

Accelerators Development

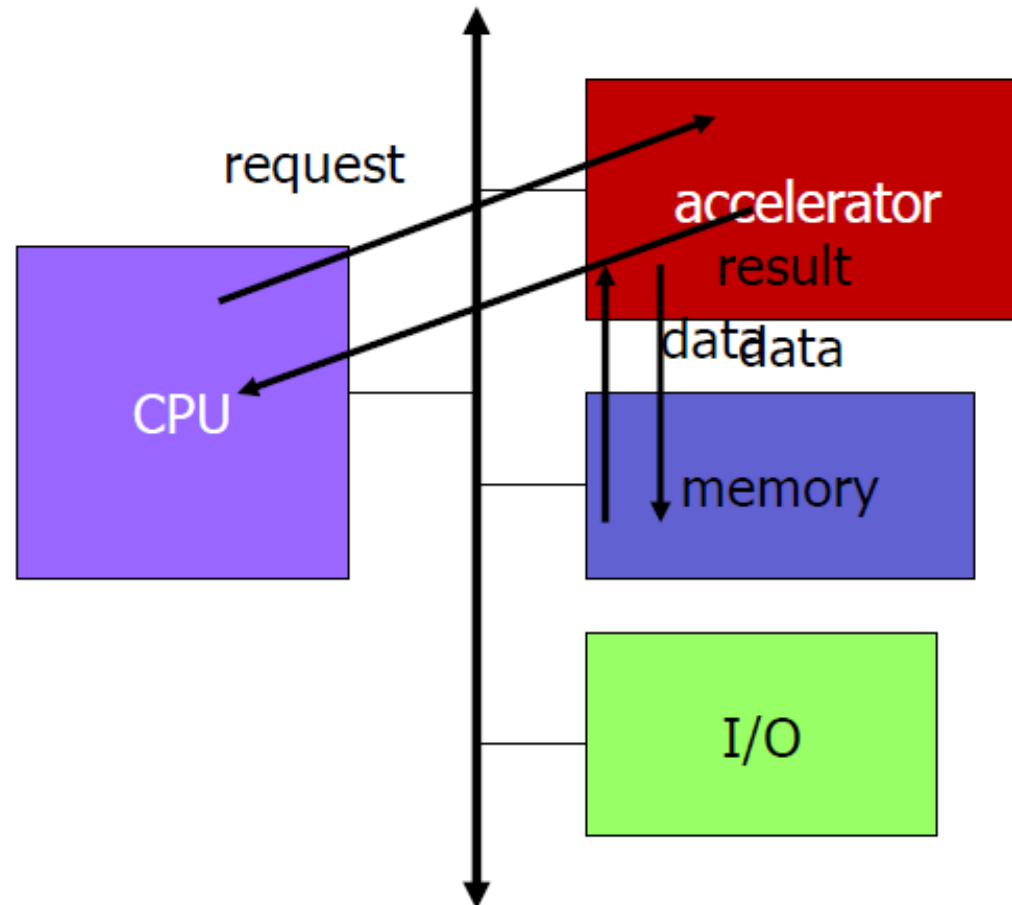
26 Feb 2020

Outline

- **Motivation**
- **Accelerators coupling to CPUs**

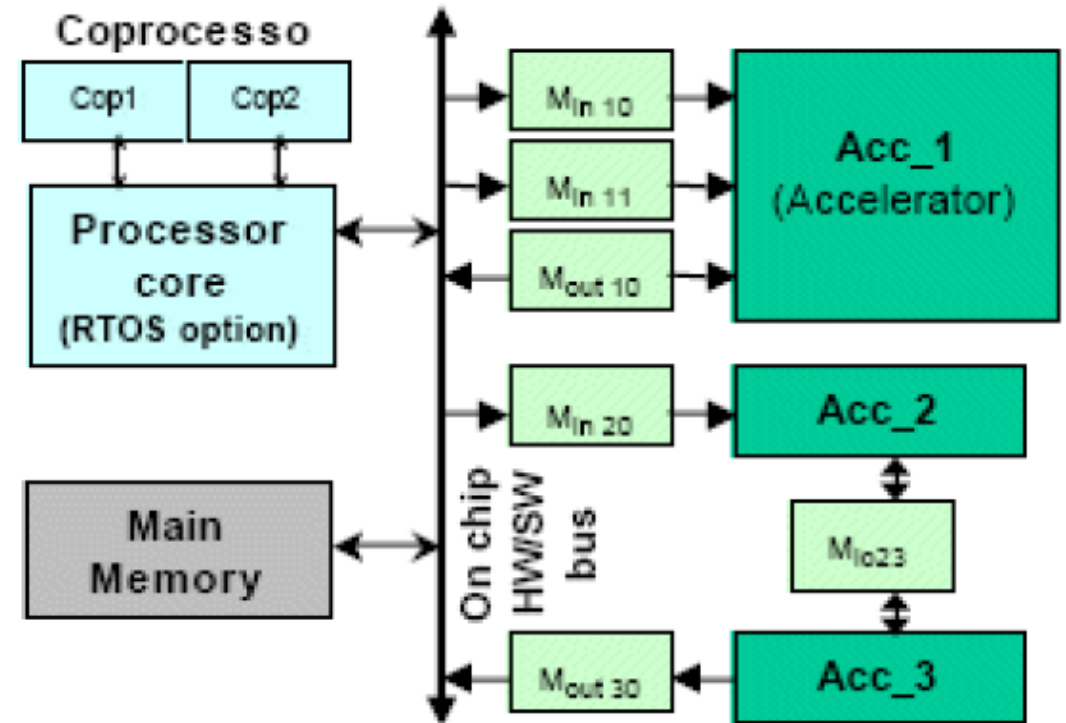
Accelerated Systems

- Use additional computational units dedicated to some functionality
