An Evolution of General Purpose Processing: Reconfigurable Logic Computing

Joel Emer
Intel Fellow

Princeton – April 2, 2009
Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.

- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards 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.
- Do not finalize a design with this information. Do not finalize a design with this information.
- This document may contain information on products in the design phase of development. The information here is subject to change without notice.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.
- Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.
- Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.
- Wireless connectivity and some features may require you to purchase additional software, services or external hardware.
- Nehalem, Penryn, Westmere, Sandy Bridge and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.
- Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.
- Intel, Intel Inside, Pentium, Xeon, Core and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.

Copyright © 2009 Intel Corporation.
The “Black Swan” Theory

Event has appeared by complete surprise

Event has a major impact

Event is explicable in retrospect


Note: The theory was described by Nassim Nicholas Taleb in his 2007 book The Black Swan
Black Swans in Computing

First Spreadsheet, 1983
VisiCalc Advanced Version

Desktop Publishing, 1985

WIMP, 1980
Window, Icon, Menu, Pointing device

Computer Graphics

Another Black Swan

Web Browsers

PodCasts

VidCasts

The Rise of the Internet
Breeding Black Swans

Gap

Power is the limiter

Demand architectural innovations to fill in the gap
Architecture Evolution: Addressing the Gap

CPU
- ISA Extensions: MIMD, vectors

Accelerators
- Fixed functions: GPU, video encoders, encryption

Multi-threading, Multi-core, Many Core

Programmability
- Throughput Performance

Fully Programmable, Partially Programmable, Fixed Function
Today’s White Swan in Computing

TCP Offload Engine
260K Transistors

Special Purpose Accelerators

Source: Intel Labs
### Field Programmable Gate Arrays (FPGA)

**Logic Unit (LUT)**

**Latch**

<table>
<thead>
<tr>
<th>And</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Or</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Random Access Memory (RAM)**

A fixed function compromise
Evolving reconfigurable logic usage

Logic replacement
- Low design cost and effort
- For low volume applications
- Often replaced with ASIC as volume increases

Algorithmic Computation
- Offloads a general purpose processor
- Used for many algorithms
- ASIC replacement not expected
## Performance Acceleration with FPGA-based accelerators

<table>
<thead>
<tr>
<th>Applications</th>
<th>HW (FPGA)</th>
<th>SW Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hough &amp; Inverse Hough processing</td>
<td>2 seconds of processing time</td>
<td>12 minutes processing time</td>
</tr>
<tr>
<td></td>
<td>@20Mhz</td>
<td>Pentium 4 - 3GHz</td>
</tr>
<tr>
<td></td>
<td>370x faster</td>
<td></td>
</tr>
<tr>
<td>AES 1MB data processing/cryptography rate</td>
<td>424 ms / 19.7 MB/s</td>
<td>5,568 ms / 1.51 MB/s</td>
</tr>
<tr>
<td>Encryption</td>
<td></td>
<td>5,562 ms / 1.51 MB/s</td>
</tr>
<tr>
<td>Decryption</td>
<td>13x faster</td>
<td></td>
</tr>
<tr>
<td>Smith-Waterman search34 from FASTA</td>
<td>100 sec FPGA processing</td>
<td>6461 sec processing time</td>
</tr>
<tr>
<td></td>
<td>64x faster</td>
<td>Opteron</td>
</tr>
<tr>
<td>Multi-dimensional hypercube search</td>
<td>1.06 Sec FPGA@140Mhz Virtex II</td>
<td>119.5 Sec</td>
</tr>
<tr>
<td></td>
<td>113x faster</td>
<td>Opteron - 2.2 Ghz</td>
</tr>
<tr>
<td>Monte-Carlo Analysis 64,000 paths</td>
<td>10 sec of Processing @200 Mhz</td>
<td>100 sec processing time</td>
</tr>
<tr>
<td></td>
<td>FPGA system</td>
<td>Opteron - 2.4 Ghz</td>
</tr>
<tr>
<td></td>
<td>10x faster</td>
<td></td>
</tr>
<tr>
<td>BJM Financial Analysis 5 million paths</td>
<td>242 sec of Processing @61 Mhz</td>
<td>6300 sec processing time</td>
</tr>
<tr>
<td></td>
<td>FPGA system</td>
<td>Pentium 4 – 1.5 Ghz</td>
</tr>
<tr>
<td></td>
<td>26x faster</td>
<td></td>
</tr>
<tr>
<td>Black-Scholes</td>
<td>18 msec FPGA@110Mhz Virtex-4</td>
<td>3.7 Sec 1M iterations</td>
</tr>
<tr>
<td></td>
<td>203x faster</td>
<td>Opteron - 2.2 Ghz</td>
</tr>
</tbody>
</table>

* Chart from Celoxica
HPC Accelerator Whitepaper Rev 0.9 (Intel), September 14, 2006, by Steve Duvall, Tom Marchok
Fine grain parallelism and state

- Example: numerically solving partial differential equation – Laplace's equation
  \[ \nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \]

- In traditional cores
  - Max parallel degree: \( p \) (threads)
  - \( K^2/p \) cycles to do one iteration

- In RL
  - Max parallel degree: \( K^2/2 \)
  - 2 cycles to do computation in one iteration: one for all “o”, one for all “x”
  - Note: the sequential version of this algorithm is not suitable in traditional cores with cache – break the law of spatial locality

Source: Intel Labs, Tao Wang
The Good

Custom operations/data types – RL allows custom operations/data types

• Fine grain parallelism - replicated logic permits easy parallelism
• Local access to state - local state elements allows parallel state access
• Custom communication - explicit direct inter-module communication
• Flexible flow control - Control flow based on arbitrary state machine
• Better power efficiency - its custom computation / logic & interconnect
The Bad and the Ugly

**Insufficient capacity**
- Being overcome by Moore’s Law
- Addressable in system architecture

**Slower Cycle Time**
- Parallelism is already offsetting lower frequency
- Being addressed by higher-level/asynchronous fabrics

**Difficult to Program and Debug**
- Applications typically must develop everything
- No computation or system architecture
- No standard environment
Is there space for RL on-die?

