HandsOn Training

Intel® FPGA Technical Training

Parallel Computing with OpenCL for Intel FPGAs

Course Description

OpenCL is a standard for writing parallel programs for heterogeneous systems. In the FPGA environment, OpenCL constructs are synthesized into custom logic. This course introduces the basic concepts of parallel computing. It covers the constructs of the OpenCL standard & Intel flow that automatically converts kernel C code into hardware that interacts with the host.

In hands-on labs, you'll write programs to run on both the CPU & FPGA.

In addition, the course covers the optimization techniques needed to implement a high performance OpenCL solution on an FPGA using the Intel SDK for OpenCL. We will discuss good coding design practices, ways to improve data processing efficiency, memory access efficiencies, and host side optimizations. We will also focus on Intel SDK for OpenCL specific features that can significantly improve OpenCL performance on FPGAs compared to other platforms.

Course Duration

3 days

Goals

1. Describe high-level parallel computing concepts and challenges
2. Understand the advantages of using Intel's OpenCL solution
3. Know the basics of the OpenCL standard
4. Write simple programs in OpenCL
5. Compile and run OpenCL programs using the Intel solution
6. Use good coding design practices to implement a high performance OpenCL FPGA system
7. Apply multiple methods to improve data processing efficiency of OpenCL kernels
8. Reduce host-device communication overhead by improving memory access efficiencies and using channels
9. Understand Intel specific features that will significantly boost the performance of an OpenCL solution

Intended Users

Hardware and software engineers who develop and work with parallel algorithms and computationally intense applications for Intel FPGAs

Previous Knowledge

FPGA design, VHDL/Verilog, basic understanding of the C programming language

Course Material

1. Synthesizer and Place & Route: Quartus Prime
2. Microsoft® Visual C++® 2010 (Express)
3. Windows® SDK 7.1
4. Intel® SDK for OpenCL™ Application 2012
5. Course book (including labs)

Table of Contents

Day #1

- **The OpenCL Standard**
  - OpenCL definition
  - OpenCL characteristics
  - OpenCL history
  - Versions of OpenCL standard
  - OpenCL specification
  - OpenCL heterogeneous platform model
  - OpenCL device
  - Heterogeneous platform model: GPGPU
  - Heterogeneous platform model: FPGA
  - OpenCL execution model
  - Data and task parallelism
  - OpenCL properties
  - Host managed OpenCL objects
  - OpenCL flow
  - OpenCL APIs
  - Program set up
  - Query and select the platform
  - Platform profile
  - OpenCL platform layer
  - Context creation
  - OpenCL command queue
  - Queue creation API
  - Events
  - Memory objects
  - Buffers
  - Memory management flags
  - Physical memory space
  - Data transfers
  - Error management
- **Lab #1: Setting Up OpenCL Host-Side Application**

- **Writing OpenCL Programs**
  - Data parallelism
  - Kernels
  - Work-item
  - Example Kernel
  - OpenCL work-item hierarchy

  - NDRange
  - Workgroup
  - The OpenCL programming model
  - Programs and kernels
  - OpenCL API: creating a program
  - Creating programs from Binary for FPGAs
  - Building programs
  - Creating Kernels
  - Setting up Kernel argument list in C
  - OpenCL Kernel launch
  - Writing Kernels
  - Kernel restrictions
  - Identifying work-items
  - Data types
  - Synchronization
  - Memory model
  - Mapping OpenCL memory to FPGAs
  - OpenCL memory qualifiers
  - Clean up memory
- **Lab #2: Writing a Simple Kernel**

Day #2

- **Compiling for Intel FPGAs**
  - Compiling OpenCL Kernel to FPGAs
  - aoc output files
  - Kernel mapping to FPGA architecture
  - Generated FPGA system
  - FPGA architecture for OpenCL implementation
  - OpenCL Kernels to dataflow circuits
  - Standardized basic block interface
  - Inside of a basic block
  - Mapping multithreaded Kernels to FPGAs
  - The Intel SDK for OpenCL overview
  - Hardware requirements using preferred boards
  - Custom platforms
  - SoC platforms