- Hardware/software co-design: joint design of hardware and software architectures



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 IO)



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

Amdahl's Law

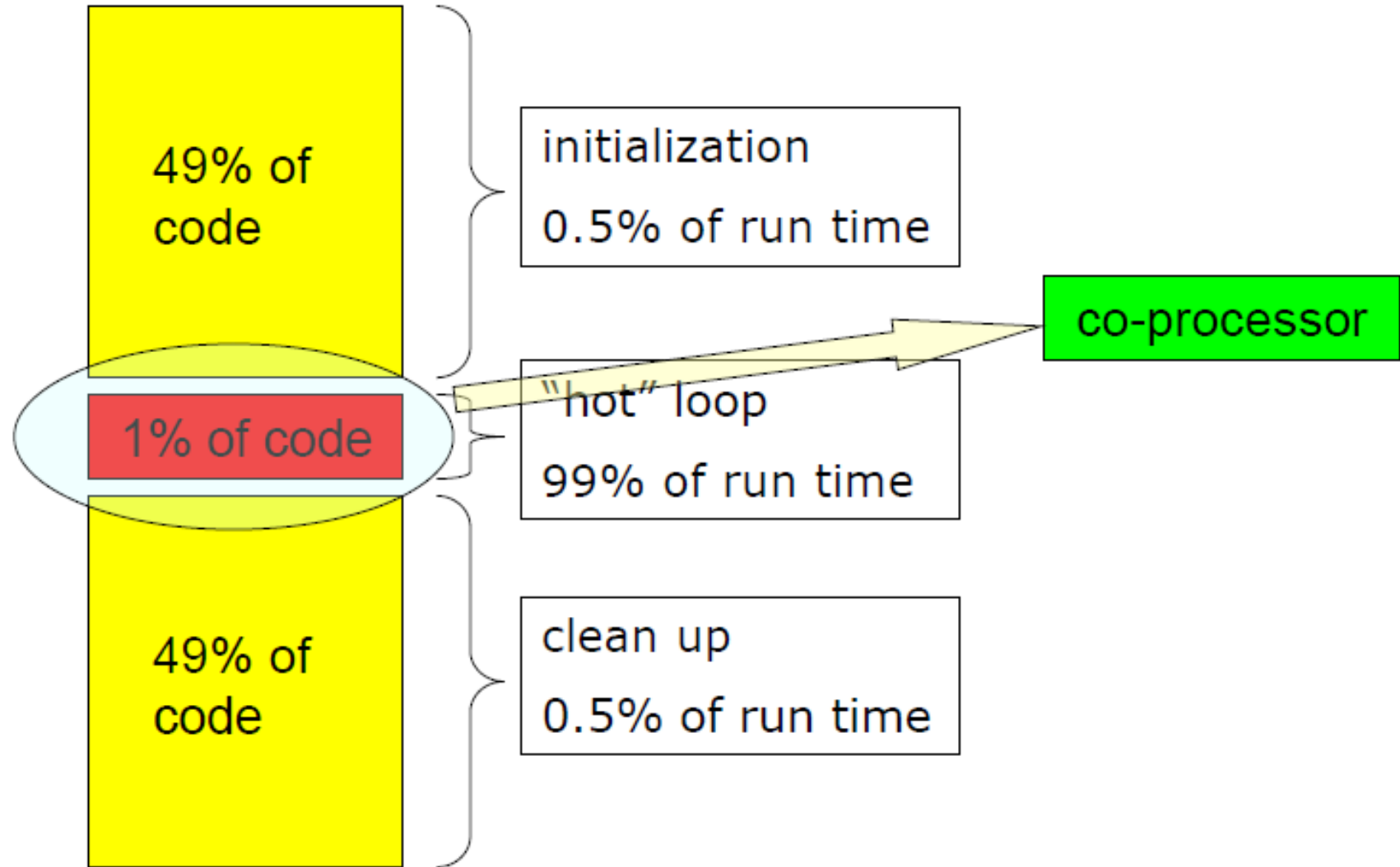
- If an optimization improves a fraction f of execution time by a factor of a

