# 058165 - PARALLEL COMPUTING

Fabrizio Ferrandi

a.a. 2022-2023

# Acknowledge

**Material from:**

Parallel Computing  lectures from prof. Kayvon and prof. Olukotun Stanford University

# Heterogeneous processing

**Observation: most "real world" applications have complex workload characteristics**

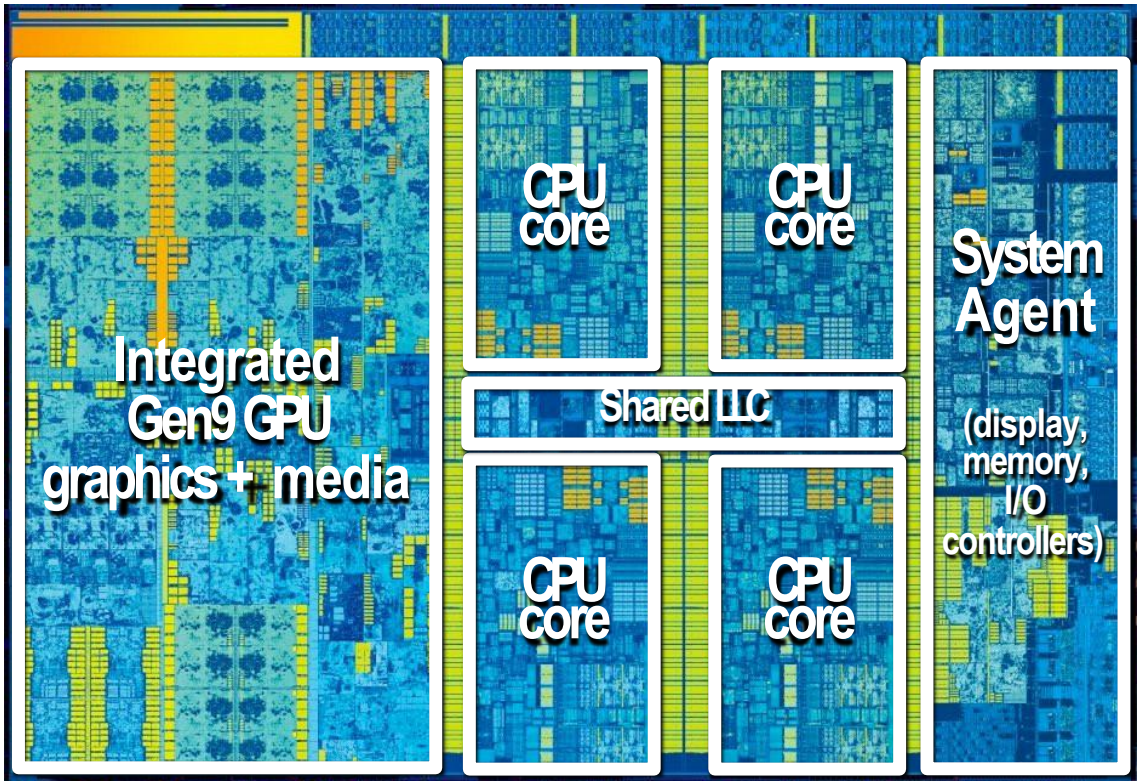
| | |
|---|---|
| They have components that can be widely parallelized. | And components that are difficult to parallelize. |
| They have components that are amenable to wide SIMD execution. | And components that are not. (divergent control flow) |
| They have components with predictable data access | And components with unpredictable access, but those accesses might cache well. |

**Idea: the most efficient processor is a heterogeneous mixture of resources ("use the most efficient tool for the job")**

# Examples of heterogeneity

# Example: Intel “Skylake" (2015)

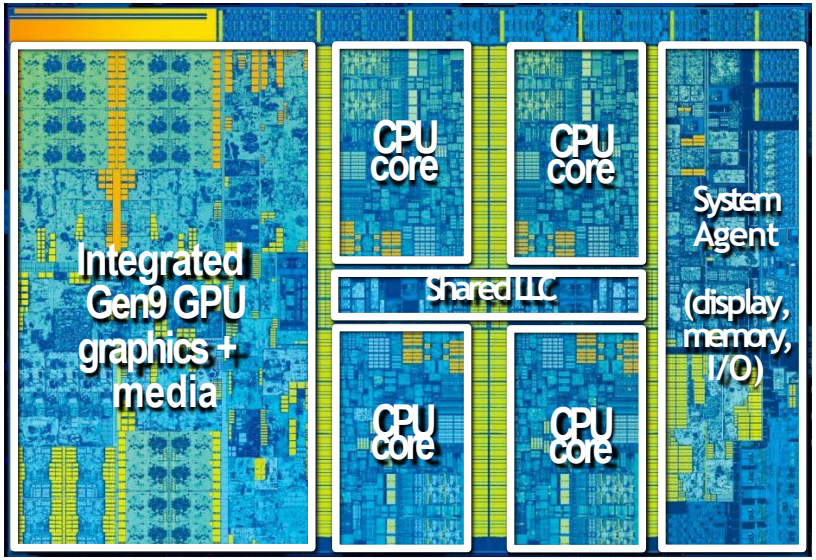
**(6th Generation Core i7 architecture)**




**4 CPU cores + graphics cores + media accelerators**

# Example: Intel “Skylake" (2015)

(6th Generation Core i7 architecture)




- CPU cores and graphics cores share same memory system
- Also share LLC (L3 cache)
  - Enables, low-latency, high- bandwidth communication between CPU and integrated GPU
- Graphics cores are cache coherent with CPU cores

