CSE 560
Computer Systems Architecture

Domain-Specific Accelerators

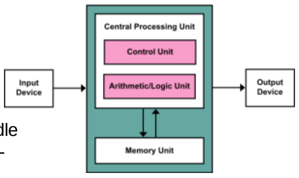
Slides originally developed by Steven Harris

Motivation

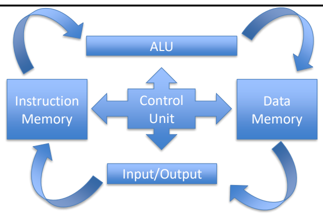
The von Neumann bottleneck is an architectural throughput limitation due to a limited transfer rate between memory and the CPU

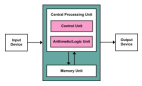
It can cause the CPU to wait idle for long periods due to the low-speed memory transactions

It is also referred to as the "memory wall"



Von-Neumann Bottleneck Mitigation



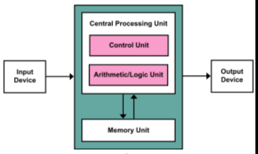


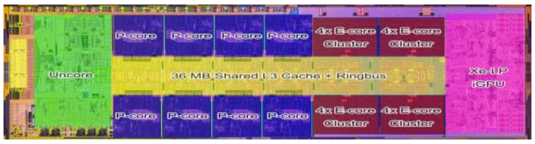
A few implementation suggestions for improving performance include:

- Introduction of cache between the CPU and main memory
- Define separate access paths for data and instructions
- Branch Predictor algorithms and logic
- On-chip scratchpad memory

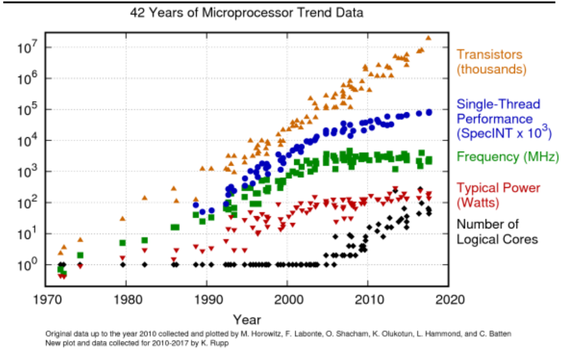
Von-Neumann Processor Journey (Thus far)

- 1st, 2nd, 3rd, & 4th - level caches
- 512-bit SIMD floating-point units
- 15+ stage pipelines
- Branch prediction
- Out-of-order execution
- Speculative prefetching
- Multithreading
- Multiprocessing
- E.g., Intel Core i9-13900K





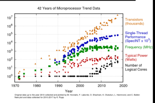
Processor Trends – Performance Plateaus



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Performance Walls



Power

- Increased frequency leads to increased **power density**
- Difficult to mitigate dynamic/static **power dissipation**

Memory

- **Compute bandwidth** continues to outpace **memory bandwidth**
- **Data migration** can become the limiting factor on performance

Instruction Level Parallelism

- Increasingly difficult to find **parallelism** in single-instruction streams
- Diminishing returns on additional ILP hardware

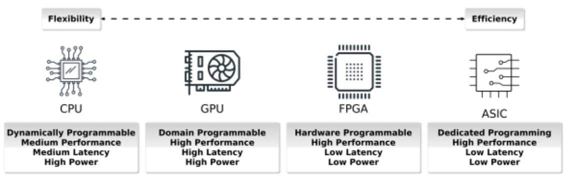
This Unit: Domain-Specific Accelerators

- Survey of Hardware Accelerators (Taxonomy)
  - Architecture
  - Software Aspects
  - Host Coupling
  - General Aspects
  - Domain-Specific Accelerators
- Graphics Engines
  - The Pipeline
  - Shaders
  - GPU Architectural Features
  - Other uses for GPUs

7

7

Common Accelerators

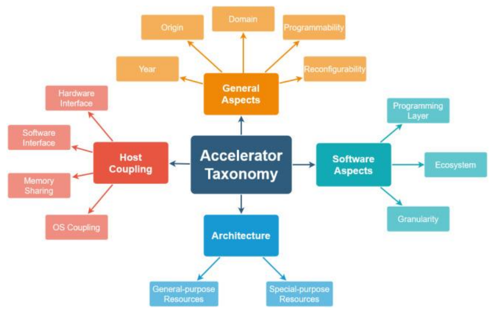


Harris, Steven. "Investigating Single Precision Floating General Matrix Multiply in Heterogeneous Hardware." (2020).

