LLM Inference Lectures
Part 1: GPU

(Harvard CS265 – DATA/AI Systems LAB)

Milad Rezaei Hajidehi
(milad@seas.harvard.edu)

Inference: Descent Through Abstraction

Hugging Face

Model is a blackbox.
Many details abstracted away (tokenization, loading weights)

DASlab @ Harvard SEAS

Inference: Descent Through Abstraction

Hugging Face

PyTorch

Neural Operators with details exposed;
operators, tensors, and control flow are explicit.

DASlab @ Harvard SEAS

Inference: Descent Through Abstraction

Hugging Face

PyTorch

Torch Dispatch
ATen

Canonical Math Operators;
Choosing backend implementations and parameters.

DASlab @ Harvard SEAS

Inference: Descent Through Abstraction

Hugging Face

PyTorch

Torch Dispatch
ATen

cuDNN
cuBLAS
Custom Kernels

Vendor Libraries Highly Optimized GEMMs
Custom GPU codes for even more optimized execution.

DASlab @ Harvard SEAS

Inference: Descent Through Abstraction

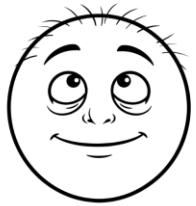
Hugging Face

PyTorch

Torch Dispatch
ATen

cuDNN
cuBLAS
Custom Kernels

PTX
HW Runtime
(memory, caches, SMs)



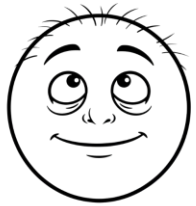
All physical works!
Data-Movement & Actual Compute!

DASlab @ Harvard SEAS

This Session: GPU/Performance

✓ GPU Structure, Compute/Kernels Workflow.
✓ Performance Considerations, Design Decisions.
✓ Self-Designing Kernels.

PTX
HW Runtime (memory, caches, SMs)



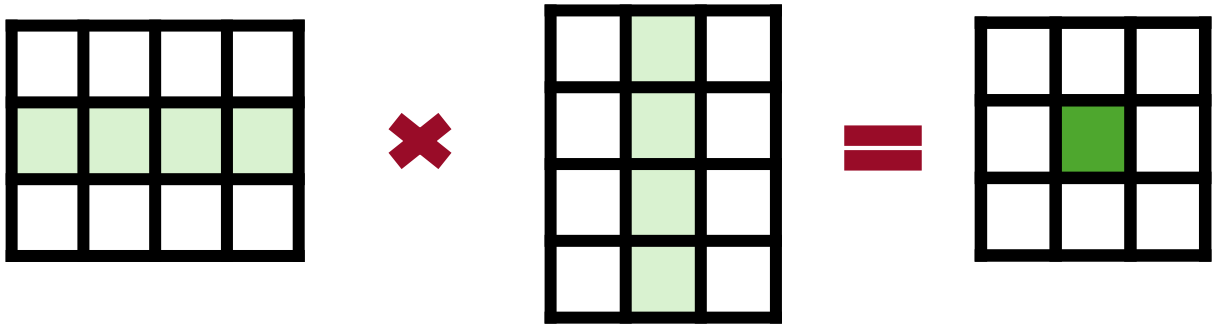
💡 What happens in GPU when you run a model?
💡 Why increasing batch-size improve performance?
💡 What factors/metrics yield GPU performance?
💡 Why writing "good" kernel is hard?

It's all About Performance!

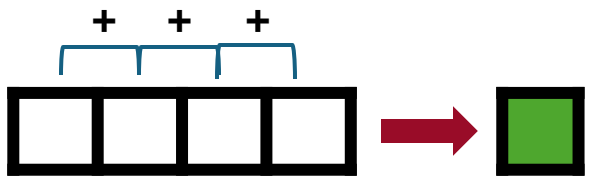
= Physical Operations

= Compute Always Active ≈ Data-Movement Is Optimized

LLM Operators



Matrix Mult.
(QKV, Att. Score, MLP, Output Proj.)



Reduce
(RMSNorm, Softmax)

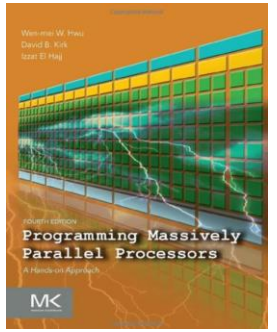
And some piecewise operators…

DASlab @ Harvard SEAS

GPU Programming Model
Single Instruction Multiple Thread

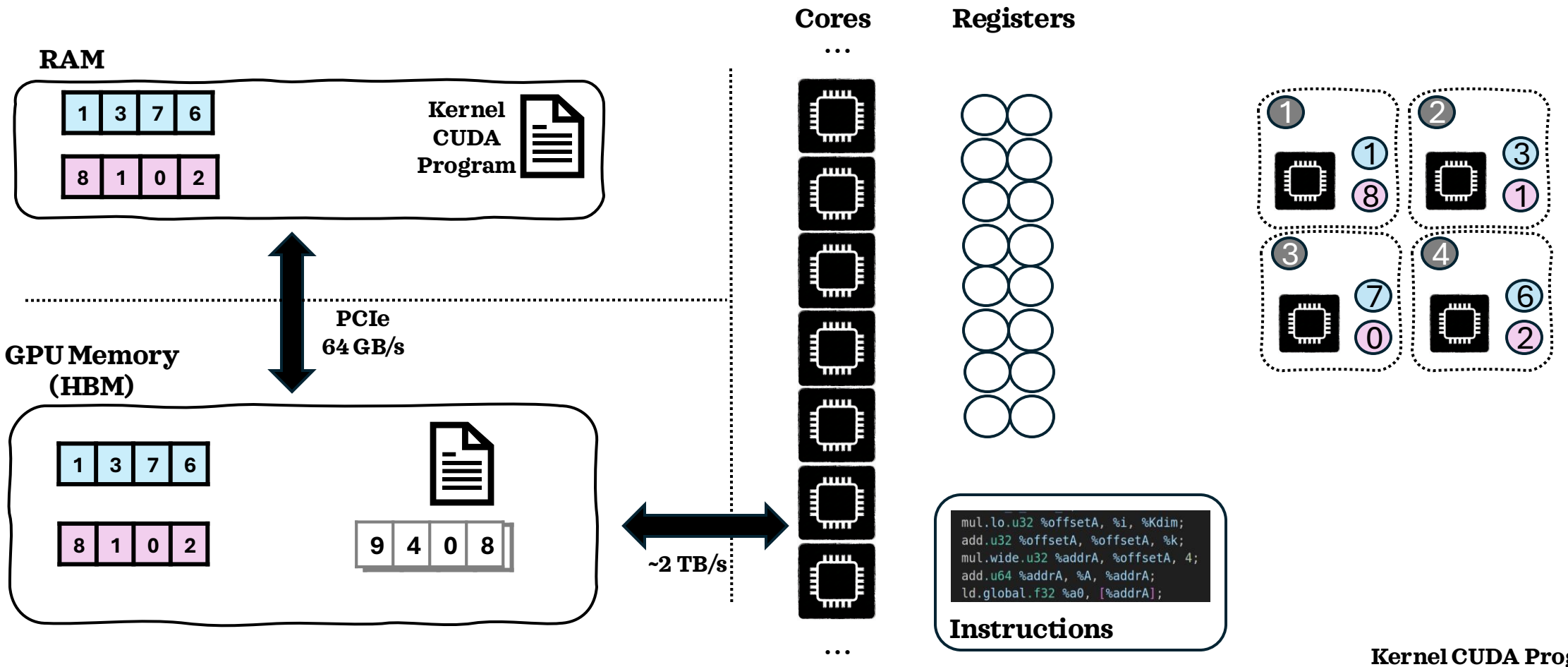
```
__global__ void vadd(const float* a, const float* b,
    float* y, int n) {
    int tid = ... // Syntax for calc. thread id
    y[tid] = a[tid] + b[tid];
}
```

