

Parallel Processing and Applied Mathematics  
September 11-14, 2011  
Toruń, Poland

# Scientific Computing with GPUs Autotuning GEMMs – Fermi GPUs

Innovative Computing Laboratory  
Electrical Engineering and Computer Science  
University of Tennessee

Jakub Kurzak (presenter)  
Stanimire Tomov  
Jack Dongarra (CINC)

# TOC

- Motivation
- Optimizations Recap
- GEMM Basics
- CUDA GEMM
- **Search Space & Its Pruning**
- Success Story & ASTRA

difficulty:

LOL



WTH

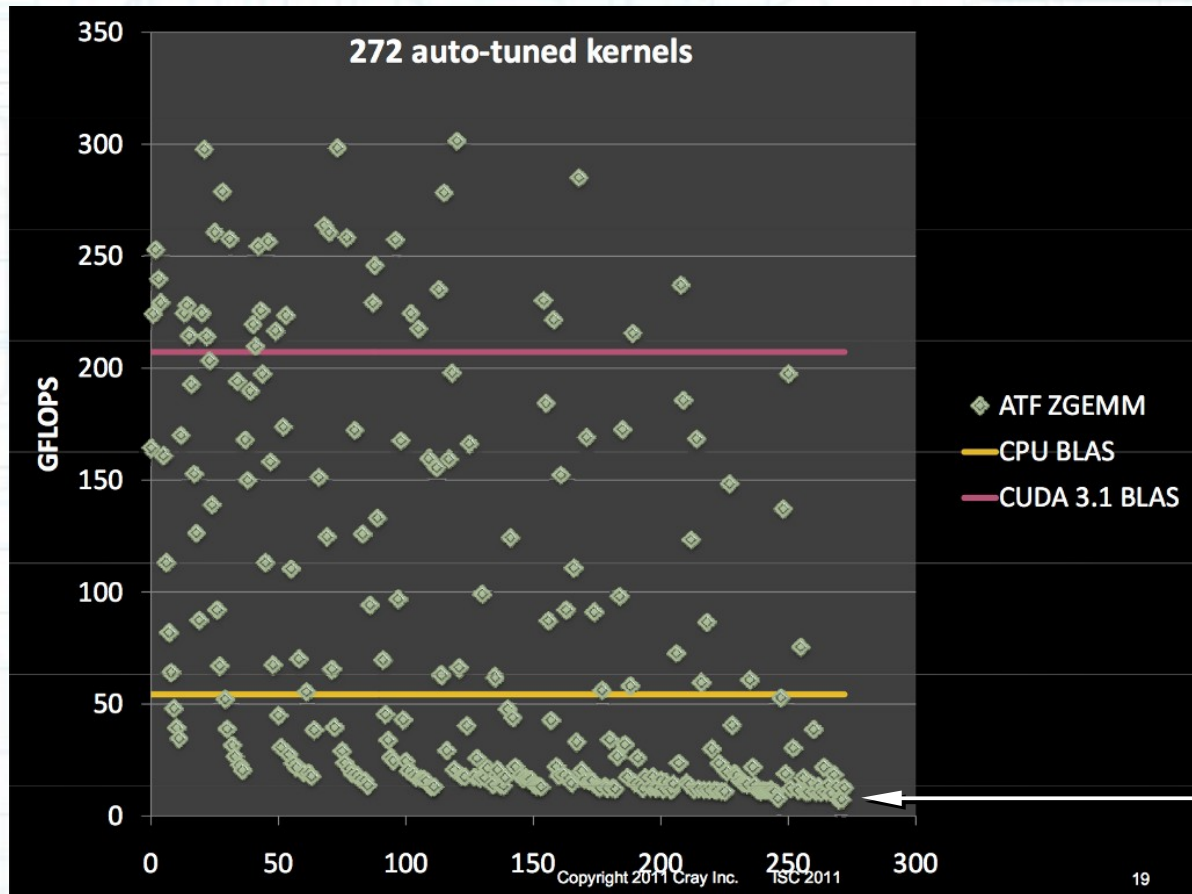


OMG

# Motivation

## Vendors Autotune

Cray ZGEMM autotuning sweep



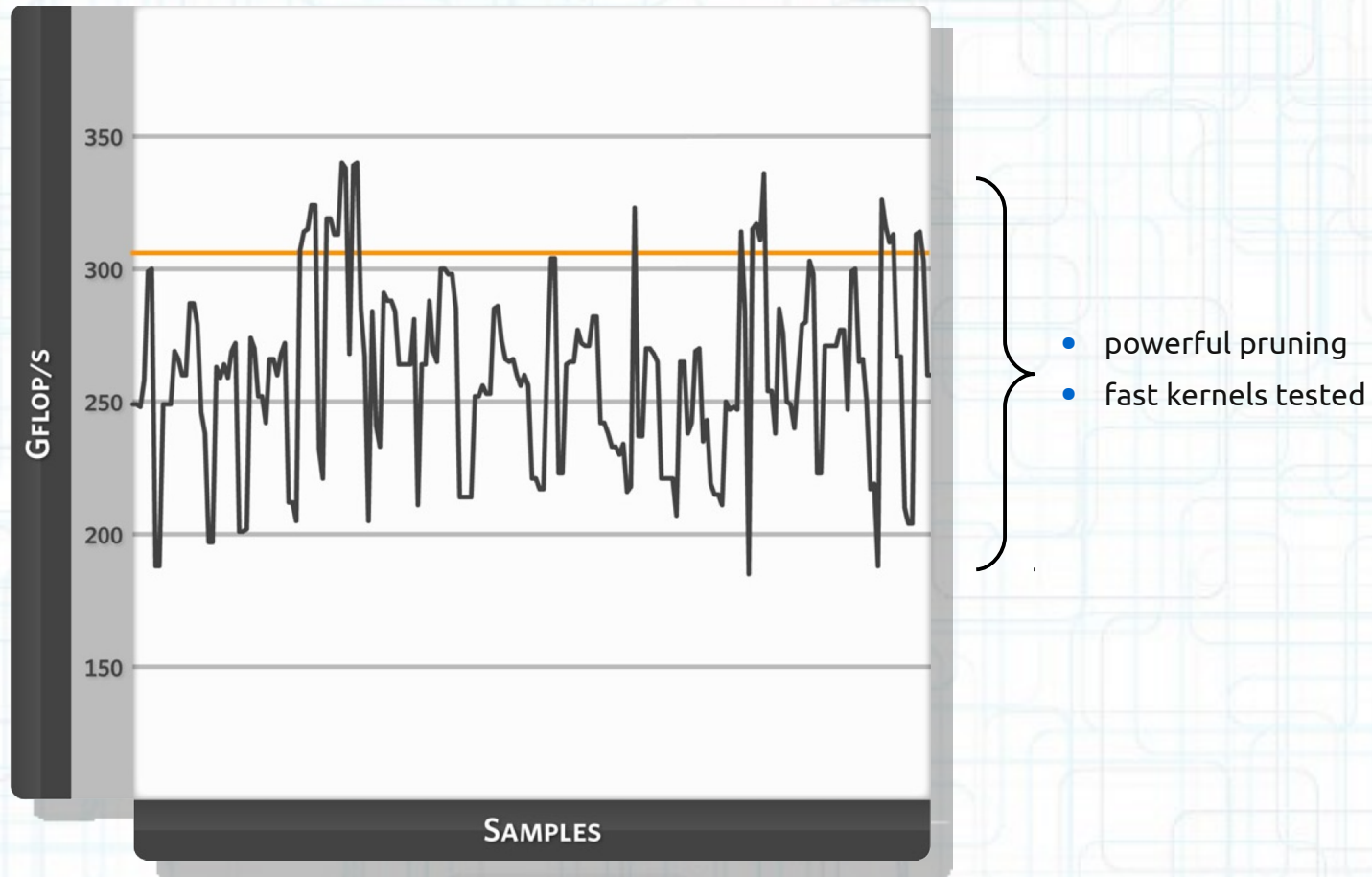
- no complex pruning
- slow kernels tested

