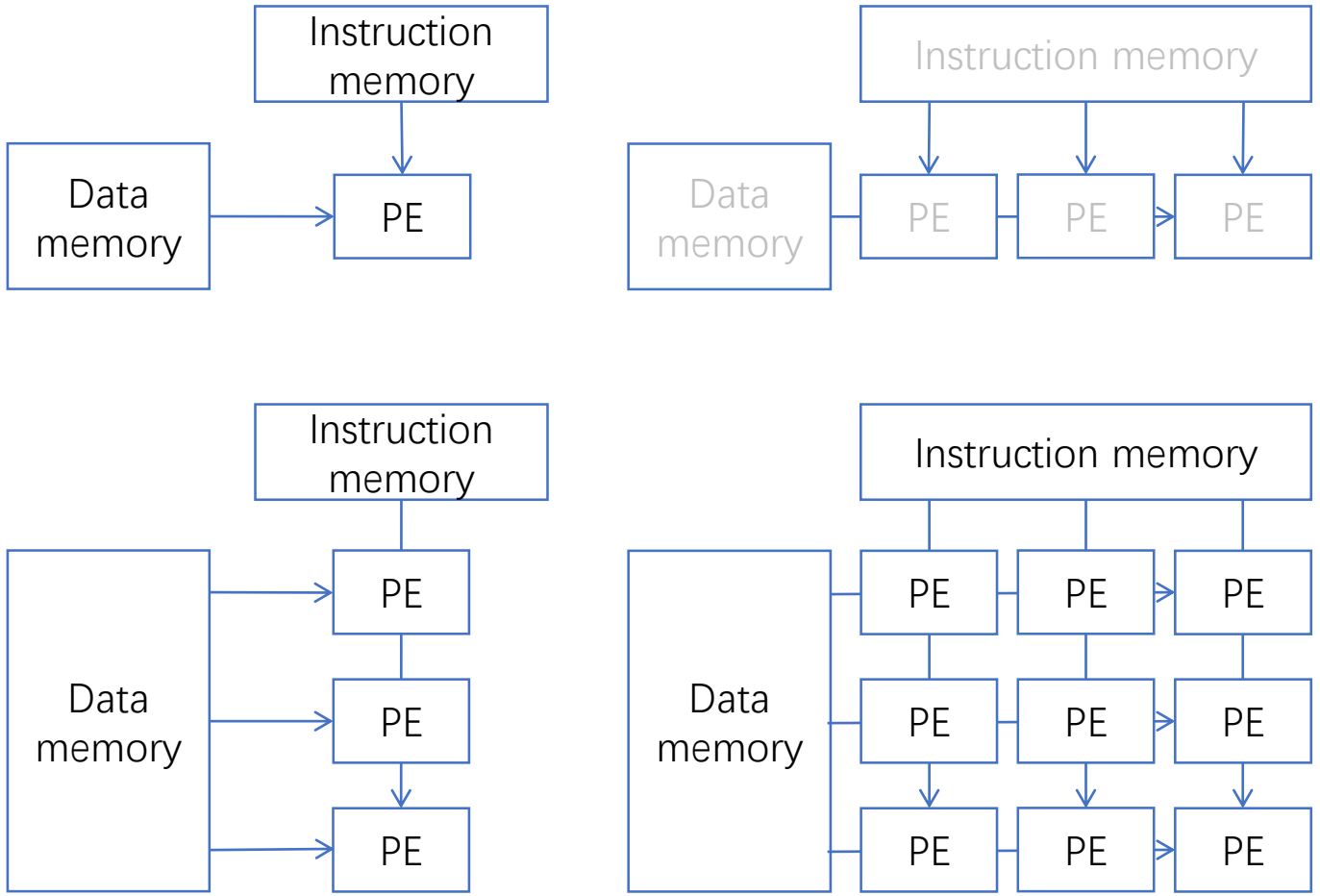
More Parallelism

- Flynn's Taxonomy
- SISD (scalar)
- **SIMD (data-level parallelism)**
  - **Requires that no dependency exists between data!**
  - Vector architectures
  - GPU (SIMT/SPMD)
- MISD
- MIMD



立志成才报国裕民

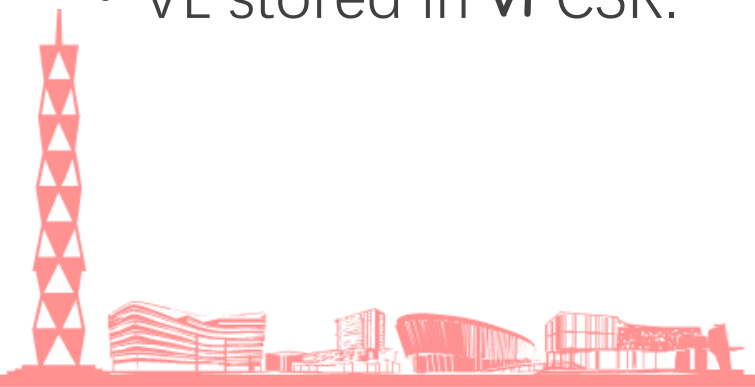
SIMD – Vector Extension in RISC-V

- ISA extension
  - Vector registers
  - Additional parameters



Vector Extension – Other Important Parameters

- Design philosophy: Vector Length Agnostic (VLA)
- SEW: standard element width, the bit width of each element, can be {8, 16, 32, 64};
- LMUL: vector length multiplier, maximum number of vector registers used for one operation, can be {1/8, ¼, ½, 1, 2, 4, 8};
- VL: vector length, number of effective vector elements, VL ⩽ VLMAX = VLEN * LMUL/SEW
- Use **vsetvli** to automatically setup VL.
- SEW, LMUL and mask/tail settings are stored in **vtype** CSR.
- VL stored in **vl** CSR.



立志成才报国裕民

Vector Extension – Example

$$B[i] = B[i] + A, \qquad 0 \le i < N$$

- Assume *A* locates at (**x4**), *B* locates at (**t1**), total vector length *N* = **$x1**

```
flw      ft0, 0(x4)          # load A
loop:
   vsetvli  x2, x1, e32, m1, ta, ma # e32, m1, SEW=32, LMUL=1; set vl, x2; ma ta mask/tail agnostic
                               # x2/vl = min(VLMAX, $x1), hardware knows what to do
   vfmv.v.f  v1, ft0           # broadcast A to vector register v1
   vle32.v   v0, (t1)          # load B[i : i+VL-1]
   vfadd.vv  v2, v0, v1        # v2[i] = v0[i] + v1[i]
   vse32.v   v2, (t1)          # store the update B[i : i+VL-1]
   slli      x5, x2, 2         # calculate the storage consumed, VL * 4
   add       t1, t1, x5        # update the vector pointer
   sub       x1, x1, x2        # update the remaining number of elements
   bnez      x1, loop          # continue with the remaining elements
```

立志成才报国裕民

Vector Extension vs. Scalar Machine

$$B[i] = B[i] + A, \qquad 0 \le i < N$$

- Assume *A* locates at (**x4**), *B* locates at (**t1**), total vector length *N* = **$x1**

```
flw      ft0, 0(x4)
loop:
   vsetvl  x2, x1, e32, m1, ta, ma

   vfmv.v.f  v1, ft0
   vle32.v   v0, (t1)
   vfadd.vv  v2, v0, v1
   vse32.v   v2, (t1)
   slli      x5, x2, 2
   add       t1, t1, x5
   sub       x1, x1, x2
   bnez      x1, loop
```

Scalar assembly

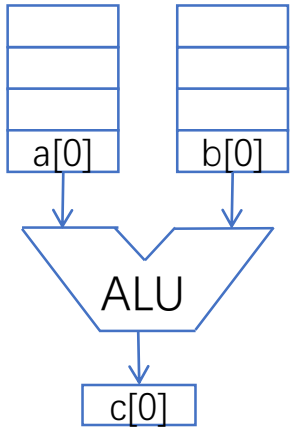
```
flw      ft0, 0(x4)      # Load scalar A
li       x2, 0           # i = 0
loop:
   bge      x2, x1, done    # If i >= N, exit
   flw      ft1, 0(t1)      # Load B[i]
   fadd.s   ft1, ft1, ft0   # B[i] + A
   fsw      ft1, 0(t1)      # Store result to B[i]
   addi     t1, t1, 4       # Advance pointer by 4 bytes
   addi     x2, x2, 1       # i++
   j        loop
```

