Spotlight on accelerators

Bill Jenkins
Senior Product Specialist for Programming Language Solutions, Altera
System Acceleration Overview

Bill Jenkins
Altera Sr. Product Specialist for Programming Language Solutions
Industry Trends

- Increasing product functionality and performance
- Smaller time-to-market window
- Shorter product lifecycle
- Limited design resources

© 2015 Altera Corporation—Public
“A form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently (in parallel)”

# Need for Parallel Computing

CPUs aren’t getting faster

<table>
<thead>
<tr>
<th>Information</th>
<th>Memory Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="Diagram" /></td>
<td>Memory architectures have limited bandwidth, and can’t keep up with the processor. DRAM cores are slow!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Realization (cost)</th>
<th>Power Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="Diagram" /></td>
<td>Process scaling (28nm/20nm/14nm) trends towards exponentially increasing power consumption.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computation</th>
<th>ILP Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="Diagram" /></td>
<td>Compilers don’t find enough parallelism in a single instruction stream to keep a Von Neuman architecture busy.</td>
</tr>
</tbody>
</table>

**Performance now come from parallelism**
Challenges in Parallel Programming

Finding Parallelism
- What activities can be executed concurrently?
  - Is parallelism explicit (programmer specified) or implicit?

Data sharing and synchronization
- What happens if two activities access the same data at the same time?
  - Hardware design implications
    - eg. Uniform address spaces, cache coherency

Applications exhibit different behaviors
- Control
  - Searching, parsing, etc…
- Data intensive
  - Image processing, data mining, etc…
- Compute intensive
  - Iterative methods, financial modeling, etc…
Finding Parallelism

- **Scatter-gather**
  - Separate input data into subsets that are sent to each parallel resource and then combine the results
  - Data parallelism

- **Divide-and-conquer**
  - Decompose problem into sub-problems that run well on available compute resources
  - Task parallelism

- **Pipeline Parallelism**
  - Task parallelism where tasks have a producer consumer relationship
  - Different tasks operate in parallel on different data
Granularity of Parallelism

- Ratio of computation to communication
- Fine-grain vs. Coarse-grain
- Most efficient granularity heavily depends on application and hardware environment
Fine-grained parallelism

- Low computation intensity
- Small task size
- Data transferred frequently

Benefits
- Easy to load balance among task

Drawback
- Synchronization and communication overhead can overshadow benefits of parallelism

Example
- GPU threads
- Instruction level parallelism
Coarse-grained parallelism

- High arithmetic intensity
- Data transferred infrequently

Benefit
- Low overhead

Drawback
- Difficult to load balance

Example
- Threads running on different CPUs
Programmers Dilemma

“The way the processor industry is going, is to add more and more cores, but nobody knows how to program those things. I mean, two yeah; four not really; eight, forget it.”

~ Steve Jobs, 1955-2011
Heterogeneous computing refers to systems that use more than one kind of processor. These are systems that gain performance not just by adding the same type of processors, but by adding dissimilar processors, usually incorporating specialized processing capabilities to handle particular tasks. (applications)
Example Heterogeneous Computing Devices

- Multi-core, general purpose, central processing units (CPUs)
  - Include multiple execution units ("cores") on the same chip

- Digital Signal Processing (DSPs) processors
  - Optimized for the operational needs of digital signal processing

- Graphics Processing units (GPUs)
  - Heavily optimized for computer graphics processing

- Field Programmable Gate Arrays (FPGAs)
  - Custom architectures for each problem being solved
Efficiency via Specialization

Source: Bob Broderson, Berkeley Wireless group
What is OpenCL?

- A low level software programming model for software engineers and a software methodology for system architects
  - First industry standard for heterogeneous computing
- Provides increased performance with hardware acceleration
  - Low Level Programming language
  - Based on ANSI C99
- Open, royalty-free, standard
  - Managed by Khronos Group
  - Altera active member
  - Conformance requirements
    - V1.0 is current reference
    - V2.0 is current release
  - http://www.khronos.org
Heterogeneous Platform Model

OpenCL Platform Model

- Host
- Global Memory
- Device

Example Platform: x86

PCIe
OpenCL Constructs

OpenCL standard provides abstract models

- Generic: able to be mapped on to significantly different architectures
- Flexible: able to extract high performance from every architecture
- Portable: vendor and device independent
Parallelism in OpenCL

