北京大学
PEKING UNIVERSITY

22530007

人工智能与芯片设计

4-Graphics Processing Unit

燕博南

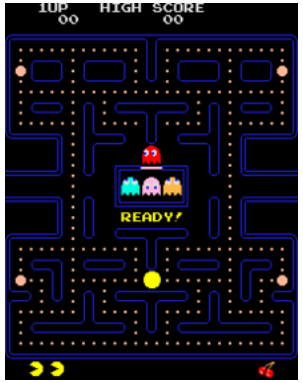
2023秋

Outline

- History of GPU
- GPU Core Idea
- GPU hardware

History of GPU

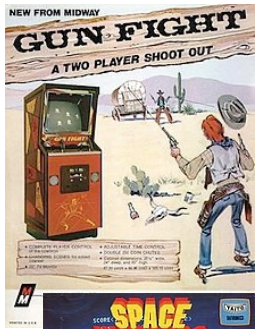
namco®

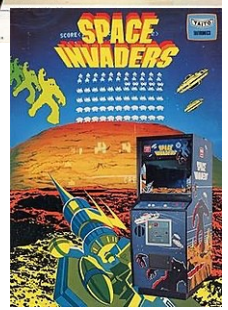


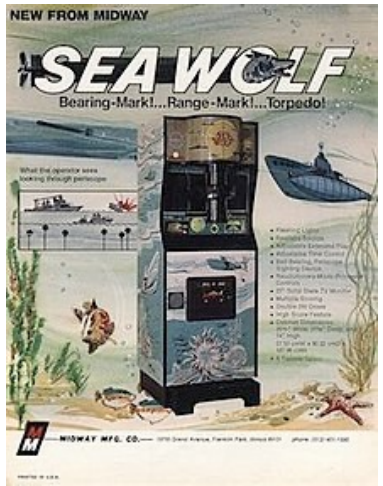
Framebuffer Loading

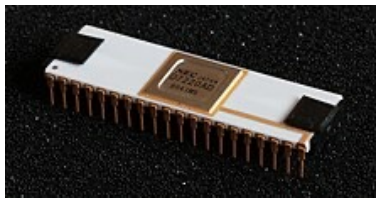
1970s

1980s





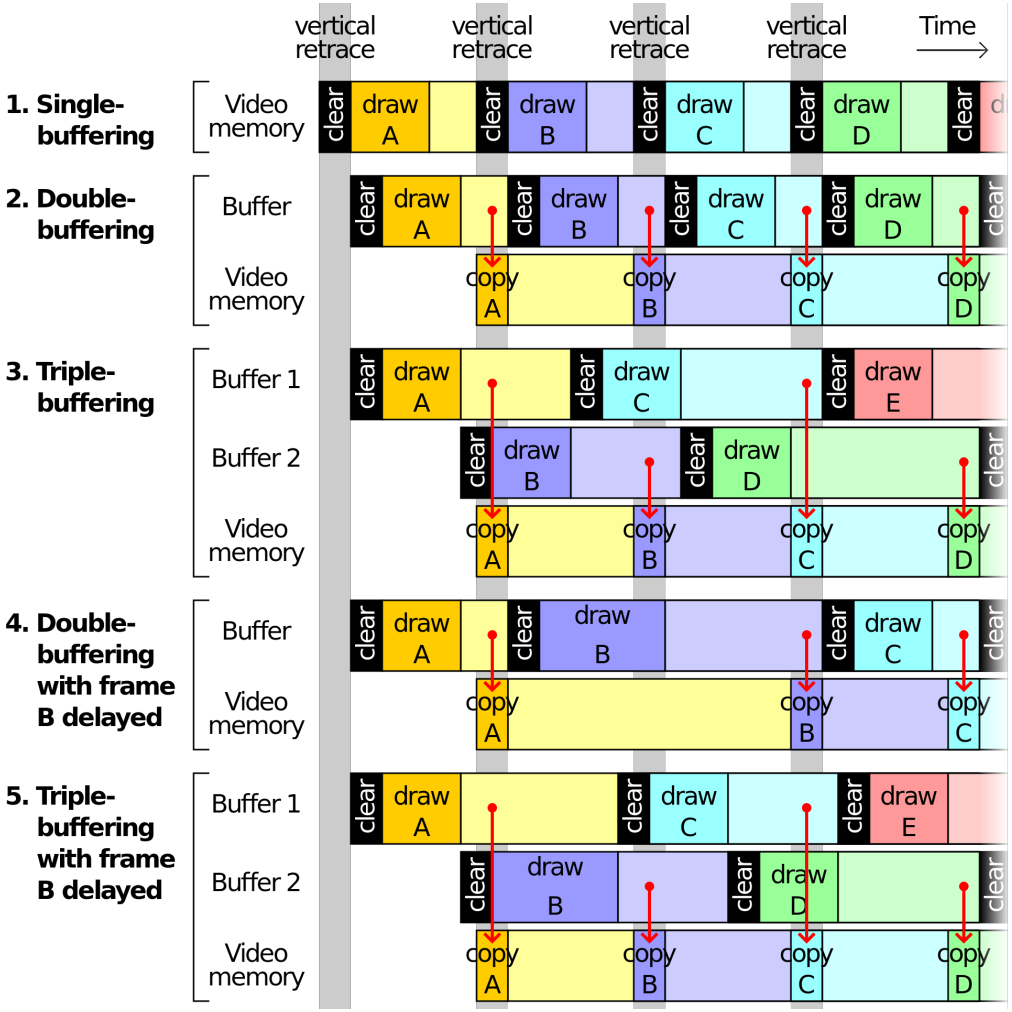


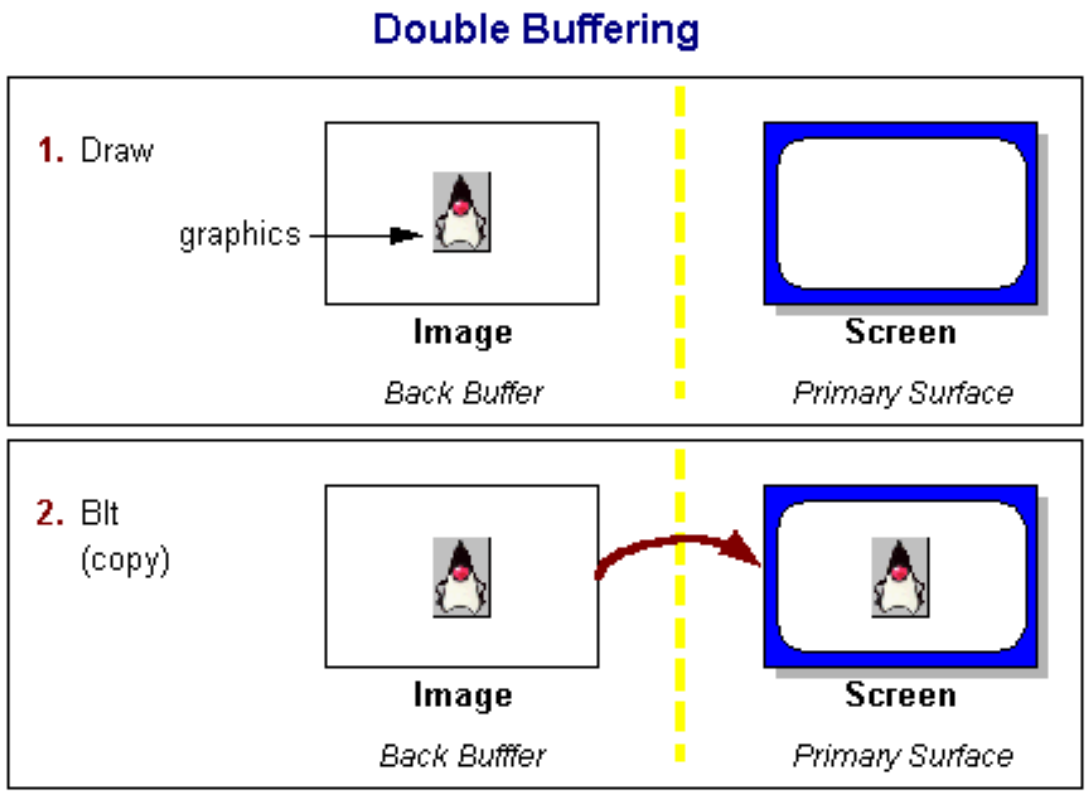