立志成才报国裕民

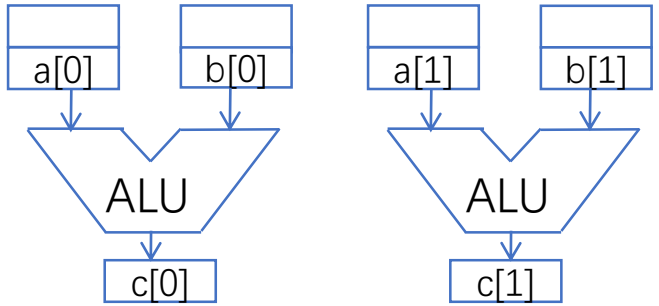
Vector extension – Hardware – ALU

- Deeper/wider pipeline/ALU

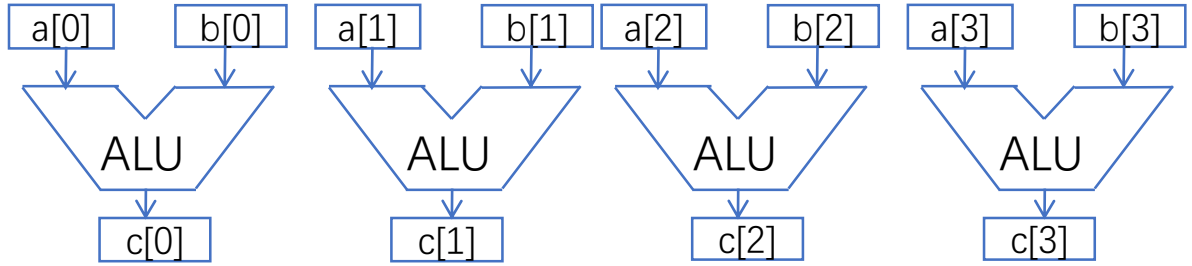
Serial, pipelined



Mixed, multi-lane



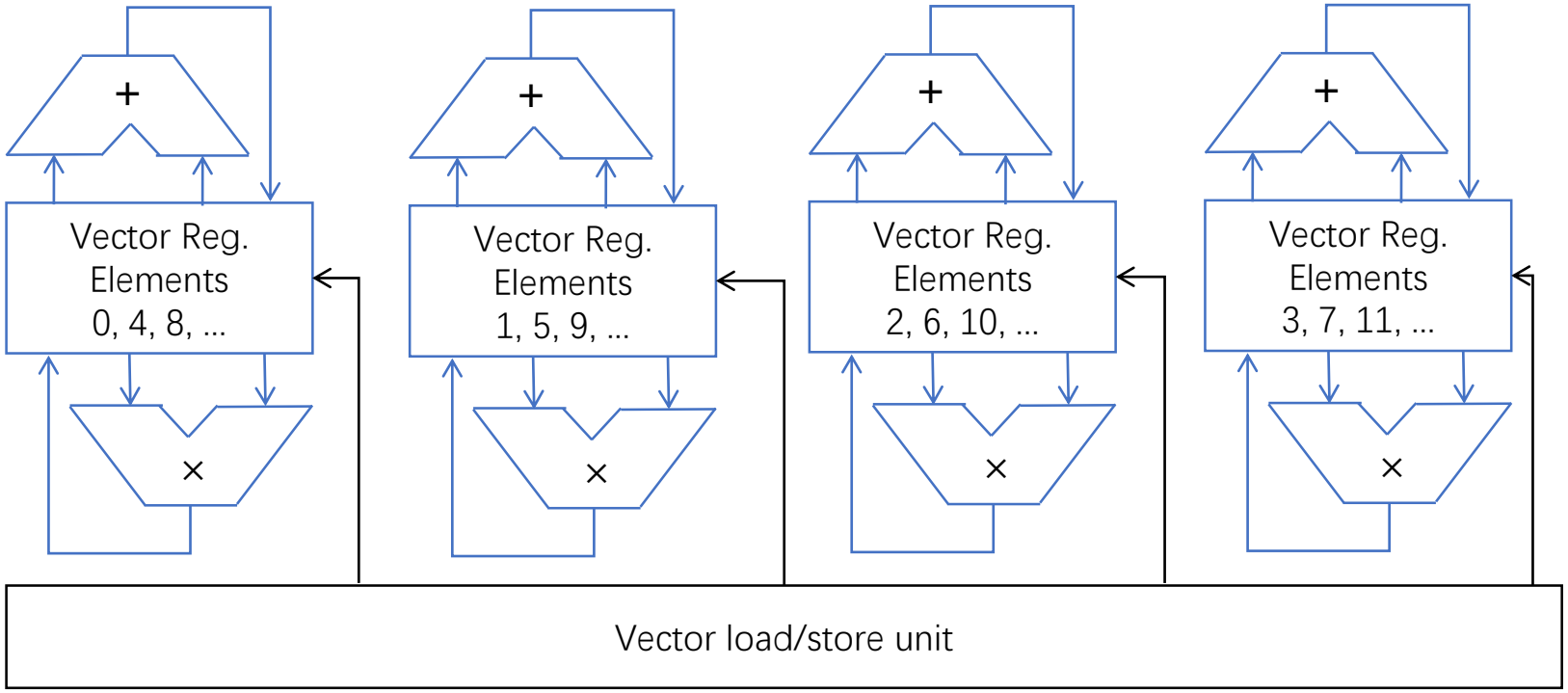
Fully parallel



	Serial	Mixed	Parallel
Latency			
Bandwidth			
Hardware			

Vector ALU & Registers

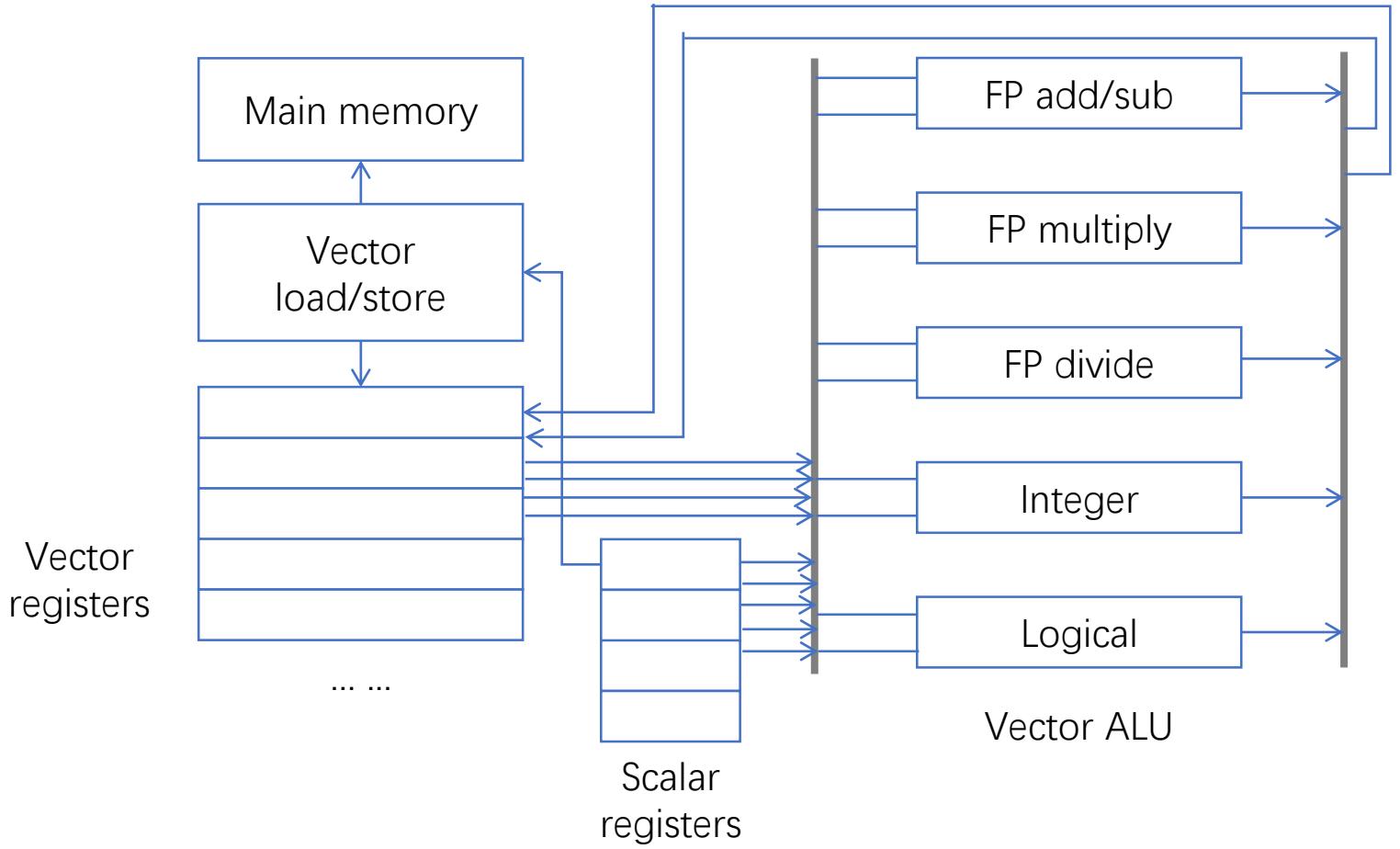
- Can also allocate vector registers to different ALUs




```
# Integer multiply-add
# overwrite addend
vmacc.vv vd, vs1, vs2, vm
# vd[i] += (vs1[i] * vs2[i])
vmacc.vx vd, rs1, vs2, vm
# vd[i] += (x[rs1] * vs2[i])
```

立志成才报国裕民

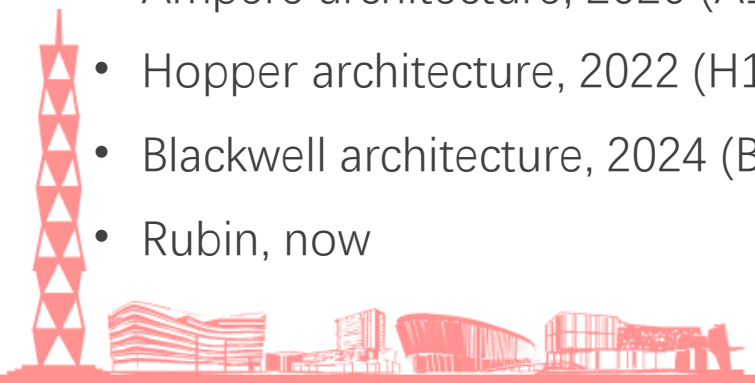
A Possible Vector Processor Implementation



立志成才报国裕民

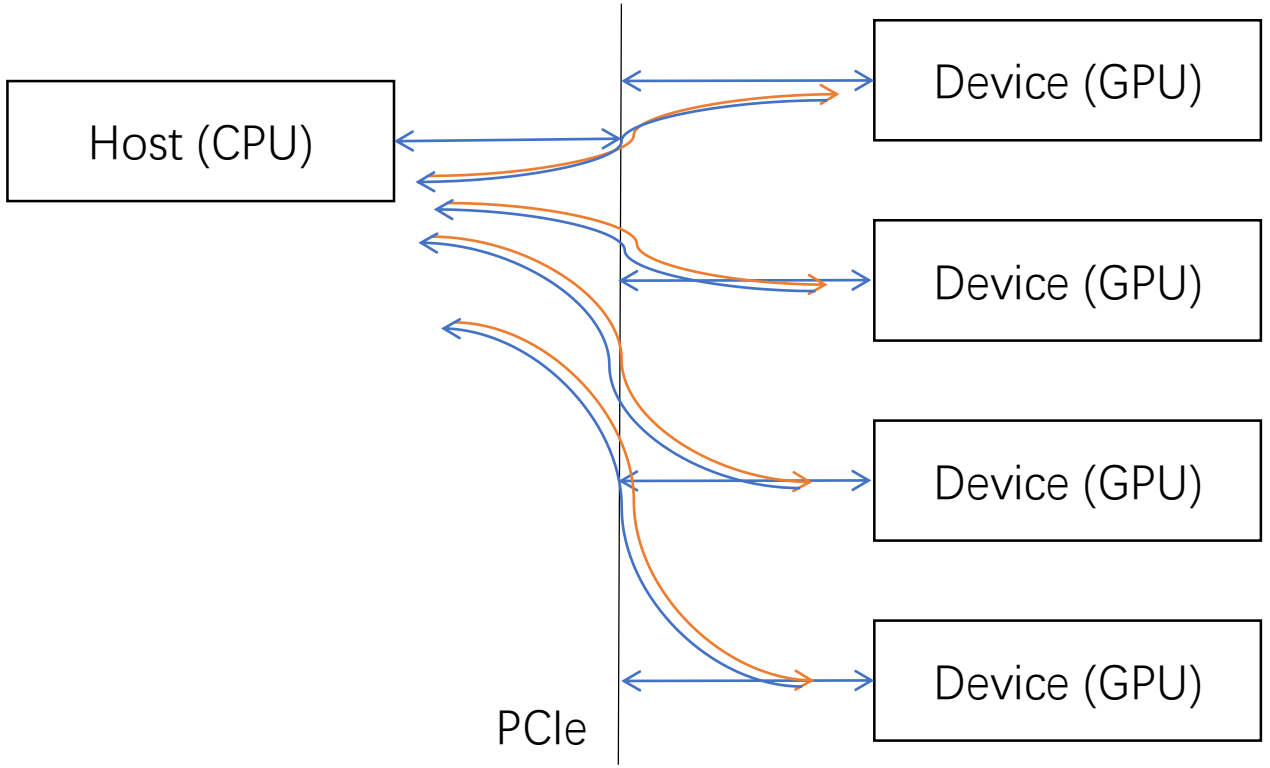
GPU Architecture – History

- 1993, Jen-Hsun Huang co-founded Nvidia;
- NV1 (3D game, 2MB VRAM, ST microelectronics 1995), NV2, NV3 (AKA, RIVA 128/128ZX, polygon-based redering, 2D/3D, 1997 4/8MB), NV4, NV5, …
- First graphic card to call “GPU”, GeForce 256, 1999, IPO at 12$/share
- First programmable GPU GeForce3, 2001
- SLI (Scalable Link Interface), enable mutli-GPU connection, 2004
- CUDA, 2006
- Fermi architecture, 2009
- Kepler architecture, 2012 (AlexNet appears the same year)
- Ampere architecture, 2020 (A100)
- Hopper architecture, 2022 (H100)
- Blackwell architecture, 2024 (B200/GB200)
- Rubin, now