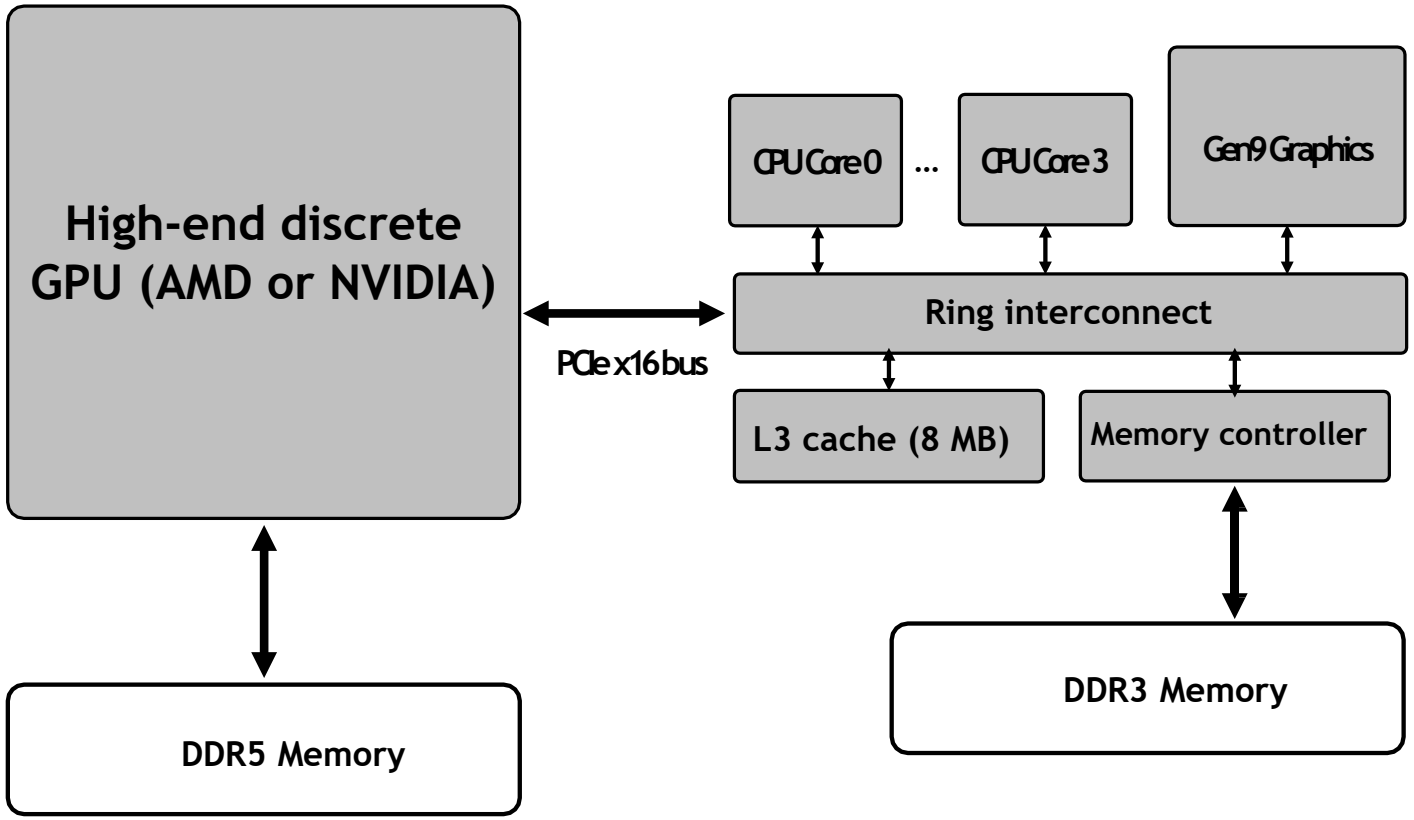
# Intel’s 12th generation


10nm ESF/Intel 7 Alder Lake die shot (~209mm²) from Intel via Andreas Schilling on Twitter:
https://twitter.com/aschilling/status/1453391035577495553

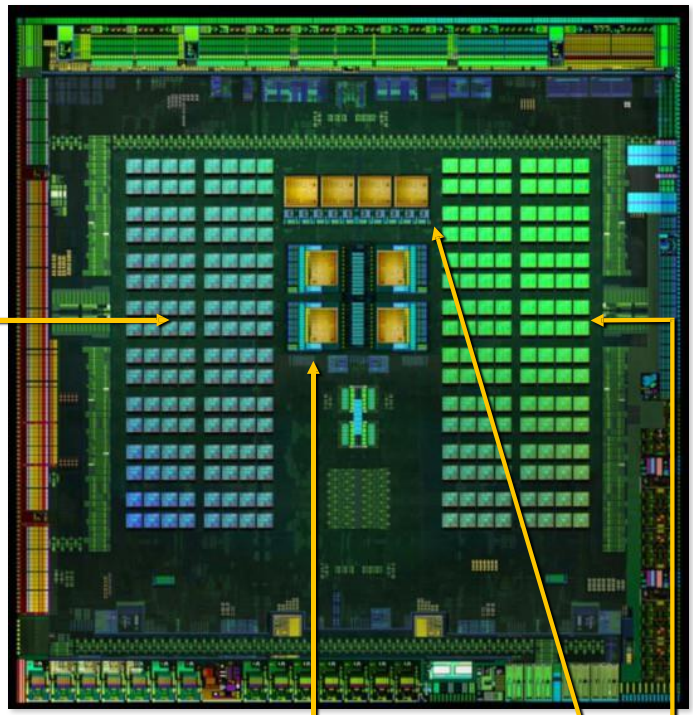
Die shot interpretation by Locuza, October 2021

# More heterogeneity: add discrete GPU

**Keep discrete (power hungry) GPU unless needed for graphics-intensive applications Use integrated, low power graphics for basic graphics/window manager/UI**



# Mobile heterogeneous processors



NVIDIA Tegra X1
Four ARM Cortex A57 CPU cores for applications
Four low performance (low power) ARM A53 CPU cores
One Maxwell SMM (256 "CUDA" cores)

A11 image credit: TechInsights Inc.'
* Disclaimer: estimates by TechInsights, not an official Apple reference.


Apple A11 Bionic *
Two "high performance" 64 bit ARM CPU cores
Four "low performance" ARM CPU cores
Three "core" Apple-designed GPU
Image processor
Neural Engine for DNN acceleration
Motion processor

# GPU-accelerated supercomputing


**Summit (at Oak Ridge National Lab)**
**(world's #1 in Fall 2018)**
**9,216 IBM Power9 22-core CPUs**
**27,648 NVIDIA V100 GPUs**
**10 Petabytes DRAM**

# Frontier supercomputer (world's #1 in Fall 2022)

**Hewlett Packard Enterprise Frontier, or OLCF-5, is the world's first exascale supercomputer**
Oak Ridge Leadership Computing Facility


- 9,472 AMD Epyc 7A53s "Trento" 64 core 2 GHz CPUs (606,208 cores)
- 37,888 Radeon Instinct MI250X GPUs (8,335,360 cores)
- Frontier consumes 21 MW
- Speed: 1.102 exaFLOPS (Rmax) / 1.685 exaFLOPS (Rpeak)

# Energy-constrained computing

# Performance and Power

$$Power = \frac{Ops}{second} \times \frac{Joules}{Op}$$

Performance

Energy efficiency

FIXED

What is the magnitude of improvement from specialization?

**Specialization (fixed function) ⇒ better energy efficiency**

**Pursuing highly efficient processing…**
**(specializing hardware beyond just parallel CPUs and GPUs)**

# Efficiency benefits of compute specialization

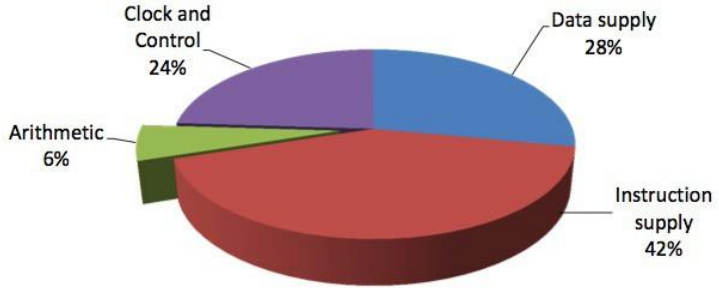
- Rules of thumb: compared to high-quality C code on CPU...
- Throughput-maximized processor architectures: e.g., GPU cores
  - Approximately 10x improvement in perf / watt
  - Assuming code maps well to wide data-parallel execution and is compute bound

- Fixed-function ASIC ("application-specific integrated circuit")
  - Can approach 100-1000x or greater improvement in perf/watt
  - Assuming code is compute bound and is not floating-point math

[Source: Chung et al. 2010 , Dally 08]

# Consider the complexity of executing an instruction on a modern processor…

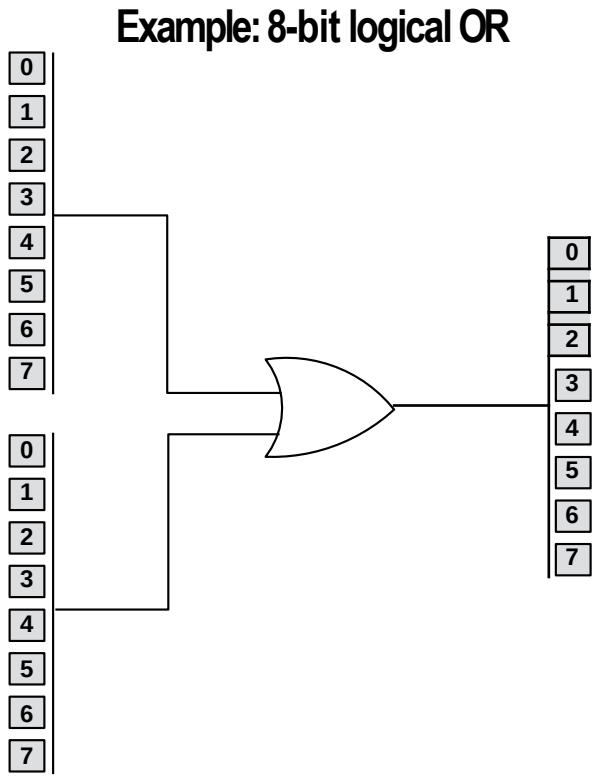
Read instruction — Address translation, communicate with icache, access icache, etc.
Decode instruction — Translate op to uops, access uop cache, etc.
Check for dependencies/pipeline hazards
Identify available execution resource
Use decoded operands to control register file SRAM (retrieve data)
Move data from register file to selected execution resource
Perform arithmetic operation
Move data from execution resource to register file
Use decoded operands to control write to register file SRAM




