Advanced Computer Architecture

高 级 计 算 机 体 系 结 构

第12讲：概述、ISA&ILP (3)

张献伟

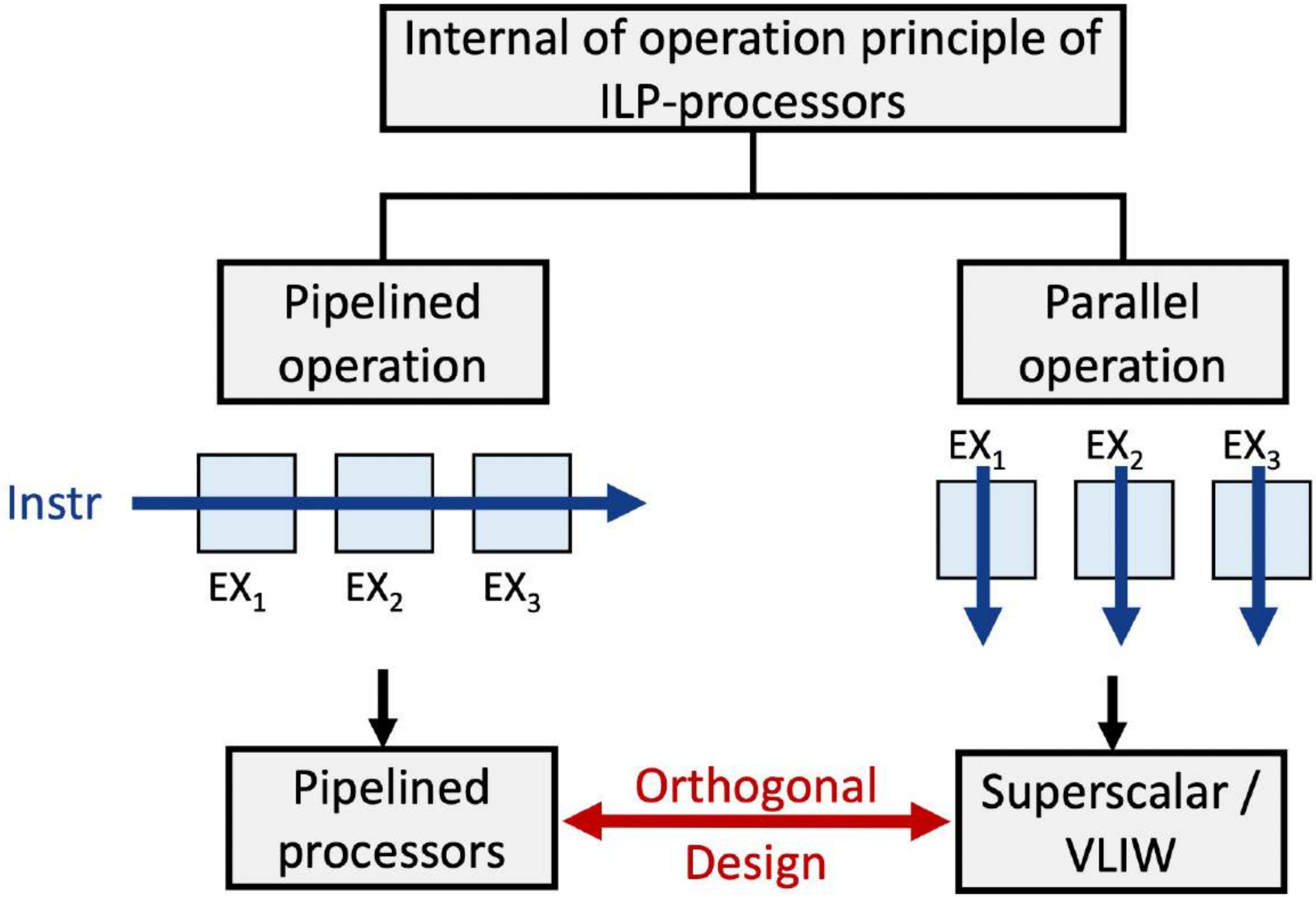
xianweiz.github.io

DCS5637, 11/16/2022

Part-III: ISA & ILP

中山大學
SUN YAT-SEN UNIVERSITY

Another ILP



http://camelab.org/uploads/Main/lecture06-istruction-paralllel-processing.pdf

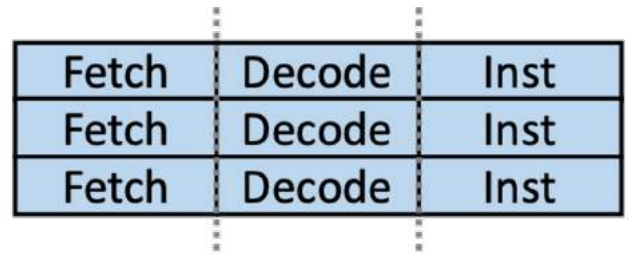
Multiple Issue[多发射]

- To achieve CPI < 1, need to complete multiple instructions per clock
- Solutions:
  - Statically scheduled superscalar processors
  - VLIW (very long instruction word) processors
  - Dynamically scheduled superscalar processors

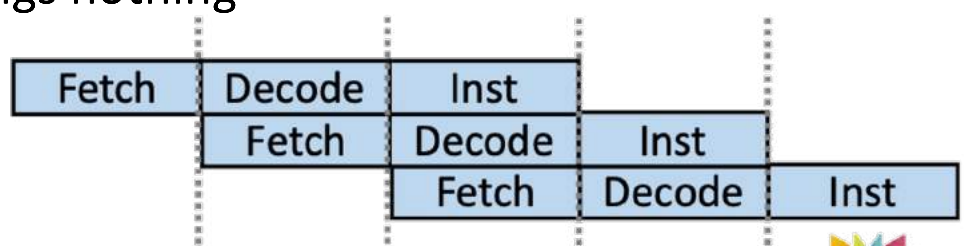
Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the Cortex-A53
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

Superscalar[超标量]

- Superscalar architectures allow several instructions to be issued and completed per clock cycle
- A superscalar architecture consists of a number of pipelines that are working in parallel (N-way Superscalar)
  - Can issue up to N instructions per cycle
- Superscalarity is Important
  - Ideal case of N-way Super-scalar
    - All instructions were independent
    - Speedup is “N” (Superscalarity)



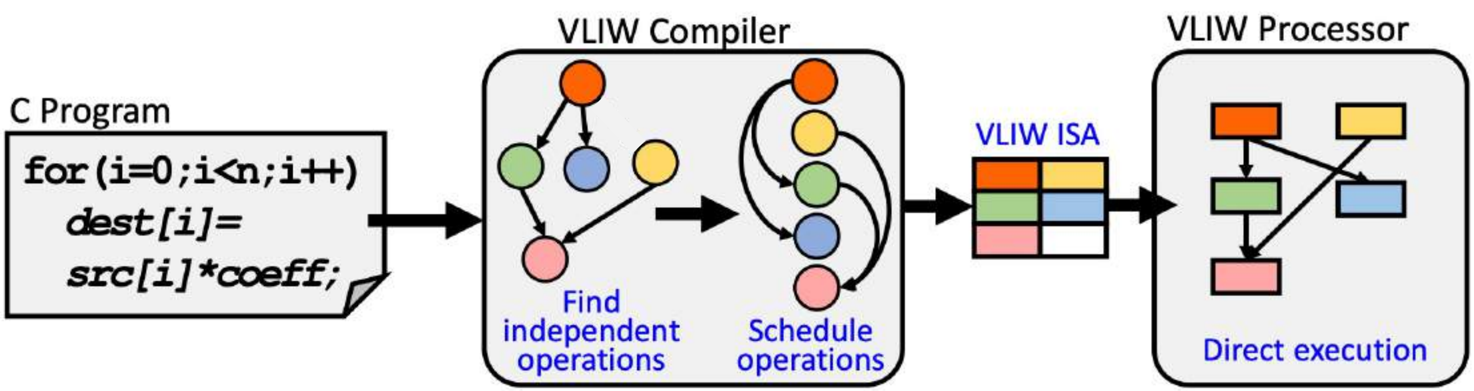
  - What if all instructions are dependent?
    - No speed up, super-scalar brings nothing
    - (Just similar to pipelining)



http://camelab.org/uploads/Main/lecture06-istruction-paralllel-processing.pdf

VLIW Processor[超长指令字]

- Static multiple-issue processors (decision making at compile time by the compiler)
  - Package multiple operations into one instruction
- Key idea: replace a traditional sequential ISA with a new ISA that enables the compiler to encode ILP directly in the hw/sw interface
  - Sub-instructions within a long instruction must be independent
  - Multiple "sub-instructions" packed into one long instruction
  - Each "slot" in a VLIW instruction for a specific functional unit

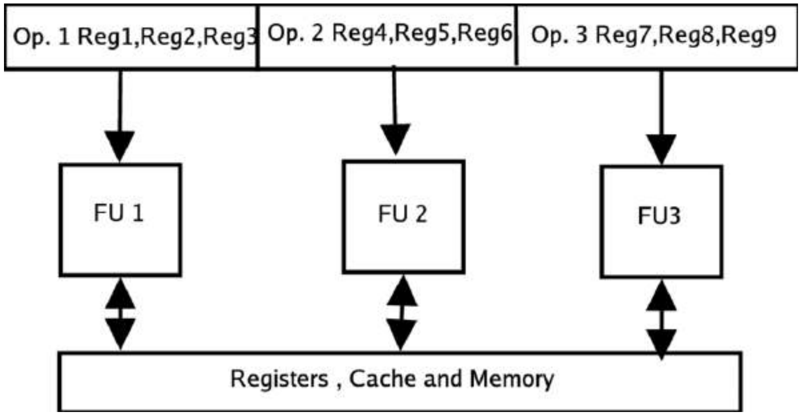


http://camelab.org/uploads/Main/lecture06-istruction-paralllel-processing.pdf

VLIW Processor (cont.)