立志成才报国裕民

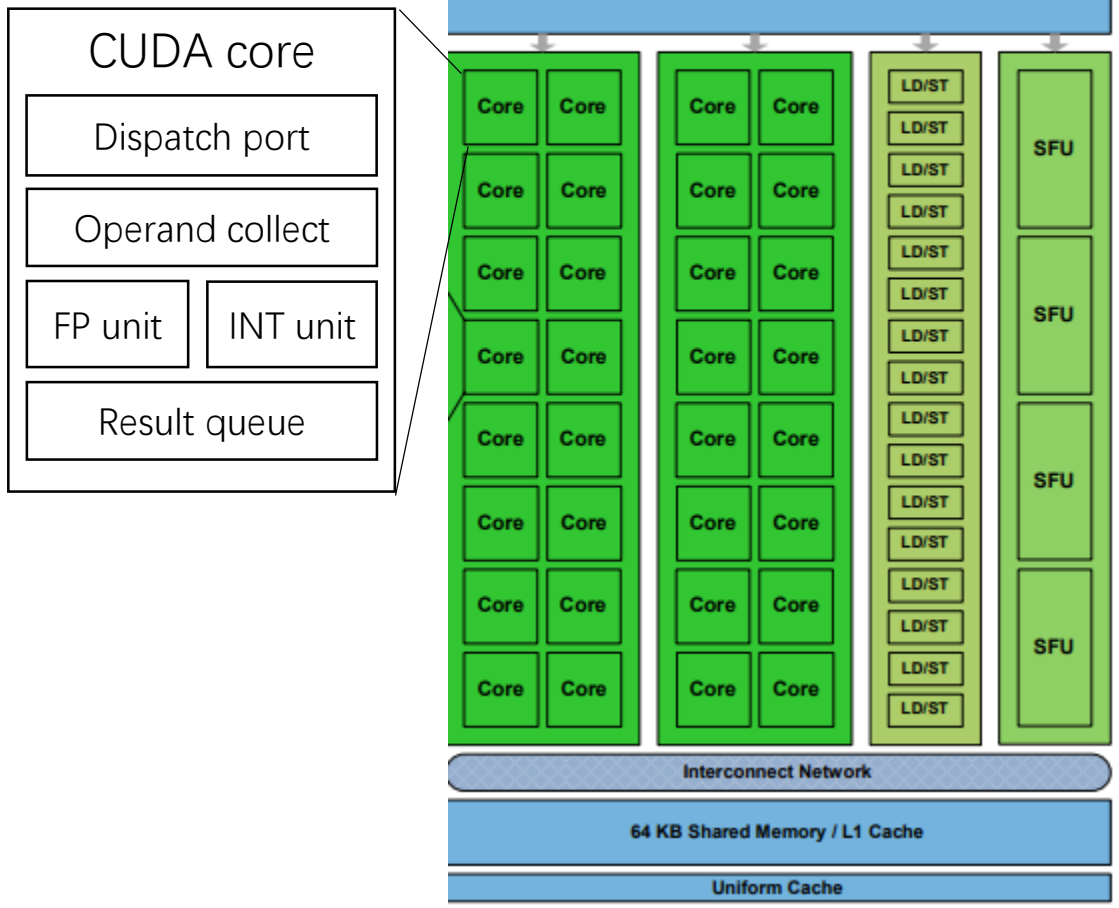
GPU Work Flow



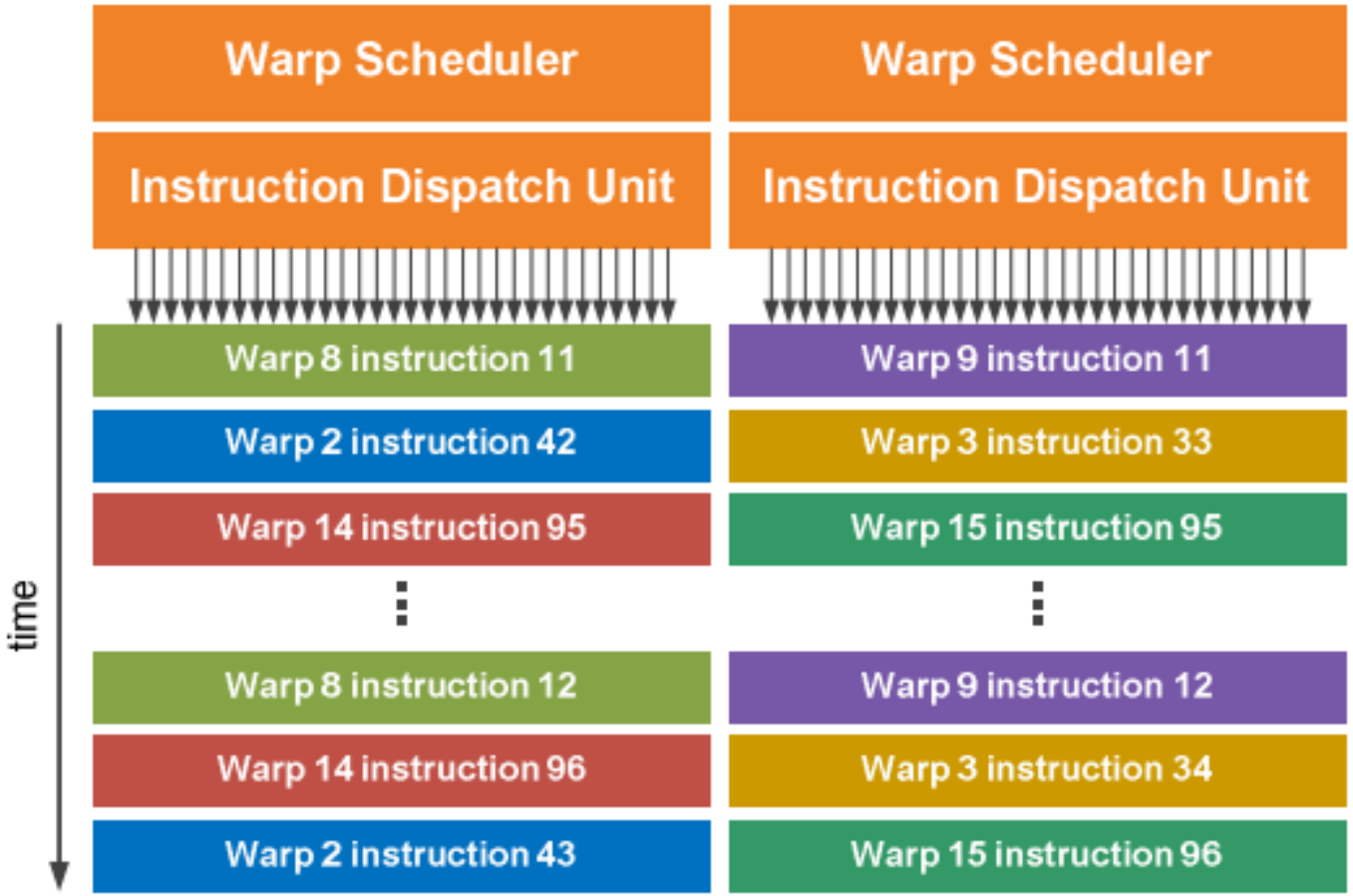
立志成才报国裕民

Fermi Architecture

- The SM schedules threads in groups of 32 parallel threads called warps (thread of vector instructions)
  - 2 warp schedulers
  - 2 instruction dispatch units
  - Dispatch 1 instruction for each warp of
    - 16 cores
    - 16 load/store units
    - 4 SFUs
  - Fermi's scheduler does not need to check for dependency
    - 2 INT instructions/2 FP instructions
    - 1 mixed INT/FPU/load/store/SFU instruction
    - Double instruction does not support dual dispatch



Warp Scheduler



Because threads of SIMD instructions are independent, the scheduler may select a different SIMD Thread each time.

CUDA (Compute Unified Device Architecture)

- Enable NVIDIA GPUs to execute programs written in C, C++, Fortran, OpenCL, etc.
- CUDA's hierarchy of threads

CUDA program

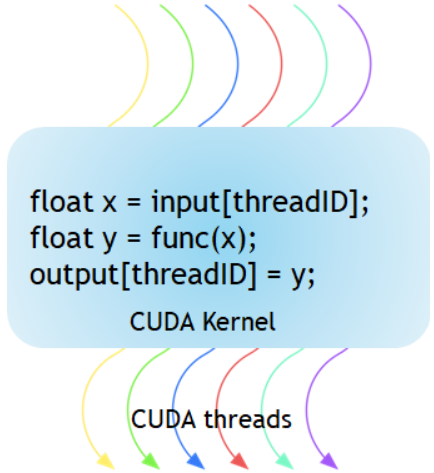
Kernels

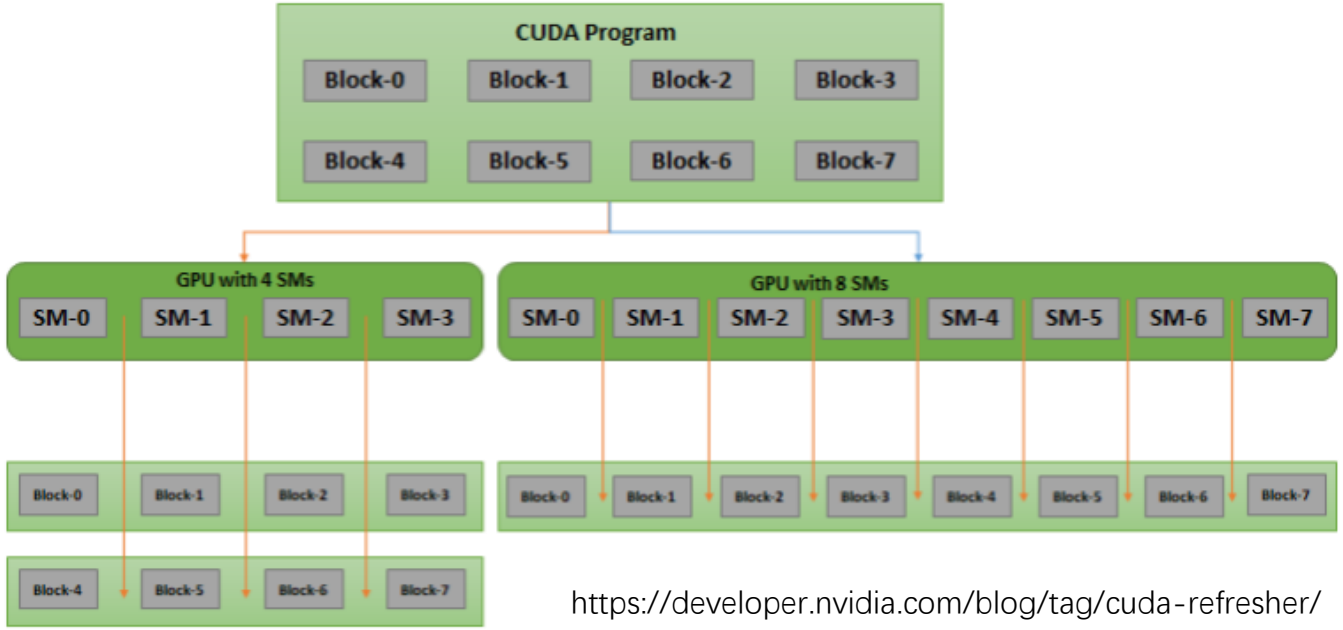
Grids

Blocks

Threads

```
/** CUDA kernel device code - CUDA Sample Codes
 * Computes the vector multi. of A and B into C. The three vectors have the same number of elements asnumElements.
 */
__global__ void vectorMulti( float *A, float *B, float *C, int numElements) {
  int i = blockDim.x * blockIdx.x + threadIdx.x;
  if (i < numElements) {
    C[i] = A[i] * B[i];
  }
}
```