*Efficient Embedded Computing [Dally et al. 08]*
[Figure credit Eric Chung]

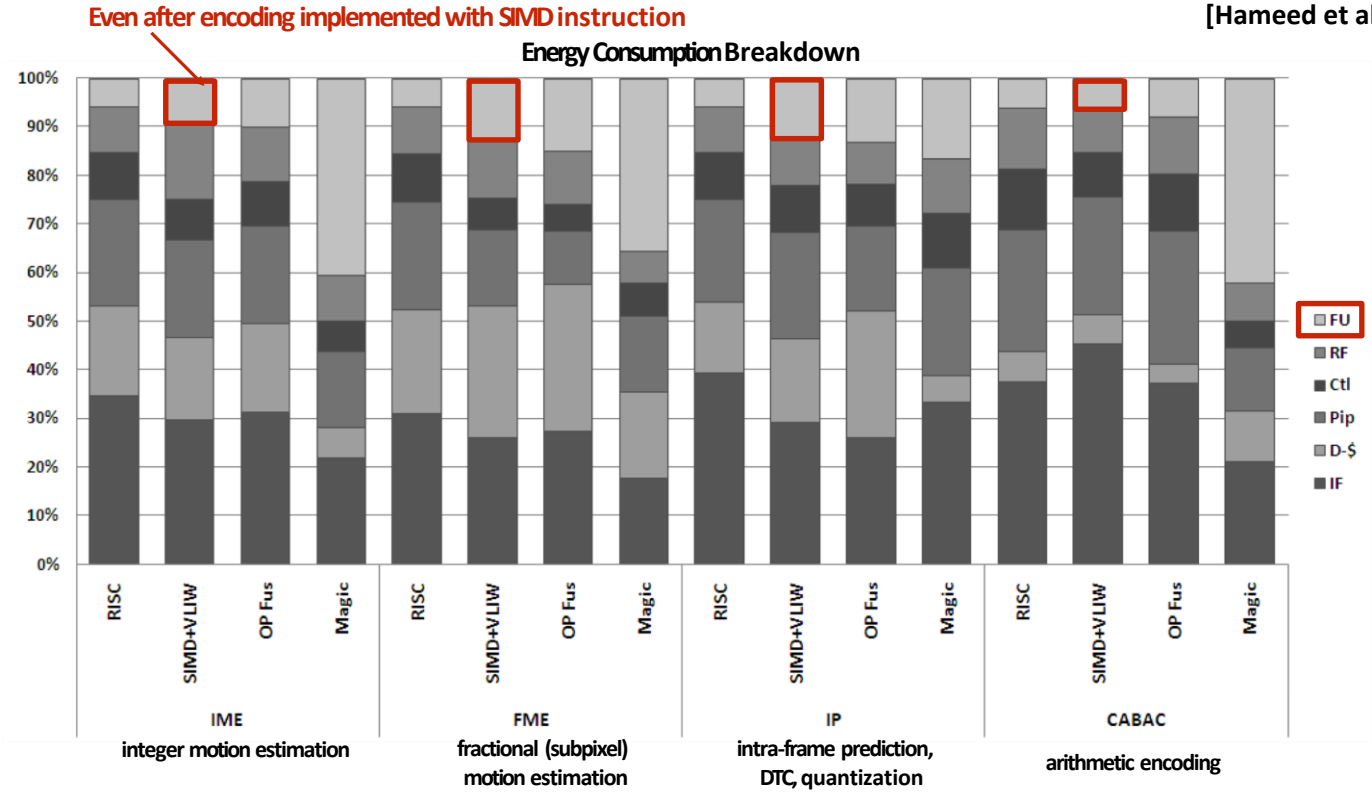
**Review question:**
**How does SIMD execution reduce overhead of certain types of computations?**
**What properties must these computations have?**

# Contrast that complexity to the circuit required to actually perform the operation



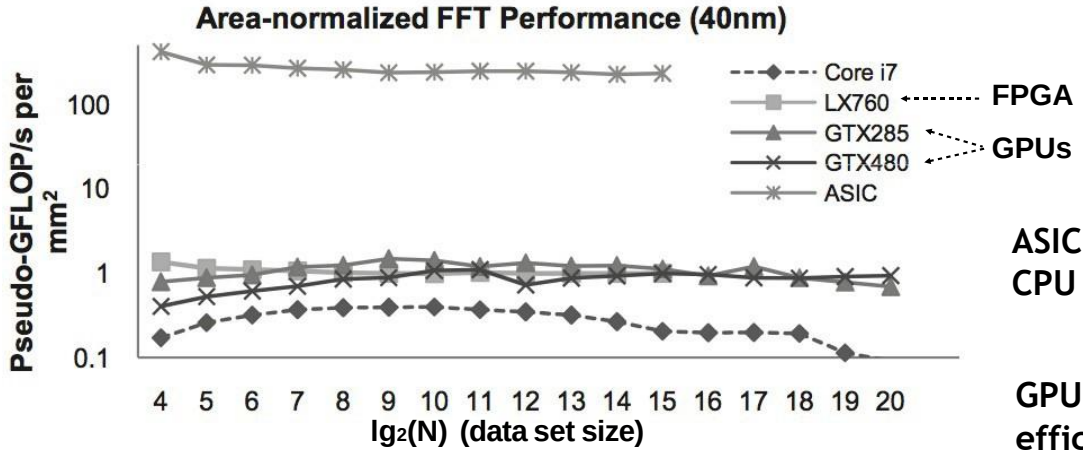
# H.264 video encoding: fraction of energy consumed by functional units is small (even when using SIMD)

[Hameed et al. ISCA 2010]



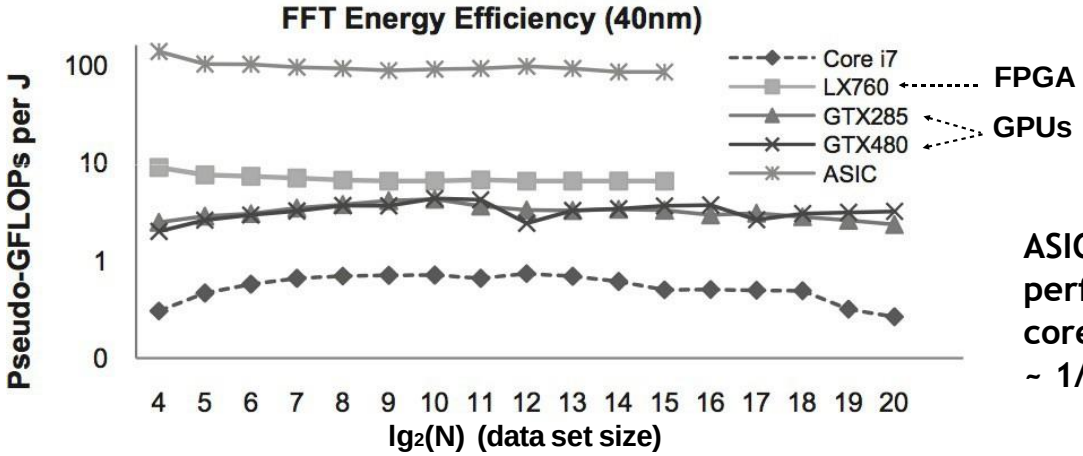

| | |
|---|---|
| FU = functional units | Pip = pipeline registers (interstage) |
| RF = register fetch | D-$ = data cache |
| Ctrl = misc pipeline control | IF = instruction fetch + instruction cache |

# Fast Fourier transform (FFT): throughput and energy benefits of specialization




ASIC delivers same performance as one CPU core with ~ 1/1000th the chip area.