$$\text{Speedup} = \frac{T_{old}}{[(1-f) + f/a] \times T_{old}} = \frac{1}{(1-f) + f/a}$$

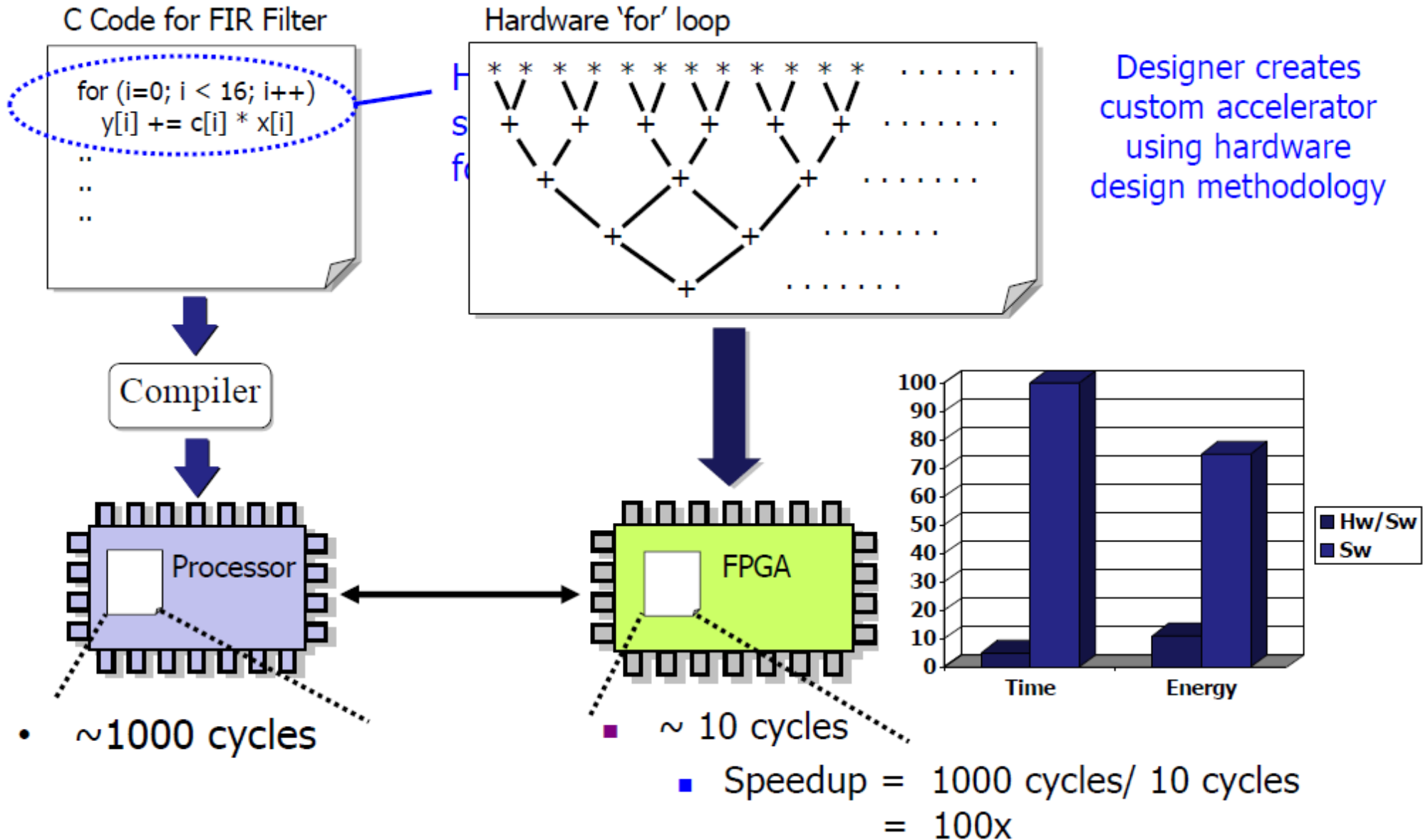
- This formula is known as Amdahl's Law
- Lessons from
 - If $f \rightarrow 100\%$, then speedup = a
 - If $a \rightarrow \infty$, the speedup = $1 / (1 - f)$
- Summary
 - Make the common case fast
 - Watch out for the non-optimized component
-