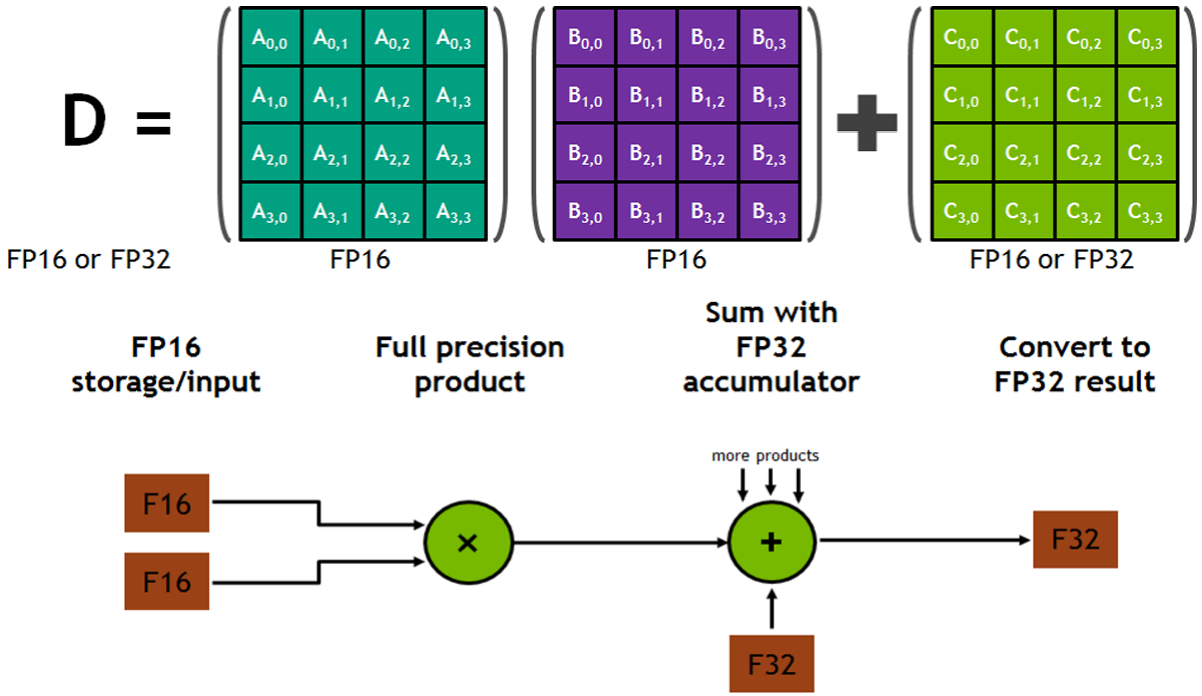


https://developer.nvidia.com/blog/tag/cuda-refresher/

立志成才报国裕民

Recent NVIDIA GPUs

- Volta architecture with tensor core



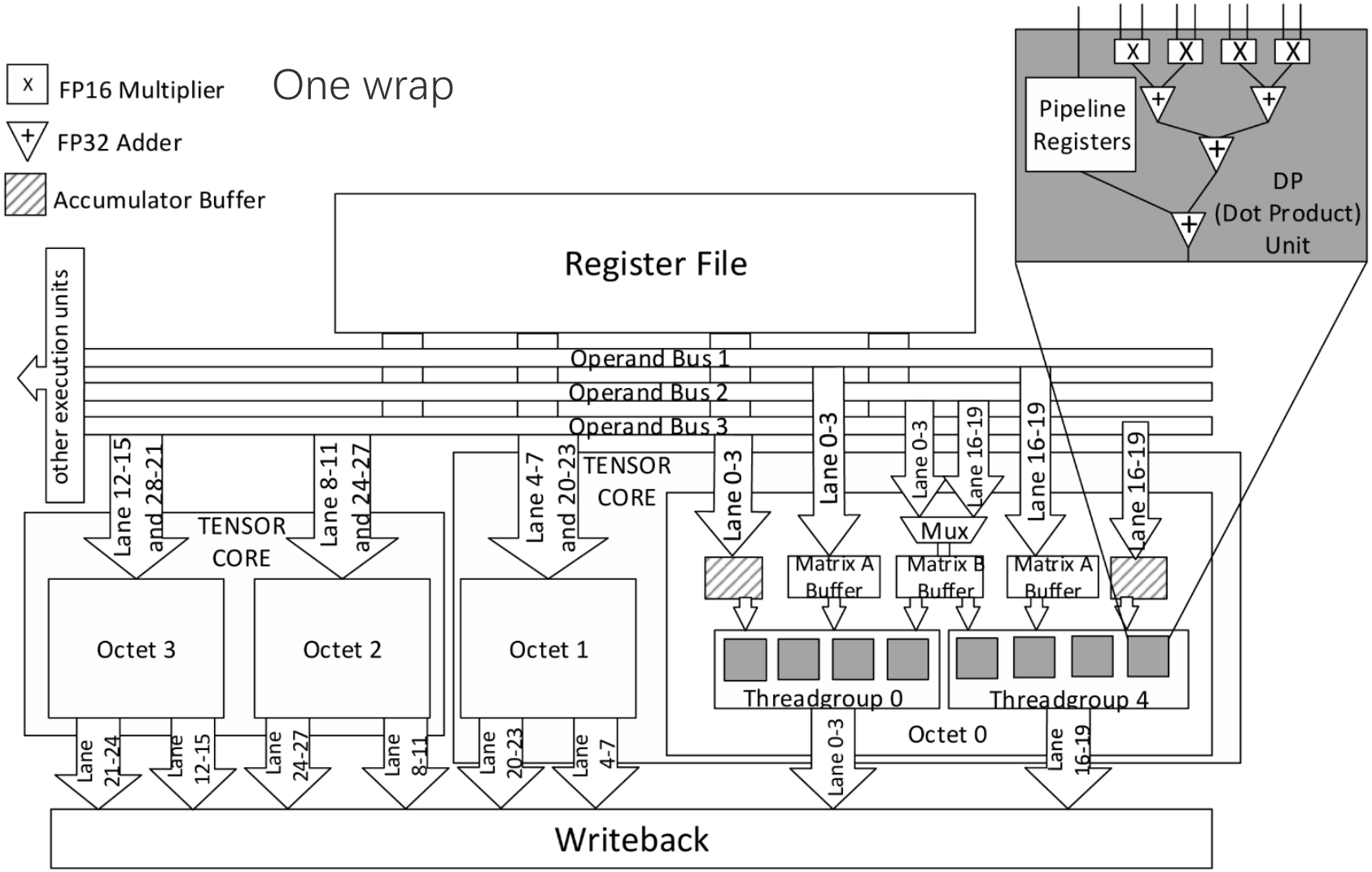


- **wmma** API for tensor core programming, e.g.

```
wmma::load_matrix_sync()
wmma::mma_sync()
wmma::store_matrix_sync()
```

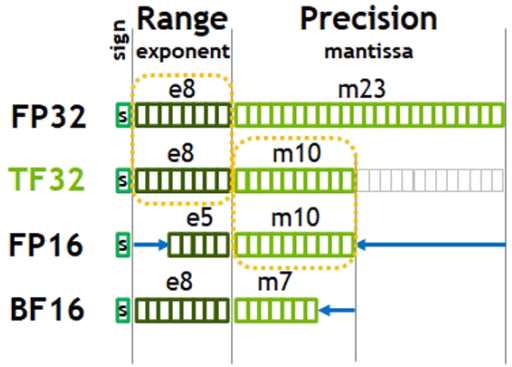

立志成才报国裕民

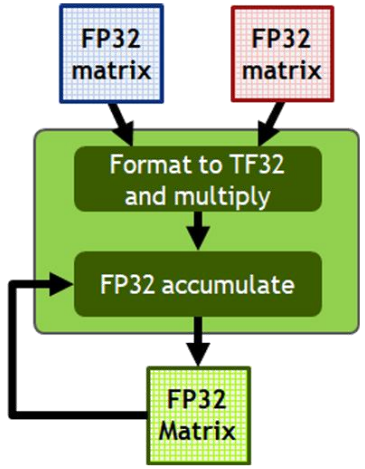
Demystifying Tensor Core



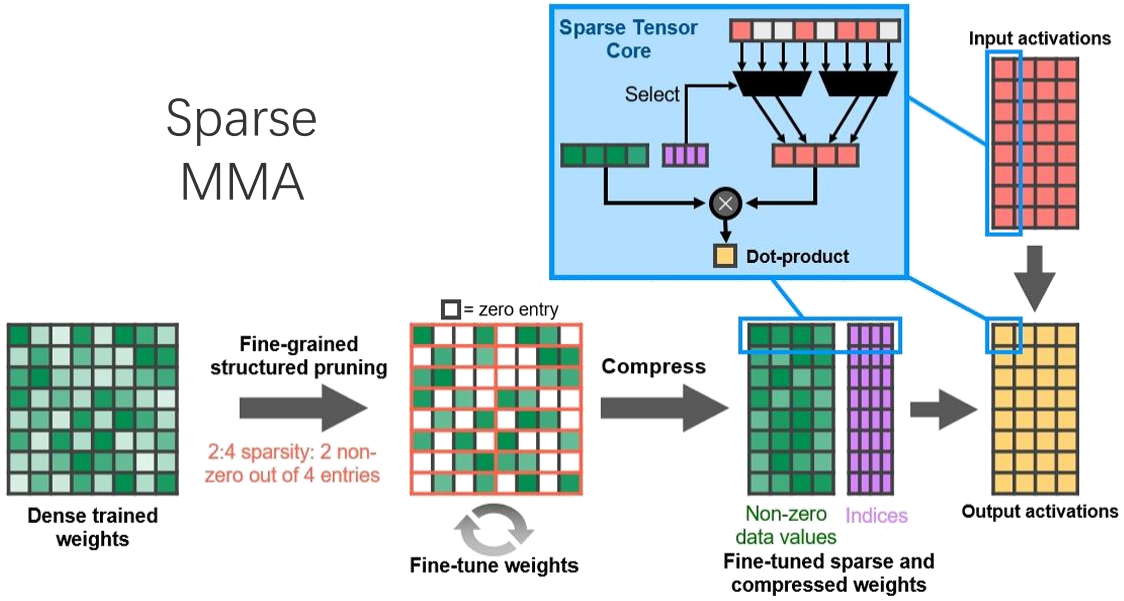
M. A. Raihan, N. Goli and T. M. Aamodt, "Modeling Deep Learning Accelerator Enabled GPUs," *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Madison, WI, USA, 2019,

