Lecture 15:

Heterogeneous Parallelism and Hardware Specialization

Parallel Computing
Stanford CS149, Fall 2019

I want to begin this lecture by reminding you…

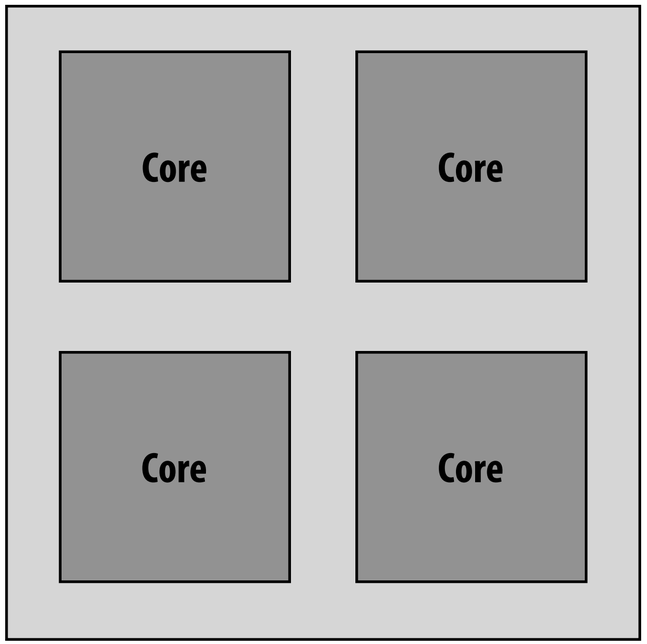
In assignment 1 we observed that a well-optimized parallel implementation of a compute-bound application is about 40 times faster on my quad-core laptop than the output of single-threaded C code compiled with gcc -O3.

(In other words, a lot of software makes inefficient use of modern CPUs.)

Today we're going to talk about how inefficient the CPU in that laptop is, even if you are using it as efficiently as possible.


You need to buy a
new computer…

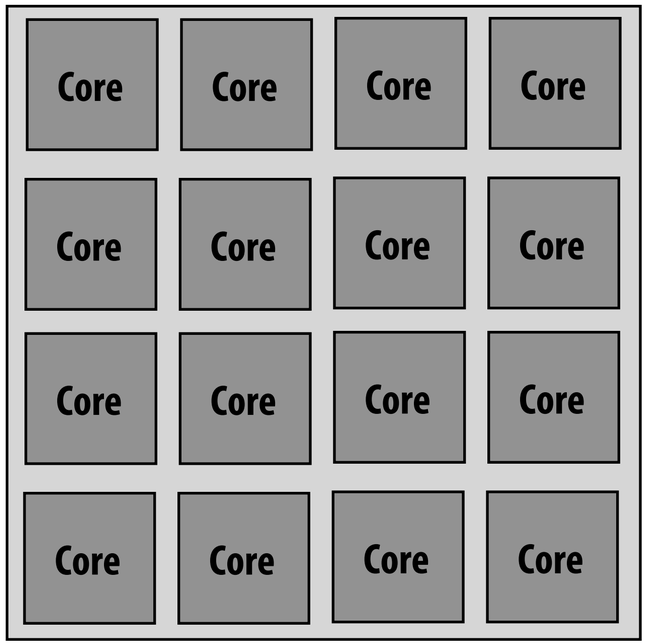
You need to buy a computer system



Processor A

4 cores

Each core has sequential performance P



Processor B

16 cores

Each core has sequential performance P/2

All other components of the system are equal.

Which do you pick?

Recall Amdahl's law

$$\text{speedup}(f, n) = \frac{1}{(1 - f) + \frac{f}{n}}$$

f = ***fraction of program that is parallelizable***

n = ***parallel processors***

Assumptions:

Parallelizable work distributes perfectly onto n processors of equal capability

Rewrite Amdahl's law in terms of resource limits

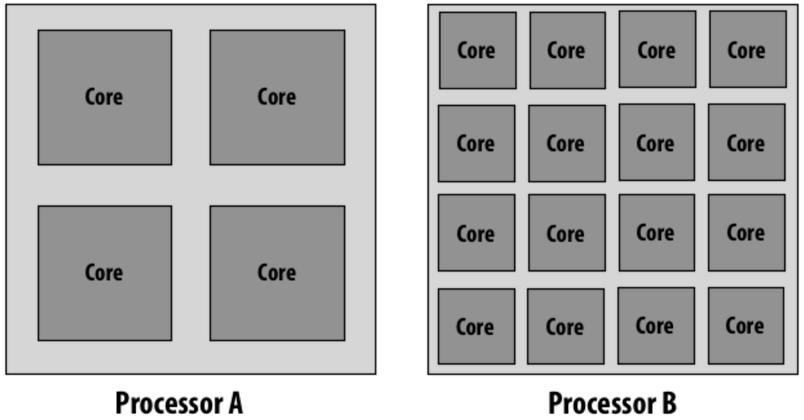
$$\text{speedup}(f, n, r) = \frac{1}{\frac{1-f}{\text{perf}(r)} + \frac{f}{\text{perf}(r) \cdot \frac{n}{r}}}$$

Speedup relative to processor with 1 unit of resources, n=1
Assume perf(1) = 1

f = ***fraction of program that is parallelizable***
n = ***total processing resources (e.g., transistors on a chip)***
r = ***resources dedicated to each processing core,***
(each of the n/r cores has sequential performance $perf(r)$

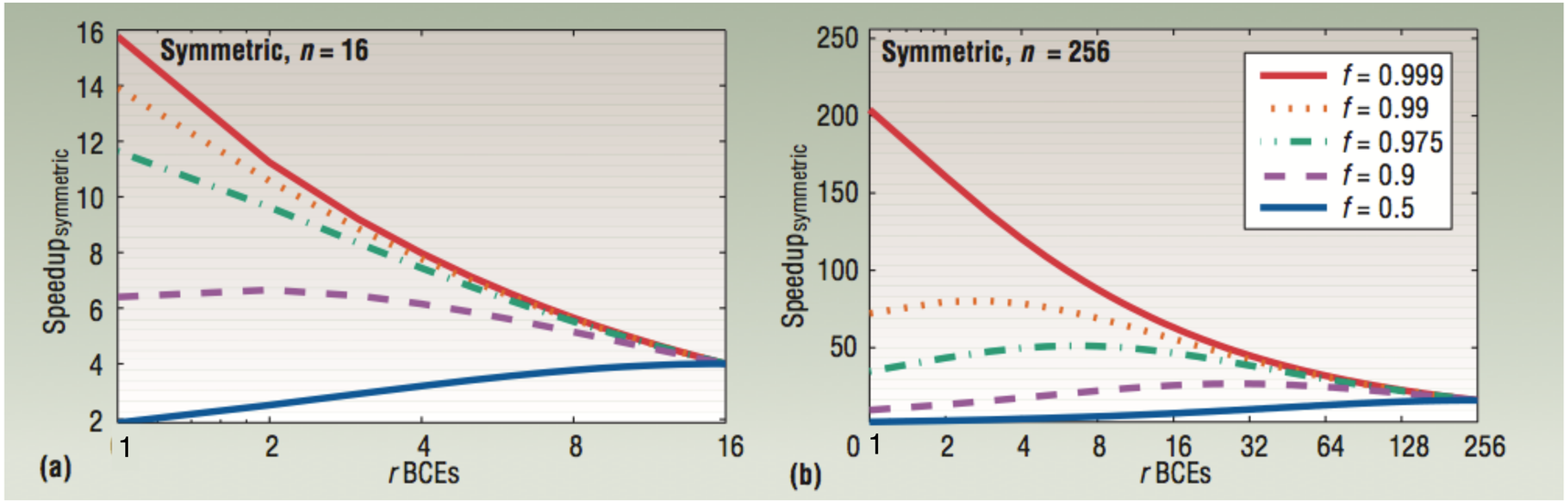
More general form of Amdahl's Law in terms of f, n, r

Two examples where n=16
r_A = ***4***
r_B = ***1***



[Hill and Marty 08]

Speedup (relative to n=1)



Up to 16 cores (n=16)

Up to 256 cores (n=256)

X-axis = r (chip with many small cores to left, fewer "fatter" cores to right)

Each line corresponds to a different workload

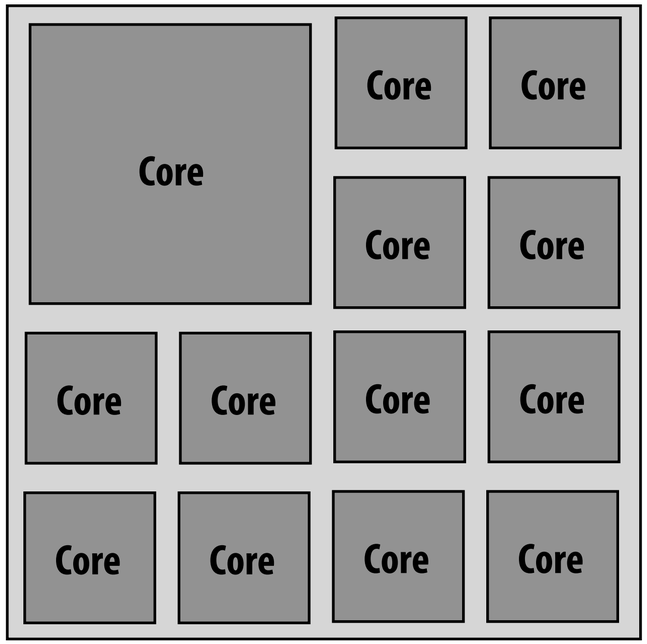
Each graph plots performance as resource allocation changes, but total chip resources resources are kept the same (constant n per graph)

$perf(r)$ modeled as $\sqrt{r}$

[Figure credit: Hill and Marty 08]

Asymmetric set of processing cores

Example: *n*=16
One core: $r = 4$
Other 12 cores: $r = 1$



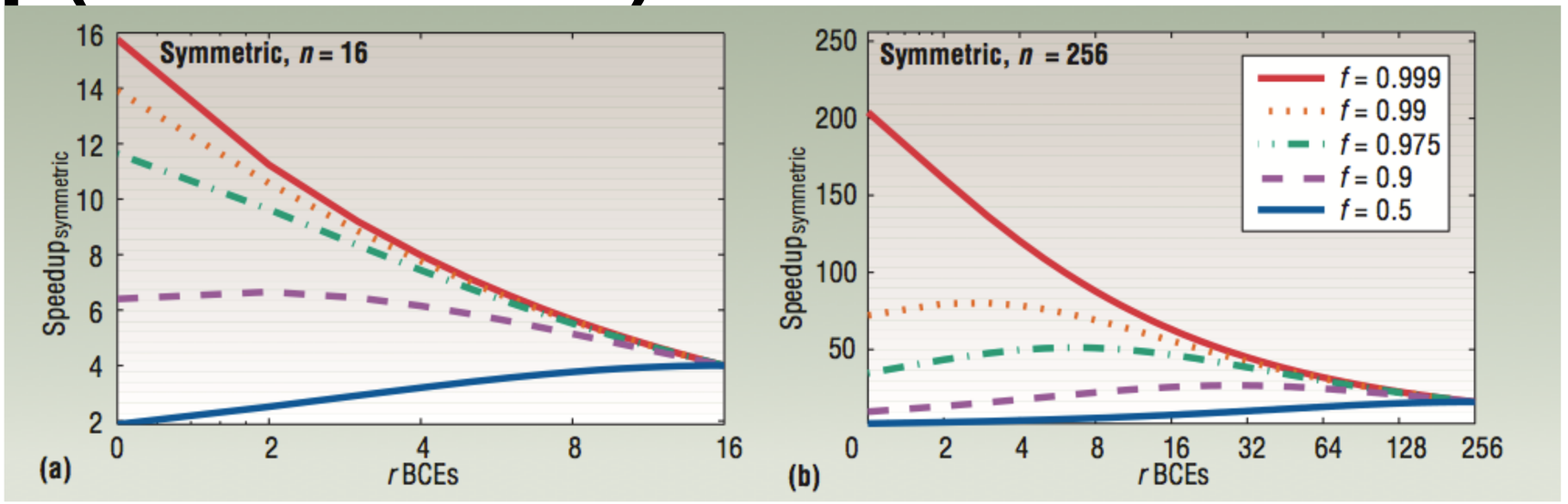
$$\text{speedup}(f, n, r) = \frac{1}{\frac{1-f}{\text{perf}(r)} + \frac{f}{\text{perf}(r)+(n-r)}}$$