8

8

Hardware Accelerators

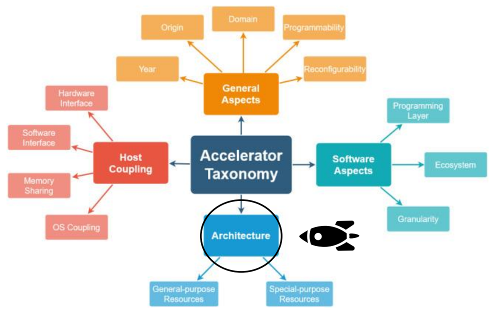


Peccerillo, Biagio, et al. "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives." Journal of Systems Architecture (2022): 102561.

9

9

Hardware Accelerator Architecture

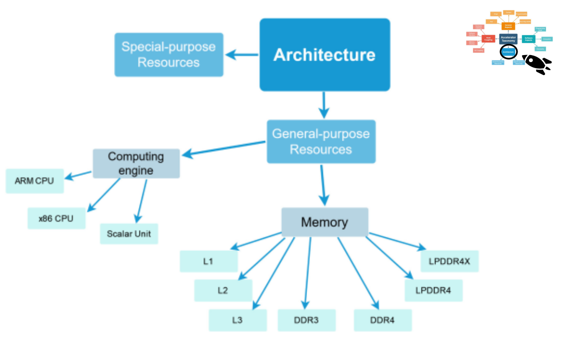


Peccerillo, Biagio, et al. "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives." Journal of Systems Architecture (2022): 102561.

10

10

Architecture -- Enumerated

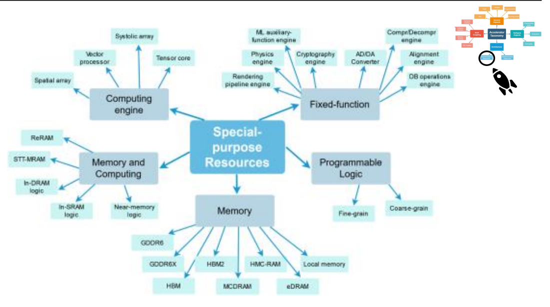


Peccerillo, Biagio, et al. "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives." Journal of Systems Architecture (2022): 102561.

11

11

Special-purpose Resources



Peccerillo, Biagio, et al. "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives." Journal of Systems Architecture (2022): 102561.

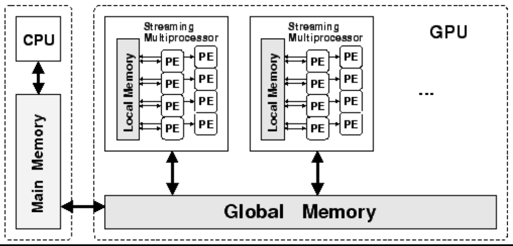
12

12

Graphics Engines

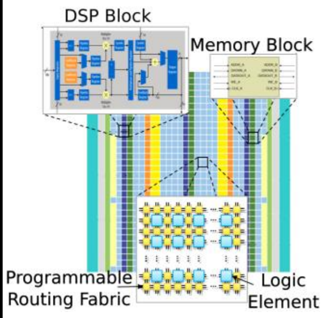
Heterogeneous Multiprocessor

- Many processing elements (PE), many threads per PE
- Collections of threads execute in lock-step (SIMD-like)
  - Hide latency to memory by switching threads



13

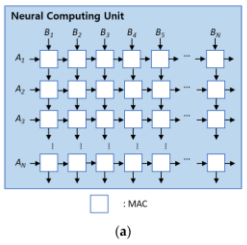
What is an FPGA?

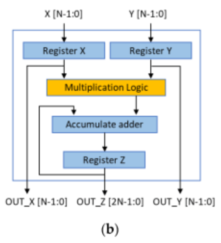


- Field-programmable gate array
  - Array of logic gates
  - Programmable in the “field”
  - Basically, custom logic on a chip
- Enables hardware design
  - Custom data path
  - Can be very fast and energy efficient
- Challenge is now to architect design
  - HW has many degrees of freedom

14

Systolic Array

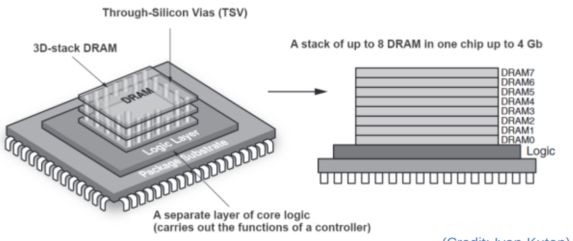




Cho et al., “Efficient Systolic-Array Redundancy Architecture for Offline/Online Repair,” *Electronics*, 9(2):338, 2020.

