Roofline and TPU Performance

CS217

Kunle Olukotun

Performance Models / Simulators

- Historically, many performance models and simulators tracked latencies to predict performance (i.e. counting cycles)

- The last two decades saw a number of latency-hiding techniques…
  - Out-of-order execution (hardware discovers parallelism to hide latency)
  - HW stream prefetching (hardware speculatively loads data)
  - Massive thread parallelism (independent threads satisfy the latency-bandwidth product)

- Effectively latency hiding has resulted in a shift from a latency-limited computing regime to a **throughput-limited computing regime**

Roofline Model

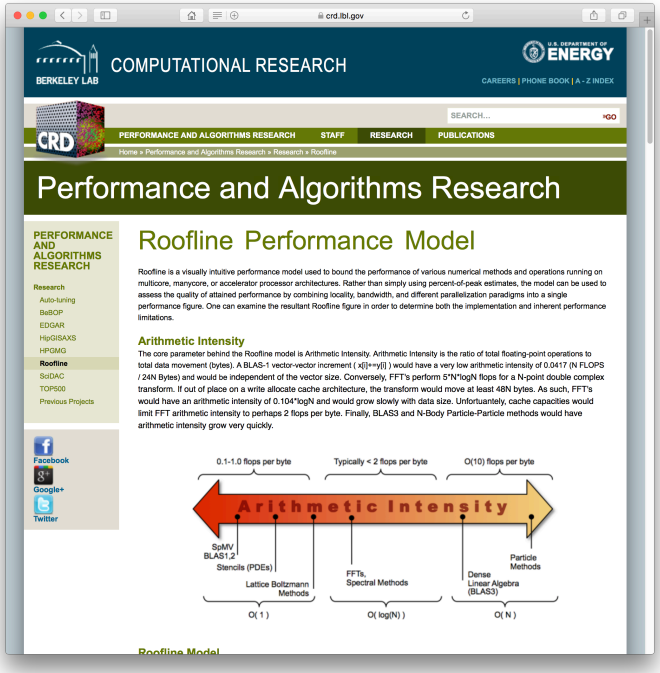
- The **Roofline Model** is a throughput-oriented performance model…
  - Tracks rates not times
  - Augmented with Little's Law

    (concurrency = latency*bandwidth)
  - Independent of ISA and architecture

    (applies to CPUs, GPUs, Google TPUs, etc…)
- Two Components:
  - Machine Characterization

    (realistic performance potential of the system)
  - Theoretical Application Bounds

    (how well could my app perform with perfect compilers, caches, overlap, …)

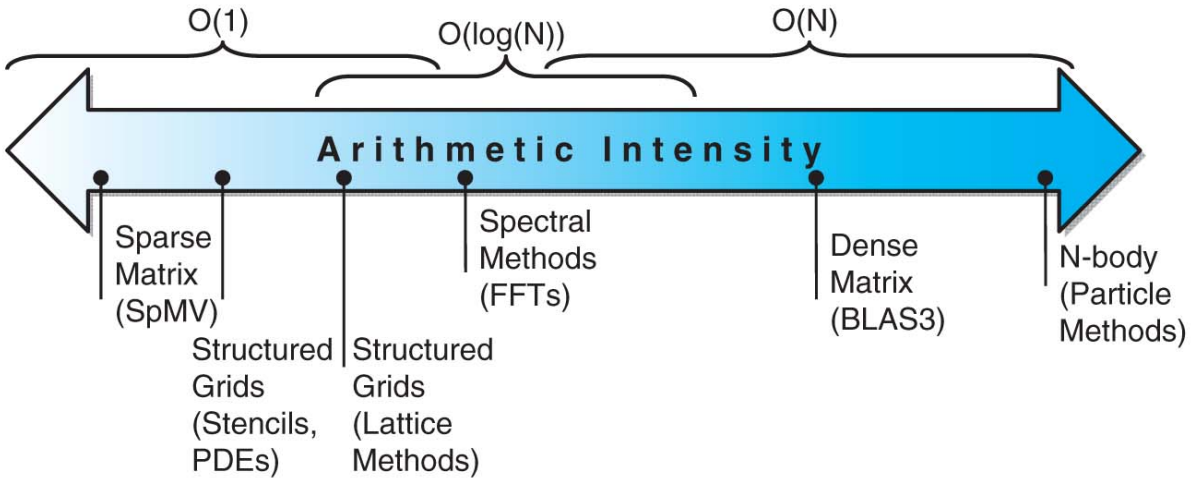


https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

Application Bounds of Numerical Applications

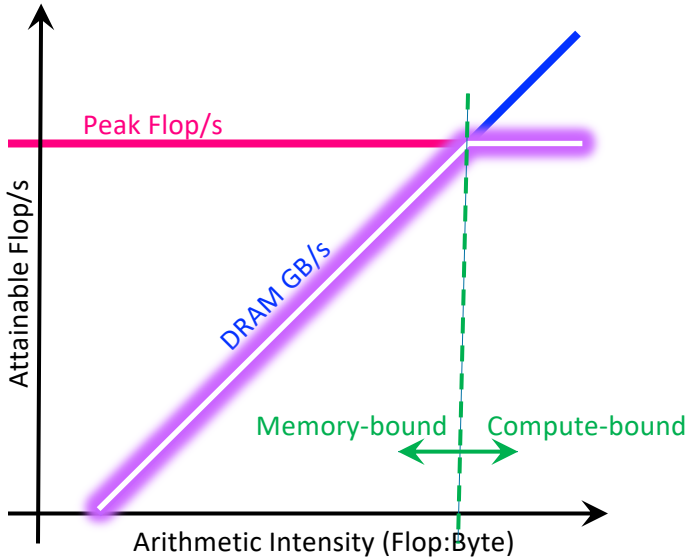
- Arithmetic intensity of a kernel
  - FLOPs per byte of memory accessed