(of heterogeneous processor with *n* recourses, relative to uniprocessor with one unit worth of resources, n=1)

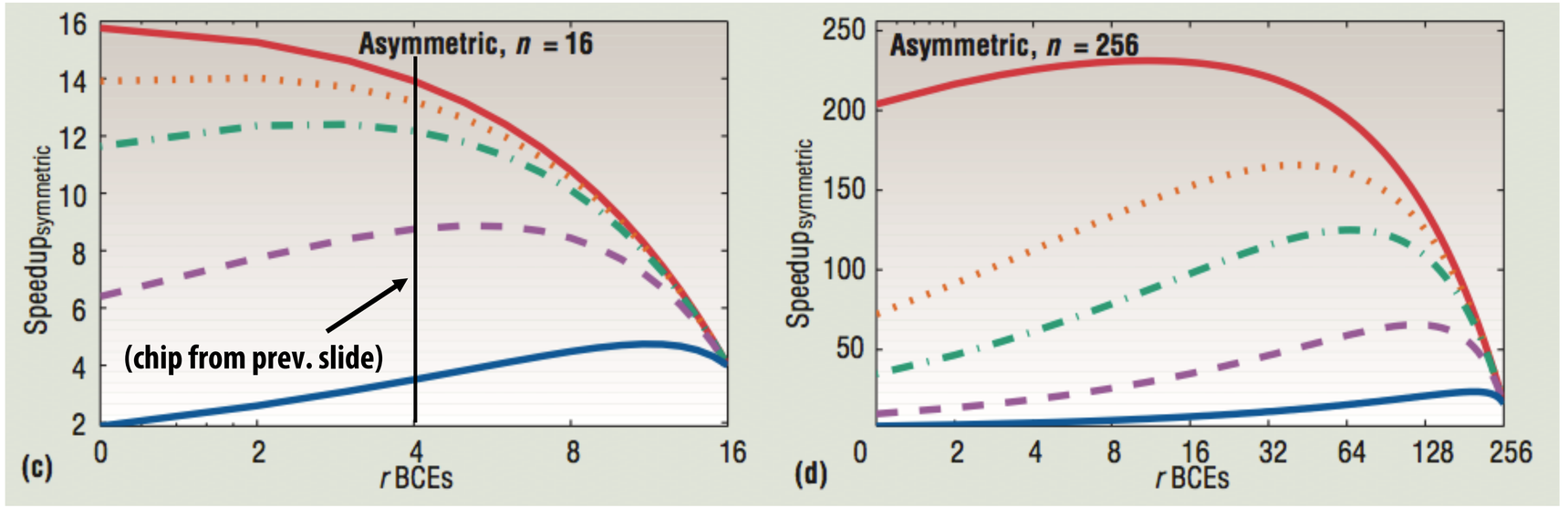
one perf(r) processor + (n-r) perf(1)=1 processors

Speedup (relative to n=1)

[Source: Hill and Marty 08]



X-axis for symmetric architectures gives r for all cores (many small cores to left, few "fat" cores to right)



X-axis for asymmetric architectures gives r for the single "fat" core (assume rest of cores are $r = 1$)

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

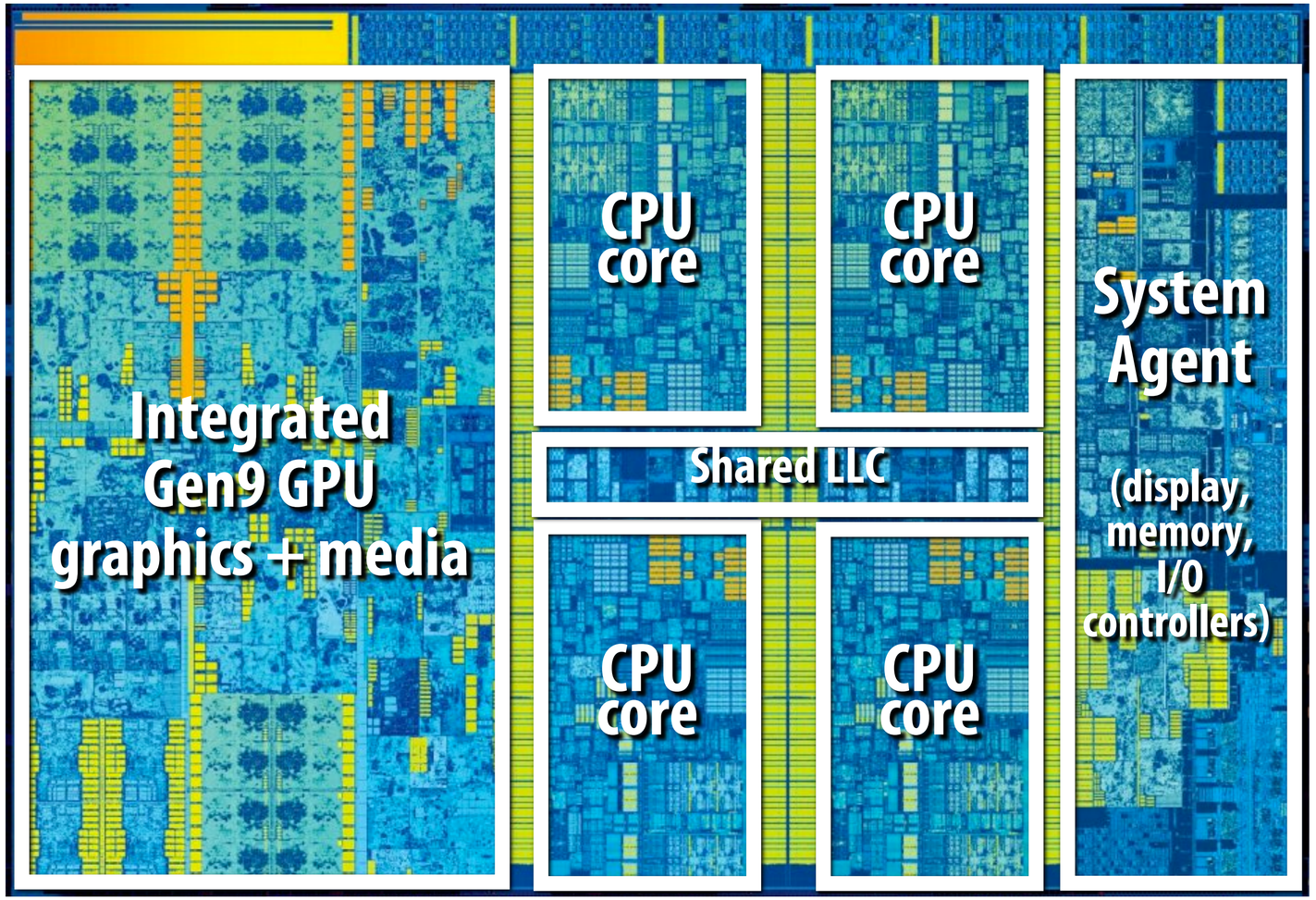
And components with unpredictable access, but those accesses might cache well.

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)

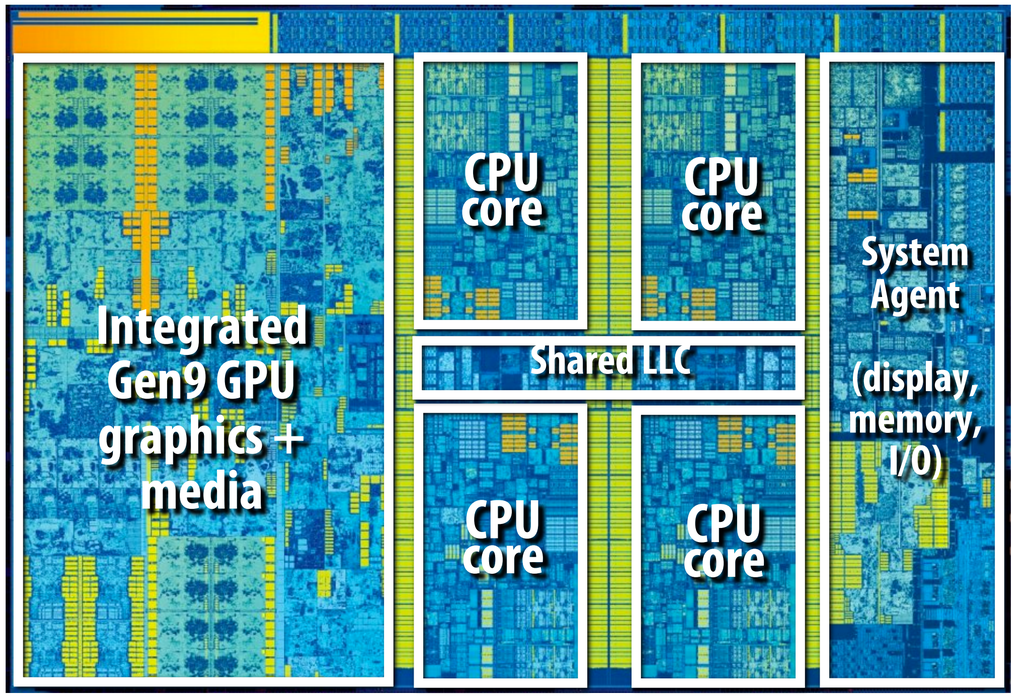
(6th Generation Core i7 architecture)



4 CPU cores + graphics cores + media accelerators

Example: Intel “Skylake" (2015)

(6th Generation Core i7 architecture)