NVIDIA A100 Architecture





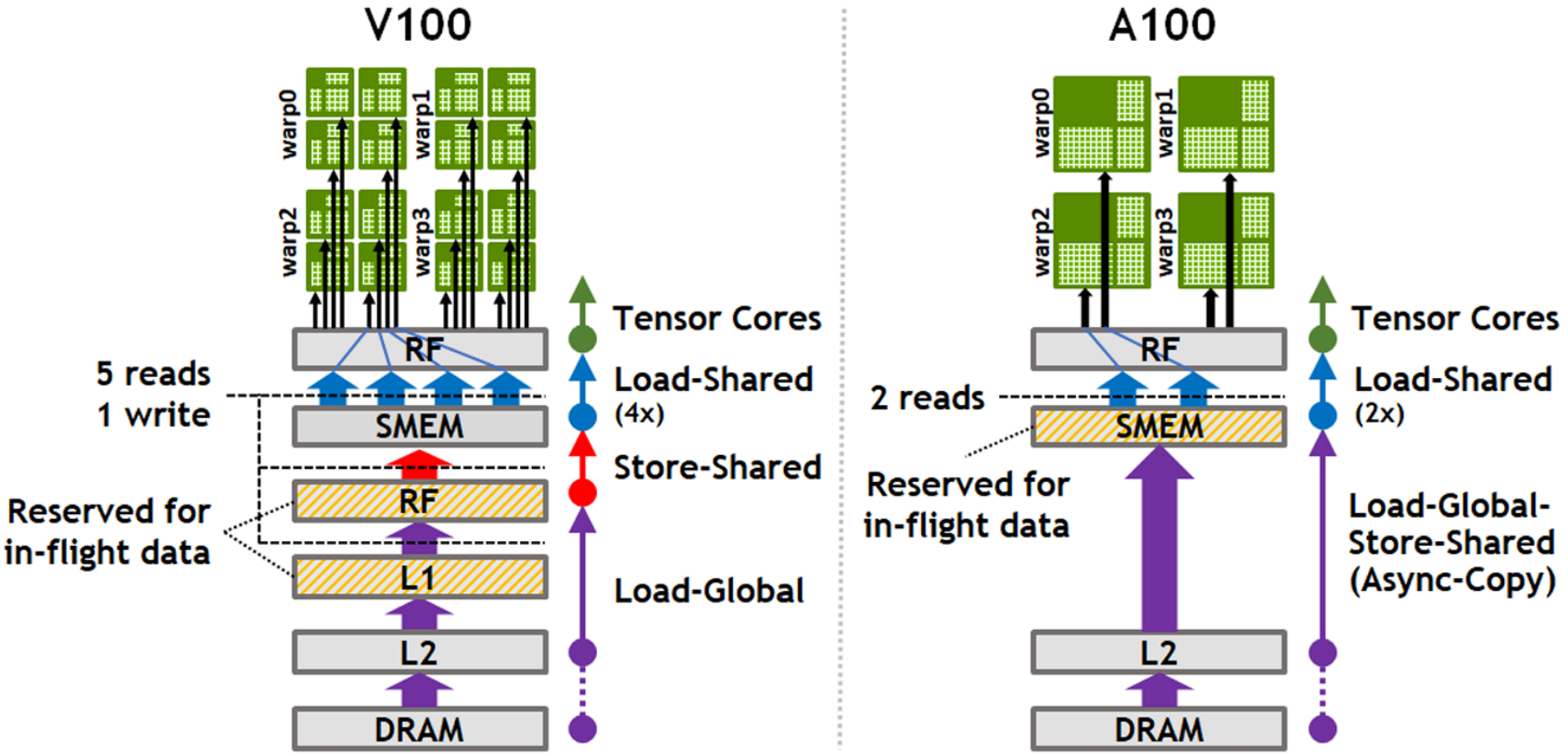
Sparse MMA



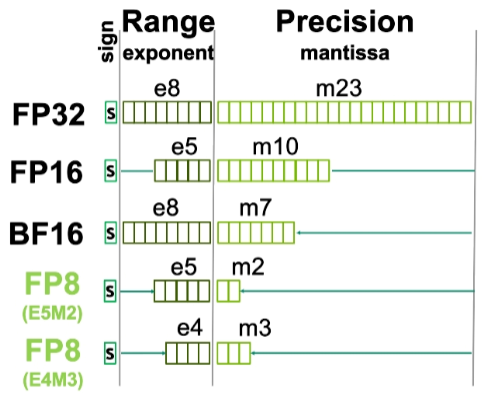


NVIDIA A100 Architecture – More Data Reuse

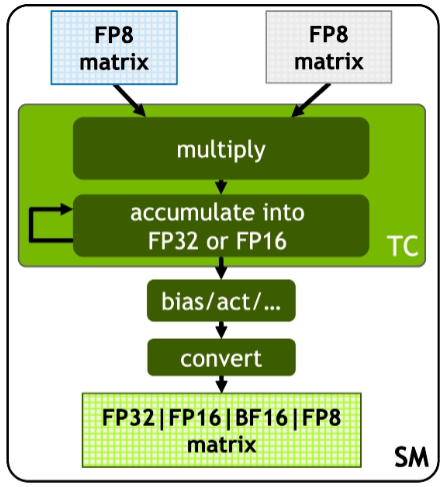
- Enable async. read to the shared memory;
- Data shared across all 32 threads in a warp, compared to 8 threads on Volta's tensor core.



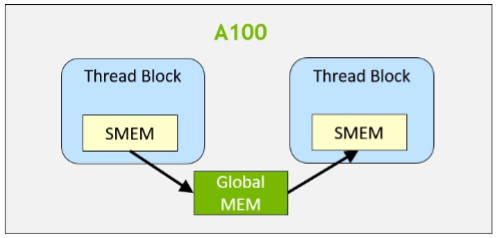
NVIDIA H100 Architecture

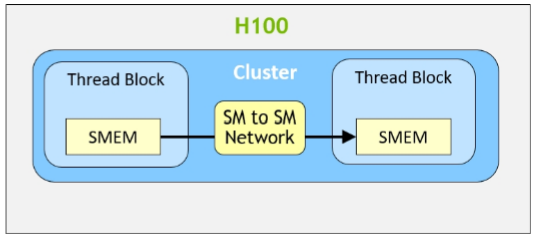


Allocate 1 bit to either range or precision



Support for multiple accumulator and output types





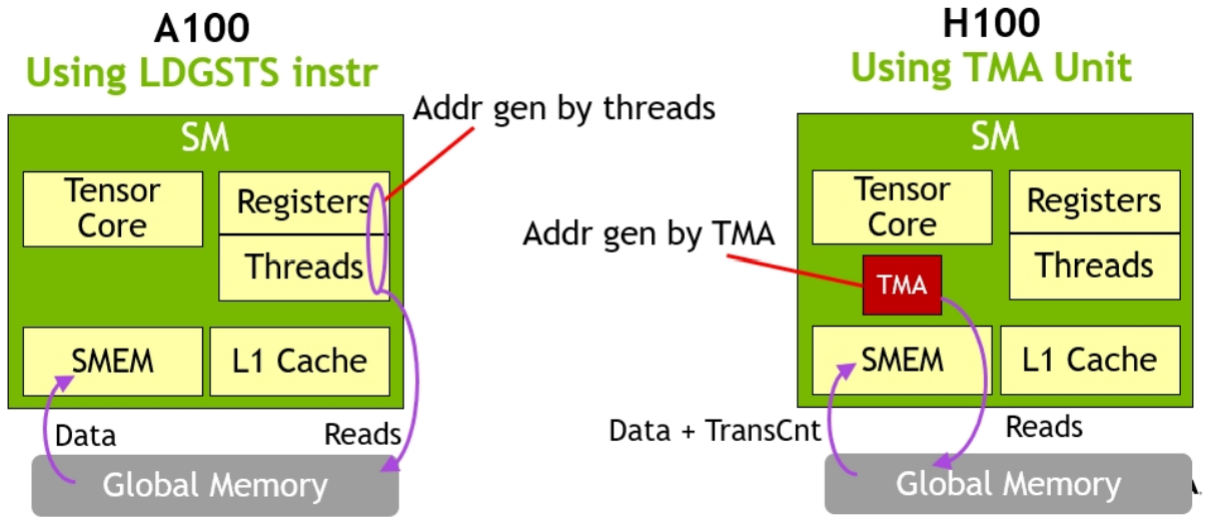
Distributed SMEM

CUDA Programming Model Exposure	A100	New for H100
Barrier.arrive(),	Asynchronous Barrier	**Asynchronous Transaction Barrier**
Barrier.wait()	Waiter spins in SMEM	**Waiter sleeps until all threads arrive**
Memcpy_async()	Direct copy to SMEM	**Asynchronous mem copy unit (called TMA)**

