

# CprE 488 – Embedded Systems Design

## Lecture 8 – Hardware Acceleration

Joseph Zambreno

Electrical and Computer Engineering

Iowa State University

[www.ece.iastate.edu/~zambreno](http://www.ece.iastate.edu/~zambreno)

[rcl.ece.iastate.edu](http://rcl.ece.iastate.edu)

*First, solve the problem. Then, write the code. – John Johnson*

# Motivation: Moore's Law

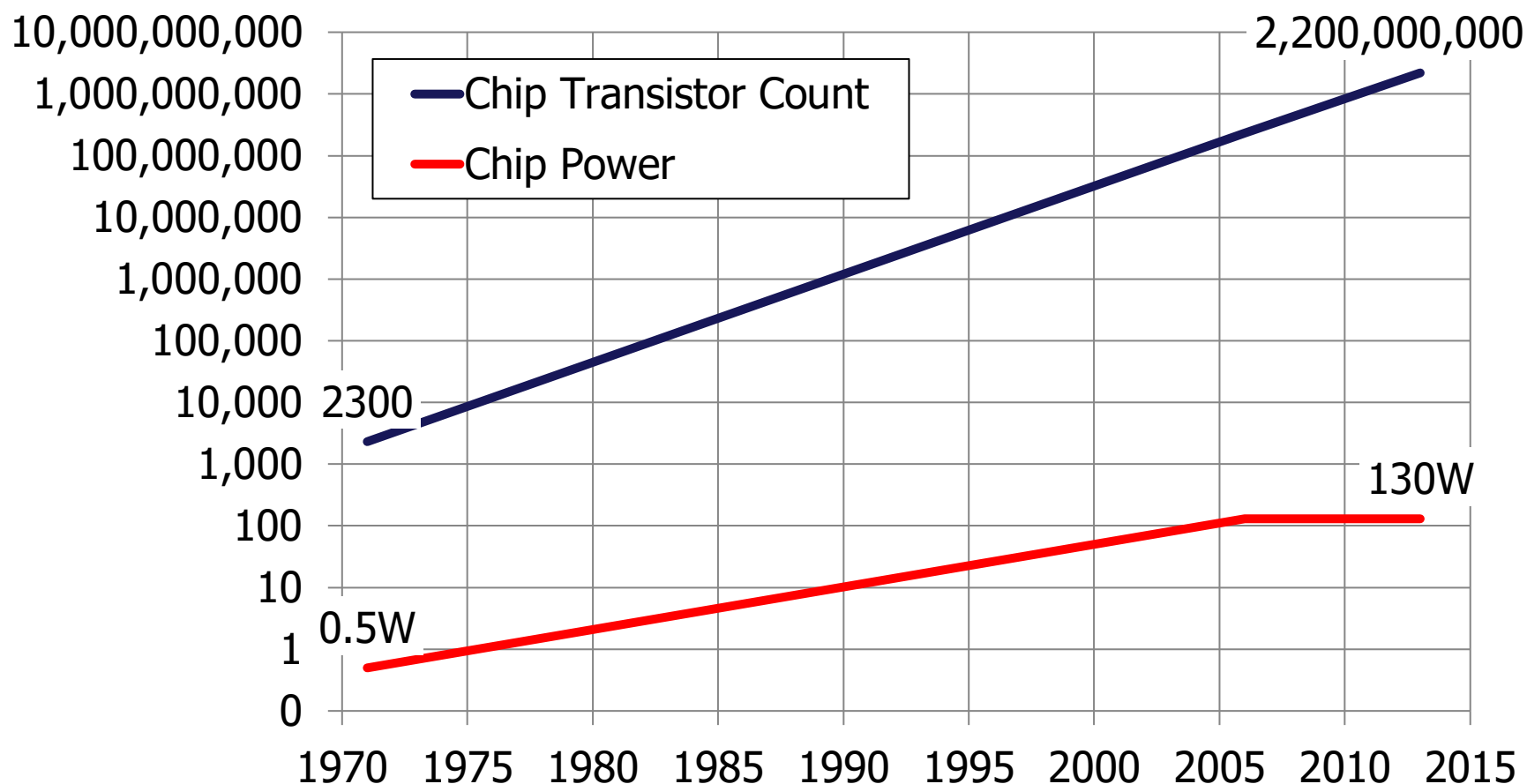
- Every two years:
  - Double the number of transistors
  - Build higher performance general-purpose processors
    - Make the transistors available to the masses
    - Increase performance ( $1.8\times\uparrow$ )
    - Lower the cost of computing ( $1.8\times\downarrow$ )
- Sounds great, what's the catch?



Gordon Moore

# Motivation: Moore's Law (cont.)

- The “catch” – powering the transistors without melting the chip!
  - See 2003 – 2004 news:



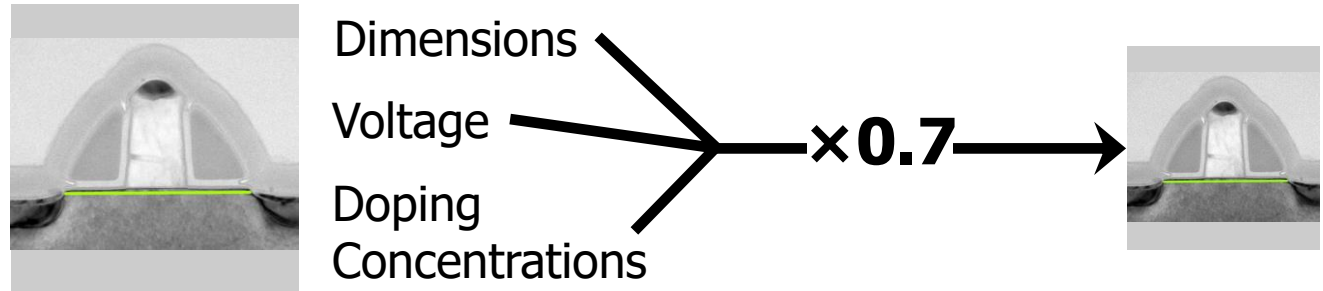
# Motivation: Dennard Scaling

- As transistors get smaller their power density stays constant



Robert Dennard

## Transistor: 2D Voltage-Controlled Switch



Area  $\longrightarrow$   $0.5\times \downarrow$   $\longrightarrow$

Capacitance  $\longrightarrow$   $0.7\times \downarrow$   $\longrightarrow$

Frequency  $\longrightarrow$   $1.4\times \uparrow$   $\longrightarrow$

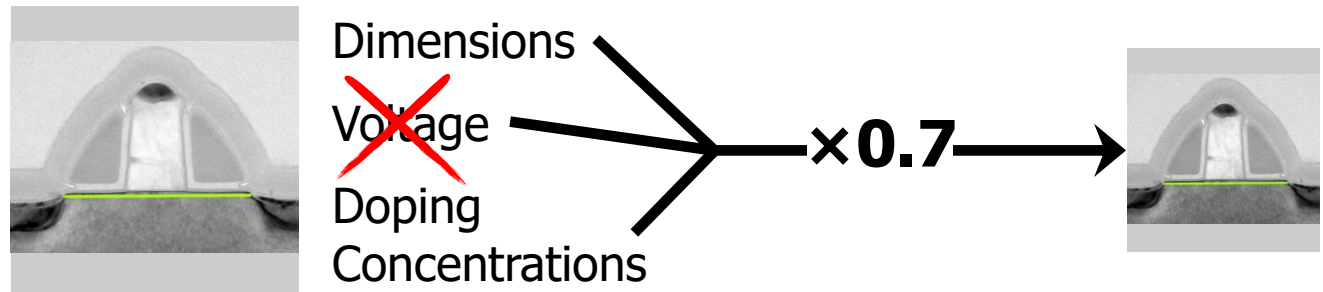
$$\text{Power} = \text{Capacitance} \times \text{Frequency} \times \text{Voltage}^2$$