GPU Programming is Easy if You Don't Care About Performance
– GPU Holy Book, Chapter 1



DASlab @ Harvard SEAS

Kernels & GPU Workflow – Add Two Vectors

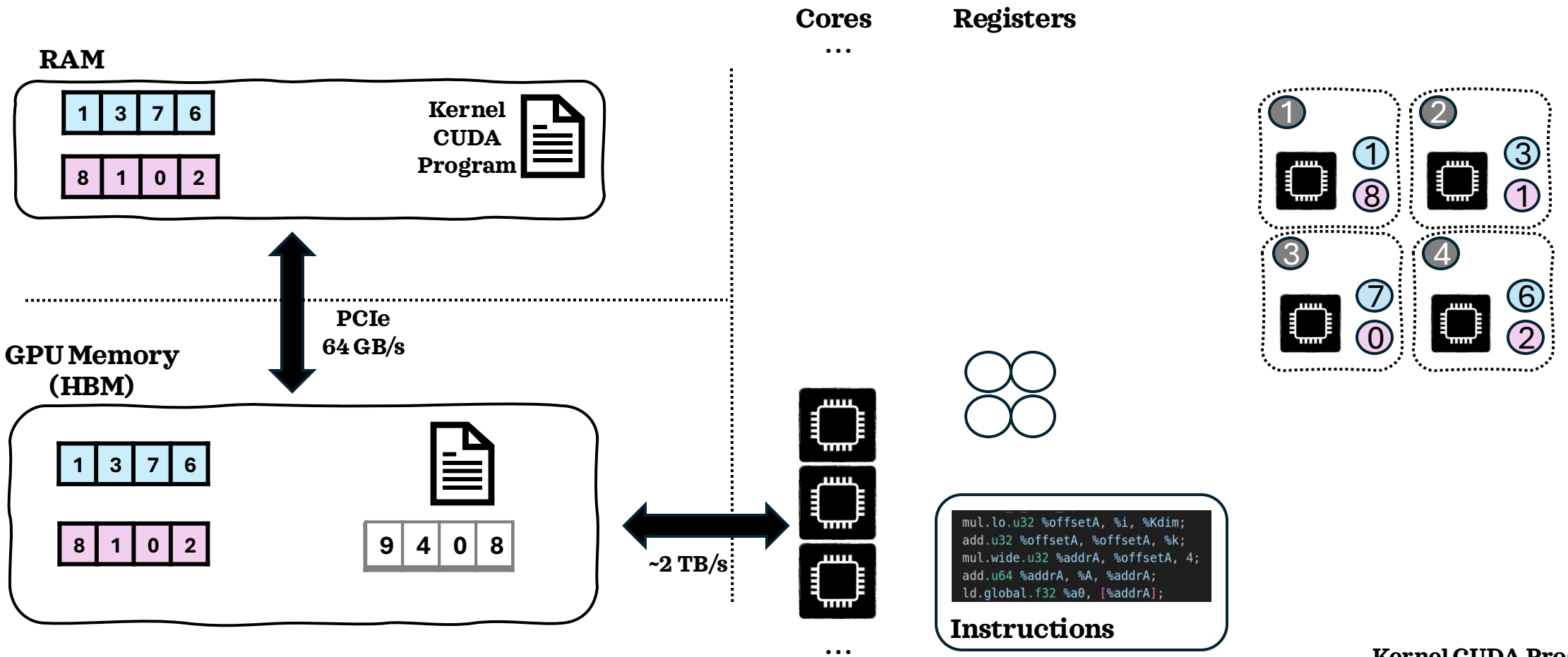


Kernel CUDA Program

```
__global__ void vadd(const float* a, const float* b,
    float* y, int n) {
    int tid = ... // Syntax for calc. thread id
    y[tid] = a[tid] + b[tid];
}
```

DASlab @ Harvard SEAS

What may Become Bottleneck Here?

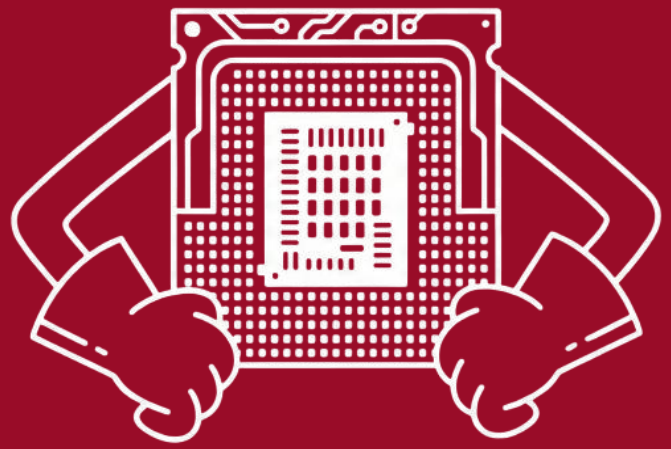


Kernel CUDA Program

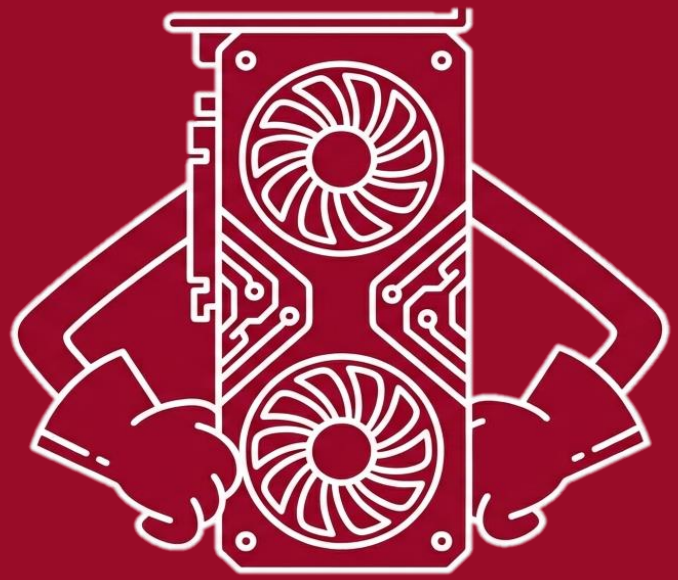
```
__global__ void vadd(const float* a, const float* b,
    float* y, int n) {
    int tid = ... // Syntax for calc. thread id
    y[tid] = a[tid] + b[tid];
}
```

DASlab @ Harvard SEAS

CPU vs. GPU
It's all About Circuit Area!