Support 1024*1024 resolution

Color display processor

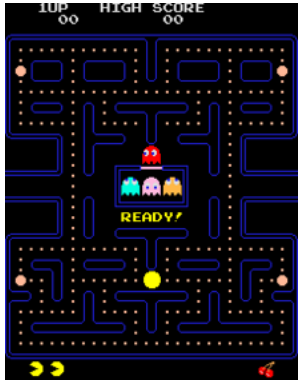
Framebuffer





History of GPU

namco®

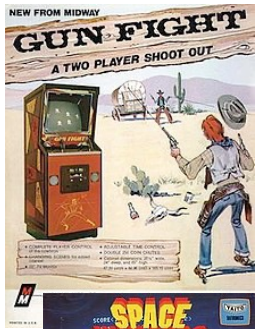


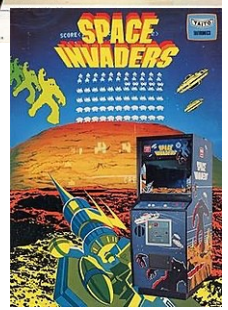
Framebuffer Loading

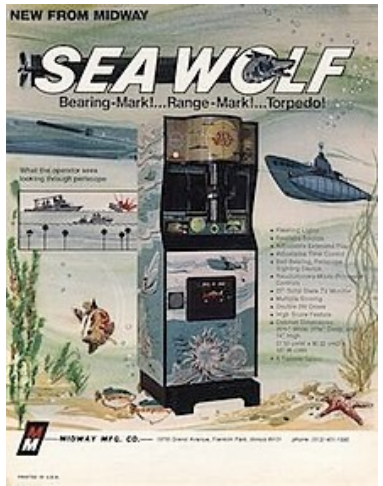
NVIDIA GeForce 256 (1999)

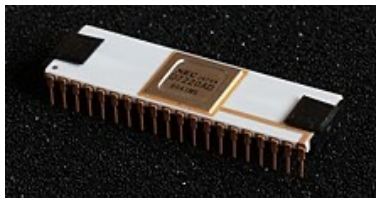
- ability to write to arbitrary memory addresses from a shader
- scratchpad memory to limit off-chip bandwidth