15

3D Stacked Memory Technology


Through-Silicon Vias (TSV)
3D-stack DRAM
A stack of up to 8 DRAM in one chip up to 4 Gb
DRAM
Logic Layer
Package Substrate
DRAM7
DRAM6
DRAM5
DRAM4
DRAM3
DRAM2
DRAM1
DRAM0
Logic
A separate layer of core logic (carries out the functions of a controller)

(Credit: Ivan Kuten)

16

2.5D GPU System

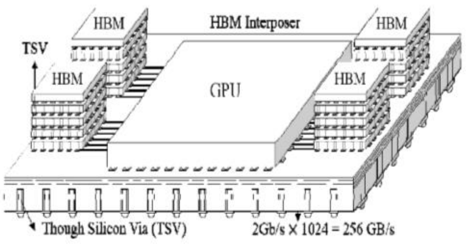
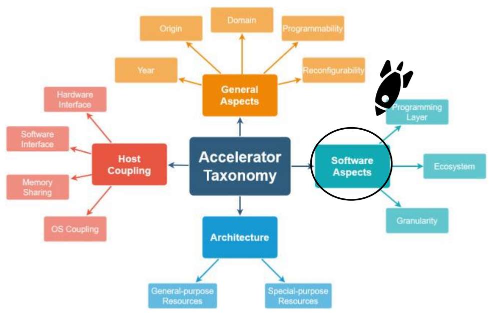


Figure 1. The Conceptual view of HBM interposer employed 4 HBMs and 1 graphic processing unit (GPU) for TB/s bandwidth module

Cho et al., DOI: 10.1109/ECTC.2016.84

17

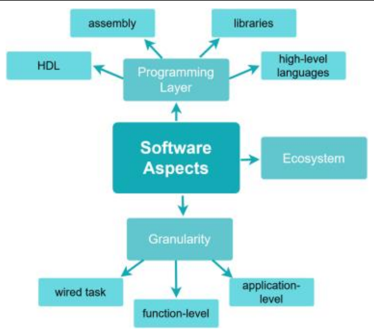
Software Aspects



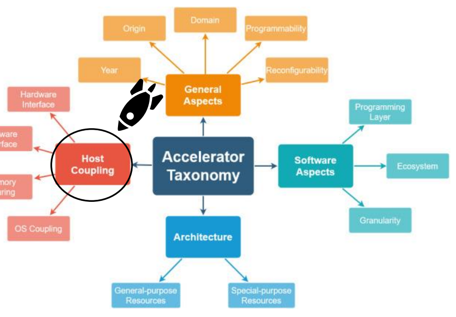
Peccerillo, Biagio, et al. "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives." Journal of Systems Architecture (2022): 102561.

