ADVANCING UBIQUITOUS FPGA USE IN HPC AND ALGORITHMIC ACCELERATION

Dr. Suleyman Demirsoy
Algorithmic Acceleration Architect
Networking and Custom Logic Group
Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

© 2019 Intel Corporation. Intel, the Intel logo and other Intel marks are trademarks of Intel Corporation or its subsidiaries.

*OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. Other names and brands may be claimed as the property of others.
AGENDA

- Heterogeneous computing, FPGA in data centres
- Platform level and Architectural differentiation
- Obstacles in the way of Adoption
- An acceleration scenario
- How to bring ease of use with High level Design
MY JOURNEY WITH FPGA

HW engineering background

Started as an IP designer (RTL → automation)

Communications Systems designer (RTL, DSP Builder Blockset on Simulink)

DSP Specialist (DSP Builder, HLS, OpenCL)

Acceleration Specialist (HLD Tools, Optimization)

Acceleration Architect (HLD Tools, Optimizations, Platforms)
What if 10 Peta Flops, 10 Peta Bytes were <10 Milliseconds away from every person?
The future is a diverse mix of scalar, vector, matrix, and spatial architectures deployed in CPU, GPU, AI, FPGA and other accelerators.
Race to expand applicability
Best of many worlds
  Built-in accelerators
  Chiplets (AI, memory, processors, interfaces ...)
  Chipsets
Various levels of interconnects
Delivering more performance and **usability**
PLATFORM/ARCHITECTURE DIFFERENTIATION

More than our fair share
Microsoft Catapult Architecture

Interconnected FPGAs form a separate plane of computation

Can be managed and used independently from the CPU

Traditional software (CPU) server plane

Hardware acceleration plane

Deep neural networks

Web search ranking

SDN offload

SQL
FPGAs enable platform-level and architectural differentiation.
**WHAT IS CXL?**

- CXL is an alternate protocol that runs across the standard PCIe physical layer
- CXL uses a flexible processor port that can auto-negotiate to either the standard PCIe transaction protocol or the alternate CXL transaction protocols
- First generation CXL aligns to 32 Gbps PCIe Gen5
EASE OF MEMORY ACCESS WITH CXL

Cache coherent access to data from CPU/FPGA

- Removes capacity problems for big data processing
- Host access to memory objects through native methods

Accelerators with Memory

Usages:
- GPU, FPGA
- Dense Computation

Protocols:
- CXL.io
- CXL.cache
- CXL.memory

Memory Buffers

Usages:
- Memory BW expansion
- Memory capacity expansion
- 2LM

Protocols:
- CXL.io
- CXL.mem

Networking and Custom Logic Group
COMPREHENSIVE ACCELERATION STACK

Applications/Orchestration

Orchestration / Rack Level Management

User Applications

Vertical Software Frameworks/Libs

(DL, Networking, Genomics, etc.)

Frameworks

Intel Xeon FPGA Acceleration Libraries

Intel® DAAL
Intel® MKL
Intel® MKL-DNN
Intel® DL Deployment Toolkit

Rack Scale Design

Deep Learning, Networking, Analytics, etc.

Common Infrastructure

✓ Simplify FPGA programming model

FPGA HW & SW Tool Chains

Open Programmable Acceleration Engine (OPAE Software API)

Operating Systems

Common Linux kernel driver, tools and software programming layer, HLS/HDL FPGA design tools

OS Enablement: Linux, Windows, ESXi,

FPGA Images

FPGA Interface Manager (FIM)

Loadable accelerator image

IP Libraries: Developed in-house and from third parties

Hardware

FPGA Platforms (Programmable Acceleration Cards)

Networking and Custom Logic Group

Intel® DL Deployment Toolkit
OBSTACLES FOR ADOPTION

More than our fair share
NOT INVENTED HERE (NIH) SYNDROME

Blackbox designs not always fulfil the needs.

▪ Sensitivity on owning the IP.
▪ Multiple customizations for different use cases.
▪ Incremental improvements
  – RTL vs automation
▪ Complexities in licensing models

Empower the SW developer with

▪ SW programmability
▪ Optimized library modules
BENCHMARKING ASKS

Expectations vary hugely.