Power  $\longrightarrow$   $0.5\times \downarrow$   $\longrightarrow$

# Motivation Dennard Scaling (cont.)

- In mid 2000s, Dennard scaling "broke"

Transistor: 2D Voltage-Controlled Switch



Area  $\longrightarrow$   $0.5\times \downarrow$   $\longrightarrow$

Capacitance  $\longrightarrow$   ~~$0.7\times \downarrow$~~   $\longrightarrow$

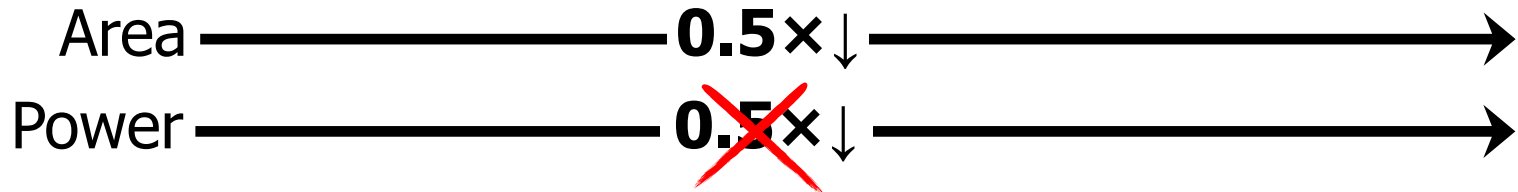
Frequency  $\longrightarrow$   ~~$1.4\times \uparrow$~~   $\longrightarrow$

$$\text{Power} = \text{Capacitance} \times \text{Frequency} \times \text{Voltage}^2$$

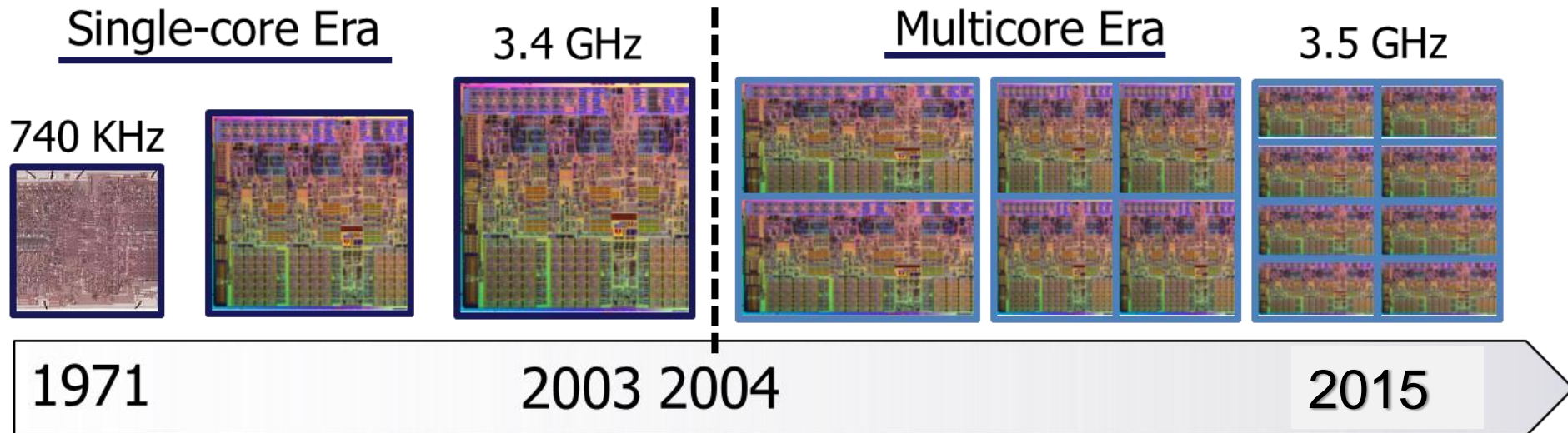
Power  $\longrightarrow$   ~~$0.5\times \downarrow$~~   $\longrightarrow$

# Motivation: Dark Silicon

- **Dark silicon** – the fraction of transistors that need to be powered off at all times (due to power + thermal constraints)



- Processor evolution strongly motivated by Dennard scaling ending
  - Expected continued evolution towards HW specialization/accel



# This Week's Topic

- Hardware Acceleration:
  - Performance analysis and overhead
  - Coprocessors vs accelerators
  - Common acceleration techniques
  - Acceleration examples
- Reading: Wolf section 10.4-10.5

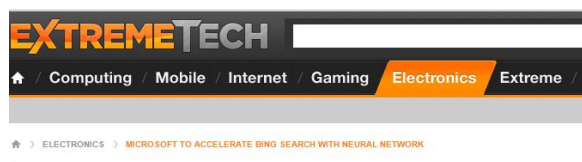






# And Today?

- Reconfigurable logic supporting the data center, specializing clusters, and tightly integrated on-chip



## Microsoft to accelerate Bing search with neural network

By John Hewitt on February 25, 2016 at 3:46 pm | 19 Comments



### Share This Article



deep web — not to be confused with the dark web — is buried further down in the sites that make up the visible surface web. The indexes of competitors like Yahoo and Bing (around 15 billion pages each) are still only half as large as Google's. To close this gap, Microsoft has recently pioneered sophisticated new Field-Programmable Gate Array (FPGA) technology to make massive web crawls more efficient, and faster.

Google's engineers have previously estimated that a typical 0.2-second web query reflects a quantity of work spent in indexing and retrieval equal to about 0.0003 kWh of energy per search. With over 100 billion looks per month at their petabyte index, well-executed page ranking has become a formidable proposition. Microsoft's approach with Bing has been to break the ranking portion of search into three parts — feature extraction, free-form expressions, and machine learning scoring:

When we search Google's web index, we are only searching around 10 percent of the half-a-trillion or so pages that are potentially available. Much of the content in the larger



Home Systems Software Datacenter Cloud Storage Networks Insight Sectors

## IBM Accelerates Power8 Clusters With GPUs, FPGAs, And Flash

October 2, 2014 by Timothy Prickett Morgan



designed to take workloads away from X86 clusters.

As *EnterpriseTech* has previously reported, IBM has been telling customers to expect larger Power8-based machines with more than two sockets as well as systems that would use field programmable gate arrays (FPGAs). IBM has also been hinting that OpenPower partner and GPU coprocessor maker Nvidia would be working together to get a Power8-Tesla hybrid system into the field before the end of the year.

