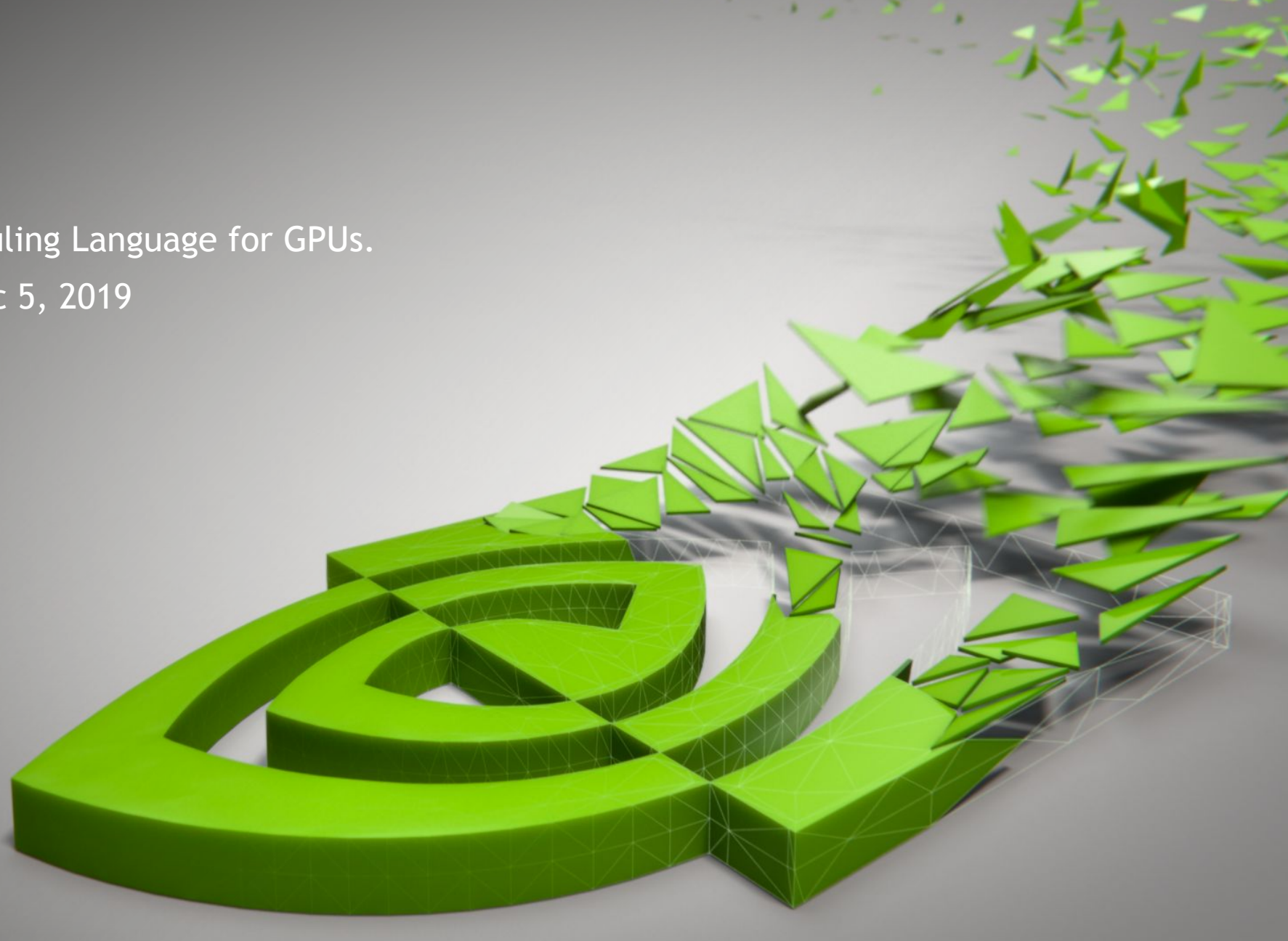
Fireiron - A Scheduling Language for GPUs.

Vinod Grover | Dec 5, 2019



Acknowledgments

Joint work with :
- Bastian Hagedorn
- Sam Elliott
- Henrik Barthels
- Ras Bodik

And contributions from many others at NVIDIA.