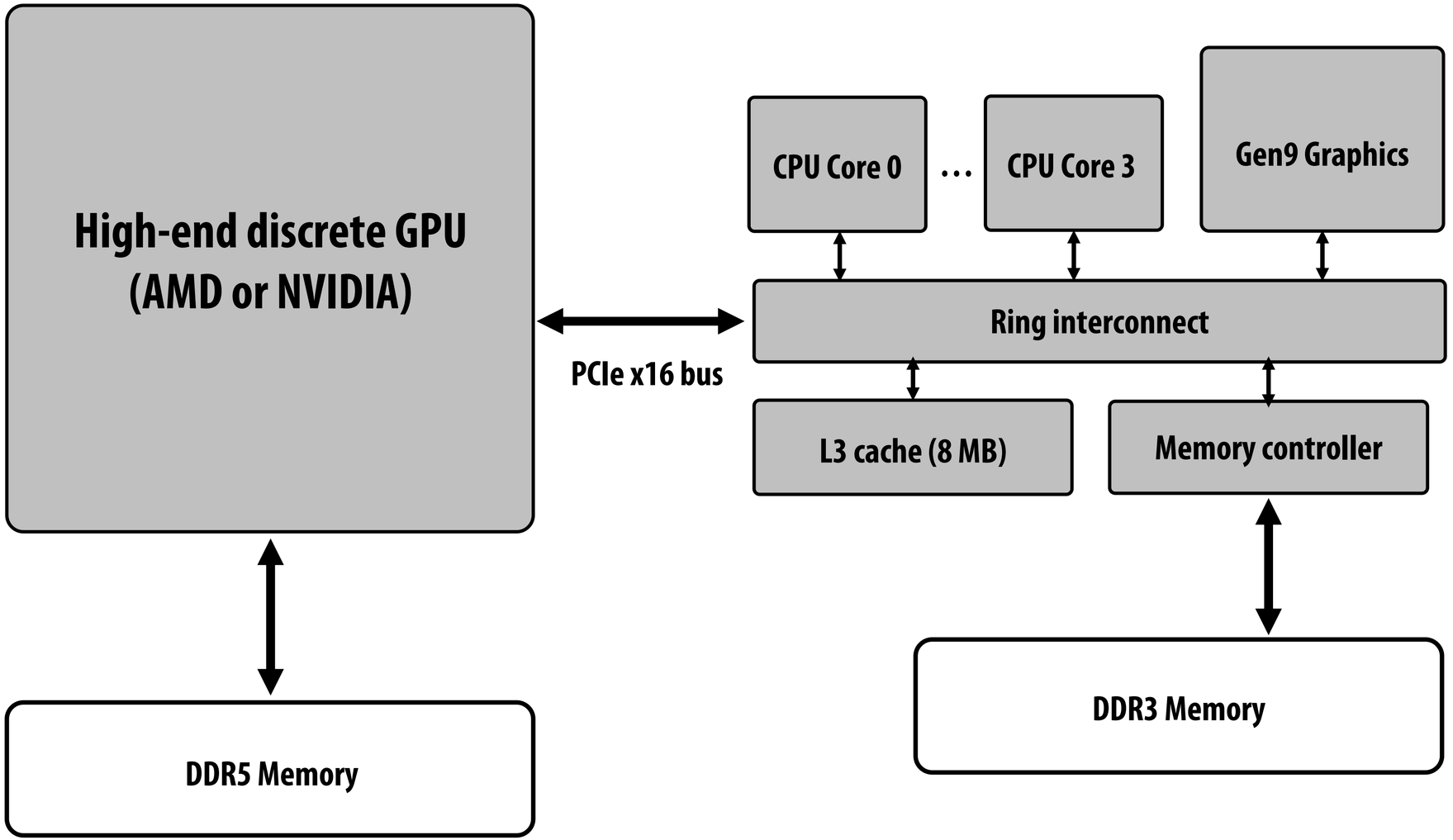


- CPU cores and graphics cores share same memory system
- Also share LLC (L3 cache)
  - Enables, low-latency, high-bandwidth communication between CPU and integrated GPU
- Graphics cores are cache coherent with CPU cores

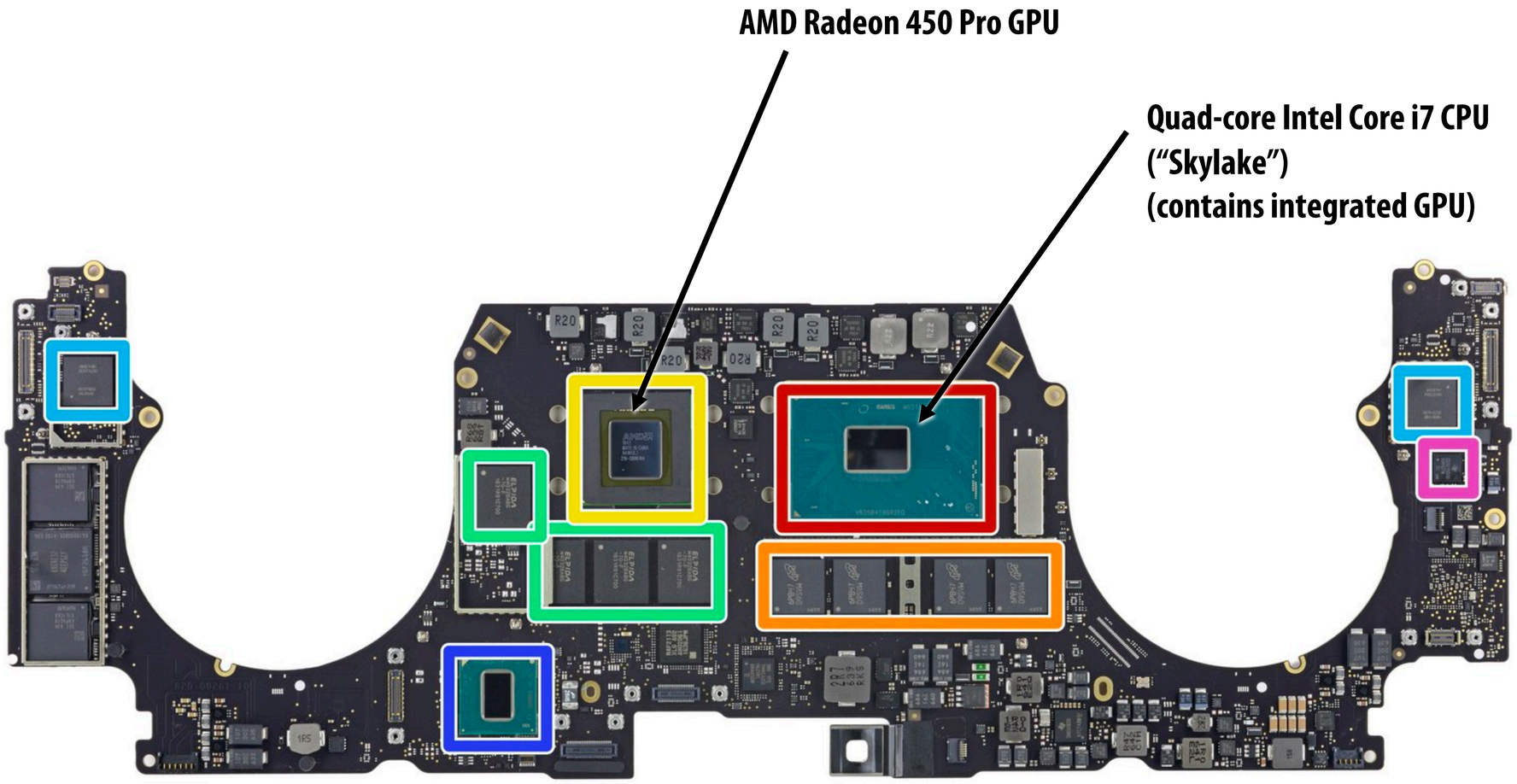
More heterogeneity: add discrete GPU

Keep discrete (power hungry) GPU unless needed for graphics-intensive applications

Use integrated, low power graphics for basic graphics/window manager/UI

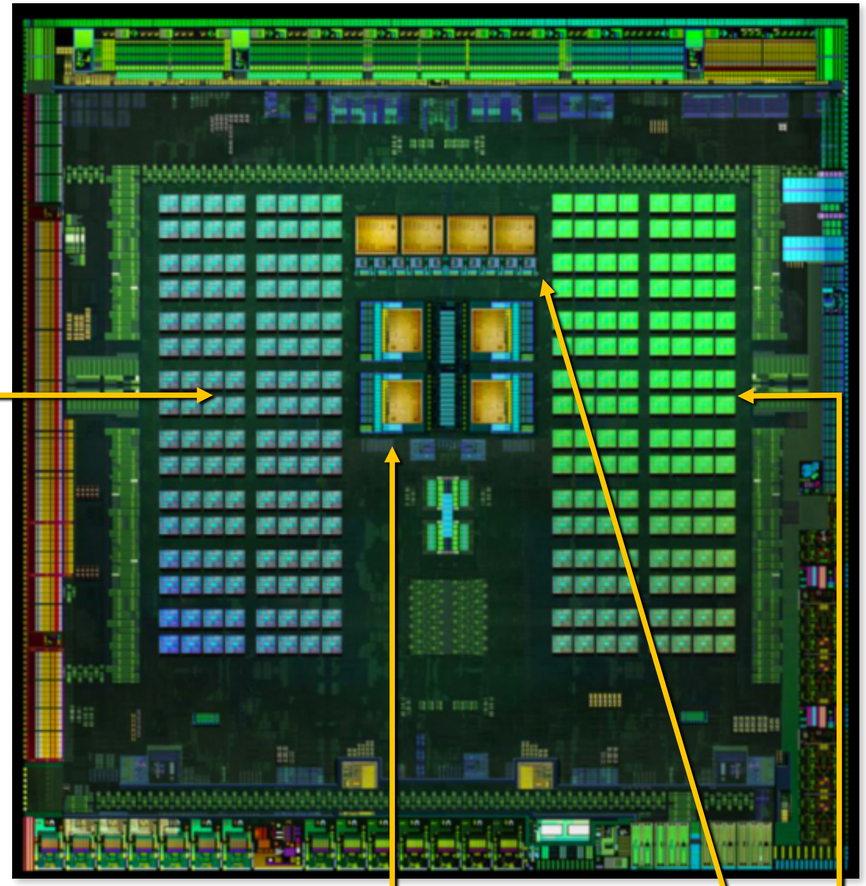


15in Macbook Pro /w Touch Bar (2016) (two GPUs)



From ifixit.com teardown

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)

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

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

Supercomputers use heterogeneous processing

Los Alamos National Laboratory: "Roadrunner"

Fastest US supercomputer in 2008, first to break Petaflop barrier: 1.7 PFLOPS

Unique at the time due to use of two types of processing elements

(IBM's Cell processor served as "accelerator" to achieve desired compute density)

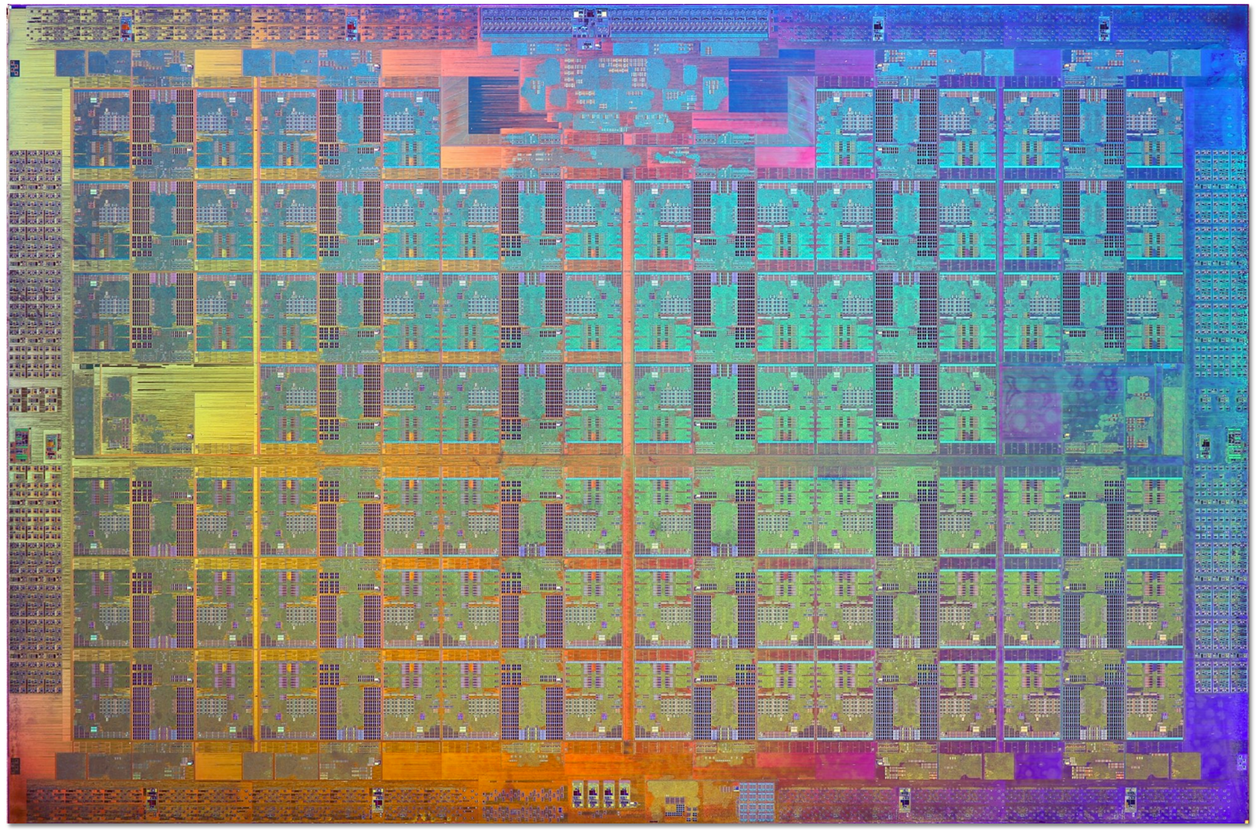
- **6,480 AMD Opteron dual-core CPUs (12,960 cores)**
- **12,970 IBM Cell Processors (1 CPU + 8 accelerator cores per Cell = 116,640 cores)**
- **2.4 MWatt (about 2,400 average US homes)**

