the FPGA compile time is still quite slow (typically several hours) from the software development aspect.

2. BACKGROUND

2.1 OpenCL application

An OpenCL application consists of host and kernel programs. The OpenCL host program is written in standard C/C++ that runs on most of modern microprocessors. The host allocates data arrays in the global memory that will be read by the kernel. When the data are ready for the kernel, the host can launch the kernel that will be executed on an FPGA device. A kernel typically executes computation by reading data from global memory as specified by the host, processing it, and then writing the results back into global memory. When the results are ready, they can be read by the host for post-processing.

2.2 Nallatech 385A

Nallatech provides OpenCL board support packages for OpenCL users. Nallatech 385A is a PCIe-based FPGA accelerator card. It features an Arria 10 GX1150 FPGA device, PCIe x8 Generation 3 host interface, and two banks of 4GB DDR3 memory. The theoretical peak floating-point performance is 1.5 TFLOPS and the theoretical peak memory bandwidth approximately 34 GB/s.

3. KERNEL APPLICATION

3.1 OpenCL vector add

Our case study is the OpenCL single-precision floating-point vector add kernel based on the design example [1]. The vector add kernel sums up the two vectors *X* and *Y* and then store the results in another vector *Z*. Figure 1 shows the OpenCL vector add kernel. Each work item (thread) in the global space reads two elements from *X* and *Y* and writes the sum into *Z*. The inputs and output are read from and written to the global external DDR memory. This kernel can illustrate the impact of different kernel configurations on the performance of the kernel implementations using the Intel Altera SDK.

```
__kernel void vector_add(__global float *x,
                         __global float *y,
                         __global float *z)
{
   int i = get_global_id(0);
   z[i] = x[i] + y[i];
}
```

Figure 1. The OpenCL vector add kernel

Contact Address: Zheming Jin, Leadership Computing Facility, 9700 South Cass Avenue, Bldg, 240, Argonne, IL 60439-4854 USA

Email: zjin@anl.gov

3.2 Kernel optimizations

As described in [2], users can take advantage of compute unit replication and kernel SIMD vectorization to achieve higher throughput or lower kernel time. The compute device replication generates multiple compute units for each kernel. Each compute unit has its own memory access interface. The SIMD vectorization duplicates only the data path of the compute unit without generating additional memory interfaces. When the kernel is vectorized, the static memory coalescing is performed automatically by the compiler to generate a memory interface that can coalesce the multiple memory loads into a single wide load. While there is no limit to the number of kernel copies that users can specify, the number of SIMD lanes must be a power of two. The compiler will give a warning when the width of all the lanes exceeds the memory interface data width.

4. PERFORMANCE EVALUATION

In this work, a host system is set up with two 2.6 GHz Intel Xeon processors and 32GB DDR3 memory for each node. The PCI Express provides a Gen3×8 connection. CentOS 6.8 with Linux kernel 2.6.32 is installed as the operating system. We used Intel's FPGA SDK for OpenCL version 16.0.2 Pro Prime for producing the experimental results.

Table 1. Resource usage of the FPGA implementations

Kernel type	Native DSP(s)	Logic utilization	Memory bits	RAM blocks
default	1	12%	5%	13%
cu2	2	12%	7%	15%
cu4	4	13%	8%	18%
cu8	8	16%	8%	24%
cu16	16	20%	10%	37%
cu32	32	29%	13%	62%
cu48	48	38%	15%	86%
simd2	2	13%	5%	13%
simd4	4	13%	5%	13%
simd8	8	13%	5%	13%
simd16	16	13%	5%	13%
simd16+cu2	32	13%	8%	15%
simd16+cu4	64	15%	8%	18%
simd16+cu8	128	18%	9%	24%
simd16+cu16	256	26%	12%	35%
simd16+cu32	512	41%	16%	58%
simd16+cu48	768	57%	21%	81%

Table 1 lists the FPGA resource usage reported by the SDK for the different implementations of the kernel. The *default* kernel is the example shown in Figure 1 without any kernel optimization. Replication of compute unit is represented as "cuX" where X indicates the replication times. We also include the combination of kernel duplication and 16-lane vectorization at the bottom of the table as "simd16+cuX". The number of floating-point DSP blocks generated by the SDK depends on the kernel configurations. The logic utilization ranges from 12% to 57%. The RAM block usage increases significantly from 13% to 86% as the number of compute units increase from 2 to 48. When the number of compute units is 64 (cu64), the SDK fails to implement the design that requires more RAM blocks than the target device can provide. On the other hand, the SIMD vectorization maintains 13% logic utilization as the number of lanes increases from 2 to 16. Kernel vectorization duplicates only the data path of the kernel with little resource overhead for the additional control logics.