OVERVIEW

High Performance DSL for linear algebra on GPUs

A hierarchical scheduling language based on Halide and TVM

- designed to express GPU optimizations for maximum performance

Can directly represent elements of

- storage hierarchy
  - registers, fragments, shared memory
- compute hierarchy
  - threads, warps, blocks, kernels

Can reason about tensorcore and machine level operations.

Suitable for auto-scheduling and auto-tuning

DECOMPOSING MATMUL

Exploiting Hierarchical Structure of GPU Kernels

```
1  __global__ void MatMul(const float A[M * K], const float B[K * N], float C[M * N]) {
2
3    __shared__ float ASH[128][8], BSH[8][128];
4    float ARF[8][1], BRF[1][8], CRF[8][8];
5
6    iBlock ← 128 * blockIdx.x;
7      jBlock ← 128 8 blockIdx.y;
8
9      CRF ← 0;
10
11     for (k ← 0; k < K / 8; k++) {
12
13       GlbToSh(A → ASH (8×128), start at (iBlock, jBlock))
14       GlbToSh(A → BSH (128×8), start at (iBlock, jBlock))
15       __syncthreads();
16
17       iWarp ← iBlock + warpIdx.x * 64;
18         jWarp ← jBlock + warpIdx.y * 32;
19
20         iThread ← iWarp + threadIdx.x * 8;
21           jThread ← jWarp + threadIdx.y * 8
22
23           for (kk ← 0; kk < 8; kk++)
24
25             ShToPvt(ASH → ARF (8×1), start at (iThread, jThread))
26             ShToPvt(BSH → BRF (1×8), start at (iThread, jThread))
27
28             for (i ← 0; i < 8; i ++)
29               for (j ← 0; j < 8; j++)
30
31                 CRF[i][j] += ARF[i][0] * BRF[0][j];
32
33               endfor
34             endfor
35
36         endfor
37
38
39     endfor
40
41     PvtToGlb(CRF → C (128×128), start at iBlock, jBlock)
42
43 } // end kernel
```

implements

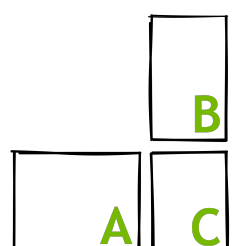
sizes of matrices *location of A, B and C* *responsible level of compute hierarchy*

`MatMul(M,N,K)(GL,GL,GL)(Kernel)`

Matrix Multiplication Kernel

Describing the problem this box implements

Hierarchical Structure:
Original Problem is **decomposed** into
“smaller” instances of the same type of problem



DECOMPOSING MATMUL

Exploiting Hierarchical Structure of GPU Kernels

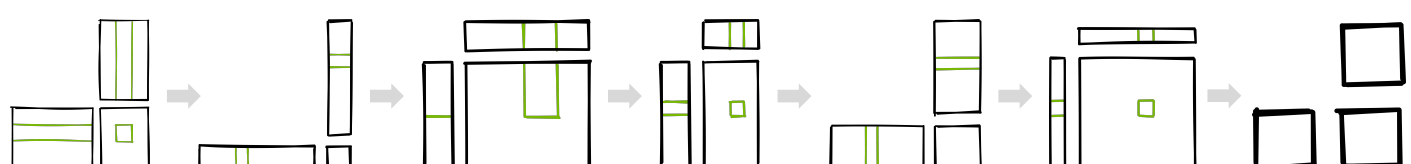
```
1  __global__ void MatMul(const float A[M * K], const float B[K * N], float C[M * N]) {
2
3    __shared__ float ASH[128][8], BSH[8][128];
4    float ARF[8][1], BRF[1][8], CRF[8][8];
5
6    iBlock ← 128 * blockIdx.x;
7      jBlock ← 128 8 blockIdx.y;
8
9      CRF ← 0;
10
11     for (k ← 0; k < K / 8; k++) {
12
13       GlbToSh(A → ASH (8×128), start at (iBlock, jBlock))
14       GlbToSh(A → BSH (128×8), start at (iBlock, jBlock))
15       __syncthreads();
16
17       iWarp ← iBlock + warpIdx.x * 64;
18         jWarp ← jBlock + warpIdx.y * 32;
19
20           iThread ← iWarp + threadIdx.x * 8;
21             jThread ← jWarp + threadIdx.y * 8
22
23             for (kk ← 0; kk < 8; kk++)
24
25               ShToPvt(ASH → ARF (8×1), start at (iThread, jThread))
26               ShToPvt(BSH → BRF (1×8), start at (iThread, jThread))
27
28               for (i ← 0; i < 8; i ++)
29                 for (j ← 0; j < 8; j++)
30
31                   CRF[i][j] += ARF[i][0] * BRF[0][j];
32
33                 endfor
34               endfor
35
36             endfor
37
38
39     endfor
40
41     PvtToGlb(CRF → C (128×128), start at iBlock, jBlock)
42
43 } // end kernel
```