18

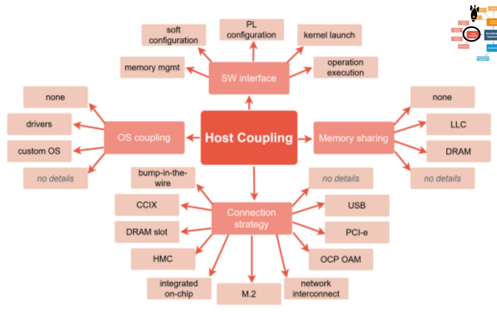
Software Aspects -- Enumerated



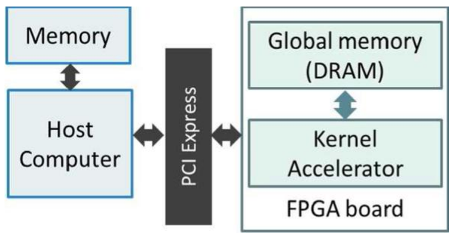
Host Coupling



Host Coupling -- Enumerated

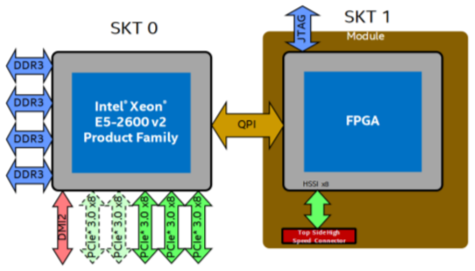


Traditional – Via I/O Bus

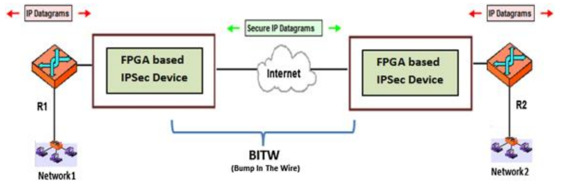


Intel HARP - Hardware Accelerator Research Program

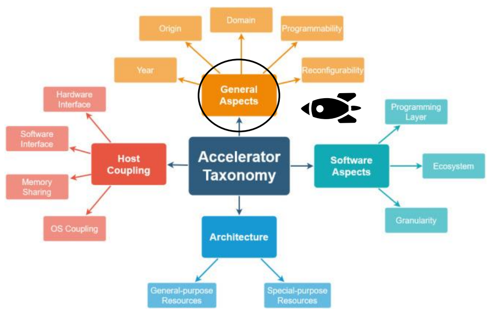
- FPGA tied to last-level cache on Xeon
- Cache coherent interconnect



Bump In The Wire

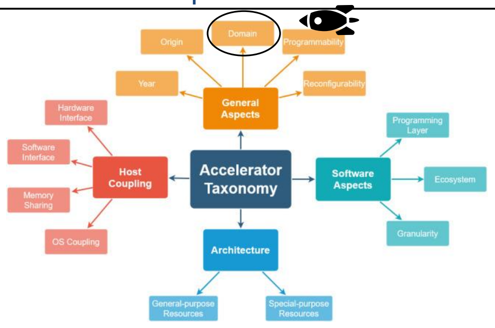


General Aspects



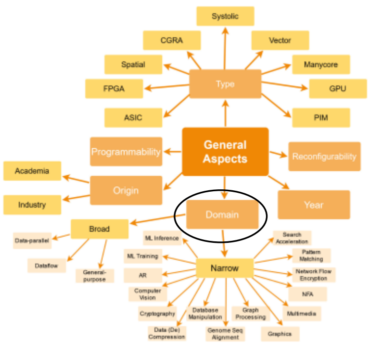
25

Domain-Specific Accelerators



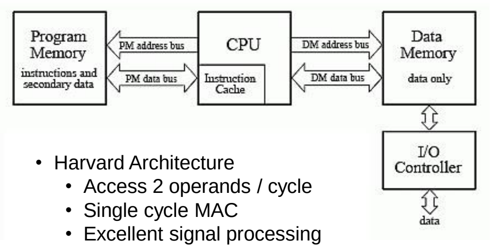
26

General Aspects – Domain



27

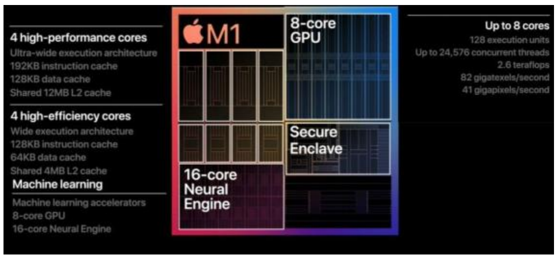
Digital Signal Processor



- Harvard Architecture
  - Access 2 operands / cycle
  - Single cycle MAC
  - Excellent signal processing performance

28

Apple M1 Chip



4 high-performance cores
Ultra-wide execution architecture
192KB instruction cache
128KB data cache
Shared 12MB L2 cache

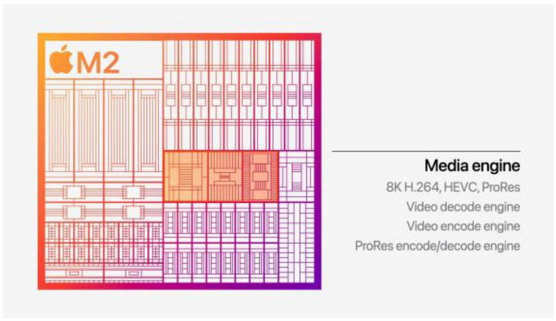
4 high-efficiency cores
Wide execution architecture
128KB instruction cache
64KB data cache
Shared 4MB L2 cache