Large Control
Out-of-Order + Speculation
Large Caches
Wide Pipelines
Big Instruction Cache
Independent Cores



Many Cores/ALU
Large Register Files
Simple Control
Shared Instructions
Very Fast Interconnect

GPU Map: Hierarchy

- Managing Massive Parallelism is Hard.
- Memory and Compute divided into a Hierarchy.
- GPU -> Stream Multi-Processor -> Quadrant -> Cores
- Various memory and caches in each level.

DASlab @ Harvard SEAS

GPU Map: Compute Units

Logical

Threads: Programming Unit. Per Block.

Block/CTA: Group of threads, can have synchronization, always on same SM, can share a shared-memory.

Grid: A kernel specified by number of blocks and threads.

Physical

CUDA Core: General Purpose ALU. ~128 Per SM.

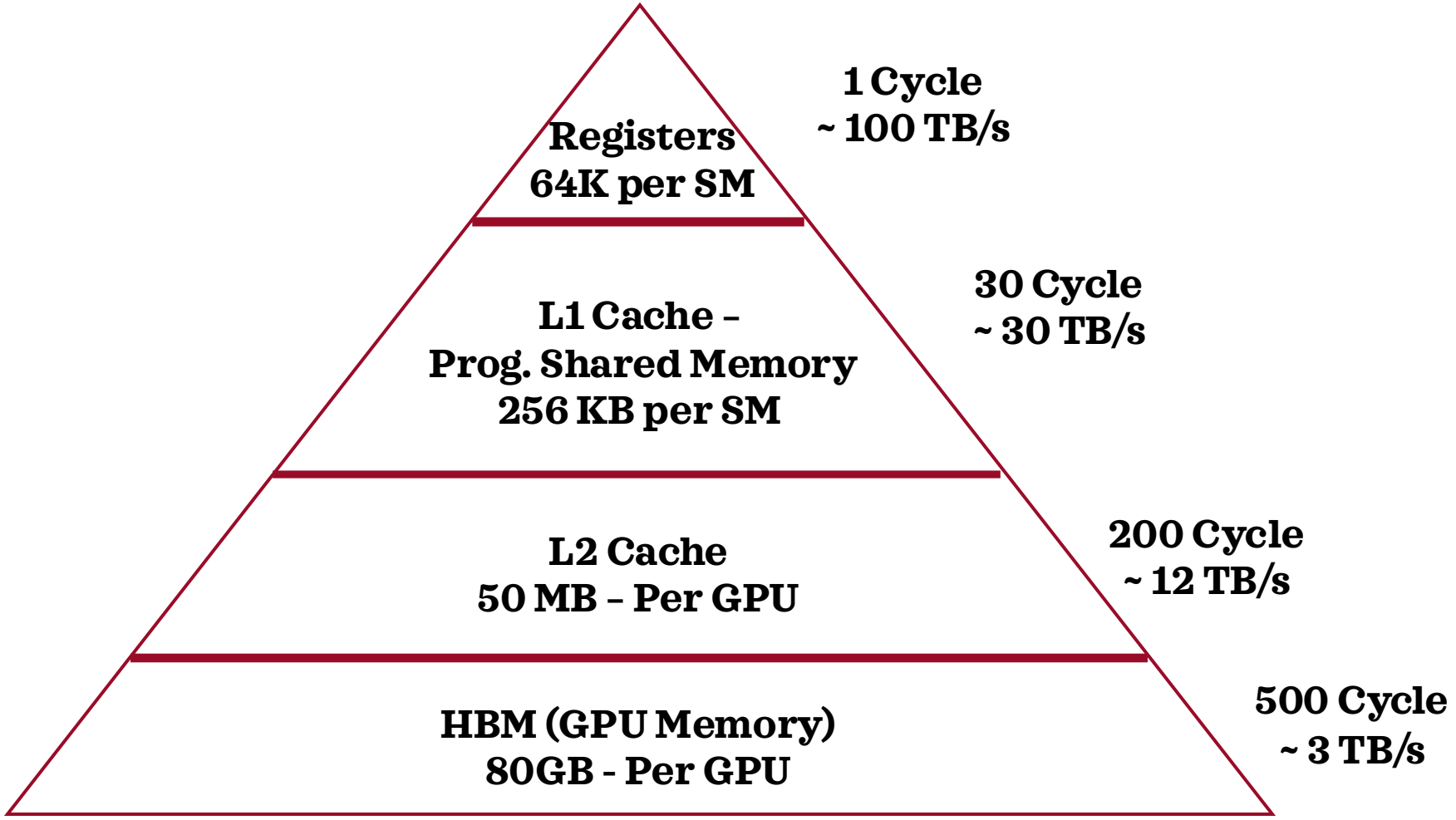
Tensor Core: Special units for MatMul. ~4 Per SM.

SM: ~ 100 per GPU. An independent collection of cores/registers/memory. Does scheduling, pipelining.

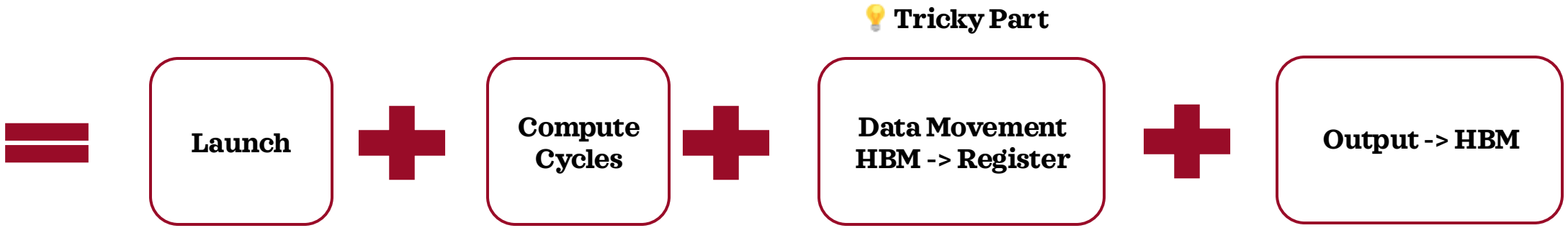
WARP: Group of 32 Threads always scheduled together and have their own Program Counter.