implements

sizes of matrices	*location of A, B and C*	*responsible level of compute hierarchy*
MatMul(M,N,K)	(GL,GL,GL)	(Kernel)
MatMul(128,128,K)	(GL,GL,GL)	(Block)
MatMul(128,128,K)	(GL,GL,RF)	(Block)
MatMul(128,128,8)	(GL,GL,RF)	(Block)
MatMul(128,128,8)	(SH,SH,RF)	(Block)
MatMul(64,32,8)	(SH,SH,RF)	(Warp)
MatMul(8,8,8)	(SH,SH,RF)	(Thread)
MatMul(8,8,1)	(SH,SH,RF)	(Thread)
MatMul(8,8,1)	(RF,RF,RF)	(Thread)
MatMul(1,1,1)	(RF,RF,RF)	(Thread)

Describing the problem this box implements

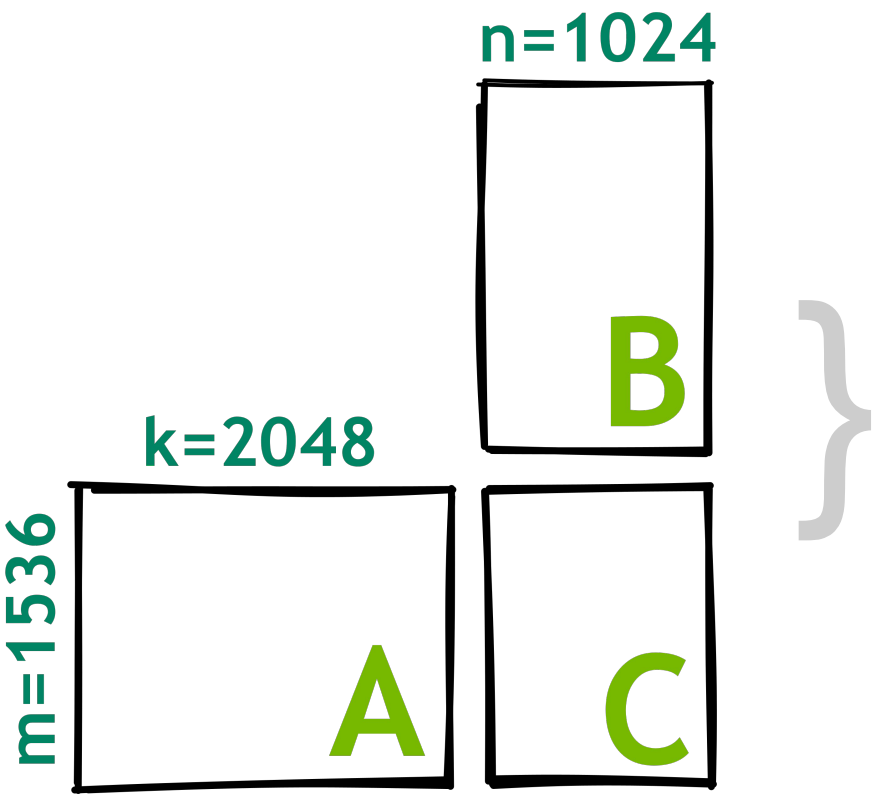
Hierarchical Structure:
Original Problem is **decomposed** into
“smaller” instances of the same type of problem



Matrix Multiplication Kernel

INTRODUCTION

GEMM Spec(ification)