$$\frac{\text{FP ops}}{\text{Bytes accessed}}$$

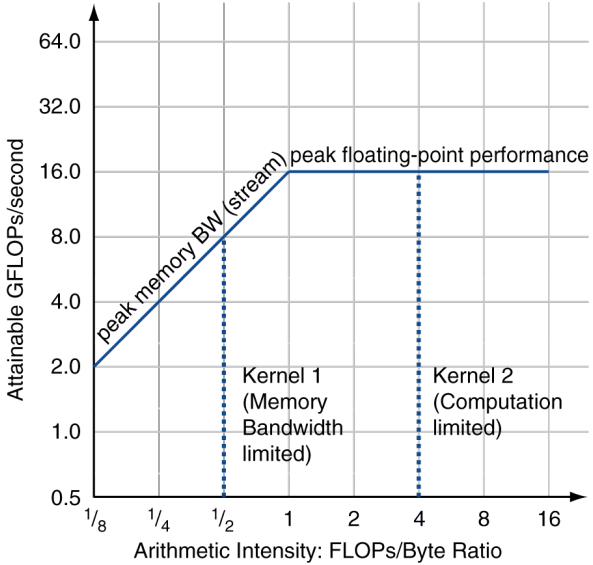


(DRAM) Roofline

- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance
- Consider idealized processor/caches
- Plot the performance bound using Arithmetic Intensity (AI) as the x-axis…
  - AI = Flops / Bytes presented to DRAM
  - **Attainable Flop/s = min(peak Flop/s, AI * peak GB/s)**
  - **Log-log scale** makes it easy to doodle, extrapolate performance along Moore's Law, etc…
  - Kernels with AI less than machine balance are ultimately DRAM bound (we'll refine this later…)



The Roofline Model – Simple Model



- For memory intensive FP kernels
  - Log-log scale
  - Data does not fit in cache
- Example
  - 16 GFLOPS peak
  - 16 GB/sec peak
- What does "ridge-point" indicate?

- Attainable GPLOPs/sec
  = Min (Peak Memory BW × Arithmetic Intensity, Peak FP Performance)

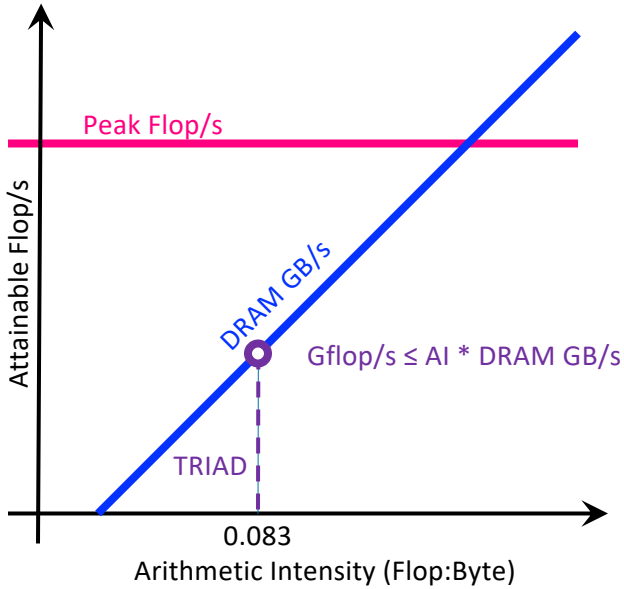
Roofline Example #1

- Typical machine balance is 5-10 flops per byte…
  - 40-80 flops per double to exploit compute capability
  - Artifact of technology and money
  - **Unlikely to improve**