GPU cores: ~ 5-7 times more area efficient than CPU cores.




ASIC delivers same performance as one CPU core using only ~ 1/100th the energy

[Chung et al. MICRO 2010]

# Mobile: benefits of increasing efficiency

- **Run faster for a fixed period of time**
  - Run at higher clock, use more cores (reduce latency of critical task)
  - Do more at once

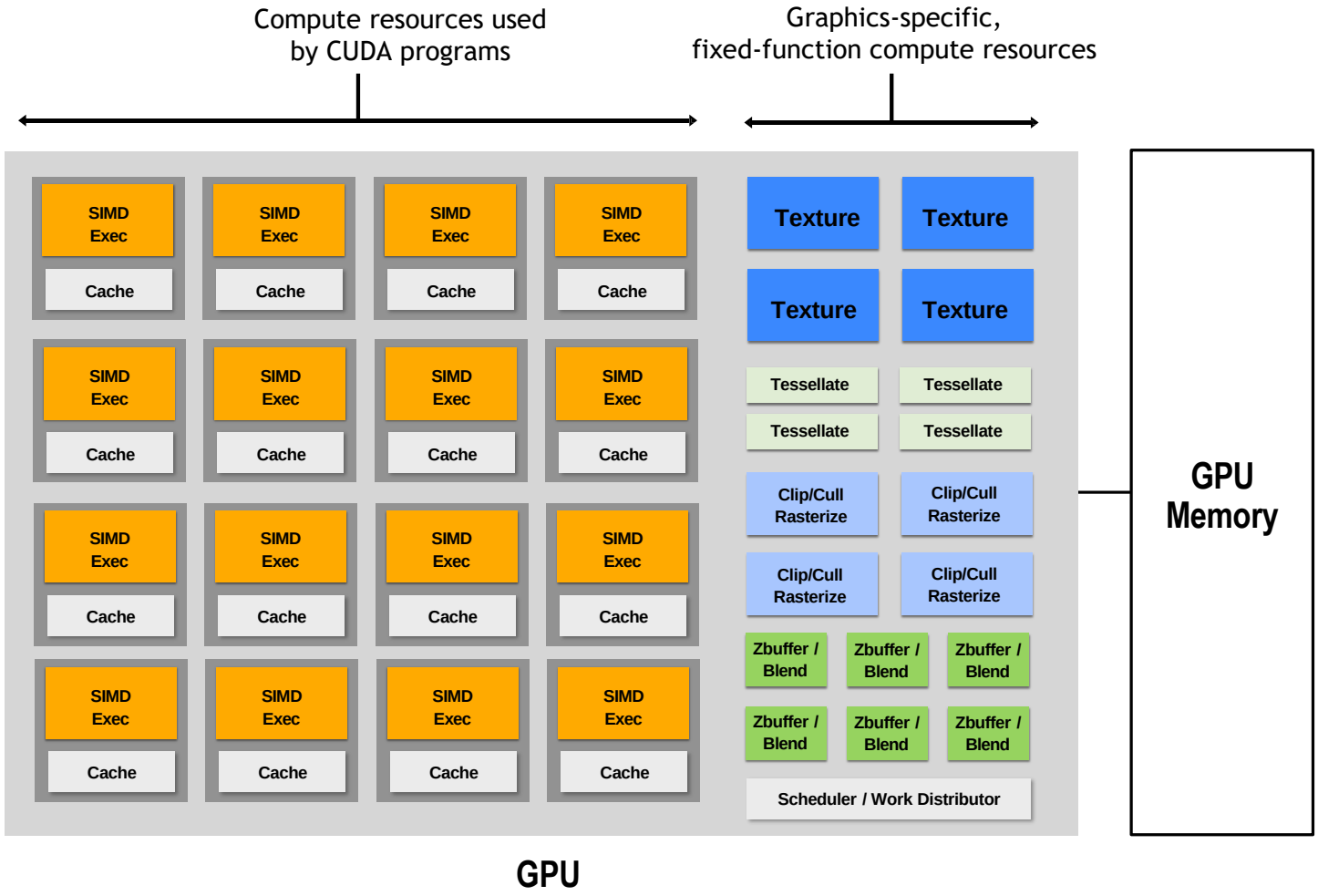
- **Run at a fixed level of performance for longer**
  - e.g., video playback, health apps
  - Achieve “always-on” functionality that was previously impossible


iPhone:
Siri activated by button press or holding phone up to ear


Amazon Echo / Google Home Always listening

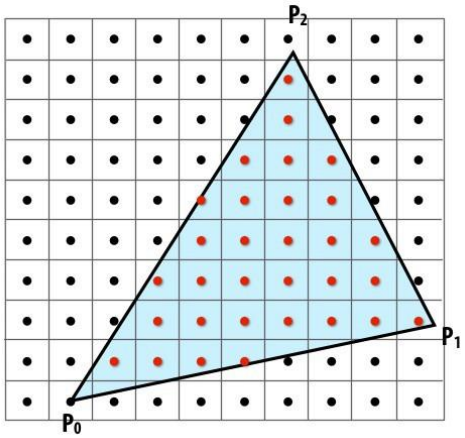
Google Glass: ~40 min recording per charge (nowhere near “always on”)

# GPU's are themselves heterogeneous multi-core processors

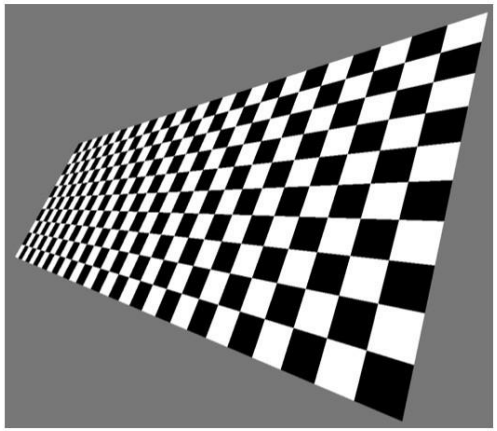


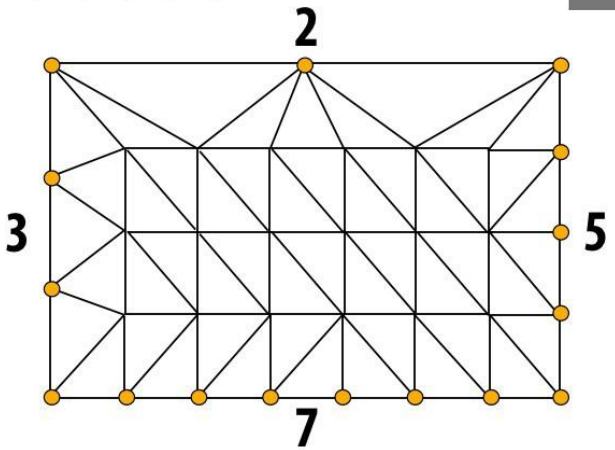
# Example graphics tasks performed in fixed-function HW

