Evaluating FPGA Accelerator Performance with a Parameterized OpenCL Adaptation of Selected Benchmarks of the HPCChallenge Benchmark Suite

Marius Meyer
Tobias Kenter, Christian Plessl
Paderborn University, Germany
Paderborn Center for Parallel Computing

H2RC’20, everywhere, 13. November 2020
An FPGA-adapted implementation of HPCC

- **OpenCL** kernels and C++ host code – Measure hardware and tools
- Support for **Intel** and **Xilinx** FPGAs
- **Configuration Options** to adapt to resources and architecture
- It’s **open source** and already available on GitHub!
The HPC Challenge Suite

**Synthetic Benchmarks**
- STREAM
- RandomAccess
- b_eff

**Benchmark Applications**
- GEMM
- PTRANS
- FFT
- HPL

**Base runs:**
Use unmodified provided benchmark implementations

**Optimized runs:**
Modifications allowed with respect to the benchmark rules

**Idea:** Memory access patterns of other application will always be a combination of the patterns implemented by these benchmarks
We focus on **base** implementations for now…

Two main concepts to increase resource utilization and performance:

**Scaling**:
- Match data width of fixed interfaces
- Increase parallelism to make use of more resources
- Individual options for every benchmark

**Replication**:
- Utilize all available interfaces
- Increase resource usage
- Option: `NUM_REPLICATIONS`
Nallatech 520N
- Intel Stratix 10 GX 2800
- 4x 8 GB DDR4 SDRAM
- x8 PCIe 3.0

Intel PAC D5005
- Intel Stratix 10 SX 2800
- Direkt access to host memory using SVM
- x16 PCIe 3.0

Xilinx Alveo U280
- XCU280
- 32x 256 MB HBM2 on FPGA
- 2x 16 GB DDR4 SDRAM
- x8 PCIe 4.0
Benchmark Implementation
Operations Measured by STREAM for FPGA

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Kernel Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe Write</td>
<td>Write arrays to device</td>
</tr>
<tr>
<td>Copy</td>
<td>$C[i] = A[i]$</td>
</tr>
<tr>
<td>Scale</td>
<td>$B[i] = j \cdot C[i]$</td>
</tr>
<tr>
<td>Add</td>
<td>$C[i] = A[i] + B[i]$</td>
</tr>
<tr>
<td>Triad</td>
<td>$A[i] = j \cdot C[i] + B[i]$</td>
</tr>
<tr>
<td>PCIe read</td>
<td>Read arrays from device</td>
</tr>
</tbody>
</table>

**Configuration Options:**
- **DATA_TYPE**
- **VECTOR_COUNT**
- **GLOBAL_MEM_UNROLL**: Unroll the loops
- **DEVICE_BUFFER_SIZE**: Size of the local memory buffer
- **NUM_REPLICATIONS**: One kernel per memory bank

---

**Algorithm 1 STREAM kernel**

1: $B = \text{DEVICE\_BUFFER\_SIZE}$
2: $\text{buffer} = \text{ARRAY} [B]$
3: for $i = 1$ to $\frac{N}{B}$ do
4: for $j = 1$ to $B$ do
5: $\text{buffer}[j] = \text{scalar}$ input1[$i \cdot B + j$]
6: end for
7: if second_input then
8: for $j = 1$ to $B$ do
9: $\text{buffer}[j] = \text{buffer}[j] + \text{input2}[i \cdot B + j]$  
10: end for
11: end if
12: for $j = 1$ to $B$ do
13: output[$i \cdot B + j$] = buffer[j]
14: end for
Observations

- Kernel needs to support two different kernel designs to work best with all global memory types
- STREAM achieves a high memory efficiency independent of operation for half-duplex memory interfaces
**Description:** Update values in a large data array in pseudo random order. Update errors allowed!