- Consider STREAM Triad…

```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

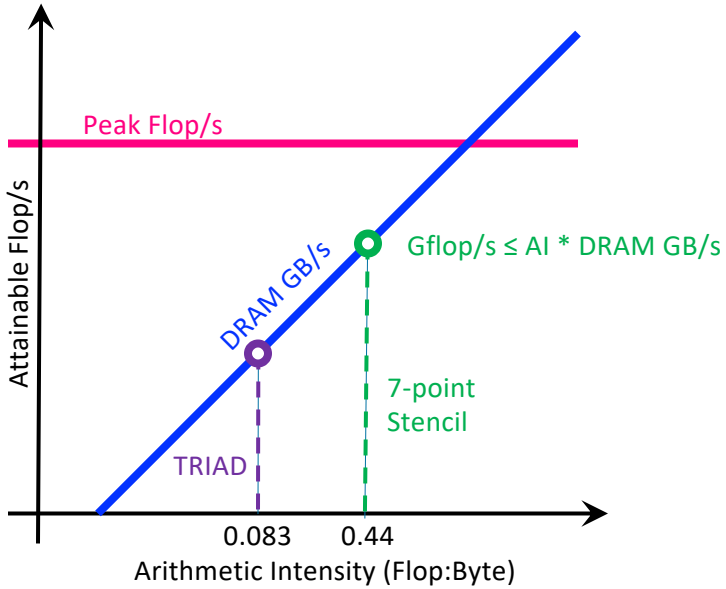
  - 2 flops per iteration
  - Transfer 24 bytes per iteration (read X[i], Y[i], write Z[i])
  - **AI = 0.083 flops per byte == Memory bound**



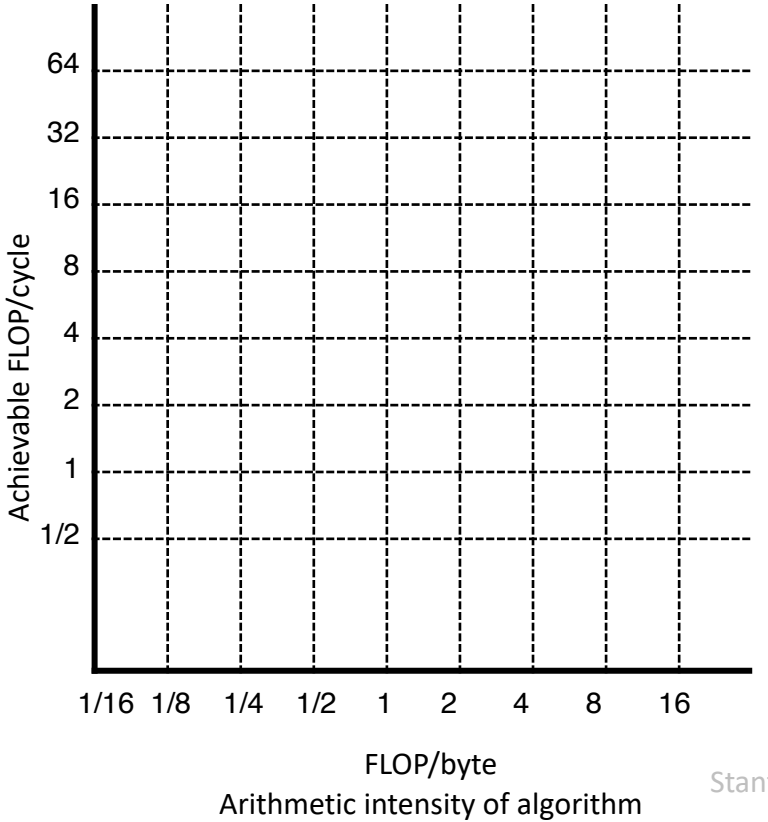
Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
  - 7 flops
  - 8 memory references (7 reads, 1 store) per point
  - Cache can filter all but 1 read and 1 write per point
  - **AI = 0.44 flops per byte == memory bound,**
  - **but 5x the flop rate**

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  int ijk = i + j*jStride + k*kStride;
  new[ijk] = -6.0*old[ijk        ]
             + old[ijk-1         ]
             + old[ijk+1         ]
             + old[ijk-jStride]
             + old[ijk+jStride]
             + old[ijk-kStride]
             + old[ijk+kStride];
}}}
```



Roofline Example #3



Given a processor with:

1. 8 fully pipelined FP units
2. 8 byte/cycle memory BW
3. 4 byte floats

What's the performance bound on the SAXPY loop below?

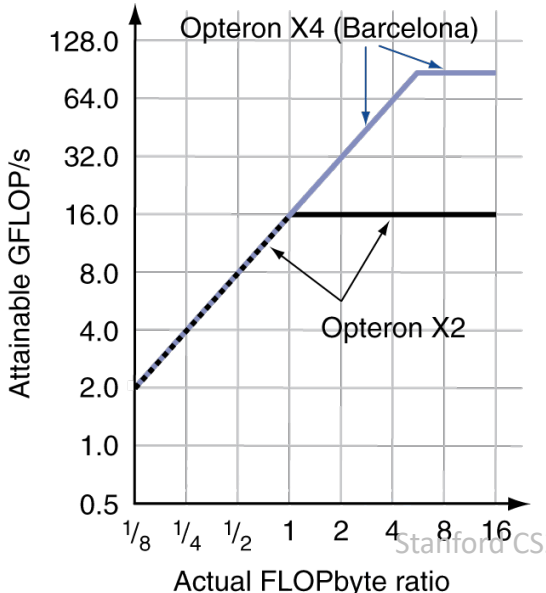
1. X and Y are in main memory

2. C loop:
```
for (i=0; i < 100,000; i++)
    Y(i) = a*X(i) + Y(i);
```

```
foo:    LF      F2, 0 (R1) // load X(i)
        MULTF   F4, F2, F0 // multiply a*X(i)
        LF      F6, 0 (R2) // load Y(i)
        ADDF    F6, F4, F6 // add a*X(i) + Y(i)
        SF      0 (R2), F6 // store Y(i)
        ADDI    R1, R1, #4 // increment X index
        ADDI    R2, R2, #4 // increment Y index
        SGTI    R3, R1, #100000// test if done
        BEQZ    R3, foo    // loop not done
```

Comparing Systems

- Example: Opteron X2 vs. Opteron X4
  - 2-core vs. 4-core, 2× FP performance/core, 2.2GHz vs. 2.3GHz
  - Same memory system



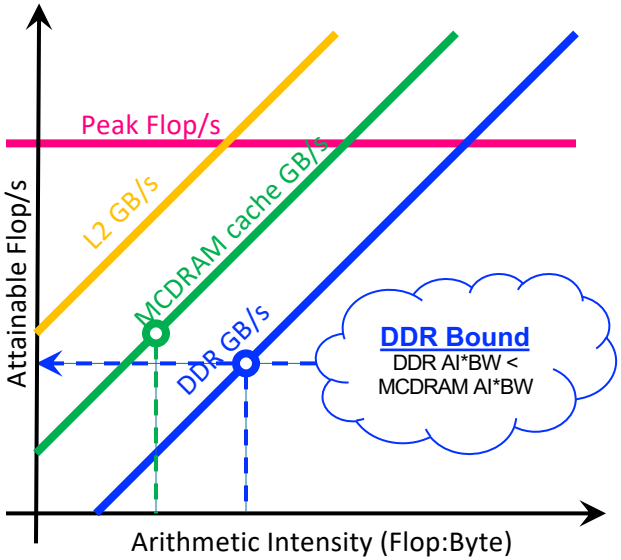
- To get higher performance on X4 than X2
  - Need high arithmetic intensity
  - Or working set must fit in X4's 2MB L-3 cache