! **Warp Divergence:** Threads in same warp should have same control flow. Otherwise it Causes Idle Cores and Reduce Utilization.

GPU Map: Memory Units



Latency of a Kernel



Push/Pull Kernels:
Bind threads to Input or Output.

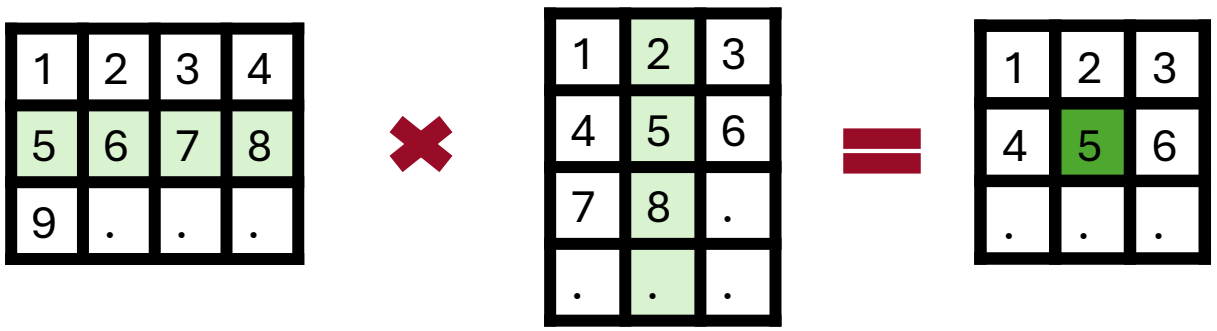
Number of Blocks & Threads:
shape & hardware sensitive.

Let's See how we can improve this!

```cpp
// Naive GEMM with 1D: one thread computes one C(i,j)
__global__ void matmul_naive_1d(const float* A, const float* B, float* C,
                                int M, int N, int K) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    int total = M * N;
    if (tid < total) {
        int i = tid / N;       // row
        int j = tid - i * N;  // col (tid % N)

        float acc = 0.0f;
        for (int k = 0; k < K; ++k)
            acc += A[i*K + k] * B[k*N + j];
        C[i*N + j] = acc;
    }
}

// Launch (1D)
int total = M * N;
int block = 256;
int grid  = (total + block - 1) / block;
matmul_naive_1d<<<grid, block>>>(A, B, C, M, N, K);
```



DASlab @ Harvard SEAS

Occupancy – Zero-Latency Context Switch

Ultimate Dream = Best Throughput = Compute Cores Always Active

Impossible! Due to Memory Reads.

What to do with Intermediate State?

DASlab @ Harvard SEAS

Occupancy – Zero-Latency Context Switch

What to do with Intermediate data?

That is why we have large register files!
-> Zero-Latency Context Switch, Storing State in Registers

We have to oversubscribe GPU! How much?

64 Active Warp is Optimal. But we limited by Registers!
Occupancy = active warps on an SM / maximum possible warps on that SM

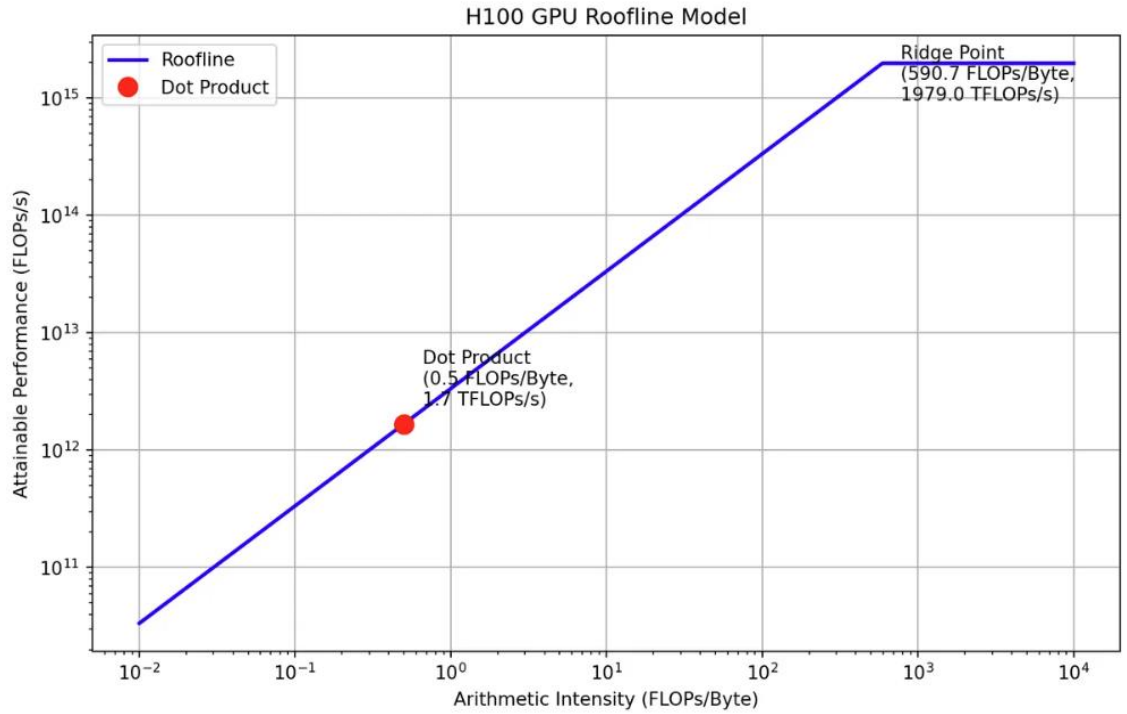
Arithmetic Intensity

Arithmetic Intensity: Classify Kernels to Memory- or Compute- Bound.

For each Thread: Flops / (Byte Moved from HBM)

Roofline Model:
Peak performance changes for different intensities.

Occupancy Mostly Helps Memory-Bound Kernels.