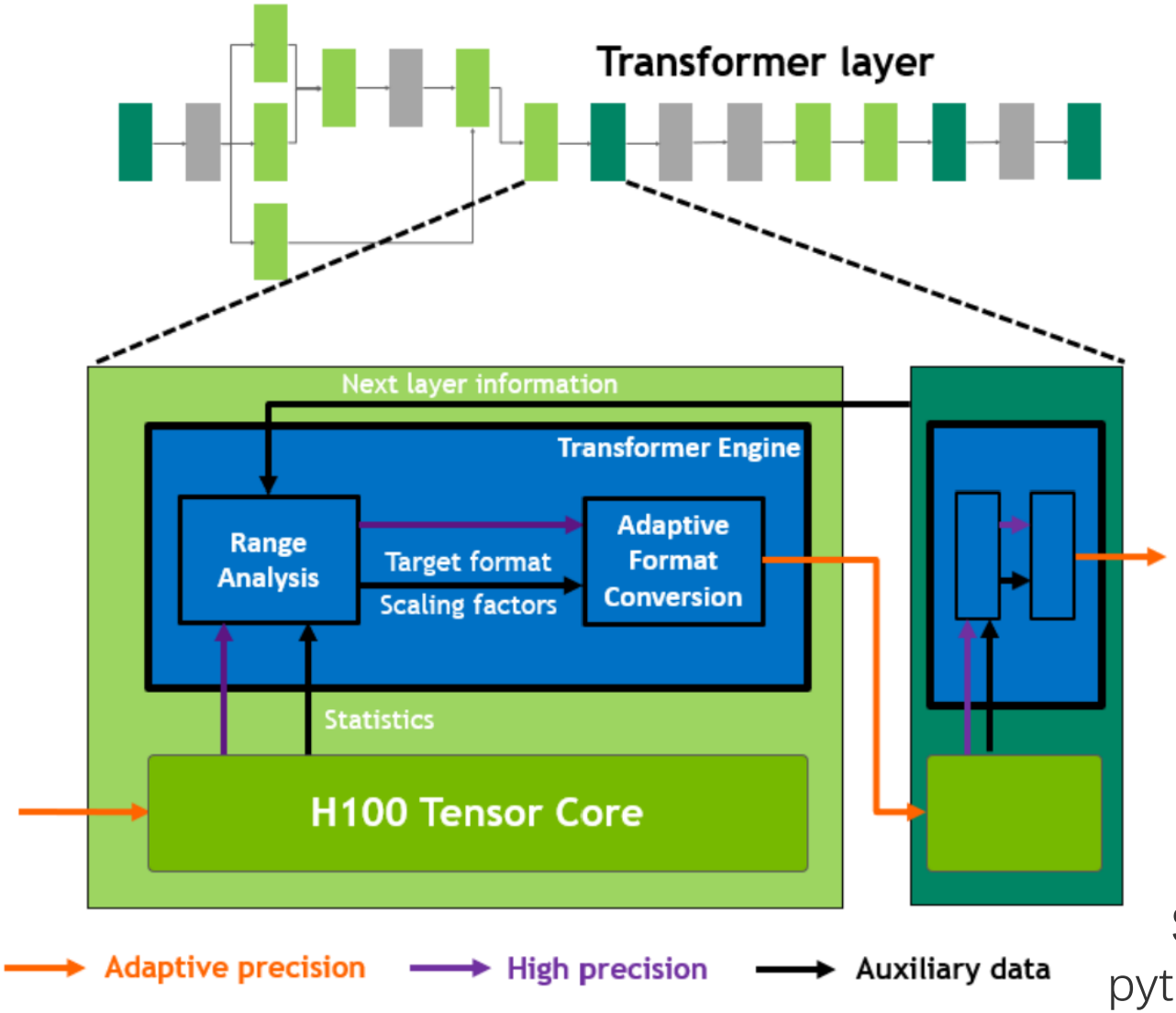


NVIDIA H100 Architecture – TMA

- Tensor memory accelerator



NVIDIA H100 Architecture – Transformer Engine



Supported by JAX and pytorch, details not revealed

Summary

- Old wisdom can be applied to modern AI accelerator design
  - ILP (pipeline and multi-issue/VLIW)
  - DLP (SIMD)



立志成才报国裕民

Roofline Model

Instructor: Siting Liu

School of Information Science and Technology (SIST), ShanghaiTech University

Shanghai Institute of Semiconductor Innovation (SISI)

https://faculty.sist.shanghaitech.edu.cn/liust/EE219/EE219.html



Computer Architecture - EE219

faculty.sist.shanghaitech.edu.cn





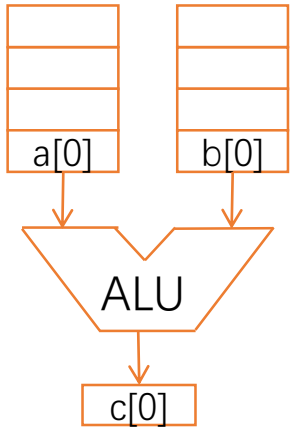
Vector ALU

- Deeper/wider pipeline/ALU

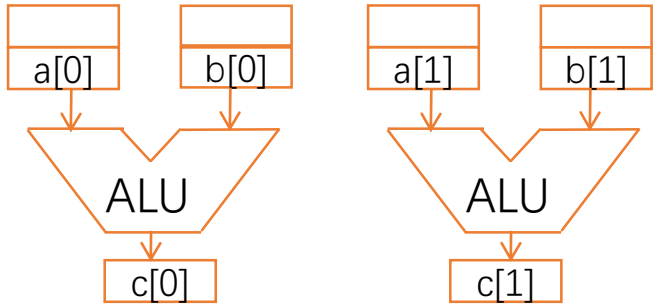


Which one to choose?

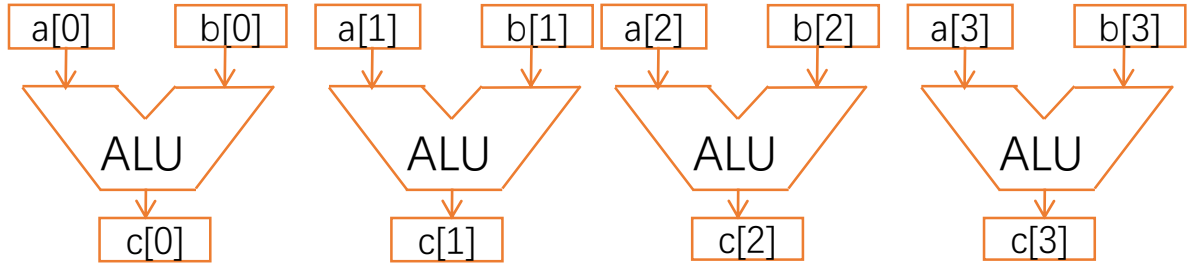
Serial, pipelined



Mixed, multi-lane



Fully parallel



	Serial	Mixed	Parallel
Latency	4 ccs	2 ccs	1 cc
Bandwidth (read)	2 elements/cycle	4 elements/cycle	8 elements/cycle
Hardware	1 ALU	2 ALUs	4 ALUs

立志成才报国裕民

Roofline Model

- **Bound & Bottleneck** analysis: Provide insight into the primary factors affecting the performance of computer (SIMD) systems.

Attainable Gflops = Min(Peak Floating Point Performance, Peak Memory Bandwidth × Operational Intensity)

Number of functional units/PEs

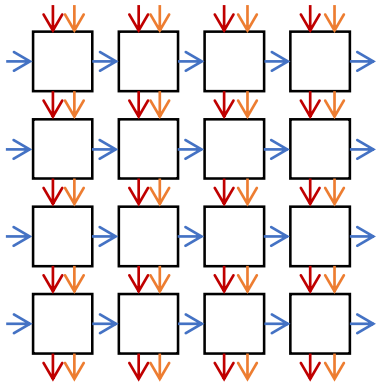
Average speed of data movement, not necessarily the physical connection ports

Depends on applications

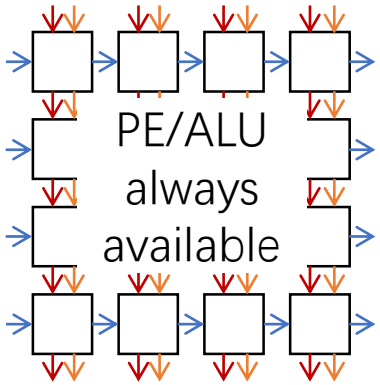
$$\frac{\text{Number of computations}}{\text{Number of data access}}$$

