UNIVERSITAT POLITÉCNICA DE VALÉNCIA

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

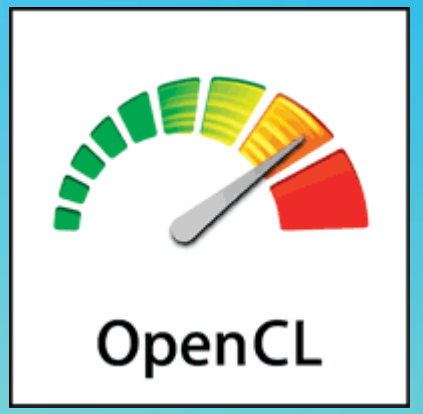
Chapter 4. Backpropagation Algorithm

Chapter 5. integrate IPs in OpenCL and Results

Conclusion

Introduction

OpenCL

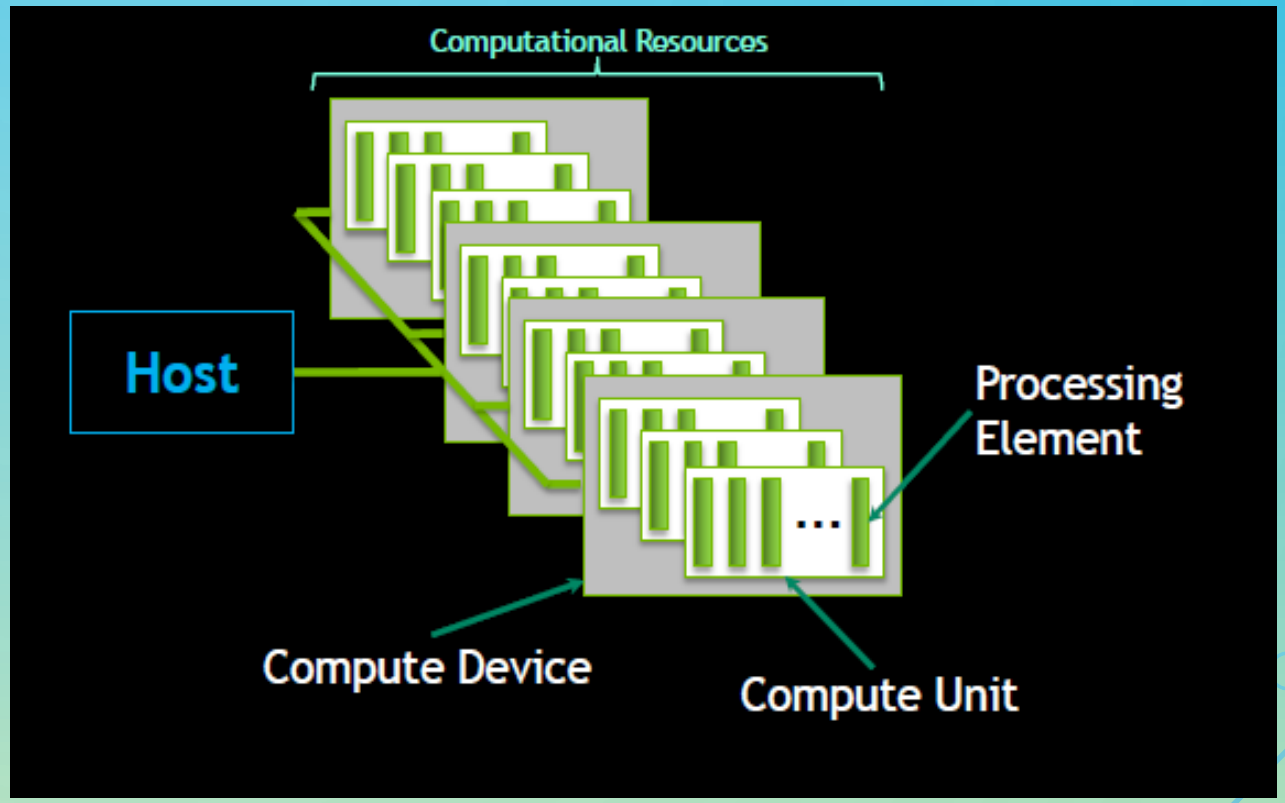


- The open standard for parallel programming of heterogeneous systems
- found in personal computers, servers, mobile devices and embedded platforms
- improves the speed and responsiveness of a wide spectrum of applications in numerous market categories such gaming and medical software