**Rasterization:**
**Determining what pixels a triangle overlaps**




**Texture mapping:**
**Warping/filtering images to apply detail to surfaces**






**Geometric tessellation:**
**computing fine-scale geometry from coarse geometry**

# Digital signal processors (DSPs)

**Programmable processors, but simpler instruction stream control paths**

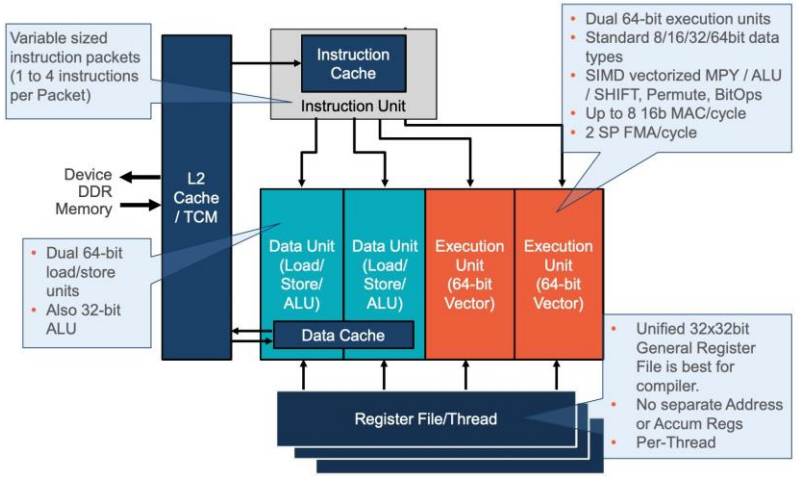
**Complex instructions (e.g., SIMD/VLIW): perform many operations per instruction (amortize cost of control)**

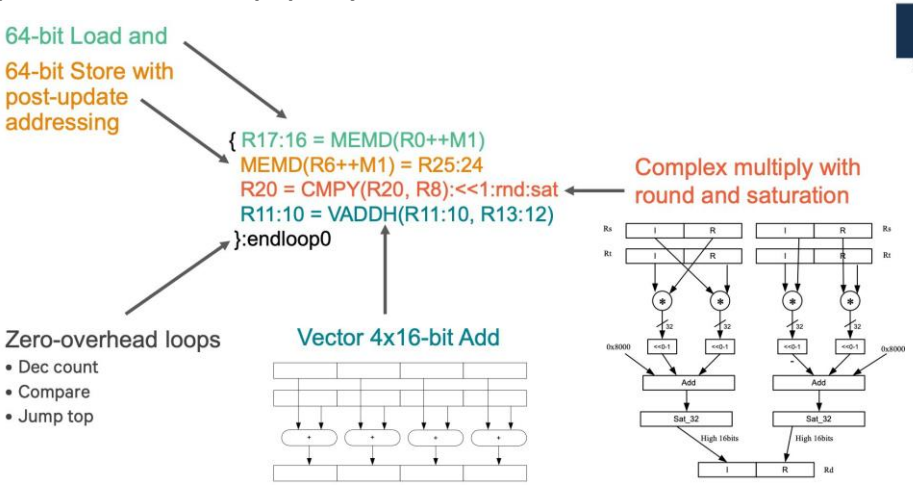
## Example: Qualcomm Hexagon DSP

**Used for modem, audio, and (increasingly) image processing on Qualcomm Snapdragon SoC processors**

**VLIW: "very-long instruction word"**
**Single instruction specifies multiple different operations to do at once (contrast to SIMD)**

**Below: innermost loop of FFT**
**Hexagon DSP performs 29 "RISC" ops per cycle**



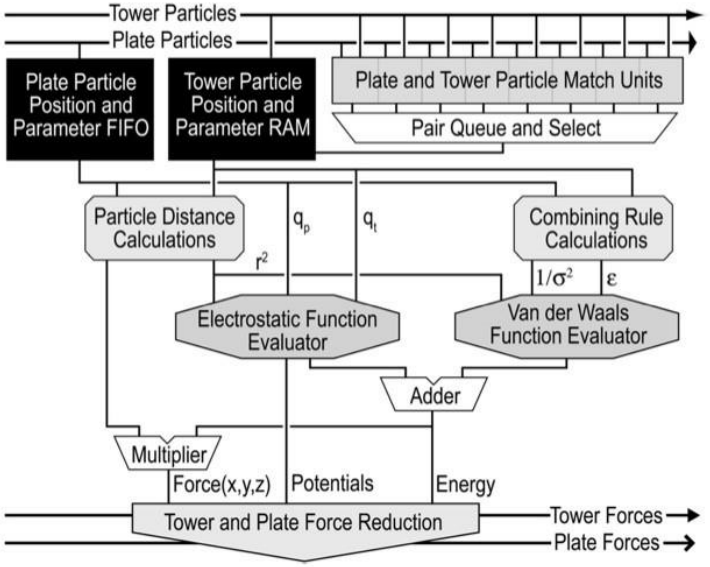

**Hexagon DSP is in Google Pixel phone**

# Anton supercomputer for molecular dynamics

[Developed by DE Shaw Research]

- Simulates time evolution of proteins
- ASIC for computing particle-particle interactions (512 of them in machine)
- Throughput-oriented subsystem for efficient fast-fourier transforms
- Custom, low-latency communication

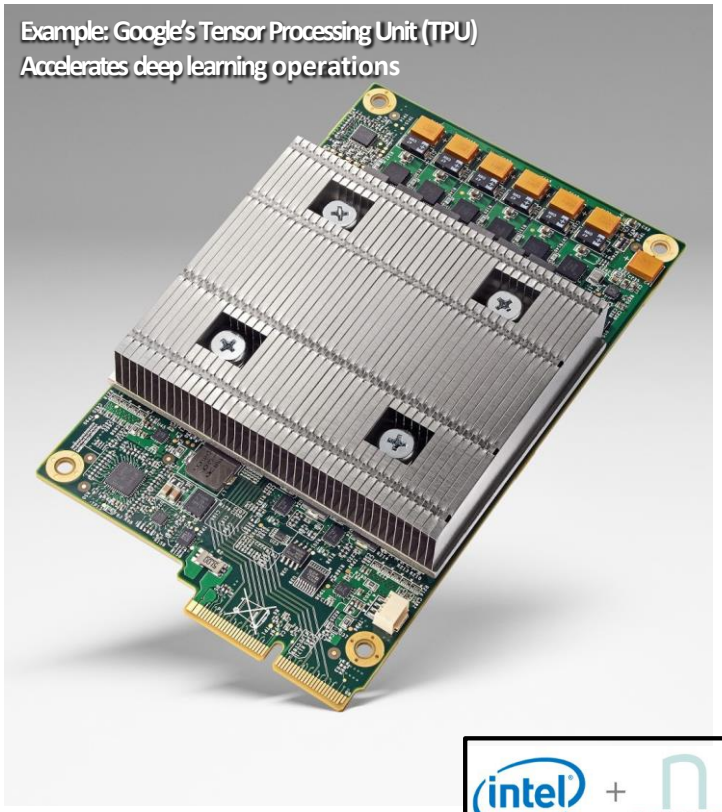
network designed for communication patterns of N-body simulations

# Specialized processors for evaluating deep networks

**Countless recent papers at top computer architecture research conferences on the topic of ASICs or accelerators for deep learning or evaluating deep networks…**

- **Cambricon: an instruction set architecture for neural networks**, Liu et al. ISCA 2016
- **EIE: Efficient Inference Engine on Compressed Deep Neural Network**, Han et al. ISCA 2016
- **Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing**, Albericio et al. ISCA 2016
- **Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators**, Reagen et al. ISCA 2016
- **vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design**, Rhu et al. MICRO 2016
- **Fused-Layer CNN Architectures**, Alwani et al. MICRO 2016
- **Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Network**, Chen et al. ISCA 2016
- **PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory**, Chi et al. ISCA 2016
- **DNNWEAVER: From High-Level Deep Network Models to FPGA Acceleration**, Sharma et al. MICRO 2016