Memory reference 1	Memory reference 2	FP operation 1	FP operation 2	Integer operation/branch
fld f0,0(x1)	fld f6,-8(x1)			
fld f10,-16(x1)	fld f14,-24(x1)			
fld f18,-32(x1)	fld f22,-40(x1)	fadd.d f4,f0,f2	fadd.d f8,f6,f2	
fld f26,-48(x1)		fadd.d f12,f0,f2	fadd.d f16,f14,f2	
		fadd.d f20,f18,f2	fadd.d f24,f22,f2	
fsd f4,0(x1)	fsd f8,-8(x1)	fadd.d f28,f26,f24		
fsd f12,-16(x1)	fsd f16,-24(x1)			addi x1,x1,-56
fsd f20,24(x1)	fsd f24,16(x1)			
fsd f28,8(x1)				bne x1,x2,Loop

- Disadvantages:
  - Statically finding parallelism
  - Code size
  - No hazard detection hardware
  - Binary code compatibility



Summary: Multiple Issue

- Single issue: ideal CPI of one
  - Issue only one inst every clock cycle
  - Techniques to eliminate data, control stalls
- Multiple issue: ideal CPI less than one
  - Issue multiple insts in a clock cycle
  - **Statically scheduled superscalar** processors
    - Issue varying number of insts per clock, execute in-order
  - **VLIW** (very long inst word) processors
    - Issue a fixed number of insts formatted as one large inst
    - Inherently statically scheduled by the compiler
  - **Dynamically scheduled superscalar** processors
    - Issue varying number of insts per clock, execute OoO

Advanced Computer Architecture

高级计算机体系结构

第12讲：DLP & GPU (1)

张献伟

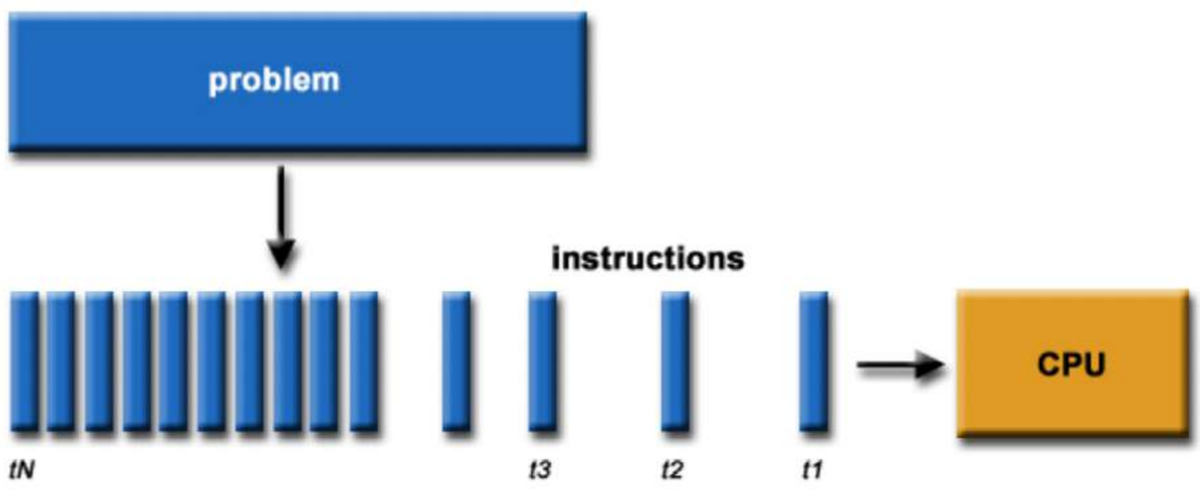
xianweiz.github.io

DCS5637, 11/16/2022

Part-I: DLP

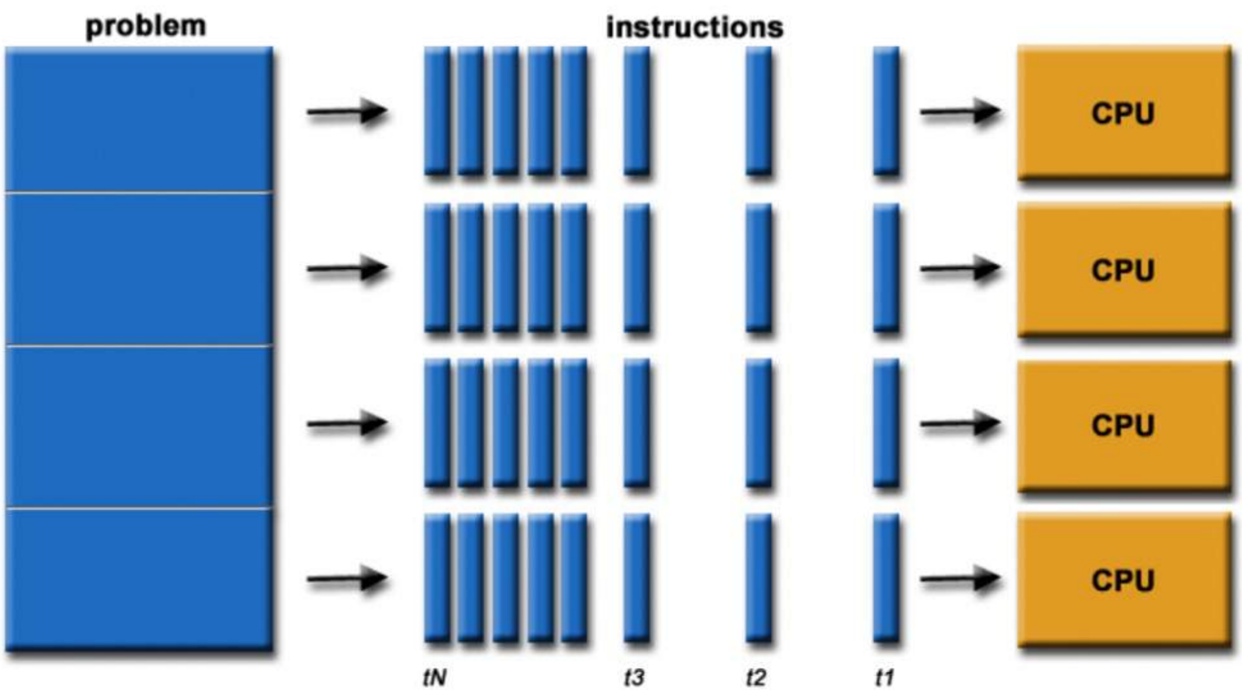
Serial Computing[串行计算]

- Traditionally, software has been written for serial computation
  - To be run on a single computer having a single CPU
  - A problem is broken into a discrete series of instructions
  - Instructions are executed one after another
  - Only one instruction may execute at any moment



Parallel Computing[并行计算]

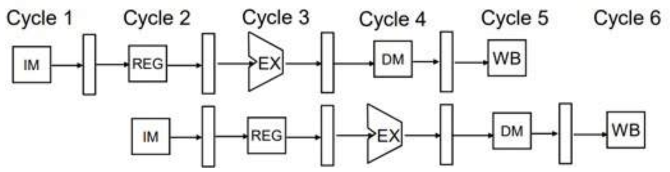
- Simultaneously use multiple compute resources to solve a computational problem
  - Typically in high-performance computing (HPC)
- HPC focuses on performance
  - To solve biggest possible problems in the least possible time



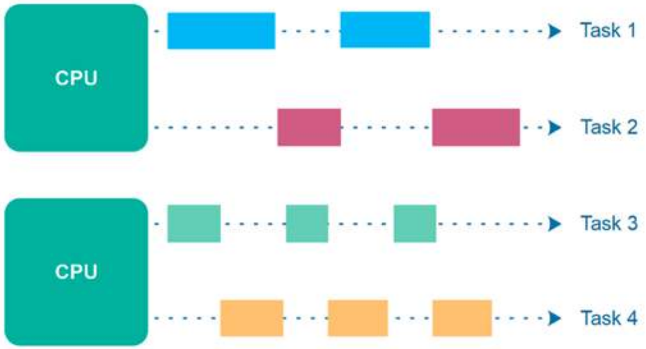
https://www.ima.umn.edu/materials/2010-2011/T11.28-29.10/10287/IMA-PPtTutorial.pdf

Types of Parallel Computing[并行类型]

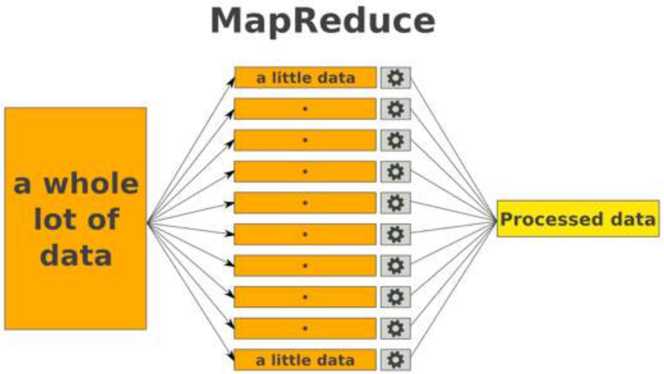
- Instruction level parallelism[指令级并行]
  - Classic RISC pipeline (fetch, …, write back)