Chapter 1. Introduction to OpenCL

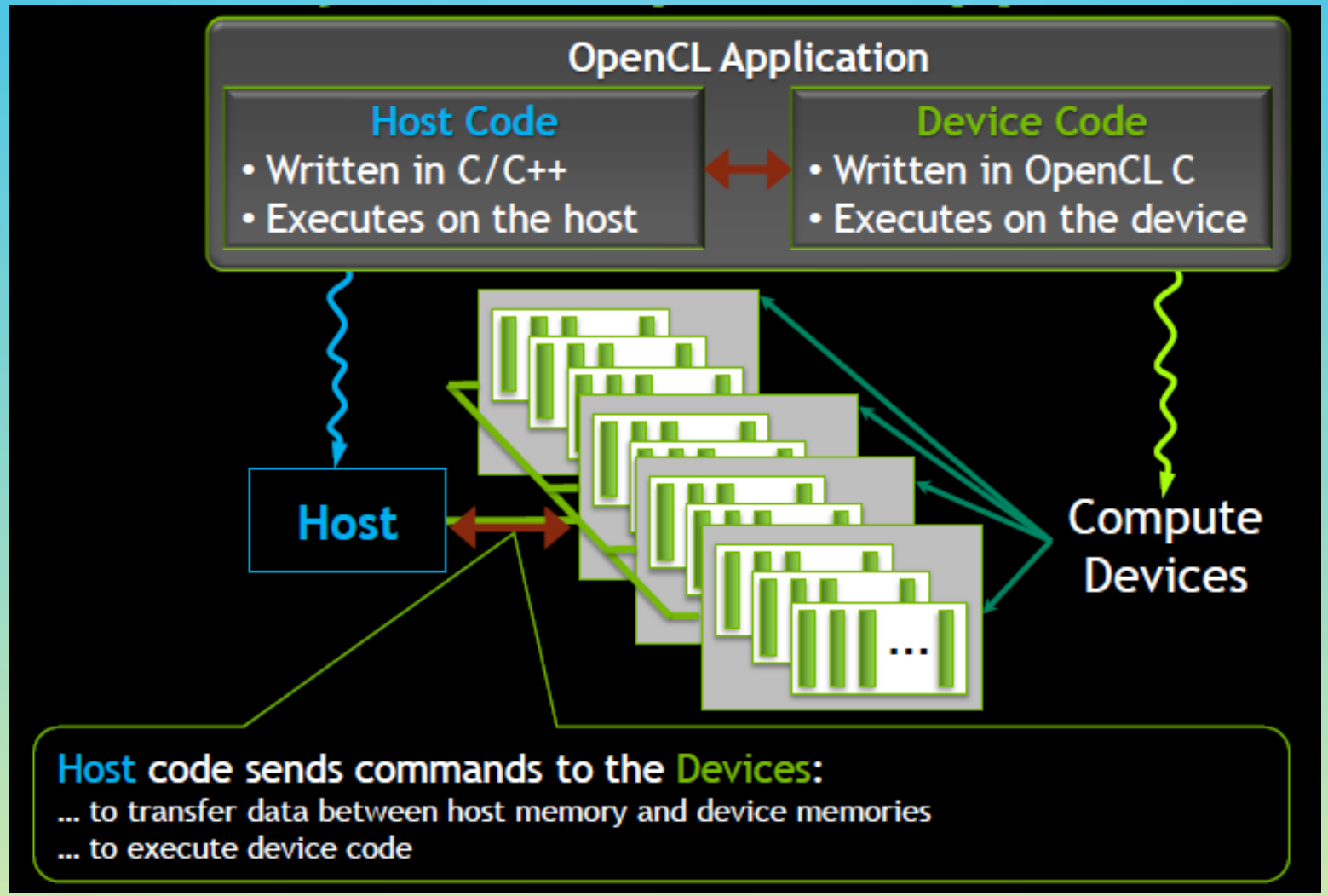
OpenCL Platform Model

- 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



Chapter 1. Introduction to OpenCL

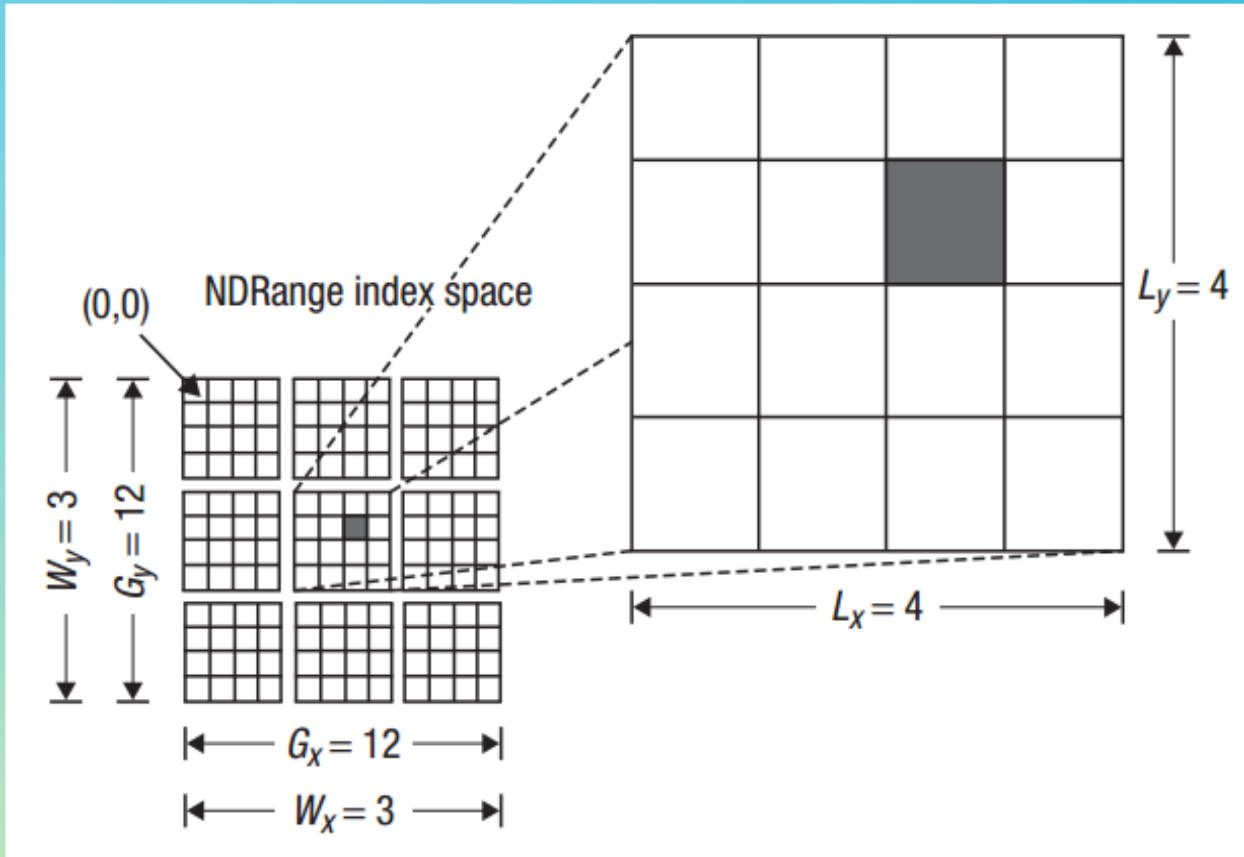
Execution Model

kernels executes 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

NDRange index Space



Chapter 1. Introduction to OpenCL

The BIG Idea behind OpenCL

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

```
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];
}
```

Traditional loop as a function in C

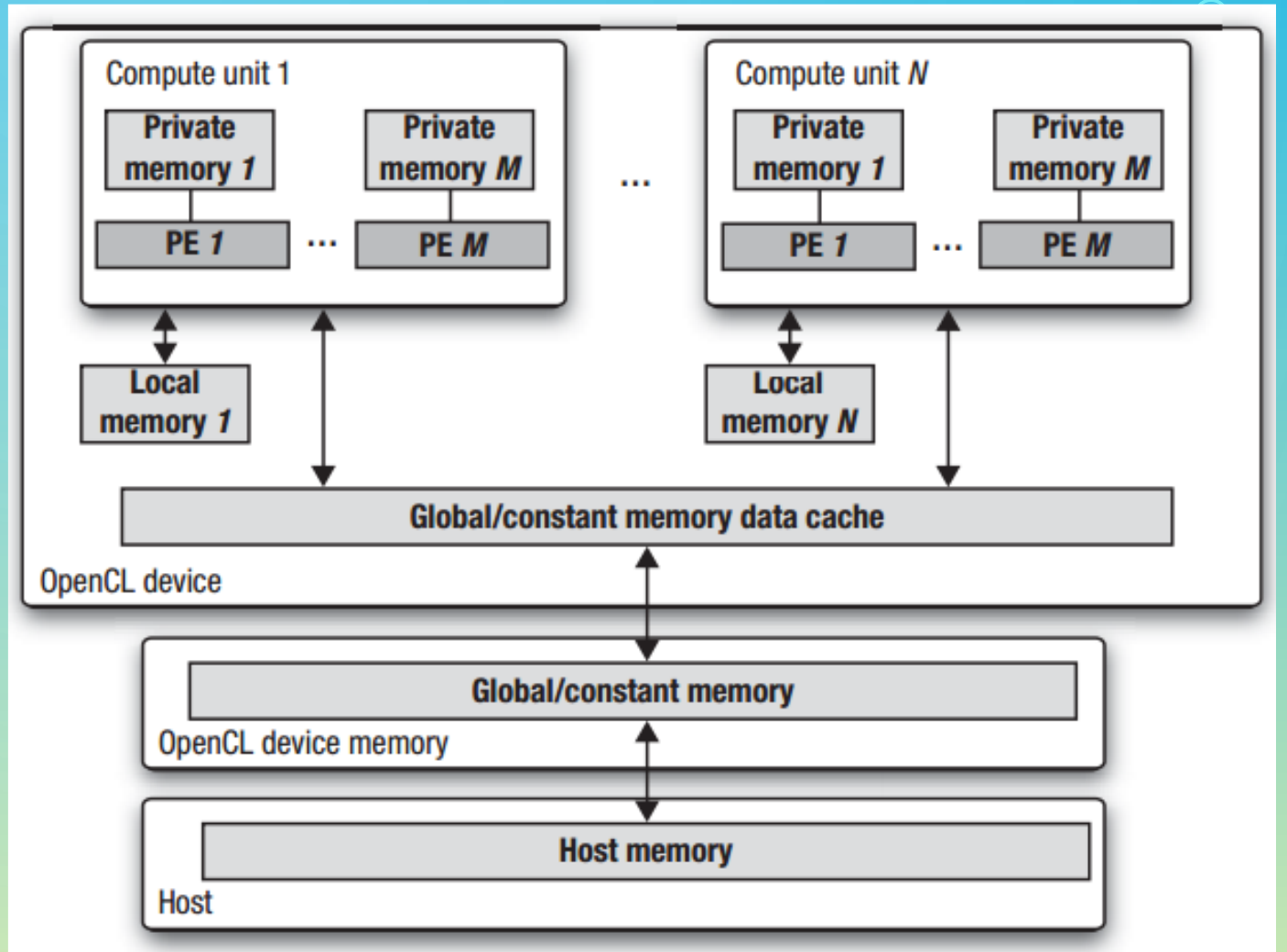
```
__kernel void
dp_mul(__global const float *a,
       __global const float *b,
       __global  float *c)
{
int id = get_global_id(0);
     c[id] = a[id] * b[id];}

  // execute over n"work items"
```

