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





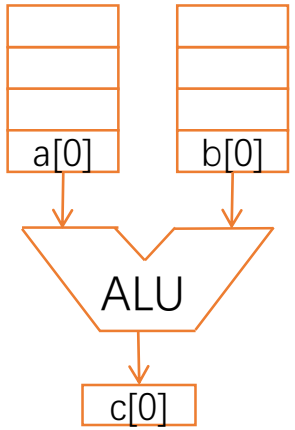
Vector ALU

- Deeper/wider pipeline/ALU

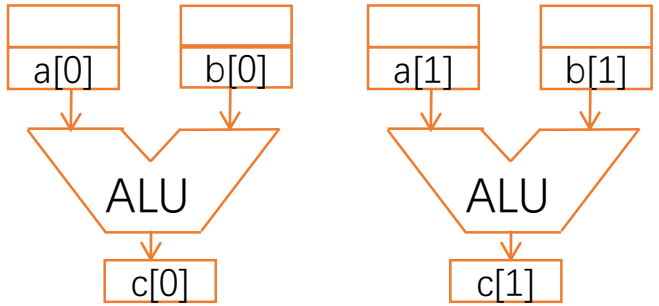


Which one to choose?

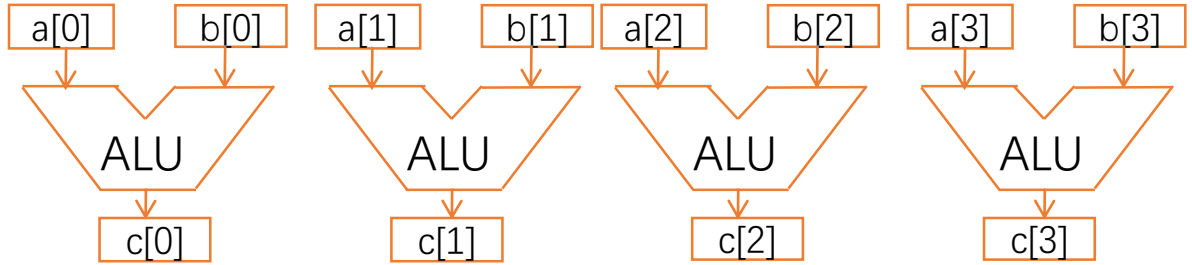
Serial, pipelined



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

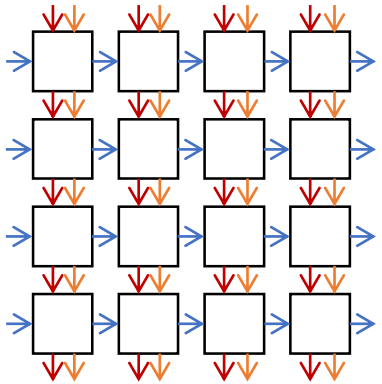
Average speed of data movement, not necessarily the physical connection ports

Depends on applications

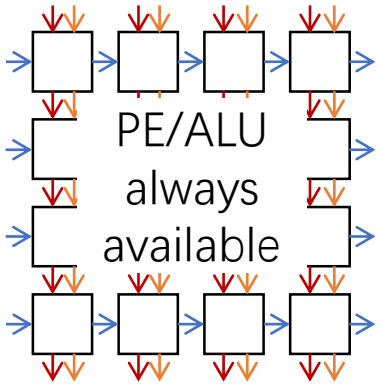
$$\frac{\text{Number of computations}}{\text{Number of data access}}$$

Reuse of data

Data always available



PE/compute-bound

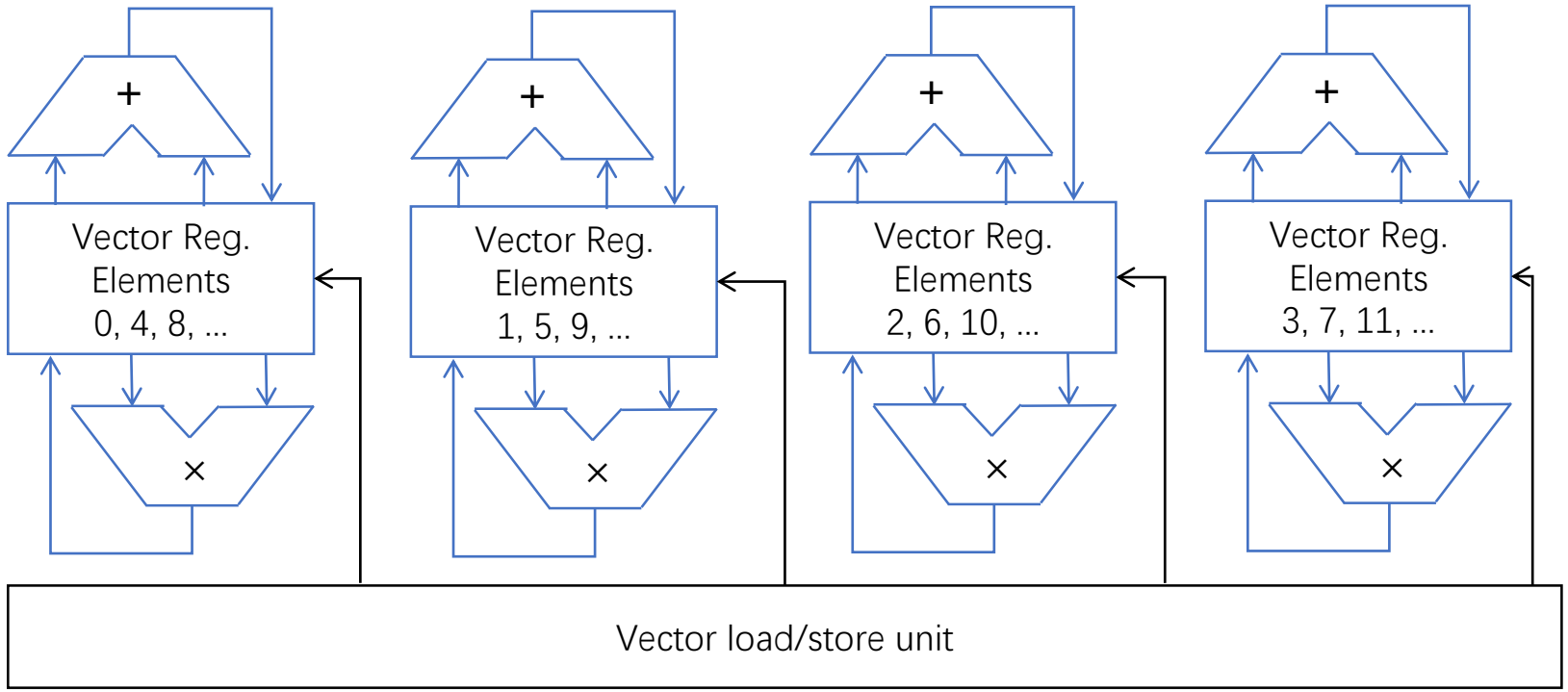


Memory/band width-bound

立志成才报国裕民

Hardware Designers' Perspective

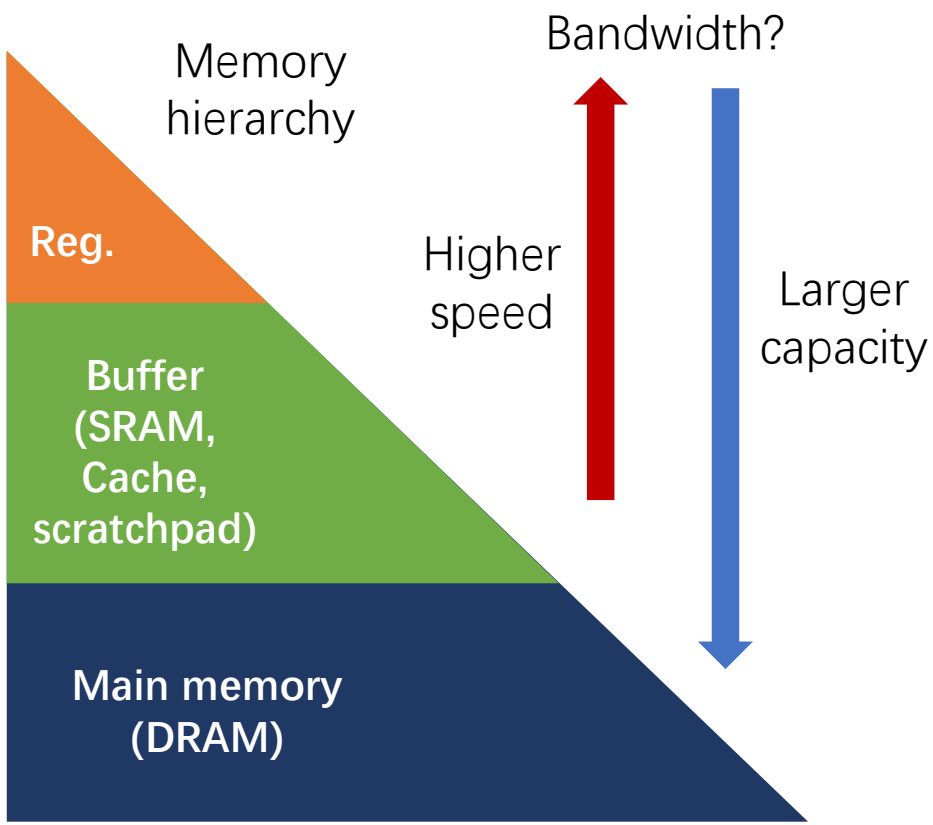
- Match the memory bandwidth and the number of compute units.