Heterogeneous Execution Model

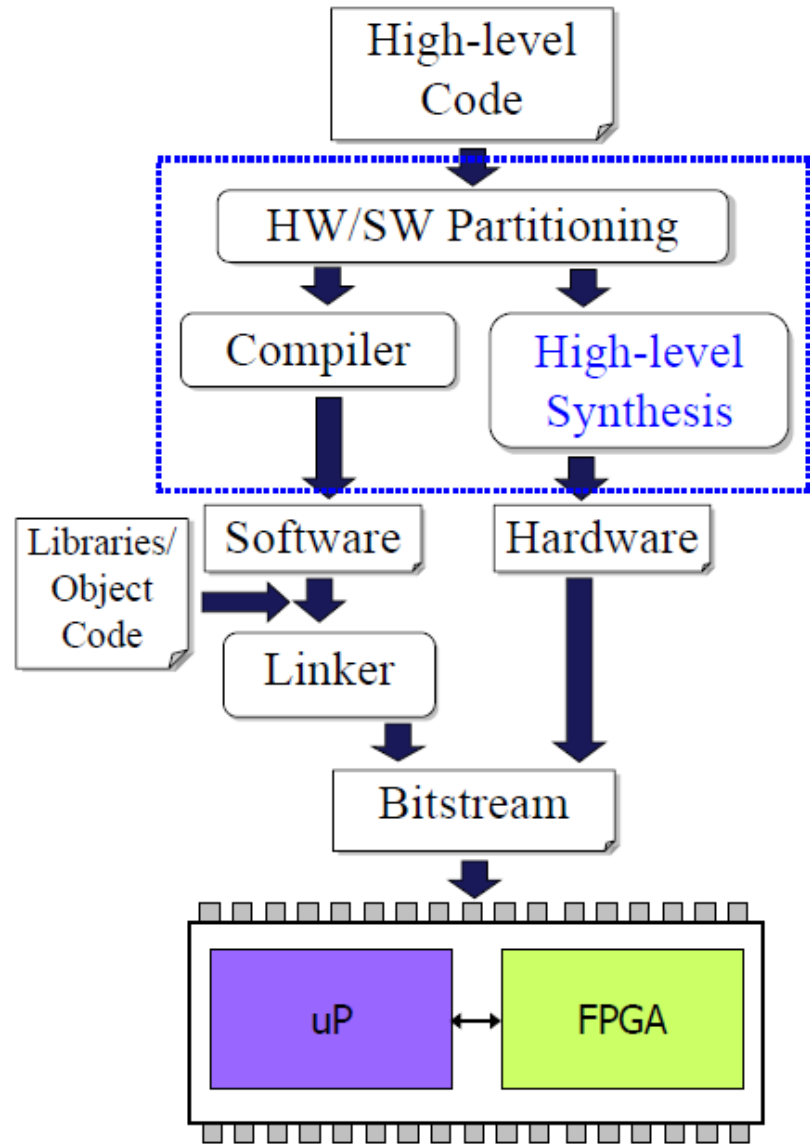
instructions executed over time



Hardware / Software Partitioning



High-Level Synthesis

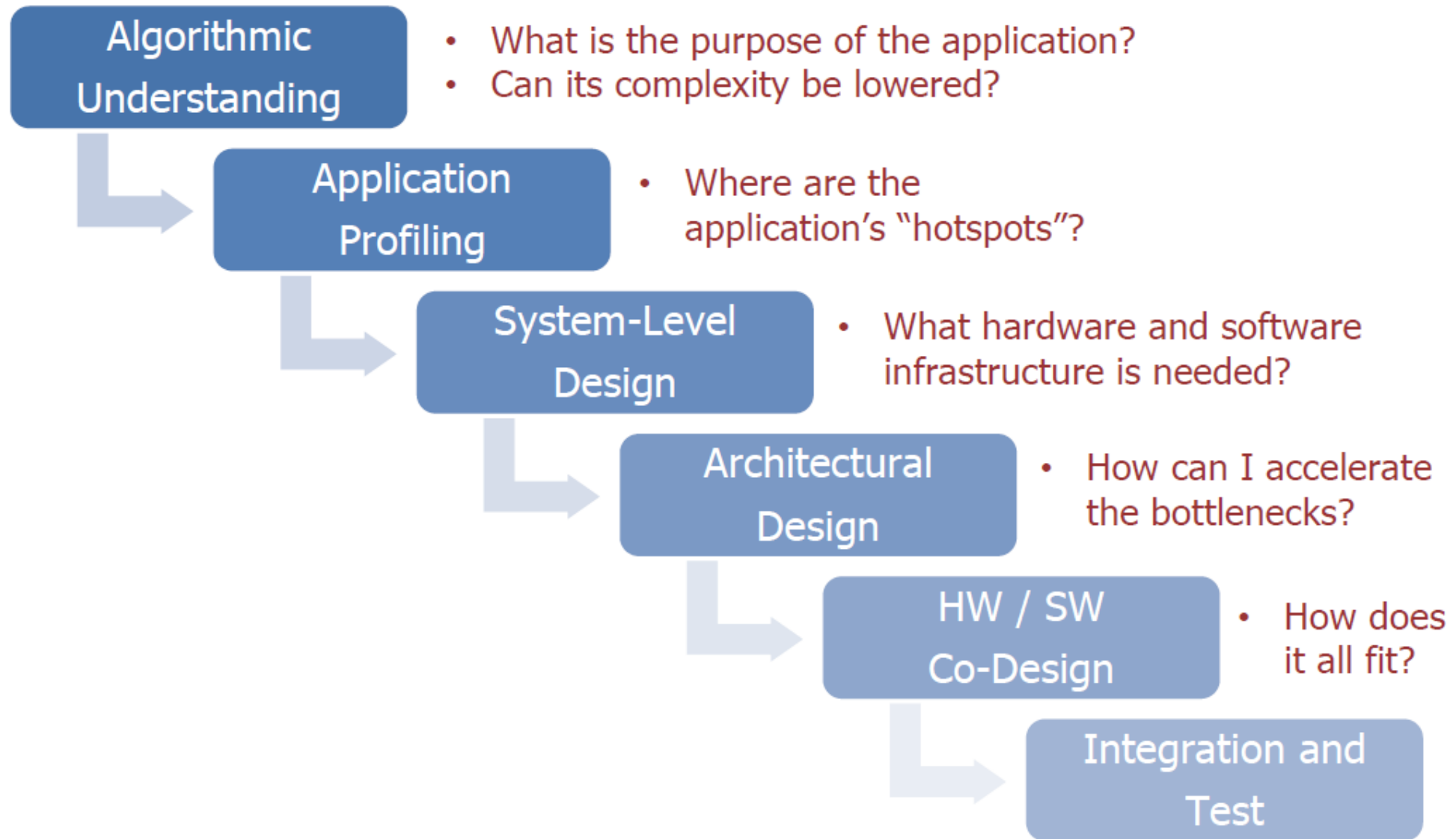


- *Problem:* Describing circuit using HDL is time consuming/difficult
- *Solution:* High-level synthesis
 - Create circuit from high-level code
 - Allows developers to use higher-level specification
 - Potentially, enables synthesis for software developers
- More on this in a bit