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

Intel Xeon Phi (Knights Landing)

- 72 "simple" x86 cores (1.1 Ghz, derived from Intel Atom)
- 16-wide vector instructions (AVX-512), four threads per core
- Targeted as an accelerator for supercomputing applications



Heterogeneous architectures for supercomputing

Source: Top500.org Fall 2018 rankings

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,397,824	143,500.0	200,794.9	9,783
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	27,154.3	2,384
6	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,072	20,158.7	41,461.2	7,578
7	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	391,680	19,880.0	32,576.6	1,649
8	**SuperMUC-NG** - ThinkSystem SD530, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path , Lenovo Leibniz Rechenzentrum Germany	305,856	19,476.6	26,873.9	

GPU

GPU

GPU

Xeon Phi

GPU

201 Petaflops (peak),
143 Petaflops (effective)
9.7 MWatt
(14.6 GFLOPS/W)

Green500: most energy efficient supercomputers

Efficiency metric: effective GFLOPS per Watt

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
1	375	**Shoubu system B** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. Advanced Center for Computing and Communication, RIKEN Japan	953,280	1,063.3	60	17.604
2	374	**DGX SaturnV Volta** - NVIDIA DGX-1 Volta36, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, NVIDIA Tesla V100 , Nvidia NVIDIA Corporation United States	22,440	1,070.0	97	15.113
3	1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,397,824	143,500.0	9,783	14.668
4	7	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	391,680	19,880.0	1,649	14.423
5	22	**TSUBAME3.0** - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz, Intel Omni-Path, NVIDIA Tesla P100 SXM2 , HPE GSIC Center, Tokyo Institute of Technology Japan	135,828	8,125.0	792	13.704
6	2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	7,438	12.723

Source: Green500 Fall 2018 rankings

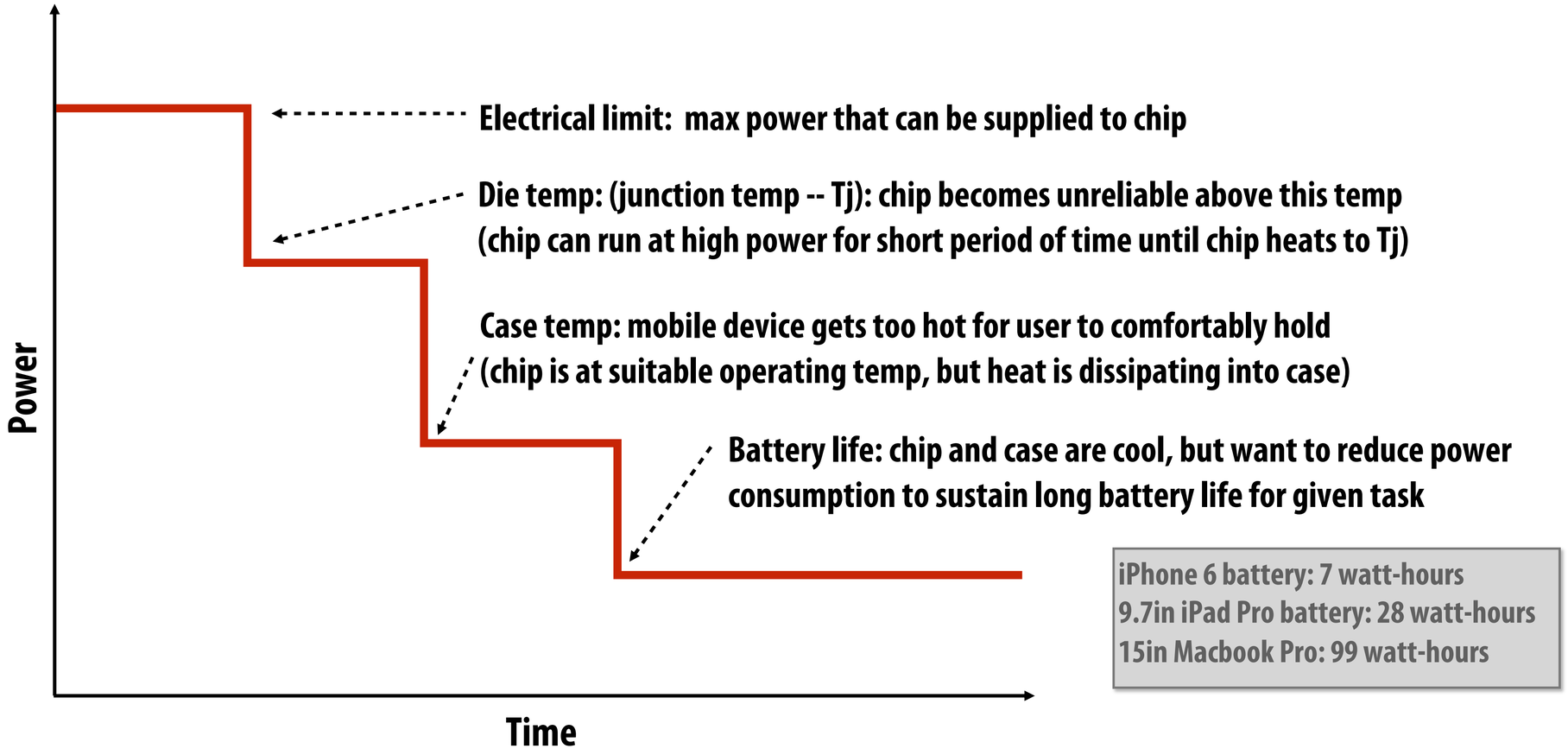
Energy-constrained computing

- Supercomputers are energy constrained
  - Due to shear scale of machine
  - Overall cost to operate (power for machine and for cooling)
- Datacenters are energy constrained
  - Reduce cost of cooling
  - Reduce physical space requirements
- Mobile devices are energy constrained
  - Limited battery life
  - Heat dissipation

Energy-constrained computing

Limits on chip power consumption

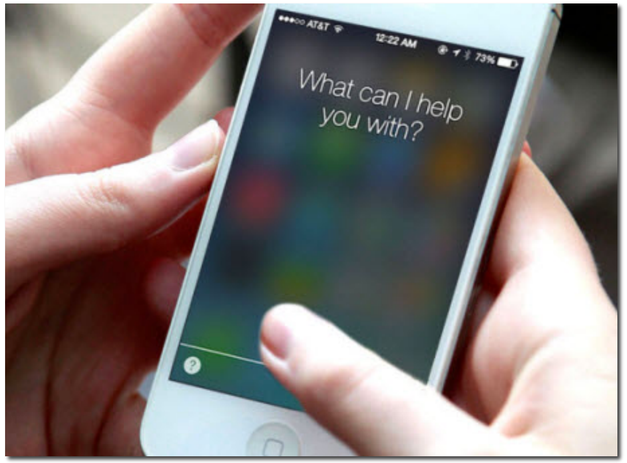
- General mobile processing rule: the longer a task runs the less power it can use
  - Processor's power consumption is limited by heat generated (efficiency is required for more than just maximizing battery life)



Slide credit: adopted from original slide from M. Shebanow: HPG 2013 keynote

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")

Modern computing: efficiency often matters more than in the past, not less

Fourth, there's battery life.

To achieve long battery life when playing video, mobile devices must decode the video in hardware; decoding it in software uses too much power. Many of the chips used in modern mobile devices contain a decoder called H.264 – an industry standard that is used in every Blu-ray DVD player and has been adopted by Apple, Google (YouTube), Vimeo, Netflix and many other companies.

Although Flash has recently added support for H.264, the video on almost all Flash websites currently requires an older generation decoder that is not implemented in mobile chips and must be run in software. The difference is striking: on an iPhone, for example, H.264 videos play for up to 10 hours, while videos decoded in software play for less than 5 hours before the battery is fully drained.

When websites re-encode their videos using H.264, they can offer them without using Flash at all. They play perfectly in browsers like Apple's Safari and Google's Chrome without any plugins whatsoever, and look great on iPhones, iPods and iPads.

Steve Jobs' "Thoughts on Flash", 2010

http://www.apple.com/hotnews/thoughts-on-flash/

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

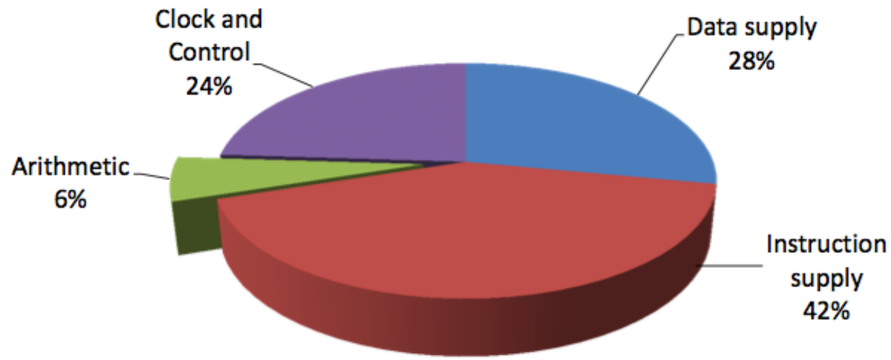
[Source: Chung et al. 2010 , Dally 08]

Why is a “general-purpose processor” so inefficient?

Wait… this entire class we’ve been talking about making efficient use out of multi-core CPUs and GPUs…
and now you’re telling me these platforms are “inefficient”?