- Software requirements
- Intel SDK for OpenCL requirements
- SDK compile flow
- Compiling the host program
- Compiling Kernels with AOC
- One-step AOC compilation
- Multi-step AOC compilation
- Compilation reports
- Optimization report
- Emulating an OpenCL Kernel steps
- Emulator features and limitations
- Rapid recompile
- Profiler
- Collecting and viewing profile information
- Profiler reports
- Specify a target FPGA board
- Intel offline compiler options
- Host runtime limits
- SDK functionality
- Kernel language support
- Currently unsupported Kernel language features
- Intel SDK for OpenCL design flow
- Setup considerations
- Execution considerations
- Acceleration using multiple SPMD engines
- Acceleration with multiple queues (SMT)
- Future Intel SDK for OpenCL enhancements

- **Lab #3: Compiling and Running the Program on FPGA**

- **Optimization Overview**
  - Accelerator optimization metrics
  - Amdahl's law
  - Goals of OpenCL optimization

- **Acceleration Through Parallelization**
  - Review of OpenCL terminologies
  - Parallelization goals
  - Data parallelism in OpenCL

- Acceleration by increasing data parallel resources
- OpenCL attributes and directives
- Kernel pragmas and attributes for parallelization
- Query workgroup size constraints from host
- Specifying workgroup size attributes
- Default workgroup size limits and runtime size
- Workgroup recommendations
- Loop unrolling
- Loop unroll restrictions and recommendations
- Kernel vectorization
- Vectorize Kernel code manually
- Attribute for automatic Kernel vectorization
- Coalesce memory accesses
- Multiple compute units
- Compute units Kernel attribute
- Compute unit replication vs SIMD vectorization
- Compute unit replication vs SIMD memory considerations
- SIMD vectorization limitations
- Combining compute unit replication and SIMD vectorization
- Automatic resource-driven optimization
- Resource-driven optimizer behavior
- Optimizer output
- Optimizer limitations
- Optimizer recommendation
- Operating FPGAs in parallel
- OpenCL task-parallel execution model
- Acceleration with multiple queues
- SMT execution model
- Synchronization points
- Asynchronous execution vs host
- clFlush and clFinish
- Blocking memory operation
- Events
- Pipelined execution
- Profiling with events
- Single work-item execution
- Parallelizing algorithms with iteration dependency
- High-throughput task-based Kernel execution
- Loop pipelining
- Data-parallel threads vs loop pipelining
- Advantages of loop pipelining
- Single work-item execution recommendations

- Recognition of task Kernels
- Launching tasks
- Task programming guidelines

- **Lab #4: Optimization using Data Parallel Techniques**

Day #3

- **Reducing Communication Latency**
  - Reducing communication latency goals
  - Need to optimize memory accesses
  - OpenCL memory model review
  - Minimize global memory accesses
  - General guidelines
  - Local memory
  - Memory access synchronization
  - Memory fences
  - Barriers
  - Caching example
  - Workgroup boundary and caching
  - Local memory access guidelines
  - Local memory Kernel argument allocation
  - Using private memory
  - Aligned memory allocation
  - Alignment of data structures
  - Using __constant buffers
  - Constant address space qualifiers
  - Using __constant parameters from the host
  - Heterogeneous memory buffers
  - Specify global memory type
  - Host code for heterogeneous memory accesses
  - AOC default optimizations
  - Contiguous memory accesses
  - Array indexing and contiguous memory accesses
  - Manually partition global memory
  - Manually partition heterogeneous global memory
  - Reducing communication latency
  - Channels

  - Kernel-to-Kernel channel performance gains
  - IO channel performance gains
  - Enabling channels and channel handle declaration
  - Blocking channel reads and writes
  - Non-blocking channel reads and writes
  - Channel data behavior
  - Unbuffered vs buffered channels
  - Channel call ordering
  - IO channels
  - Channel interaction with work-items
  - Current restrictions
  - Channel example application
- **Lab #5: Memory Optimization**
- **Lab #6: Kernel-to-Kernel Channels**

- **Good Design Practices**
  - Floating point optimizations
  - Automatic order of operation rules
  - Tree balancing
  - Rounding operations
  - Reducing rounding operations
  - Floating-point vs fixed-point representation
  - Automatic operation considerations
  - Using Intel offline compiler
  - Data type considerations
  - Avoid pointer aliasing
  - Avoid expensive functions
  - Inexpensive functions
  - Avoid work-item ID-dependent backward branching
  - Host machine memory requirements
  - Host side restrictions
  - Using printf
  - Maintain similar structures for vector type elements
  - Tips for GPU programmers