

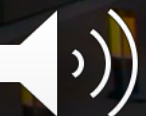
Experiences Porting the SU3_Bench Microbenchmark to the Intel Arria 10 and Xilinx Alveo U280 FPGAs



BERKELEY LAB

Douglas Doerfler,
Farzad Fatollahi-Fard, Colin MacLean, Tan Nguyen,
Samuel Williams and Nicholas J. Wright
Lawrence Berkeley National Laboratory
Berkeley, California

Marco Siracusa
Politecnico di Milano
Milano Italy



Introduction

- We examine porting SU3_Bench to two different FPGAs
- And in particular evaluate the efficiency achieved using the OpenCL toolchains on their respective devices
 - **It is NOT the purpose to compare the two FPGAs head to head**
 - Performance is evaluated relative to the capabilities of the device
 - In particular, to the maximum performance of their respective memory subsystems
 - Studies with GPU architectures have shown 75%-90% architectural efficiencies
- In addition, evaluate the portability considerations
 - In moving between the two vendor toolchains
 - And in moving from a GPU based kernel design

SU3_Bench Microbenchmark

- Lattice Quantum Chromodynamics SU(3) matrix-matrix multiply microbenchmark
- Matrix-vector, and matrix-matrix operations are key computational elements of Lattice QCD codes
- SU3_Bench kernel has been ported to numerous programming languages: CUDA, HIP, OpenCL, OpenMP, OpenACC, SYCL ...
- A member of the NERSC proxy application suite
https://gitlab.com/NERSC/nersc-proxies/su3_bench
Use the FPGA branch: `$ git checkout fpga`

Nominal SU3_Bench SWI Kernel


```
#define CMULSUM(a,b,c) {
    (c).real += (a).real*(b).real - (a).imag*(b).imag;
    (c).imag += (a).real*(b).imag + (a).imag*(b).real; }

__kernel void k_mat_nn(
__global const site*      restrict a,
__global const su3_matrix* restrict b,
__global      site*      restrict c,
      const int      total_sites) {
for (int i=0; i<total_sites; ++i)
for (int j=0; j<4; ++j)
for (int k=0; k<3; k++)
for (int l=0; l<3; l++) {
    Complx cc = {0.0, 0.0};
    for (int m=0; m<3; m++)
        CMULSUM(a[i].link[j].e[k][m], b[j].e[m][l], cc);
    c[i].link[j].e[k][l] = cc;
}
}
```

SU3 Matrix & Site structure definitions

```
typedef struct {std::complex<float> e[3][3];} fsu3_matrix;
typedef struct {std::complex<double> e[3][3];} dsu3_matrix;
#if (PRECISION==1)
    #define su3_matrix fsu3_matrix
#else
    #define su3_matrix dsu3_matrix
#endif
```

```
typedef struct {
    su3_matrix link[4]; // the fundamental gauge field
    int x,y,z,t; // coordinates of this site
    int index; // my index in the array
    char parity; // even or odd site
#if (PRECISION==1)
    int pad[2]; // pad to 64 byte alignment
#else
    int pad[10];
#endif
} site;
```

Source code listings can be found in the appendix of the  <https://doi.org/10.1145/3456669.3456671>

NDRange vs. SWI Kernels

NDRange Kernel:

- Nominal OpenCL parallel programming model with multiple work items
- Can utilize multiple compute units, each operating on an independent subset of the entire data set

```
int i = get_global_id(0);
if (i < total_sites )
    for (int j=0; j<4; ++j)
        *
```