It turns out that IBM is launching three different systems tuned up for three different kinds of workloads that are based on its "scale-out" Power8 systems. By scale-out, IBM means a system is designed with one or two sockets and is intended to be used in clusters that have distributed applications that scale their capacity by adding multiple nodes in a loosely coupled fashion. This is distinct from "scale-up" machines, which more tightly couple server nodes and their main memory together, usually using non-uniform memory access (NUMA) technology, to create what is in essence a single large processor to run fat applications or their databases.

Big Blue is also rolling out scale-up versions of its Power8 systems, which it has also promised would come this year, ahead of the Enterprise2014 event. So don't think the Power8 rollout is only about creating a Power8 alternative to the workhorse, two-socket server based on Intel's Xeon E5-2600 processors. (We will report on these NUMA machines, which are called the Power Enterprise Systems, in a separate story.)

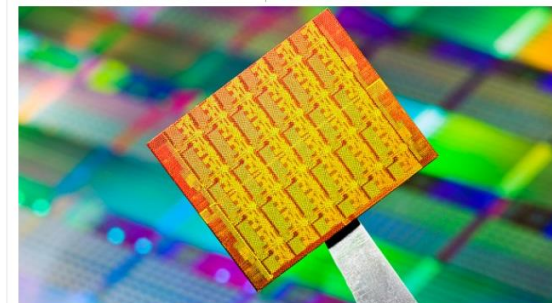
The new Power S824L is a Linux-only version of the existing Power S824 machine that IBM announced back

It is perhaps a lucky stroke of timing or perhaps by design that only days after Big Blue sold off its System x X86 server business to Lenovo Group for \$2.1 billion that the company is coming out swinging with Power8 servers that are augmenting their performance using a variety of adjunct co-processors and flash storage. But ahead of next week's Enterprise2014 event in Las Vegas, where it will be talking about its increasing focus on Power Systems and System z mainframes, the company is launching a number of systems that are



## Intel unveils new Xeon chip with integrated FPGA, touts 20x performance boost

By Sebastian Anthony on June 19, 2014 at 1:19 pm | 59 Comments



### Share This Article



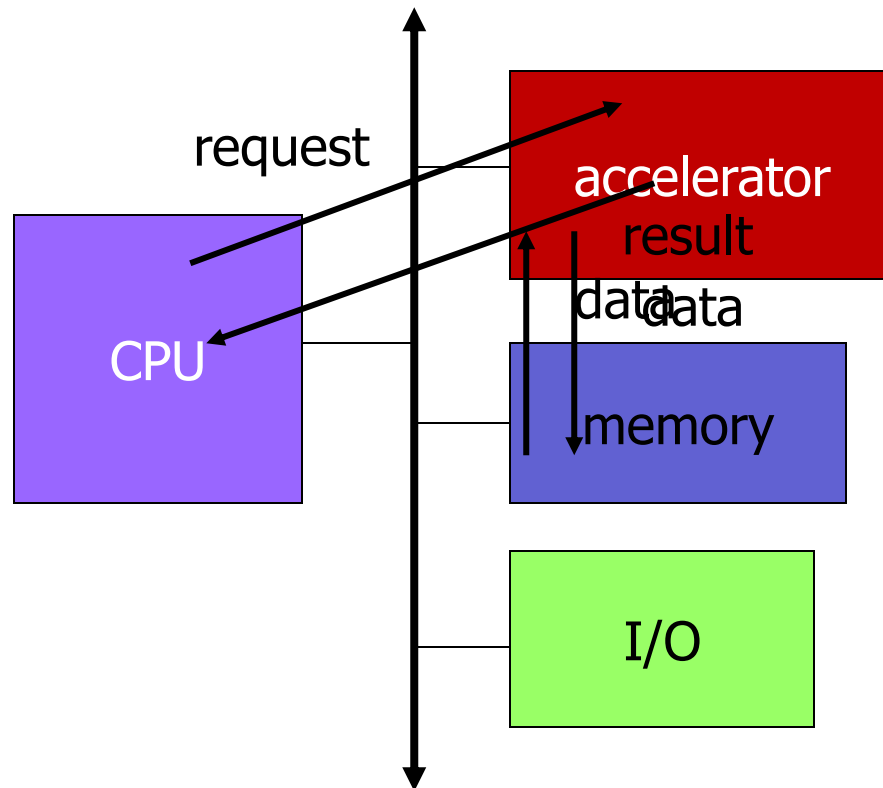
Xeon+FPGA chip will fit in the standard E5 LGA2011 socket, but the integrated FPGA will allow each chip to be customized to specific workloads. This move is almost certainly intended to make Intel-x86 a better all-round platform for a wider variety of workloads in enterprise and data center settings, and to dissuade customers from switching to GPGPU accelerators from the likes of Nvidia.

The Xeon+FPGA also raises the question of whether Intel would ever consider integrating an FPGA into its consumer Core line of chips — it's exceedingly unlikely, but it's hard to deny how awesome it would be if next-gen games and apps had access to an FPGA to speed up core processes. But more on that at the end of the story.

What is an FPGA?

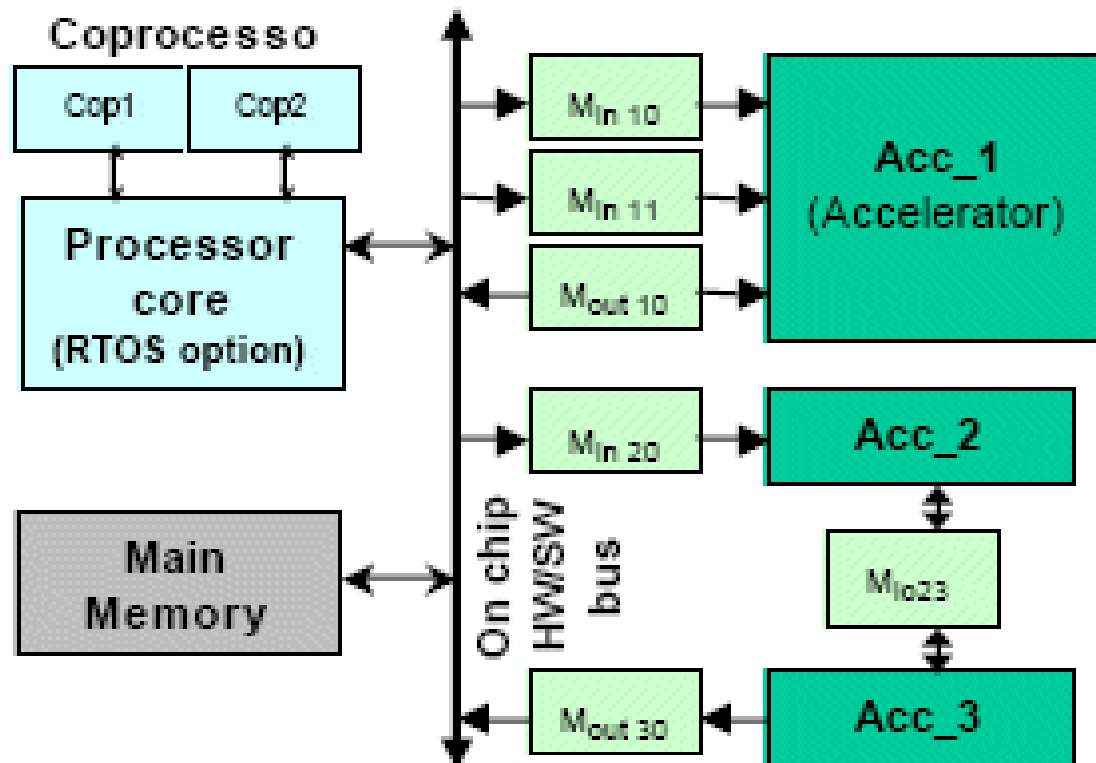
# Accelerated Systems

- Additional computational units dedicated to some functionality
- **Hardware/software co-design**: joint design of HW & SW architect