DASlab @ Harvard SEAS

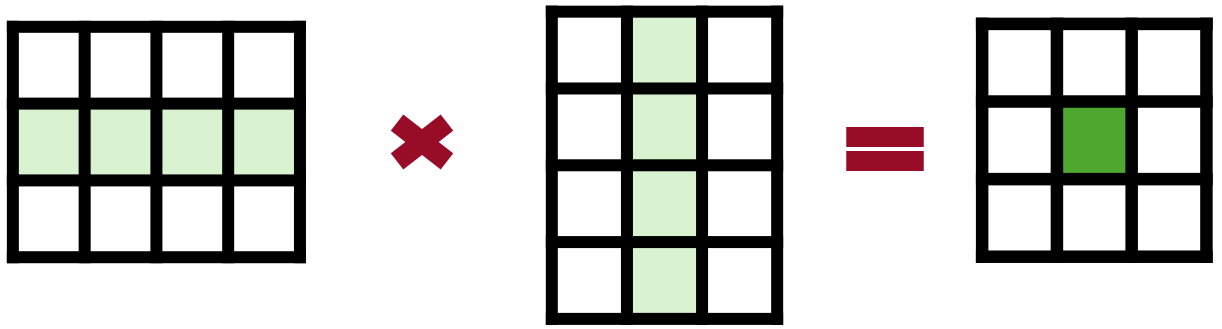
Arithmetic Intensity - Examples

Add Kernel: 1 FLOPs for Adding / (2 Float Loads for Input + 1 Float Store for Output)
1 / 12 = 0.083 !!!

Matrix Mult. A=(n,k) B=(k,m):

K loads from A, K loads from B, 1 Store. Each 4 Bytes.
K Multiply, K-1 Adds.

~ 1/4

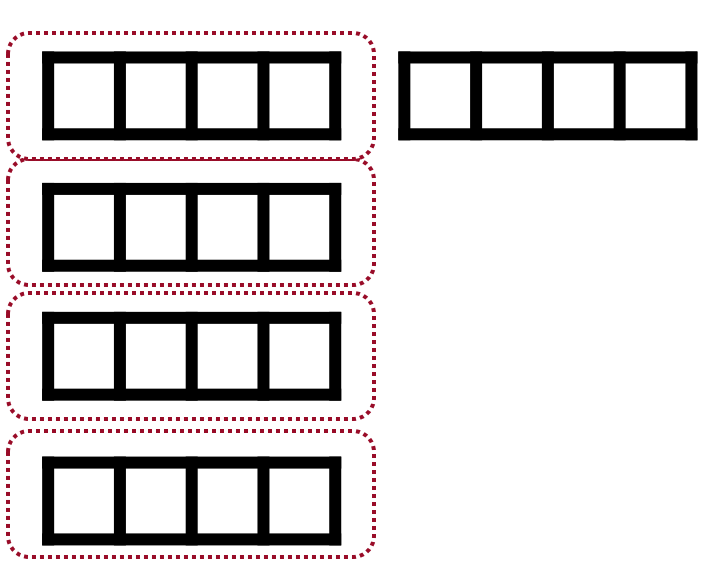


DASlab @ Harvard SEAS

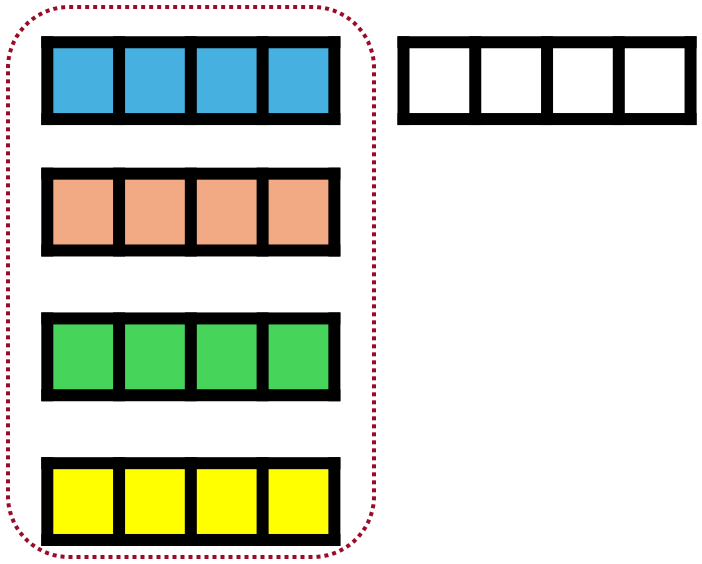
Why Batching Operators is Good for Performance?

Batching

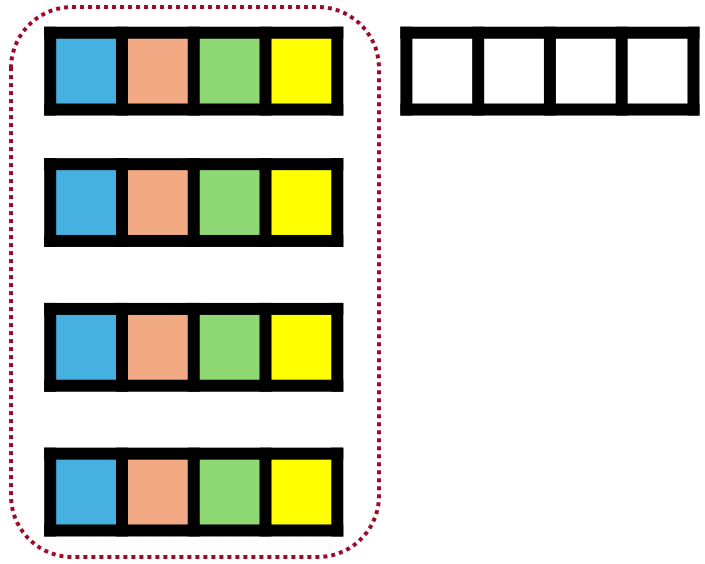
Workload: Add/Dot Four Vectors with One Fixed Vector. Four Cores Available.