OpenCL C kernel

Chapter 1. Introduction to OpenCL

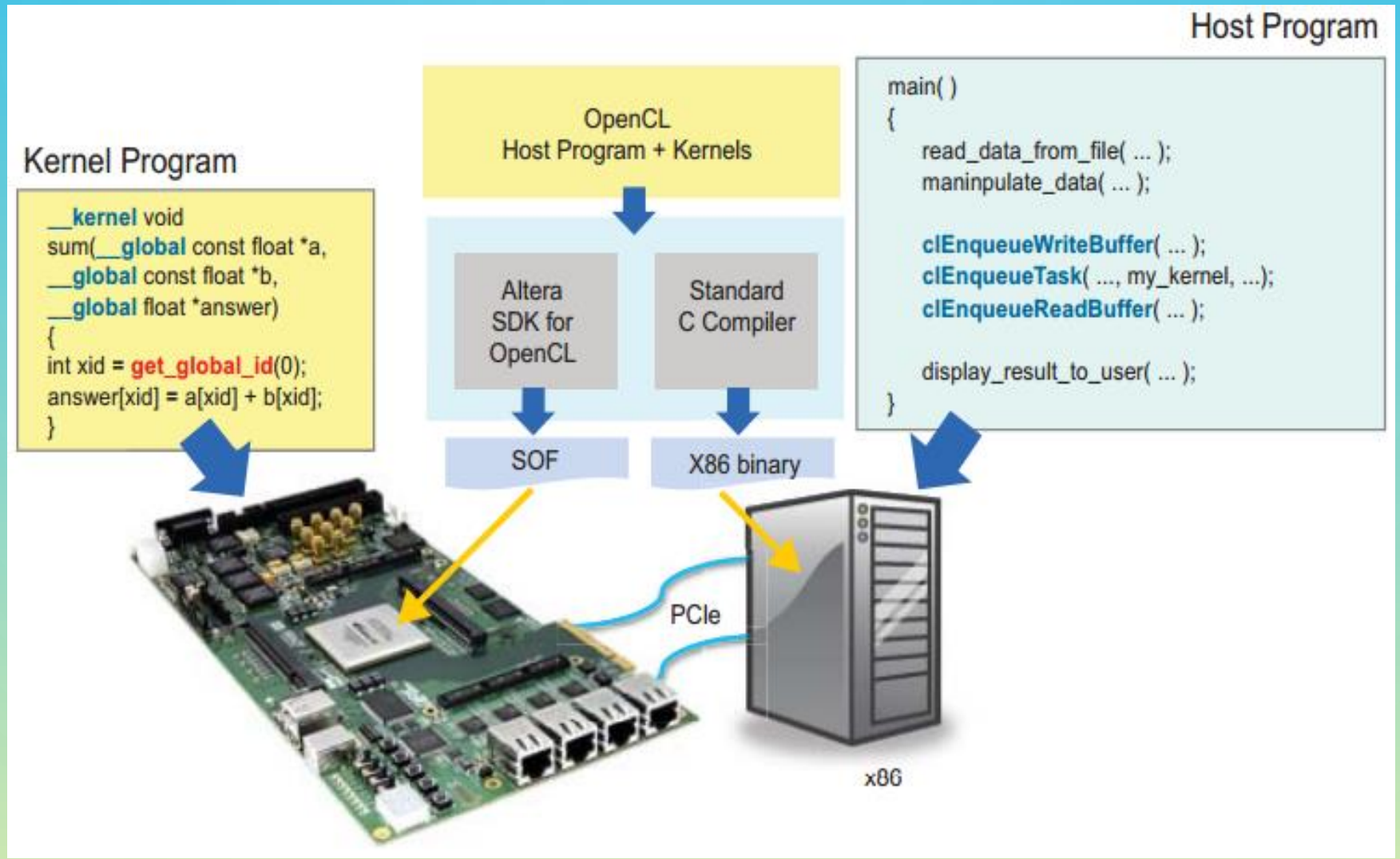
Memory Model

- 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
  is resides in off-chip DRRx memory
- Host Memory
  On the host CPU



Chapter 2. OpenCL with Altera SDK

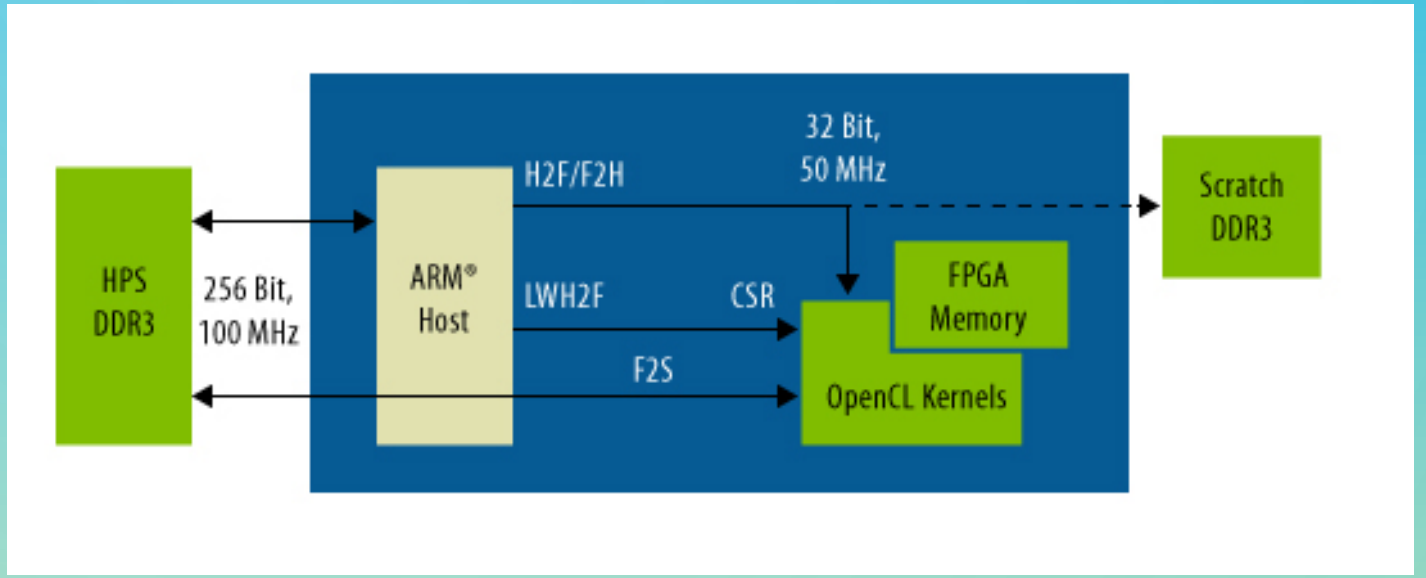
Overview of OpenCL



Chapter 2. OpenCL with Altera SDK

FPGA OpenCL Architecture

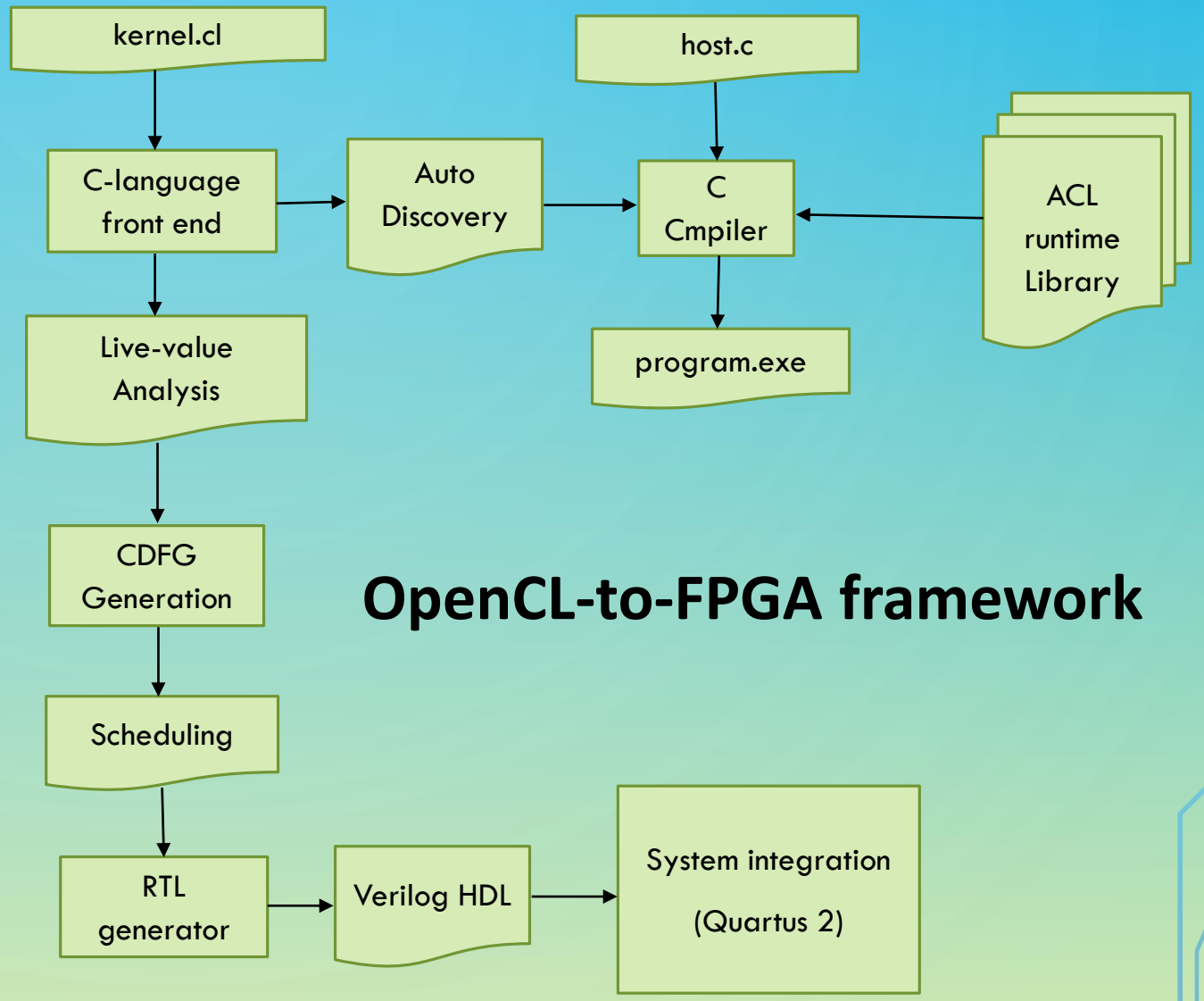
The SoC platform resembles the traditional OpenCL model with a shared global memory that is used to pass data between the ARM host and the FPGA accelerator.



