Final Project of Master

Implementation of resilient algorithm Backpropagation on ARM-FPGA through OpenCL
Contents

Introduction

Chapter 1. Introduction to OpenCL

Chapter 2. OpenCL with Altera SDK

Chapter 3. Matrix-multiplication Tiling

Chapter 4. Backpropagation Algorithm

Chapter 5. integrate IPs in OpenCL and Results

Conclusion
Introduction

OpenCL

- The open standard for parallel programming of heterogeneous systems
- found in personal computers, servers, mobile devices and embedded platforms
- improves the speed and responsiveness of a wide spectrum of applications in numerous market categories such gaming and medical software
Chapter 1. Introduction to OpenCL

OpenCL Platform Model

- A host is connected to one or more OpenCL devices
- OpenCL device is collection of one or more compute units
- A compute unit is composed of one or more processing elements
Chapter 1. Introduction to OpenCL

OpenCL Platform Model - Application

- **Host Code**
  - Written in C/C++
  - Executes on the host

- **Device Code**
  - Written in OpenCL C
  - Executes on the device

- **Host**
  - Sends commands to the **Devices**
    - to transfer data between host memory and device memories
    - to execute device code
Chapter 1. Introduction to OpenCL

Execution Model

Kernels execute on one or more OpenCL devices and a host program executes on the host.

The host defines:
- The collection of OpenCL devices
- The OpenCL functions
- Applications queue kernels and data transfers
- Performed in-order or out-of-order

NDRRange index Space

[Diagram showing NDRRange index space with grid and indices]
Chapter 1. Introduction to OpenCL

The BIG Idea behind OpenCL

- Define N-dimensional computation domain
- Execute a kernel at each point in computation domain

Traditional loop as a function in C

```c
void trad_mul(int n,
    const float *a,
    const float *b,
    float *c)
{
    int i;
    for (i=0; i<n; i++)
        c[i] = a[i] * b[i];
}
```

OpenCL C kernel

```c
__kernel void dp_mul(__global const float *a,
    __global const float *b,
    __global float *c)
{
    int id = get_global_id(0);
    c[id] = a[id] * b[id];
    // execute over n"work items"
}
```
Chapter 1. Introduction to OpenCL

Memory Model

- **Private Memory**
  Per work-item
  implemented with registers
- **Local Memory**
  Per workgroup
  implemented using on-chip memory
- **Global/Constant Memory**
  Per read/write work-items
  resides in off-chip DRRx memory
- **Host Memory**
  On the host CPU
Chapter 2. OpenCL with Altera SDK

Overview of OpenCL
FPGA OpenCL Architecture

The SoC platform resembles the traditional OpenCL model with a shared global memory that is used to pass data between the ARM host and the FPGA accelerator.
Chapter 2. OpenCL with Altera SDK

Kernel Compiler

- **C-language front end**: parses a kernel description and creates an LLVM Intermediate Representation (IR)
- **Live-value analysis**: identifies variables consumed and produced by each basic block.
- **CDFG Generation**: create a Control-Data Flow Graph (CDFG) to represent the operations inside basic block,
- **Scheduling**: to determine the clock cycles in each operation is performed.
- **Hardware generation**: To generate a hardware circuit for a kernel
Chapter 2. OpenCL with Altera SDK

OpenCL Host Program

1. Query host for OpenCL devices
2. Create a context to associate OpenCL devices
3. Create programs for execution on one or more associated devices
4. Select kernels to execute from the programs
5. Create memory objects accessible from the host and/or the device
6. Copy memory data to the device as needed
7. Provide kernels to command queue for execution
8. Copy results from the device to the host
OpenCL Kernels