Specs define the current problem to optimize

Fireiron MatMul Spec

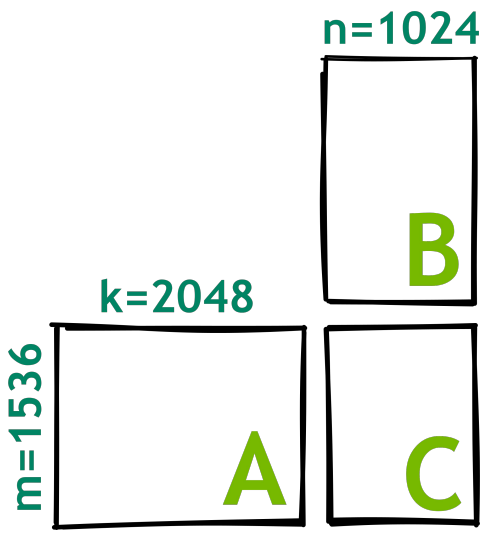
```
MatMul(Kernel,
    A: Matrix(1536,2048,GL,FP32,RowMajor),
    B: Matrix(2048,1024,GL,FP32,ColMajor),
    C: Matrix(1536,1024,GL,FP32,ColMajor))
```

and contain enough information to fully describe it

Idea: *A programmer should be able to provide a valid implementation for a given spec!*

INTRODUCTION

Working with Specs



Fireiron MatMul Spec

```
MatMul(Kernel,
      A: Matrix(1536,2048,GL,FP32,RowMajor),
      B: Matrix(2048,1024,GL,FP32,ColMajor),
      C: Matrix(1536,1024,GL,FP32,ColMajor))
```

Goal: Generate high-performance MatMul Kernel
-> We start with Kernel-level Spec

Given a Spec, you can:

a) Provide a handwritten microkernel, or
b) Arrive at an *executable* Spec, or
c) *Decompose* it into a “smaller” spec

INTRODUCTION

Working with Specs

Fireiron MatMul Spec

```
MatMul(Kernel,
      A: Matrix(1536,2048,GL,FP32,RowMajor),
      B: Matrix(2048,1024,GL,FP32,ColMajor),
      C: Matrix(1536,1024,GL,FP32,ColMajor))
```

Goal: Generate high-performance MatMul Kernel
-> We start with Kernel-level Spec

Given a Spec, you can:

a) Provide a handwritten microkernel, or
b) Arrive at an *executable* Spec, or
c) *Decompose* it into a “smaller” spec

DECOMPOSITIONS

Halide-like transformations constructing the IR

Every *Decomposition*:

1. is a function: *Spec -> Spec* (returning a “smaller” subspec)
2. provides a *partial implementation* to our code generator

Two Main Decompositions:

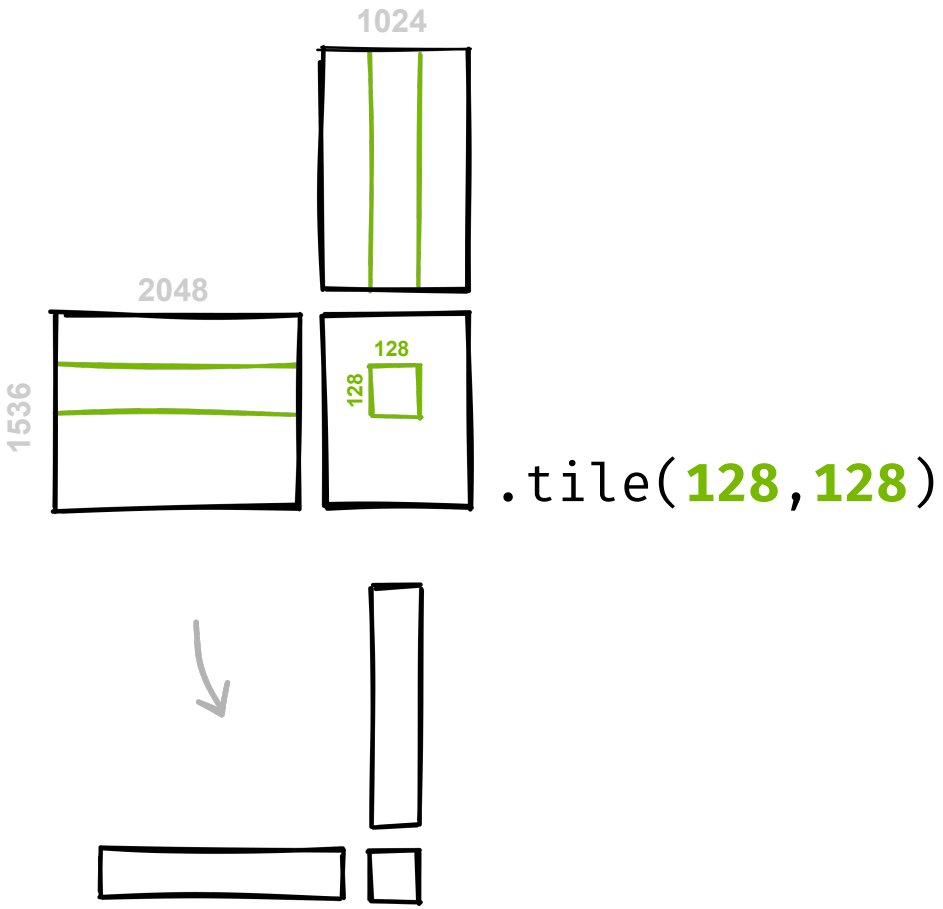
- **`.tile(m,n)`** - enables descending the compute-hierarchy
- `.load(matrix, loc, impl)` - enables descending the memory hierarchy