4 Kernels



1 Kernels

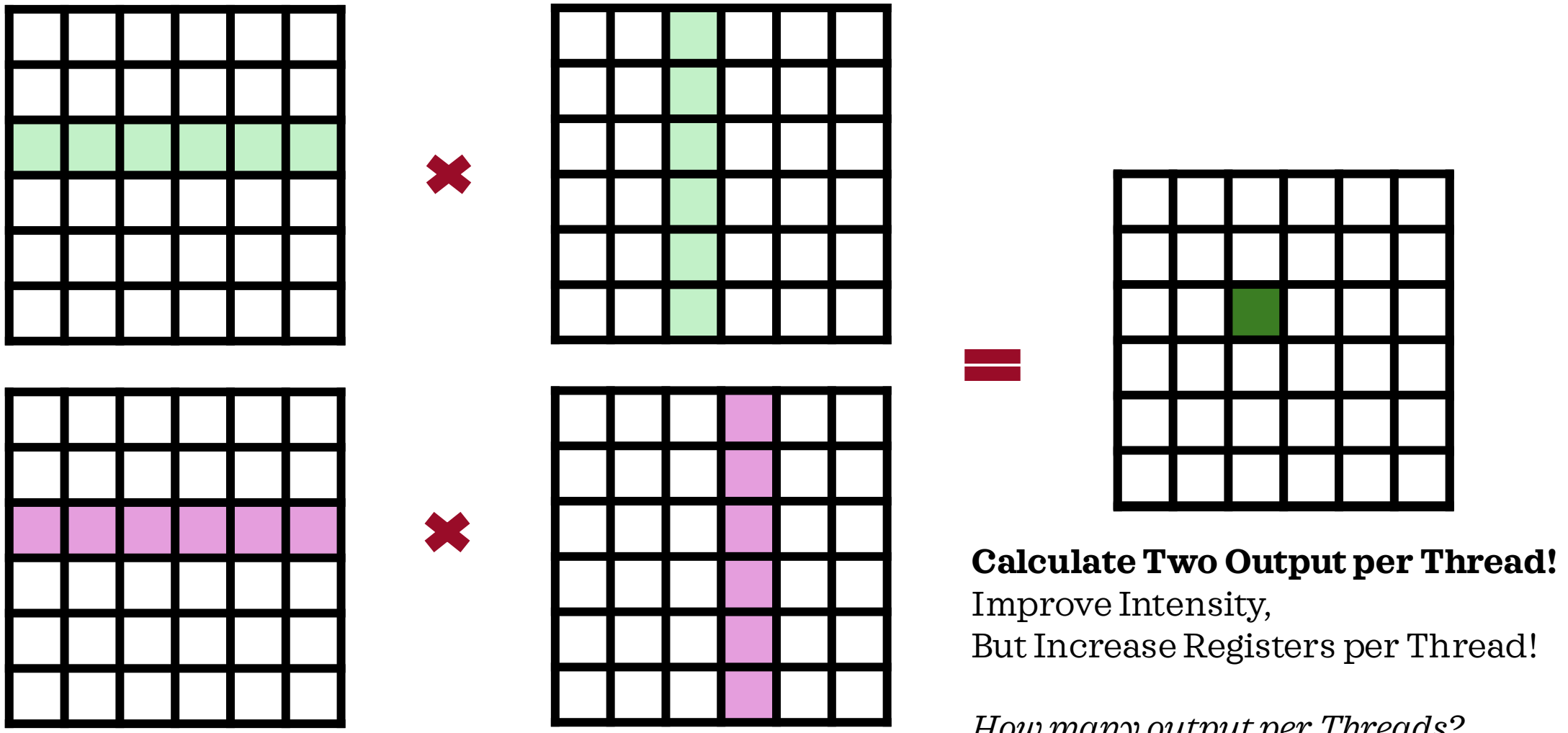


1 Kernels

Batching can improve launch overhead, intensity, occupancy, and coarsening opportunity.

How to layout inputs?
How to bind threads?
How to set batch size?

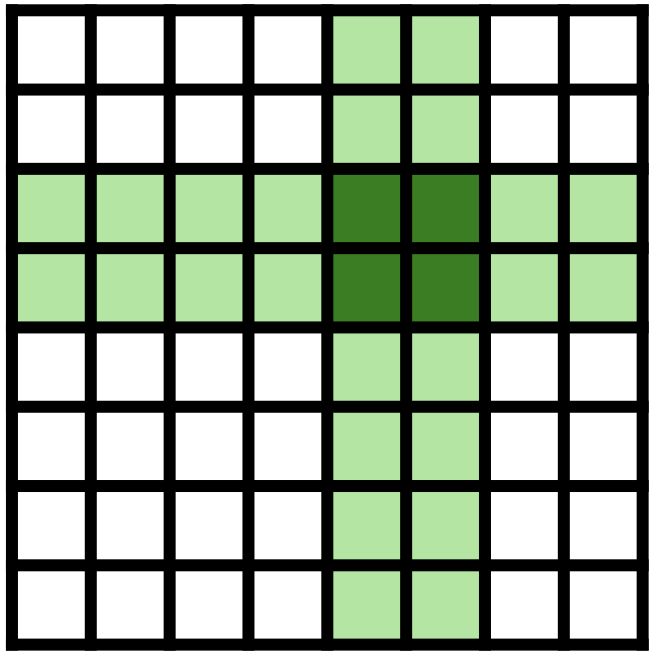
Improving Matmul Intensity - Coarsening



Calculate Two Output per Thread!
Improve Intensity,
But Increase Registers per Thread!

How many output per Threads?

Improving Matmul Intensity - Tiling



Threads Compute Tiles Together!

Threads collaborate to bring data in shared memory!

Compute the Accumulator Changes.

What is Optimal Tile Size?
Is Optimal Tile Size Square?

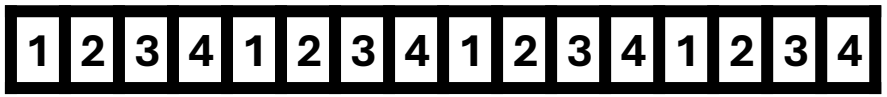
Memory Coalesce

Optimal Layout for Data Access?

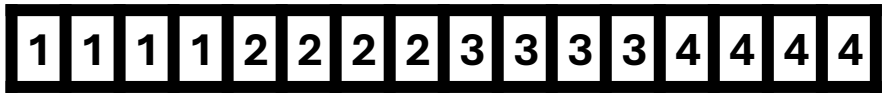
Global Memory Access:
Per Warps, Aligned 32byte chunks as a memory transaction

Neighbouring Threads (warp) Should Read Neighbour Memory at a Time!

A[4×i+thread_id]



A[4×thread_id+i]



Can be 4x to 32x better!

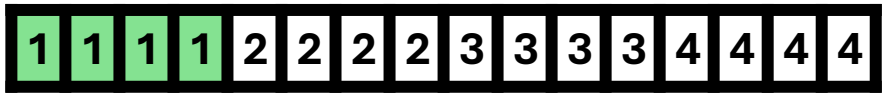
How to Layout Data/Matrices?
How to Bind Threads?

DASlab @ Harvard SEAS

Vectorize Load

Vectorize Load:
GPU Allows Few Load in One Instruction.

Sometimes up to 50% better!



Which One is better?

```
float a0 = x[i+0] + c;  float a1 = x[i+1] + c;
float a2 = x[i+2] + c;  float a3 = x[i+3] + c;
y[i+0] = a0; y[i+1] = a1; y[i+2] = a2; y[i+3] = a3;
```

**More Registers,
More Load/Store Instructions.**

```
float4 v = px[i4];
v.x += c; v.y += c;
v.z += c; v.w += c;
y[i4] = v;
```

**More Registers,
Less Load/Store
Instructions.**

```
y[i+0] = x[i+0] + c;
y[i+1] = x[i+1] + c;
y[i+2] = x[i+2] + c;
y[i+3] = x[i+3] + c;
```

**Less Registers,
More Load/Store
Instructions.**

DASlab @ Harvard SEAS

How Much We Should Vectorize?
We usually need Coarse Threads to Have Vectorize Opportunity.

```
float sum = 0;
for (int k = 0; k < K; k++)
  sum += a[k] * b[k];
```

How to Optimize

```
float sum0=0, sum1=0, sum2=0, sum3=0;
for (int k = 0; k < K; k += 4) {
  sum0 += a[k]   * b[k];
  sum1 += a[k+1] * b[k+1];
  sum2 += a[k+2] * b[k+2];
  sum3 += a[k+3] * b[k+3];
}
float sum = sum0 + sum1 + sum2 + sum3;
```

Less Control/Branch

```
float sum = 0.0f;
for (int i = 0; i < K / 4; i++) {
  float4 va = A[i];
  float4 vb = B[i];
  sum += va.x*vb.x + va.y*vb.y + va.z*vb.z + va.w*vb.w;
}
return sum;
```

Vectorize Load

DASlab @ Harvard SEAS

```
float sum = 0.0f;
for (int i = 0; i < K / 4; i++) {
  float4 va = A[i];
  float4 vb = B[i];
  sum += va.x*vb.x + va.y*vb.y + va.z*vb.z + va.w*vb.w;
}
return sum;
```

How to Optimize ?