- Typical FPGA account:
  - How do you compare against RTL?"

- Typical HPC account:
  - Show us your results on Gromacs...OpenFoam, LAMMPS...?
  - ... Linpack? ... FFT? SGEMM?
  - What can you show?

Simple functions may be misleading:

- Functions with dedicated ASIC blocks in CPU/GPU

Small parts of large applications are also misleading.

- Area estimates for a given performance multiplier.
- Bigger/deeper pipeline changes a lot of assumptions

**Bigger/deeper pipe with data streaming from one block to next provides much higher differentiation**
Enabling SW Community - Adoption

- Performance alone is not enough for adoption.
  - Why?

- Development environment
  - Debugging, profiling
  - Reporting
  - IDE

- Ease of use in:
  - Host code modifications,
  - Data transfers
  - Getting insightful feedback from compiler
ENABLING THE BROAD BASE – TRICKLE-DOWN STRATEGY

Not all user personas will be using HLD Tools for development.

Building optimized PRIMITIVES and LIBRARIES that are re-used by the broad base.

Direct Programming

API Programming

Integrating and using pre-built PRIMITIVES and LIBRARIES in acceleration workloads.
AN ACCELERATION SCENARIO
BENCHMARKING / DEVELOPMENT STEPS

... while (cond) {
  some_data_manipulations;
  func1();
  some_more_data_manipulations;
  func2();
  func3();
  cond_calculations;
}
...

Accelerate
  ▪  func1();
  ▪  While loop
# Accelerator Optimization Metrics

## System Acceleration Metrics

<table>
<thead>
<tr>
<th>Description</th>
<th>Timespent</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_S$</td>
<td>Time spent in an un-accelerated system</td>
</tr>
<tr>
<td>$T_{AS}$</td>
<td>Time spent in an accelerated system</td>
</tr>
<tr>
<td>$T_h$</td>
<td>Time spent in main application, running on the host (excluding un-accelerated function)</td>
</tr>
<tr>
<td>$T_f$</td>
<td>Time spent in un-accelerated function</td>
</tr>
<tr>
<td>$T_c$</td>
<td>Time spent in communication between the host and accelerator</td>
</tr>
<tr>
<td>$T_{af}$</td>
<td>Time spent executing accelerated function</td>
</tr>
<tr>
<td>$A_s$</td>
<td>System Acceleration = $T_S/T_{AS}$</td>
</tr>
</tbody>
</table>
Where is the input data coming from?
What is the CPU execution time.
Any other benchmarks (i.e. GPU)?
Has func1() been optimized for CPU execution? (vectorized, multi-thread etc)?
**ACCELERATOR OPTIMIZATION METRICS**

System Acceleration Metrics

<table>
<thead>
<tr>
<th>Description</th>
<th>Time spent in an un-accelerated system</th>
<th>Time spent in an accelerated system</th>
<th>Time spent in main application, running on the host (excluding un-accelerated function)</th>
<th>Time spent in un-accelerated function</th>
<th>Time spent in communication between the host and accelerator (Tctrl + Tdata)</th>
<th>Time spent executing accelerated function</th>
<th>System Acceleration $A_s$ = $T_S/T_{AS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_S$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{AS}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_h$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_f$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_c$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{af}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**You are always asked to report** $T_c + T_{af}$

**In most cases, you are in control of** $T_{data}$ **and** $T_{af}$
INTEL VTUNE SHOWS CPU/FPGA INTERACTION

Intel® VTune™ Amplifier – Performance Profiler
while (cond) {
    some_data_manipulations;
    start_timer();
    func1();
    finish_timer();
    some_more_data_manipulations;
    func2();
    func3();
    cond_calculations;
}

Ways to get performance in your func1() accelerator

- Instantiation
- Vectorization
- Reducing data transfer and control overheads
- Resolve dependencies to parallelize
ACCELERATING THE LARGER FUNCTION

... while (cond) {
    some_data_manipulations;
    func1();
    some_more_data_manipulations;
    func2();
    func3();
    cond_calculations;
}
...