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

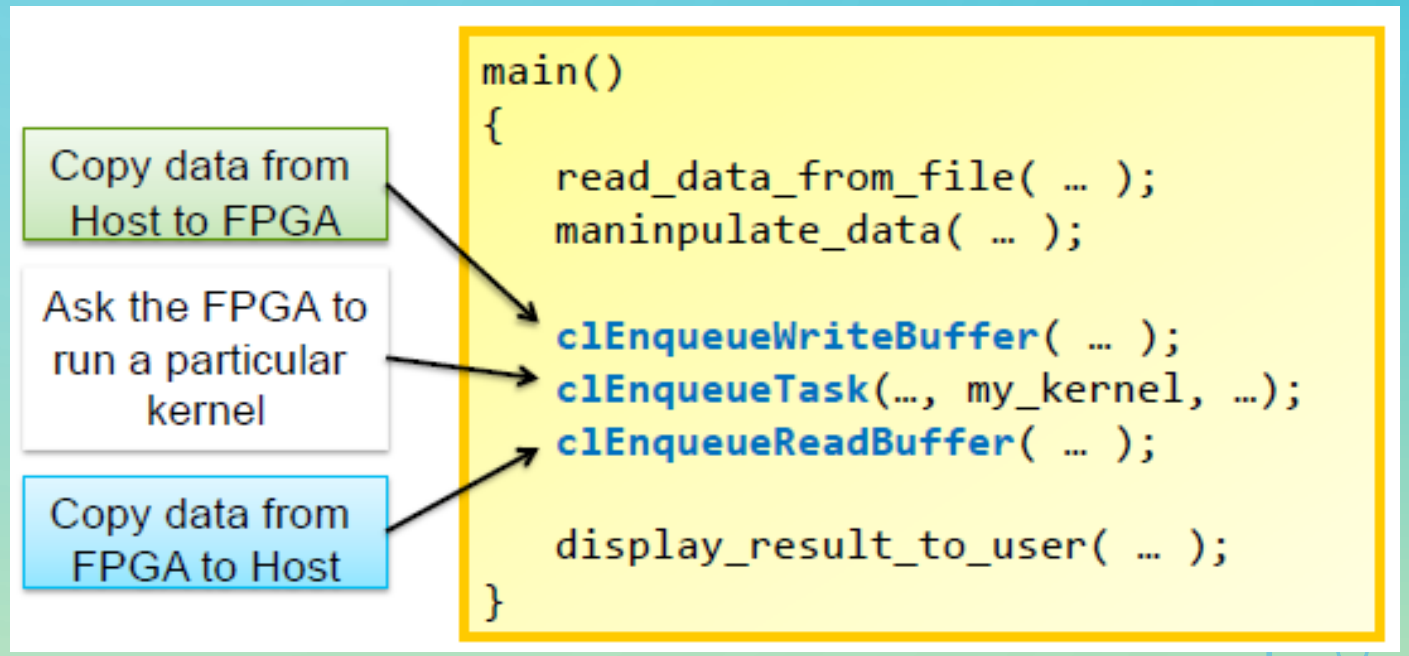


OpenCL-to-FPGA framework

Chapter 2. OpenCL with Altera SDK

OpenCL Host Program

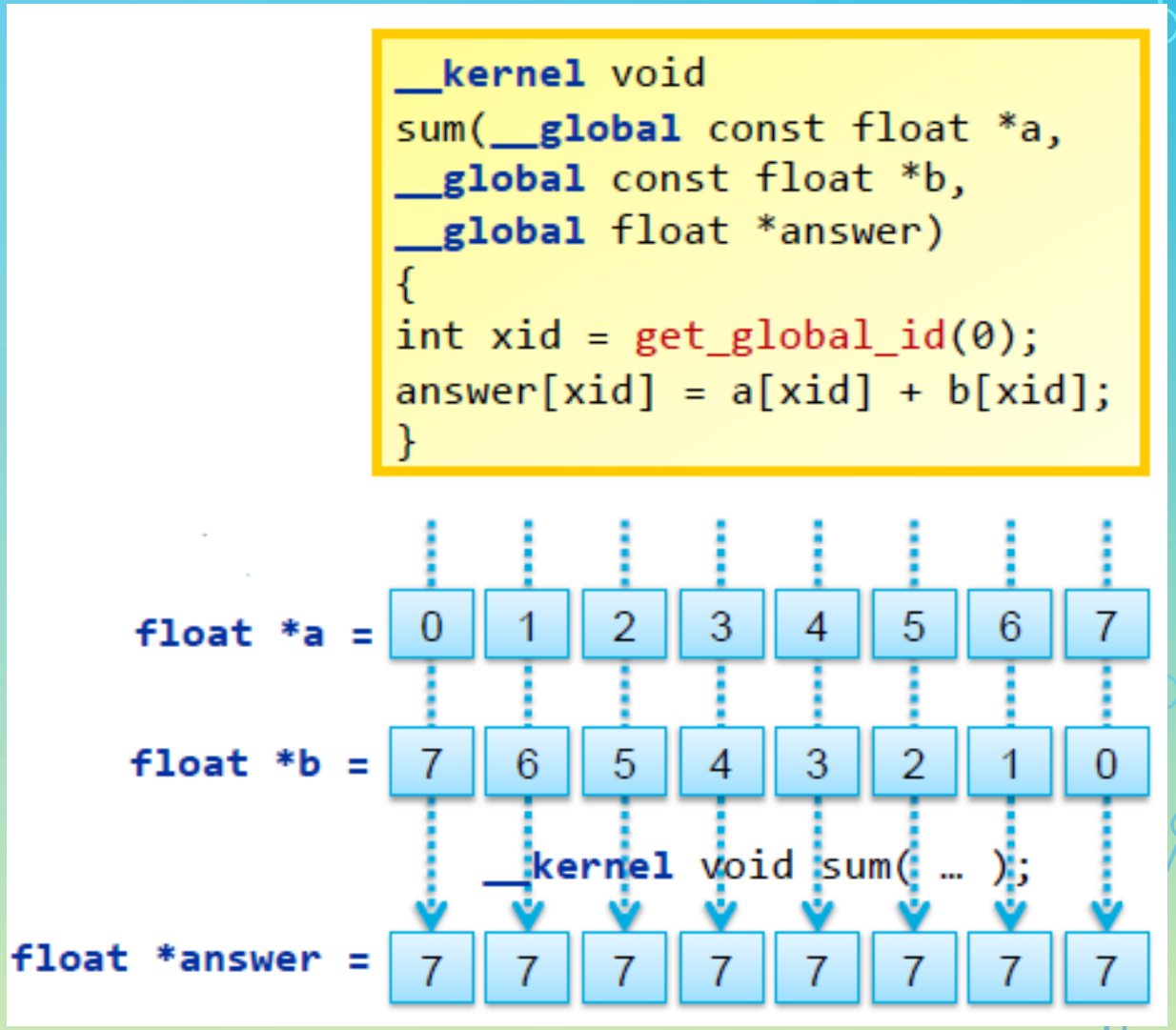
1. Query host for OpenCL devices
2. Create a context to associate OpenCL devices
3. Create programs for execution on one or more associated devices
4. Select kernels to execute from the programs
5. Create memory objects accessible from the host and/or the device
6. Copy memory data to the device as needed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