We allow to define operation-specific Decompositions:

- **`.split(k)`**
- `.epilog(...)`
- `...`

DESCENDING THE COMPUTE HIERARCHY

.tile(m,n)



Current Spec

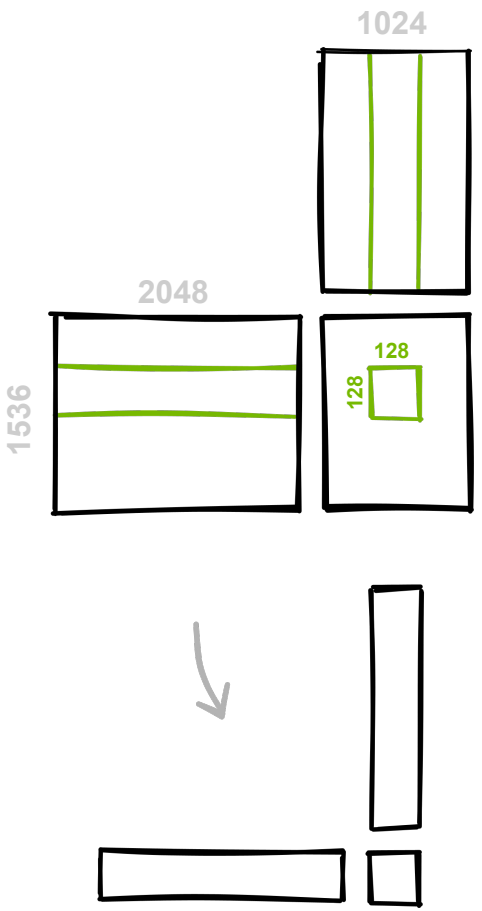
```
MatMul(Kernel,
     A:Matrix(1536,2048,GL,FP32,RowMajor),
     B:Matrix(2048,1024,GL,FP32,ColMajor),
     C:Matrix(1536,1024,GL,FP32,ColMajor))
```

New Spec

```
MatMul(Kernel,
     A:Matrix(128 ,2048,GL,FP32,RowMajor),
     B:Matrix(2048, 128,GL,FP32,ColMajor),
     C:Matrix(128 , 128,GL,FP32,ColMajor))
```

DESCENDING THE COMPUTE HIERARCHY

.tile(m,n)



.tile(**128,128**)
.to(**Block**)

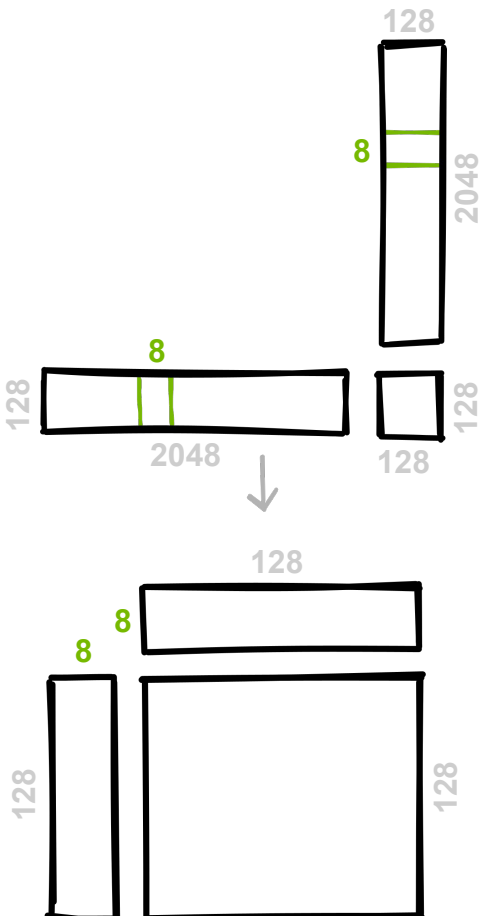
“Refinement”:
Adding implementation details

```
__global__ void MatMul(const float A[M * K], const float B[K * N], float C[M * N]) {
  MatMul(M,N,K)(GL,GL,GL)(Kernel)
}
```

```
__global__ void MatMul(const float A[M * K], const float B[K * N], float C[M * N]) {
  iBlock ← 128 * blockIdx.x;
    jBlock ← 128 * blockIdx.y;
  MatMul(128,128,K)(GL,GL,GL)(Block)
}
```