**Configuration Option:**
- **DEVICE_BUFFER_SIZE:** Size of the local memory buffer
- **NUM_REPLICATIONS:** One kernel per memory bank

**Every kernel:**
- Calculates the same pseudo random number sequence
- Update only, if address is in memory bank
- Two pipelines used to remove dependencies between reads and writes

---

**RandomAccess Implementation**

Data $D$  

Index of next value in data array $k$

Random Numbers $R$  

Update value $D_k$

Local memory buffer

Bank 1  

Bank 2

$2^n$
## RandomAccess Results

<table>
<thead>
<tr>
<th>Option</th>
<th>520N DDR</th>
<th>U280 DDR</th>
<th>U280 HBM2</th>
<th>PAC SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_REPLICATIONS</td>
<td>4</td>
<td>2</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>DEVICE_BUFFER_SIZE</td>
<td>1</td>
<td>1,024</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Observations:
- Compiler support for ignoring data dependencies has a huge impact on performance.
- Number of kernel replications has negative impact on performance.
**Description:** Batched calculation of 1d FFTs

**Configuration Options:**
- **LOG_FFT_SIZE:** $\log_2$ of the 1d FFT size
- **NUM_REPLICATIONS:** One kernel for **two** memory banks
  - Implementation is fully pipelined
  - Fetch: BRAM
  - FFT: BRAM/Logic, DSPs

**Performance Model**

$$p_{FFT} = 5 \cdot \text{LOG}_2\text{FFT}_\text{SIZE} \cdot f_{mem} \cdot \text{NUM}_\text{REPLICATIONS} \cdot 8$$
### FFT Results

#### Option

<table>
<thead>
<tr>
<th></th>
<th>520N DDR</th>
<th>U280 DDR</th>
<th>U280 HBM2</th>
<th>PAC SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_REPLICATIONS</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>LOG_FFT_SIZE</td>
<td>17</td>
<td>9</td>
<td>5</td>
<td>17</td>
</tr>
</tbody>
</table>

**Observations**

- Design allows high utilization of the global memory for a broad range of FFT sizes
- Performance can be achieved equally over both configuration options
Description: Multiply square matrices  
\[ C' = \alpha \cdot A \cdot B + \beta \cdot C \]  
where \( A, B, C, C' \in \mathbb{R}^{n \times n} \) and \( \alpha, \beta \in \mathbb{R} \)

Configuration Parameters:
- **DATA_TYPE**: Used data type
- **GLOBAL_MEM_UNROLL**: Number of values that are loaded into local memory per clock cycle (\( u \))
- **BLOCK_SIZE**: Size of the local memory block (\( b \))
- **GEMM_SIZE**: Size of the register block (\( g \))
- **NUM_REPLICATIONS**: Used to fill FPGA resources

GEMM Implementation

\[ t_{exe} = \frac{b^2}{u \cdot f_{mem}} + \frac{b^3}{g^3 \cdot f_k} + \frac{b^2}{u \cdot \frac{n}{b} \cdot f_{mem}} \]
GEMM Results

<table>
<thead>
<tr>
<th>Option</th>
<th>520N DDR</th>
<th>U280 DDR</th>
<th>U280 HBM2</th>
<th>PAC SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_TYPE</td>
<td>float</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLOBAL_MEM_UNROLL</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEMM_SIZE</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLOCK_SIZE</td>
<td>512</td>
<td>256</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>NUM_REPLICATIONS</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Observations

- Large in-register multiplication leads to low kernel frequencies
- HBM2 can also improve the performance of mainly compute bound applications

Normalized Performance to 100MHz and a single Kernel Replication

Kernel Frequency [MHz]
It is a challenging task to create unbiased base implementations.

The implementations show a similar performance efficiency on the tested devices.

The implementations allow to adjust the utilization of relevant resources for a broad range of FPGAs.

Next Steps:
- Implement remaining base implementations
- Offer support for multi-FPGA execution of the benchmarks
- Utilize inter-FPGA networks