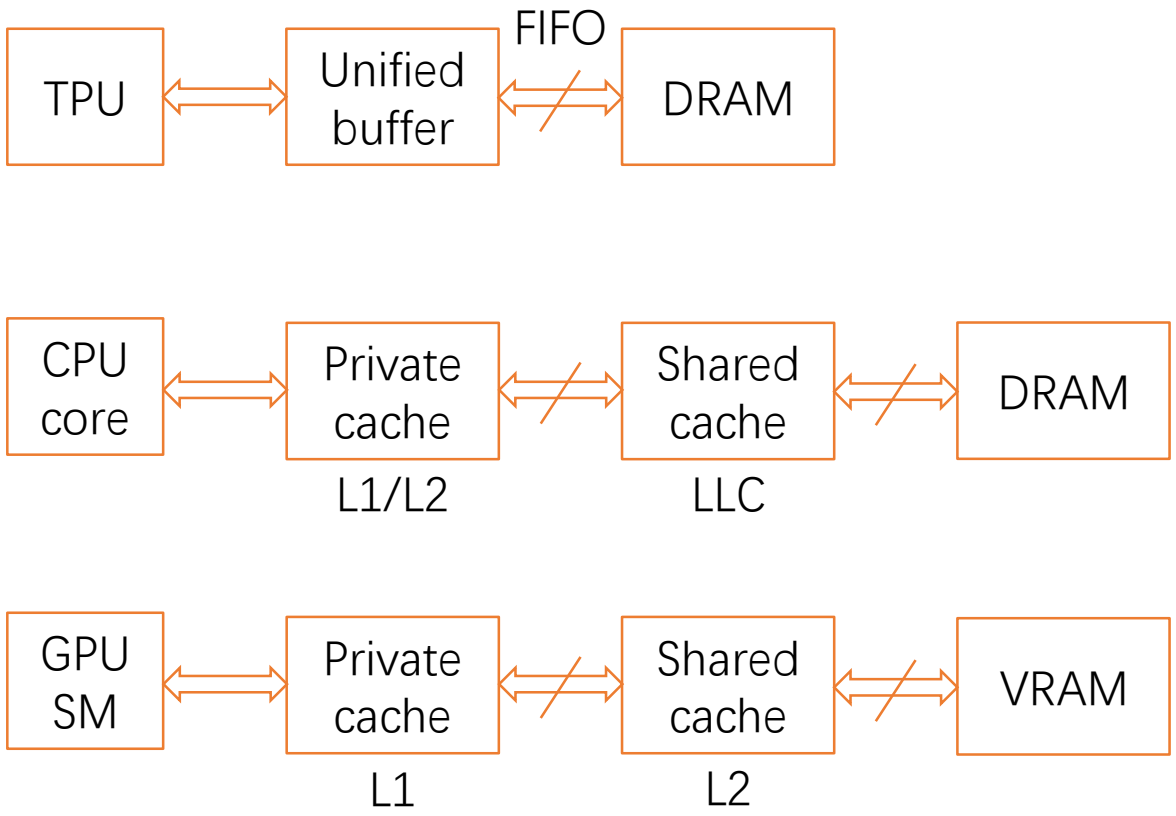


立志成才报国裕民

But …

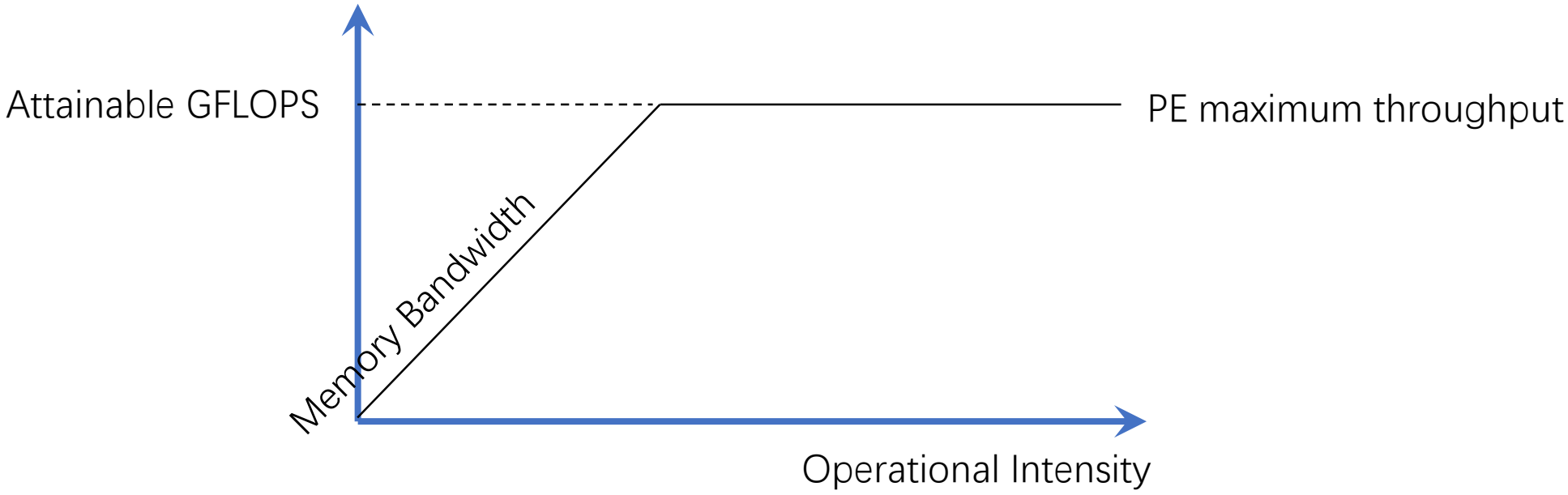
- Lower-level memory cannot match the speed/frequency of PEs/cores





Implications of Roofline Model

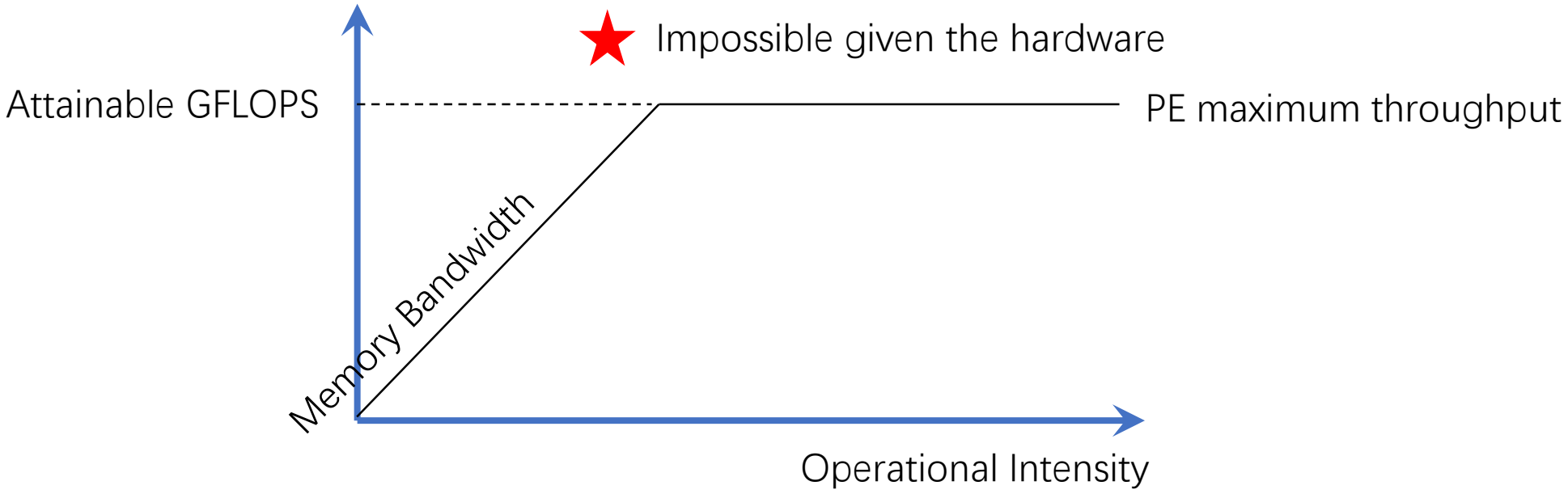
Attainable GFLOPS=Min(Peak Floating Point Performance,
Peak Memory Bandwidth × Operational Intensity)



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)

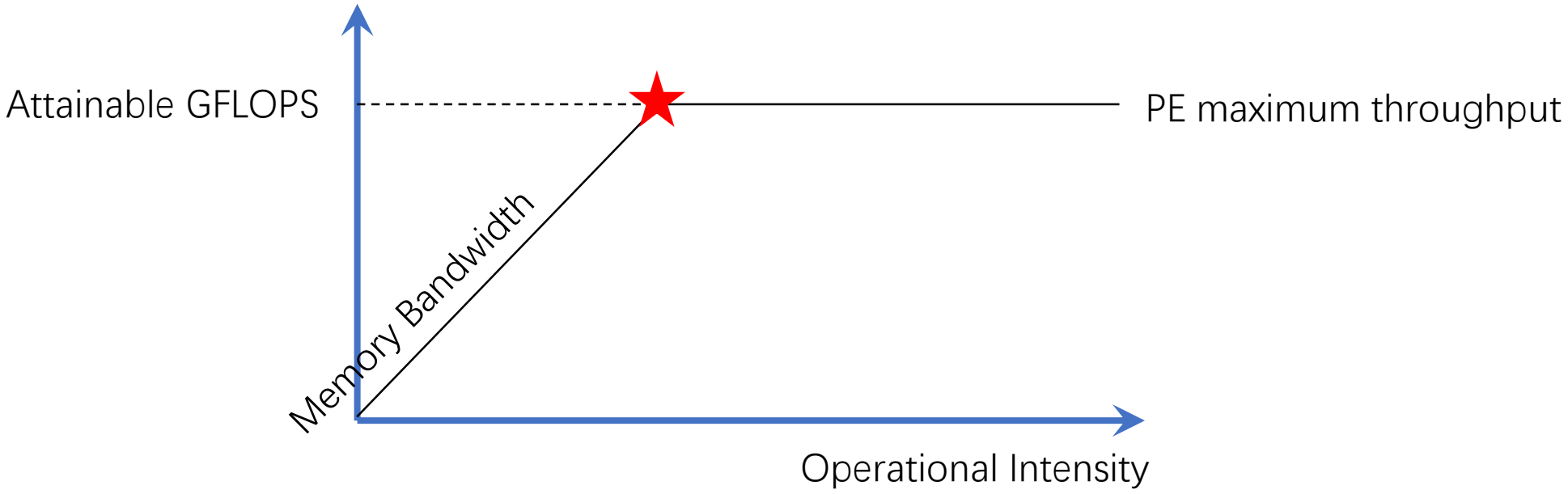


立志成才报国裕民

Implications of Roofline Model (Cont'd)

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

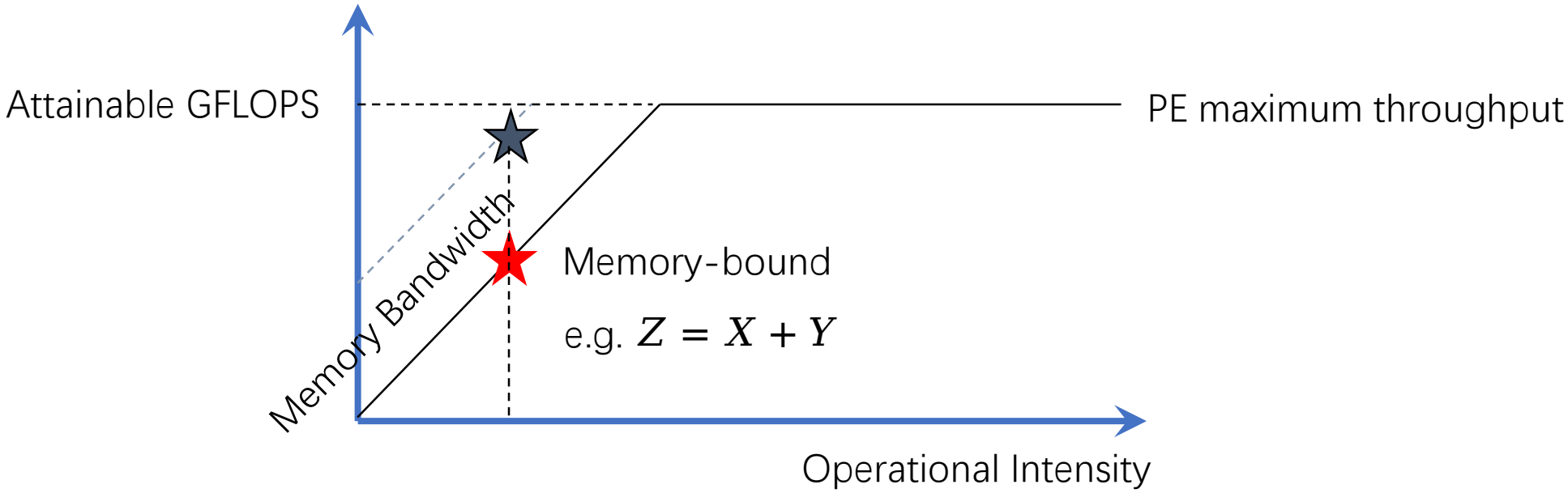
Attainable GFLOPS=Min(Peak Floating Point Performance, Peak Memory Bandwidth × Operational Intensity)