# Accelerator vs. Coprocessor

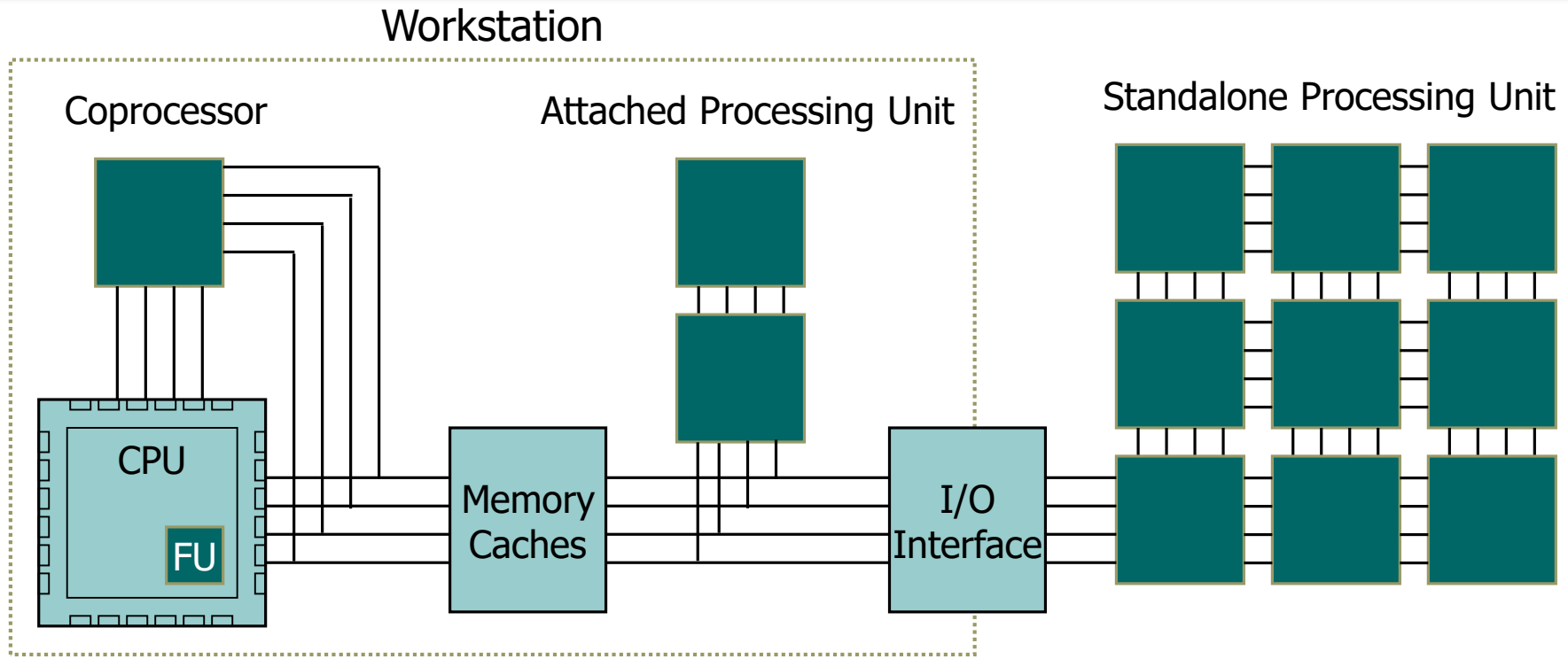
- A **coprocessor** executes instructions
  - Instructions are dispatched by the CPU
- An **accelerator** appears as a device on the bus
  - Typically controlled by registers (memory-mapped I/O)



# Accelerated System Design

- First, determine that the system really needs to be accelerated
  - How much faster will the accelerator on the core function?
  - How much data transfer overhead? Compute bound vs memory bound vs I/O bound?
- Design accelerator and system interface
- If tighter CPU integration required:
  - Create a functional unit for augmented instructions
  - Compiler techniques to identify/use new functional unit

# Accelerator Proximity



- Although self-reconfiguration is possible, some SW integration with a reconfigurable accelerator is almost always present
- CPU – FPGA proximity has implications for programming model, device capacity, I/O bandwidth

# Will Hardware Acceleration Help?

# Amdahl's Law

- The total application speedup, when an optimization (accelerator) improves a selected fraction ( $f$ ) of an application's execution time by a factor  $a$  is:

$$\textit{Application\_Speedup} = \frac{T_{org}}{[(1 - f) + f/a] * T_{org}} = \frac{1}{(1 - f) + f/a}$$



Gene Amdahl



# Will Hardware Acceleration Help?

$$Application\_Speedup = \frac{T_{org}}{T_{new}} = \frac{T_{org}}{T_{org} - (T_f - T_{fa})}$$

where:

- $T_{org}$  : Application original exec. time
- $T_{new}$  : Application exec. time after accel
- $f$  : fraction of the Application time that a **selected portion** of the App. runs.
- $T_f$  : amount of time the **selected portion** of the App. runs, before accel.
- $T_{fa}$  : amount of time the **selected portion** of the App. runs after accelerated by  $a$
- $a$  : factor by which accelerator speeds up **selected portion** of Application

$$= \frac{T_{org}}{T_{org} - (T_f - T_f/a)}$$

$$= \frac{T_{org}}{T_{org} - ([f * T_{org}] - [f * T_{org}]/a)}$$

$$= \frac{T_{org}}{(1 - ([f] - [f]/a)) * T_{org}}$$

$$= \frac{T_{org}}{(1 + (-[f] + [f]/a)) * T_{org}}$$

$$Application\_Speedup = \frac{T_{org}}{[(1 - f) + f/a] * T_{org}} = \frac{1}{(1 - f) + f/a}$$

# Amdahl's Law

- The total application speedup, when an optimization (accelerator) improves a selected fraction ( $f$ ) of an application's execution time by a factor  $a$  is:

$$\text{Application\_Speedup} = \frac{T_{org}}{[(1 - f) + f/a] * T_{org}} = \frac{1}{(1 - f) + f/a}$$