1970s 1980s 1990s 2000s









Support 1024*1024 resolution
Color display processor

TSMC 220nm process
Function:
Transform, clipping, and lighting

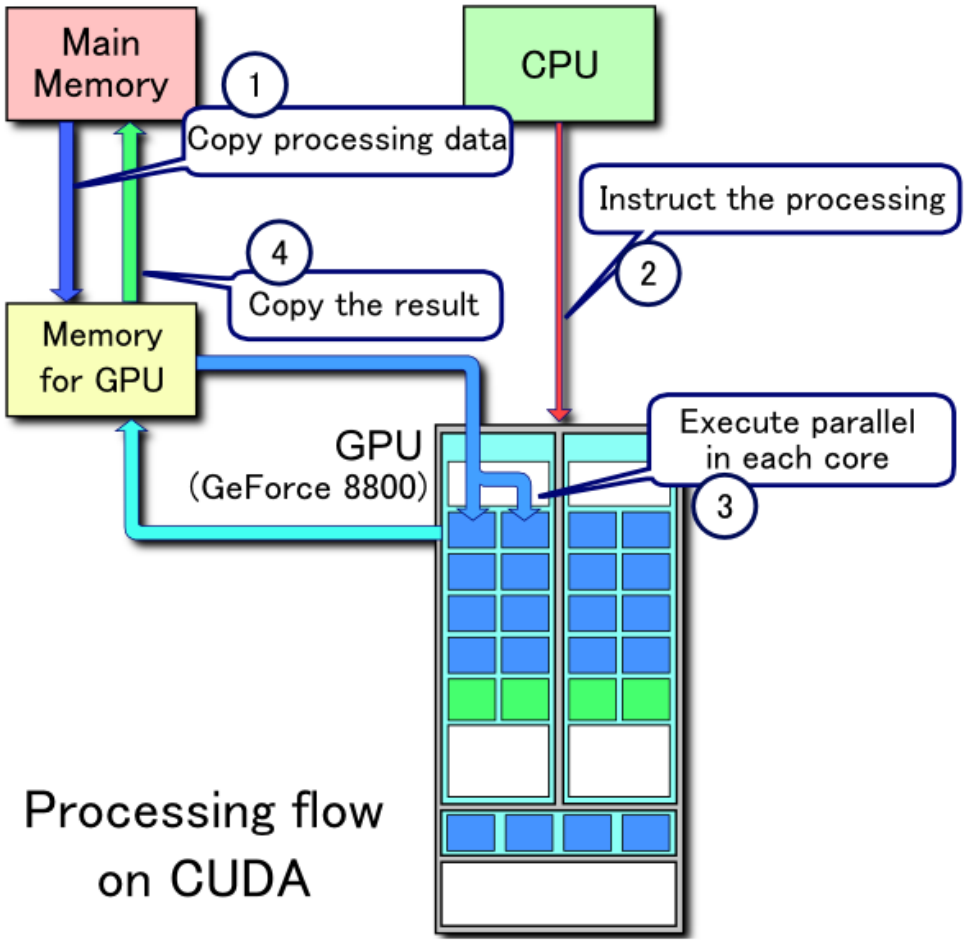
NVIDIA GeForce 8 Series

Linear Algebra Computation

GPU- From Graphics to Parallel Computation

CUDA 8.0 comes with the following libraries (for compilation & runtime, in alphabetical order):

- cuBLAS – CUDA Basic Linear Algebra Subroutines library
- CUDART – CUDA Runtime library
- cuFFT – CUDA Fast Fourier Transform library
- cuRAND – CUDA Random Number Generation library
- cuSOLVER – CUDA based collection of dense and sparse direct solvers
- cuSPARSE – CUDA Sparse Matrix library
- NPP – NVIDIA Performance Primitives library
- nvGRAPH – NVIDIA Graph Analytics library
- NVML – NVIDIA Management Library
- NVRTC – NVIDIA Runtime Compilation library for CUDA C++



GPU Program Model

```
1  void saxpy_serial(int n, float a, float *x, float *y)
2  {
3    for (int i = 0; i < n; ++i)
4      y[i] = a*x[i] + y[i];
5  }
6  main() {
7    float *x, *y;
8    int n;
9    // omitted: allocate CPU memory for x
10   saxpy_serial(n, 2.0, x, y); // Invoke
11   // omitted: use y on CPU, free memory
12 }
```

CPU code

GPU code

```
1  __global__ void saxpy(int n, float a, float *x, float *y)
2  {
3     int i = blockIdx.x*blockDim.x + threadIdx.x;
4     if(i<n)
5       y[i] = a*x[i] + y[i];
6  }
7  int main() {
8    float *h_x, *h_y;
9    int n;
10   // omitted: allocate CPU memory for h_x and h_y and initialize contents
11   float *d_x, *d_y;
12   int nblocks = (n + 255) / 256;
13   cudaMalloc( &d_x, n * sizeof(float) );
14   cudaMalloc( &d_y, n * sizeof(float) );
15   cudaMemcpy( d_x, h_x, n * sizeof(float), cudaMemcpyHostToDevice );
16   cudaMemcpy( d_y, h_y, n * sizeof(float), cudaMemcpyHostToDevice );
17   saxpy<<<nblocks, 256>>>(n, 2.0, d_x, d_y);
18   cudaMemcpy( h_x, d_x, n * sizeof(float), cudaMemcpyDeviceToHost );
19   // omitted: use h_y on CPU, free memory pointed to by h_x, h_y, d_x, and d_y
20 }
```

- The threads that make up a compute kernel are organized into a hierarchy composed of *a grid of thread blocks* consisting of *warps*
- NVIDIA warps consists of 32 threads