Chapter 2. OpenCL with Altera SDK

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



Chapter 3. Matrix-multiplication Tiling

C = A * B

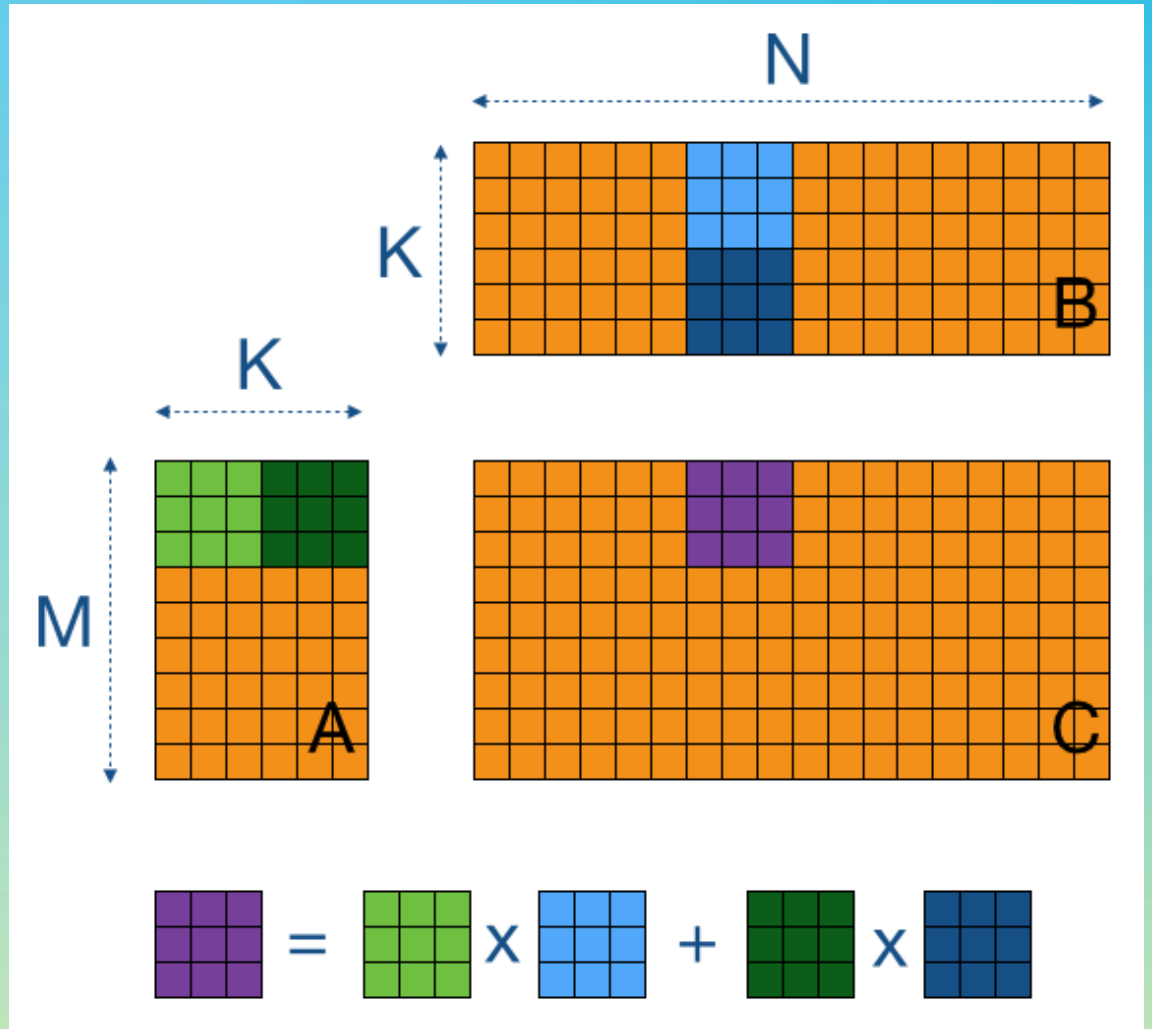
sub-C=sub-A*sub-B

sub-C=(sub-A1*sub-B1)

+(sub-A2*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.



Chapter 4. Backpropagation Algorithm

- 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 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)

input : A training set (Inputs_train,Targets_train) and a test set (Inputs_tets,Targets_test)
output: Number of errors in test set

Initweights (${}^0w^I, {}^0w^J, {}^0w^L$)← 0;
for $i \leftarrow 1$ **to** E **do**
  Initcweights (${}^0cw^I, {}^0cw^J, {}^0cw^L$)← 0;
1 $u^I \leftarrow$ Forward_layer1($^{i-1}w^I, Inputs\_train$);
  $y^I \leftarrow$ Funct_nolineal(u^I);
  $y'^I \leftarrow$ der_Funct_nolineal(u^I);
2 $u^J \leftarrow$ Forward_layer2($^{i-1}w^J, y^I$);
  $y^J \leftarrow$ Funct_nolineal(u^J);
  $y'^J \leftarrow$ der_Funct_nolineal(u^J);
3 $u^L \leftarrow$ Forward_layer3($^{i-1}w^L, y^J$);
  $y^L \leftarrow$ Funct_nolineal(u^L);
  $y'^L \leftarrow$ der_Funct_nolineal(u^L);
  $\delta^L \leftarrow$ Backward1_layer3($Targets\_train, y^L, y'^L$);
4 $cw^L \leftarrow$ Backward2_layer3(δ^L, y^J, cw^L);
5 $^j\theta^J \leftarrow$ Backward1_layer2($^{i-1}w^L, \delta^L$);
  $\delta^J \leftarrow {}^j\theta^J \cdot y'^J$;
6 $cw^J \leftarrow$ Backward2_layer2(δ^J, y^I, cw^J);
7 $^j\theta^I \leftarrow$ Backward1_layer1($^{i-1}w^J, \delta^J$);
  $\delta^I \leftarrow {}^j\theta^I \cdot y'^I$;
8 $cw^I \leftarrow$ Backward2_layer1($\delta^I, Inputs\_train, cw^I$);
  $^iw^L \leftarrow$ Update_layer3($^{i-1}w^L, cw^L$);
  $^iw^J \leftarrow$ Update_layer2($^{i-1}w^J, cw^J$);
  $^iw^I \leftarrow$ Update_layer1($^{i-1}w^I, cw^I$);
Fitness← TestNeuralNetwork($Inputs\_test, Targets\_test, {}^Ew^I, {}^Ew^J, {}^Ew^L$);

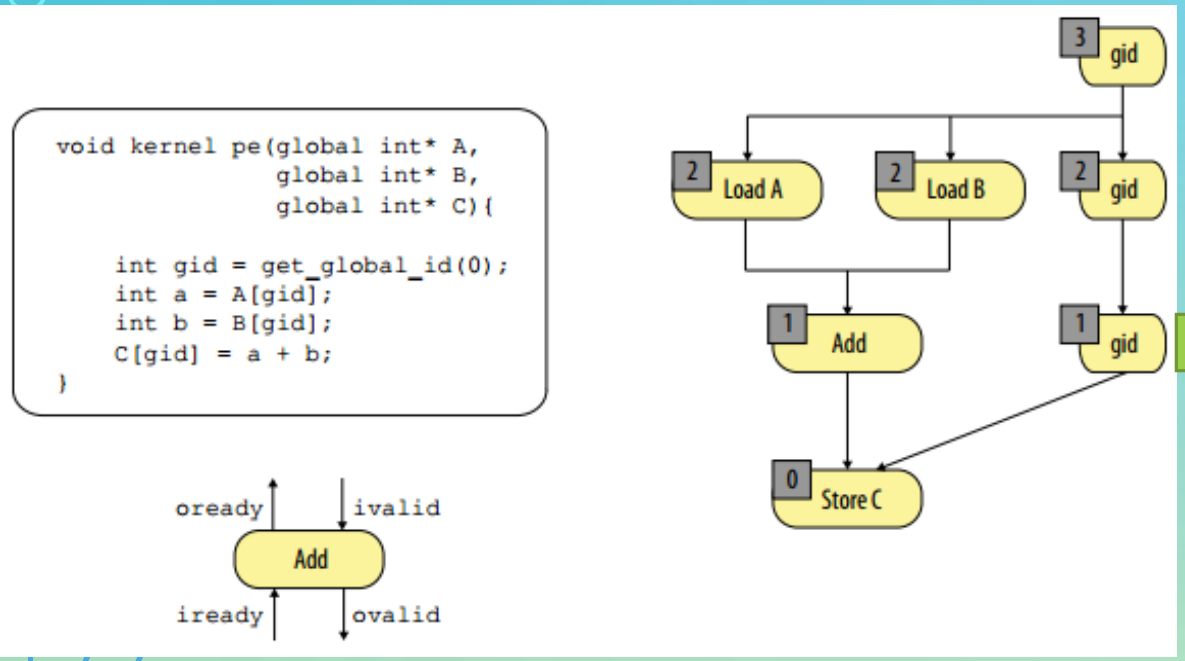
Chapter 5. integrate IPs in OpenCL and Results