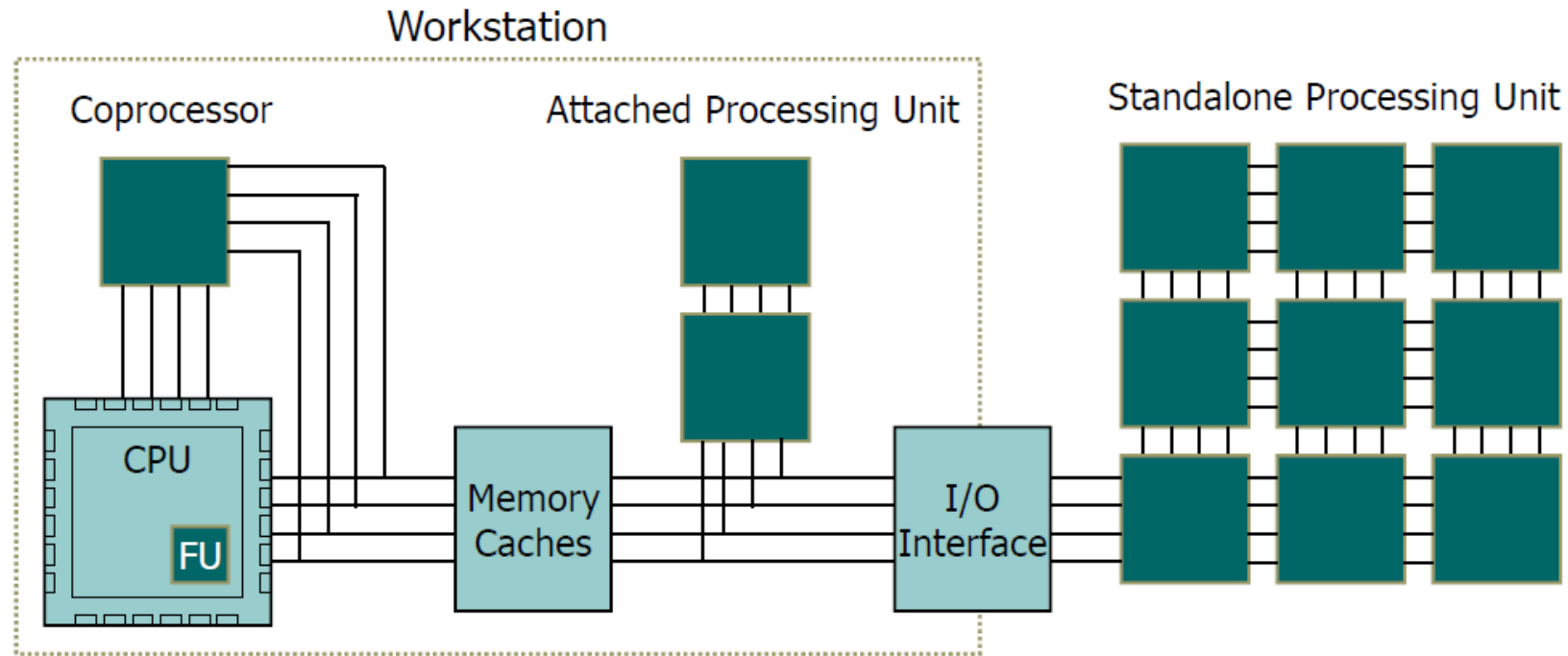
Accelerator Design Challenges

- 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 and accelerator
 - Impact on cache design, shared memory
 - Solutions looks 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)

Typical Application Acceleration Methodology



Accelerator Proximity



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

SoC: Xilinx ZYNQ (2013)

Xilinx Platform Studio (EDK_P.68d) - /remote/rs/zambreno/dev/rc1/hw-controls/Systems/Zedboard-sample/system/system.xmp - [System Assembly View]

File Edit View Project Hardware Device Configuration Debug Simulation Window Help

Navigator IP Catalog

Design Flow

- Run DRCs
- Implement Flow
- Generate Netlist
- Generate BitStream
- Export Design
- Simulation Flow
- Generate HDL Files
- Launch Simulator

IP Catalog

Description	IP Version	IP Type
EDK Install		
Analog		
Arithmetic		
Bus and Bridge		
Clock, Reset and Interrupt		
Communication High-Speed		
Communication Low-Speed		
DMA and Timer		
Debug		
FPGA Reconfiguration		
General Purpose IO		
Interprocessor Communication		
Memory and Memory Controller		
PCI		
Peripheral Controller		
Processor		
Utility		
Verification		
Video and Image Processing		
Project Local PCores		