立志成才报国裕民

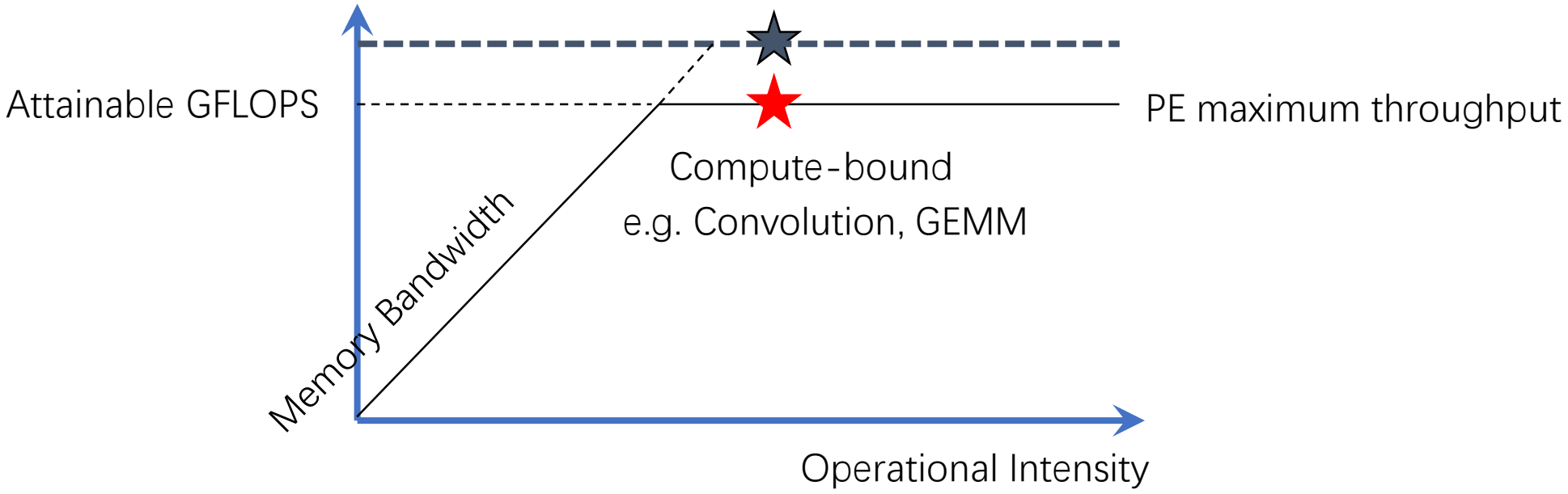
Implications of Roofline Model (Cont'd)

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



Implications of Roofline Model (Cont'd)

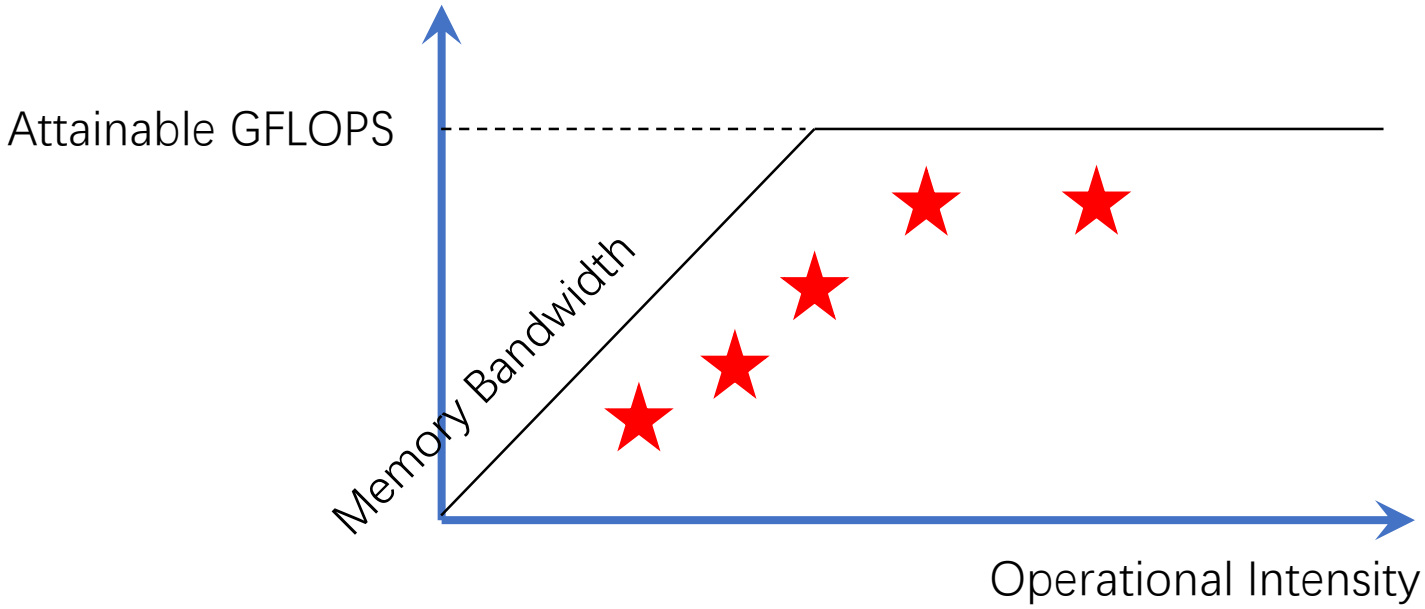
Attainable GFLOPS=Min(Peak Floating Point Performance,
Peak Memory Bandwidth × Operational Intensity)



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

Idealized Conditions Behind the Performance Roof

- Perfect buffers/cache (sufficient capacity and speed)
- Homogeneous PEs
  - CUDA cores and Tensor Cores have separate precision-specific compute roofs; one aggregate roofline model is insufficient
- Machines always keep peak bandwidth



立志成才报国裕民

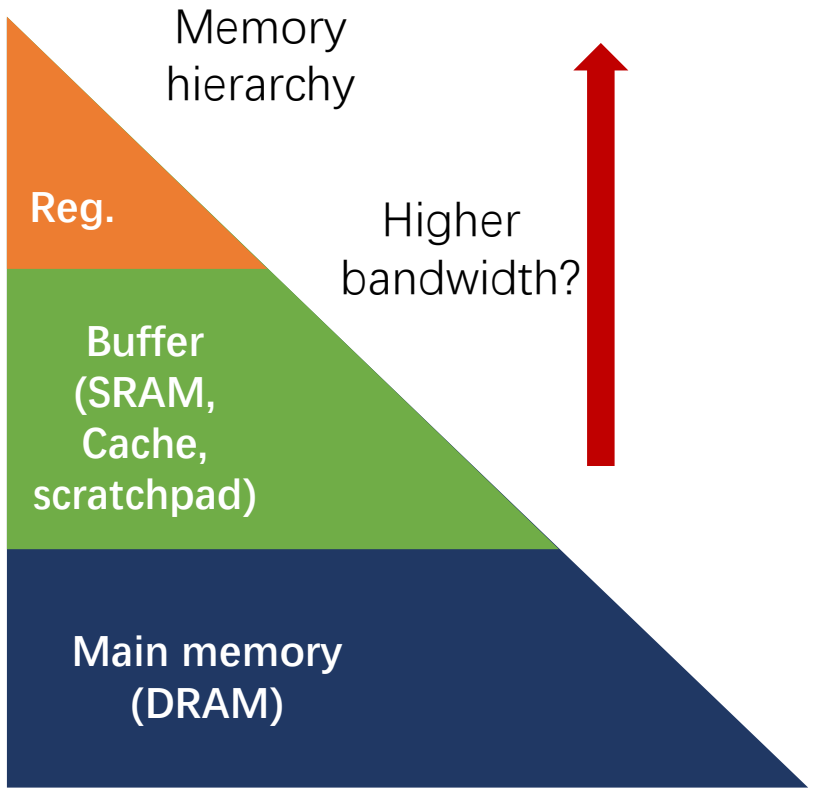
Idealized Conditions Behind the Performance Roof

- Perfect buffers/cache (sufficient capacity and speed)
- Homogeneous PEs
  - CUDA cores and Tensor Cores have separate precision-specific compute roofs; one aggregate roofline model is insufficient
- Machines always keep peak bandwidth



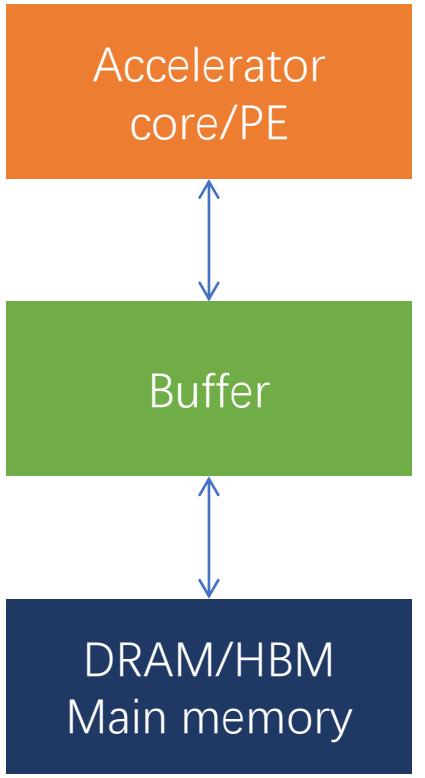
立志成才报国裕民

Buffer/Cache Bottlenecks



Attainable GFLOPS =

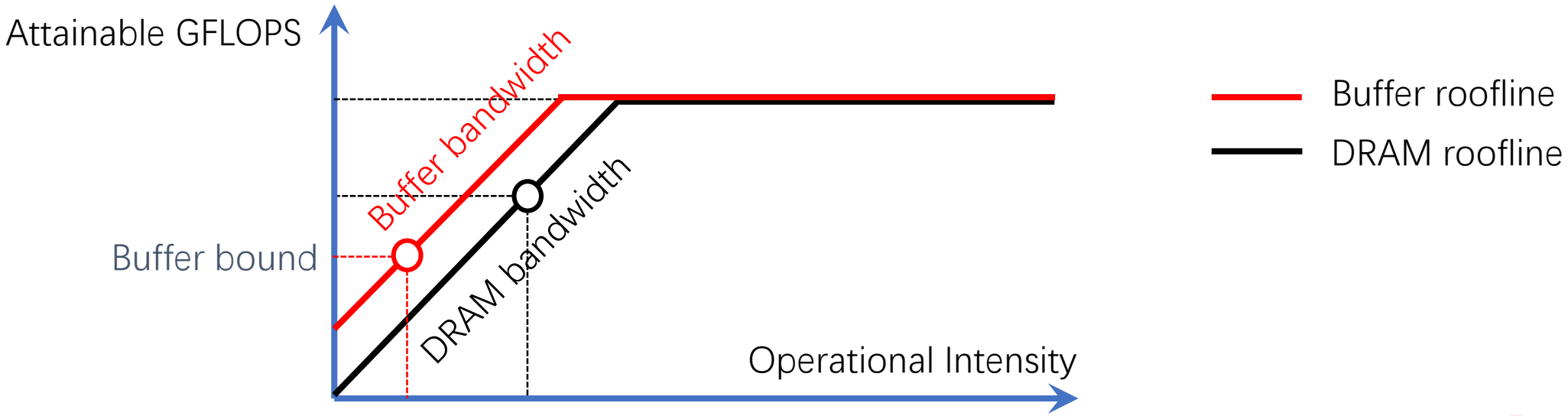
Min
- Peak Floating Point Performance
- Peak DRAM Bandwidth × Operational Intensity (DRAM)
- Peak Memory Bandwidth × Operational Intensity (buffer: can be cache, scratchpad memory, etc.)