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



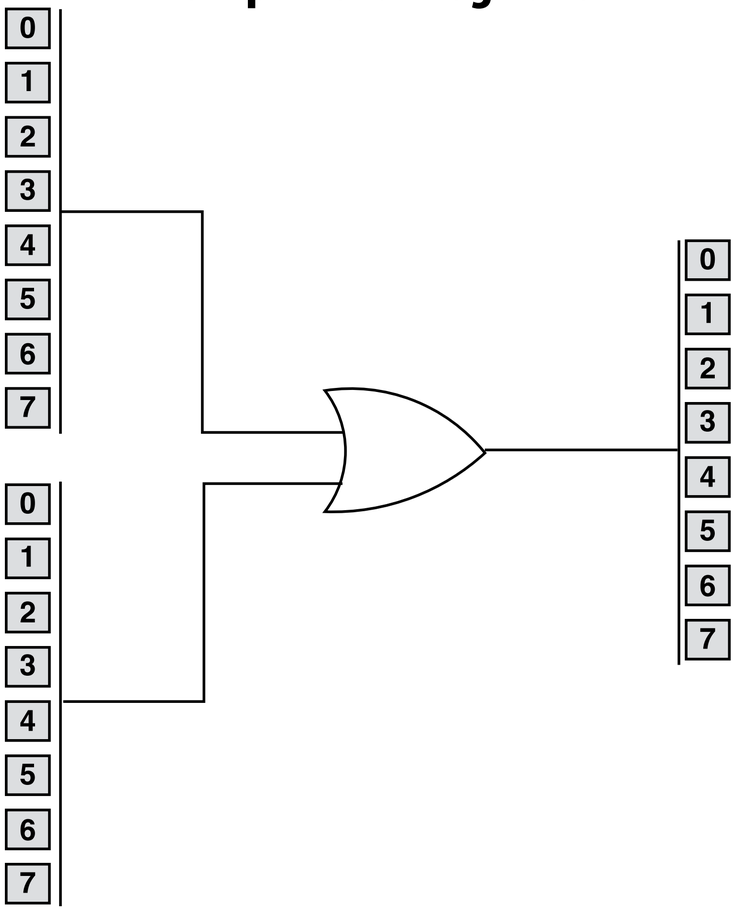
Efficient Embedded Computing [Dally et al. 08]

[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

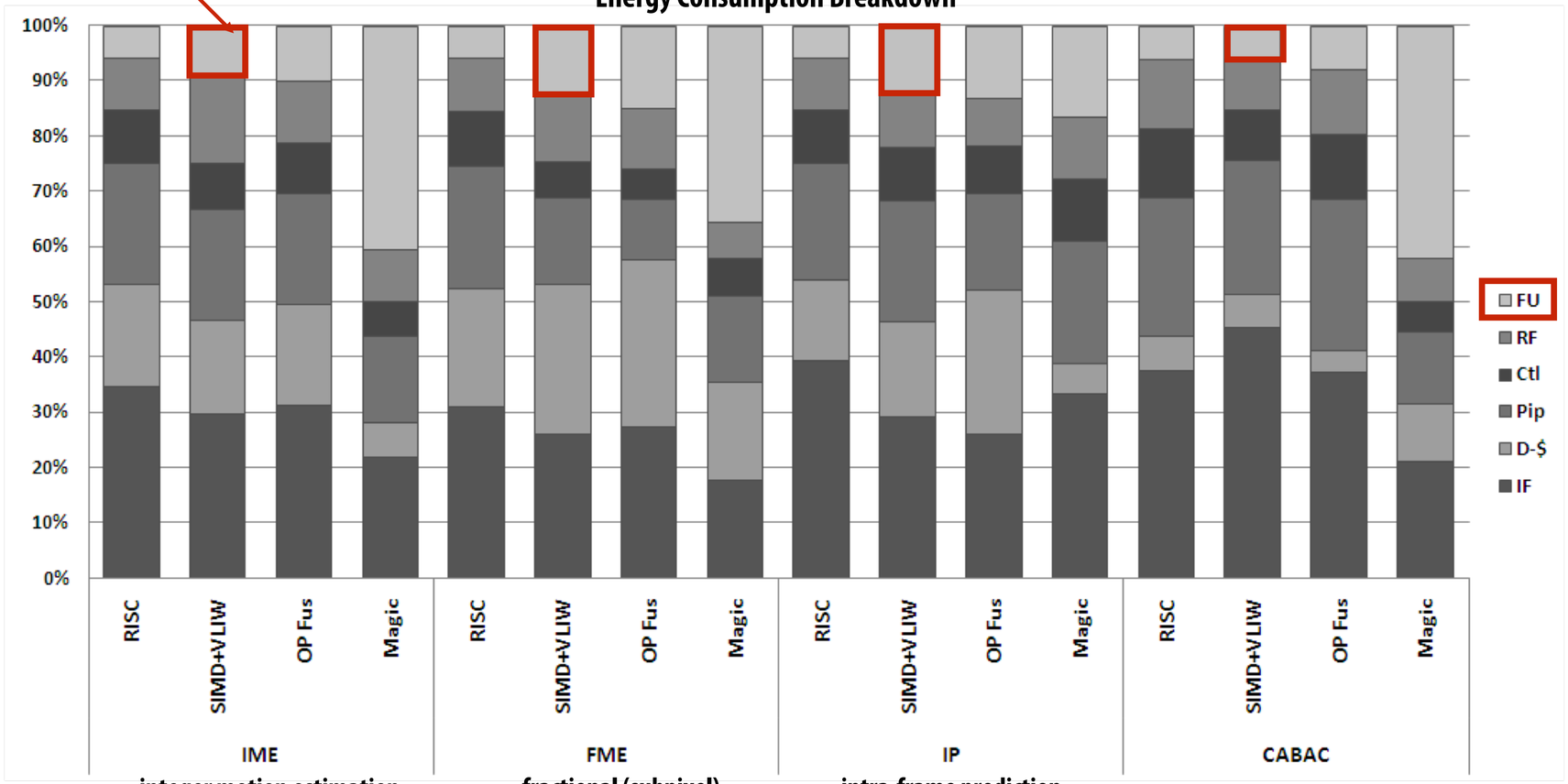
Example: 8-bit logical OR



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

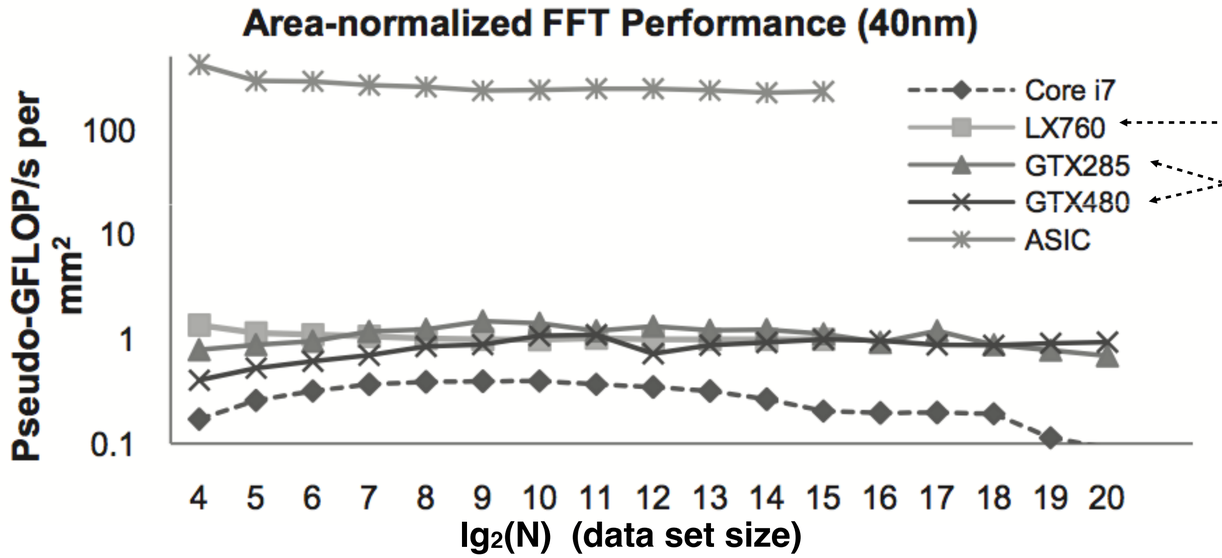
Even after encoding implemented with SIMD instruction

[Hameed et al. ISCA 2010]



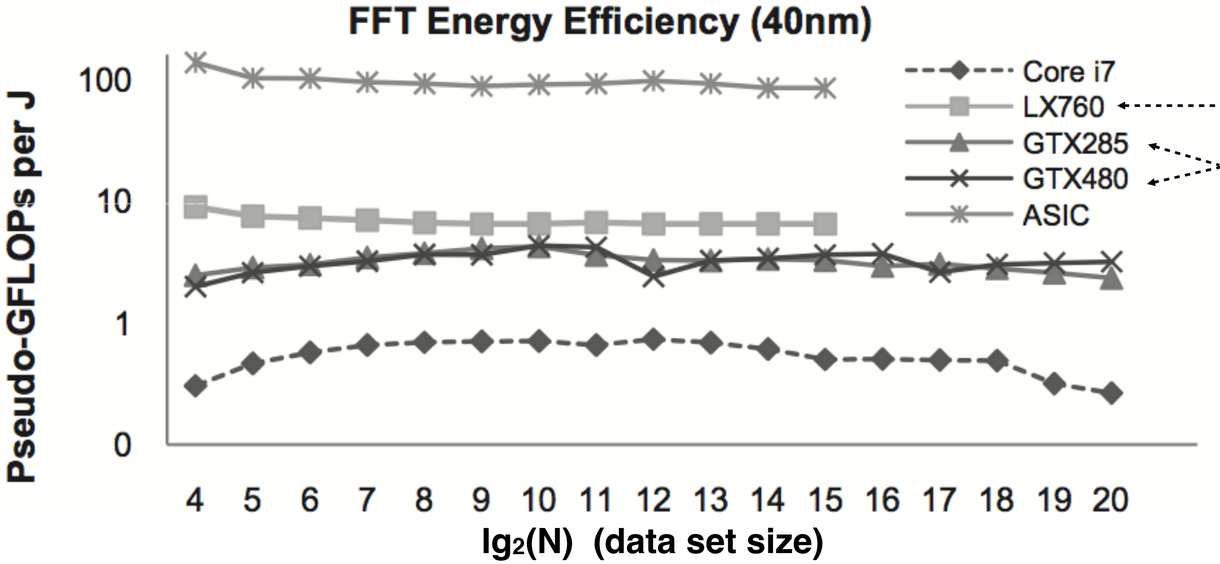
FU = functional units	Pip = pipeline registers (interstage)
RF = register fetch	D-$ = data cache
Ctrl = misc pipeline control	IF = instruction fetch + instruction cache

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



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

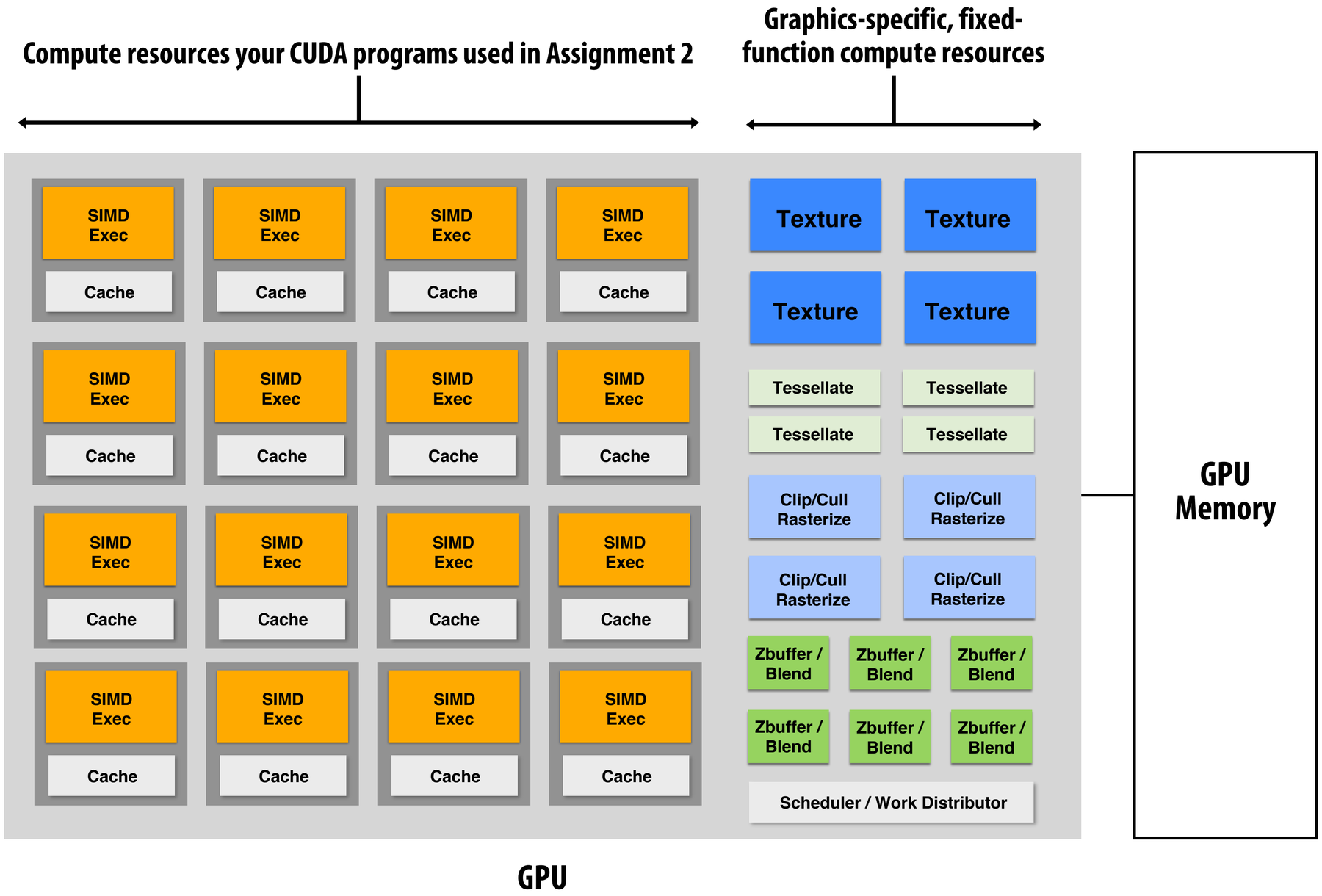
GPU cores: ~ 5-7 times more area efficient than CPU cores.