- Task parallelism[任务级并行]
  - Different operations are performed concurrently
  - Task parallelism is achieved when the processors execute on the same or different data



- Data parallelism[数据级并行]
  - Distribution of data across different parallel computing nodes
  - Data parallelism is achieved when each processor performs the same task on different pieces of the data



https://www.uio.no/studier/emner/matnat/ifi/IN5050/v20/undervisningsmaterialet/in5050-simd.pdf

Taxonomy[分类]

- **Flynn's Taxonomy** (1966) is widely used to classify parallel computers
  - Distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction Stream*** and ***Data Stream***
  - Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***
- 4 possible classifications according to Flynn

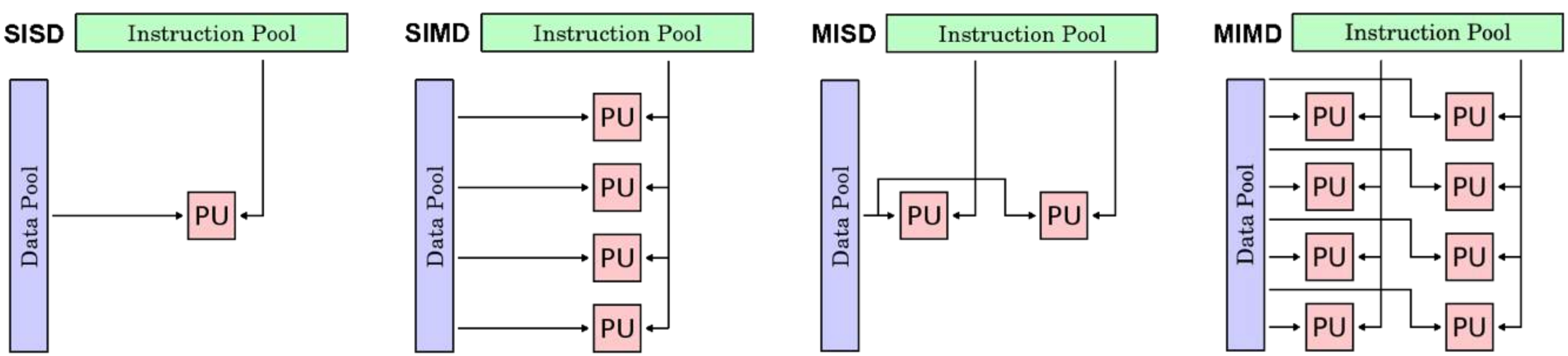
SISD Single Instruction stream Single Data stream	**SIMD** Single Instruction stream Multiple Data stream
MISD Multiple Instruction stream Single Data stream	**MIMD** Multiple Instruction stream Multiple Data stream

https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

中山大學 SUN YAT-SEN UNIVERSITY

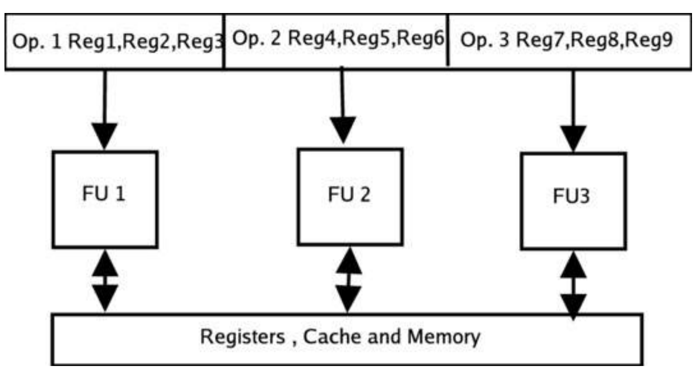
Taxonomy (cont.)

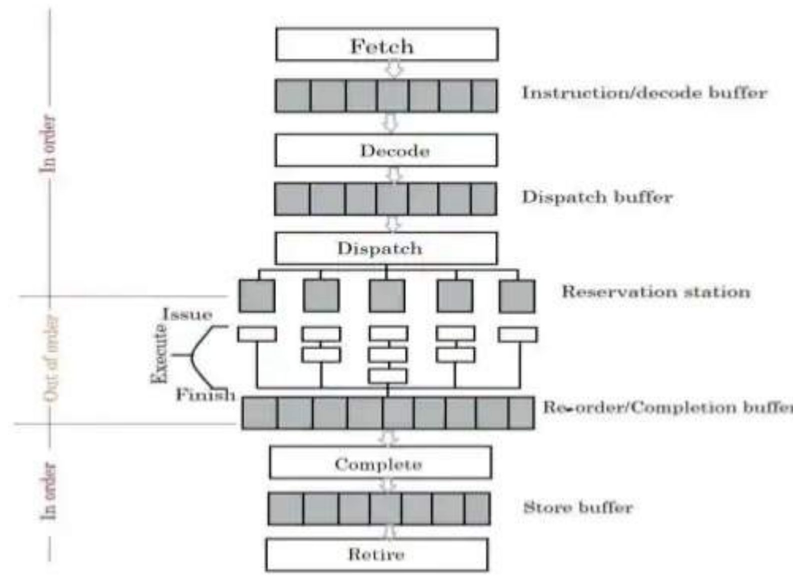
- SISD: single instruction, single data
  - A serial (non-parallel) computer
- **SIMD**: single instruction, multiple data
  - Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing
- MISD: multiple instruction, single data
  - Few (if any) actual examples of this class have ever existed
- MIMD: multiple instruction, multiple data
  - Examples: supercomputers, multi-core PCs, VLIW



SIMD: vs. superscalar and VLIW[对比]

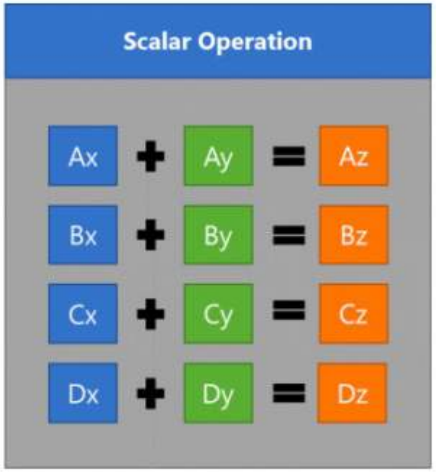
- SIMD performs the same operation on multiple data elements with one single instruction
  - Data-level parallelism
- Superscalar dynamically issues multi insts per clock[超标量]
  - Instruction level parallelism (ILP)
- VLIW receives long instruction words, each comprising a field (or opcode) for each execution unit[超长指令字]
  - Instruction level parallelism (ILP)

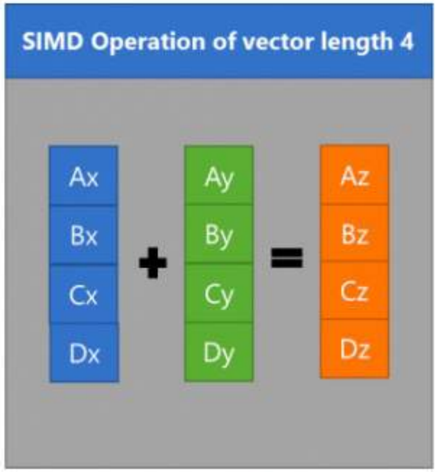




SIMD: Vector Processors[向量处理器]

- Vector processor (or array processor)[处理器]
  - CPU that implements an instruction set containing instructions that operate on one-dimensional arrays (vectors)
- People use vector processing in many areas[应用]
  - Scientific computing
  - Multimedia processing (compression, graphics, image processing, …)
- Instruction sets[指令集]
  - MMX
  - SSE
  - AVX
  - NEON
  - …

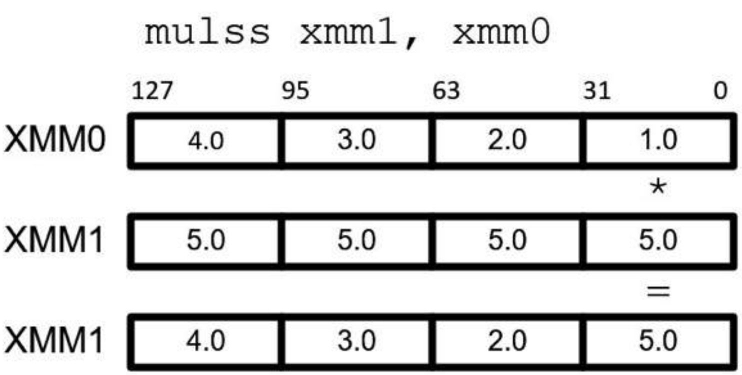


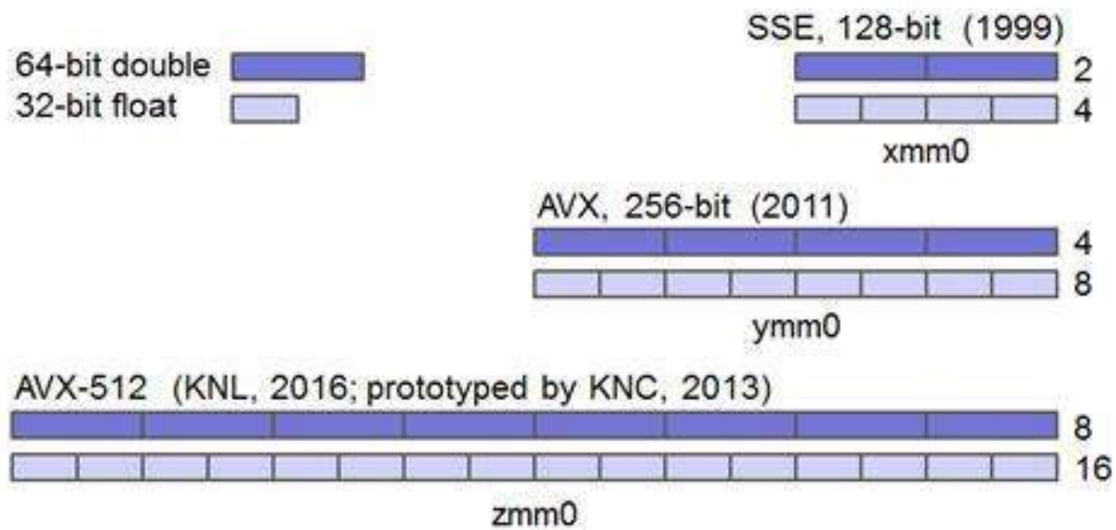


https://www.uio.no/studier/emner/matnat/ifi/IN5050/v20/undervisningsmaterialet/in5050-simd.pdf