To create an OpenCL library we need

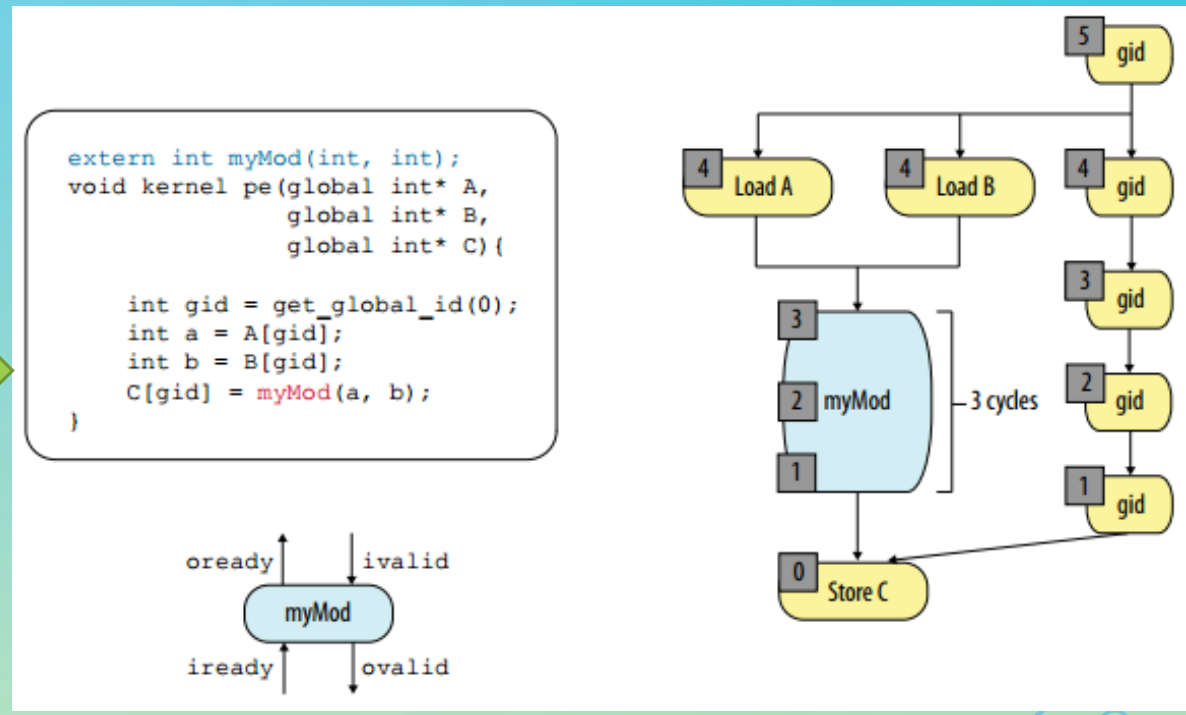
- RTL Components
  - **RTL source file:** Verilog, VHDL.
  - **eXtensible Markup Language File (.xml):** The Altera Offline Compiler uses it to integrate RTL components into OpenCL pipeline.
  - **Header file (.h):**declares the signatures of function(s) that are implement by the RTL component
  - **OpenCL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



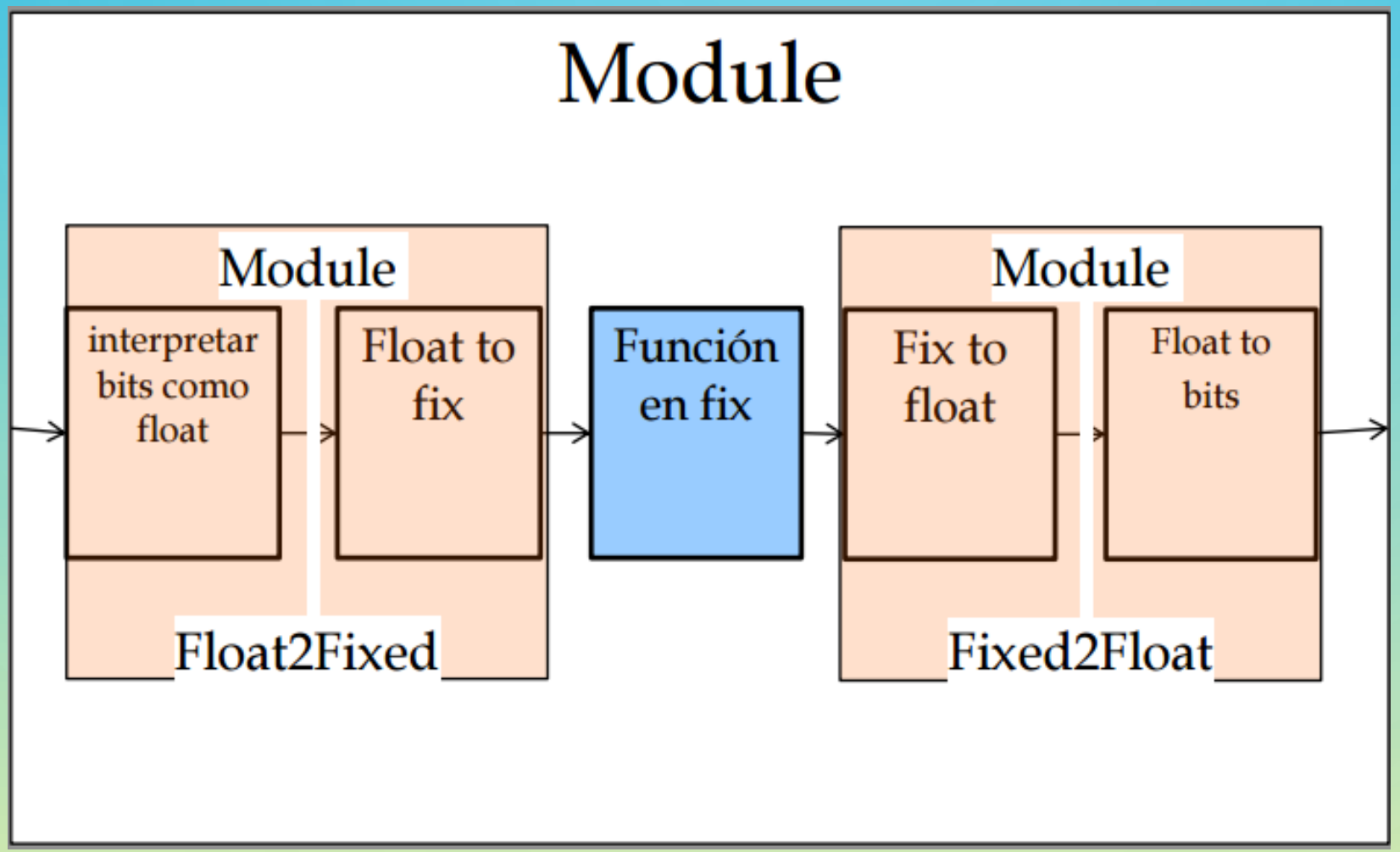
Parallel Execution Model of AOCL Pipeline Stages



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 Backprobagation 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