GPU ISA

```
.visible .entry _Z5saxpyifPfS_(
.param .u32 _Z5saxpyifPfS__param_0,
.param .f32 _Z5saxpyifPfS__param_1,
.param .u64 _Z5saxpyifPfS__param_2,
.param .u64 _Z5saxpyifPfS__param_3
)
{
.reg .pred %p<2>;
.reg .f32 %f<5>;
.reg .b32 %r<6>;
.reg .b64 %rd<8>;


ld.param.u32 %r2, [_Z5saxpyifPfS__param_0];
ld.param.f32 %f1, [_Z5saxpyifPfS__param_1];
ld.param.u64 %rd1, [_Z5saxpyifPfS__param_2];
ld.param.u64 %rd2, [_Z5saxpyifPfS__param_3];
mov.u32 %r3, %ctaid.x;
mov.u32 %r4, %ntid.x;
mov.u32 %r5, %tid.x;
mad.lo.s32 %r1, %r4, %r3, %r5;
setp.ge.s32 %p1, %r1, %r2;
@%p1 bra BB0_2;

cvta.to.global.u64 %rd3, %rd2;
cvta.to.global.u64 %rd4, %rd1;
mul.wide.s32 %rd5, %r1, 4;
add.s64 %rd6, %rd4, %rd5;
ld.global.f32 %f2, [%rd6];
add.s64 %rd7, %rd3, %rd5;
ld.global.f32 %f3, [%rd7];
fma.rn.f32 %f4, %f2, %f1, %f3;
st.global.f32 [%rd7], %f4;

BB0_2:
ret;
}
```

- **Parallel Thread Execution ISA, or PTX**
- **Similar to RISC, ARM**

```
Address     Dissassembly                                          Encoded Instruction
========    ================================================      ======================
/*0000*/         MOV R1, c[0x1][0x100];                          /* 0x2800440400005de4 */
/*0008*/         S2R R0, SR_CTAID.X;                             /* 0x2c00000094001c04 */
/*0010*/         S2R R2, SR_TID.X;                               /* 0x2c00000084009c04 */
/*0018*/         IMAD R0, R0, c[0x0][0x8], R2;                   /* 0x2004400020001ca3 */
/*0020*/         ISETP.GE.AND P0, PT, R0, c[0x0][0x20], PT;      /* 0x1b0e40008001dc23 */
/*0028*/     @P0 BRA.U 0x78;                                     /* 0x40000001200081e7 */
/*0030*/    @!P0 MOV32I R5, 0x4;                                 /* 0x18000000100161e2 */
/*0038*/    @!P0 IMAD R2.CC, R0, R5, c[0x0][0x28];               /* 0x200b8000a000a0a3 */
/*0040*/    @!P0 IMAD.HI.X R3, R0, R5, c[0x0][0x2c];             /* 0x208a8000b000e0e3 */
/*0048*/    @!P0 IMAD R4.CC, R0, R5, c[0x0][0x30];               /* 0x200b8000c00120a3 */
/*0050*/    @!P0 LD.E R2, [R2];                                  /* 0x840000000020a085 */
/*0058*/    @!P0 IMAD.HI.X R5, R0, R5, c[0x0][0x34];             /* 0x208a8000d00160e3 */
/*0060*/    @!P0 LD.E R0, [R4];                                  /* 0x8400000000402085 */
/*0068*/    @!P0 FFMA R0, R2, c[0x0][0x24], R0;                  /* 0x3000400090202000 */
/*0070*/    @!P0 ST.E [R4], R0;                                  /* 0x9400000000402085 */
/*0078*/         EXIT;                                           /* 0x8000000000001de7 */
```

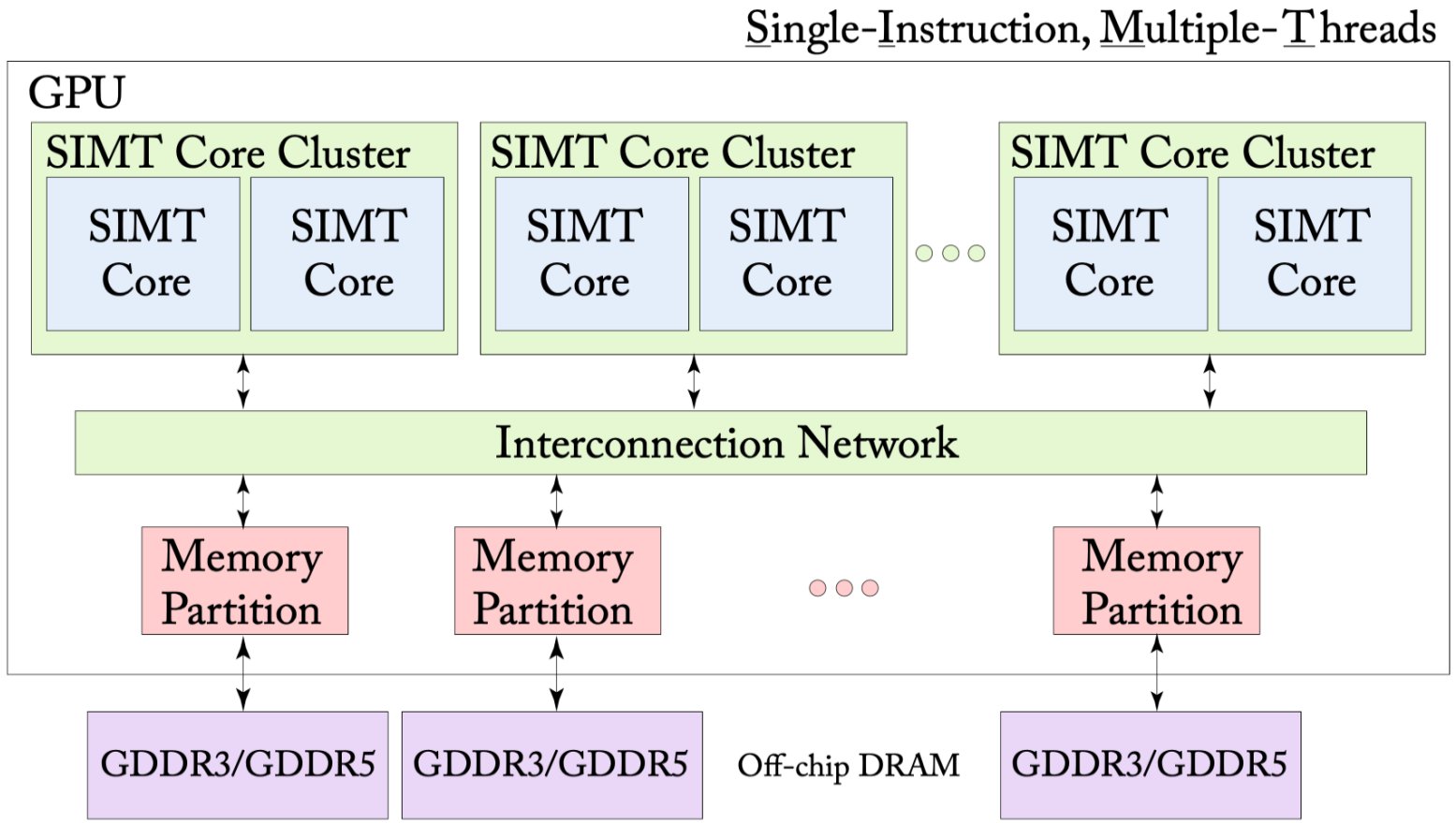
compiled with CUDA 8.0 for the NVIDIA Fermi Architecture, sm_20