SIMD: SSE & AVX

- SSE: Streaming SIMD Extensions
  - 8 new 128-bit registers (xmm0 ~ 7) for FP32 computations
    - Since each register is 128-bit long, we can store total 4 FP32 numbers
- AVX: Advanced Vector Extensions
  - A new-256 bit instruction set extension to SSE
    - 16-registers available in x86-64
  - Yet a proposed extension is AVX-512
    - A 512-bit extension to the 256-bit XMM

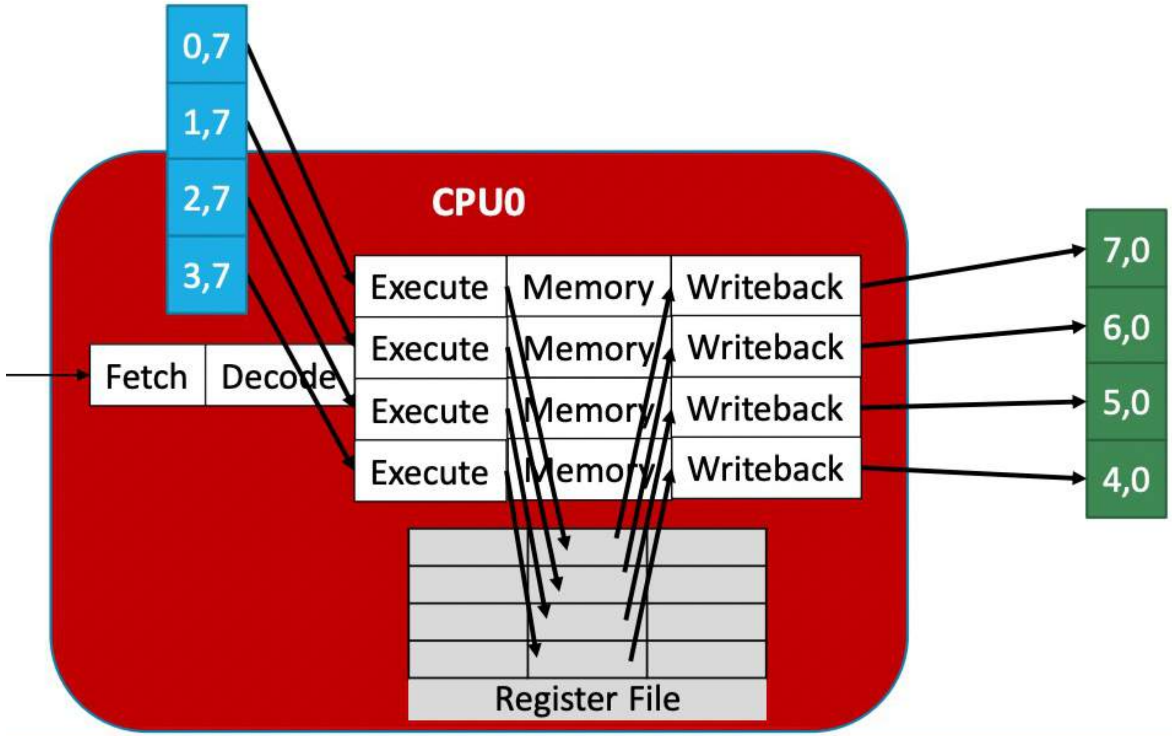




https://www.uio.no/studier/emner/matnat/ifi/IN5050/v20/undervisningsmaterialet/in5050-simd.pdf

Data Parallelism: SIMD

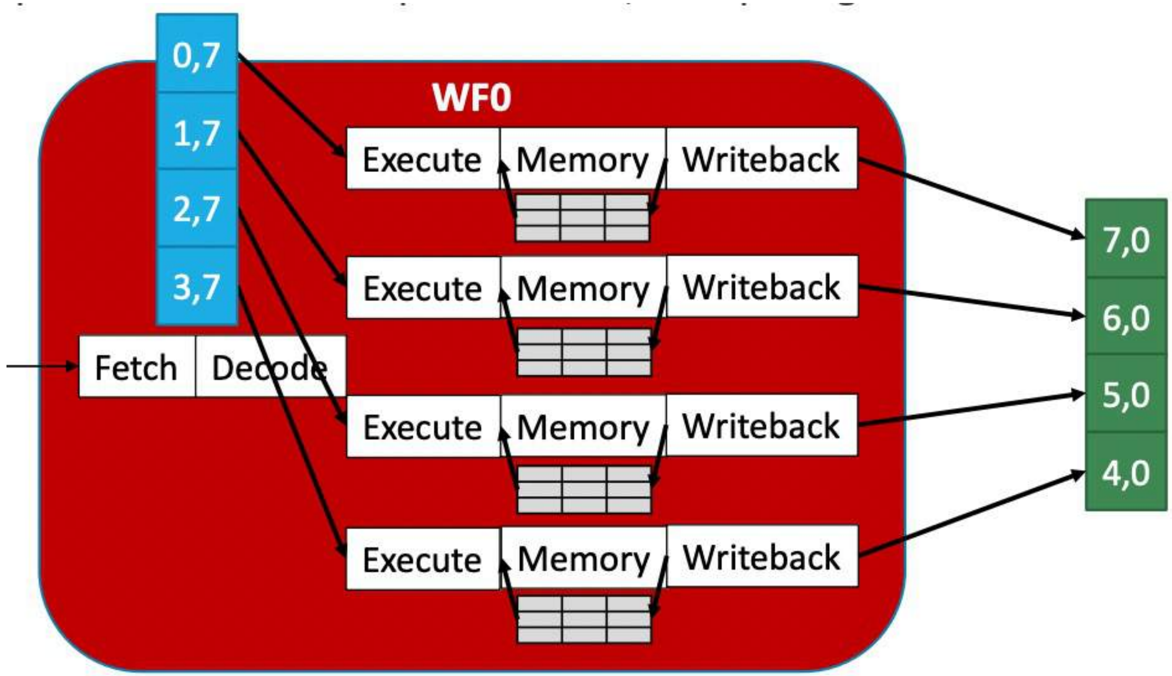
- Single Instruction Multiple Data
  - Split identical, independent work over multiple execution units (lanes)
  - More efficient: eliminate redundant fetch/decode
  - One Thread + Data Parallel Ops → Single PC, single register file



https://courses.cs.washington.edu/courses/cse471/13sp/lectures/GPUsStudents.pdf

Data Parallelism: SIMT

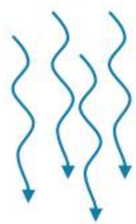
- Single Instruction Multiple Thread
  - Split identical, independent work over multiple threads
  - Multiple Threads + Scalar Ops → One PC, multiple register files
  - ≈ SIMD + multithreading
  - Each thread has its own registers



https://courses.cs.washington.edu/courses/cse471/13sp/lectures/GPUsStudents.pdf

Execution Model[执行模型]

MIMD



Multiple independent threads

Multicore CPUs

SIMD

One thread with wide execution datapath

x86 SSE/AVX

SIMT

Multiple lockstep threads

GPUs

- SI(MD/MT)
  - Broadcasting the same instruction to multiple execution units
  - Replicate the execution units, but they all share the same fetch/decode hardware

SIMD and SIMT are used interchangeably

https://courses.cs.washington.edu/courses/cse471/13sp/lectures/GPUsStudents.pdf

SIMD: GPU vs. CPU/Traditional

- Traditional SIMD contains a single thread
  - Programming model is SIMD (no threads)
  - SW needs to know vector length
  - ISA contains vector/SIMD instructions
- GPU SIMD consists of multiple scalar threads executing in a SIMD manner (i.e., same instruction executed by all threads)
  - Each thread can be treated individually (i.e., placed in a different warp) → programming model not SIMD
    - SW does not need to know vector length
    - Enables memory and branch latency tolerance
  - ISA is scalar → vector instructions formed dynamically
- Essentially, it is SPMD programming model implemented on SIMD hardware

https://course.ece.cmu.edu/~ece740/f13/lib/exe/fetch.php?media=onur-740-fall13-module5.1.3-simd-and-gpus-part3-vliw-dae-systolic.pdf

Example: add two vectors

C:

```
for(i=0;i<n;++i) a[i]=b[i]+c[i];
```

Matlab:

```
a=b+c;
```

SIMD:

```
void add(uint32_t *a, uint32_t *b, uint32_t *c, int n) {
    for(int i=0; i<n; i+=4) {
        //compute c[i], c[i+1], c[i+2], c[i+3]
        uint32x4_t a4 = vld1q_u32(a+i);
        uint32x4_t b4 = vld1q_u32(b+i);
        uint32x4_t c4 = vaddq_u32(a4,b4);
        vst1q_u32(c+i,c4);
    }
}
```