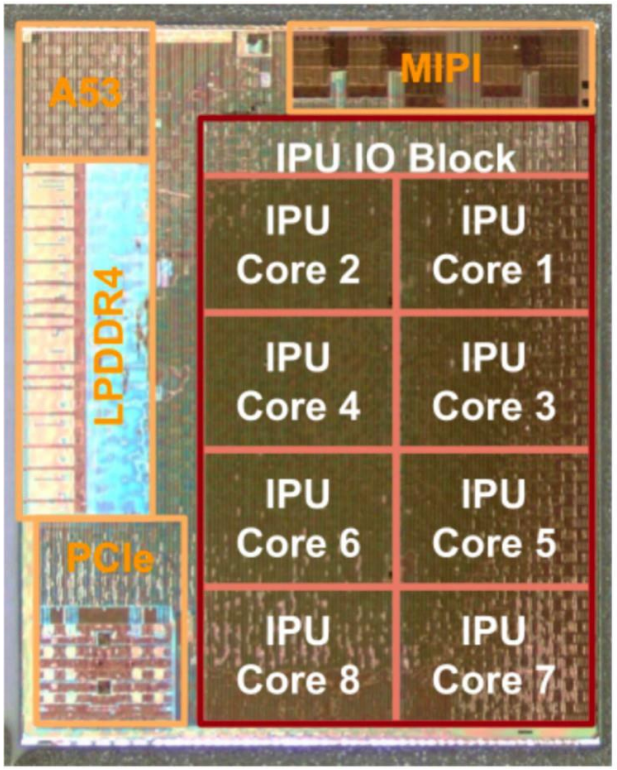



Intel Lake Crest ML accelerator (formerly Nervana)

# Example: Google's Pixel Visual Core

**Programmable "image processing unit" (IPU)**

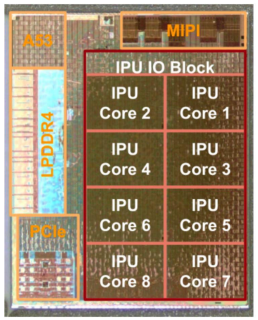
- **Each core = 16x16 grid of 16 bit multiply-add ALUs**
- **~10-20x more efficient than GPU at image processing tasks (Google's claims at HotChips '18)**



# Let's crack open a modern smartphone

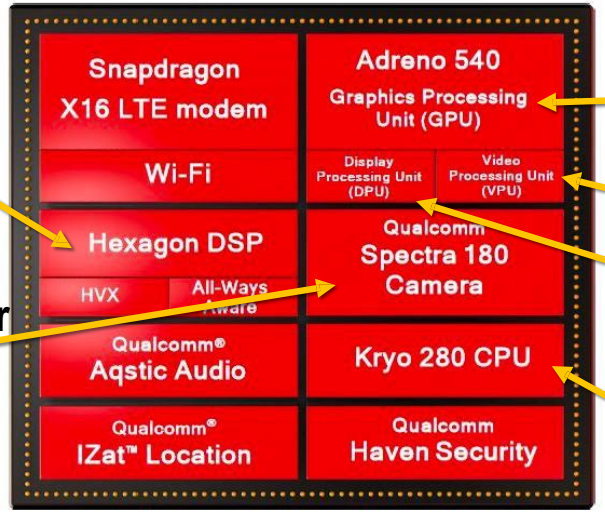
**Google Pixel 2 Phone:**
**Qualcomm Snapdragon 835 SoC + Google Visual Pixel Core**


**Visual Pixel Core**
Programmable image processor and DNN accelerator




**"Hexagon" Programmable DSP**
data-parallel multi-media processing

**Image Signal Processor**
ASIC for processing camera sensor pixels




**Multi-core GPU**
(3D graphics, OpenCL data-parallel compute)

**Video encode/decode ASIC**

**Display engine**
(compresses pixels for transfer to high-res screen)

**Multi-core ARM CPU**
4 "big cores" + 4 "little cores"

# FPGAs (Field Programmable Gate Arrays)

- Middle ground between an ASIC and a processor
- FPGA chip provides array of logic blocks, connected by interconnect
- Programmer-defined logic implemented directly by FGPA

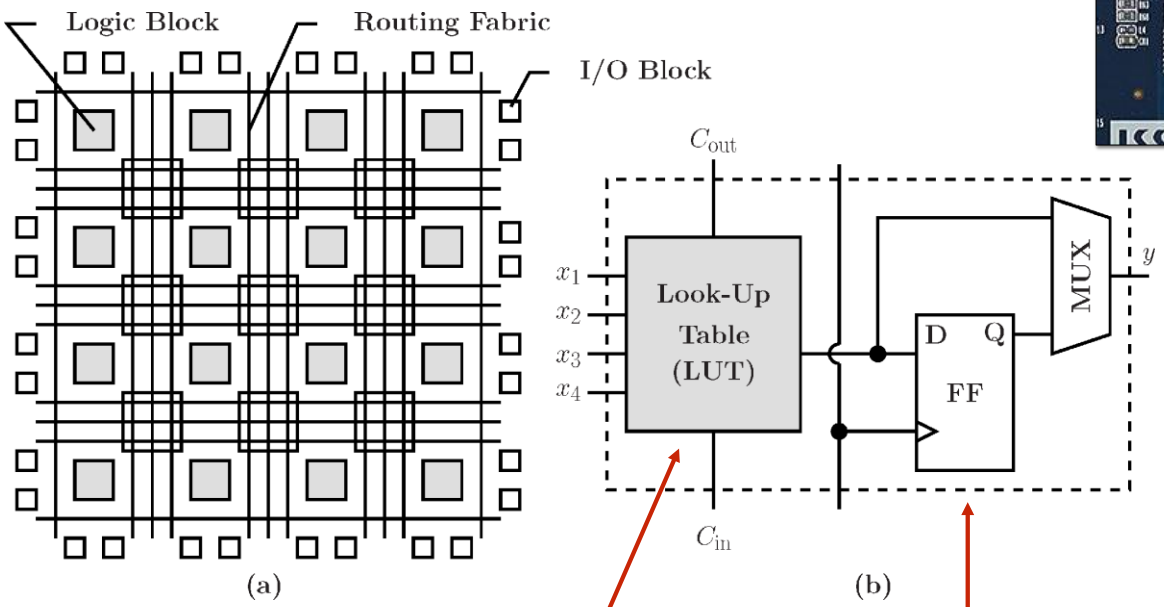
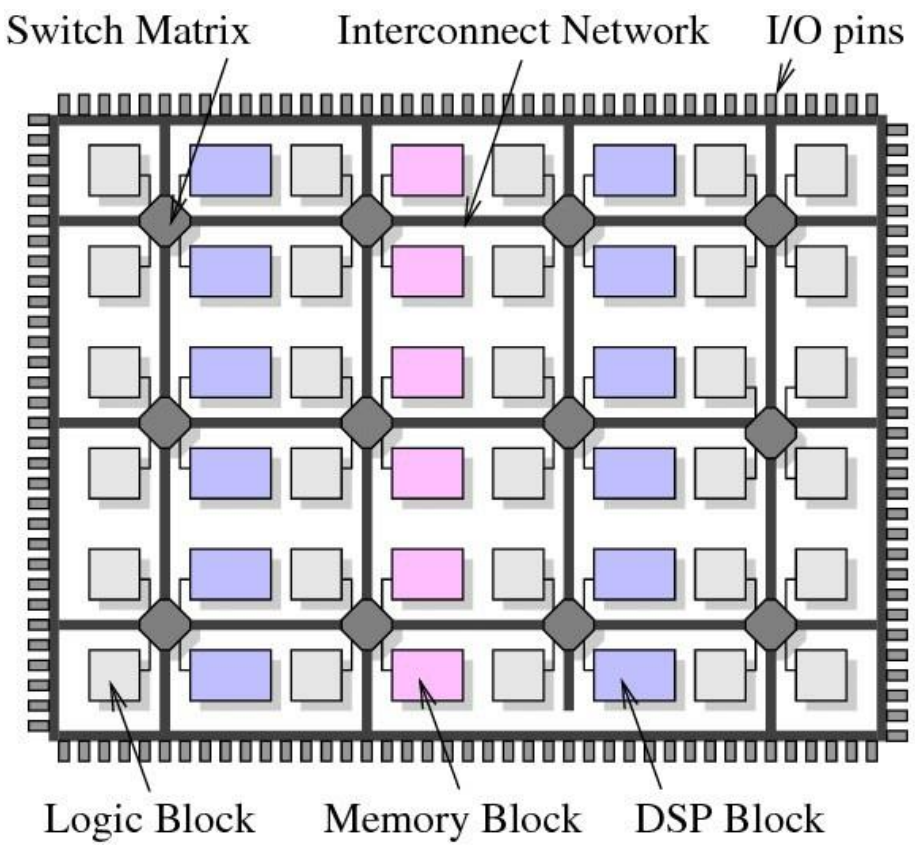