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

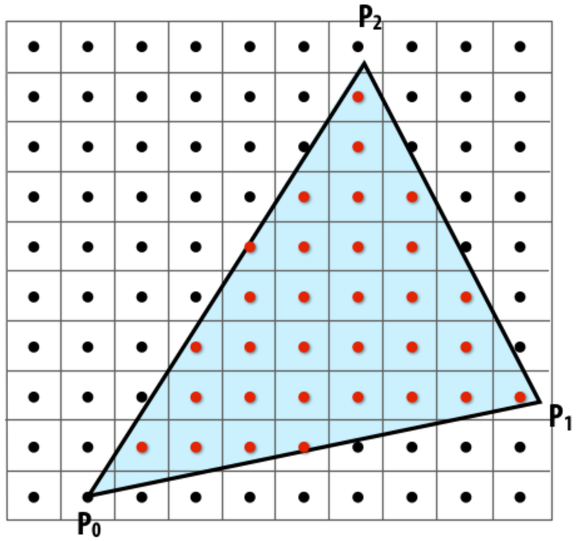
[Chung et al. MICRO 2010]

GPU's are themselves heterogeneous multi-core processors

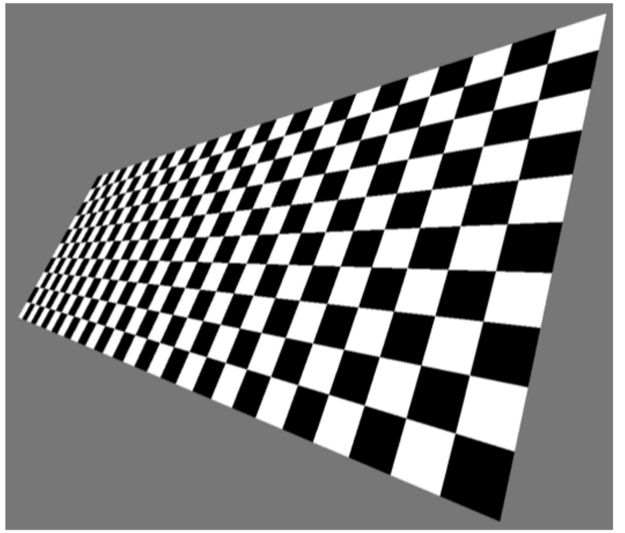


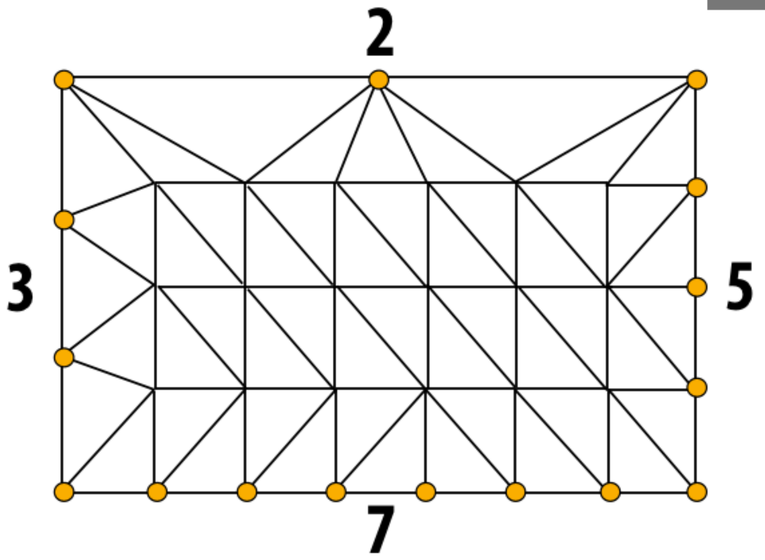
Example graphics tasks performed in fixed-function HW

Rasterization:
Determining what pixels a triangle overlaps



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





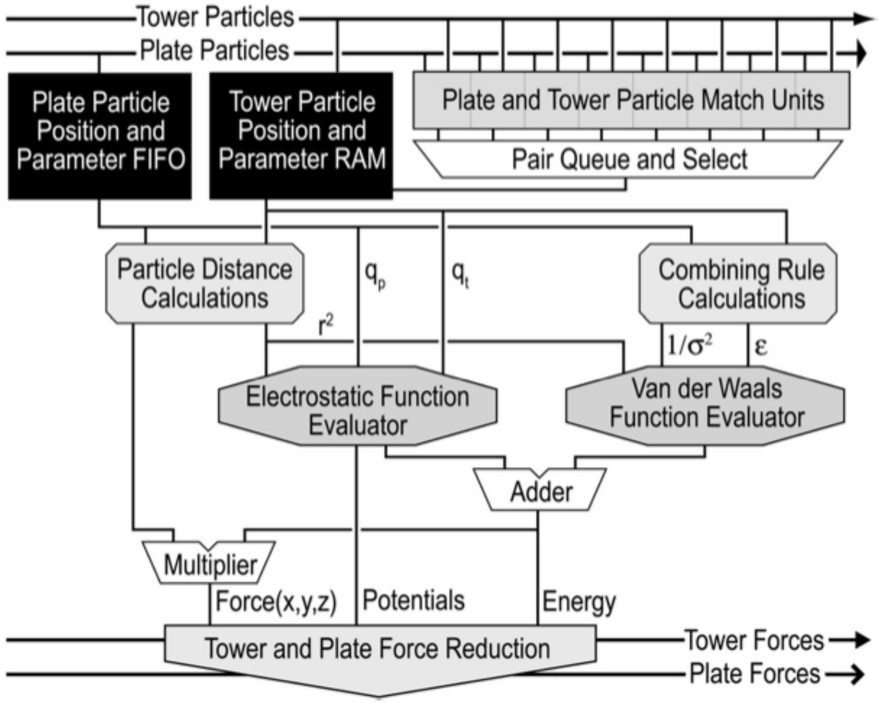
Geometric tessellation:
computing fine-scale geometry from coarse geometry

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

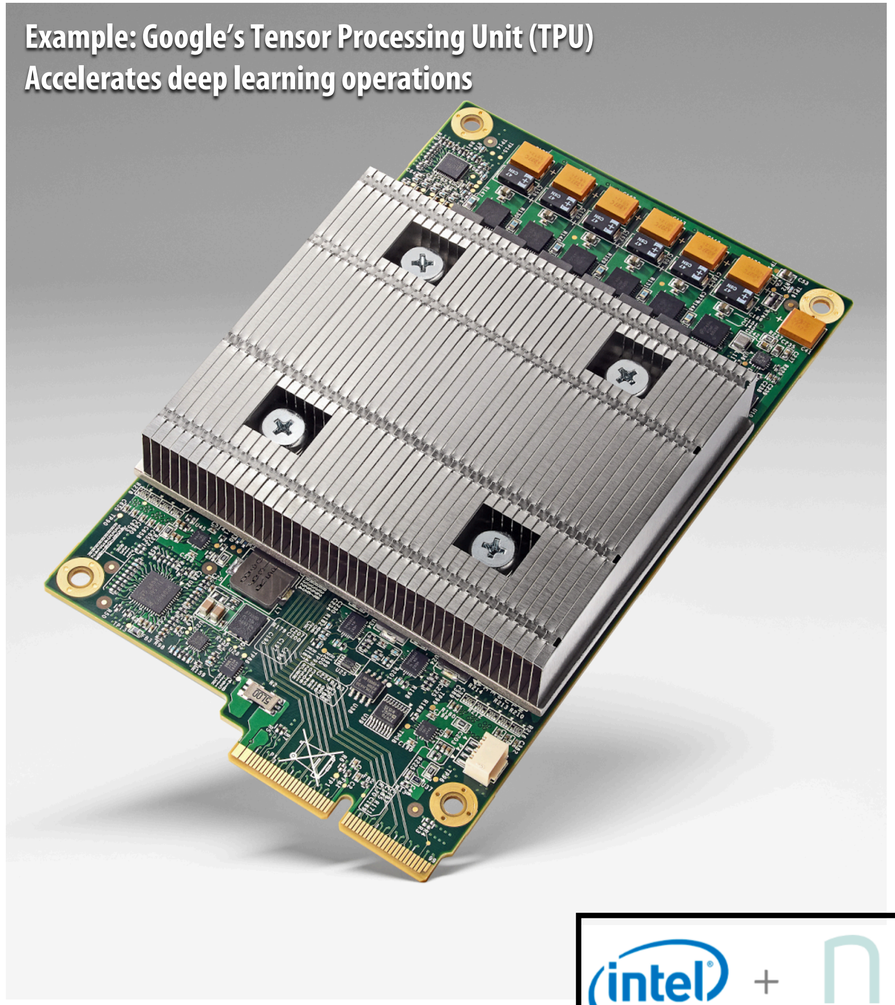
network designed for communication patterns of N-body simulations



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)

Digital signal processors (DSPs)

Programmable processors, but simpler instruction stream control paths

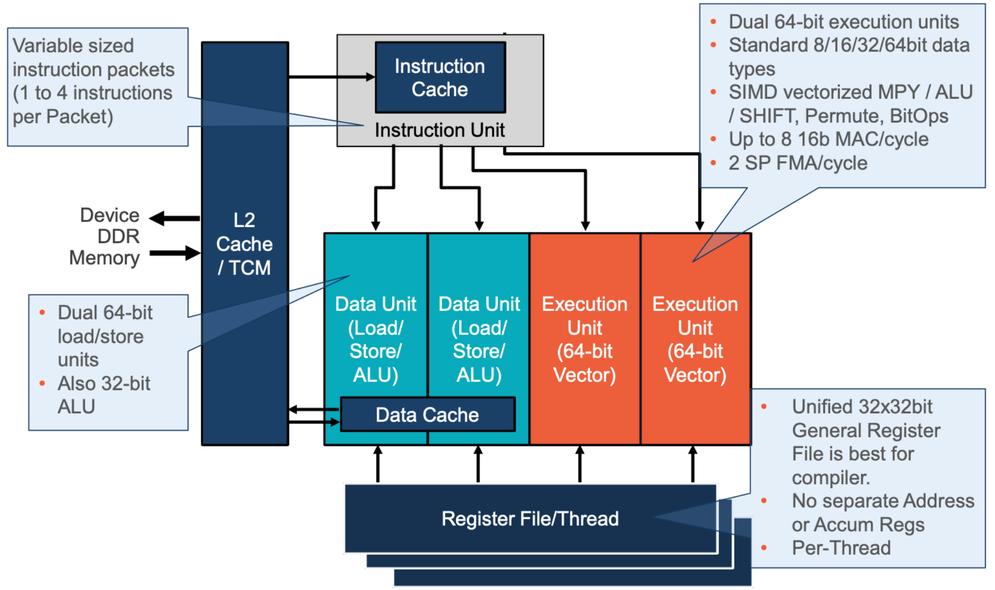
Complex instructions (e.g., SIMD/VLIW): perform many operations per instruction (amortize cost of control)