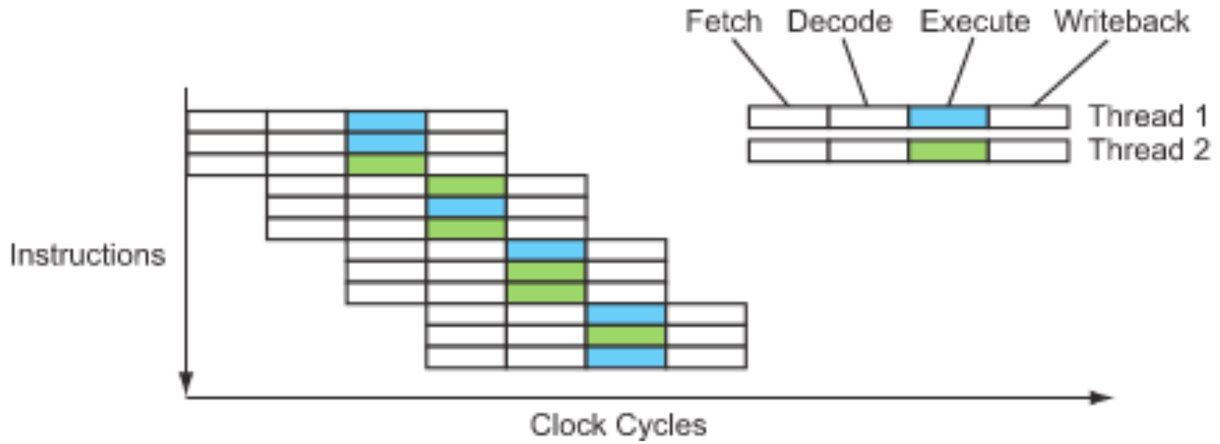
SIMT:

```
__global__ void add(float *a, float *b, float *c) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    a[i]=b[i]+c[i]; //no loop!
}
```

SUN YAT-SEN UNIVERSITY

SMT[多线程]

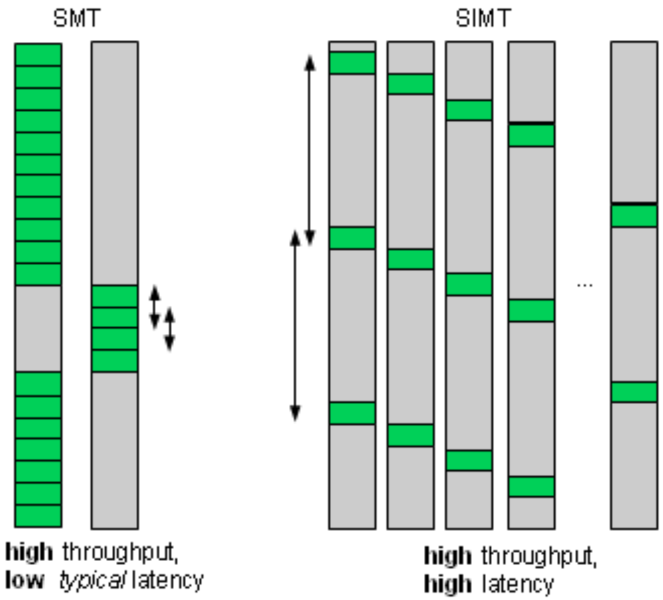
- SMT: simultaneous multithreading
  - Instructions from multiple threads issued on the same cycle
    - Use register renaming and dynamic scheduling facility of multi-issue architecture
  - Needs more hardware support
    - Register files, PC's for each thread
    - Support to sort out which threads to get results from which instructions
    - Thread scheduling, context switching
  - Maximize utilization of execution units



http://yosefk.com/blog/simd-simt-smt-parallelism-in-nvidia-gpus.html

SMT vs. SIMT[比较]

- SMT: maximize the chances of an instruction to be issued without having to switch to another thread
  - superscalar execution
  - out-of-order execution
  - register renaming
  - branch prediction
  - speculative execution
  - cache hierarchy
  - speculative prefetching

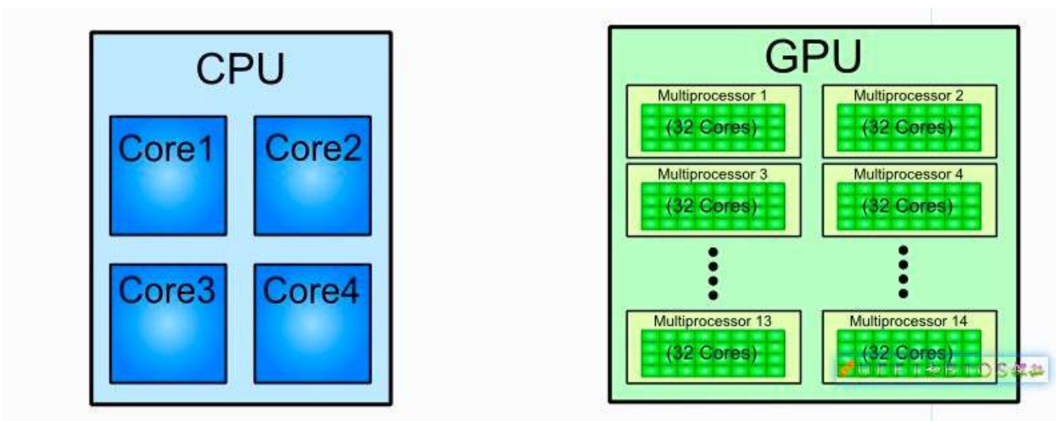


- SIMT: keep massive threads to achieve high throughput
  - Hardware becomes simpler and cheaper
  - No OoO, no prefetching, …

http://yosefk.com/blog/simd-simt-smt-parallelism-in-nvidia-gpus.html

CPU vs. GPU[比较]

- CPU
  - Low compute density
  - Complex control logic
  - Fewer cores optimized for serial operations
    - Fewer execution units (ALUs)
    - Higher clock speeds
  - Low latency tolerance
- GPU
  - High compute density
  - Simple control logic
  - 1000s cores optimized for parallel operations
    - Many parallel execution units (ALUs)
    - Lower clock speeds
  - High latency tolerance

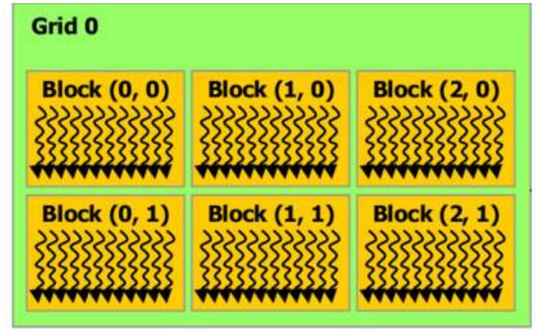


Part-II: GPU Overview

GPU Overview

Software

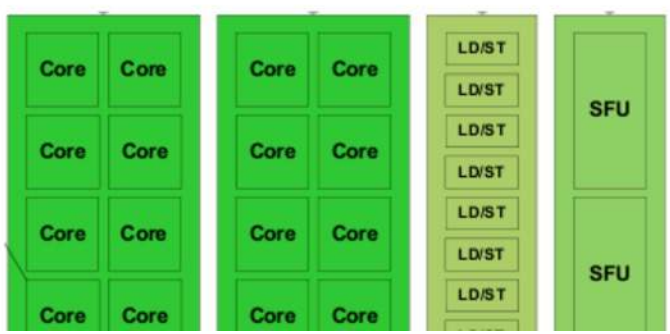
```
__global__ void scale(float a, float * X)
{
    unsigned int tid;
    tid = blockIdx.x * blockDim.x
          + threadIdx.x;
    X[tid] = a * X[tid];
}
```



Architecture: **multi-thread** programming model

SIMT microarchitecture

Hardware datapaths: **SIMD** execution units



Hardware

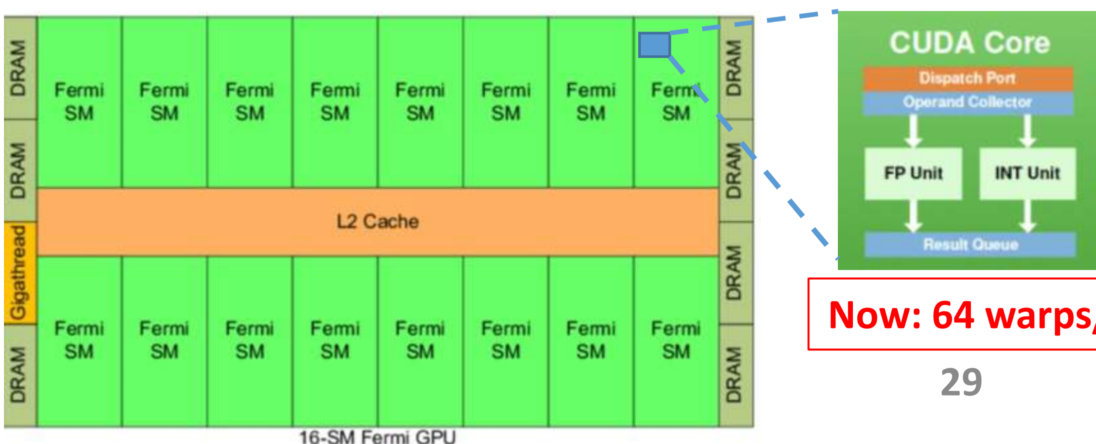
3

GPU Overview(cont.)

