Did I Just Do That on a Bunch of FPGAs?

Paul Chow

High-Performance Reconfigurable Computing Group

Department of Electrical and Computer Engineering
University of Toronto

September 4, 2015
About the Talk Title

• It’s the measure of success, at least regarding the “masses”

Why? If you want to be like software…

• Software programmers (usually)
  – Don’t need to know what hardware they run on
    • x86, ARM, Sun
  – Don’t worry about memory (much)
  – Don’t need to make PCIe work first

• Software programmers work on top of abstractions
  – Programming environments – languages and compilers, programming models, O/S, debuggers, other tools
SO, WHERE ARE WE?
SOFTWARE PROGRAMMING ENVIRONMENTS FOR FPGAS
HLS is only the beginning

• HLS makes it easier to build the circuit using something closer to a software abstraction
  – Performance not always great, but getting better fast
• Domain-specific languages will provide the best route to performance for software developers

Still missing a lot of other stuff!
(Kunle said it...)
Why is Software “Easy”? 

• Abstractions everywhere  
  – Memory model  
  – I/O  
  – Services  

• Don’t need to worry about memory controllers, PCIe interfaces, ethernet MACs, building a network stack… 

• FPGAs have lacked all of these things, at least in an open standard way
Our MPI Approach (FPL 2006)

Step 1 - All-X86
Application Prototype

Sequential Program

Step 2 - All-X86
Manual Parallelization

MPI Network

Process A
Process B
Process C

Step 3 - FPGA-X86 mix
Accelerator Modeling

TMD-MPI Network

Process A
Process B
Process C

Step 4 - FPGA-X86 mix
Accelerator Substitution

TMD-MPI Network

Hardware Engine
Embedded Processor
X86 Processor

Also a system simulation

HLS can do this
Software control and dataflow easily maps to hardware.

```c
main () {
    MPI_Init()
    MPI_Recv()
    Compute()
    MPI_Send()
    MPI_Finalize()
}
```

**Software**

**Hardware**

- **On-chip Network Infrastructure**
- **Message Passing Engine (MPE)**
- **Commands**
- **Data**

**Message Passing Engine (MPE)**

- **Software control and dataflow easily maps to hardware**
- **Compute()**
- **MPI FSM**
- **Data**
- **Commands**

**Software**

- **main()**
- **MPI_Init()**
- **MPI_Recv()**
- **Compute()**
- **MPI_Send()**
- **MPI_Finalize()**

**Hardware**

- **Message Passing Engine (MPE)**
- **Commands**
- **Data**
- **Compute()**
- **MPI FSM**
Communication Middleware

![Diagram of Communication Middleware]

- **FSB Interface**
- **MPI FSB Bridge**
- **FSB Engine**
- **MPI LVDS Bridge**
- **LVDS interface**
- **MPI MGT Bridge**
- **MGT Interface**
- **Xilinx FPGA**

**Packet Switch**
Achieving Portability

- Portability is achieved by using a Middleware abstraction layer. MPI natively provides software portability.
- Our MPI provides a Hardware Middleware to enable hardware portability. The MPE provides the portable hardware interface to be used by a hardware accelerator.
Molecular Dynamics

• Simulate motion of molecules at atomic level
• Highly compute-intensive
• Understand protein folding
• Computer-aided drug design

• FPGA simulator built and specified by a biochemistry Ph.D. student
Platform Evolution

FPGA portability and design abstraction facilitated ongoing migration.

Network of Five V2Pro PCI Cards (2006)

- First to integrate hardware acceleration
- Simple LJ fluids only

Network of BEE2 Multi-FPGA Boards (2007)

- Added electrostatic terms
- Added bonded terms
2010 – Xilinx/Nallatech ACP

Stack of 5 large Virtex-5 FPGAs + 1 FPGA for FSB PHY interface

Quad socket Xeon Server
Target Platform for MD

- 12 short range nonbond FPGAs
- 2-3 pipelines/NBE FPGA; Each runs 15-30x CPU
- NBE 360-1080x
- 2 PME FPGAs with fast memory and fibre optic interconnects
- PME 420x
- Bonds on quad-core Xeon server
- Bonds 1x

Initial Breakdown of CPU Time

- Short range Nonbonded
- Long range Electrostatic
- Bonds

Quad Xeon

Sys Mem

8.5 GB/s @ 1066 MHz

FSB

Socket 1

Socket 2

Socket 3

72.5 GB/s
Problem: Difficult to mathematically predict the expected speedup \textit{a priori} due to the contentious nature of many-to-many communications.

Solution: Measuring the non-deterministic behaviour using Jumpshot on the software version and back-annotate the deterministic behaviour.

- Make use of existing tools!
Single Timestep Profile

Timestep = 108 ms (327 506 atoms)
Debugging: Multi-FPGA System-Level Simulation

- Test SW and HW ranks all at once
- No need to resynthesize
- Full visibility into the FPGA
- Good for modeling application-defined protocols at initial stages of development

ACP2
ACP1
ACP0

ModelSim
ModelSim
ModelSim

Intel Quad-core Xeon

MPI Messages

FPGA signals

Stdout
Some Proprietary Solutions

- Impulse C
- Maxeler
- Others…
- Provide integrated environments using HLS from a proprietary language that abstracts many of the issues
- For the masses, we need open standards
Now OpenCL

- Adopted by both Altera and Xilinx
- Provides a higher-level software abstraction
  - Don’t see PCIe
  - OpenCL runtime manages bitstreams, memory allocation, data transfer
  - Includes HLS for the kernels
- A knowledgeable software person can use
  - Must understand parallelism, basic architectural concepts, latency and throughput, I/O for data in terms of structure and protocols
  - Doesn’t need to know about clocks
- Early days still, but you can see where it’s going
AND INTO THE CLOUD
SCALING OUT (FIRST WITH OPENCL)
Philosophy

• FPGAs should be capable as first class citizens just like processors

• Think about all processing elements as equal
  – Easier from a programming model perspective
  – Just have different benefits

• “Bury” FPGAs under existing programming models and platforms
  – Like we did with MPI
Now we can “boot” a network connected FPGA accelerator on demand, in seconds!