More computational steps
  - Deeper pipe
  - More area consumption

func1() implementation needs to adapt
  - Less parallelism
  - Data source/sink are different

Maximize on chip data access/retrieval
  - While loop latency better

Be ready for surprises
... while (cond) {
    some_data_manipulations;
    func1();
    some_more_data_manipulations;
    func2();
    func3();
    cond_calculations;
} ...

Different library abstraction levels possible

- The whole of the larger function
- func1(), func2(), func3()
- Primitives for functions
- Compatible interfaces, flow control

Architecture aware coding at every level is required for getting good performance
LIBRARIES FOR SPATIAL ARCHITECTURE
multitude of flexibilities needed

Data types (fixed point, half, float, double etc)

Parallelism
- instantiation, vectorization

Ingress / Egress styles

Latency limit
- Is this component standalone or part of a pipeline
- implications to interfaces, control complexity

- samples/cycle
- streaming vs buffered
- dependencies to higher level function

Data storage location

Portability between device families, architectures

- onchip vs offchip
- bandwidth/latency assumptions greatly differ

Challenges with the RTL based flow to capture all of the above

High level language based frameworks needed to cater for the flexibility
EASE OF USE WITH HLD
Removing the Barriers of Adoption

Hardware Developers
- Intel® Quartus® Prime Design Software
- DSP Builder
- High-Level Design Backend Compiler
- LLVM Compiler
- Intel® HLS Compiler

Software Developers
- Deep Learning Accelerator
- AI Frameworks
- Parallel Compilers
- Libraries
- Primitives
- Software Stacks

Intel® FPGA SDK for OpenCL

Acceleration Stack Framework

Networking and Custom Logic Group
ACCELERATING HLD TOOL IMPROVEMENTS

**Intel® HLS Compiler**

C/C++ Front-End

- Accelerated Improvement of Quality of Results

Front End Tools and Reporting

- Compiler Optimizations

- Compiler Infrastructure

Platform

- DSP Builder Front-End
- OpenCL Front-End

- Actionable feedback and power user control
- Accelerated support for advanced features of our products
INTEL FPGA TOOLS PORTFOLIO

catering different developer personna

Software Programmer

Algorithm Designer

Embedded Designer

Hardware Designer

oneAPI

Intel® FPGA SDK for OpenCL™

Intel® HLS Compiler

DSP Builder for Intel® FPGAs

HDL Code
Quartus, Platform Designer
Project One API will deliver a unified programming model to simplify development across diverse architectures.

Common developer experience across Scalar, Vector, Matrix and Spatial architectures (CPU, GPU, AI and FPGA).

Based on industry standards and open specifications.

More details on Language and compiler from Andrei Hagiescu on Friday.
ONE API FOR CROSS-ARCHITECTURE PERFORMANCE

Optimized Applications

Optimized Middleware & Frameworks

One API Product

Direct Programming
- Data Parallel C++

API-Based Programming
- Libraries

Analysis & Debug Tools

Direct Programming
- Data Parallel C++

API-Based Programming
- Libraries

Analysis & Debug Tools

Some capabilities may differ per architecture.
Productive performance analysis across SVMS architectures

Performance Profiler
Parallelization Assistant
Debugger
Collaboration through library

- Solution and platform developers growing... Great!!
- For broad adoption,
  - FPGA device to be natively detected and used in the OS (driver upstreaming)
  - FPGA based acceleration/offload to be added to the main branch (FPGA manager class upstream?)
    - RTE, data transfer management
  - Building on the work of others
  - Collaboration on high level design based library flows
CONCLUSION

▪ FPGAs can differentiate with platform level and architectural innovations.

▪ Adoption of FPGA use in the SW community requires ease of use on development, debugging and optimization. Performance alone is not enough for adoption.

▪ Benchmarking efforts should consider as much of the final solution as possible to show FPGAs' worth. Simple functions are not enough to highlight strengths of FPGA.

▪ Libraries for FPGA development should reflect the strength and flexibilities of spatial design.

▪ For broad adoption in algorithmic space, an ecosystem of library developers contributing to a common framework is needed.