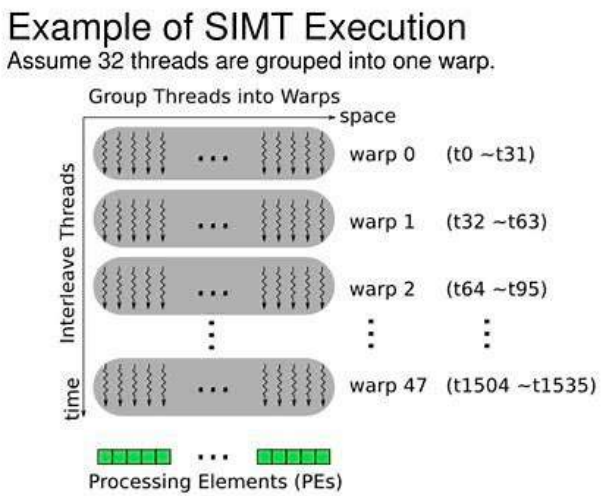
- A GPU contains several largely independent processors called "**Streaming Multiprocessors**" (SMs)
  - Each SM hosts multiple "cores", and each "core" runs a thread
  - For instance, Fermi(2010) has up to 16 SMs w/ 32 cores per SM
    - So up to 512 threads can run in parallel

A100: 128 SMs w/ 64 cores per SM
H100: 144 SMs w/ 128 cores per SM

- Some SIMT threads are grouped to execute in lockstep
  - One warp contains 32 threads
- Multiple 'groups' can be executed simultaneously
  - For Fermi, up to 48 warps per SM

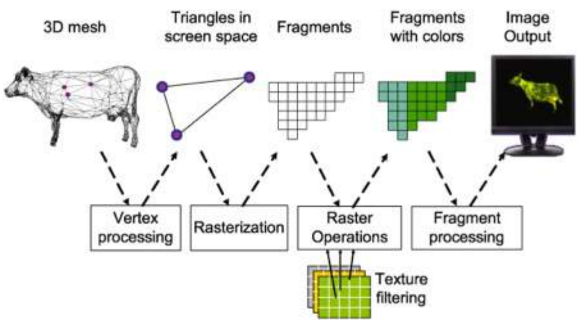


Now: 64 warps/SM

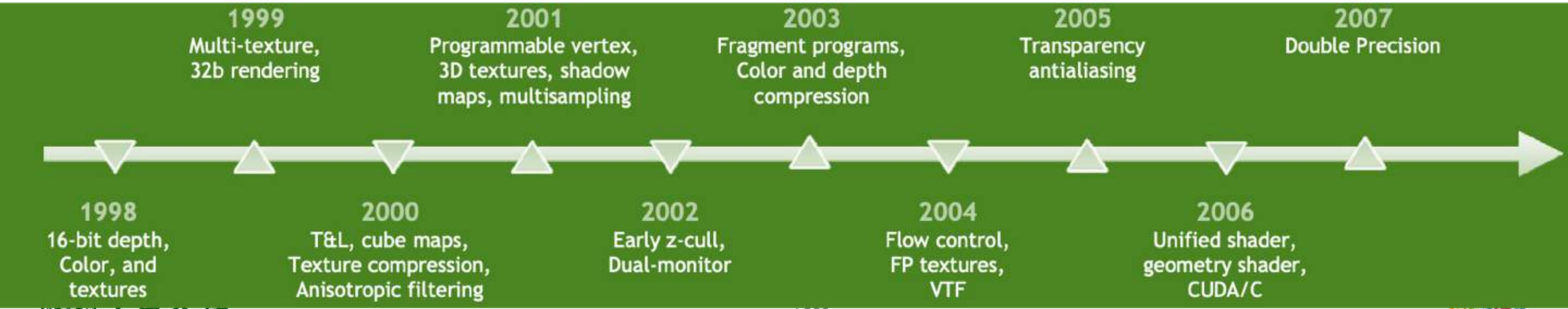


GPU Evolution[演进]

- Arcade boards and display adapters (1951 - 1995)
  - ATI: founded in 1985
  - Nvidia: founded in 1993
- 3D revolution (1995 - 2006)
  - Term “graphics processing unit”: 1999
    - Nvidia GeForce 256
  - Rivalry between ATI and Nvidia



- General purpose GPU (2006 - present)
  - AI, data analytics, scientific computing, graphics rendering, etc.

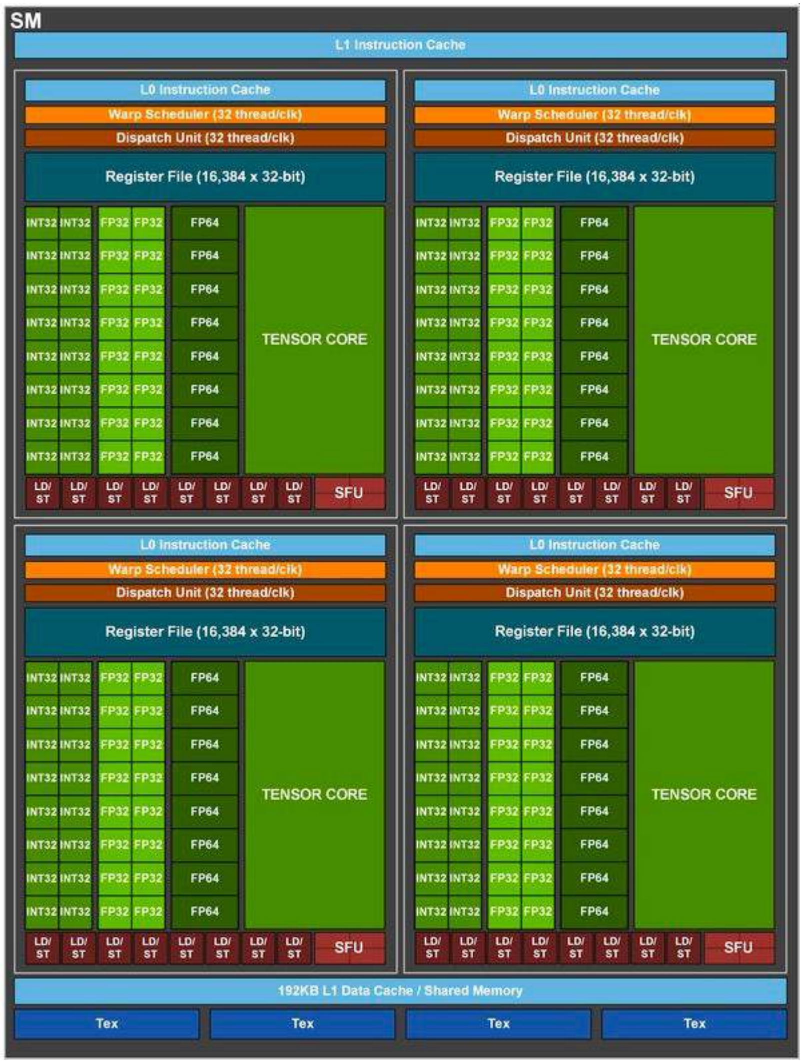


GPGPU History (2006 -)

Year	AMD	Nvidia	Note
2006	AMD acquired ATI	Tesla (CUDA Launch)	Unified shader model
2007	TeraScale		Unified shader uarch
2009	TeraScale 2		
2010	TeraScale 3	Fermi / GTX580	First compute GPU
2011	GCN 1.0 / gfx6		VLIW → SIMD
2012		Kepler / GTX680	CUDA cores: 512 → 1536
2013	GCN 2.0 / gfx7		
2014	GCN 3.0 / gfx8	Maxwell / GTX980	Energy efficiency
2016	GCN 4.0 / gfx8	Pascal / GTX1080 / P100	
2017	GCN 5.0 / gfx9	Volta / GV100	First chip with Tensor cores
2018	GCN 5.1 / gfx9	Turing / RTX2080	
2019	RDNA 1.0 / gfx10		
2020	RDNA 2.0 / gfx10 CDNA 1.0 / gfx9	Ampere / RTX3090 / A100	First chip with Matrix cores
2022	RDNA 3.0 / gfx10 CDNA 2.0 / gfx9	Hopper / RTX4090 / H100	Chiplet

TFLOPS[衡量算力]

- A100 Tensor Core GPU
  - 108 SMs
    - GA100 Full GPU with 128 SMs
  - Base clock: 1065 MHz
  - Boost clock: 1410 MHz
  - Performance
    - FP64: 9.7 TFLOPS
    - FP32: 19.5 TFLOPS