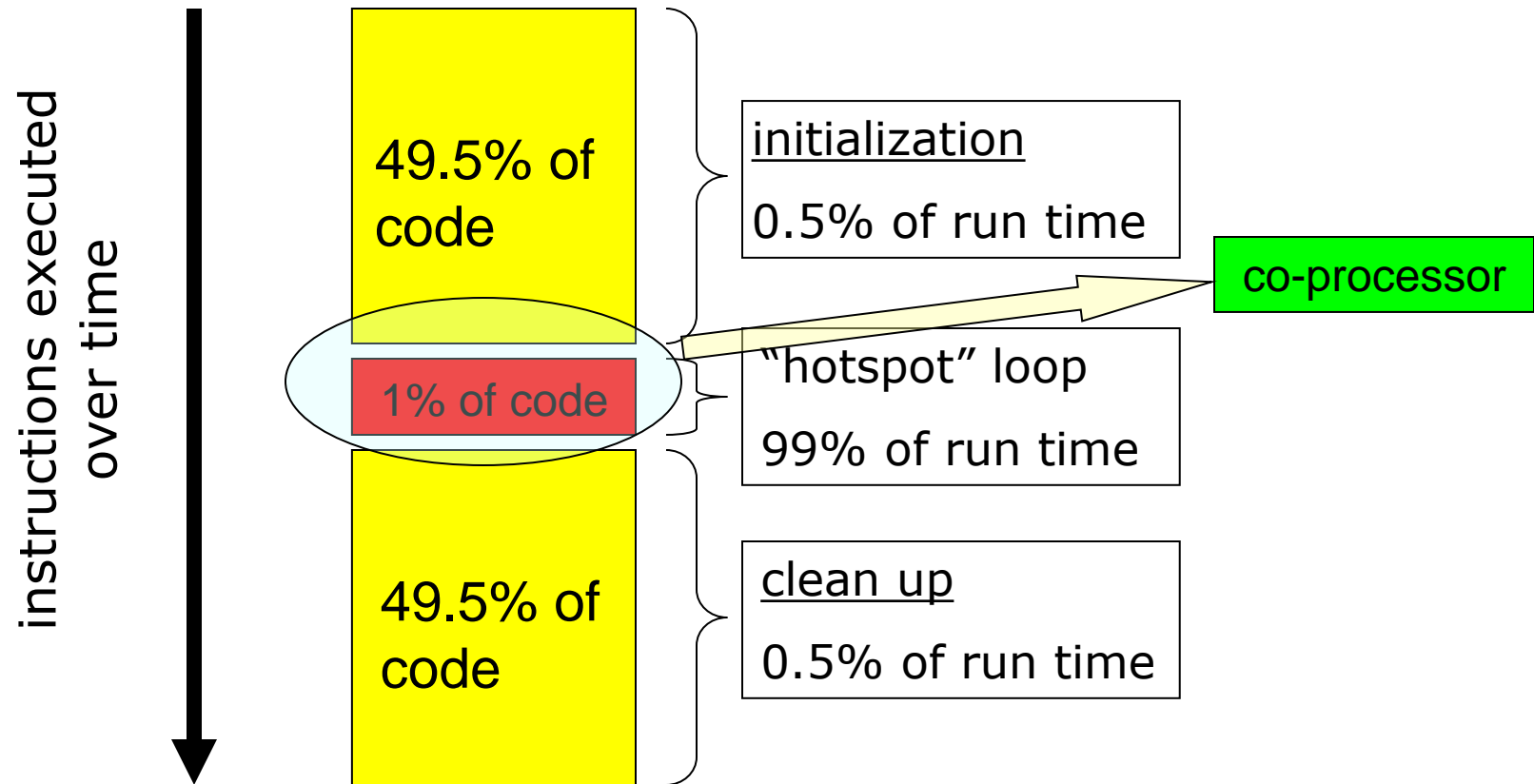
- This formula is known as Amdahl's Law, where:
  - $T_{org}$  is the execution time of the whole Application before any optimization/accelerator (i.e., original execution time)
  - $f$  is the fraction of the Application time that a **selected portion** of the Application takes to run.
  - $a$  is the factor by which an optimization/accelerator speeds up the **selected portion** of the Application
- Lessons from this law:
  - If  $f \rightarrow 1$  (i.e., 100%), then  $\text{Application\_Speedup} = a$
  - If  $a \rightarrow \infty$ , then  $\text{Application Speedup} = 1 / (1 - f)$
- Summary
  - Make the common case fast
  - Watchout for serial parts of an Application (ie, parts that cannot be accelerated)



Gene Amdahl

# Heterogeneous Execution Model

- Example: Assume an Application where only 1% of the code runs for 99% of the Application execution time (i.e.,  $f = .99$ ). How much will the overall Application speedup, if we create an accelerator to speedup this selected portion (i.e., "hotspot") by  $a$



# Heterogeneous Computing: Performance

- Move “**bottleneck/hotspot**” computation from software to hardware
- Example:
  - Application requires a **week** of CPU time (i.e., 168 hours)
  - One “**hotspot**” computation consumes **99%** of execution time

<b>Hotspot Speedup(a)</b>	<b>Application speedup</b>	<b>Execution time</b>
<b>50</b>	34	5.0 hours
<b>100</b>	50	3.3 hours
<b>200</b>	67	2.5 hours
<b>500</b>	83	2.0 hours
<b>1000</b>	91	1.8 hours

# Heterogeneous Computing: Performance

- Move “**bottleneck/hotspot**” computation from software to hardware
- Example:
  - Application requires a **week** of CPU time (i.e., 168 hours)
  - One “**hotspot**” computation consumes **99%** of execution time

<b>Design Time</b>	<b>Hardware Resources</b>	<b>Hotspot Speedup(a)</b>	<b>Application speedup</b>	<b>Execution time</b>
<b>1-week</b>	1-FPGA	<b>50</b>	34	5.0 hours
<b>2-months</b>	2-FPGA	<b>100</b>	50	3.3 hours
<b>6-months</b>	4-FPGA	<b>200</b>	67	2.5 hours
<b>2-years</b>	9-FPGA	<b>500</b>	83	2.0 hours
<b>4-years</b>	20-FPGA	<b>1000</b>	91	1.8 hours

Is the effort and resources needed to create the Hotspot accelerator worth the corresponding Application speedup?

# Heterogeneous Computing: Performance

- Move “**bottleneck/hotspot**” computation from software to hardware
- Example:
  - Application requires a **week** of CPU time (i.e., 168 hours)
  - One “**hotspot**” computation consumes **99%** of execution time

<b>Design Time</b>	<b>Hardware Resources</b>	<b>Hotspot Speedup(a)</b>	<b>Application speedup</b>	<b>Execution time</b>
<b>1-week</b>	1-FPGA	<b>50</b>	34	5.0 hours
<b>2-months</b>	2-FPGA	<b>100</b>	50	3.3 hours
<b>6-months</b>	4-FPGA	<b>200</b>	67	2.5 hours
<b>2-years</b>	9-FPGA	<b>500</b>	83	2.0 hours
<b>4-years</b>	20-FPGA	<b>1000</b>	91	1.8 hours

Is the effort and resources needed to create the Hotspot accelerator worth the corresponding Application speedup?

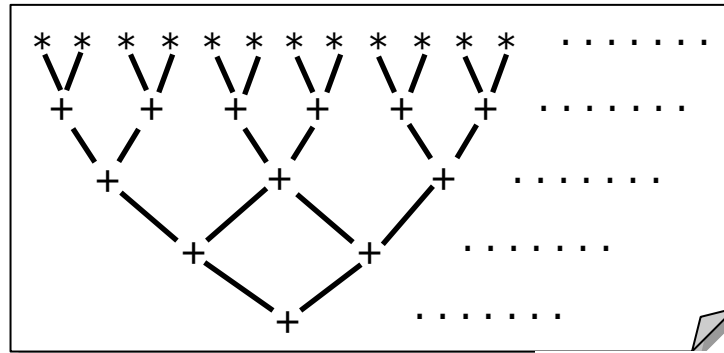
What is the best-case Application speed-up (i.e., App speedup if Hotspot is sped up  $\infty$ )?