Hierarchical Roofline

- Real processors have multiple levels of memory
  - Registers
  - L1, L2, L3 cache
  - MCDRAM/HBM (KNL/GPU device memory)
  - DDR (main memory)
  - NVRAM (non-volatile memory)
- Applications can have locality in each level
  - Unique data movements imply unique AI's
  - Moreover, each level will have a unique bandwidth

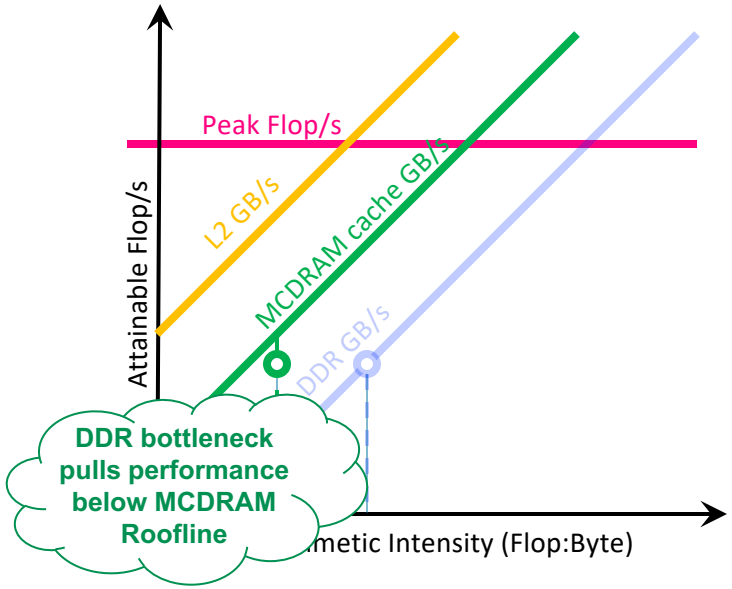
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although a loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



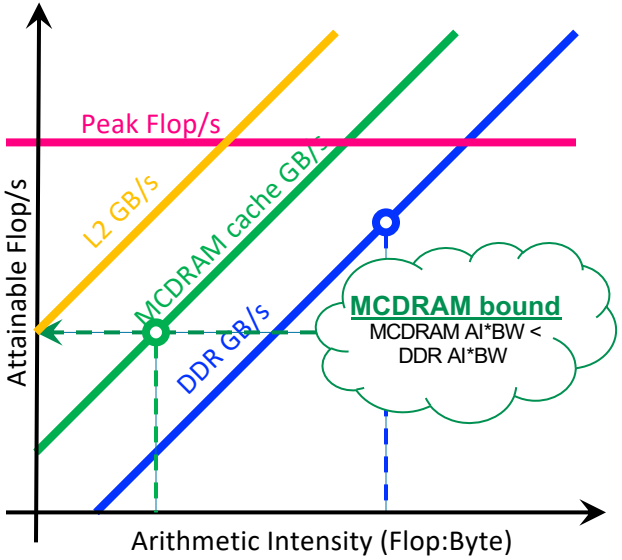
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



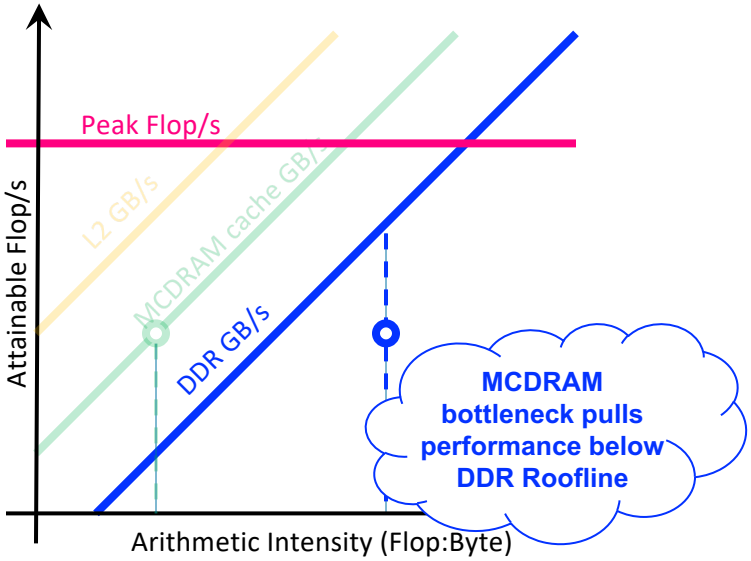
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