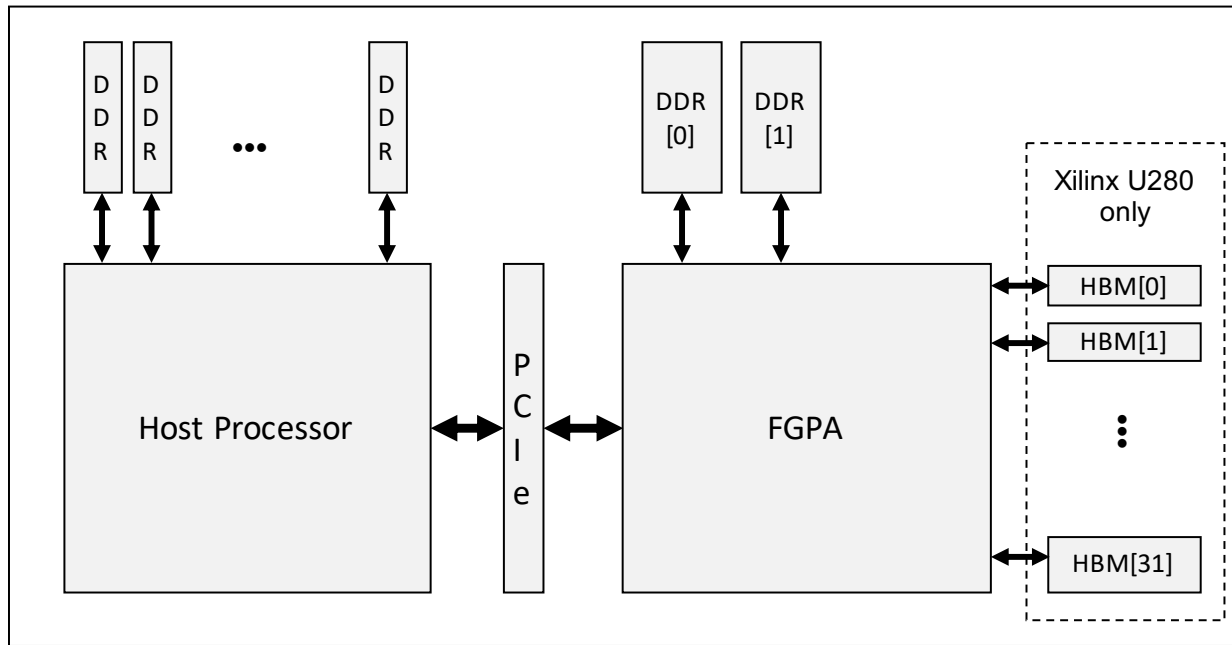
Single work item (SWI) Kernel:

- Sequential programming model
- Single compute unit (CU) operates on the entire data set

```
for (int i=0; i<total_sites; ++i)
    for (int j=0; j<4; ++j)
        *
```

Target FPGA Architectures

Generalized Test Platform



1. Host Processor: Intel® Xeon® CPU E5-2650 v2
2. Only 2 HBM channels used in this study

	Intel Arria 10 GX	Xilinx Alveo U280
# DDR Channels	2	2
DDR Capacity (GB)	8	32
DDR Peak BW (GB/s)	34.1	38.4
# HBM Channels	N/A	32 ⁽²⁾
HBM Peak BW (GB/s)	N/A	460 (14.1/ch.)
Toolchain	Quartus Prime	Vitus Unified Software
Compiler Version	Version 17.1.1	V2019.2.1

Intel Arria 10 FPGA Results

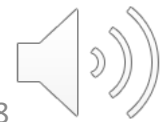
Kernel Type	GB/s	% Sustained ⁽¹⁾
NDRange Kernel w/Unroll	3.95	15.8 %
With 4 Compute Units	3.47	13.9 %
SWI Kernel with Unroll	11.9	47.6 %
No Interleaving DRAM channels, single channel	6.95	27.8 %
No Interleaving, capacity footprint spanning both channels	13.3	53.2 %

1. Peak BW is 34.1 GB/s, and maximum sustained BW is 25.0 GB/s
(T. Nguyen, S. Williams, M. Siracusa, C. MacClean, D. Doerfler, and N.J. Wright. The Performance and Energy Efficiency Potential of FPGAs in Scientific Computing. *In 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 8–19. <https://doi.org/DOI10.1109/PMBS51919.2020.00007>)



Intel Arria 10 FPGA Analysis

- Best Performance, 47.6 % max. sustained BW, was obtained with a SWI Kernel with a single unroll directive
- NDRange Kernel is 3.0x slower
 - Increasing the number of compute units to 4 results in even lower performance
- By NOT interleaving the DRAM channels, a 1.12x improvement is observed
 - However, if capacity footprint is small this speedup can't be achieved
 - May not be a very practical configuration
- Intel toolchain synthesizes a cache using local memory (block RAM)



Performance Potential for Kernel Types?