Parallelism is explicit
- Identify parallel regions of algorithm and implement as kernels executed by many work-items
- Task (SMT) or Data (SPMD)

Hierarchy of work-items (threads)
- Work-items are grouped into workgroups
  - Size of workgroups is usually restricted by hardware implementation (256-1024)
- Work-items within a workgroup can explicitly synchronize and share data
  - Otherwise free to executed independently
- Work-groups are always independent

Explicit memory hierarchy
- **Global** memory visible to all workgroups and work-items
- **Local** memory visible only to work-items in a workgroup
- **Private** memory visible only to a single work-item
High-Level CPU / DSP Architectures

- Optimized for latency
  - Large caches, HW prefetch

- Complicated control
  - Superscalar, out-of-order execution, etc.

- Comparatively few execution units
  - Vector units (e.g. Intel SSE/AVX, ARM® NEON)
Compiling OpenCL Standard To CPUs

- Execute different workgroups across cores
- Potentially ‘fuse’ work-items together to execute on vector units
  - Or ‘vectorize’ kernels to work with explicit vector types
- Synchronization between work-items is handled entirely in software
- No dedicated hardware for sharing data between work-items
  - Rely on caches instead
High-Level GPU Architectures

- Multiple Compute Units ("Cores")
  - Conceptually many parallel threads
  - Vector-like execution
  - Each cores with dedicated resources
    - Registers
    - Local memory/L1 cache
    - Hardware synchronization
- Wide memory bus
  - High bandwidth
  - High latency (800 cycles!)
- Small read/write caches
- PCIe® board
Compiling OpenCL Standard to GPUs

- Distribute workgroups across compute-units
- Work-items execute in parallel on the vector-like cores
- Use dedicated hardware resources for efficient synchronization of work-items and sharing data between work-items
- Run ‘enough’ work-items per compute-unit to mask latencies
  - Work-items contend for fixed resources (registers, local memory)
  - Hardware limits too!
  - Number of work-items/workgroups per compute unit
  - Translates into a requirement for thousands of work-items!
  - NVIDIA provides tool to help calculate this
Greater Challenges Require More Complex Solutions: FPGAs

Tell me what it doesn’t have and you’ll see it in next generation

Intel fab improves performance from 20% to 2X!
- 10Tflops throughput
- Quad A53 ARM processors
- 100’s interface protocols
- GB’s of on-die DRAM

1990’s
- >50K Logic Elements

2000’s
- >500K Logic Elements

2010’s
- >5M Logic Elements
FPGA Architecture: Fine-grained Massively Parallel

Let’s zoom in

- Millions of reconfigurable logic elements
- Thousands of 20Kb memory blocks
- Thousands of Variable Precision DSP blocks
- Dozens of High-speed transceivers
- Multiple High Speed configurable Memory Controllers
- Multiple ARM® Cores
FPGA Architecture: Basic Elements

Configured to perform any 1-bit operation:
AND, OR, NOT, ADD, SUB
FPGA Architecture: Flexible Interconnect

Basic Elements are surrounded with a flexible interconnect
Wider *custom* operations are implemented by configuring and interconnecting Basic Elements.
FPGA Architecture: Custom Operations Using Basic Elements

Wider custom operations are implemented by configuring and interconnecting Basic Elements.

- 16-bit add
- 32-bit sqrt
- Your custom 64-bit bit-shuffle and encode
FPGA Architecture: Memory Blocks

Can be configured and grouped using the interconnect to create various cache architectures
FPGA Architecture: Memory Blocks

Memory Block

20 Kb

Can be configured and grouped using the interconnect to create various cache architectures

addr

data_in

data_out

Lots of smaller caches

Few larger caches
FPGA Architecture: Floating Point Multiplier/Adder Blocks

Dedicated floating point multiply and add blocks
FPGA Architecture: Configurable Routing

Blocks are connected into a custom data-path that matches your application.
Mapping a simple program to an FPGA

High-level code

Mem[100] += 42 * Mem[101]

CPU instructions

R0 ← Load Mem[100]
R1 ← Load Mem[101]
R2 ← Load #42
R2 ← Mul R1, R2
R0 ← Add R2, R0
Store R0 → Mem[100]
First let’s take a look at execution on a simple CPU