Machine learning
Machine learning accelerators
8-core GPU
16-core Neural Engine

Up to 8 cores
128 execution units
Up to 24,576 concurrent threads
2.6 teraflops
82 gigatexels/second
41 gigapixels/second

29

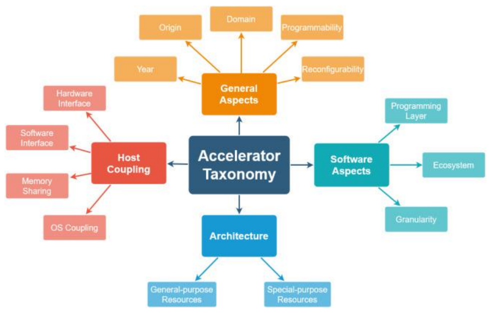
Apple M2 Chip



Media engine
8K H.264, HEVC, ProRes
Video decode engine
Video encode engine
ProRes encode/decode engine

30

Hardware Accelerators



Peccerillo, Biagio, et al. "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives." Journal of Systems Architecture (2022): 102561.

31

31

Graphics Engines

32

32

Outline

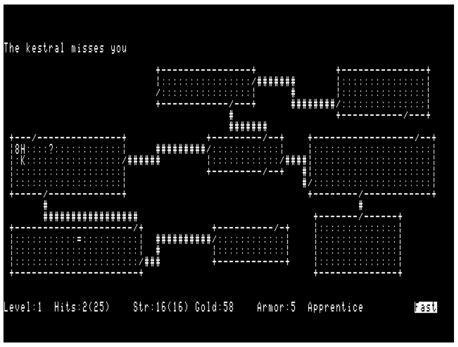
- History of Consumer Level Graphics
- Motivation
- The Modern Graphics Pipeline
- Shader Programs
- GPU Architectural Features
- Other uses for GPUs

Based on "From Shader Code to a Teraflop: How GPU Shader Cores Work", By Kayvon Fatahalian, Stanford University

33

33

In the Beginning…



```
The kestral misses you

Level:1  Hits:2(25)    Str:16(16) Gold:58    Armor:5  Apprentice          Fast
```

34

34

Simple Graphics Modes

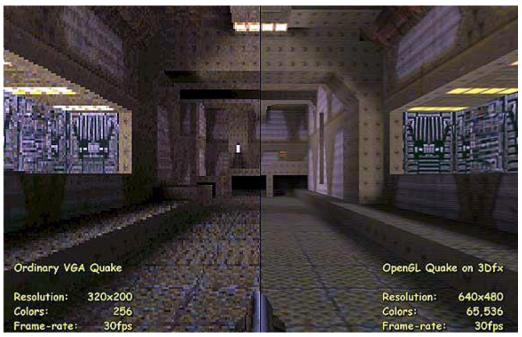
Bitmap Mode

Character Mode

35

35

The 3D Reckoning



Ordinary VGA Quake
Resolution: 320x200
Colors: 256
Frame-rate: 30fps

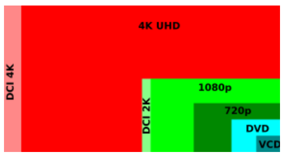
OpenGL Quake on 3Dfx
Resolution: 640x480
Colors: 65,536
Frame-rate: 30fps

36

36

Motivating Problem

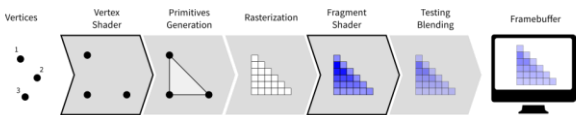
- Convert 3D models into something that can be drawn on screen
- Consumer displays easily hit 4K resolution (8mil+ pixels)
- Most applications target 30+ frames per second minimum
- It is infeasible to do this on even the fastest single threaded devices today



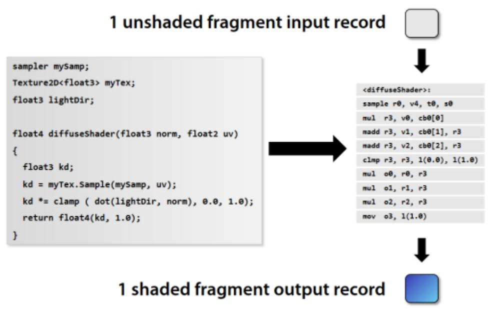
Graphics Engines

Conceptual model

- Apply simple sequential programs to all items in a set
- Eg, Vertices, Faces, Fragments, Pixels
- Many programs (called shaders) connected in series to form a graphics pipeline