Process & Thread

- **A thread includes the program counter, the register state, and the stack. It is a lightweight process; whereas threads commonly share a single address space, processes don't.**

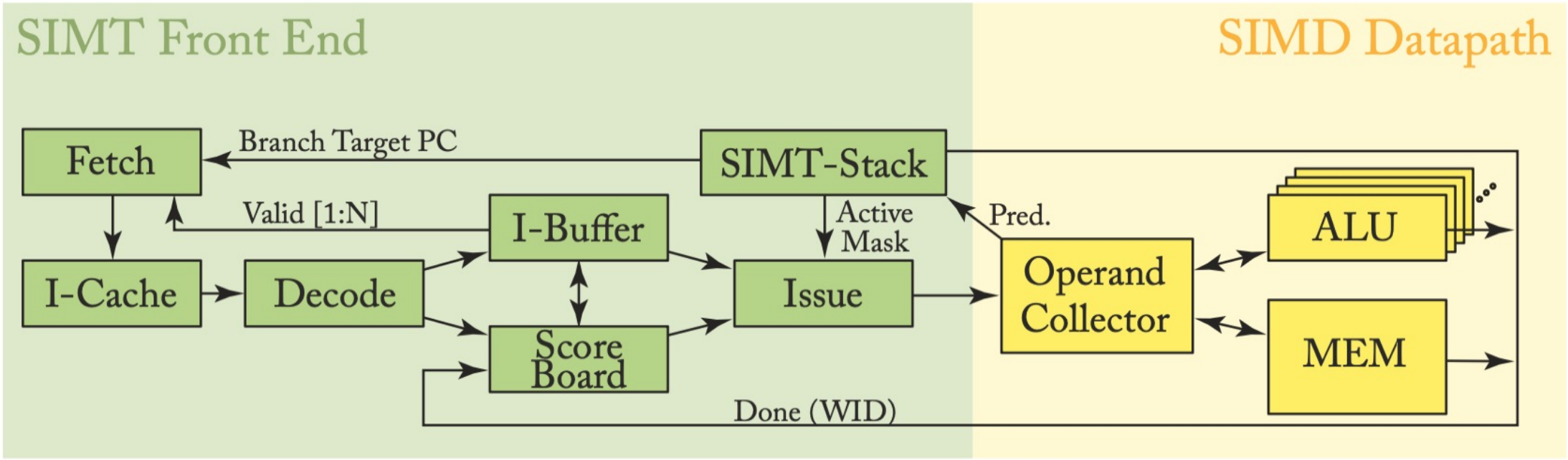
- **A process includes one or more threads, the address space, and the operating system state. Hence, a process switch usually invokes the operating system, but not a thread switch.**

GPU Hardware Architecture



SIMT & SIMD

- **SIMT: single-instruction multiple-thread**
- **SIMD: single-instruction multiple-data**