Image credit: Bai et al. 2014

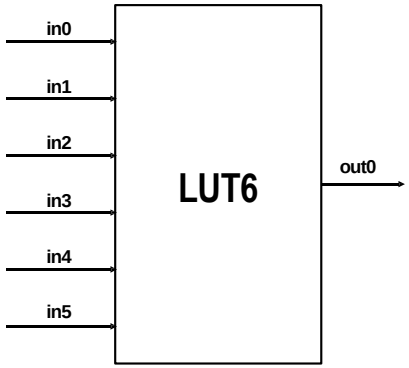
# Modern FPGAs




- A lot of area devoted to hard macros
  - Memory blocks (SRAM)
  - DSP blocks (multiplier)

# Specifying combinatorial logic as a LUT

- Example: 6-input, 1 output LUT in Xilinx Virtex-7 FPGAs
  - Think of a LUT6 as a 64 element table




Example:
6-input AND

| In | Out |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| ⋮ | ⋮ |
| 63 | 1 |

40-input AND constructed by chaining outputs of eight LUT6's (delay = 3)

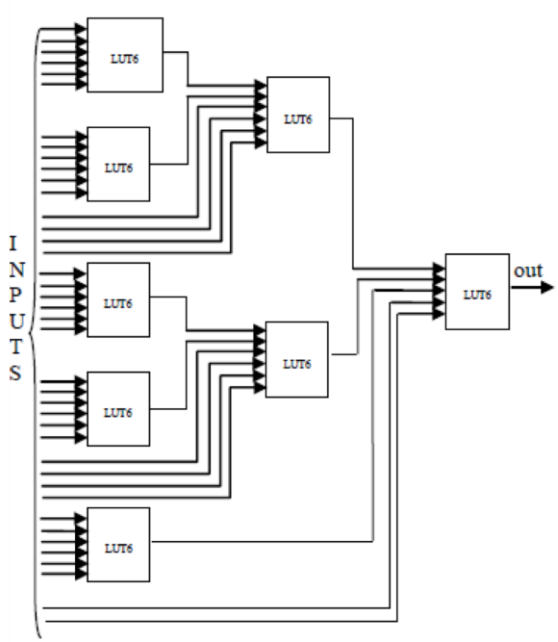



Image credit: [Zia 2013]

# Project Catapult [Putnam et al. ISCA 2014]

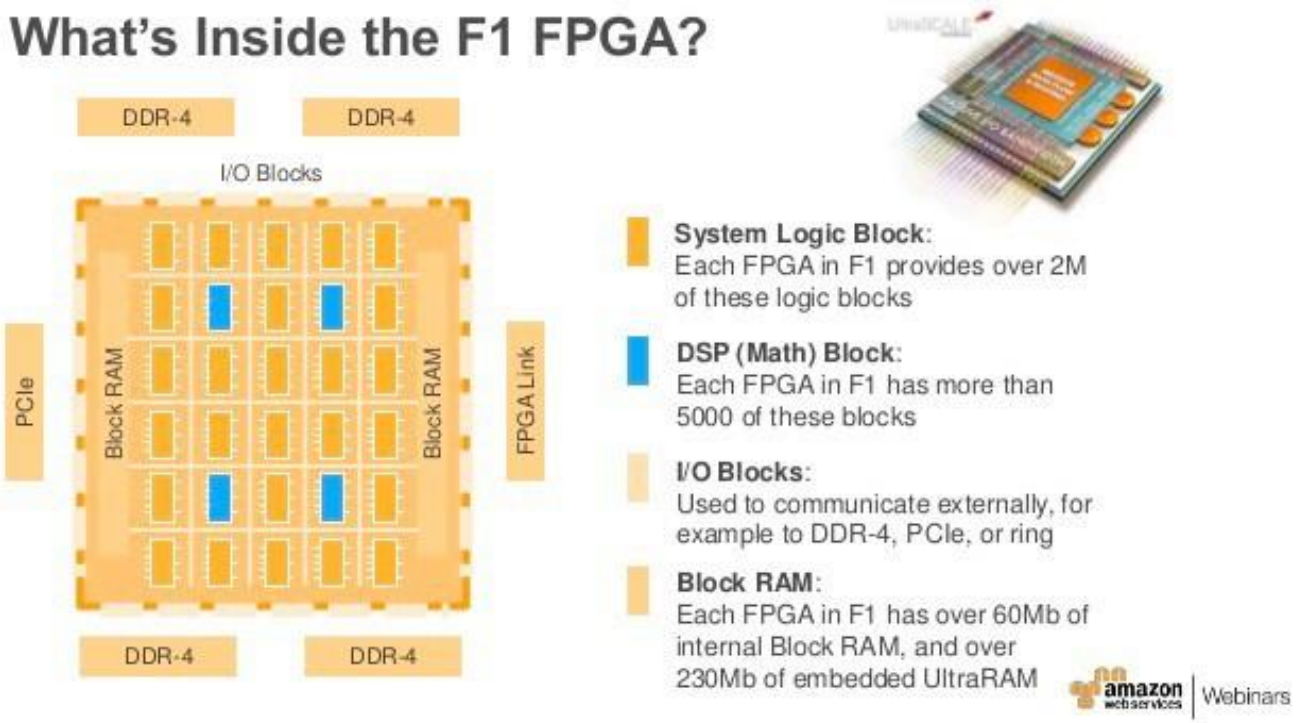
- Microsoft Research investigation of use of FPGAs to accelerate datacenter workloads
- Demonstrated offload of part of Bing search's document ranking logic