OUTER PRODUCT BLOCKED GEMM

.split(*kBlock*)



.split(**8**)

Current Spec

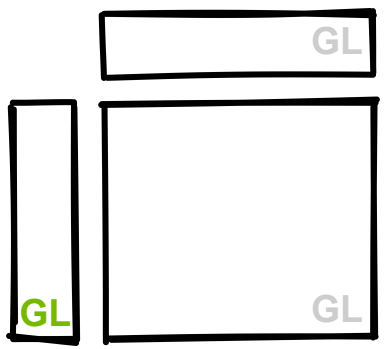
```
MatMul(Block,
     A:Matrix(128,2048,GL,FP32,RowMajor),
     B:Matrix(2048,128,GL,FP32,ColMajor),
     C:Matrix(128 ,128,GL,FP32,ColMajor))
```

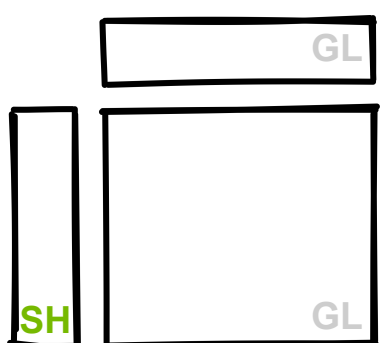
New Spec

```
MatMul(Block,
     A:Matrix(128,  8,GL,FP32,RowMajor),
     B:Matrix(8   ,128,GL,FP32,ColMajor),
     C:Matrix(128,128,GL,FP32,ColMajor))
```

DESCENDING THE MEMORY HIERARCHY

.load(*Matrix, Location, Strategy*)





`.load(A,SH,strategy)`

new Spec describing data movement

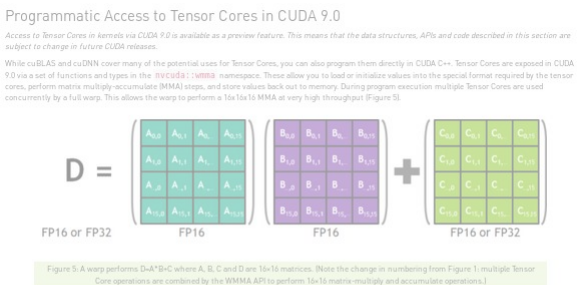
this spec is decomposed with the given strategy

```
MatMul(128,128,8)(GL,GL,GL)(Block)
```

```
for(/* iterate over A */) {
    Move(A)(GL→SH)(Block)
}
__syncthreads();
{
    MatMul(128,128,8)(SH,GL,GL)(Block)
}
```