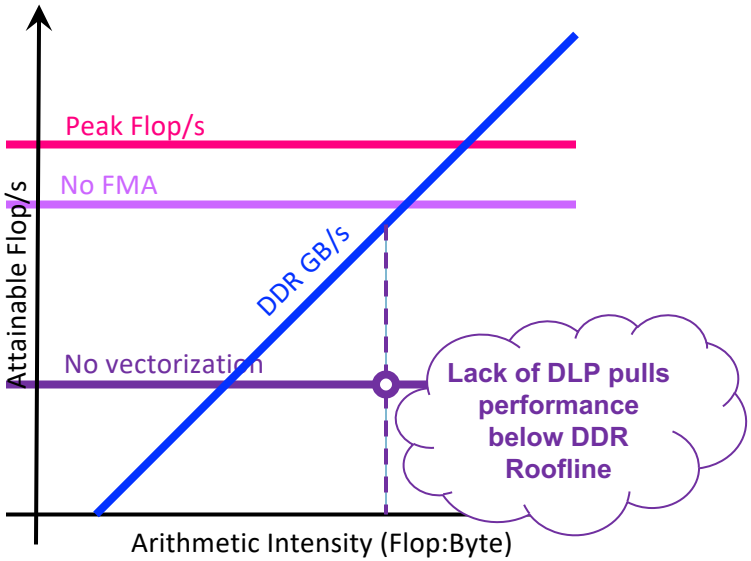
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



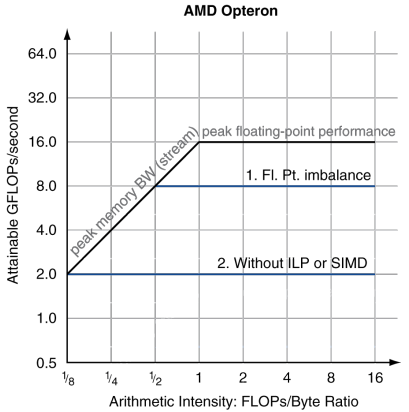
Data, Instruction, Thread-Level Parallelism…

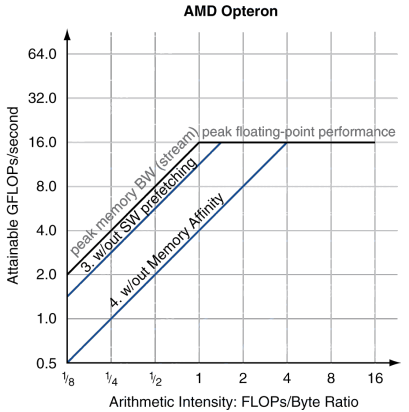
- We have assumed one can attain peak flops
- In reality, this is premised on sufficient…
  - Use special instructions (e.g. fused multiply-add)
  - Vectorization/SIMD (16 flops per instruction)
  - unrolling, out-of-order execution (hide FPU latency)
  - OpenMP across multiple cores
- Without these, …
  - Peak performance is not attainable
  - Some kernels can transition from memory-bound to compute-bound
  - n.b. in reality, DRAM bandwidth is often tied to DLP and TLP (single core can't saturate BW w/scalar code)



Optimizing Performance

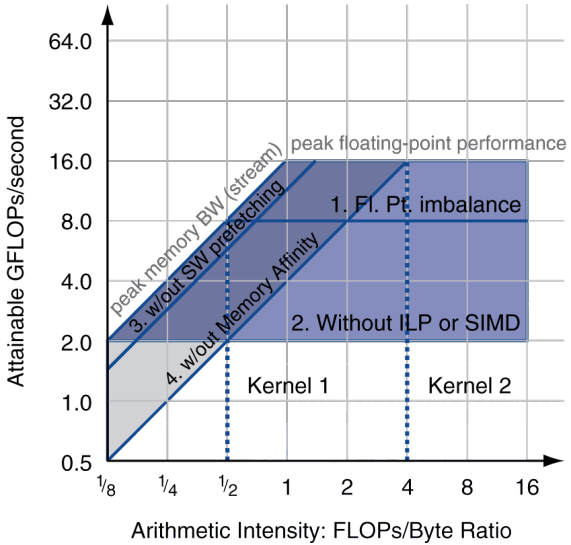
- Increase thread parallelism
- Optimize FP performance
  - Balance adds & multiplies
  - Improve superscalar ILP and use of SIMD instructions
  - Loop unrolling, software pipelining
- Optimize memory usage
  - Software prefetch (SW pipelining)
    - Avoid load stalls
  - Memory affinity
    - NUMA
    - Avoid non-local data accesses
- Optimizations: "Ceiling"





Optimizing Performance

- Choice of optimization depends on arithmetic intensity of code (Kernel 1 vs. Kernel 2)