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

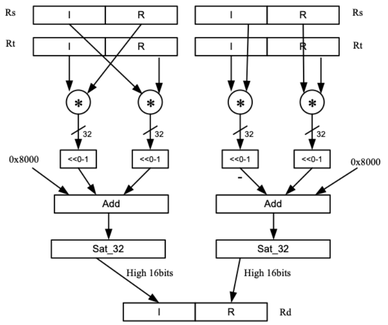


64-bit Load and

64-bit Store with post-update addressing

```
{ R17:16 = MEMD(R0++M1)
  MEMD(R6++M1) = R25:24
  R20 = CMPY(R20, R8):<<1:rnd:sat
  R11:10 = VADDH(R11:10, R13:12)
}:endloop0
```

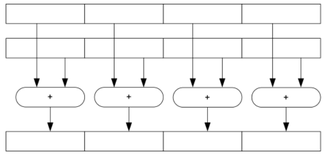
Complex multiply with round and saturation



Zero-overhead loops

- Dec count
- Compare
- Jump top

Vector 4x16-bit Add



Hexagon DSP is in Google Pixel phone

Original iPhone touchscreen controller

Separate digital signal processor to interpret raw signal from capacitive touch sensor (do not burden main CPU)

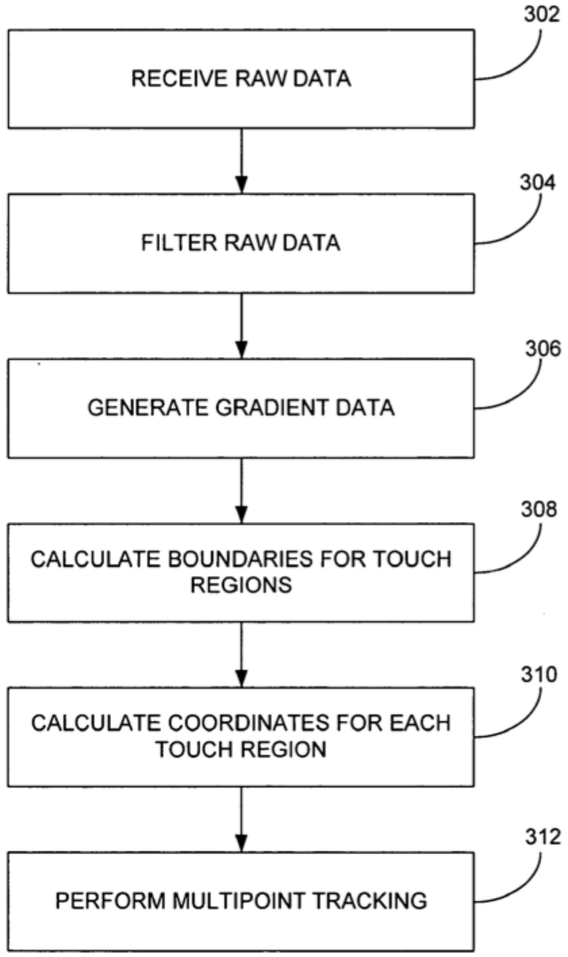


FIG. 16

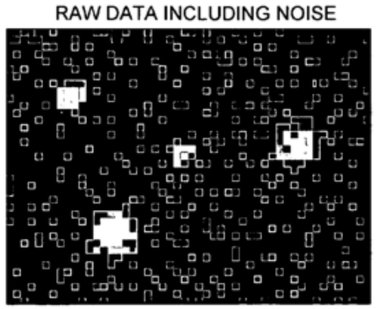


FIG. 17A

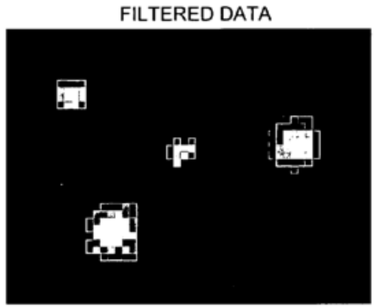


FIG. 17B

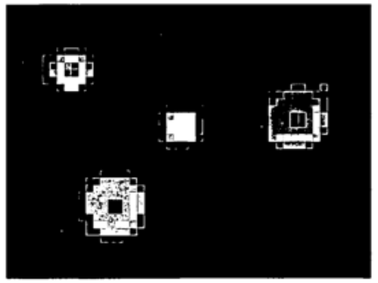


FIG. 17C

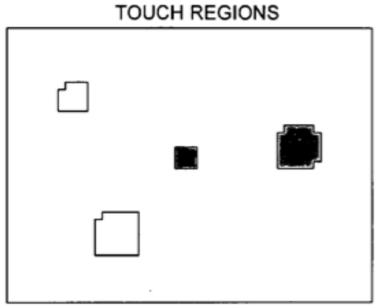


FIG. 17D

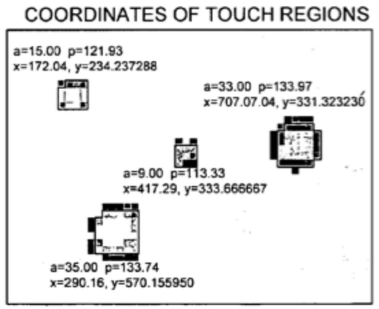


FIG. 17E

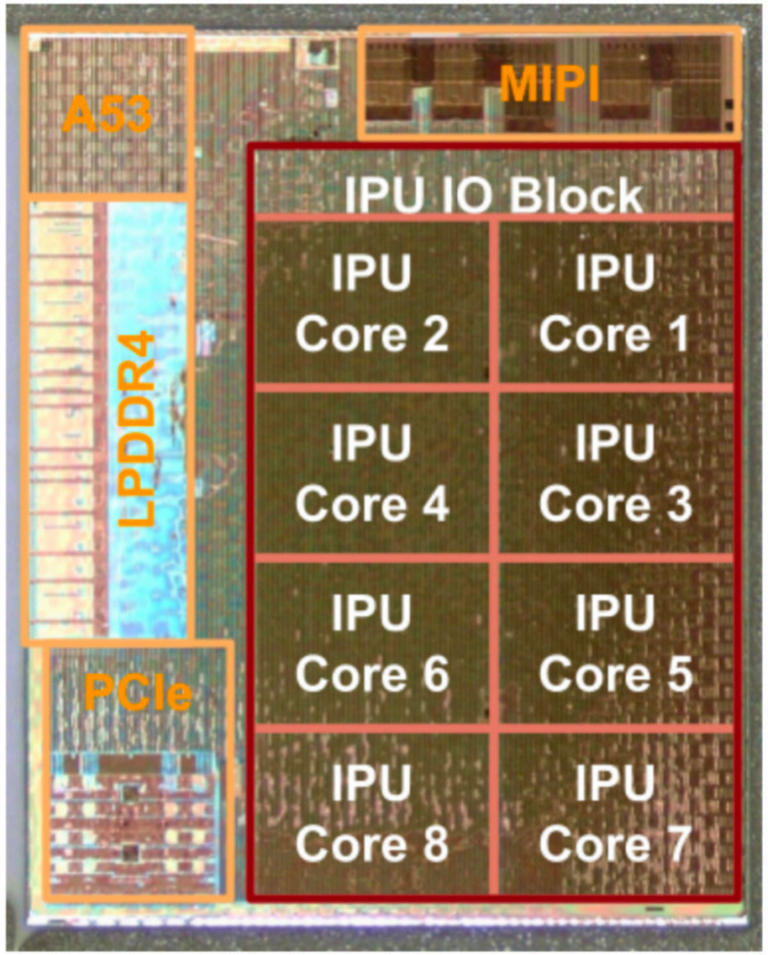
From US Patent Application 2006/0097991

Example: Google's Pixel Visual Core

Programmable "image processing unit" (IPU)

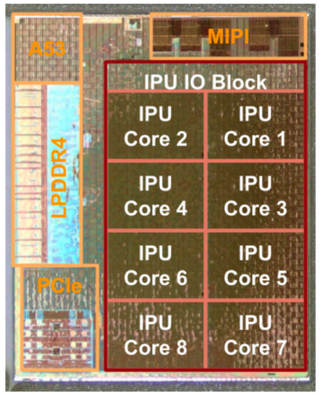
- Each core = 16x16 grid of 16 bit multiply-add ALUs
- ~10-20x more efficient than GPU at image processing tasks

  (Google's claims at HotChips '18)

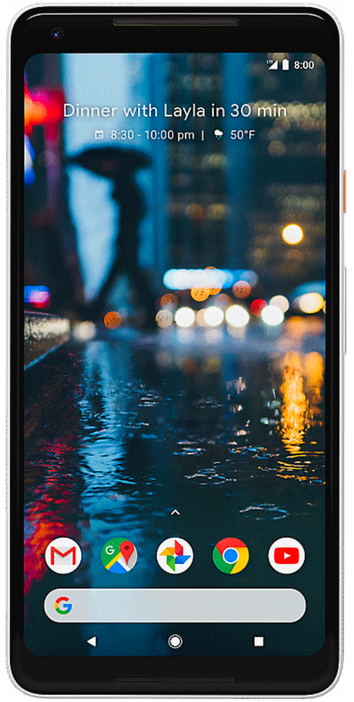


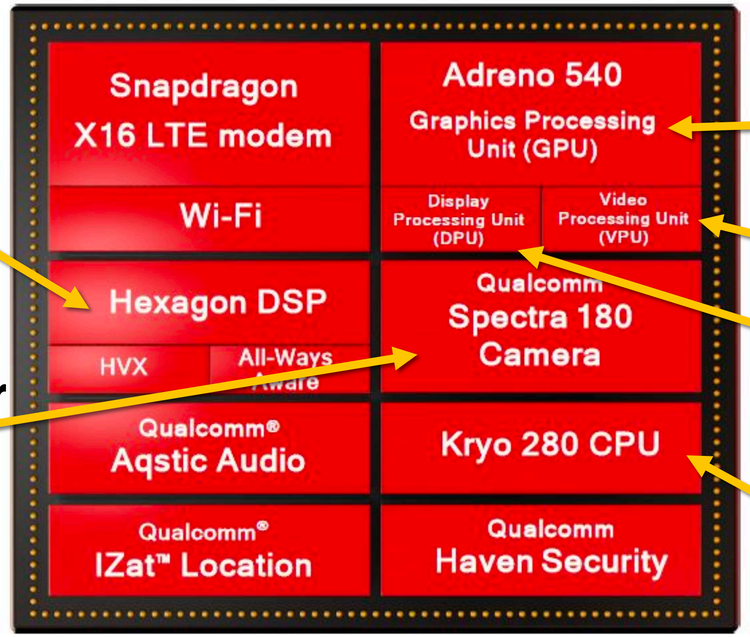
Let's crack open a modern smartphone

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)