- Data-parallel function
  - Defines many parallel threads of execution
  - Each thread has an identifier specified by "get_global_id"
  - Contains keyword extensions to specify parallelism and memory hierarchy
- Executed by OpenCL device
  - CPU
  - GPU
  - Accelerator

```cpp
__kernel void sum(__global const float *a,
                  __global const float *b,
                  __global float *answer)
{
    int xid = get_global_id(0);
    answer[xid] = a[xid] + b[xid];
}
```
Chapter 3. Matrix-multiplication Tiling

\[ C = A \times B \]

\[ \text{sub-}C = \text{sub-}A \times \text{sub-}B \]

\[ \text{sub-}C = (\text{sub-}A1 \times \text{sub-}B1) + (\text{sub-}A2 \times \text{sub-}B2) \]

- Implementation doesn't perform so well because the accessing of kernel device to off-chip memory.

- Tiling saves time accessing of kernel device to off-chip memory.
main candidates for the kernels are the lines 1, 2, 3, 4, 5, 6, 7 and 8 of algorithm.

Each kernel would have a similar structure based on a matrix multiplication.

the better solution exist is implement a unique kernel and reuse 7 times in each iteration.

matrix-multiplications of forward phase(1, 2 and 3) backward1 phase(5 and 7) backward2 phase(4, 6 and 8)
Chapter 5. Integrate IPs in OpenCL and Results

To create an OpenCL library we need:

- **RTL Components**
  - **RTL source file**: Verilog, VHDL.
  - **eXtensible Markup Language (XML)**: The Altera Offline Compiler uses it to integrate RTL components into OpenCL pipeline.
  - **Header file (.h)**: declares the signatures of function(s) that are implemented by the RTL component.
  - **OpenCL emulation model file (.cl)**: Provides a C model for the RTL component that is used only for emulation.

- **OpenCL Functions**
  - **OpenCL source files (.cl)**: Contains definitions of the OpenCL functions.
  - **Header file (.h)**: declares the signatures of function(s) that are defined in the OpenCL source files.
Chapter 5. Integrate IPs in OpenCL and Results

Integration of an RTL Module into the AOCL Pipeline

Parallel Execution Model of AOCL Pipeline Stages

Integration of an RTL Module into an AOCL Pipeline
Chapter 5. Integrate IPs in OpenCL and Results

Function tangent hyperbolic ‘tanh’
Chapter 5. integrate IPs in OpenCL and Results

Study of Backpropagation algorithm

1. ARM-CPU
2. Kernel and Kernel with ‘Tanh’
   1. Block Size = 4 and Work Items = 4
   2. Block Size = 8 and Work Items = 8
   3. Block Size = 16 and Work Items = 4

- numHidden1_MAX = {4, 8, 16, 32, 48, 64}
- numHidden2_MAX = {4, 8, 16, 32, 48, 64}
- numPatterns_MAX = {256, 1024, 4096, 16384, 65536}
- numInputs_max and numOutputs_max are fixed
## Results

### In seconds (s)

<table>
<thead>
<tr>
<th>numPattern s_MAX</th>
<th>256</th>
<th>1024</th>
<th>4096</th>
<th>16384</th>
<th>65536</th>
</tr>
</thead>
<tbody>
<tr>
<td>numHidden 1</td>
<td>16</td>
<td>8</td>
<td>48</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>numHidden 2</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>CPU time</td>
<td>9.72</td>
<td>51.6</td>
<td>411.84</td>
<td>2288.62</td>
<td>9257.73</td>
</tr>
<tr>
<td>Forward time</td>
<td>5.04</td>
<td>15.78</td>
<td>63.02</td>
<td>256.38</td>
<td>1005.28</td>
</tr>
<tr>
<td>Backward time</td>
<td>5.52</td>
<td>35.57</td>
<td>348.63</td>
<td>2031.91</td>
<td>8251.92</td>
</tr>
</tbody>
</table>