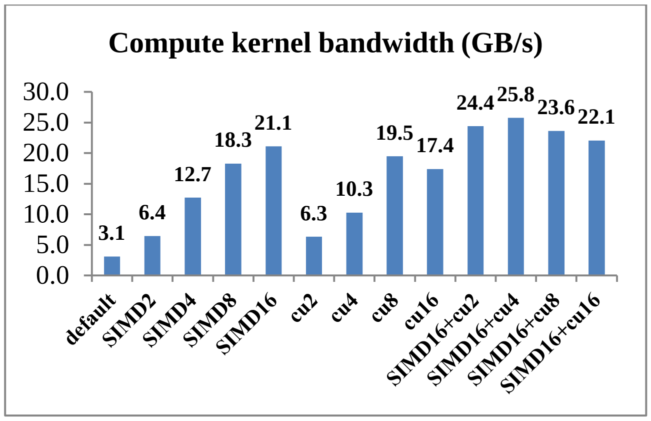

Figure 2. Compute kernel bandwidth

We use a vector size of 512M to measure the compute kernel bandwidth for each kernel mentioned above. The compute kernel bandwidth is defined as follows:

$$BW_{kernel} = 512 \times 2^{20} \times 12 \,/\, (\text{kernel execution time})$$

Each single-precision floating-point operation accesses 12 bytes from the memory. As shown in Figure 2, the maximum bandwidth is 25.8 GB/s, approximately 76% of the peak memory bandwidth on the Nallatech 385A board. The hybrid kernel (*SIMD16+cu4*) achieves the maximum bandwidth. Kernel duplication increases the kernel bandwidth from 3.1GB/s (*default*) to 19.5GB/s (*cu8*) and from 21.1GB/s (*SIMD16*) to 25.8 GB/s (*SIMD16+cu4*). As more duplicate kernels are added, the bandwidth starts to diminish due to the external memory accesses contention. So the results for more than 16 duplicate kernels are not included in the report.

Table 2 compares the ideal and actual speedup for each kernel using the 512M vector size. The ideal speedup is the number of compute units using kernel duplication and/or vectorization for a kernel while the actual speedup is the ratio of the execution time of the baseline kernel (*default*) to the execution time of the kernel using kernel duplication and/or vectorization. As seen in Table 2, *SIMD2*, *SIMD4* and *cu2* achieve the expected speedup while the gap between the ideal and actual speedup gradually widens for other cases. Overall it is worthwhile to duplicate the kernel eight times or utilize eight vector lanes to achieve higher performance if resource usage is not a constraint. However, the performance gain diminishes significantly using more than eight DSPs for both kernel duplication and vectorization.

Table 2. The ideal speedup vs. the actual speed

Kernel type	Kernel time (ms)	Ideal speedup	Actual speedup
default	1954	1	1
SIMD2	932	2	2
SIMD4	472	4	4
SIMD8	328	8	6
SIMD16	284	16	6.9
cu2	950	2	2
cu4	584	4	3.3
cu8	308	8	6.3
cu16	345	16	5.7
SIMD16+cu2	246	32	7.9
SIMD16+cu4	233	64	8.4
SIMD16+cu8	254	128	7.7
SIMD16+cu16	272	256	7.2

Figure 3 presents the power consumption in Watts of the FPGA device when each kernel is running on the Nallatech 385A. The power measured is for 12V power supply. FPGA power consumption ranges from 29W to 42W. As shown in Figure 3, adding more compute kernels generally increases the power consumption.

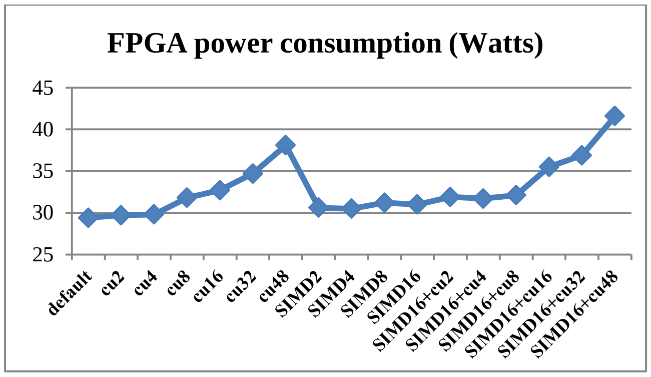

Figure 3. FPGA power consumption

5. CONCLUSION

The performance evaluation of the single-precision floating-point vector add OpenCL kernel on the Nallatech 385A FPGA illustrates the impact of kernel optimizations on the performance and power consumption of the FPGA implementations. Compute kernel duplication and kernel vectorization can reduce the kernel execution time at the cost of more hardware resources. Compute device duplication requires a memory interface for each duplicated kernel while kernel vectorization only duplicates the data path of the kernel to utilize memory bandwidth more efficiently.

Given the 512-bit user interface of the DDR3 memory controller implemented on the FPGA, the 16-lane SIMD-based vector add kernels achieve the best compute kernel bandwidths. For the same number of DSPs and logic utilization, kernel vectorization is more efficient to improve the performance than the duplication. Heavy kernel duplication significantly diminishes the performance gain due to the memory access contentions. Overall SIMD kernel vectorization is the preferred optimization technique to reduce the kernel execution time when the external memory interface and the number of DSPs support SIMD computation.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy Office of Science, under contract DEAC02-06CH11357.

7. REFERENCES

[1] https://www.altera.com/support/support-resources/design-examples/design-software/opencl/vector-addition.html

[2] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/aocl_optimization_guide.pdf

Argonne Leadership Computing Facility
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439

www.anl.gov