Less Register Pressure!

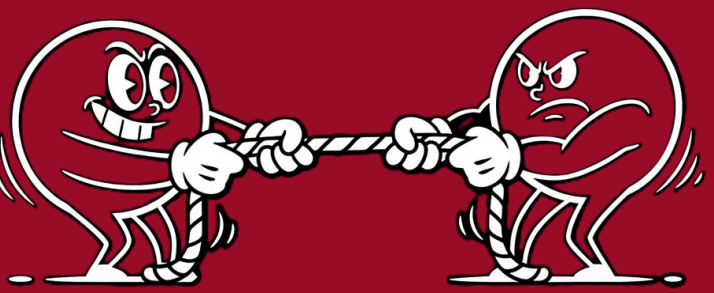
```
float sum = 0;
for (int k = 0; k < K; k++)
  sum += a[k] * b[k];
```

DASlab @ Harvard SEAS

Why so Many People Invest/Work on ML Compilers?

It's All About Trade-Offs.

Latency Throughput



End-To-End Metrics

Occupancy

Arithmetic Intensity

Memory Efficiency

Instruction Efficiency

Performance Parameters

Vectorization

Parallelization

Coarse Factor

Tile Size

Register Pressure

Thread Binding

Cache Policy

#Blocks

Matrix Layout

Loop Tiling

#Threads per Block

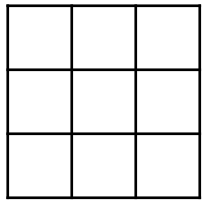
Reduction Strategy

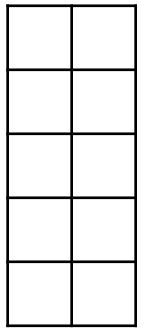
Design Decisions

Self-Designing Kernels

Synthesize Kernel For Target Latency/Throughput

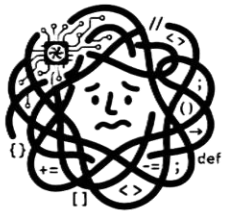
Shape/Input Specific





Hardware Specific

Hard !!!



💡 **First Step To *Self-Designing AI***

Once You Find It, You Can Use It Forever!

DASlab @ Harvard SEAS

Self-Designing Kernels

Representation: Designing *Basis Set* for Space of Possible Kernels for an Operator.

Cost Model: Predict/Measure Performance of a Kernel.

Search: Converge to Good Part of the Space.

OctoAI

TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

Tianqi Chen[1], Thierry Moreau[1], Ziheng Jiang[1,2], Lianmin Zheng[3], Eddie Yan[1]

Meghan Cowan[1], Haichen Shen[1], Leyuan Wang[4,2], Yuwei Hu[5], Luis Ceze[1], Carlos Guestrin[1], Arvind Krishnamurthy[1]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington

[2] AWS, [3]Shanghai Jiao Tong University, [4]UC Davis, [5]Cornell

Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations

Philippe Tillet
Harvard University
USA

H. T. Kung
Harvard University
USA

David Cox
Harvard University, IBM
USA

Ansor: Generating High-Performance Tensor Programs for Deep Learning

Lianmin Zheng [1], Chengfan Jia [2], Minmin Sun [2], Zhao Wu [2], Cody Hao Yu [3],
Ameer Haj-Ali [1], Yida Wang [3], Jun Yang [2], Danyang Zhuo [1,4],
Koushik Sen [1], Joseph E. Gonzalez [1], Ion Stoica [1]

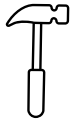
[1] *UC Berkeley,* [2]*Alibaba Group,* [3]*Amazon Web Services,* [4] *Duke University*

DASlab @ Harvard SEAS

How to synthesize/superoptimize/design?

Navigate Space

Register-level Data Movement



Per Instruction: Float Behaviour, Cache Management Hints, Vectorization

Special Instruction: New | Compiler Faults

CUDA → PTX → SASS

On Abstract Machine (JVM)

Forward Compatible

Exposed to us

Actual Physical Machine

Different per Model of GPU

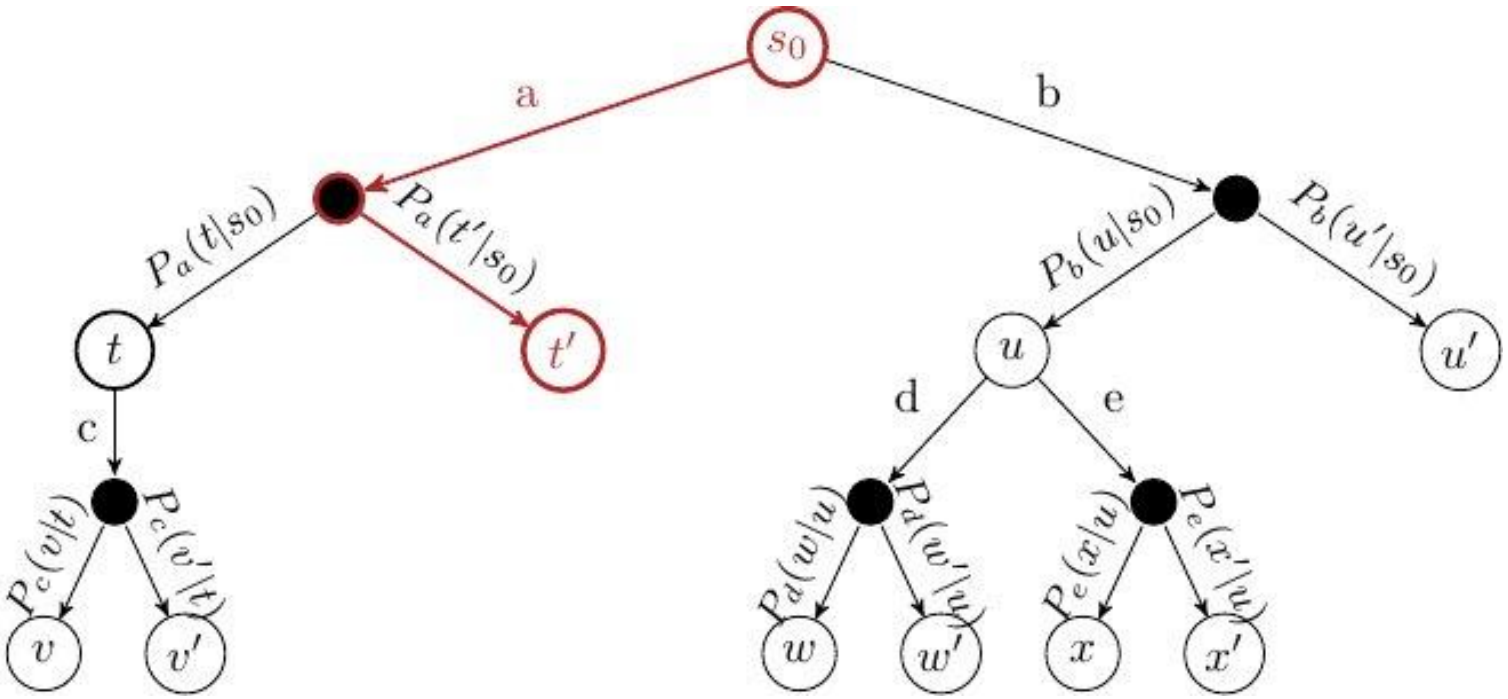
No Documentation for Recent Architectures