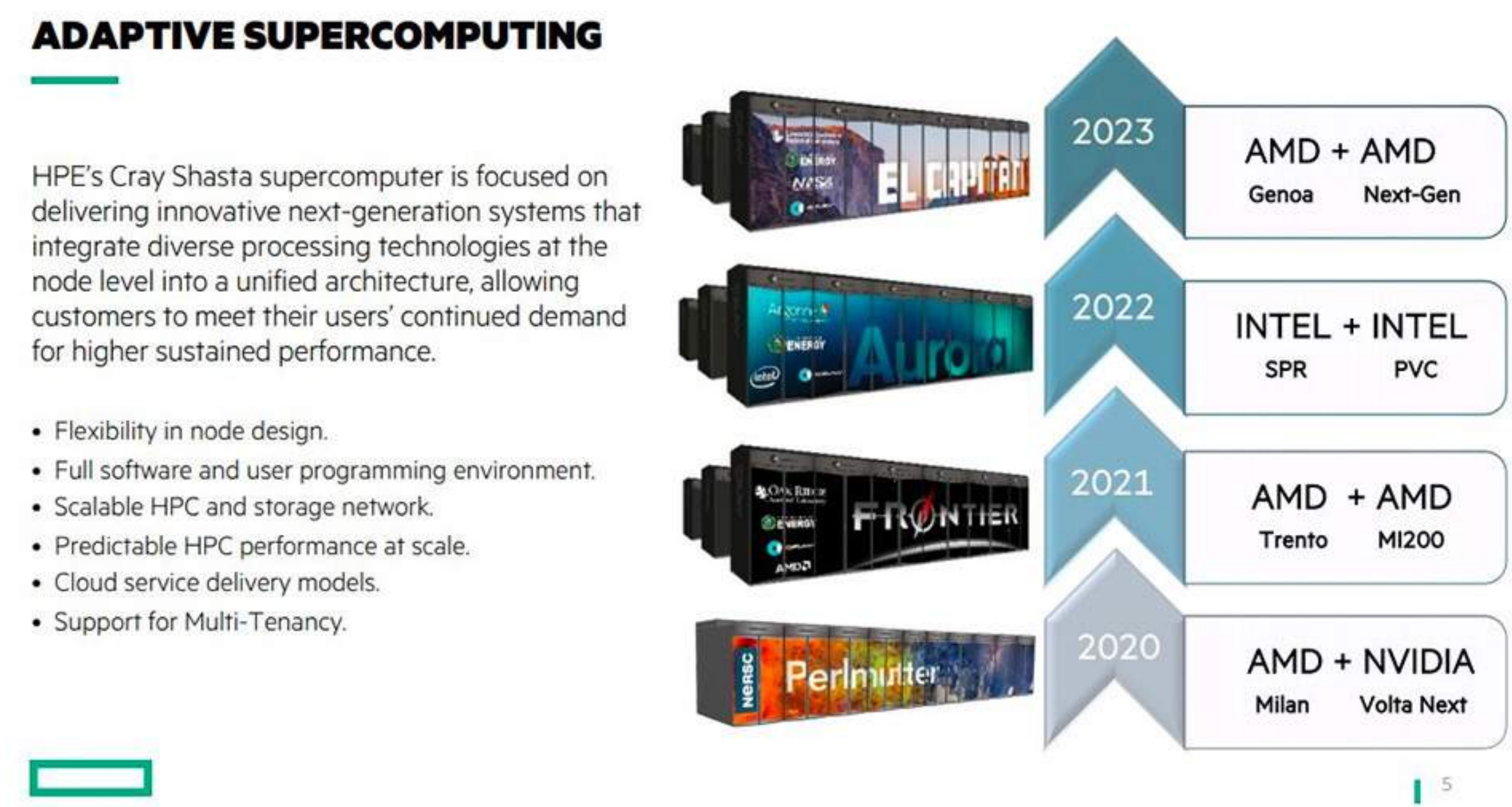


- Calculate TFLOPS
  - FP64: 1410 MHz x (32 x 2) ops/clock x 108 SMs

中山大學 SUN YAT-SEN UNIVERSITY

https://images.nvidia.cn/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

US Supercomputers[美国超算]



2.0E

1.0E

1.5E

90P

https://tekdeeps.com/the-amd-instinct-mi200-is-the-companys-first-multi-circuit-gpu/

https://www.nersc.gov/news-publications/nersc-news/science-news/2021/berkeley-lab-targets-exascale-with-perlmutter-and-nesap/

SC: Frontier

https://www.nextplatform.com/2021/10/26/china-has-already-reached-exascale-on-two-separate-systems/ (**1.3EFLOPS peak/1.05E sustained, 03/2021**)

- Frontier @ Oak Ridge National Laboratory*
  - 1.5 exaFLOPS, 29 MW, 2021
  - Compute node
    - 1 AMD EPYC CPU (Zen 3) + 4 AMD Radeon Instinct GPU (MI200)
  - Interconnect
    - Node: AMD **Infinity Fabric**, coherent memory across the node
    - System:
      - Multiple Slingshot NICs providing 100 GB/s network bandwidth.
      - Slingshot dragonfly network w/adaptive routing
  - Programming models:
    - AMD ROCm, MPI, OpenMP, HIP C/C++, Fortran
  - Applications: modeling and simulation, data analytics, AI

* https://www.olcf.ornl.gov/wp-content/uploads/2019/05/frontier_specsheet.pdf

SC: Aurora

Intel's Aurora Supercomputer Now Expected to Exceed 2 ExaFLOPS Performance

by Ryan Smith on October 27, 2021 3:25 PM EST

- Aurora @ Argonne National Laboratory*
  - 1 exaFLOPS, ≤ 60 MW, 2018-2021-2022
  - Compute node
    - 2 Intel Xeon Sapphire Rapids CPUs, 6 Xe GPUs
      - First enterprise CPUs to support CXL standard
      - GPUs communicates directly to each other via CXL
    - Unified memory architecture
  - Interconnect
    - CPU-GPU: PCIe, GPU-GPU: Xe Link
    - System: HPE Slingshot 11; Dragonfly topology with adaptive routing
  - Programming models
    - Intel oneAPI, MPI, OpenMP, C/C++, Fortran, SYCL/DPC++
  - Applications: climate change, cancer, new materials

* https://www.alcf.anl.gov/aurora

SC: El Capitan

- El Capitan @ Lawrence Livermore National Laboratory*
  - 2 exaFLOPS, 40 MW, early 2023
  - Compute node
    - 1 AMD EPYC Zen 4 CPU + 4 Radeon Instinct GPUs
  - Interconnect
    - Node: AMD Infinity Fabric, coherent memory across the node
    - System: Cray's own Slingshot interconnect
  - Programming models:
    - AMD ROCm, MPI, OpenMP, HIP C/C++, Fortran
  - Applications: nuclear weapon modeling
  - Misc: optical data transmission

* https://www.llnl.gov/news/llnl-and-hpe-partner-amd-el-capitan-projected-worlds-fastest-supercomputer

A Comparison[对比]

- 50 GFLOPS/Watt (goal) → **51.7** GFLOPS/Watt

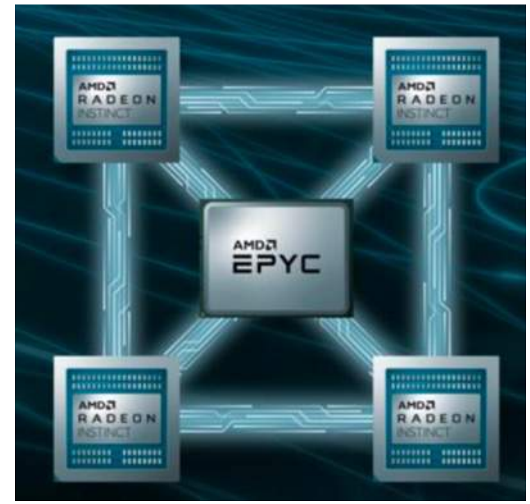
System	Titan (2012)	Summit (2017)	Frontier (2021)
Peak	27 PF	200 PF	> 1.5 EF
# nodes	18,688	4,608	> 9,000
Node	1 AMD Opteron CPU 1 NVIDIA Kepler GPU	2 IBM POWER9™ CPUs 6 NVIDIA Volta GPUs	1 AMD EPYC CPU 4 AMD Radeon Instinct GPUs
Memory		2.4 PB DDR4 + 0.4 HBM + 7.4 PB On-node storage	4.6 PB DDR4 + 4.6 PB HBM2e + 36 PB On-node storage, 75 TB/s Read 38 Write
On-node interconnect	PCI Gen2 No coherence across the node	NVIDIA NVLINK Coherent memory across the node	AMD Infinity Fabric Coherent memory across the node
System Interconnect	Cray Gemini network 6.4 GB/s	Mellanox Dual-port EDR IB 25 GB/s	Four-port Slingshot network 100 GB/s
Topology	3D Torus	Non-blocking Fat Tree	Dragonfly
Storage	32 PB, 1 TB/s, Lustre Filesystem	250 PB, 2.5 TB/s, IBM Spectrum Scale™ with GPFS™	695 PB HDD+11 PB Flash Performance Tier, 9.4 TB/s and 10 PB Metadata Flash. Lustre
Power	9 MW	13 MW	29 MW

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

https://www.hpcwire.com/2021/07/14/frontier-to-meet-20mw-exascale-power-target-set-by-darpa-in-2008/

Frontier: 1.5 EFLOPS, How???