Processing System (PS)

Application Processor Unit (APU)

- NEON™/FPU Engine
- Cortex™-A9 MPCore™ CPU
- MMU
- 32 KB I Cache
- 32 KB D Cache
- Snoop Control Unit
- 512 KB L2 Cache & Controller
- 256 KB OCM
- BootROM
- OCM Interconnect
- Memory Interfaces
- DDR2/3, LPDDR2 Controller
- High Performance AXI 32b/64b Slave Ports
- XADC
- PCie Gen2

Programmable Logic (PL)

- Bank0 MIO (15:0)
- Bank1 MIO (63:16)
- I/O MUX (MIO)
- I/O Peripherals: SPI 0, SPI 1, I2C 0, I2C 1, CAN 0, CAN 1, UART 0, UART 1, GPIO, SD 0, SD 1, USB 0, USB 1, Enet 0, Enet 1
- FLASH Memory Interfaces: SRAM/NOR, NAND, Quad SPI
- SMC Timing Calculation
- Clock Generation
- Extended MIO (EMIO)
- PS to PL Clock Ports
- 32b GP AXI Master Ports
- 32b GP AXI Slave Ports
- DMA Channels
- Config AES/SHA
- IRQ
- DMA Sync
- DEVC
- DAP
- CoreSight Components
- GIC
- DMA Channel
- SWDT
- TTC
- System Level Control Regs
- Reset
- General Settings
- 64b AXI ACP Slave Port
- Select I/O

AMBA® Connection Legend

- Arrow direction shows control, Data flows both directions
- Configurable AXI3 32 bit/64 bit
- AXI3 64 bit / AXI3 32 bit / AHB 32 bit / APB 32 bit

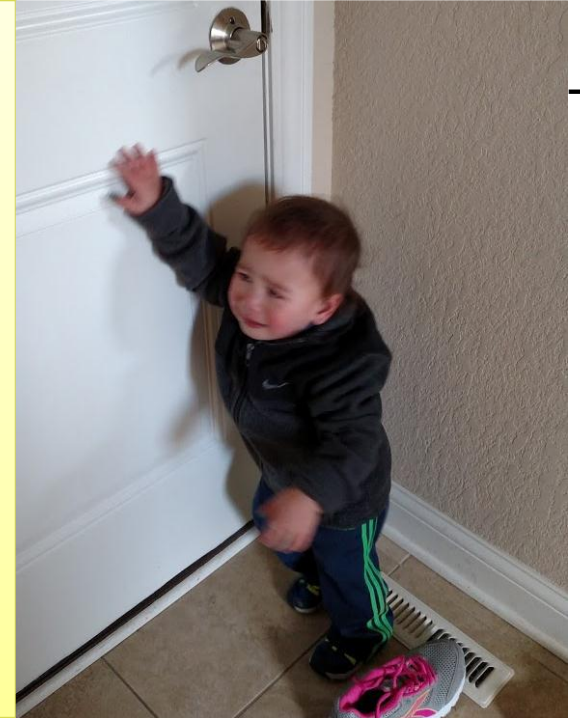
Console

```
WARNING:EDK:4092 - IPNAME: processing_system7, INSTANCE: processing_system7_0 - Pre-Production version not verified on hardware for architecture 'zynq' - /remote/rs/zambreno/dev/rc1/hw-conf
WARNING:EDK:4092 - IPNAME: processing_system7, INSTANCE: processing_system7_0 - Pre-Production version not verified on hardware for architecture 'zynq' - /remote/rs/zambreno/dev/rc1/hw-conf
```

Solution: Automation (?)

- Developer effort
 - Numerous tools
 - level description
 - C-to-HDL
 - Handel-C
 - ROCCC (RISC Open Compiler for Custom Computing)
 - Catapult (Synopsys)
 - SystemC
 - Cynthesizer (Synopsys)
 - ImpulseC (Synopsys)
 - 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....

Escaping the Lab ...???



- Many other comparable approaches:
 - HDL Coder (Mathworks)
 - Vivado High-Level Synthesis (Xilinx)
 - Bluespec, SystemVerilog

Important Remarks

- 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

System-level Design