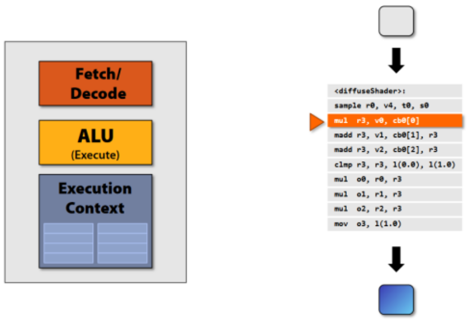
Shader Program Example


1 unshaded fragment input record
1 shaded fragment output record

```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
  float3 kd;
  kd = myTex.Sample(mySamp, uv);
  kd *= clamp ( dot(lightDir, norm), 0.0, 1.0);
  return float4(kd, 1.0);
}
```

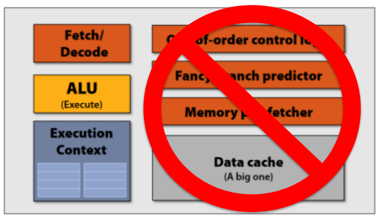
```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```

Executing a Shader

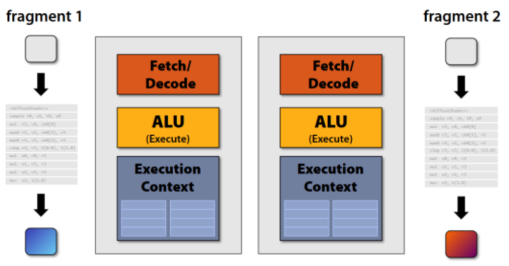


```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```

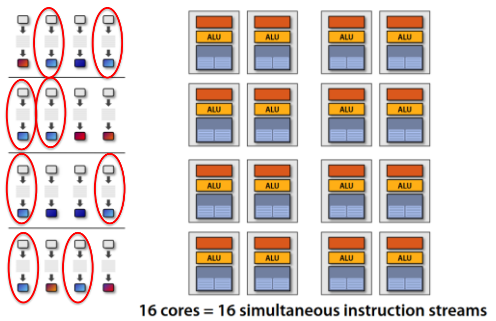
CPU-lite



More Room=More Cores



Scaling up

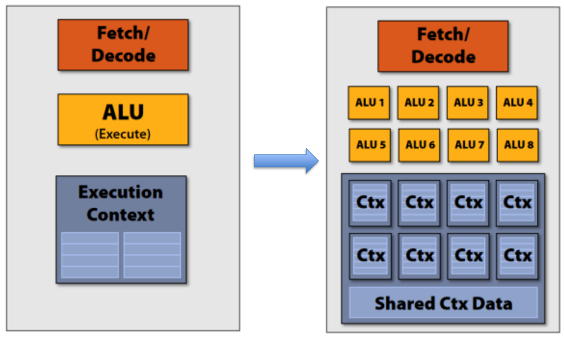


16 cores = 16 simultaneous instruction streams

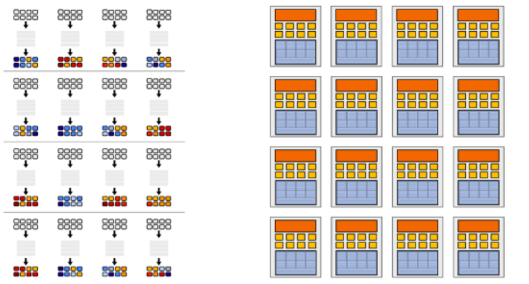
Flynn’s Taxonomy

- Proposed by Michael Flynn in 1966
- SISD – single instruction, single data
  - Traditional uniprocessor
- SIMD – single instruction, multiple data
  - Execute the same instruction on many data elements
  - Vector machines, graphics engines
- MIMD – multiple instruction, multiple data
  - Each processor executes its own instructions
  - Multicores are all built this way
  - SPMD – single program, multiple data (extension proposed by Frederica Darema)
    - MIMD machine, each node is executing the same code
- MISD – multiple instruction, single data
  - Systolic array

SIMD Cores



Why Not Both?