GPU Warps

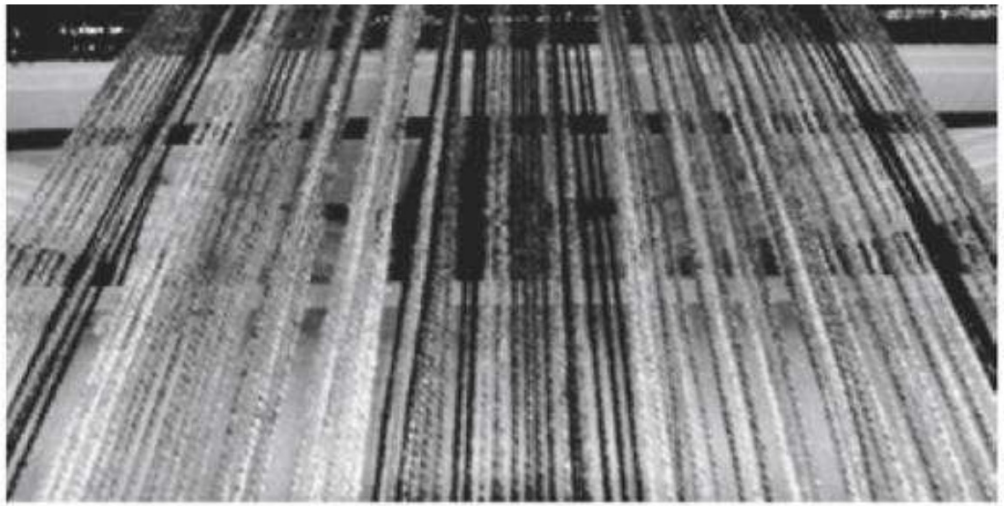
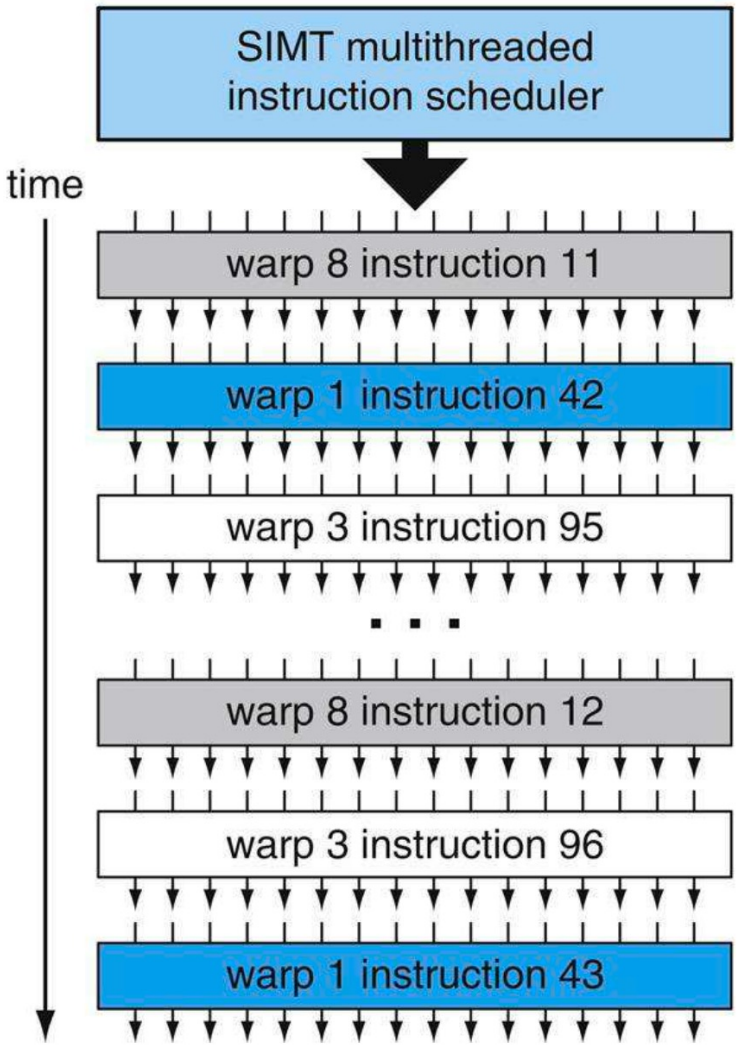
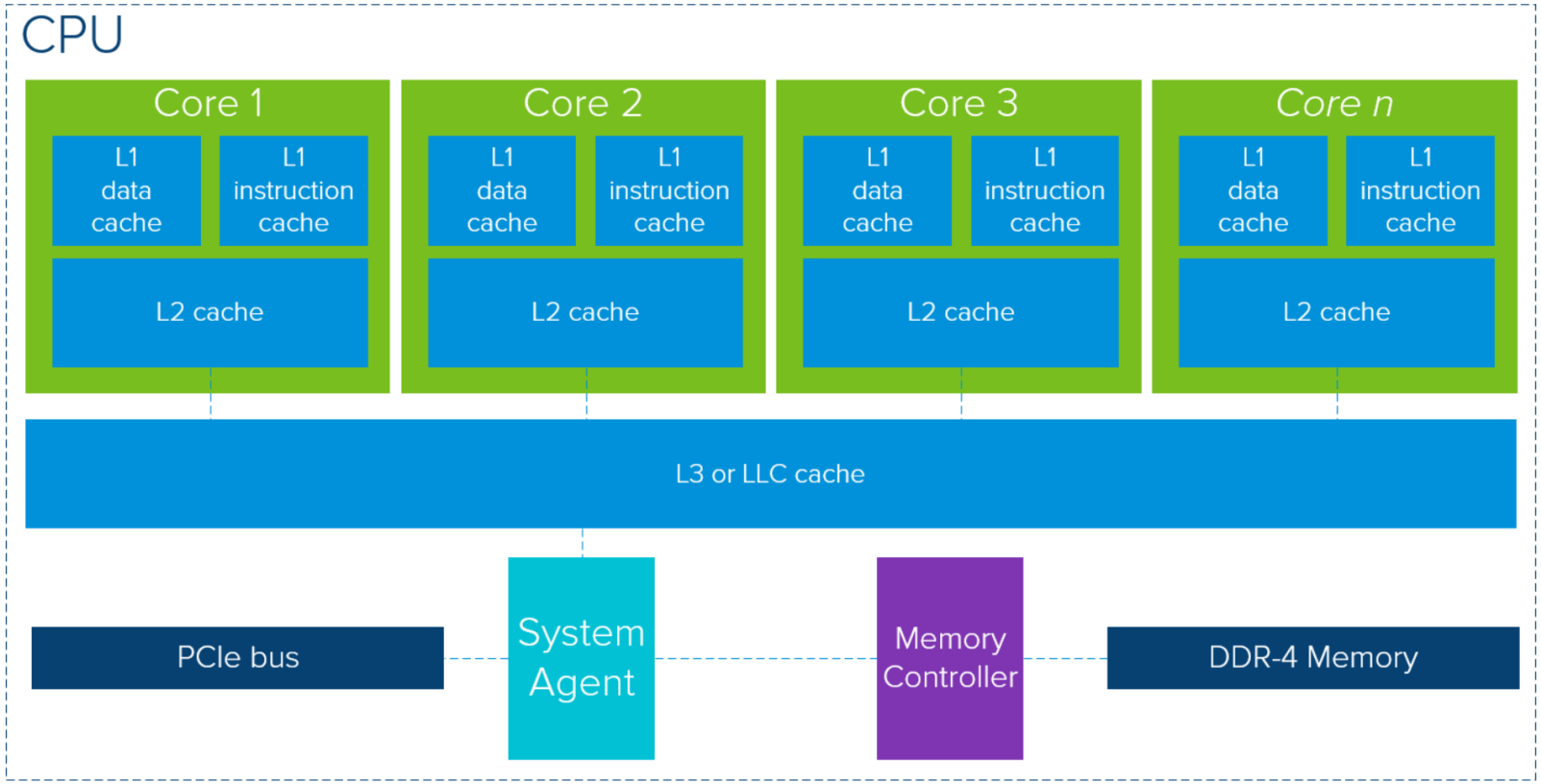