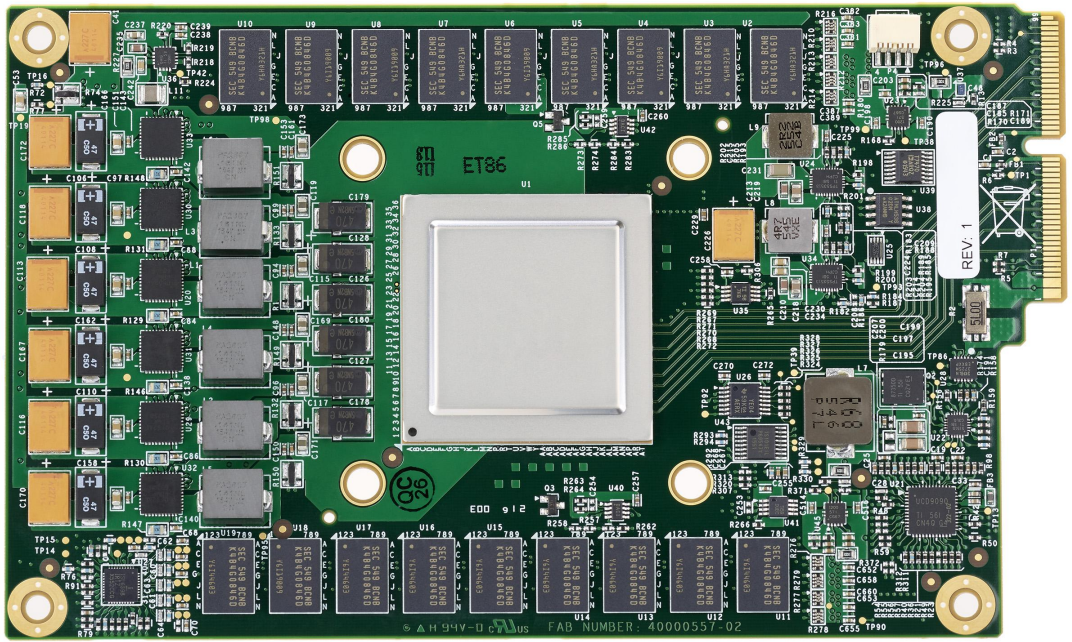


- Arithmetic intensity is not always fixed
  - May increase with problem size
    - Dense matrix, N-body
    - Weak scaling advantage
  - Caching reduces memory accesses
    - Increases arithmetic intensity
    - Blocking
    - Use hierarchical roofline

TPU: Avoid Success/Disaster for NN Apps

- 2013: NNs apps would require 2x–3x CPUs
- Custom hardware to reduce TCO of NN
- Improve inference by 10x vs. GPUs
- Short development cycle
  - Start in 2014 deployed 15 months later in datacenter

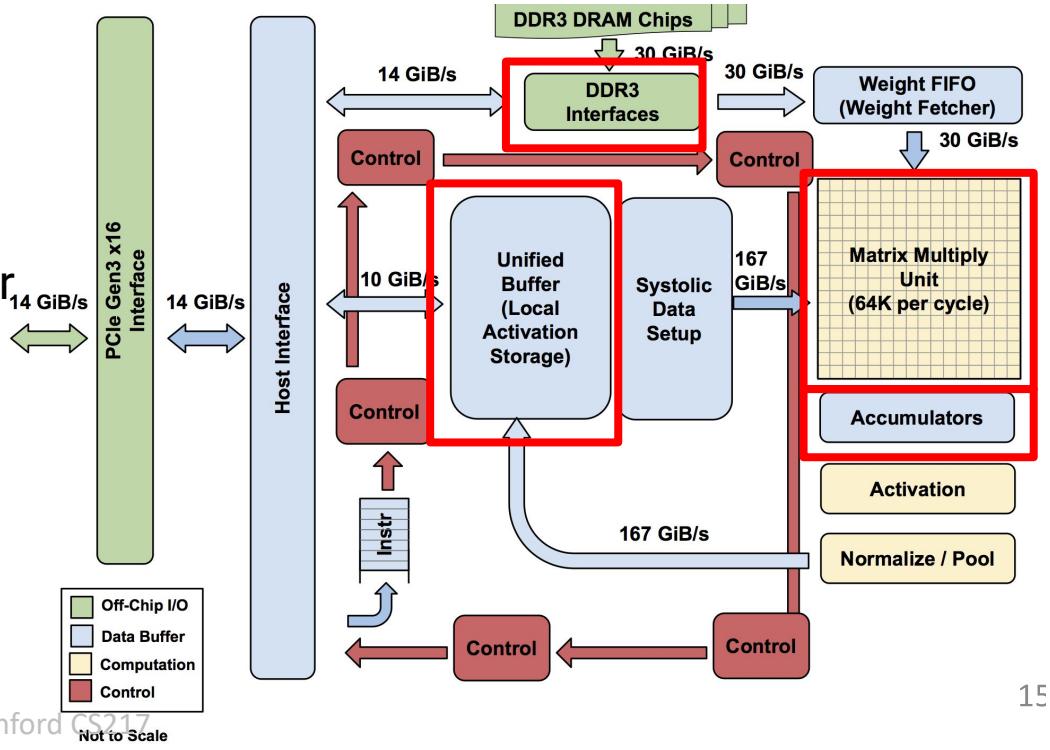
TPU Card



Up to 4 cards per server

TPU High-level Chip Architecture

- The Matrix Unit: 65,536 (256x256) 8-bit multiply-accumulate units
  - Systolic array
- 700 MHz clock rate
- Peak: 92T operations/second
  - 65,536 * 2 * 700M
- 4 MB of on-chip Accumulator memory
- 24 MB of on-chip Unified Buffer (activation memory)
- Two 2133MHz DDR3 DRAM channels
- 8 GB of off-chip weight DRAM memory
- vs GPU and CPU
  - >25X as many MACs vs GPU
  - >100X as many MACs vs CPU



TPU Programmers View

- Five key CISC instructions (CPI > 10)

```
Read_Host_Memory
Write_Host_Memory
Read_Weights
MatrixMultiply/Convolve
Activate(ReLU,Sigmoid,Maxpool,LRN,...)
```

- Complexity in software
  - No branches
  - In-order issue
  - Software controlled buffers
  - Software controlled pipeline synchronization

Three Types of NNs

1. Multilayer Perceptrons
   - Each new layer applies nonlinear function F to weighted sum of all outputs from prior layer ("fully connected") $x_n = F(Wx_{n-1})$
2. Convolutional Neural Network
   - Like MLPs, but same weights used on nearby subsets of outputs from prior layer
3. Recurrent NN/"Long Short-Term Memory"
   - Each new layer a NL function of weighted sums of past *state* and prior outputs; same weights used across time steps

2016 NN Datacenter Workload

Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% Deployed*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	0.1k	5				5	ReLU	20M	200	200	61%
MLP1	1k	4				4	ReLU	5M	168	168	
LSTM0	1k	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1.5k	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1k		16			16	ReLU	8M	2888	8	5%
CNN1	1k	4	72		13	89	ReLU	100M	1750	32	