Chapter 5. integrate IPs in OpenCL and Results

Results

In seconds (s)

numPatterns_MAX	256	1024	4096	16384	65536
numHidden 1	16	8	48	32	64
numHidden 2	64	64	64	64	64
CPU time	9,72	51,6	411,84	2288,62	9257,73
Forward time	5,04	15,78	63,02	256,38	1005,28
Backward time	5,52	35,57	348,63	2031,91	8251,92

ARM-CPU

numPatterns_MAX	256	1024	4096	16384	65536
numHidden 1	64	64	64	64	64
numHidden 2	64	64	64	64	64
CPU time	4,95	16,86	64,8	263,59	1048,06
Forward time	3,2	11,69	45,86	179,74	717,26
Backward time	1,51	4,89	18,68	83,56	330,18

Kernel, BZ=4 and WI=4

numPatterns_MAX	256	1024	4096	16384	65536
numHidden 1	16	8	48	32	64
numHidden 2	64	64	64	64	64
CPU time	2,78	8,22	30,16	125,72	485,97
Forward time	0,97	2,96	11	41,82	160,41
Backward time	1,63	5,04	18,95	83,61	324,99

Kernel with ‘tanh’, BZ=4 and WI=4

Chapter 5. integrate IPs in OpenCL and Results

Comparison

Hardware

	ARM-CPU	BZ=4, WI=4	BZ=8, WI=8	BZ=16,WI=4
Logic utilization (in ALMs)		14,328 / 32,070 (45 %)	26,978 / 32,070 (84%)	25,536 / 32,070 (80 %)
Total registers		27659	56719	49801
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	1392.92 mW
Total FPGA and HPS Power		2524.70 mW	3030.51 mW	2980.95 mW

Hardware Comparison, Just a Kernel

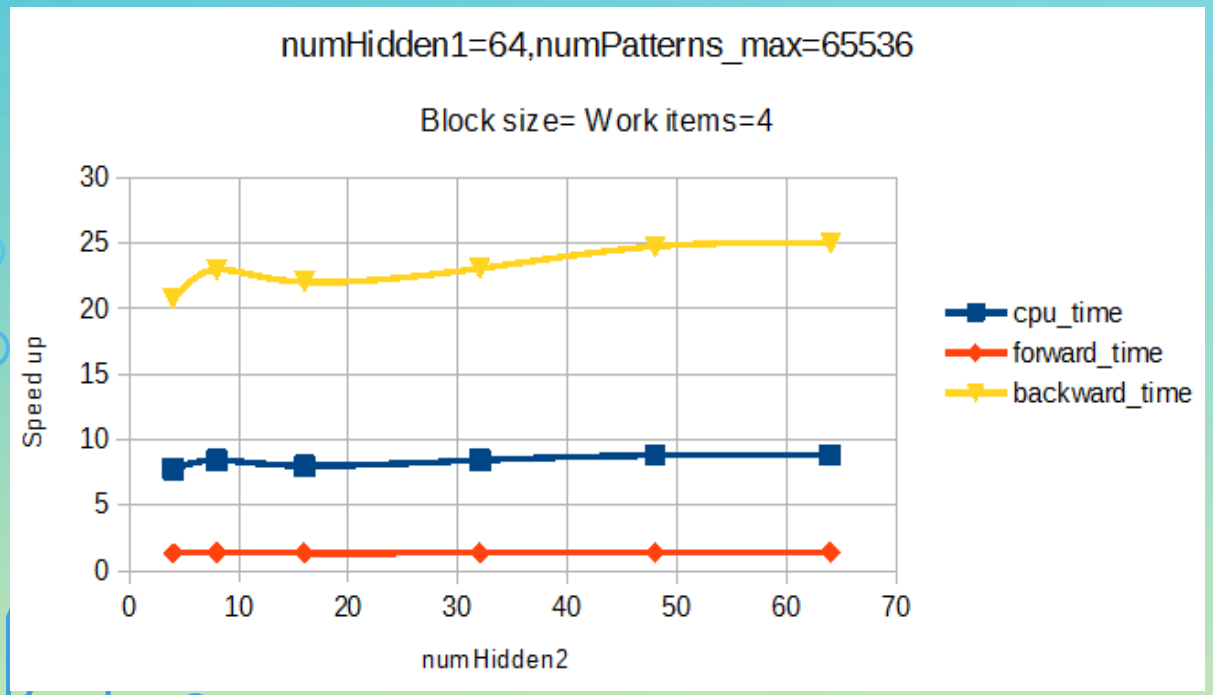
	ARM-CPU	BZ=4, WI=4	BZ=8, WI=8	BZ=16,WI=4
Logic utilization (in ALMs)		15,047 / 32,070 (47 %)	28,327 / 32,070 (88%)	26,241 / 32,070 (82 %)
Total registers		28690	58763	50914
Total block memory bits		783,992 / 4,065,280 (19%)	961,720 / 4,065,280 (24%)	1,327,024 / 4,065,280 (33 %)
Total DSP Blocks		26/87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

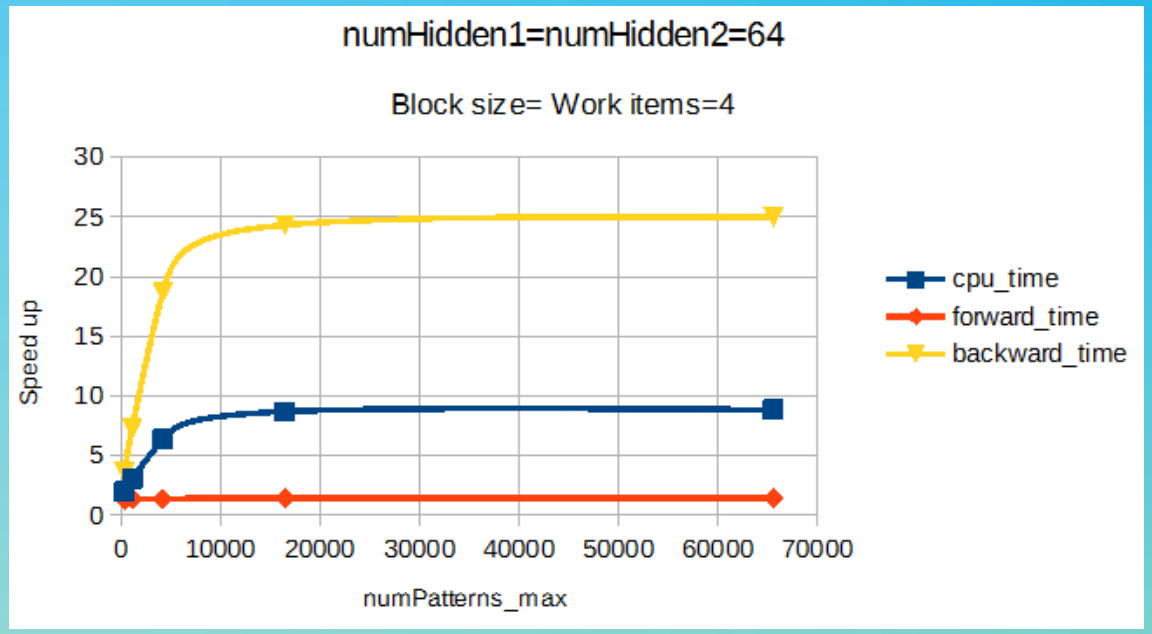
Hardware Comparison, Kernel with 'tanh'