FPGA board

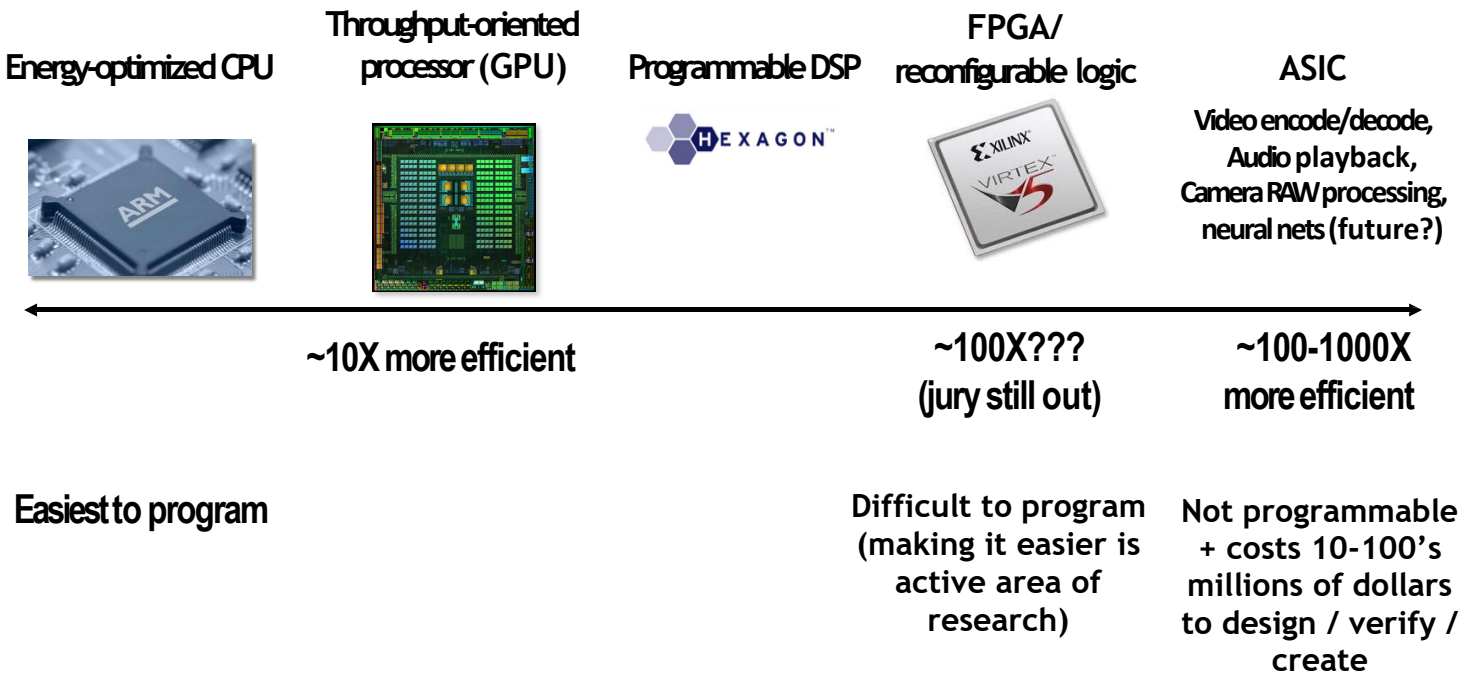

1U server (Dual socket CPU + FPGA connected via PCIe bus)

# Amazon F1

- **FPGA's are now available on Amazon cloud services**



# Summary: choosing the right tool for the job




Credit: Pat Hanrahan for this slide design

# Challenges of heterogeneous designs:

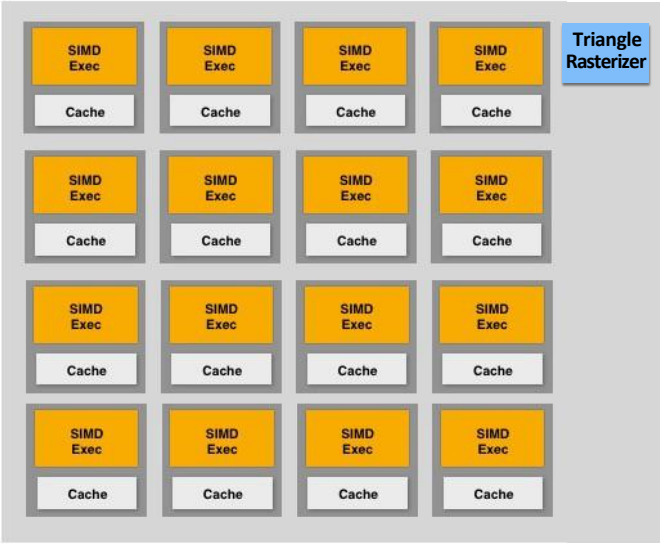
**(it's not easy to realize the potential of specialized, heterogeneous processing)**

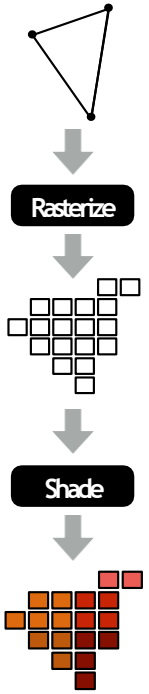
# Challenges of heterogeneity

- **Heterogeneous system: preferred processor for each task**
- **Challenge to software developer: how to map application onto a heterogeneous collection of resources?**
  - **Challenge: "Pick the right tool for the job": design algorithms that decompose into components that each map well to different processing components of the machine**
  - **The scheduling problem is more complex on a heterogeneous system**
- **Challenge for hardware designer: what is the right mixture of resources?**
  - **Too few throughput oriented resources (lower peak throughput for parallel workloads)**
  - **Too few sequential processing resources (limited by sequential part of workload)**
  - **How much chip area should be dedicated to a specific function, like video?**

# Pitfalls of heterogeneous designs

[Molnar 2010]




**Consider a two stage graphics pipeline:**
**Stage 1: rasterize triangles into pixel fragments (using ASIC)**
**Stage 2: compute color of fragments (on SIMD cores)**

**Let's say you under-provision the rasterization unit on GPU:**
**Chose to dedicate 1% of chip area used for rasterizer to achieve throughput T fragments/clock**
**But really needed throughput of 1.2T to keep the cores busy (should have used 1.2% of chip area for rasterizer)**

**Now the programmable cores only run at 80% efficiency (99% of chip is idle 20% of the time = same perf as 79% smaller chip!) So tendency is to be conservative and over-provision fixed-function components (diminishing their advantage)**

# Reducing energy consumption idea 1: use specialized processing

**(use the right processor for the job)**

# Reducing energy consumption idea 2: move less data

# Data movement has high energy cost

- Rule of thumb in mobile system design: always seek to reduce amount of data transferred from memory
  - Earlier in class we discussed minimizing communication to reduce stalls (poor performance). Now, we wish to reduce communication to reduce energy consumption
- "Ballpark" numbers [Sources: Bill Dally (NVIDIA), Tom Olson (ARM)]
  - Integer op: ~ 1 pJ *
  - Floating point op: ~20 pJ *
  - Reading 64 bits from small local SRAM (1mm away on chip): ~ 26 pJ
  - Reading 64 bits from low power mobile DRAM (LPDDR): ~1200 pJ

Suggests that recomputing values, rather than storing and reloading them, is a better answer when optimizing code for energy efficiency!

- Implications
  - Reading 10 GB/sec from memory: ~1.6 watts
  - Entire power budget for mobile GPU: ~1 watt (remember phone is also running CPU, display, radios, etc.)
  - iPhone 14 battery: ~12.6 watt-hours
  - Exploiting locality matters!!!

* Cost to just perform the logical operation, not counting overhead of instruction decode, load data from registers, etc.

# Three trends in energy-optimized computing

- **Compute less!**
  - Computing costs energy: parallel algorithms that do more work than sequential counterparts may not be desirable even if they run faster
- **Specialize compute units:**
  - Heterogeneous processors: CPU-like cores + throughput-optimized cores (GPU-like cores)
  - Fixed-function units: audio processing, "movement sensor processing" video decode/encode, image processing/computer vision?
  - Specialized instructions: expanding set of AVX vector instructions, new instructions for accelerating AES encryption (AES-NI)
  - Programmable soft logic: FPGAs
- **Reduce bandwidth requirements**
  - Exploit locality (restructure algorithms to reuse on-chip data as much as possible)
  - Aggressive use of compression: perform extra computation to compress application data before transferring to memory (likely to see fixed-function HW to reduce overhead of general data compression/decompression)

# Summary: heterogeneous processing for efficiency

- **Heterogeneous parallel processing: use a mixture of computing resources that fit mixture of needs of target applications**
  - Latency-optimized sequential cores, throughput-optimized parallel cores, domain-specialized fixed-function processors
  - Examples exist throughout modern computing: mobile processors, servers, supercomputers

- **Traditional rule of thumb in "good system design" is to design simple, general-purpose components**
  - This is not the case in emerging systems (optimized for perf/watt)
  - Today: want collection of components that meet perf requirement AND minimize energy use

- **Challenge of using these resources effectively is pushed up to the programmer**
  - Current research challenge: how to write efficient, portable programs for emerging heterogeneous architectures?