What should be our Source/Target Language?

DASlab @ Harvard SEAS

Runtime

Placement
Scheduling
Zero-latency Context Switch



Nodes: An Implementation of Specific Operator

Edges: A Mutation/Change on Program

Goal: Reaching to Good Programs.

DASlab @ Harvard SEAS

```
(a0) = load(A, (i, k), cache = cg, vec = 1)
(b0) = load(B, (k, j), cache = ca, vec = 1)
acc0 = compute_fma(a0, b0, acc0)
...
(b1) = load(B, (k, j+1), cache = ca, vec = 1)
acc1 = compute_fma(a0, b1, acc1)
```

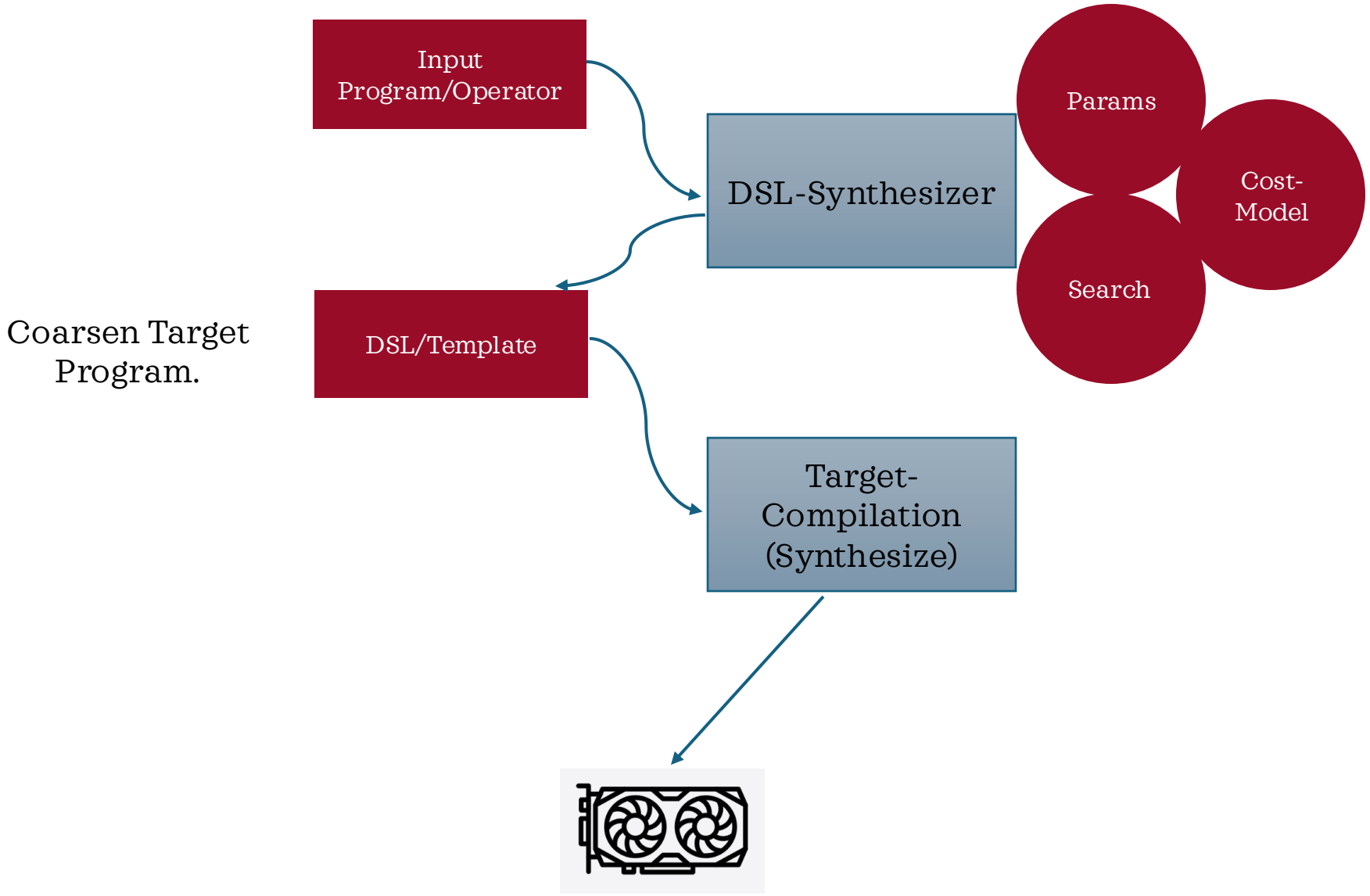
Vectorize Load

```
(a0) = load(A, (i, k), cache = cg, vec = 1)
(b0, b1) = load(B, (k, j), cache = ca, vec = 2)
acc0 = compute_fma(a0, b0, acc0)
...
acc1 = compute_fma(a0, b1, acc1)
```

DASlab @ Harvard SEAS

How to Represent Programs?

Sketch of Any Compiler/Synthesizer



It Is Just the Beginning…

There are many other things to learn about GPUs:

- Syntax!!!
- Many Tensor Core Optimizations
- Multi-GPU Primitives
- Fusion (we discuss it next lecture)
- PTX Optimizations/Tricks

And…

Next Session: What are our LLM workload/operators that use GPU…