Fixed and general architecture:
- General “cover-all-cases” data-paths
- Fixed data-widths
- Fixed operations
Load constant value into register

Very inefficient use of hardware!
CPU activity, step by step

R0 $\leftarrow$ Load Mem[100]

R1 $\leftarrow$ Load Mem[101]

R2 $\leftarrow$ Load #42

R2 $\leftarrow$ Mul R1, R2

R0 $\leftarrow$ Add R2, R0

Store R0 $\rightarrow$ Mem[100]
On the FPGA we unroll the CPU hardware...

R0 ← Load Mem[100]

R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]
… and specialize by position

1. Instructions are fixed. Remove “Fetch”
... and specialize

1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
... and specialize

1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store

R0 ← Load Mem[100]
R1 ← Load Mem[101]
R2 ← Load #42
R2 ← Mul R1, R2
R0 ← Add R2, R0
Store R0 → Mem[100]
... and specialize

1. Instructions are fixed.
   Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly!
   And propagate state.

R0 ← Load Mem[100]

R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]
... and specialize

1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly! And propagate state.
5. Remove dead data.

```
R0 ← Load Mem[100]
R1 ← Load Mem[101]
R2 ← Load #42
R2 ← Mul R1, R2
R0 ← Add R2, R0
Store R0 → Mem[100]
```
... and specialize

1. Instructions are fixed. 
   Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly! 
   And propagate state.
5. Remove dead data.
6. Reschedule!

R0 ← **Load** Mem[100]

R1 ← **Load** Mem[101]

R2 ← **Load** #42

R2 ← **Mul** R1, R2

R0 ← **Add** R2, R0

**Store** R0 → Mem[100]
Custom data-path on the FPGA matches your algorithm!

High-level code

\[ \text{Mem}[100] += 42 \times \text{Mem}[101] \]

Custom data-path

Build exactly what you need:
- Operations
- Data widths
- Memory size & configuration

Efficiency:
- Throughput / Latency / Power
OpenCL on FPGA

OpenCL kernels are translated into a highly parallel circuit
- **A unique functional unit is created for every operation in the kernel**
- Memory loads / stores, computational operations, registers
- Functional units are only connected when there is some data dependence dictated by the kernel

Pipeline the resulting circuit with a new thread on each clock cycle to keep functional units busy

Can launch kernels as single threaded kernels or NDRanges
- No limitations since work group is NDRange (configure internal memory)

*Amount of parallelism is dictated by the number of pipelined computing operations in the generated hardware*
FPGAs Channels Advantage

Standard OpenCL

Alterna Vendor Extension
IO and Kernel Channels

channel int DataChannel;

kernel producer(...) {
    write_channel_altera(DataChannel, value);
}

kernel consumer(...) {
    value = read_channel_altera(DataChannel);
}
Shared Virtual Memory (SVM) Platform Model

Tradational Hosted Heterogeneous Platform

New Hosted Heterogenous Platform with SVM

OpenCL 1.2

OpenCL 2.0

Traditional Hosted Heterogeneous Platform

New Hosted Heterogenous Platform with SVM

OpenCL 1.2

OpenCL 2.0
OpenCL and FPGA Acceleration in the News

**IBM and Altera Collaborate on OpenCL**
“IBM’s collaboration with Altera on OpenCL and support of the IBM Power architecture with the Altera SDK for OpenCL can bring more innovation to address Big Data and cloud computing challenges,” said Tom Rosamilia, senior vice president, IBM Systems.

**Intel Reveals FPGA and Xeon in One Socket**
"That allows end users that have applications that can benefit from acceleration to load their IP and accelerate that algorithm on that FPGA as an offload," explained the vice president of Intel's data center group, Diane Bryant.