Three Contemporary Chips

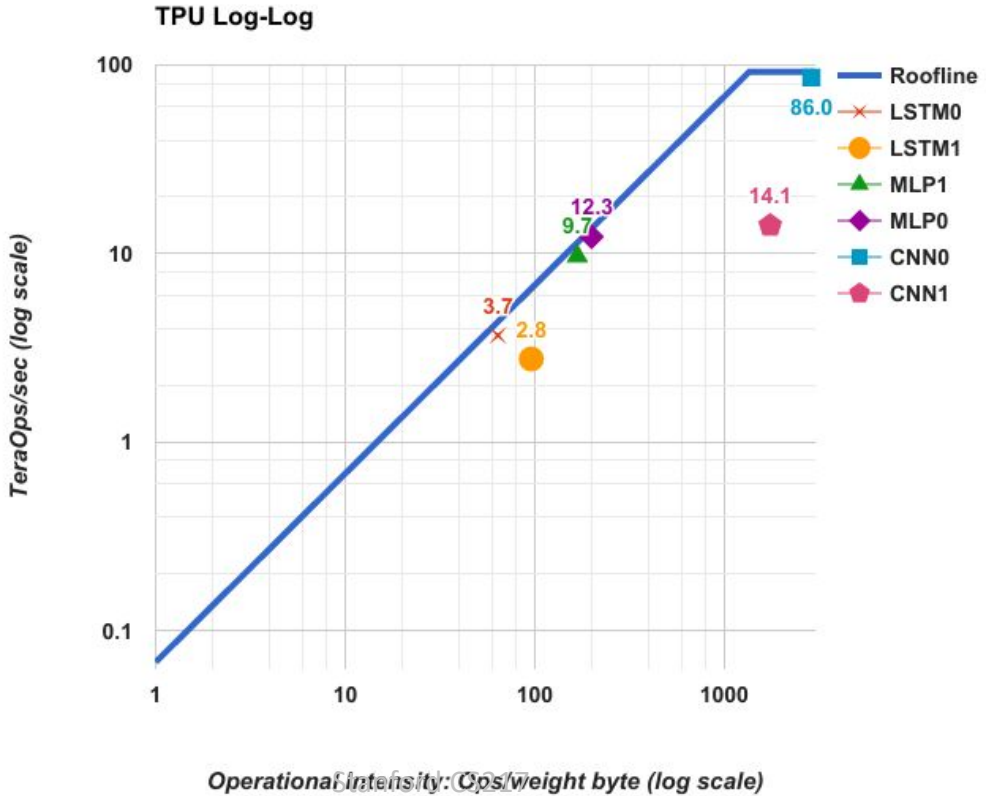
Processor	*mm²*	*Clock MHz*	*TDP Watts*	*Idle Watts*	*Memory GB/sec*	*Peak TOPS/chip*	
						8b int.	*32b FP*
CPU: Haswell (18 core)	662	2300	145	41	51	2.6	1.3
GPU: Nvidia K80 (2 / card)	561	560	150	25	160	--	2.8
TPU	<331*	700	75	28	34	91.8	--

*TPU is less than half die size of the Intel Haswell processor

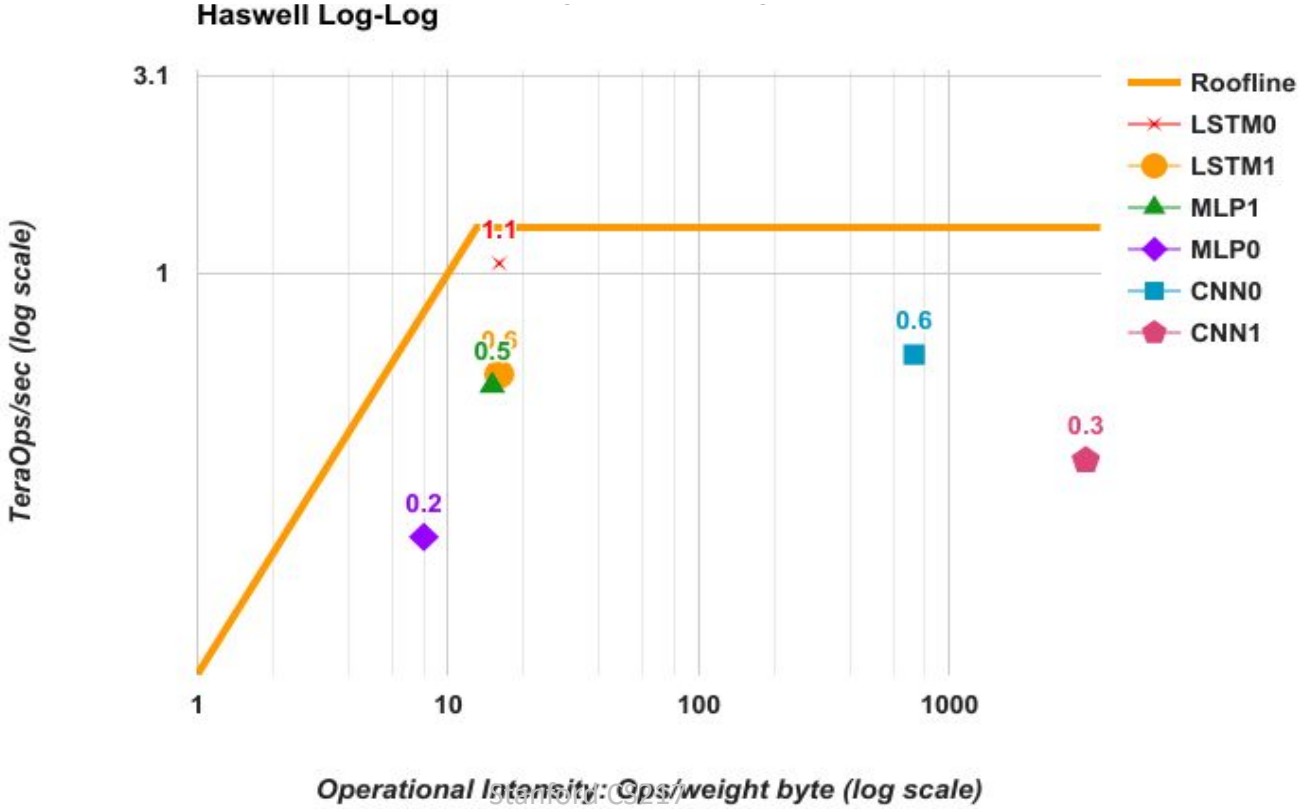
K80 and TPU in 28 nm process; Haswell fabbed in Intel 22 nm process