立志成才报国裕民

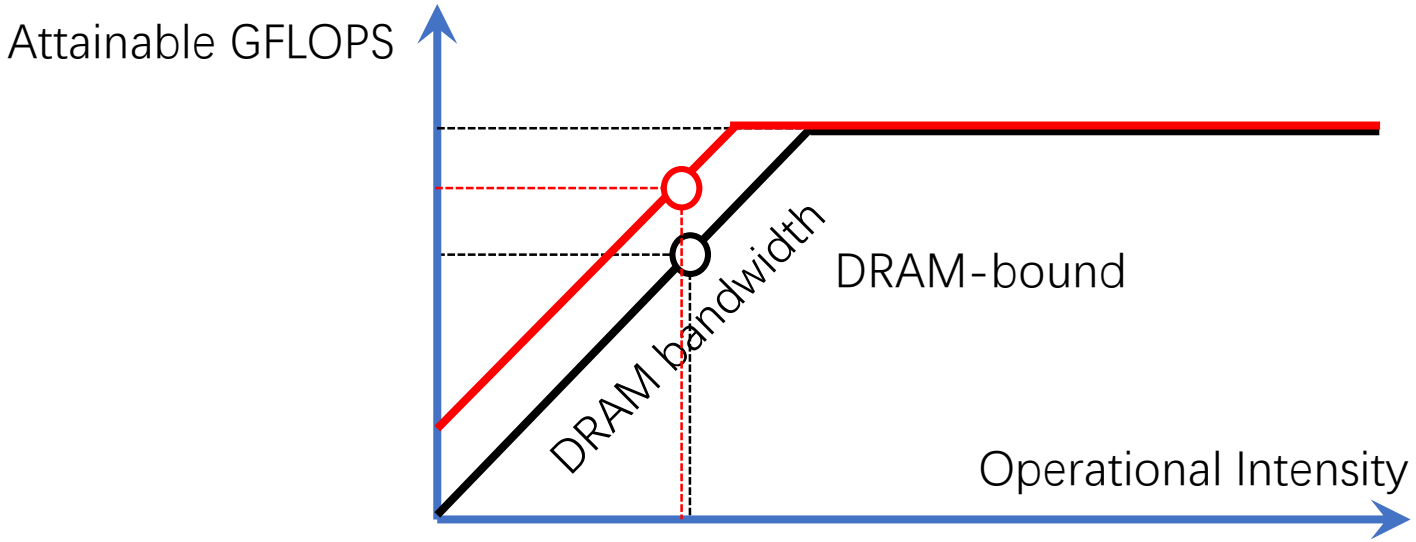
Hierarchical Roofline Model

Attainable GFLOPS = min
- Peak Floating Point Performance
- Peak DRAM Bandwidth × Operational Intensity (DRAM)
- Peak Memory Bandwidth × Operational Intensity (buffer: can be cache, scratchpad memory, etc.)



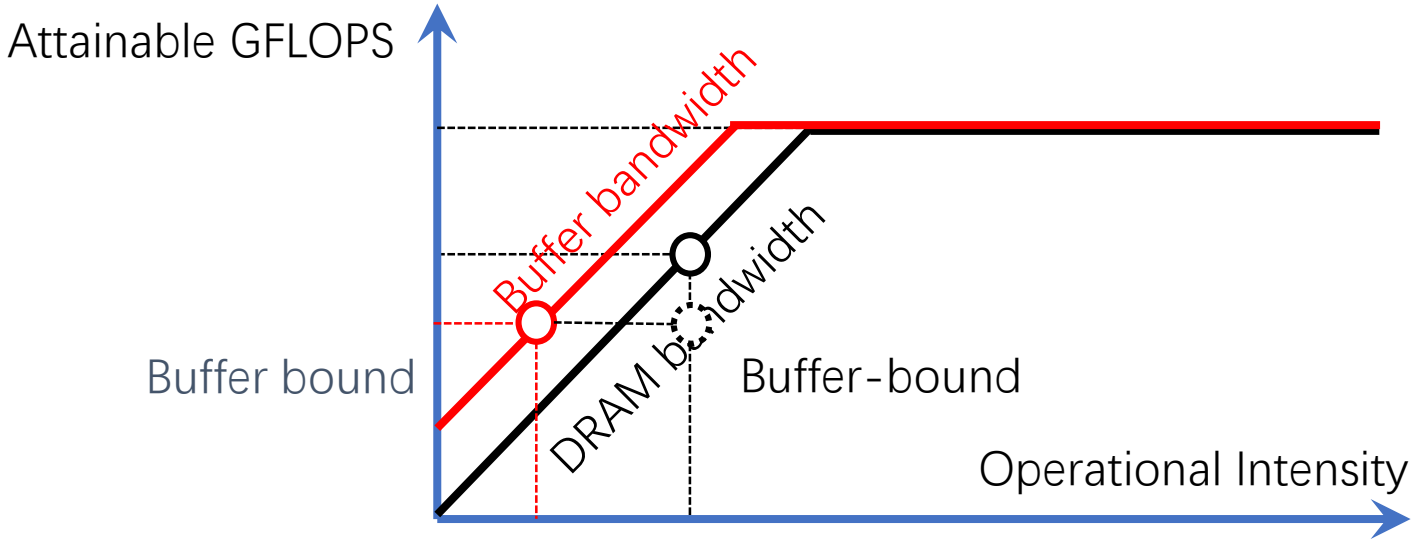
立志成才报国裕民

Bottlenecks Analysis



Buffer roofline
DRAM roofline

Little buffer data reuse (streaming)



Buffer roofline
DRAM roofline

Good buffer data reuse!

立志成才报国裕民

Potential Improvements

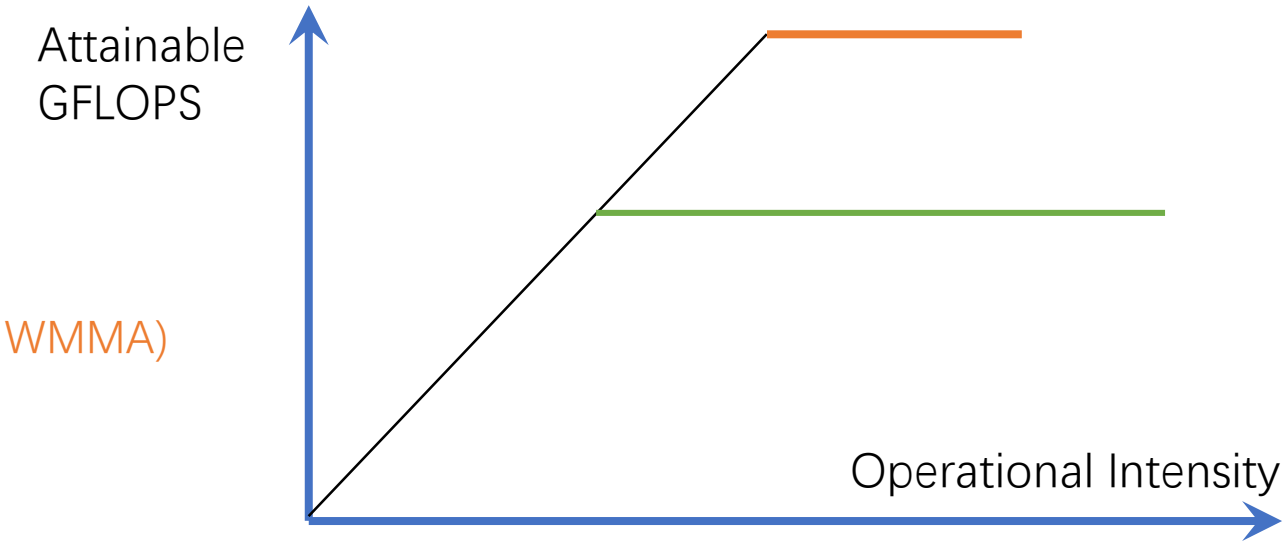
- From hardware side, improve buffer bandwidth
  - Multi-bank, and bank-aware address mapping to reduce bank conflict;
  - Cache replication for frequently accessed data, at the cost of capacity;
  - Wider and more buffer ports;
- From software side, improve data reuse
  - Reuse data at a level closer to the PE/computation;
  - E.g. loop tiling/blocking, kernel fusion, better scheduling, etc.



立志成才报国裕民

Idealized Conditions Behind the Performance Roof

- Perfect buffers/cache (sufficient capacity and speed)
- Homogeneous PEs
  - CUDA cores and Tensor Cores have separate precision-specific compute roofs; one aggregate roofline model is insufficient
- Machines always keep peak bandwidth



Consider NVIDIA Volta GPU

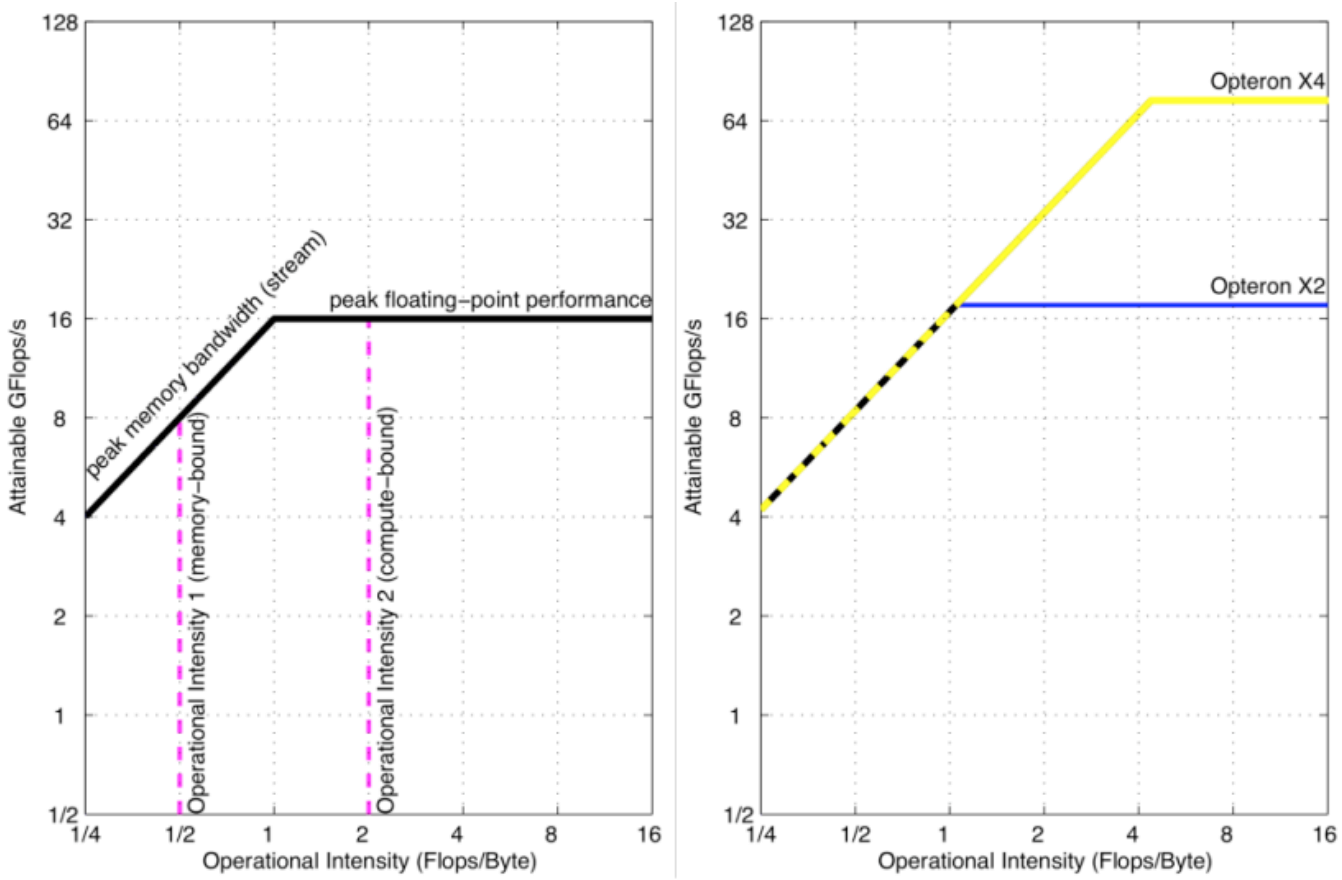
100-ish TFLOPS FP16 tensor core (Z = AB+C, WMMA)