Montecito

2 cores
28.5M Transistors each
24MB L3 Cache
1550M Transistors
Architecture: Black Swan Enabler

Architectures provide:

- Environmental stability within/across generations
  - Existing applications continue to work

- Consistency across a larger number of units

- Encouragement to create reusable foundations
  - Tool chains, Operating systems and libraries

- Enticement for application innovation
RL architecture

Precompiled RL Library
Virtual Memory
Virtual Devices
Message Passing

System Architecture
Compute Architecture
Fabric Architecture

An architectural Approach to Reconfigurable Logic
RL fabric architectures

- **Fine-grain FPGA**
  - Array of ALUs with fixed functions
  - 32 rows
  - 32 cols

- **FPGA with flexible routing/logic block**
  - 32 blocks/row for 32-bit ops

- **Row-based RL**
  - 32 rows

- **Coarse-grain RL**
  - Array of ALUs with fixed functions

**Finer (flexibility)**

**Coarser (performance)**
A CPU architecture is a specification of the interface between the machine language and the hardware.
### RL Compute Architecture Alternatives

<table>
<thead>
<tr>
<th>Architectural semantics</th>
<th>Possible name of this kind of architecture</th>
<th>Example of what RL functions as</th>
</tr>
</thead>
<tbody>
<tr>
<td>Async</td>
<td>Accesses memory</td>
<td>Has context</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
RL Architecture Hierarchy

How do we sequence RL?

**Synchronous**

- Use RL operations inside a conventional pipeline. E.g., as a separate function unit
- Control handled by standard control instructions

**Asynchronous**

- A standalone logical state machine
  - Implemented directly in RL
  - Allowing direct control input from any module
In-pipeline core-RL architecture (Type 0-3)
Peer computing RL architecture (Type 7)
Implementation alternatives

Tightly coupled
Normally reconfigurable
instructions

Medially coupled
Normally computing blocks

Loosely coupled
Normally complex application
A Processing Black Swan?

CPU
- Sequential, coarse parallel or un-pipelined algorithms
- Floating Point

RL
- Fine grained parallel or pipeline-parallel algorithms or complex flow control
- Custom operations, e.g. odd data sizes or fine grained bit-manipulations
- Integer

FSB

CPU and RL Respective Strength
RL Development Model

Specialized for a market

Accelerators for specific but common computations

Auto-programmed accelerators

User-programmable accelerators

Peer Processors

Only hardware change logic

OS/Compilers select among predefined libraries

Compilers generate logic code

Experts may write their own specialized modules

Developers freely decide on split of code between HW and SW

The quality of the environment will determine the deployment model
System Environment Evolution

Software – Then

- Languages
  - Binary
  - Assembly

- No Standardized System Environment
  - Raw Devices

- No Distributed Computing Paradigms

Software – Now

- Languages
  - C++
  - Python
  - AJAX

- Rich System Environment
  - Device Abstractions
    - File Systems
    - Character Devices
  - Virtual Memory
  - Exception Handling

- Communication Paradigms
  - Shared Memory
  - Message Passing
  - Remote Procedure Calls
Evolving RL Systems

**FPGAs – now**
- Languages
  - Verilog (~Assembly)
  - VHDL (~Assembly)
- No Standardized System Architecture
  - Raw Devices
- No Distributed Computing Paradigms

**FPGAs – looking forward**
- Languages
  - C/C++
  - Bluespec
- Standardized System Architecture
  - FPGA virtual platform
- Communication Paradigm
  - Streams
  - Remote Request Response
Sequencing controlled by ‘guarded atomic actions’ (rules with conditions) that execute a set of operations entirely or not at all. Known technology can generate high-quality HW.
WiMAX requirement is to support a throughput of 134Mbps

<table>
<thead>
<tr>
<th></th>
<th>Xilinx IP</th>
<th>Catapult-C</th>
<th>Bluespec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent Gate Count</td>
<td>297,409</td>
<td>596,730</td>
<td>267,741</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>145.3</td>
<td>91.2</td>
<td>108.5</td>
</tr>
<tr>
<td>Steady State</td>
<td>660</td>
<td>2073</td>
<td>276</td>
</tr>
<tr>
<td>(Cycles/Block)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data rate (Mbps)</td>
<td>392.8</td>
<td>89.7</td>
<td>701.3</td>
</tr>
</tbody>
</table>

Lower is better

Higher is better

Source: MIT, Abhinav Agarwal, Alfred Ng – CSG
Hybrid Instruction Emulation

FPGA

Execute

RRR Layer

Sync Registers

Emulate Instruction

Emulation Server

Instruction Simulator

Software

Functional Cache

Write Back or Invalidate

Write Line

Ack

Done

Memory Server

Emulation Server

Instruction Simulator

Time

Source: Intel, Michael Adler - VSSAD

Implemented in a day
Summary

A perspective on the role of general purpose computing in application innovation

Some possibilities for reconfigurable logic-based computing as a component of the general purpose computing environment

New opportunities for application of code generation and optimization
Acknowledgements

Arvind
Michael Adler
Azam Barkatullah
Angshuman Parashar
Michael Pellauer
Tao Wang
ZhiHong Yu
Questions?