These chips and platforms chosen for comparison because widely deployed in Google data centers

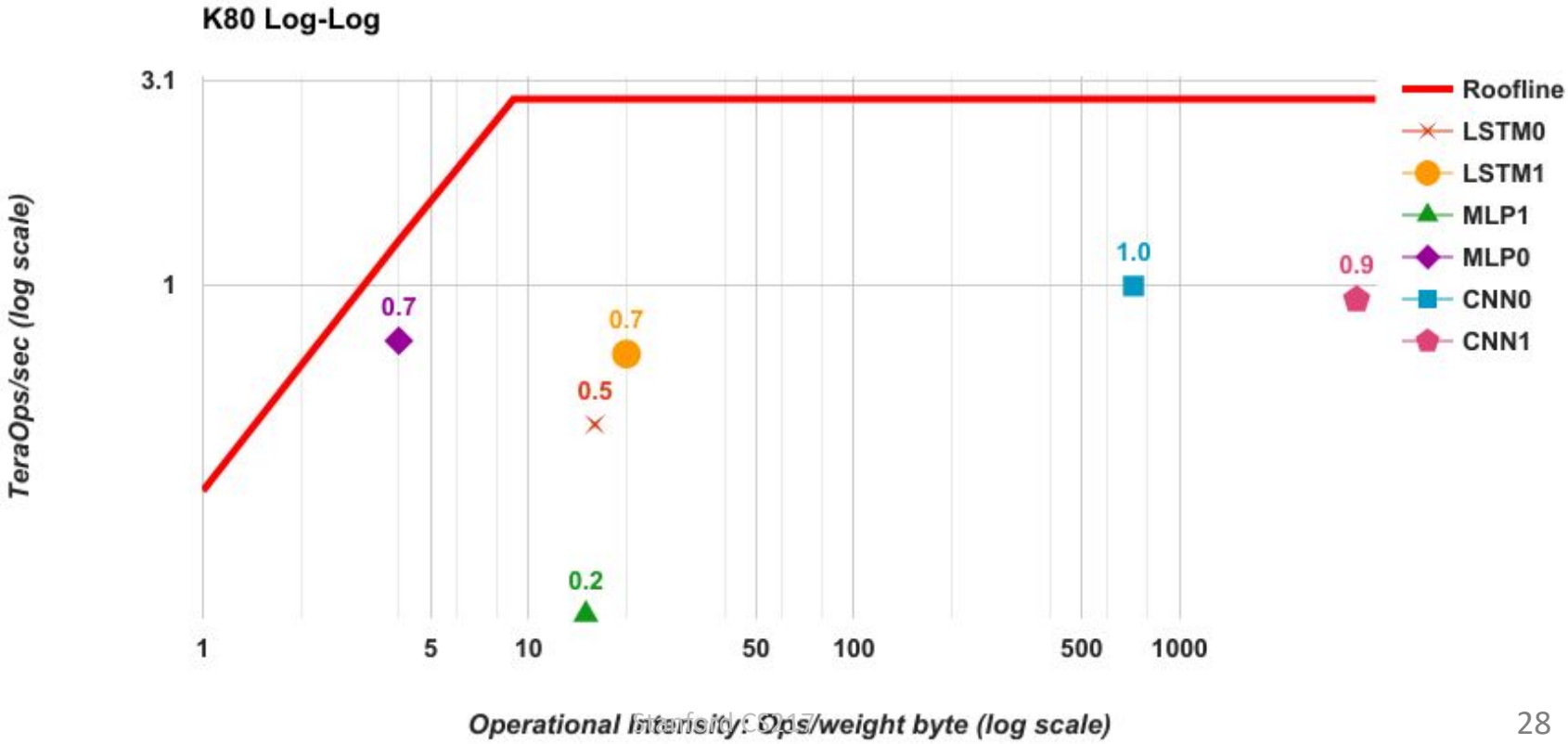
TPU Roofline



CPU (Haswell) Roofline



GPU (K80) Roofline



Why Below Rooflines (MLP0)

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%

Performance of TPU & GPU Relative to CPU

Type	*MLP*		*LSTM*		*CNN*		*Weighted Mean*
	0	*1*	*0*	*1*	*0*	*1*	
GPU	2.5	0.3	0.4	1.2	1.6	2.7	1.9
TPU	41.0	18.5	3.5	1.2	40.3	71.0	29.2
Ratio	16.7	60.0	8.0	1.0	25.4	26.3	15.3