15 TFLOPS FP32 FMA (z = ax+y)

Both SIMD(T)

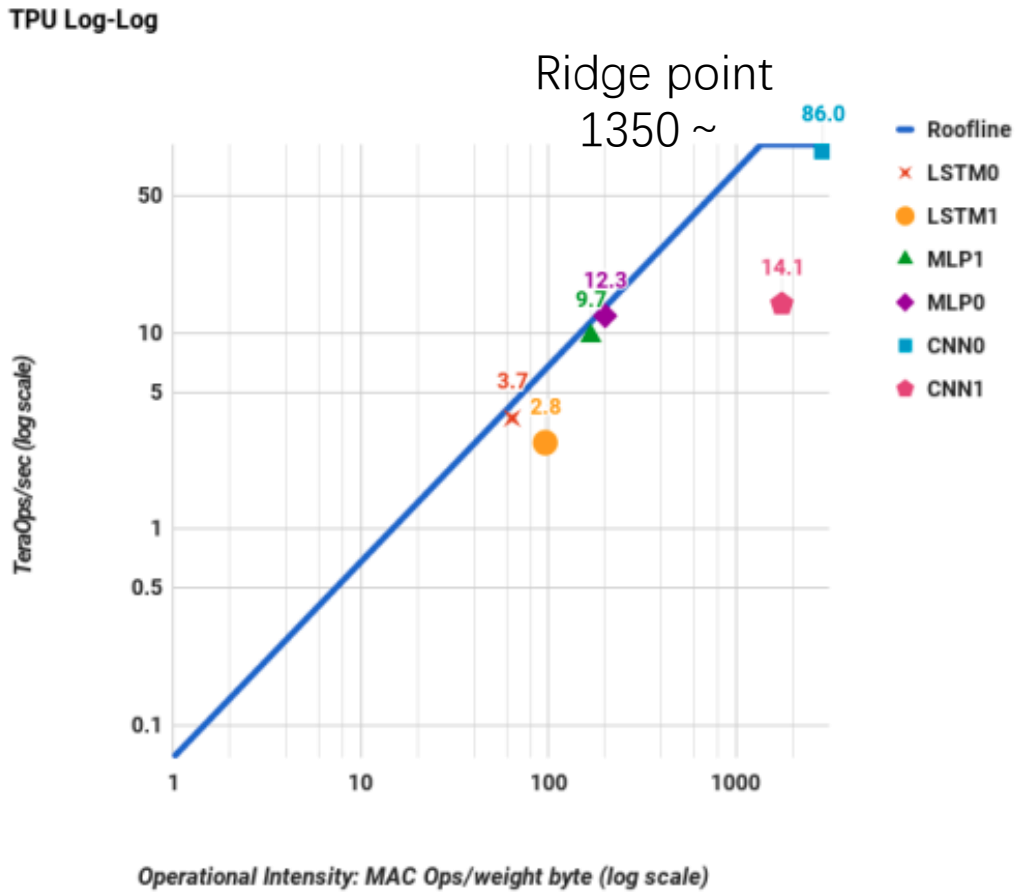
立志成才报国裕民

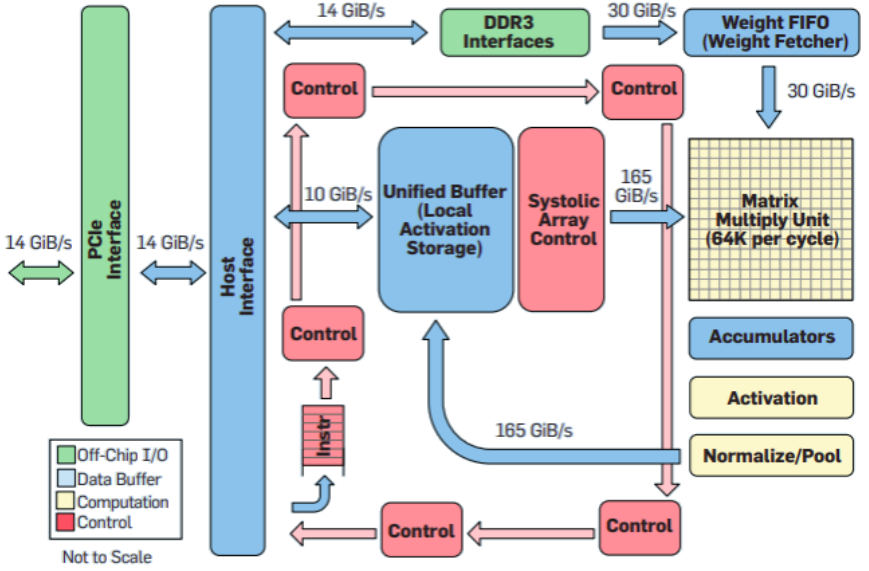
Roofline Model for CPU, GPU, etc.



Roofline Model for TPUv1

- For TPU, operational intensity re-defined as int. operations per byte of weights read.



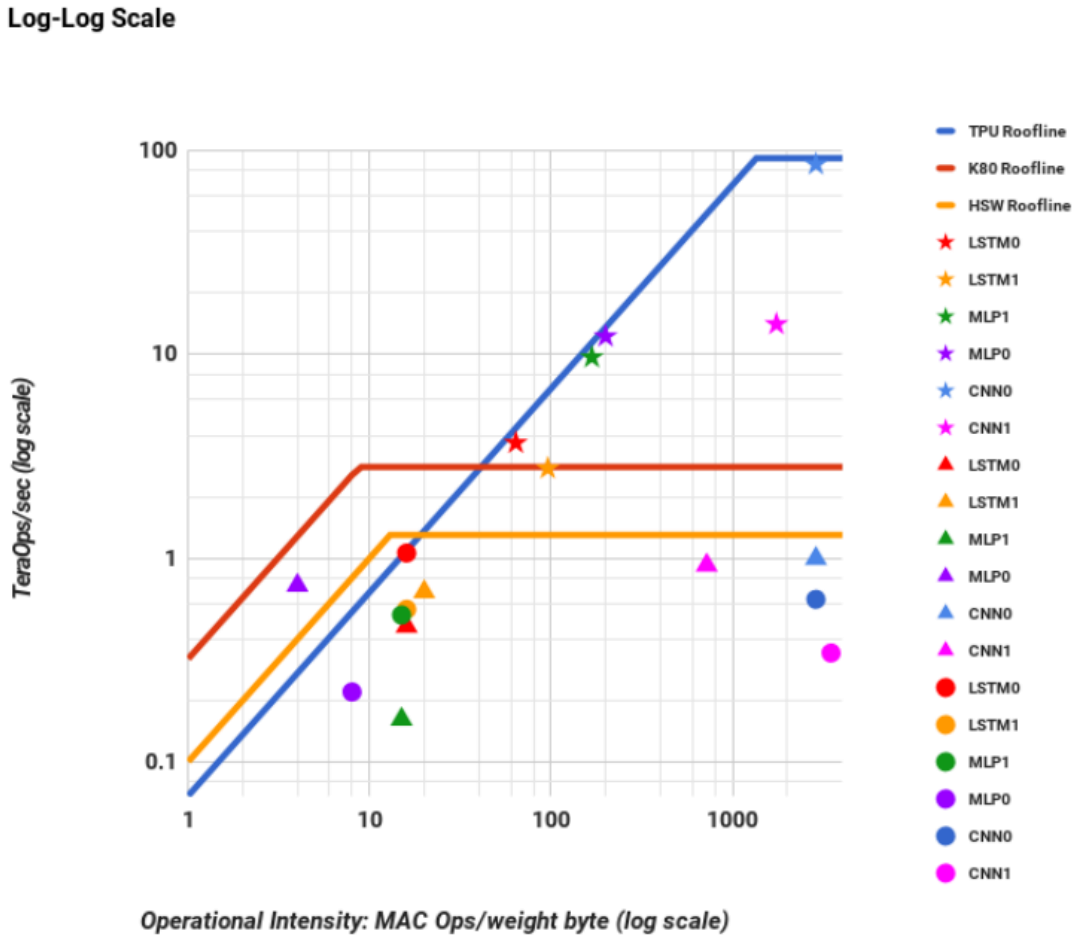


CNN1 is below the roofline:

- RAW stall takes 23% of the clock cycles;
- Operational intensity is around 32 for the 4 fully connected layers

Comparison with CPU and GPU

- For TPU, operational intensity re-defined as int. operations per byte of weights read.



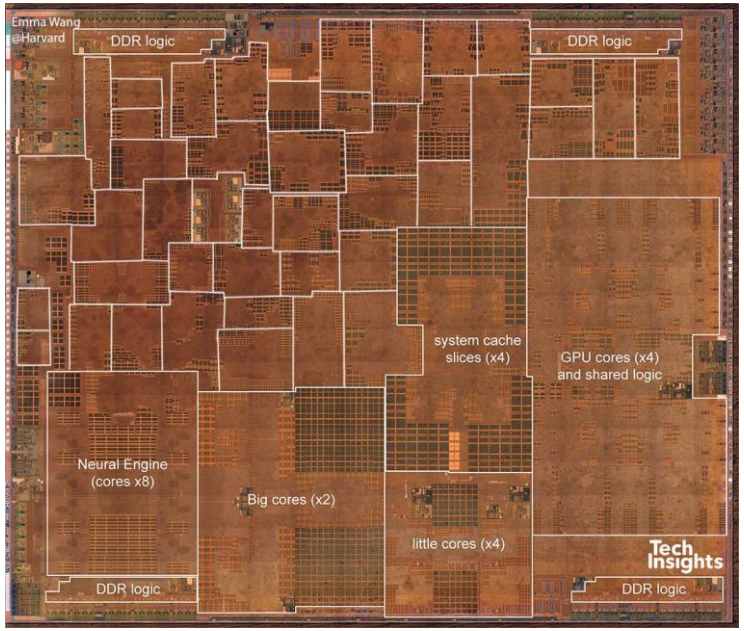
Type	*Batch*	*99th% Response*	*Inf/s (IPS)*	*% Max IPS*
CPU	16	7.2 ms	5,482	42%
CPU	64	21.3 ms	13,194	100%
GPU	16	6.7 ms	13,461	37%
GPU	64	8.3 ms	36,465	100%
TPU	200	7.0 ms	225,000	80%
TPU	250	10.0 ms	280,000	100%

Table 4. 99th% response time and per die throughput (IPS) for MLP0 as batch size varies for MLP0. The longest allowable latency is 7 ms. For the GPU and TPU, the maximum MLP0 throughput is limited by the host server overhead. Larger batch sizes increase throughput, but as the text explains, their longer response times exceed the limit, so CPUs and GPUs must use less-efficient, smaller batch sizes (16 vs. 200). They run 2.3X – 2.7X slower than if response time was unbound, but the for the more deterministic TPU, the slowdown from the 99th% response-time limit is just 1.2X.