# Hardware/Software Partitioning

C Code for FIR Filter

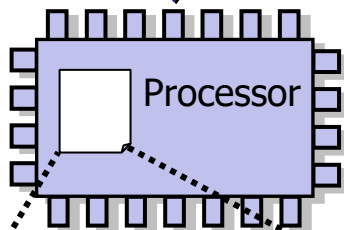
```
for (i=0; i < 16; i++)  
  y[i] += c[i] * x[i]  
..  
..  
..
```

Hardware 'for' loop

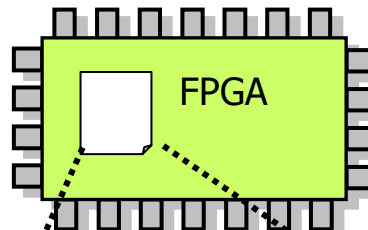


Designer creates custom accelerator using hardware design methodology

Compiler

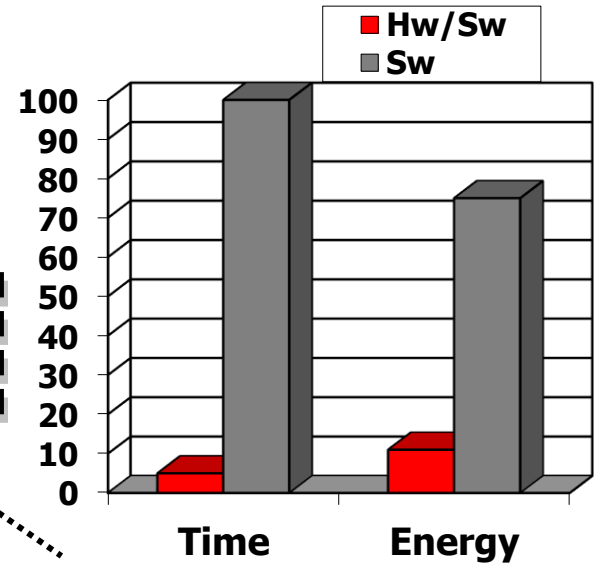


• ~1000 cycles



■ ~ 10 cycles

■ Speedup = 1000 cycles/ 10 cycles = 100x





# A Cause for Pessimism?

- HW amenability a design criterion for the Advanced Encryption Standard (AES)
  - Flurry of activity looking at AES on FPGA (1000s of implementations, opts, attacks)
  - J. Zambreno, D. Nguyen and A. Choudhary, "Exploring Area/Delay Tradeoffs in an AES FPGA Implementation". *Proc. of the Int'l Conference on Field-Programmable Logic and its Applications (FPL)*, Aug. 2004
    - Main contribution: an early exploration of the design decisions that lead to area/delay tradeoffs in an AES accelerator
    - Significant (at the time) throughput of 23.57 Gbps for AES-128E in ECB mode

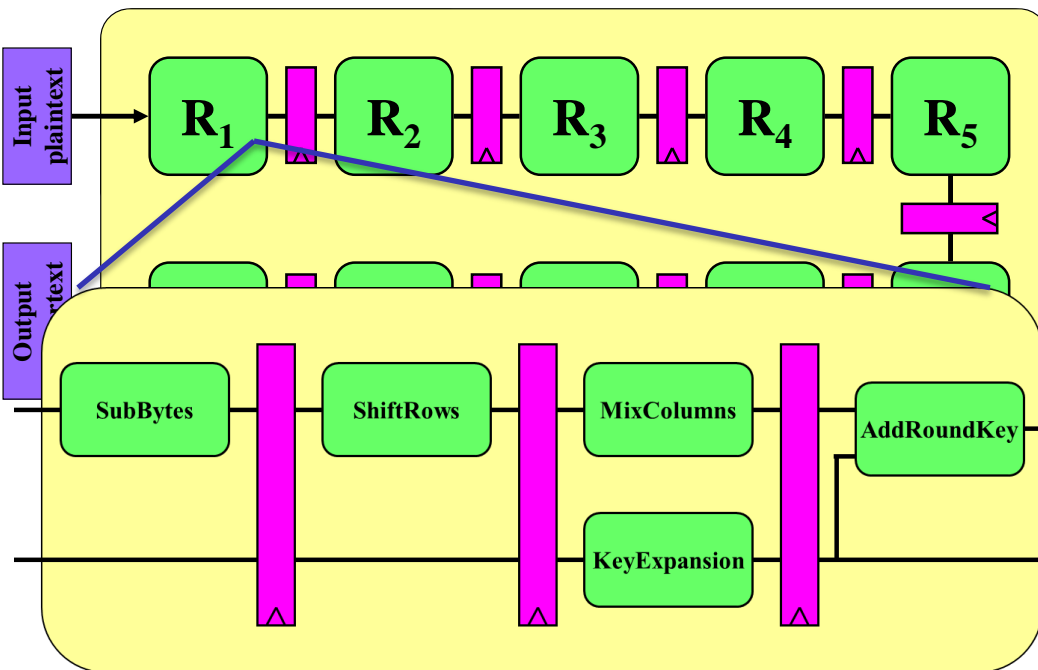
aesni\_encrypt:

```

    ..
.L000enc1_loop:
    aesenc    %xmm4,%xmm0
    decl     %ecx
    movups   (%edx),%xmm4
    leal    16(%edx),%edx
    jnz     .L000enc1_loop
    aesenclast %xmm4,%xmm0
    movups   %xmm0,(%eax)
    ret
    
```



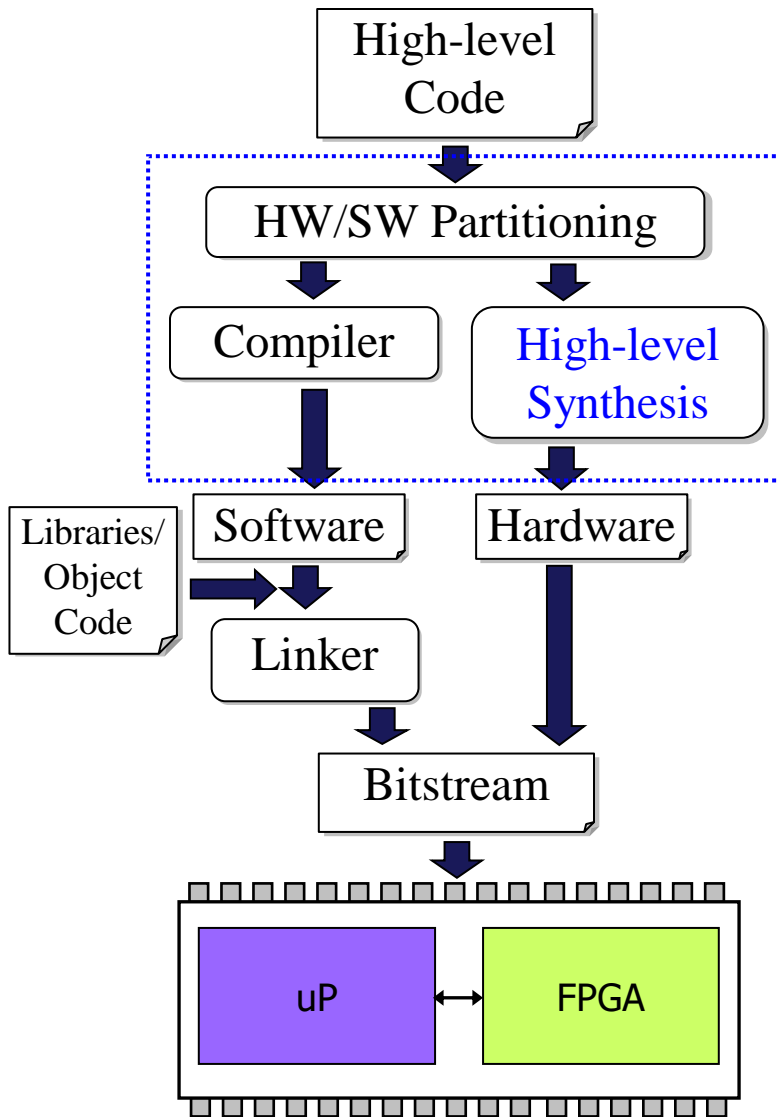
Partial libgcrypt implementation using Intel's AES-NI instruction set. Performance is ~0.75 cycles per byte (40+ Gbps per-thread with relatively easy software patches)