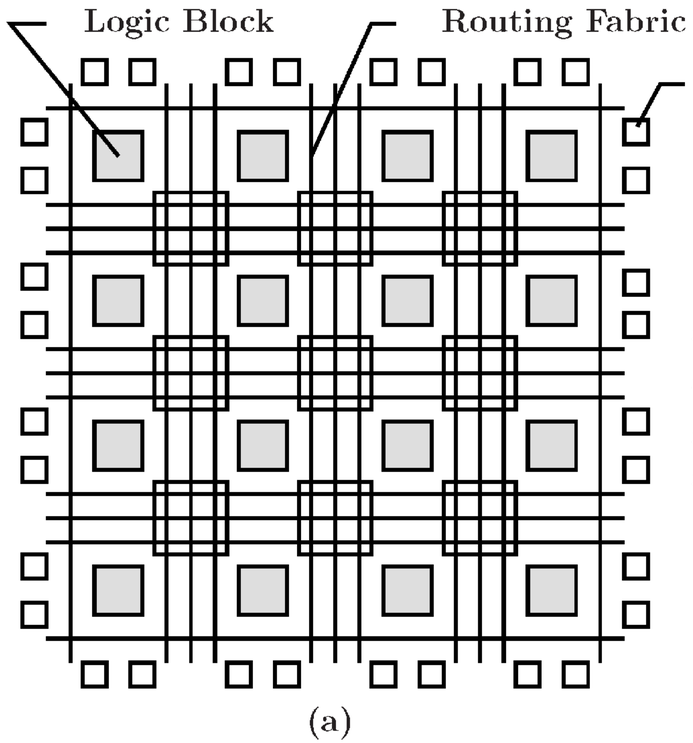
Video encode/decode ASIC

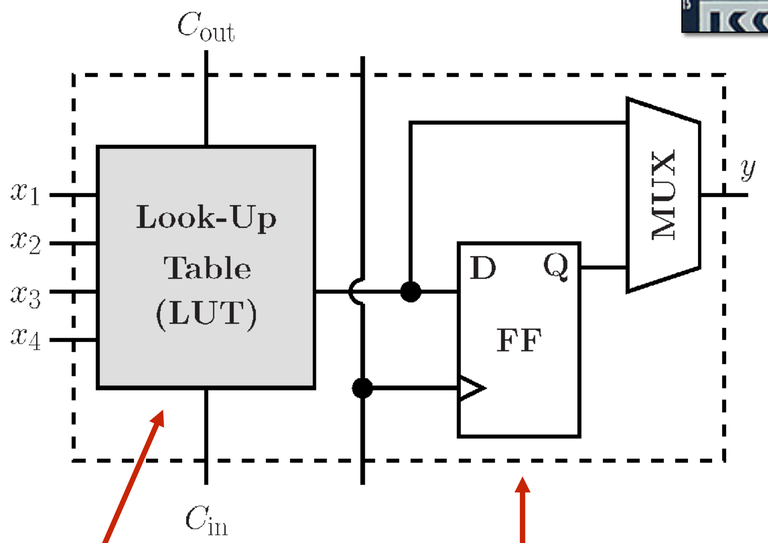
Display engine
(compresses pixels for transfer to high-res screen)

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





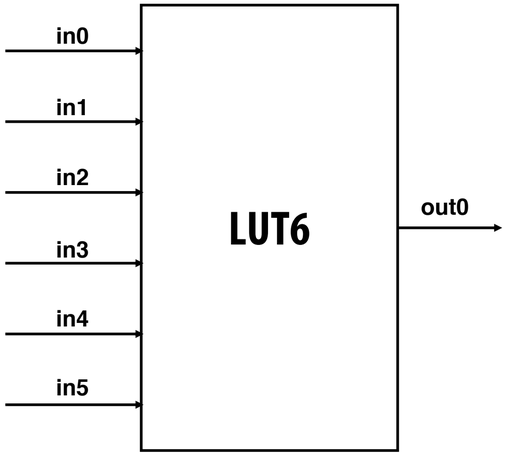
Programmable lookup table (LUT)

Flip flop (a register)

Image credit: Bai et al. 2014

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)

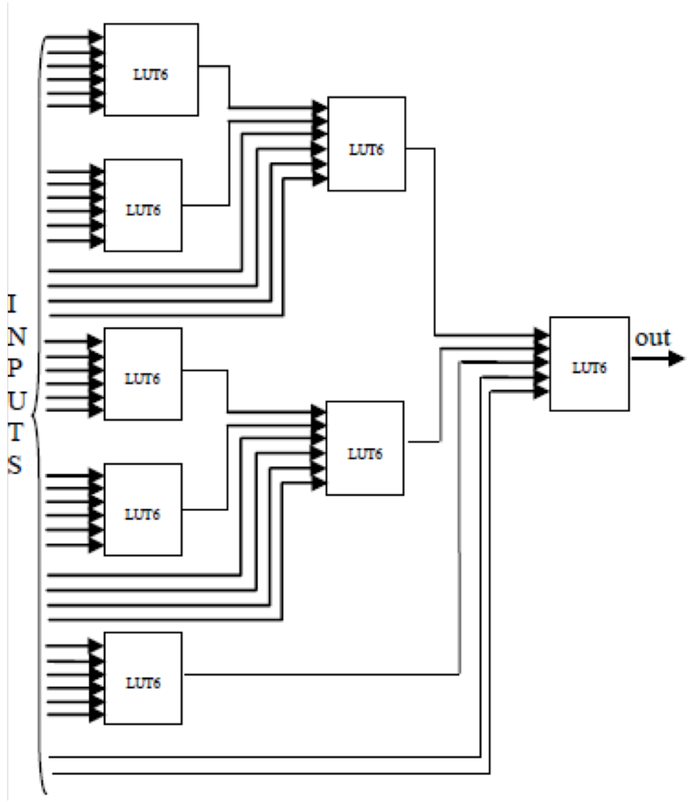


Image credit: [Zia 2013]

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

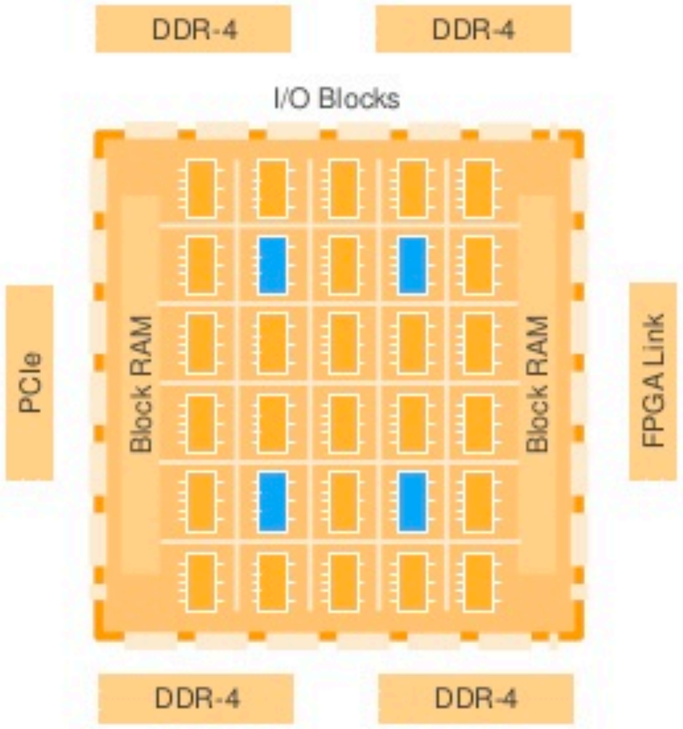
FPGA board

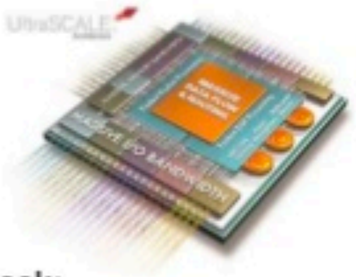
1U server (Dual socket CPU + FPGA connected via PCIe bus)

Amazon F1

- FPGA's are now available on Amazon cloud services

What's Inside the F1 FPGA?





System Logic Block:
Each FPGA in F1 provides over 2M of these logic blocks

DSP (Math) Block:
Each FPGA in F1 has more than 5000 of these blocks

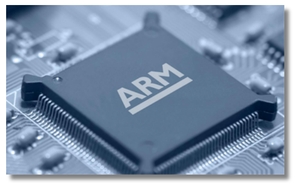
I/O Blocks:
Used to communicate externally, for example to DDR-4, PCIe, or ring

Block RAM:
Each FPGA in F1 has over 60Mb of internal Block RAM, and over 230Mb of embedded UltraRAM

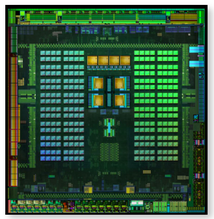
amazon web services | Webinars

Summary: choosing the right tool for the job

Energy-optimized CPU



Throughput-oriented processor (GPU)



Programmable DSP

HEXAGON™

FPGA/ reconfigurable logic

ASIC

Video encode/decode, Audio playback, Camera RAW processing, neural nets (future?)

~10X more efficient

~100X??? (jury still out)

~100-1000X more efficient

Easiest to program

Difficult to program (making it easier is active area of research)

Not programmable + costs 10-100's millions of dollars to design / verify / create

Credit: Pat Hanrahan for this slide design

Challenges of heterogeneous designs:

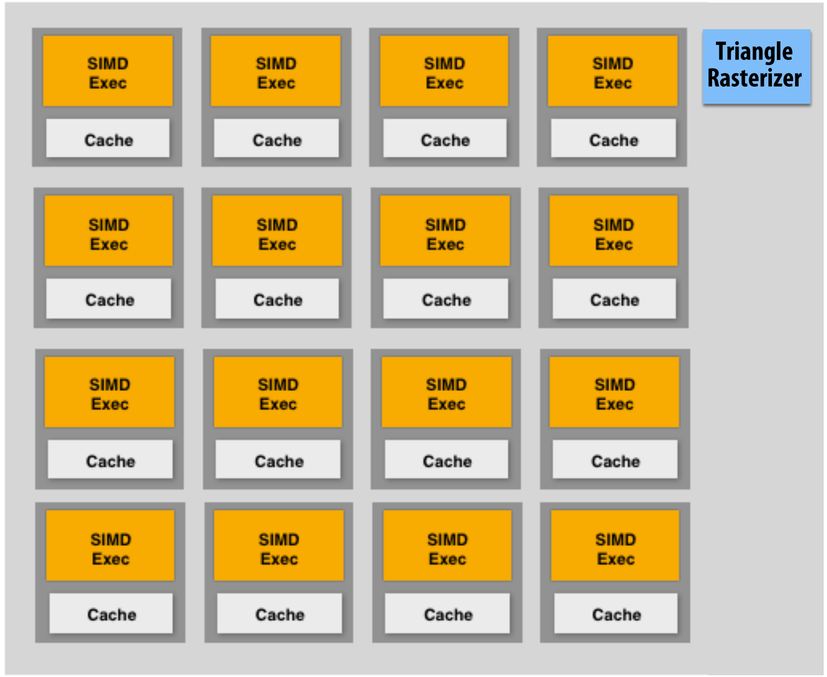
(it's not easy to realize the potential of specialized, heterogeneous processing)

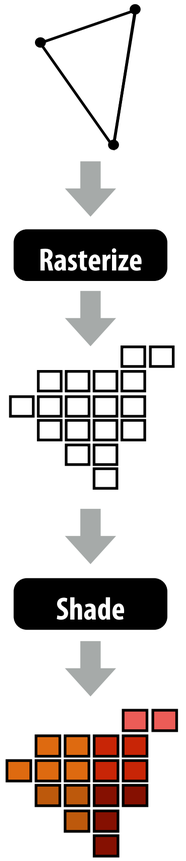
Challenges of heterogeneity

- 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 6 battery: ~7 watt-hours (note: my Macbook Pro laptop: 99 watt-hour battery)
  - 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 CS research challenge: how to write efficient, portable programs for emerging heterogeneous architectures?