Reuse of data

Data always available



PE/compute-bound

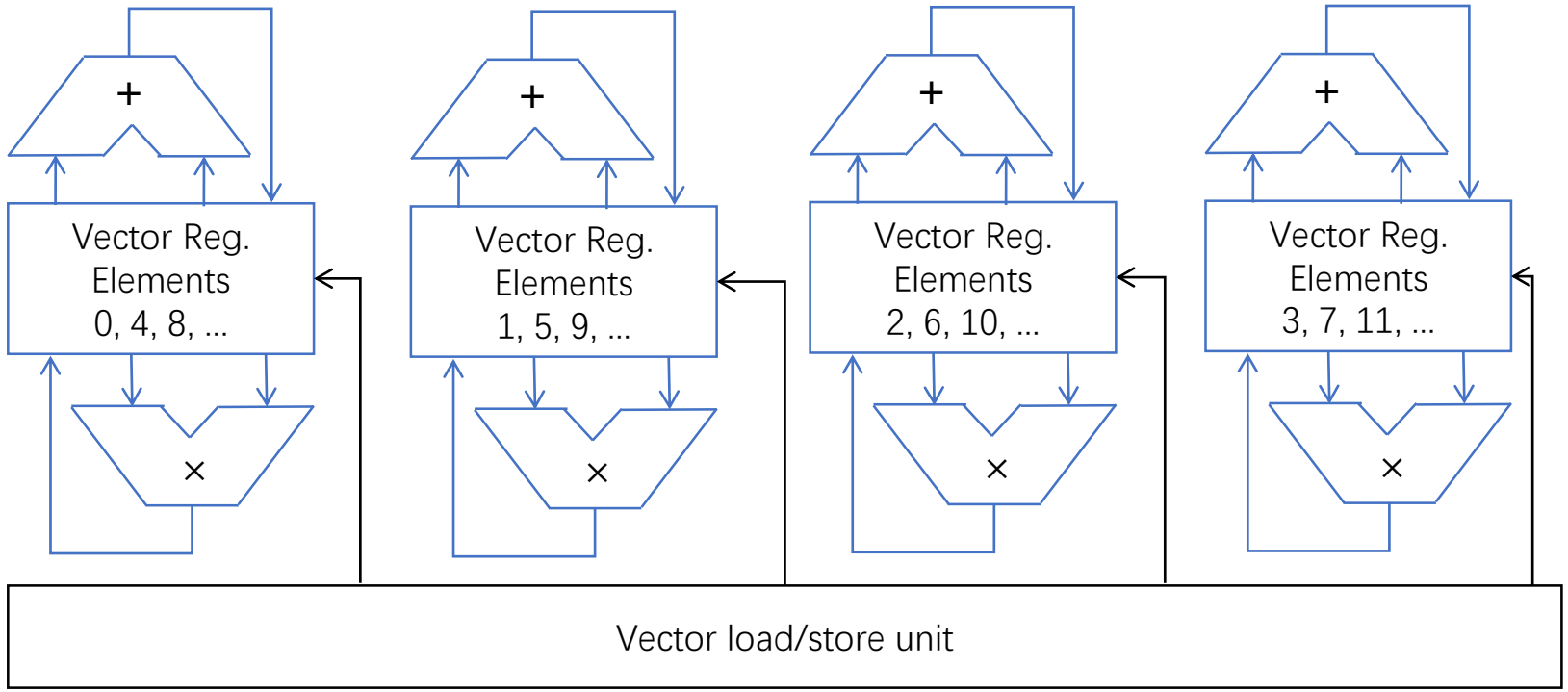


Memory/band width-bound

立志成才报国裕民

Hardware Designers' Perspective

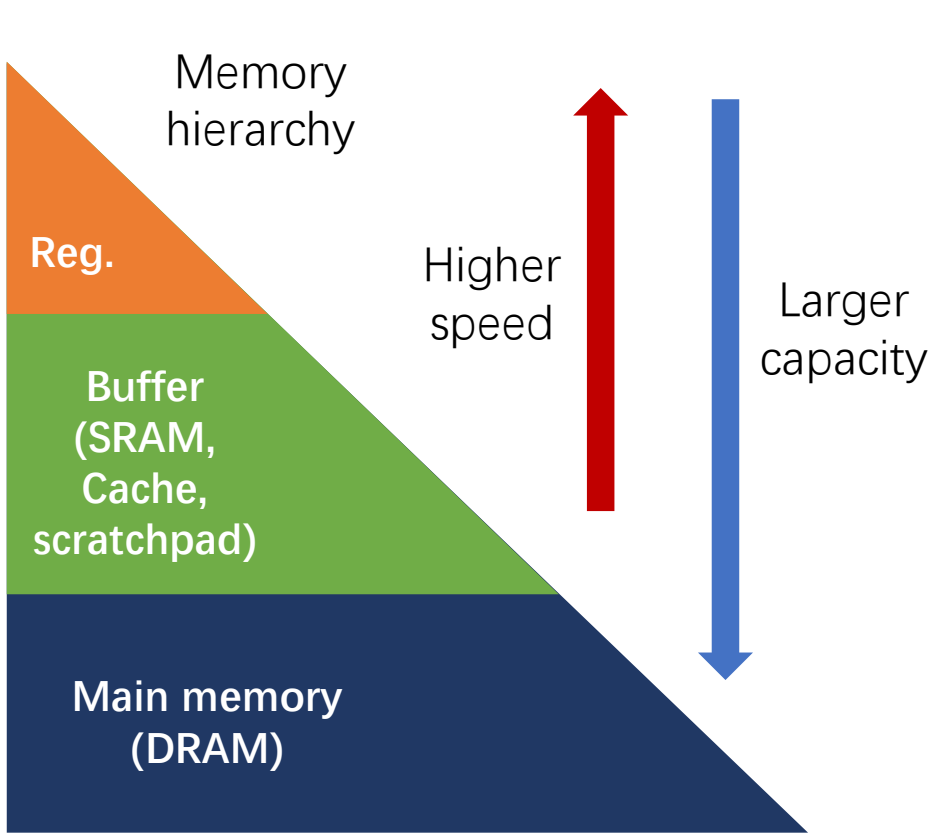
- Match the memory bandwidth and the number of compute units.

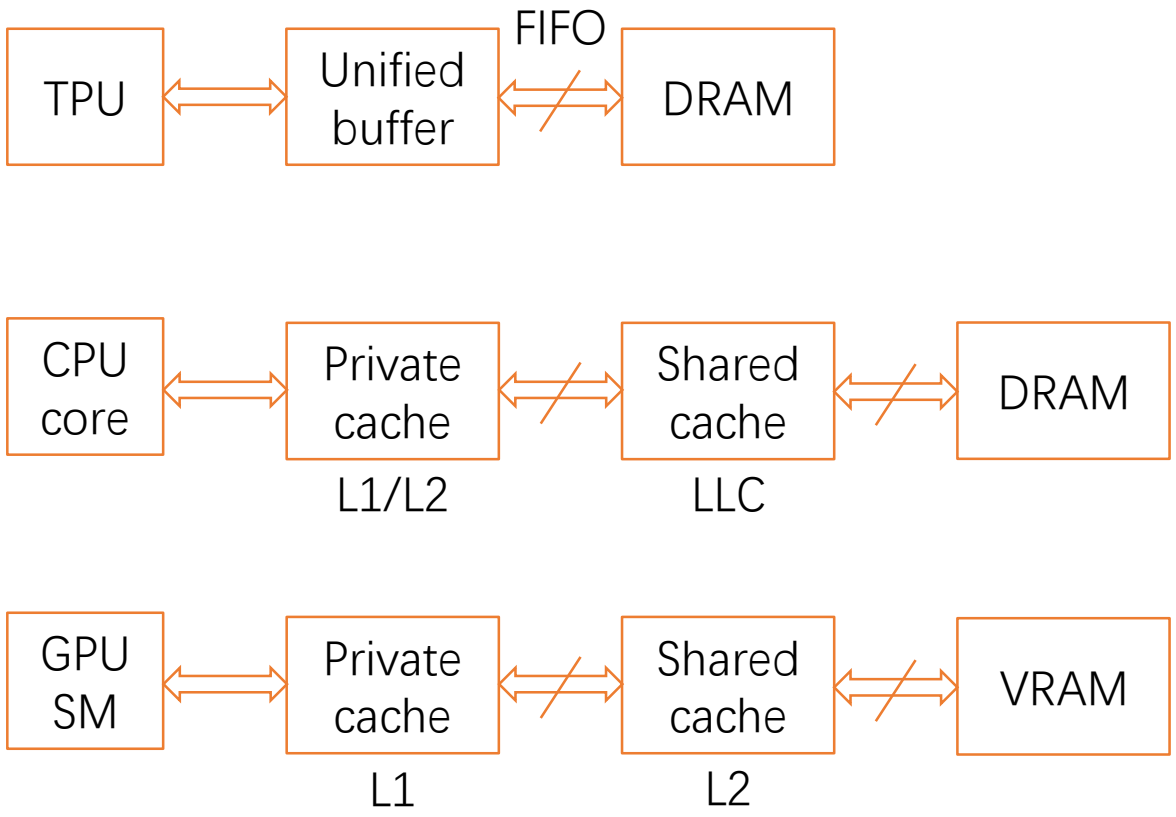


立志成才报国裕民

But …

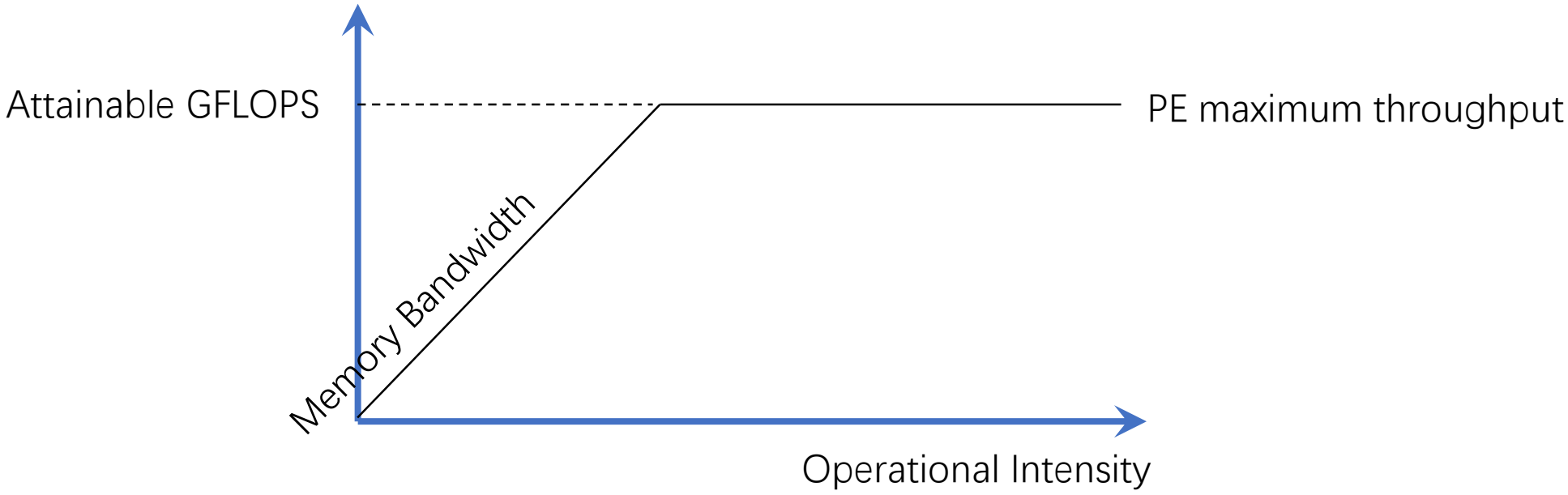
- Lower-level memory cannot match the speed/frequency of PEs/cores





Implications of Roofline Model

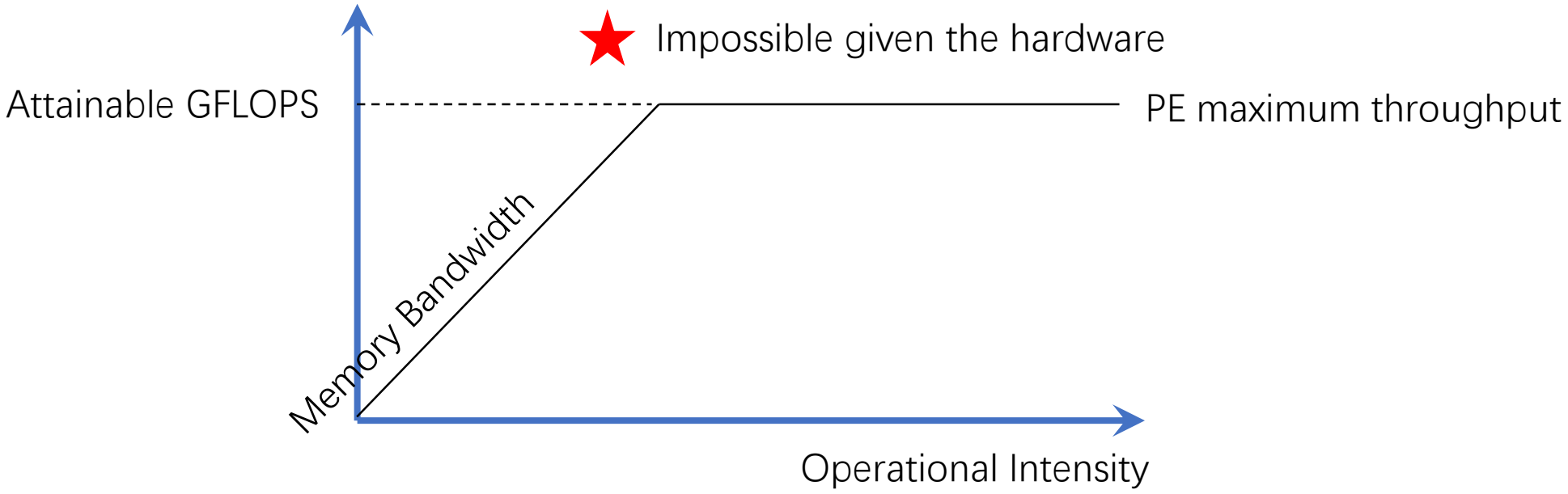
Attainable GFLOPS=Min(Peak Floating Point Performance,
Peak Memory Bandwidth × Operational Intensity)



立志成才报国裕民

Implications of Roofline Model (Cont'd)

Attainable GFLOPS=Min(Peak Floating Point Performance,
Peak Memory Bandwidth × Operational Intensity)