# Accelerator Design Challenges

- **Debugging** – how to properly test the accelerator separately, and then in conjunction with the rest of the system (hw/sw co-simulation)
- **Coherency** – how to safely share results between CPU & accelerator
  - Impact on cache design, shared memory
  - Solutions look similar to those for resource sharing in conventional operating systems, but are typically ad-hoc
- **Analysis** – determining the effects of any hardware parallelism on performance
  - Must take into account accelerator execution time, data transfer time, synchronization overhead
  - Heterogeneous multi-threading helps, but complicates design significantly
  - Overlapping I/O and computation (streaming)

# High-Level Synthesis



- **Problem**: Describing circuits using HDL is time consuming/difficult
- **Solution**: High-level synthesis
  - Create circuits from high-level code
  - Allows developers to use higher-level specification
  - Potentially, enables synthesis for software developers
- More on this in a bit

# Automation to the Rescue?

- Developer efficiency continues to be a limiting factor
- Numerous approaches to the “behavioral synthesis” problem of generating useful hardware from high-level descriptions
  - **C-to-HDL variants:**
    - Handel-C (Oxford)
    - ROCCC (UC-Riverside)
    - Catapult C (Mentor Graphics)
    - SystemC (Acclera)
    - Cynthesizer C (Cadence)
    - ImpulseC (Impulse)
  - **Many other comparable approaches:**
    - HDL Coder (Mathworks)
    - Vivado High-Level Synthesis (Xilinx)
    - Bluespec, SystemVerilog
- Opinion: these tools can automate certain classes of logic, BUT:
  - Cannot generate efficient output for “hard problems”
  - Unfamiliar / uncomfortable syntax for both SW and HW engineers
  - Similar algorithmic challenges to auto-parallelizing compilers
  - **Sorry students, you’re still learning VHDL** 😞



New RCL grad student seen trying to escape the lab

# The Right Stuff

- Applications that map well to FPGA-based acceleration tend to have common characteristics:
  - Significant kernels of computation, significant data
    - Amdahl's law is typically more relevant than Gustafson's law
    - *Profile. Don't Speculate.* – Daniel Bernstein
  - Fixed-point or integer data representation
    - If application is Gflop-constrained, use a GPU
    - Changing (slowly), see Altera Stratix 10-based systems
  - Fine-grained parallelism
    - But if working set fits in cache, will still be hard to beat x86(MHz for MHz)
    - Systolic model of computation, where FPGA pipeline depth > equivalent CPU depth and number of FPGA PEs >> number of x86 FUs
  - Real-time constraints, system integration
    - HPC workloads should go on HPCs (i.e. accelerators are an orthogonal concern)
    - Current GPUs cannot make useful latency guarantees

# Typical Application Acceleration Methodology

Algorithmic Understanding

- What is the purpose of the application?
- Can its complexity be lowered?

Application Profiling

- Where are the application's "hotspots"?

System-Level Design

- What hardware and software infrastructure is needed?

Architectural Design

- How can I accelerate the bottlenecks?

HW / SW Co-Design

- How does it all fit?

Integration and Test

# Acknowledgments

- M. C. Herbordt et al., "Achieving High Performance with FPGA-Based Computing," in *Computer*, vol. 40, no. 3, pp. 50-57, March 2007
  - [http://www.bu.edu/caadlab/IEEE\\_Computer\\_07.pdf](http://www.bu.edu/caadlab/IEEE_Computer_07.pdf)

**Table 1. HPC/FPGA application design techniques.**

<b>Type of support required</b>	<b>Methods supported</b>
Electronic design automation: languages and synthesis	Use rate-matching to remove bottlenecks Take advantage of FPGA-specific hardware Use appropriate arithmetic precision Create families of applications, not point solutions Scale application for maximal use of FPGA hardware
Function/arithmetic libraries	Use appropriate FPGA structures Use appropriate arithmetic mode
Programmer/designer FPGA awareness	Use an algorithm optimal for FPGAs Use a computing mode appropriate for FPGAs Hide latency of independent functions Minimize use of high-cost arithmetic operations
None	Living with Amdahl's law