Comparison: TPU vs. CPU vs. GPU

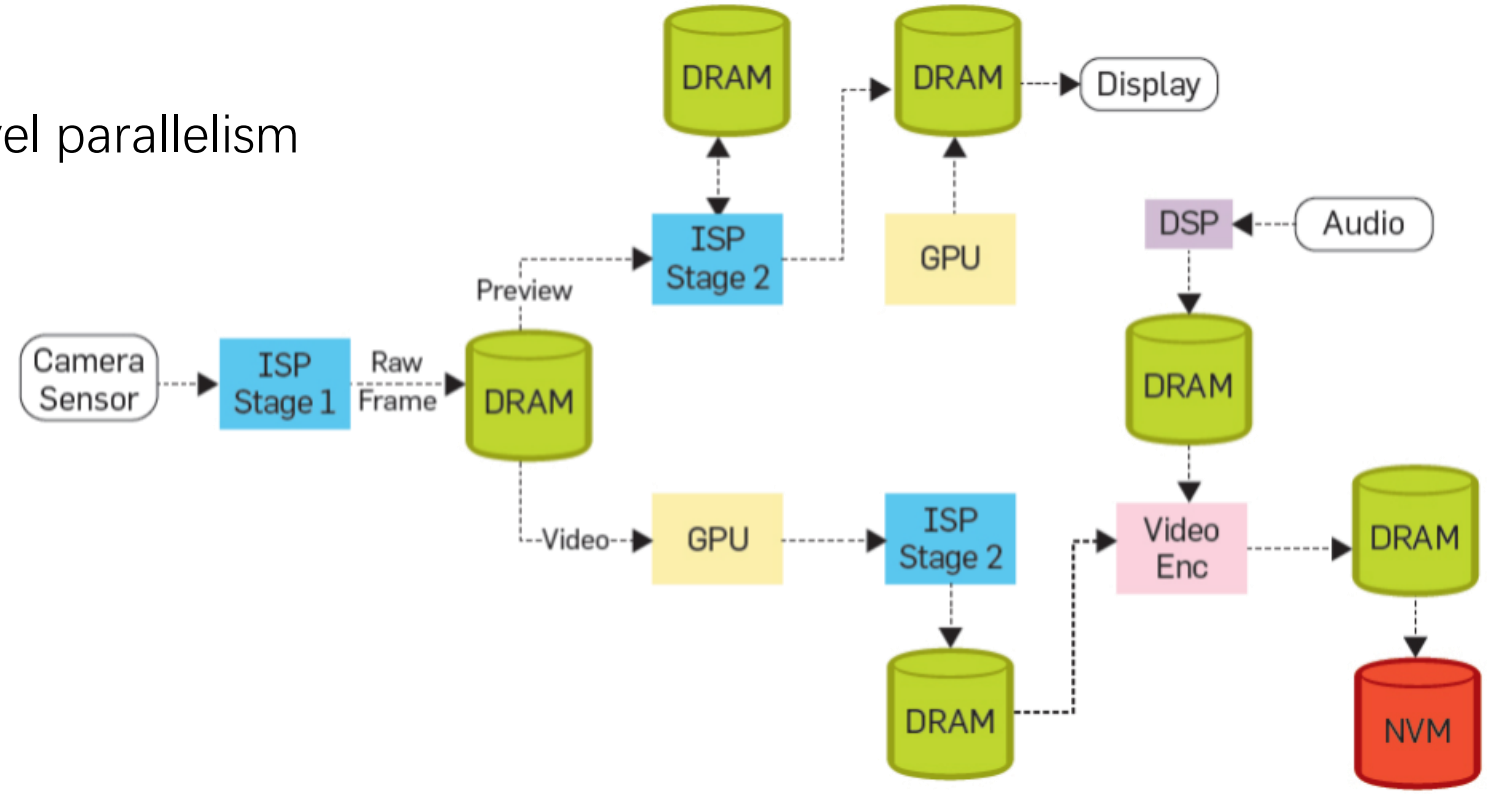
Parallelism – Evolution

- Bit-level parallelism: multi-bit processing
- Multi-issue: instruction-level parallelism
- Multi-core: thread-level parallelism
- Data-level parallelism: GPU
- System-on-chip (SoC): accelerator-level parallelism



2019, Apple A12 SoC, TechInsights

A smartphone performing video recording
The usage of the SoC



Two Billion Devices and Counting: An Industry Perspective on the State of Mobile Computer Architecture, IEEE Micro'2018.

立志成才报国裕民

Parallelism – Modern SoC

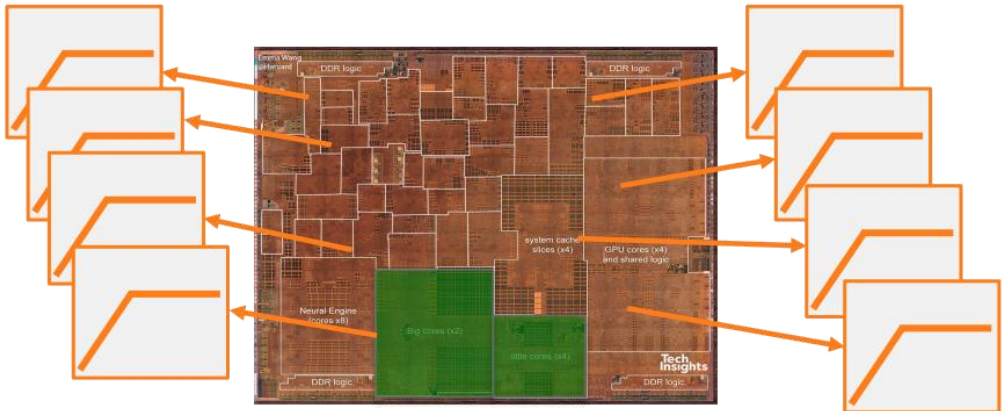
	CPU	Display	Media Scaler	GPU	ISP	JPEG	Pixel Visual Core	Video decoder	Video Encoder
Photo Enhancing	√	√		√	√	√	√		
Video Capture	√	√		√	√				√
Video Capture HDR	√	√		√	√				√
Video Playback	√	√	√	√				√	
Image Recognition	√	√	√	√					



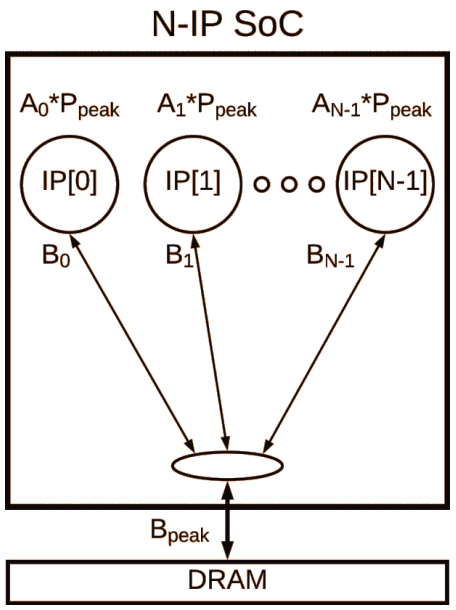
Mark Hill and Vijay Janapa Reddi, Gables: A Roofline Model for Mobile SoCs, HPCA' 2019

立志成才报国裕民

Roofline of Modern SoC



2019 Apple A12 w/ 42 accelerators



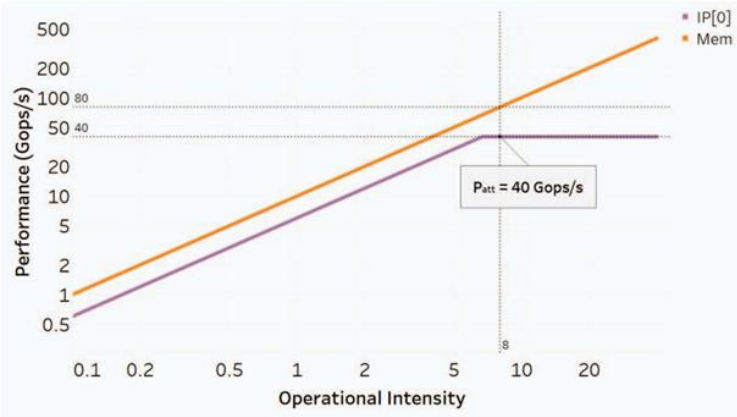
IP_i	Max comp.	Comp. time	Bandwidth	Workload	Op. Intensity	Mem. time
IP_i	$A_i * P_{peak}$	$A=f_i/(A_i * P_{peak})$	B_i	f_i	I_i	$f_i/I_i/B_i$
…	…	…	…	…	…	…
DRAM	-	-	P_{peak}	-	-	$(\sum f_i/I_i)/P_{peak}$

Assume parallel execution of all IPs, max. attainable FLOPs is the minimum of each active IP's scaled roofline bound or the shared DRAM bound.

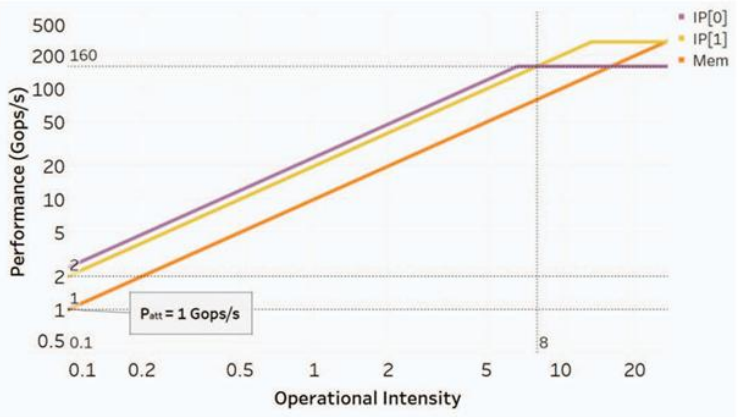
Mark Hill and Vijay Janapa Reddi, Gables: A Roofline Model for Mobile SoCs, HPCA 2019

立志成才报国裕民

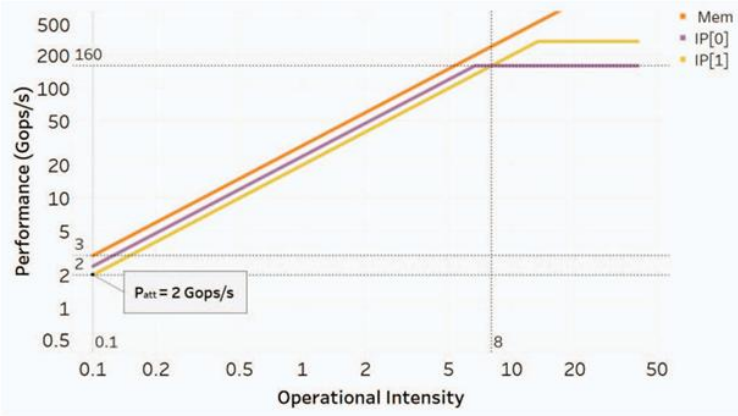
Roofline of Modern SoC



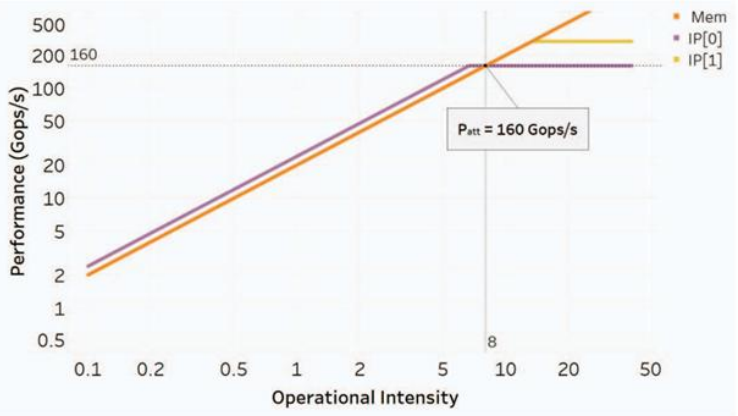
(a) With initial parameters, performance matches CPU's 40 Gops/s at assumed operational intensity $I_0 = 8$, as the GPU is not used.



(b) Changing f from 0 to 0.75 to use the GPU drops performance to 1.3 Gop/s (rounded to 1) as the GPUs poor data reuse (operational intensity $I_1 = 0.1$ at far left) makes memory bandwidth inadequate.



(c) Increasing memory bandwidth B_{peak} from 10 to 30 GB/s increases performance to only 2 Gops/s.



(d) Increasing GPU data reuse I_1 from 0.1 to 8 and decreasing B_{peak} to a sufficient 20 GB/s achieves 160 Gops/s performance with all three rooflines equal at $I = 8$, achieving a balanced design.