- Per node[单节点]
  - Custom EPYC HPC-optimized CPU
    - ”zen 3” milan w/ 64-core
  - Four Instinct GPUs
    - CDNA MI200 w/ *256* CUs
      - Full-rate FP64 (128 ops/clock/CU)



- 9000+ nodes[整体系统]
  - CPU: 9000 x 4 TFLOPS/CPU = 36 PFLOPS
  - GPU: 9000 x 4 x 42.2 TFLOPS/GPU = 1519 PFLOPS
    - Per GPU: 128 ops/clock x 1.5G x 220 = 42.2 TFLOPS
  - GPU provides **97.7%** computation power
    - 1519/(1519+36)

A100: 9.75 TFLOPS
MI100: 11.54 TFLOPS

OLCF spock training: AMD hardware and software, 05/2021, https://www.olcf.ornl.gov/wp-content/uploads/2021/04/Spock-MI100-Update-5.20.21.pdf

https://www.hpcwire.com/2021/03/15/amd-launches-epyc-milan-with-19-skus-for-hpc-enterprise-and-hyperscale/

Confirm[确认]

- Setonix: 50 PFLOPS ($48M)
  - 200K+ AMD Milan CPU cores
    - 64 cores/CPU → 3125 CPUs (4 TFLOPS)
    - 3125 x 4 = 12 PFLOPS
  - 750+ MI200 GPUs
    - 42.2 TFLOPS/GPU x 750 = 32 PFLOPS

- Frontier: 1.5 EFLOPS ($600M)
  - 9000+ nodes x (1 CPU + 4 GPU)
    - 9000 CPUs (3x): 9000x4 = 36 PFLOPS
    - 36K MI200 GPUs (48x): 36K x 42.2 = 1519 PFLOPS

GPUs are effective to increase computation: 12.5x cost → 30x FLOPS

https://insidehpc.com/2021/09/pawsey-offers-1st-look-at-setonix-48m-hpe-cray-ex-supercomputer/
https://www.hpcwire.com/2021/03/15/amd-launches-epyc-milan-with-19-skus-for-hpc-enterprise-and-hyperscale/

MI200 GPU

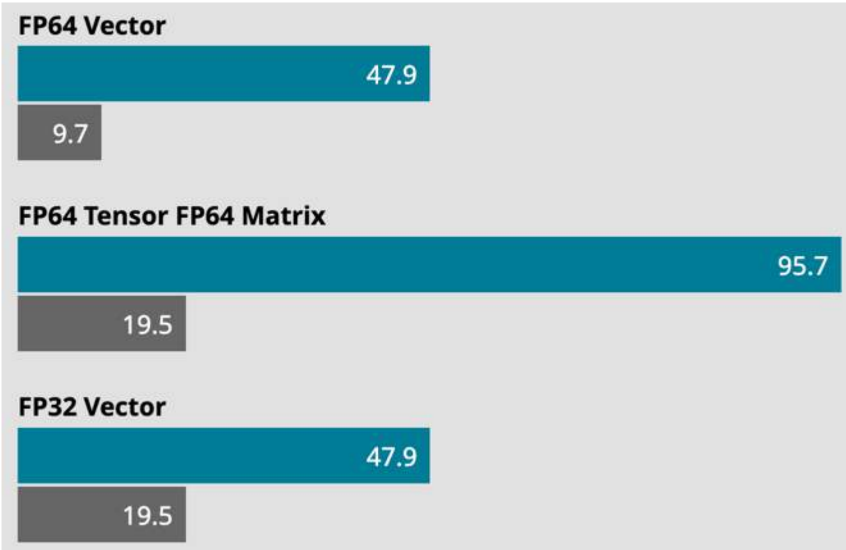
- Architecture: CDNA2
- Codename: Aldebaran
- ID number: gfx90A
  - MI100: gfx908 (CDNA1)
  - Vega20: gfx906 (GCN5)
  - Navi14: gfx1012 (RDNA1)

- Multi-chip module (MCM, or chiplet)
  - Two logic dies, eight HBM2e stacks
  - Each of the die has a 4096-bit HBM2e interface, w/ 64GB mem
    - 4x 1024-bit 16GB
  - Each of the die has 8 SEs that have 16 CUs (total: 256CUs)

A100: 40G/80G
MI100: 32G

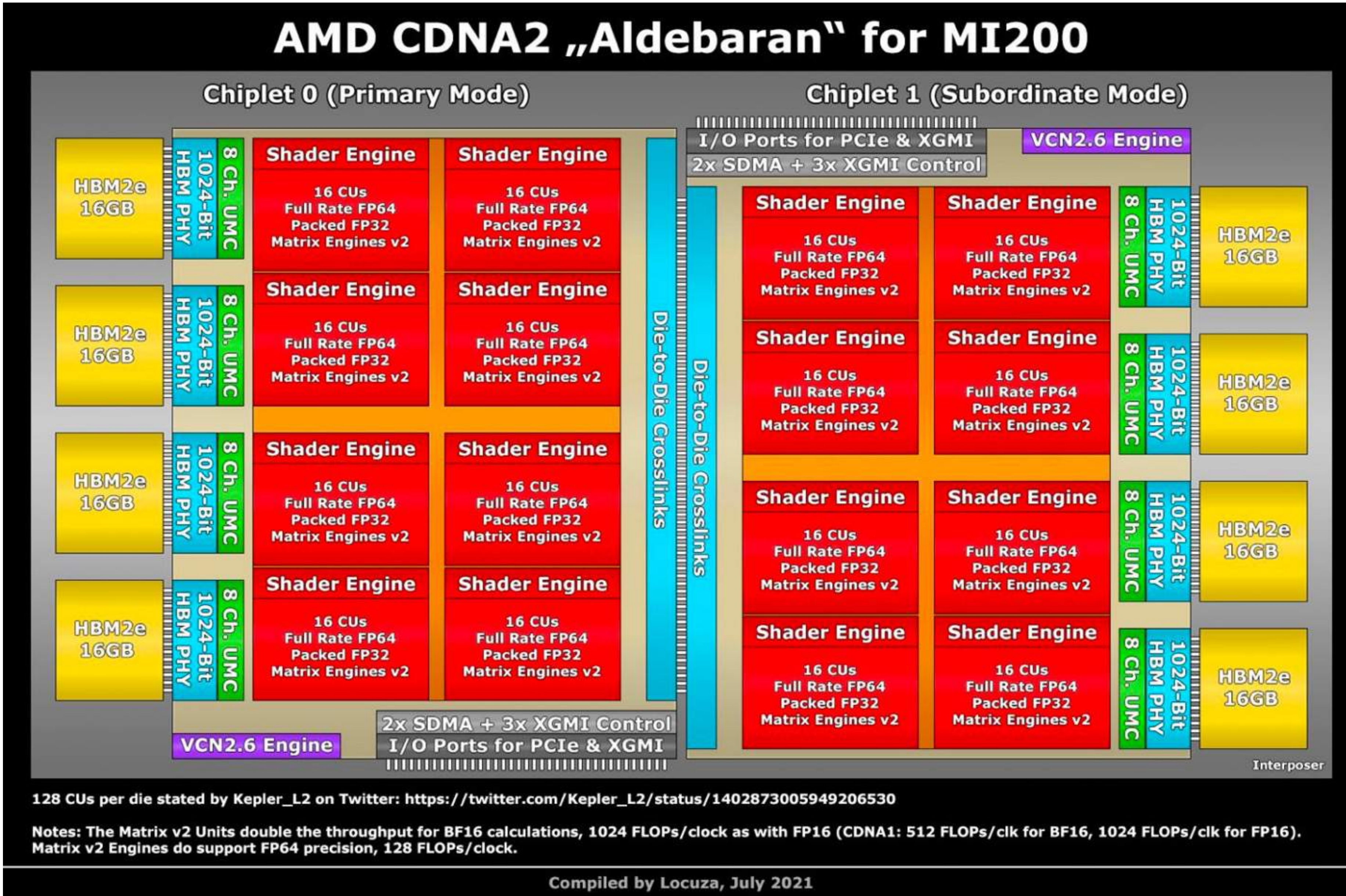
https://wccftech.com/amd-instinct-mi200-aldebaran-gpus-with-128-gb-memory-epyc-milan-cpus-power-setonix-supercomputer/
https://www.techpowerup.com/284035/amd-cdna2-aldebaran-mi200-hpc-accelerator-with-256-cu-16-384-cores-imagined
https://www.notebookcheck.net/Leaked-diagram-reveals-the-AMD-Instinct-MI200-compute-GPU-with-dual-128-CU-dies-and-128-GB-HBM2e-RAM.549102.0.html

MI200 GPU



https://www.amd.com/en/press-releases/2022-05-26-amd-instinct-mi200-adopted-for-large-scale-ai-training-microsoft-azure

MI200 (cont'd)


AMD CDNA2 „Aldebaran“ for MI200
Chiplet 0 (Primary Mode)
Chiplet 1 (Subordinate Mode)
HBM2e 16GB
1024-Bit HBM PHY
8 Ch. UMC
Shader Engine
16 CUs
Full Rate FP64
Packed FP32
Matrix Engines v2
Die-to-Die Crosslinks
2x SDMA + 3x XGMI Control
I/O Ports for PCIe & XGMI
VCN2.6 Engine
Interposer
128 CUs per die stated by Kepler_L2 on Twitter: https://twitter.com/Kepler_L2/status/1402873005949206530
Notes: The Matrix v2 Units double the throughput for BF16 calculations, 1024 FLOPs/clock as with FP16 (CDNA1: 512 FLOPs/clk for BF16, 1024 FLOPs/clk for FP16). Matrix v2 Engines do support FP64 precision, 128 FLOPs/clock.
Compiled by Locuza, July 2021

天河超算

- 2009，天河-1
  - CPU + ATI GPU
    - 2 * Xeon E5540/E5450, 1 ATI Radeon HD 4870 X2 (TeraScale)
  - 实测/峰值563.1T/1206.2T FLOPS
  - 2009.11 TOP500第五

240 GFLOPS

- 2010，天河-1A
  - CPU + Nvidia GPU
    - 2 * Intel Xeon X5670, 1 Nvidia Tesla M2050 (Fermi)
    - 2048 Galaxy "FT-1000" 1 GHz 8-core processors
  - 实测/峰值2.566P/4.7P FLOPS
  - 2010.11 TOP500第一

515 GFLOPS

Tianhe-1, https://www.top500.org/system/176546/
Tianhe-1A, https://top500.org/system/176929/
Tianhe-1A, http://blog.zorinaq.com/introducing-tianhe-1a-4702-tflops-of-gpu-power-made-in-china-and/