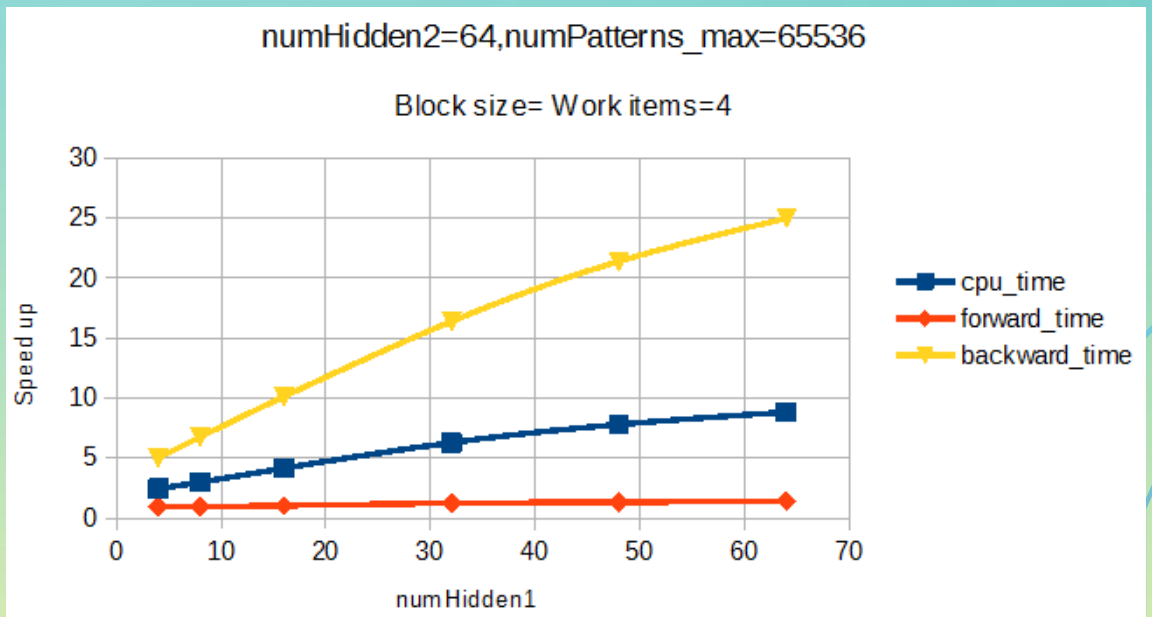
Chapter 5. integrate IPs in OpenCL and Results

Comparison

Kernel vs ARM-CPU



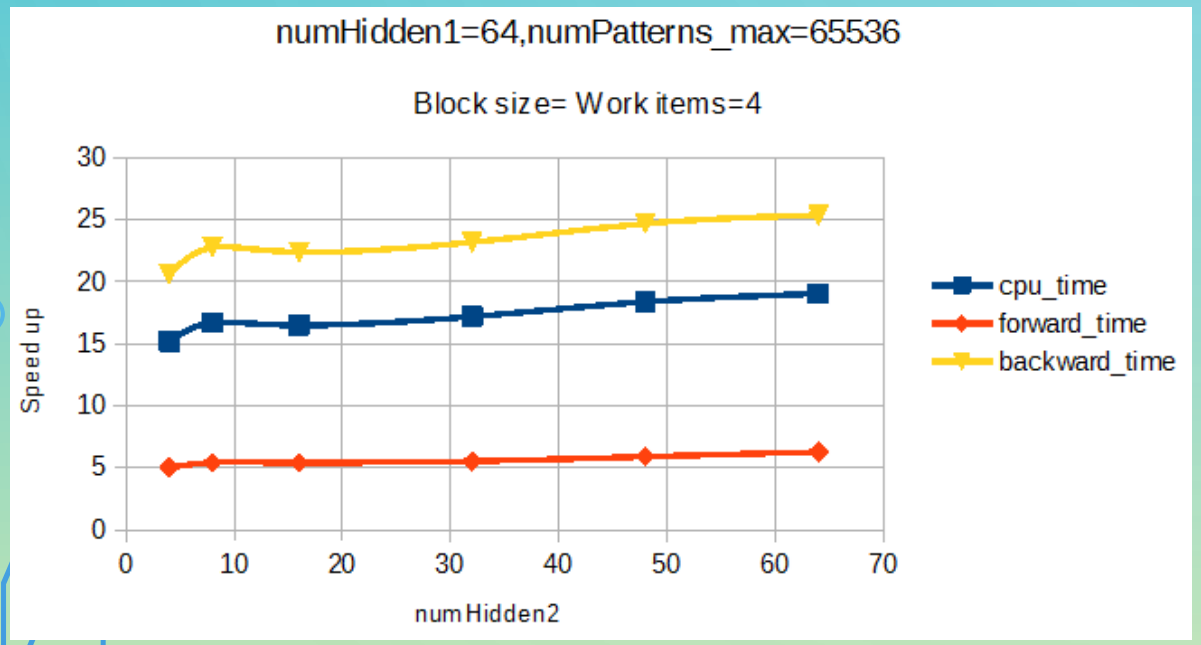


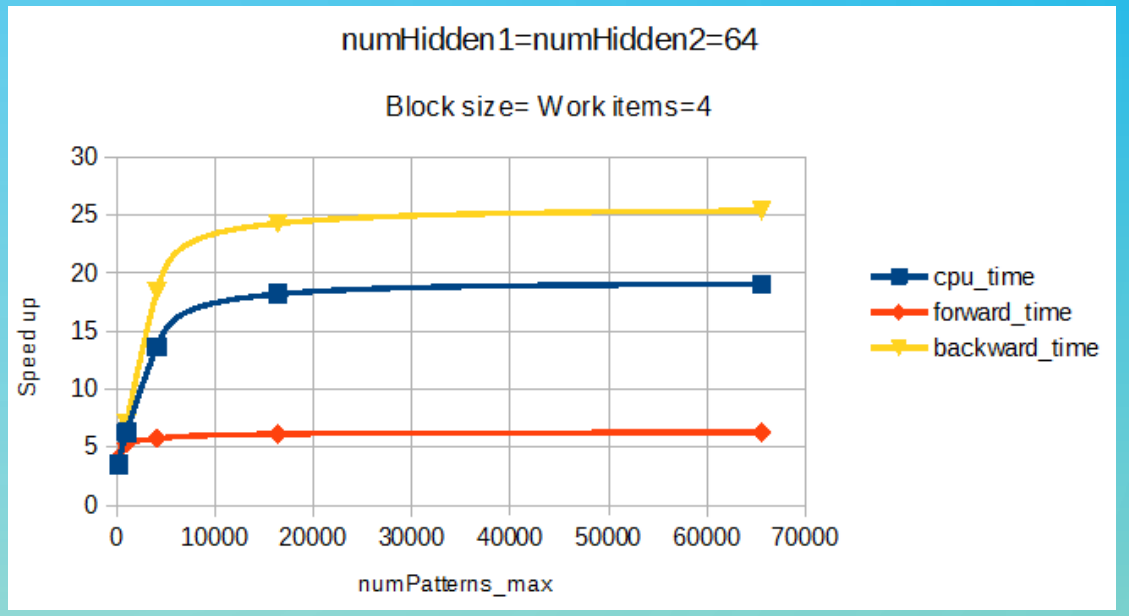


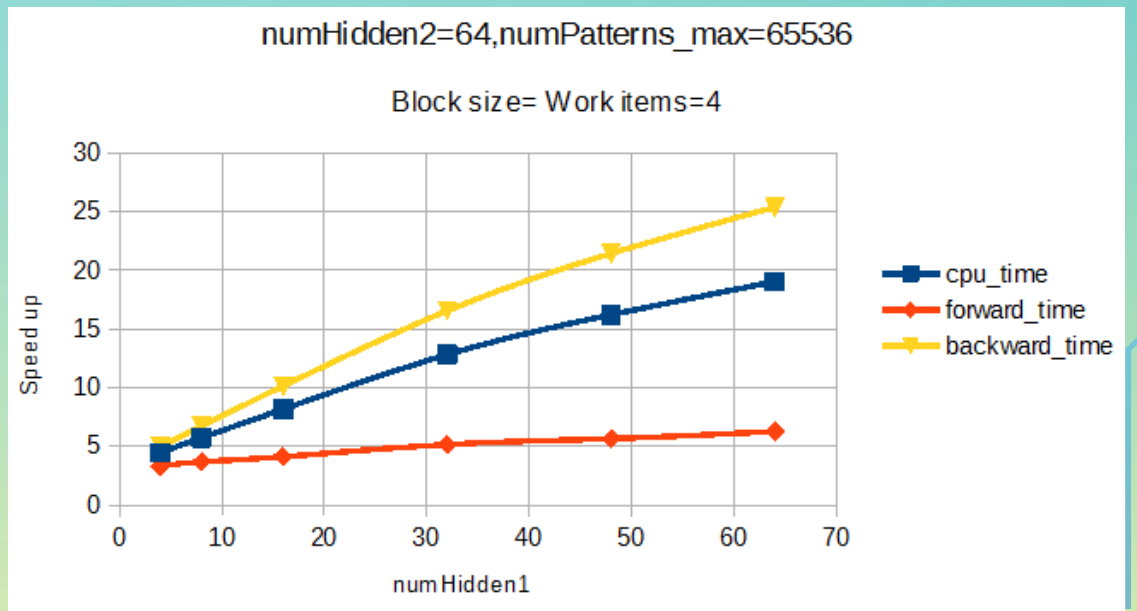
Chapter 5. integrate IPs in OpenCL and Results

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