### ARM-CPU

<table>
<thead>
<tr>
<th>numPattern s_MAX</th>
<th>256</th>
<th>1024</th>
<th>4096</th>
<th>16384</th>
<th>65536</th>
</tr>
</thead>
<tbody>
<tr>
<td>numHidden 1</td>
<td>16</td>
<td>8</td>
<td>48</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>numHidden 2</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>CPU time</td>
<td>2.78</td>
<td>8.22</td>
<td>30.16</td>
<td>125.72</td>
<td>485.97</td>
</tr>
<tr>
<td>Forward time</td>
<td>0.97</td>
<td>2.96</td>
<td>11</td>
<td>41.82</td>
<td>160.41</td>
</tr>
<tr>
<td>Backward time</td>
<td>1.63</td>
<td>5.04</td>
<td>18.95</td>
<td>83.61</td>
<td>324.99</td>
</tr>
</tbody>
</table>

Kernel, BZ=4 and WI=4

Kernel with ‘tanh’, BZ=4 and WI=4
## Chapter 5. Integrate IPs in OpenCL and Results

### Comparison

#### Hardware

<table>
<thead>
<tr>
<th></th>
<th>ARM-CPU</th>
<th>BZ=4, WI=4</th>
<th>BZ=8, WI=8</th>
<th>BZ=16, WI=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic utilization (in ALMs)</td>
<td>14,328 / 32,070 (45%)</td>
<td>26,978 / 32,070 (84%)</td>
<td>25,536 / 32,070 (80%)</td>
<td></td>
</tr>
<tr>
<td>Total registers</td>
<td>27659</td>
<td>56719</td>
<td>49801</td>
<td></td>
</tr>
<tr>
<td>Total block memory bits</td>
<td>783,992 / 4,065,280 (19%)</td>
<td>961,720 / 4,065,280 (24%)</td>
<td>1,327,024 / 4,065,280 (33%)</td>
<td></td>
</tr>
<tr>
<td>Total DSP Blocks</td>
<td>26 / 87 (30%)</td>
<td>74 / 87 (85%)</td>
<td>74 / 87 (85%)</td>
<td></td>
</tr>
<tr>
<td>HPS Dynamic (Dual core) Power</td>
<td>1392.92 mW</td>
<td>1392.92 mW</td>
<td>1392.92 mW</td>
<td></td>
</tr>
<tr>
<td>Total FPGA and HPS Power</td>
<td>2524.70 mW</td>
<td>3030.51 mW</td>
<td>2980.95 mW</td>
<td></td>
</tr>
</tbody>
</table>

|                  | ARM-CPU | BZ=4, WI=4 | BZ=8, WI=8 | BZ=16, WI=4 |
| Logic utilization (in ALMs) | 15,047 / 32,070 (47%) | 28,327 / 32,070 (88%) | 26,241 / 32,070 (82%) |
| Total registers  | 28690   | 58763      | 50914      |
| Total block memory bits | 783,992 / 4,065,280 (19%) | 961,720 / 4,065,280 (24%) | 1,327,024 / 4,065,280 (33%) |
| Total DSP Blocks | 26 / 87 (30%) | 74 / 87 (85%) | 74 / 87 (85%) |
| HPS Dynamic (Dual core) Power | 1392.92 mW | 1392.92 mW | 1392.92 mW |
| Total FPGA and HPS Power | 2539.87 mW | 3053.78 mW | 2995.10 mW |
Chapter 5. Integrate IPs in OpenCL and Results

Comparison

Kernel vs ARM-CPU
Chapter 5. Integrate IPs in OpenCL and Results

Comparison

Kernel with ‘tanh’ vs ARM-CPU
Conclusion

- For the best case,
  - kernel acceleration Matrix-multiplication with ‘tanh’
  - Work Size = 4
  - Work Items = 4
  - numPatterns_max = 65536
  - numHidden1 = 64
  - numHidden2 = 64

- Speed up
  - 9257.73/485.97 = 19.05 times faster for CPU time
  - 1005.28/160.41 = 6.27 times faster for Forward time
  - 8251.92/324.99 = 25.39 times faster for Backward time
Thank you for your attention &

Any question is welcome