Kernel Type	GB/s
NDRange Kernel with Unroll	229.6
SWI Kernel with Unroll	5,562

- Measured “performance” when the kernel has no compute component
 - No actual data movement takes place, but kernel invocation is performed
- This sets an upper bound on performance, and is a measure of runtime overhead
- For an NDRange kernel, this upper bound is still >> achieved performance?
 - It is observed that SWI kernels are implicitly pipelined, but NDRange kernels are not?
 - Perhaps this is to conserve resource usage in a multiple CU design?

Xilinx Alveo U280 FPGA Results

Kernel Type	GB/s ⁽¹⁾	% Sustained ⁽²⁾
NDRange with 16 workers	0.297	2.27 %
With 4 Compute Units	0.454	3.47 %
SWI Kernel	0.167	1.27 %
Using a scratch pad in local memory	4.61	35.2 %
With scratch pad and compute loops combined	1.23	9.39 %

1. Only 2 of the 32 HBM channels were used in this study, one dedicated to read operations and another dedicated to write operations, this avoids a known turnaround overhead in the U280 FPGA HBM controller (Xilinx Inc. Alveo U280 Data Center Accelerator Card Data Sheet. DS963 (v1.3) May 11, 2020.),
It is the intent that future studies would analyze all HBM channels
2. Peak BW of a single HBM channels is 14.4 GB/s, and maximum sustained BW is 13.1 GB/s

Xilinx Alveo U280 FPGA Analysis

- Best performance, 35.3% max. sustained, was obtained with a SWI kernel using a scratch pad to burst input data into local memory (Block RAM)
 - Scratch pad necessary as Xilinx toolchain does not synthesize a local memory cache
 - 27x performance increase over nominal kernel w/o scratch pad!
 - See the paper for extensive source code modifications, 46 LOC vs. 18 LOC
- NDRange Kernel is 15.5x slower than SWI
- A modified version of the SWI kernel integrates the scratch pad copy code into the compute loop
 - This is an elegant modification, and probably how a programmer would want to write it to begin with
 - However, this results in a 3.8x slowdown because the compiler does not schedule burst read transfers, why?

Portability Considerations

- NDRange kernel is preferable
 - It's desirable for FPGA vendors to address overheads of NDRange kernels
 - It is the how parallel OpenCL kernels are written
 - Eases the programmer's use of multiple compute units as a SWI design requires the programmer to explicitly target multiple kernels in host code
- Using Block RAM as a cache greatly simplifies the code
 - Intel's toolchain gets this right!
 - Xilinx toolchain requires the programmer to explicitly write code to copy data to local variables, AND to expose the data as 512-bit aligned to enable 512-bit transfers are scheduled

Portability Considerations

- Unrolling vs. Pipelines
 - Xilinx toolchain requires explicit pipeline directives, and implicitly unrolls
 - Intel toolchain requires explicit unroll directives, and implicitly pipelines
- Host side code changes
 - Xilinx toolchain requires that `CL_MEM_USE_HOST_PTR` be set when allocating OpenCL buffer objects
 - Xilinx toolchain complains if 4096 byte alignment is not used by host memory allocations, this required a custom allocator for `std::vector`

Conclusion

- Both toolchains were relatively easy to use and obtain initial results
 - With the caveat of long compile times when synthesizing the logic (~2 to ~8 hours)
- Demonstrated 35%-48% architectural efficiencies for FPGAs
 - Studies with GPU architectures have shown 75%-90% architectural efficiencies
- Although it is well known that SWI kernels perform better on FPGAs, this is not what CPU and GPU OpenCL codes prefer
 - **Vendors should improve the performance of their NDRange implementations to ease the transition of existing codes!**
 - Granted, it is a relatively simple transition to SWI, but it is much easier to scale the design to multiple CUs with a NDRange design
- Use the Xilinx HLS compiler instead of OpenCL (perhaps future work)
 - Enables declaration of 512-bit data types
 - Enables a systolic pipeline design which is better suited to the Read → Compute → Write data flow
 - But of course, this is at the total expense of OpenCL portability

Thank You for Your Interest!

You can find the paper here:

<https://doi.org/10.1145/3456669.3456671>