**Search Engine Gets Help From FPGA**
"Altera was really interesting in helping with the development—the resources they were willing to throw our way were more significant than those from Xilinx“  Microsoft Engr Manager

**Baidu and Altera Demonstrate Faster Image Classification**
“Altera Corp. and Baidu, China’s largest online search engine, are collaborating on using FPGAs and convolutional neural network (CNN) algorithms for deep learning applications....”
OpenCL On FPGAs Fit Into All Markets

**Automotive/Industrial**
(Pedestrian Detection, Motion Estimation)

**Military/Government**
(Crypto, Image Detection)

**Computer & Storage**
(HPC, Financial, Data Compression)

**Broadcast, Consumer**
(Video image processing)

**Data Processing Algorithms**

**Networking**
(DPI, SDN, NFV)

**Medical**
(Diagnostic Image Processing, BioInformatics)
Case Study: Image Classification

Deep Learning Algorithm
- Convolutional Neural Networking
- Based on Hinton’s CNN

Early Results on Stratix V
- 2X Perf./Power vs. gpgpu
  - despite soft floating point
  - 400 images/s
- 8+ simultaneous kernels
  - vs. 2 on gpgpu
- Exploiting OpenCL channels
  - between kernels

A10 results
- Hard floating point uses all DSPs
  - Better density and frequency
  - ~ 4X performance/watt v SV
- 6800 images/s
- No code change required

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Here are the classes in the dataset, as well as 10 random images from each:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

Hinton's CNN Algorithm
Smith-Waterman

- **Sequence Alignment**
  - Scoring Matrix

  \[
  H(i, 0) = 0, \quad 0 \leq i \leq m \\
  H(0, j) = 0, \quad 0 \leq j \leq n \\
  \text{if } a_i = b_j \text{ then } w(a_i, b_j) = w(\text{match}) \text{ or if } a_i \neq b_j \text{ then } w(a_i, b_j) = w(\text{mismatch})
  \]

  \[
  H(i, j) = \max\left\{\begin{array}{l}
  0 \\
  H(i - 1, j - 1) + w(a_i, b_j) \\
  H(i - 1, j) + w(a_i, -) \\
  H(i, j - 1) + w(-, b_j)
  \end{array}\right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n
  \]

- **Advantage FPGA**
  - Integer Arithmetic
  - SMT Streaming

- **Results**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Power (W)</th>
<th>Performance (MCUPS)</th>
<th>Efficiency (MCUPS/W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3565 Xeon Processor</td>
<td>140</td>
<td>40</td>
<td>.29</td>
</tr>
<tr>
<td>nVidia K20</td>
<td>225</td>
<td>704</td>
<td>3.13</td>
</tr>
<tr>
<td>PCIe385 SV FGPA</td>
<td>25</td>
<td>32596</td>
<td>1303.00</td>
</tr>
</tbody>
</table>
Haplotype Caller (Pair-HMM)

- Smith Waterman like algorithm
  - Uses hidden markov models to compare gene sequences
  - 3 stages: Assembler, **Pair-HMM (70%)**, Traversal + Genotyping
  - Floating point (SP + DP)
  - C++ code starting point (from JAVA)

- Whole genome takes 7.6 days!

- Results

<table>
<thead>
<tr>
<th>Platform</th>
<th>Runtime (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java (gatk 2.8)</td>
<td>10,800</td>
</tr>
<tr>
<td>Intel Xeon E5-1650</td>
<td>138</td>
</tr>
<tr>
<td>nVidia Tesla K40</td>
<td>70</td>
</tr>
<tr>
<td>Altera SV FPGA</td>
<td>15.5</td>
</tr>
<tr>
<td>Altera A10 FPGA</td>
<td>3</td>
</tr>
</tbody>
</table>

Multi-Asset Barrier Option Pricing

- Monte-Carlo simulation
  - No closed form solution possible
  - High quality random number generator required
  - Billions of simulations required
- Used GPU vendors example code
- Advantage FPGA
  - Complex Control Flow
- Optimizations
  - Channels, loop pipelining
- Results

<table>
<thead>
<tr>
<th>Platform</th>
<th>Power (W)</th>
<th>Performance (Bsims/s)</th>
<th>Efficiency (Msims/s/W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3690 Xeon Processor</td>
<td>130</td>
<td>.032</td>
<td>0.0025</td>
</tr>
<tr>
<td>nVidia Kepler20</td>
<td>212</td>
<td>10.1</td>
<td>48</td>
</tr>
<tr>
<td>SV FPGA</td>
<td>45</td>
<td>12.0</td>
<td>266</td>
</tr>
</tbody>
</table>