Photo: Judy Schoonmaker



From CPU to GPU

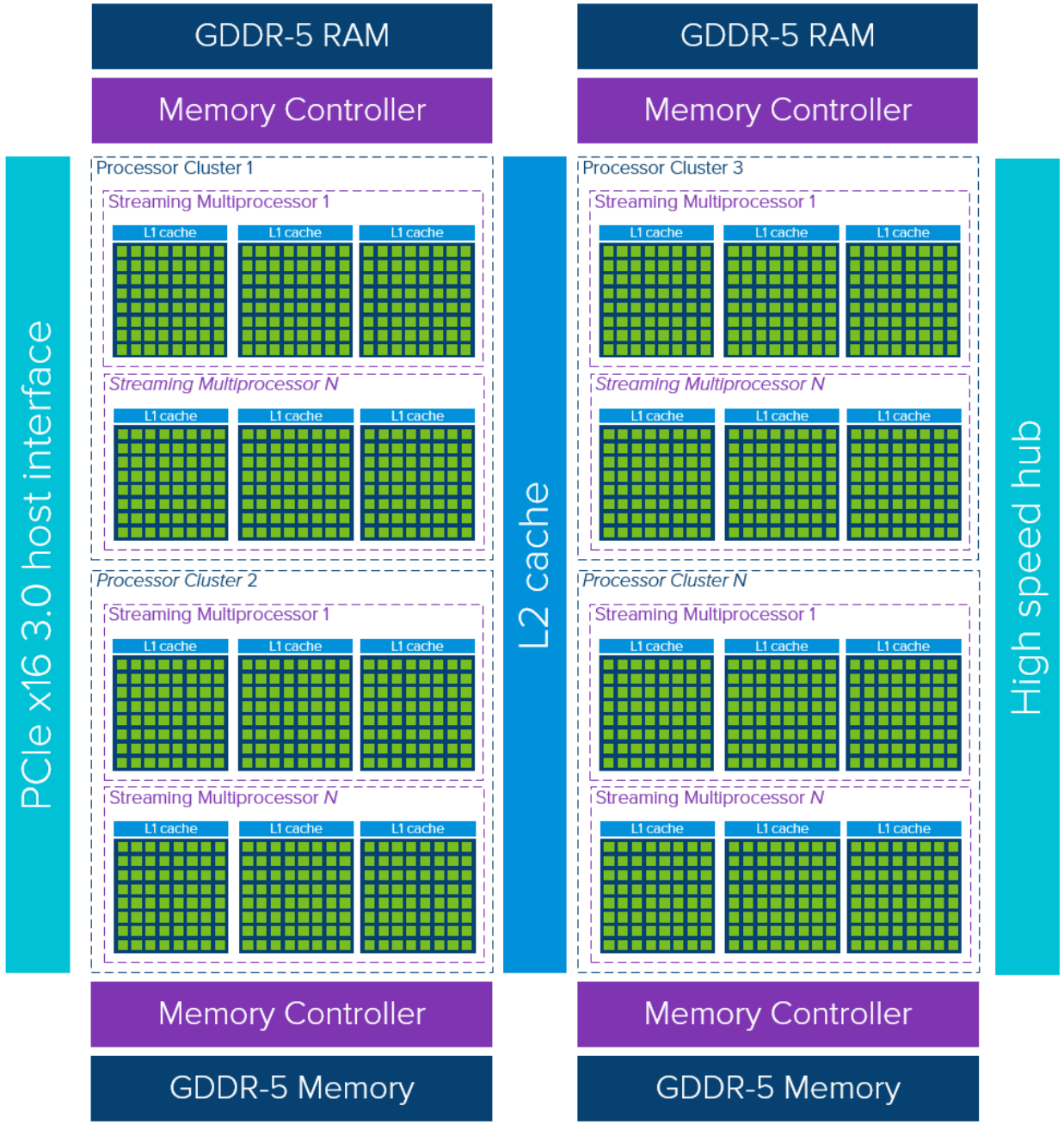
- **N-core CPU hardware architecture**



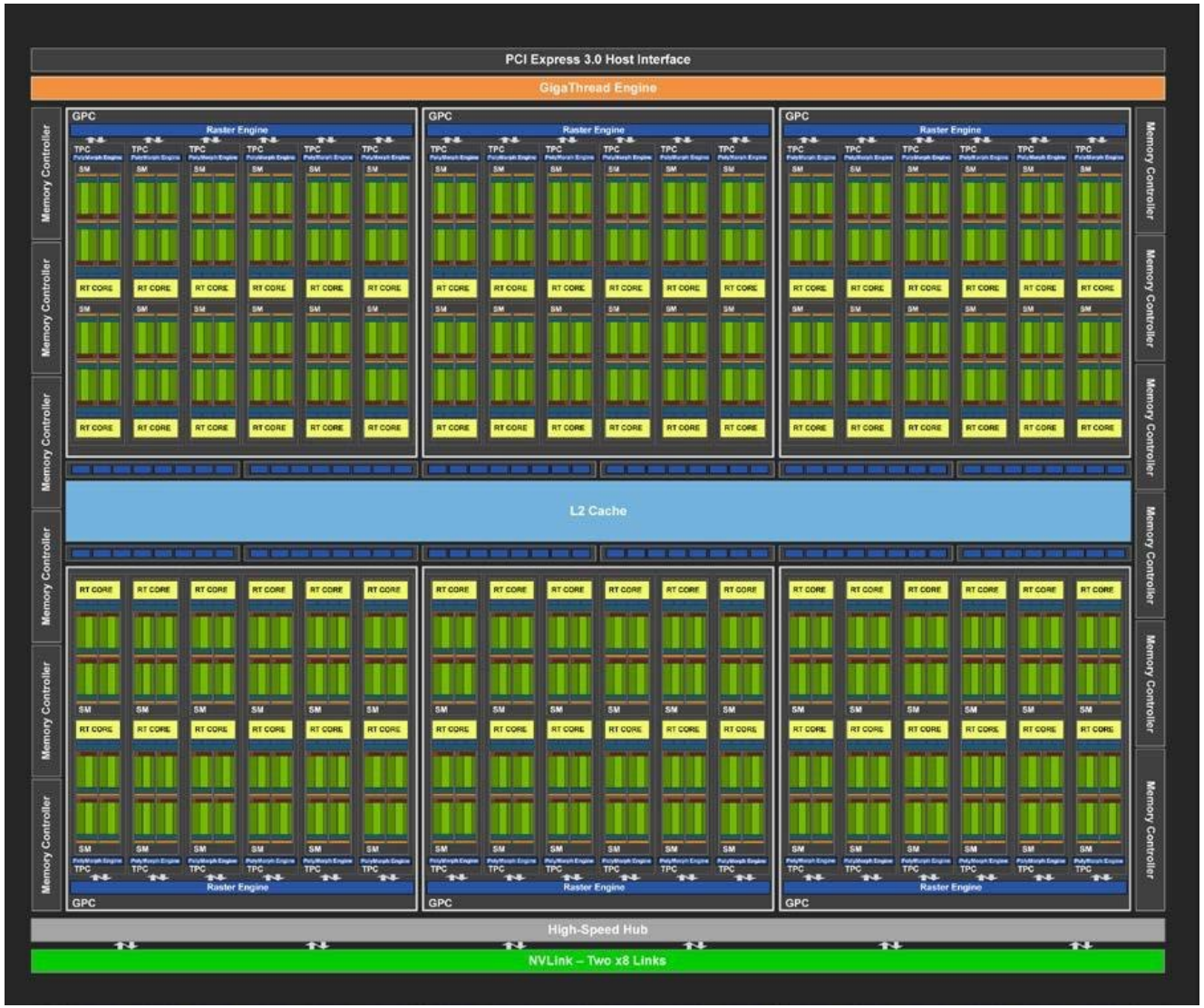
From CPU to GPU

- **GPU hardware architecture**

	CUDA Cores				Tensor Cores					
NVIDIA Architecture	FP64	FP32	FP16	INT8	FP64	TF32	FP16	INT8	INT4	INT1
Volta	32	64	128	256			512			
Turing	2	64	128	256			512	1024	2048	8192
Ampere (A100)	32	64	256	256	64	512	1024	2048	4096	16384
Ampere, sparse						1024	2048	4096	8192	



Turing102 with 72 Streaming Multiprocessors



4,608 CUDA Cores

- **72 RT Cores**
- **576 Tensor Cores**
- **288 Texture units**
- **12 32-bit GDDR6 Memory controllers (384-bits total).**

Streaming Core

- **INT32: Integer arithmetic units**
- **FP32: Floating-point arithmetic units**
- **Tensor cores: tensor transform, multiplication, etc.**
- **LD/ST: memory access load/store**
- **SFU: special function units (sin, cosine, reciprocal, and square root)**
- **Tex: texure units**