Assume P_{peak}=40 Gop/s, B_{peak}= 10 GB/s
A_0=1, A_1=5, B_0 =6, B_1=15, I_0=8, I_1=0.1

For (a), f_0=1, f_1=0

For (b), f_0=0.25, f_1=0.75

For (c), improve B_{peak}= 30 GB/s

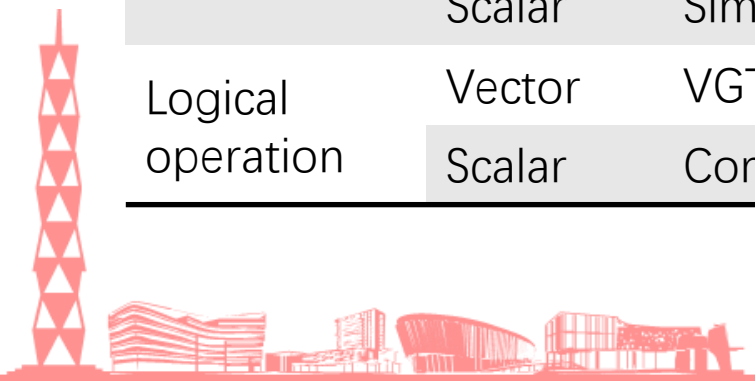
For (d), improve I_1=8

Mark Hill and Vijay Janapa Reddi, Gables: A Roofline Model for Mobile SoCs, HPCA 2019

ISA Design – Instructions

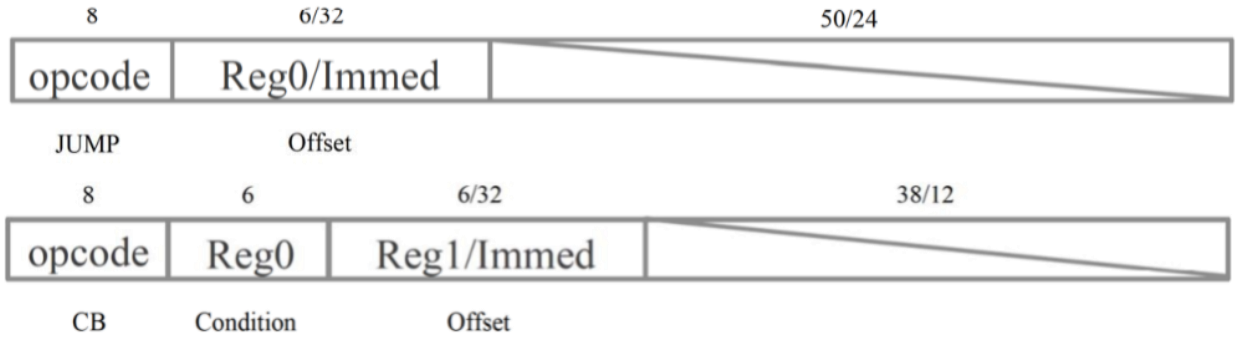
- Cambricon ISA example, adopted RISC philosophy

Type		Example	Object
Control		JUMP, CB (Conditional branch)	Register (Scalar), Immediate
Data Movement	Matrix	MLOAD/MSTORE/MMOVE	Register (Address/size, scalar), Immediate
	Vector	VLOAD/VSTORE/VMOVE	Register (Address/size, scalar), Immediate
	Scalar	SLOAD/SSTORE/SMOVE	Register (Scalar), Immediate
Compute	Matrix	MMV, VMM, MMS, MAM, MSM	Register (Address/size, scalar)
	Vector	VAV, VSV, VMV, VDV, VEXP, VLOG, IP, RV, VMAX, VMIN	Register (Address/size, scalar)
	Scalar	Similar to Vector	Register (Scalar)
Logical operation	Vector	VGT, VE, VAND, VOR, VNOT, VGTM	Register (Address/size, scalar)
	Scalar	Compare and logical operation	Register (Scalar), Immediate

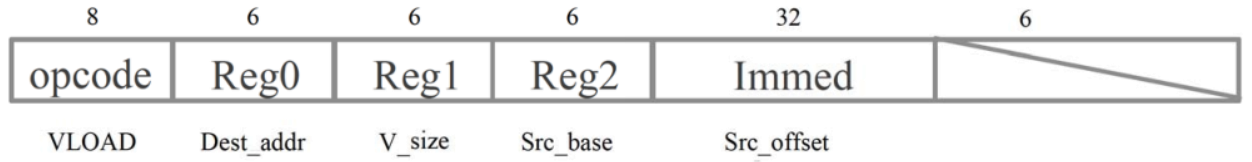


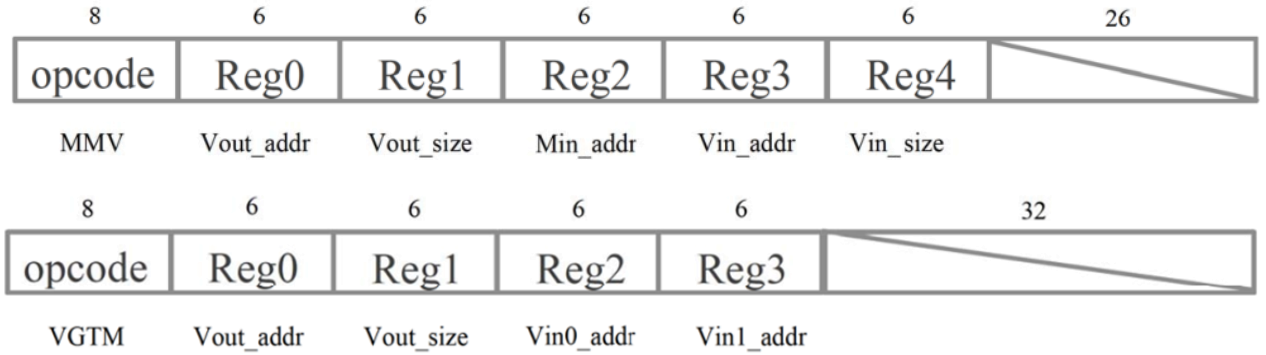
立志成才报国裕民

Cambricon ISA



- Jump (call a function)
- CB (conditional branch, for loop)
- LOAD/STORE: data exchange between main memory and on-chip memory
  - XLOAD/XSTORE, X = M (Matrix)/ V (Vector)/ S (Scalar)
- MOVE: on-chip data movement
  - MMOVE, VMOVE, SMOVE
- Compute:
  - XAX, XSX, XMX, XDX X = M (Matrix)/ V (Vector)/ S (Scalar)
  - Elementwise: VEXP, VLOG, RV
  - IP, OP, VMAX/VMIN
  - Scalar A/S/M/D, transcendental functions
- Logic: Vector: VGT, VE, VAND/OR/NOT, VGTM





VGTM: Vout[i] = (Vin0[i] > Vin1[i])?Vin0[i] : Vin1[i] (Max pooling)

Cambricon ISA (Cont'd)

- Assembly code for a FC layer

Register	Content	Register	Content
$0	Size of input vector	$3	Address of input vector
$1	Size of output vector	$4	Address of weights
$2	Size of weight matrix	$5	Address of bias
$7~$10	Address of temporary results	$6	Address of output vector

```
VLOAD    $3, $0, #100 (IMM.)   // Load input vector from address 100 to on-chip mem. $3 (addr.)
MLOAD    $4, $2, #300          // Load weight matrix from address 300 to on-chip mem. $4
MMV      $7, $1, $4, $3, $0    // WX, results is stored to addr. $7
VAV      $8, $1, $7, $5        // t=WX+b, results stored to addr. $8
VEXP     $9, $1, $8            // exp(t), results stored to addr. $9
VAS      $10, $1, $9, #1       // 1+exp(t), results stored to addr. $10
VDV      $6, $1, $9, $10       // y = exp(t)/[1+exp(t)], sigmoid, results stored to addr. $6
VSTORE   $6, $1, #200          // Store the output vector to address 200
```

立志成才报国裕民

Cambricon ISA (Cont'd)

- Assembly code for a pooling window

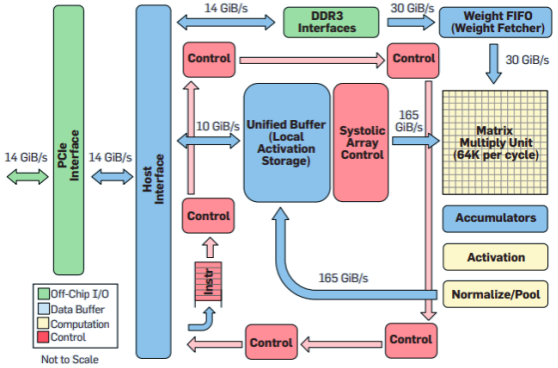
```
        VLOAD   $6, $1, #100 (IMM.)   // Load input vector from address 100 to on-chip mem. $6 (addr.)
        SMOVE   $5, $3                // Initialize vertical loop times ($5) with window size ($3)
L0:     SMOVE   $4, $3                // Initialize horizontal loop times ($4) with window size ($3)
L1:     VGTM    $7, $0, $6, $7        // output[m]=(input[c][r][m]>output[m])? input[c][r][m]:output[m]
        SADD    $6, $6, $0            // Update the address
        SADD    $4, $4, #-1           // Horizontal loop minus 1
        CB      #L1, $4               // Go to L1 if loop number>0
        SADD    $6, $6, $8            // Update the address to the next row
        SADD    $5, $5, #-1           // Vertical loop minus 1
        CB      #L0, $5               // Go to L1 if loop number>0
        VSTORE  $7, $2, #200          // Store to address 200
```

Register	Content
$0	Number of channels
$1	Size of input feature map
$2	Size of output fmap
$3	Pooling window size
$4	Horizontal loop number
$5	Vertical loop number
$6	Address of input fmap
$7	Address of output fmap
$8	Vertical step

立志成才报国裕民

TPU ISA

- Based on CISC
- Coarse-grained, domain-specific instructions
- CPI 10~20
- Hide the execution of the other instructions by overlapping with MatrixMultiply
- Key instructions
  - **Read_Host_Memory** reads data from the CPU host memory into the Unified Buffer (UB).
  - **Read_Weights** reads weights from Weight Memory into the Weight FIFO as input to the Matrix Unit.
  - **MatrixMultiply/Convolve** causes the Matrix Unit to perform a matrix multiply or a convolution from the Unified Buffer into the Accumulators. A matrix operation takes a variable-sized B*256 input, multiplies it by a 256×256 constant weight input, and produces a B*256 output, taking B pipelined cycles to complete.
  - **Activate** performs the nonlinear function of the artificial neuron, with options for ReLU, Sigmoid, and so on. Its inputs are the Accumulators, and its output is the Unified Buffer. It can also perform the pooling operations needed for convolutions using the dedicated hardware on the die, as it is connected to nonlinear function logic.
  - **Write_Host_Memory** writes data from the Unified Buffer into the CPU host memory.