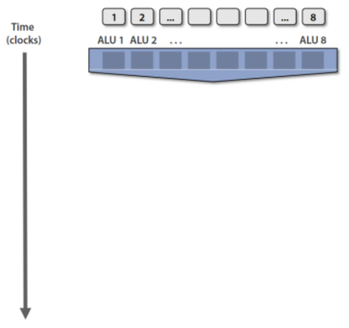


16 cores = 128 ALUs , 16 simultaneous instruction streams

What’s Missing?

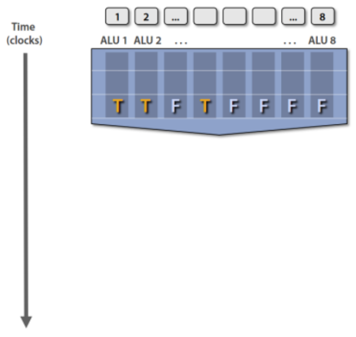
```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
  float3 kd;
  kd = myTex.Sample(mySamp, uv);
  kd *= clamp ( dot(lightDir, norm), 0.0, 1.0);
  return float4(kd, 1.0);
}
```

Divergence



```
<unconditional
 shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

Divergence

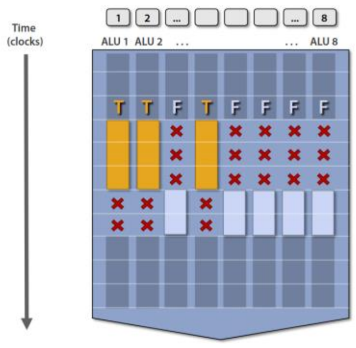


```
<unconditional
shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

50

50

Divergence



```
<unconditional
shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

51

51

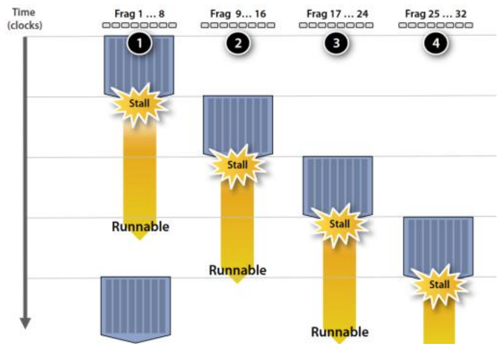
Feeding Cores with Data

- Recall that we removed the hardware that allows the CPU to avoid stalls
- OOE, branch predictors and prefetching all gone
- Question remains: How do we avoid execution stalls?

52

52

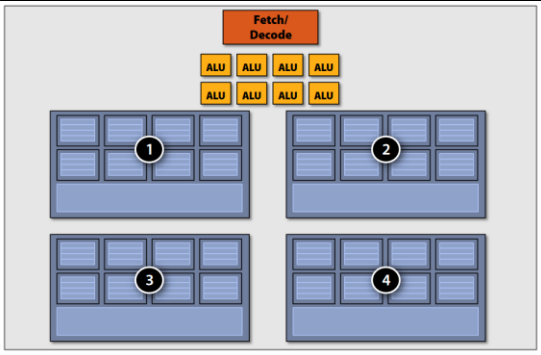
Cheating Stalls



53

53

Managing In-Flight Instruction Stream



54

54

When Interleaving Isn't Enough

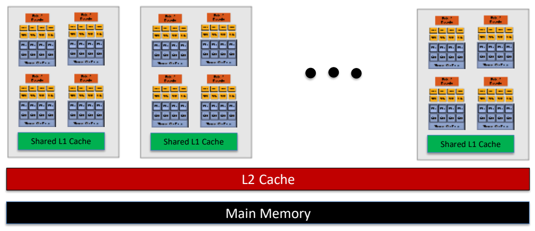
- Interleaving was great in the 00's
- GeForce 6: 2005
  - 500MHz core clock
  - 36GB/s memory bandwidth
  - 16 Pixel Processors
- GeForce 3000: 2020
  - 1700MHz core clock (OC: 2 GHz)
  - 935GB/s memory bandwidth
  - 82 SMs, 4 cores per SM
  - Plus extras

55

55

Caches are Back

- Reintroduce L1 and L2 caches
- Intends to capture locality of data



Other Uses for GPUs

- Wide Read -> Compute -> Write paradigm is actually very useful for other applications
  - Non-render image processing
  - Fluid Simulations
  - Scientific computing
  - Machine Learning

At their core these are all matrix Multiplies