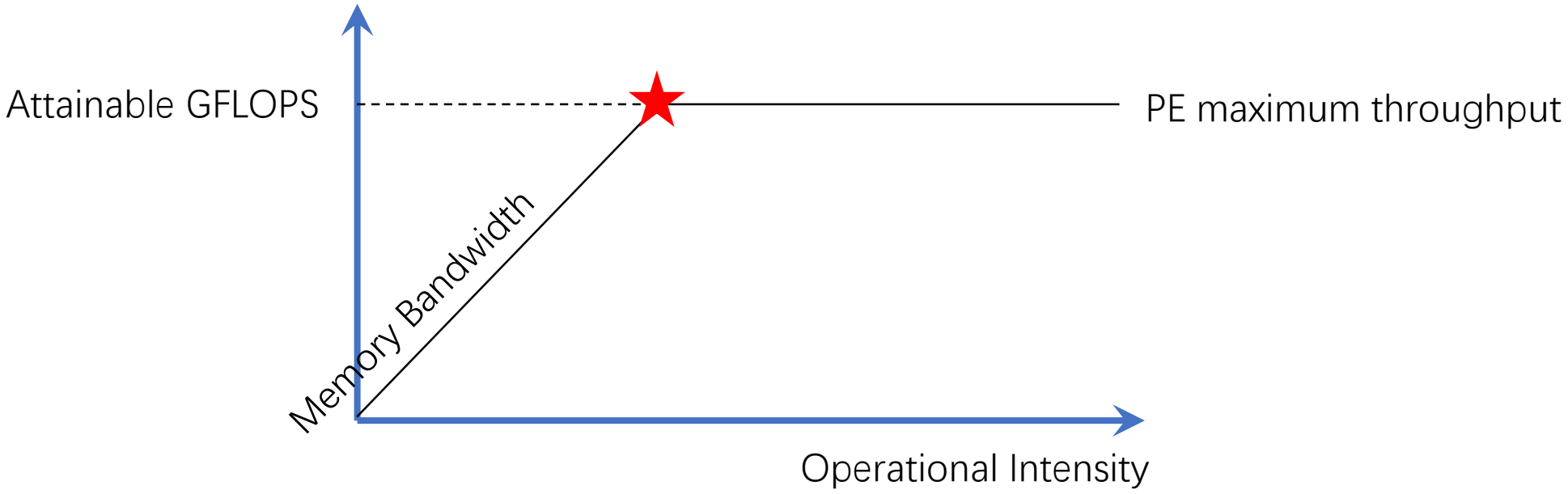


立志成才报国裕民

Implications of Roofline Model (Cont'd)

Typical machine balance (CPUs & GPUs): 5-10 FLOPs/byte
Hard to improve

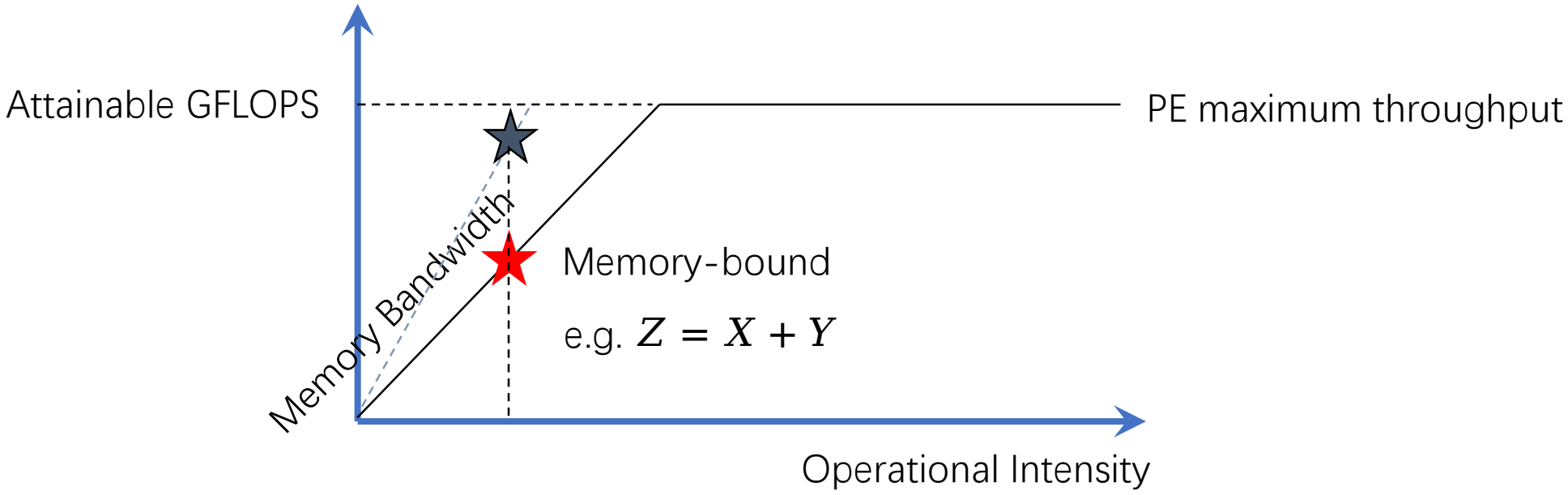
Attainable GFLOPS=Min(Peak Floating Point Performance, Peak Memory Bandwidth × Operational Intensity)



立志成才报国裕民

Implications of Roofline Model (Cont'd)

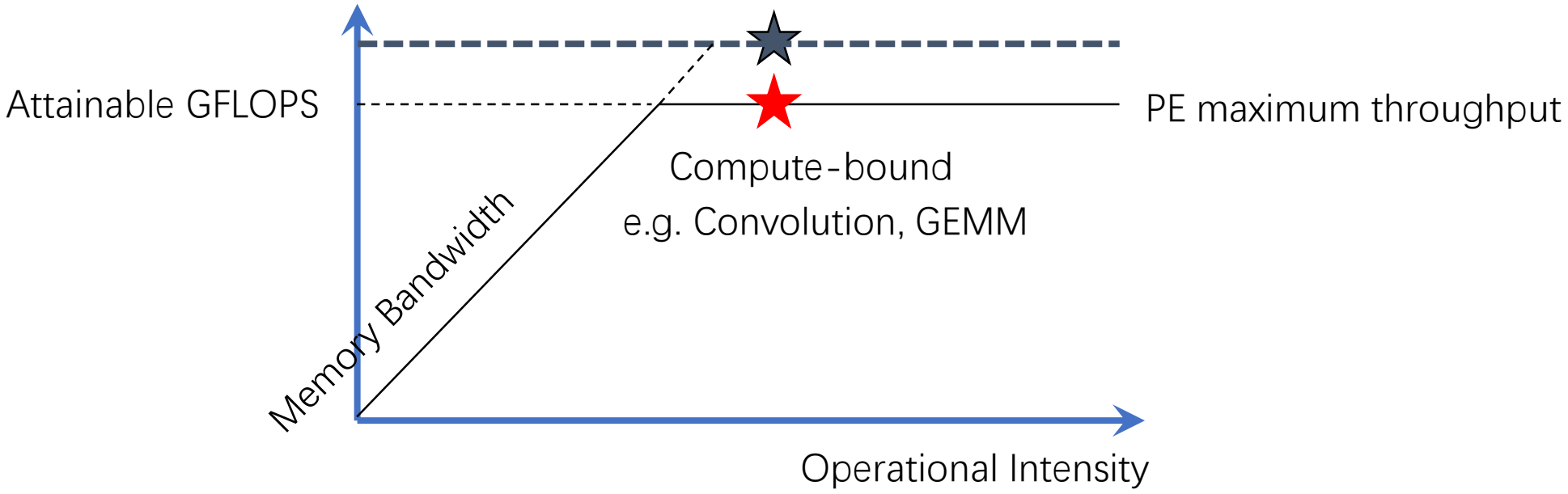
Attainable GFLOPS=Min(Peak Floating Point Performance,
Peak Memory Bandwidth × Operational Intensity)



立志成才报国裕民

Implications of Roofline Model (Cont'd)

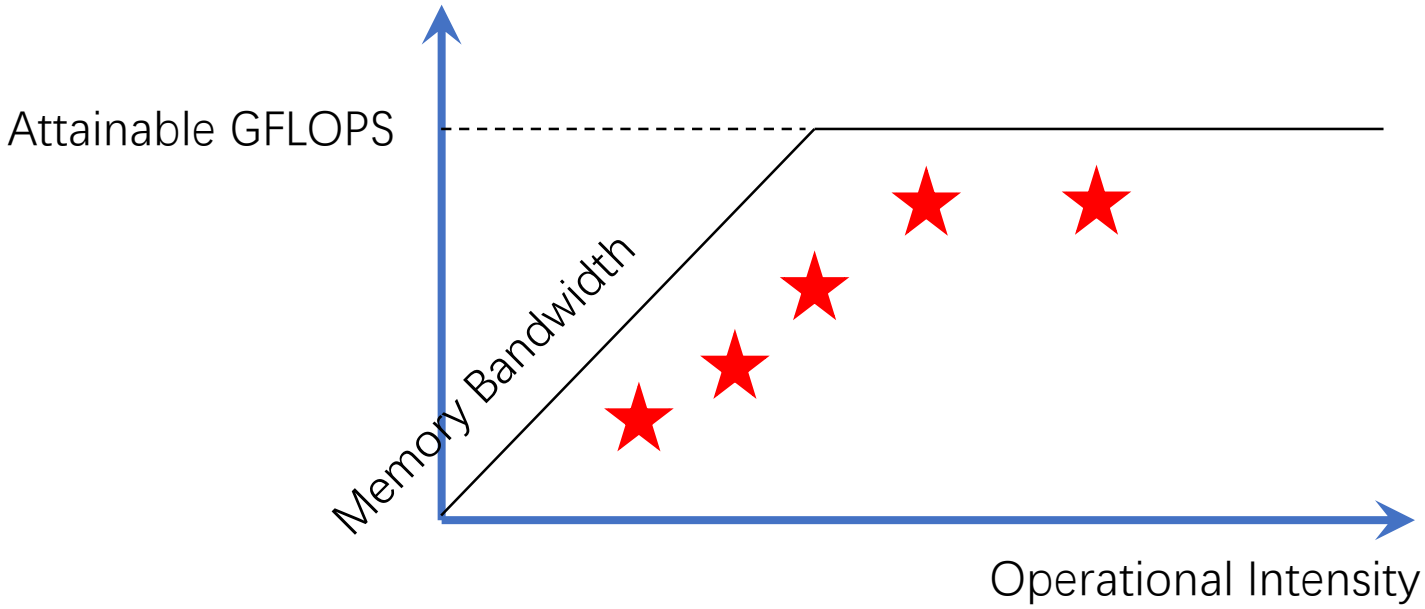
Attainable GFLOPS=Min(Peak Floating Point Performance,
Peak Memory Bandwidth × Operational Intensity)



立志成才报国裕民

Implications of Roofline Model (Cont'd)

Attainable GFLOPS=Min(Peak Floating Point Performance,
Peak Memory Bandwidth × Operational Intensity)



Optimization Methods and Insights

- Instruction-level parallelism/SIMD;
  - Perfect memory access and execute overlap
  - Achievable through software pipelining, prefetch, DMA, etc.
- Equal number of simultaneous floating-point additions and multiplications, since many computers have multiply-add instructions or because they have an equal number of adders and multipliers;
- Ensure memory affinity;
  - (Re)use local data as much as possible;
  - Distribute data to multiple HBM channels/memory banks;



立志成才报国裕民