Memory Consideration

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



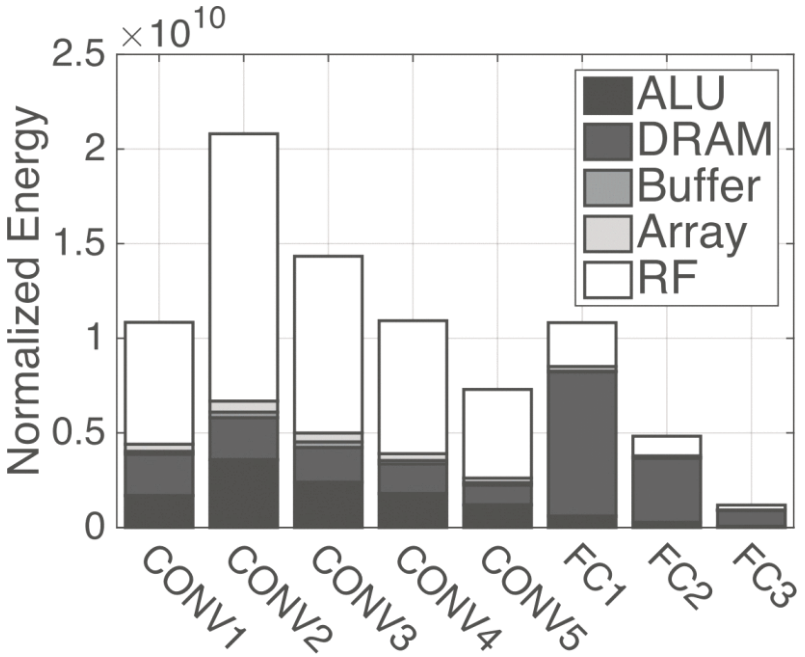
Computer Architecture - EE219

faculty.sist.shanghaitech.edu.cn

立志成才报国裕民

Memory Considerations

Memory	
Cache	(64 bit)
8 KB	10 pJ
32 KB	20 pJ
1 MB	100 pJ
DRAM	1.3-2.6 nJ
FP	
FADD	
16 bit	0.4 pJ
FMULTI	
16 bit	1 pJ

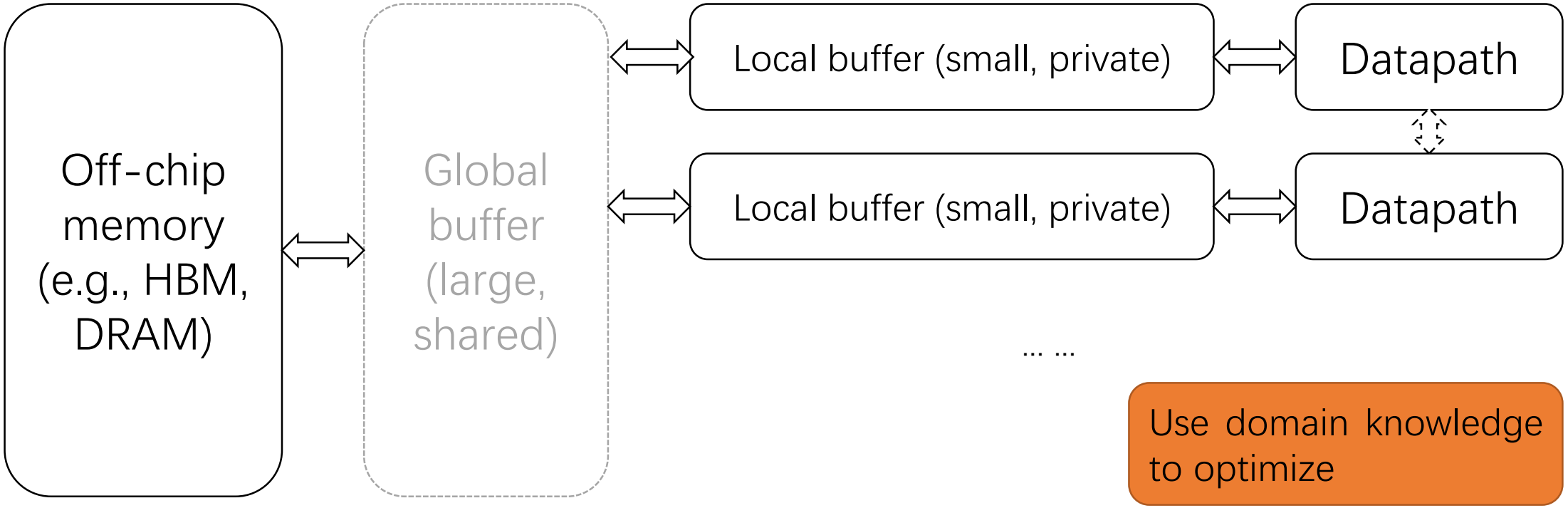


On-chip memory takes generally more than 40% of chip area.

Y. -H. Chen, J. Emer and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in 2016 ISCA.

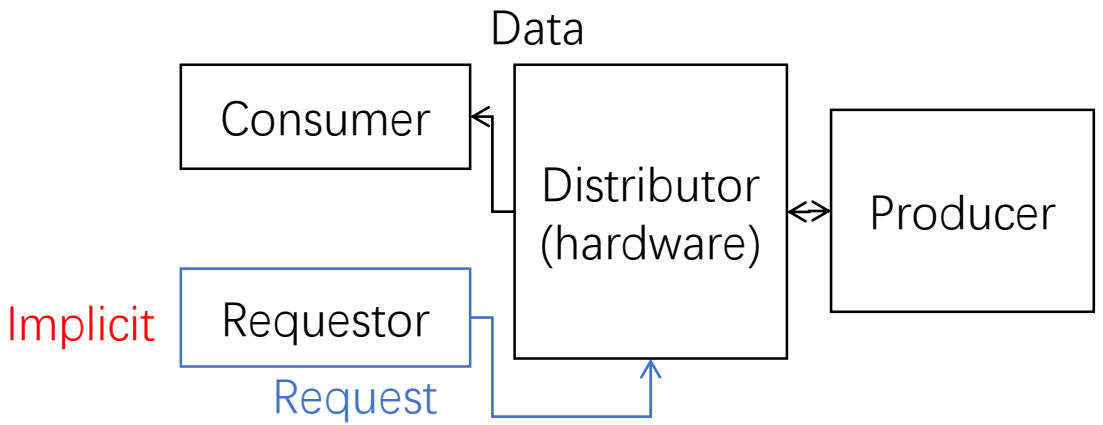
Ideally, …

- Data is ready when the datapath (functional unit or PE) needs it
- Removing data from a buffer when it is not used
- Reduce data movement as much as possible



Old Wisdom

- Cache to coordinates data movements



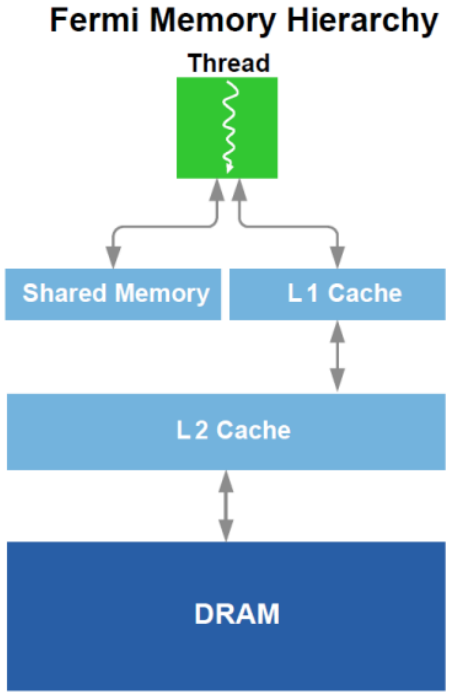
- Easy to program
- Workload agnostic
- Area/energy overhead

- Cache controller receives request and sends it to main memory if necessary.
- Cache provides data to ALU/datapath

立志成才报国裕民

New Wisdom

- Explicitly-managed memory, e.g., shared memory in NVIDIA GPUs
- In Fermi architecture, 64 KB configurable shared memory and L1 cache
- `cudaDeviceSetCacheConfig()`
- `cudaFuncSetCacheConfig()`
  - 48 KB shared memory with 16 KB L1 cache
  - 16 KB shared memory with 48 KB L1 cache
  - Shared memory: memory accesses are known beforehand
  - Synchronize threads by `__syncthreads()`



立志成才报国裕民

Use of Shared Memory in GPU

```
#include <stdio.h>

__global__ void staticReverse(int *d, int
n)
{
  __shared__ int s[64];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
   __syncthreads();
  d[t] = s[tr];
}
```

```
int main(void)
{
  const int n = 64; int a[n], r[n], d[n];
  for (int i = 0; i < n; i++) {
    a[i] = i;
    r[i] = n-i-1;
    d[i] = 0;
  }
  int *d_d; cudaMalloc(&d_d, n * sizeof(int));
  // run version with static shared memory
  cudaMemcpy(d_d, a, n*sizeof(int), cudaMemcpyHostToDevice);
  staticReverse<<<1,n>>>(d_d, n);
  cudaMemcpy(d, d_d, n*sizeof(int), cudaMemcpyDeviceToHost);
  for (int i = 0; i < n; i++)
    if (d[i] != r[i]) printf("Error: d[%d]!=r[%d] (%d, %d)/n", i, i, d[i], r[i]);
}
```

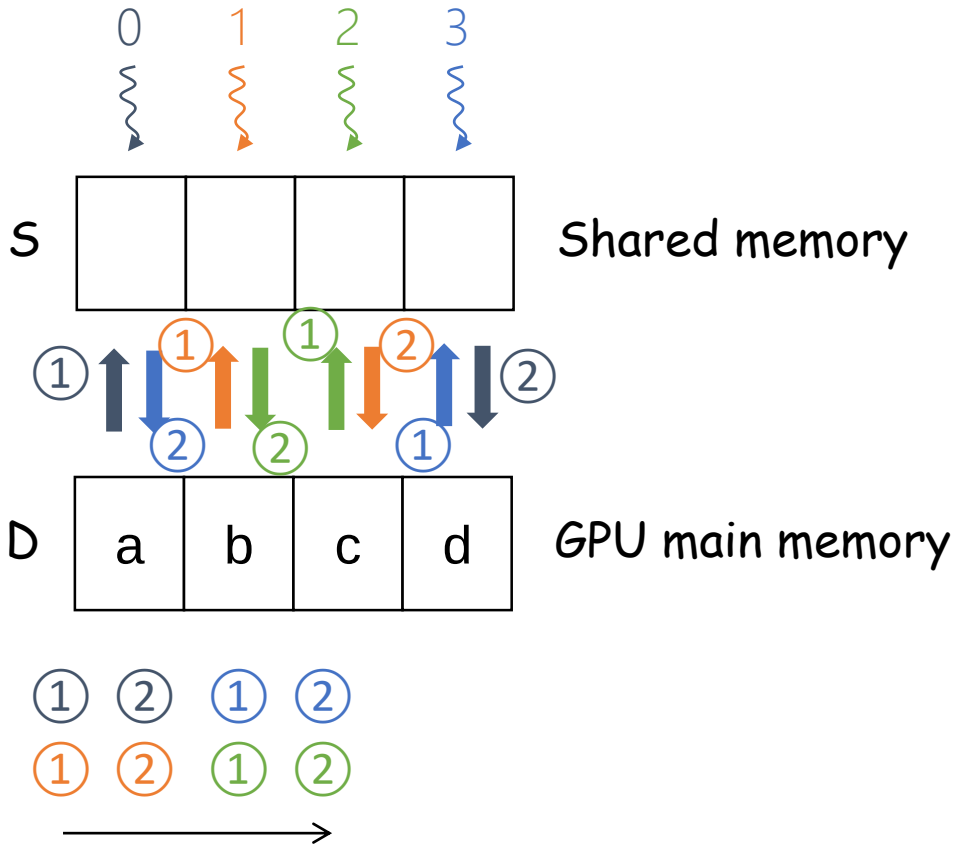
https://github.com/NVIDIA-developer-blog/code-samples/blob/master/series/cuda-cpp/shared-memory/shared-memory.cu

立志成才报国裕民

Manually Synchronize

```
#include <stdio.h>

__global__ void staticReverse(int *d, int
n)
{
  __shared__ int s[64];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
   __syncthreads();
  d[t] = s[tr];
}
```



https://github.com/NVIDIA-developer-blog/code-samples/blob/master/series/cuda-cpp/shared-memory/shared-memory.cu

立志成才报国裕民