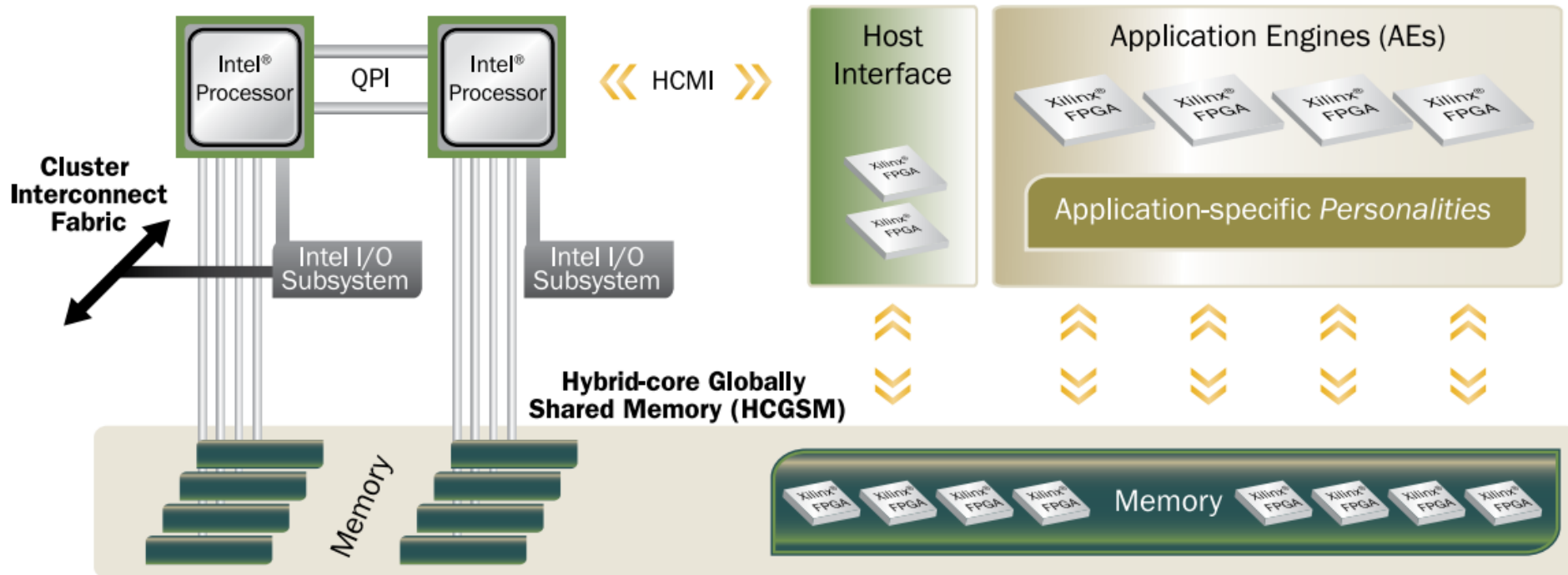




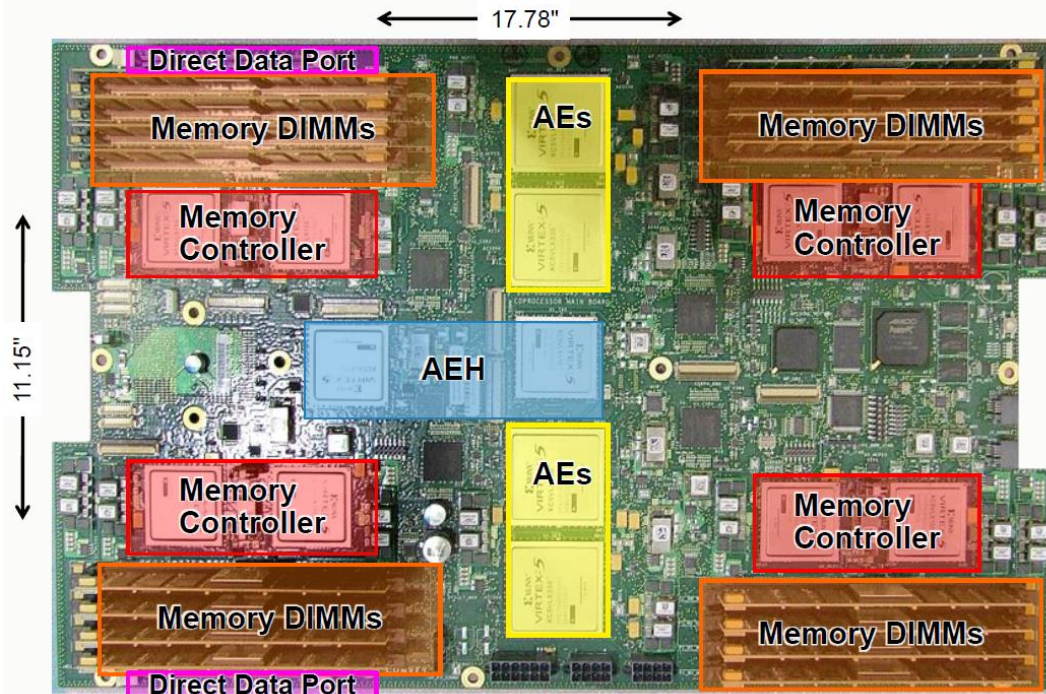


# Example High Performance Reconfigurable Platform

- Convey HC-2ex “hybrid core” computer:
  - Tightly coupled host-coprocessor via HCMI
  - Globally shared, high performance memory (80 GB/s)
  - Four Application Engines (AEs)
- Can be used as a large vector processor, or with custom “personalities” that accelerate arbitrary applications
  - Support for all interfacing logic, hardware/software co-simulation, early prototyping, performance profiling
  - Conventional C/C++ (host) and VHDL/Verilog (coprocessor) for development

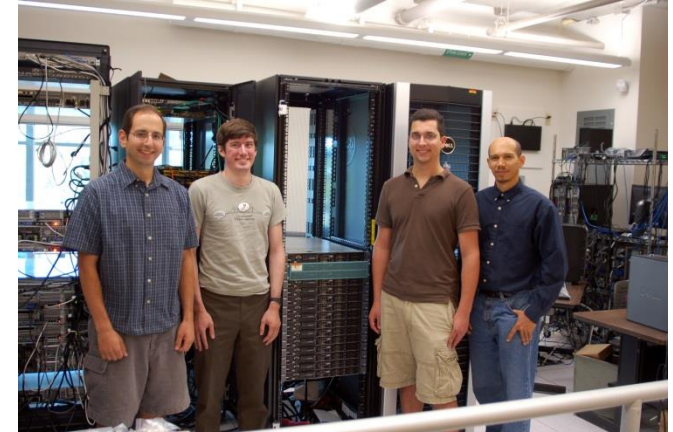


# Other Views

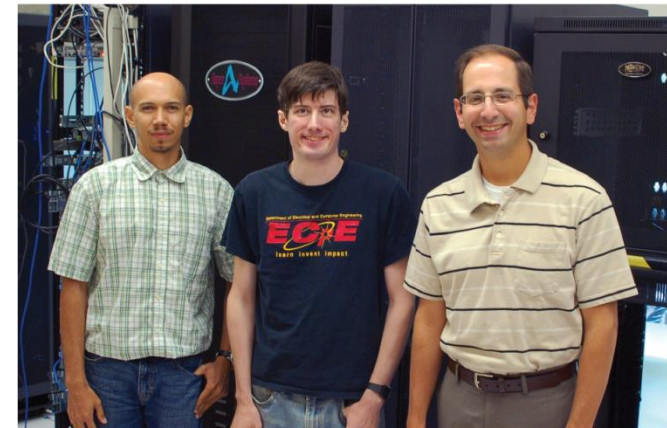


- 14 FPGAs in total:
  - 4 FPGAs used as the AEs (Xilinx Virtex 5-LX330s operating at 150 MHz)
  - 8 FPGAs are used as Memory Controllers
  - 2 FPGAs are used as the Application Engine Hub to interface with the CPU subsystem

“Iowa State University Students Win MemoCODE Design Contest Using Convey Computer HC-1”, *MarketWire*, July 2012



“Team from Iowa State Wins 2014 MemoCODE Design Contest”, *PRWeb*, Oct. 2014



# Acknowledgments

- These slides are inspired in part by material developed and copyright by:
  - Marilyn Wolf (Georgia Tech)
  - Jason Bakos (University of South Carolina)
  - Greg Stitt (University of Florida)
  - Hadi Esmaeilzadeh (Georgia Tech)

# Acknowledgments

**Table 1. HPC/FPGA application design techniques.**

<b>Type of support required</b>	<b>Methods supported</b>
Electronic design automation: languages and synthesis	Use rate-matching to remove bottlenecks Take advantage of FPGA-specific hardware Use appropriate arithmetic precision Create families of applications, not point solutions Scale application for maximal use of FPGA hardware
Function/arithmetic libraries	Use appropriate FPGA structures Use appropriate arithmetic mode
Programmer/designer FPGA awareness	Use an algorithm optimal for FPGAs Use a computing mode appropriate for FPGAs Hide latency of independent functions Minimize use of high-cost arithmetic operations
None	Living with Amdahl's law