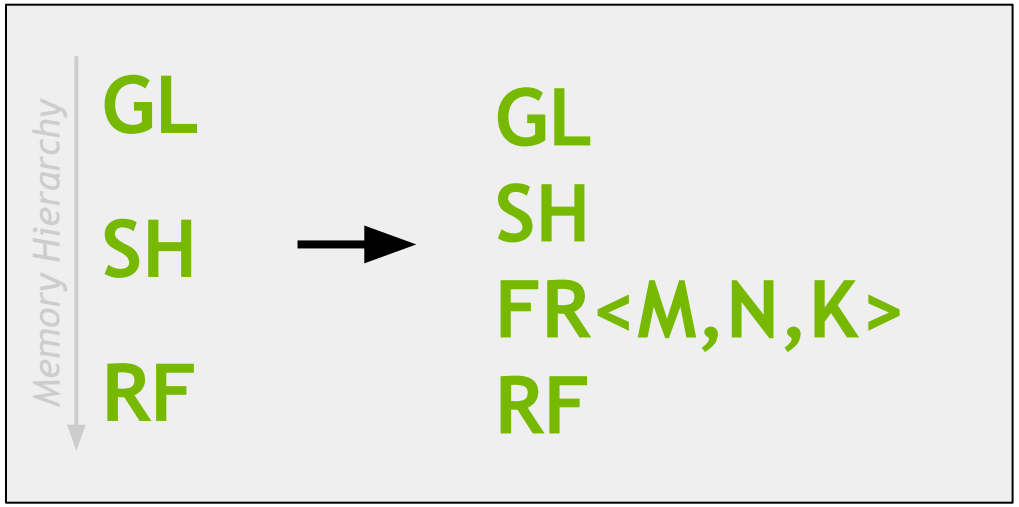
WMMA IN FIREIRON

adding support for CUDA's WMMA API



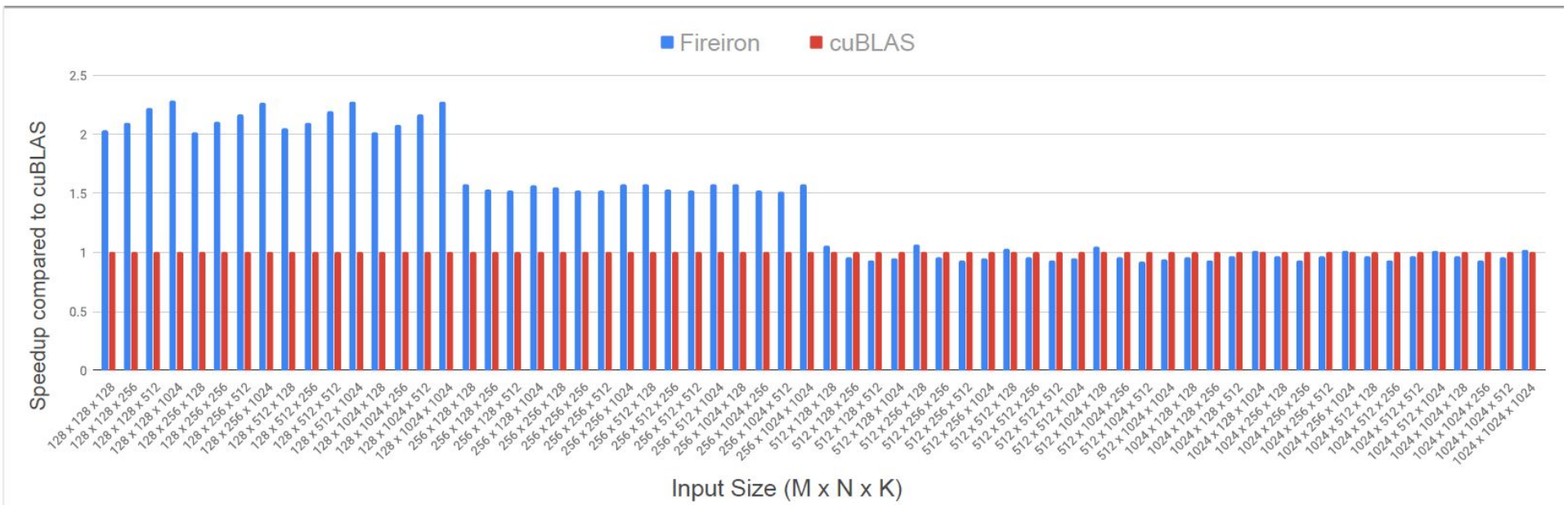

```
// Declare the fragments
wmma::fragment<wmma::matrix_a, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> a_frag;
wmma::fragment<wmma::matrix_b, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> b_frag;
wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> acc_frag;
wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> c_frag;
```

"Before the MMA operation is performed the operand matrices must be represented in the registers of the GPU. As an MMA is a warp-wide operation these **registers are distributed amongst the threads of a warp** with each thread holding a *fragment* of the overall matrix."



Updating Fireiron's Memory Hierarchy

fp16 performance on Volta



QUESTIONS?