Framework for HLS – use HLS to create and then “drop in” accelerators
OpenCLoud

Naif Tarafdar, Eric Fukada, Rohan Pavone, Jack Yuan
OpenCL Heterogenous Model

- Xilinx FPGA
- Altera FPGA
- GPU

PCI-e Connection

Host Application
OpenCL Heterogenous Model

- Xilinx FPGA
- Altera FPGA
- GPU

Network Connection

Host Application
OpenCLoud Overview

• Run OpenCL application on network cluster of FPGAs, GPUs, and CPUs

• Two-step process
  – Specify type of cluster user wants (e.g. 10 FPGAs, 5 GPUs)
  – User is provided a virtual cluster where they would run their application
Components

- **OpenStack**
  - Used to provision VM with FPGA and Xilinx Tools and Cluster tools
  - Also GPUs, so far
- **SnuCL**
  - Cluster tool that allows host application to communicate with network devices from SNU
  - OpenCL runtime using MPI to connect to remote devices
- **Automation scripts**
OpenStack

• Large cloud infrastructure providing service
• Nova -> component responsible for provisioning resources
Application

• Video Processing Application
• Dynamic number of streams sent into cluster to be processed
• The processing on the device kernel can be substituted for a library of video processing applications
  – Currently implemented object edge detection
First Application

Cluster Scheduler (3 concurrent threads)

Video Stream 1

User Host

Video Stream N

Device 1

Device M

September 4, 2015
Application Status

• Application is currently working on the cloud with multiple software-simulated FPGAs and one physical FPGA
• Streams dynamically added and dropped
• So far, only a summer project using “off-the-shelf” components to make something work
• It’s only the beginning!
BIGQUERY (APACHE DRILL) CASE STUDY
BigQuery

- Interactive query service for massive datasets.
- Provided by Google
- Uses thousands of servers
- Uses SQL
BigQuery on OpenCloud

BigQuery Client

Drillbit

Virtual Device

Data Source

Drillbit

Virtual Device

Data Source

Drillbit

Virtual Device

Data Source

Drillbit

Virtual Device

Data Source
Future Challenge

• Direct connections between FPGAs
ACCELERATING COMPUTATIONS IN APACHE SPARK
Apache Spark

• Reliable large-scale distributed data processing engine

• Key Features:
  – In-memory analysis
  – Interactive applications

• Main abstraction: Resilient Distributed Dataset (RDD)
  – Fault-tolerance abstraction representing data

• Efficient execution model

• Supports Java, Scala, and Python
System Architecture

- Set of identical CPU-FPGA nodes
System Architecture Cont’d

• Datapath

![Diagram of System Architecture](image-url)
System Architecture Cont’d

- Hardware Design

Common to all applications

Common to all MR apps

MR-specific computations
System Prototype

- Spark Cluster on 16 Zynq boards
- Could run on OpenCLoud
BACK TO THE MASSES
Who are the Masses?

• Any application developer writing software and wishing to do some computation
• Don’t want/need to know what is actually doing the computation
  – Just get the job done quickly (performance) and cheaply (short time, low cost of power, open source)

Linux fits this model
Open standard APIs are needed

• Hardware abstraction layer for software
  – Software talking to FPGA

• Hardware abstraction layer for hardware
  – Application hardware talking to “shell”
  – Enable portability of hardware modules

• Enable portability between devices and vendors

• Can build other abstractions on top
The Stack

Software Environment

- SW Application
- SW Middleware
- SW OS
- Host-specific Hardware

Heterogeneous Environment

- SW Application
- SW Middleware
- SW OS
- Host-specific Hardware
- HW OS
- HW Middleware
- HW Application
- FPGA
- Host

Standardize
Harden the Shell

- Current shells are implemented in soft logic to glue together the interfaces and provide services
- Uses significant resources
- Once the requirements are determined and volumes merit, then harden it
  - Important aspect is security and protection
  - Could still be vendor dependent, but should make the standard interface much lighter to build in soft logic
EVEN FURTHER OUT...
OpenCL for FPGAs

- Can develop board-independent applications
- But, currently vendor specific

OpenCL

HDL

Middleware HDL

Vendor Synthesis

Hardware Platform

Board configuration
SPIR-V from Khronos

- GLSL ESSL
- Game Engines
- New Languages
- Other Intermediate Forms
- SPIR-V
- Vulkan Driver Y
- Vulkan Driver X
- OpenCL Driver A
- OpenCL Driver B
What if…

- Code now vendor and platform independent!
- Could target any device
- Need **performance portability**
FINAL THOUGHTS
RC for the Masses?

• HLS is only part of the solution
• Lot’s of other infrastructure needed
• First need to take existing and familiar environments and insert FPGAs into them in a way that hides as much of the scary stuff as possible (“platforms” from the first keynote)
• Then make it easier to build new environments by developing open source standards that make it easy to port the environments easily across platforms so that they become pervasive
Questions?