272 samples

# Motivation

## Academics Autotune

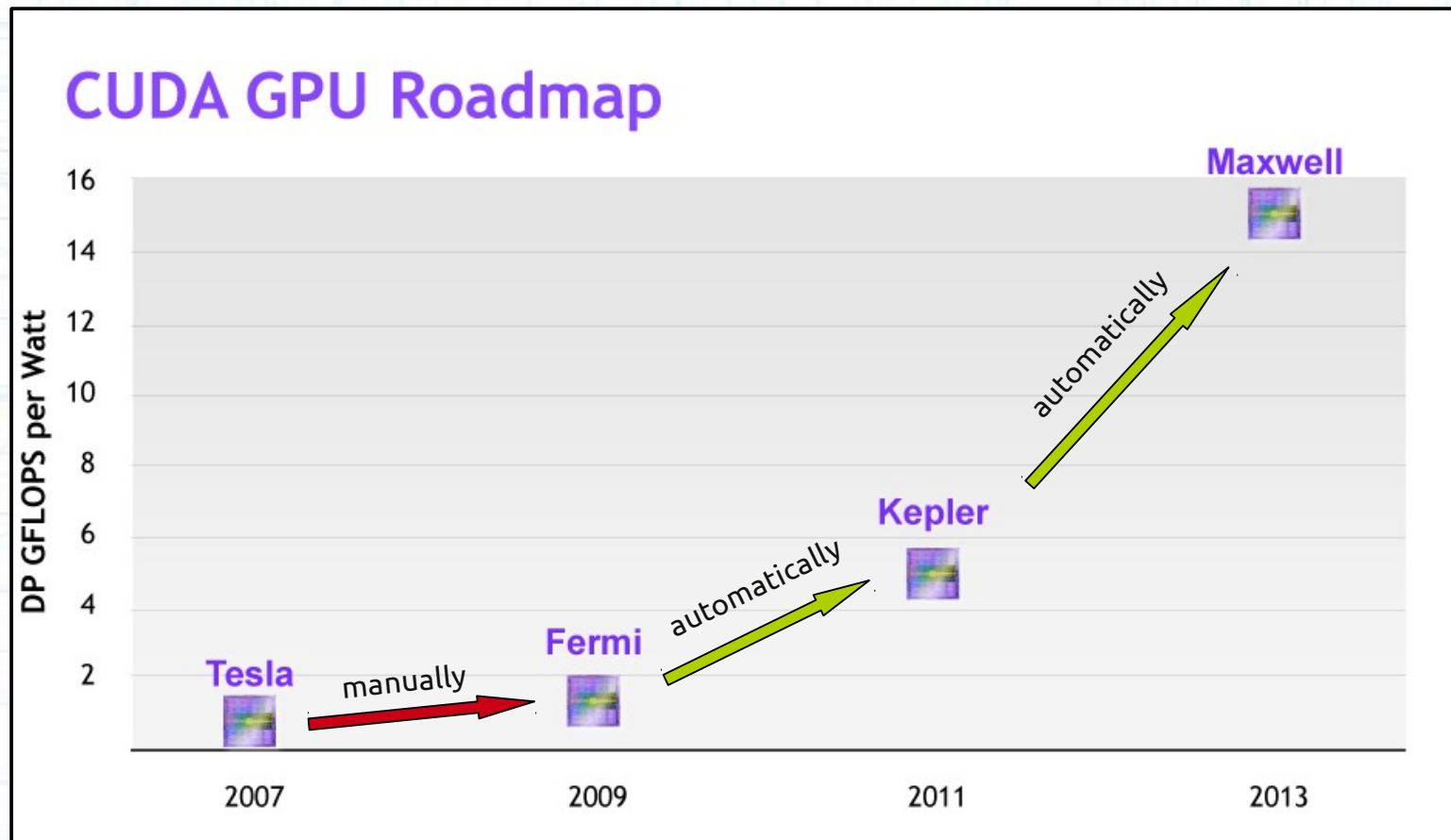
UTK ZGEMM autotuning sweep



211 samples

# Motivation

## Academically Sustainable Development



for MAGMA, PLASMA, PULSAR, ...

# CUDA Optimization

## a 31-legged race

- Thread Divergence
  - different threads in the same warp taking different conditional branches
- Warp Serialization
  - different threads in the same warp competing for the same bank of shared memory
- Memory Coalescing
  - different threads in the same warp accessing different cache lines
- Partition Camping
  - unbalanced access to memory partitions



# CUDA Optimization

## a 31-legged race

- Occupancy
  - create a massive number of threads
- Register Pressure
  - reuse data in registers
- Memory Pressure
  - reuse data in shared memory



# BLAS GEMM

## Definition

$$\mathbf{C} = \alpha \mathbf{A} \times \mathbf{B} + \beta \mathbf{C}$$

```
void cblas_xgemm (  
    const enum CBLAS_ORDER Order,  
    const enum CBLAS_TRANSPOSE TransA,  
    const enum CBLAS_TRANSPOSE TransB,  
    const int M,  
    const int N,  
    const int K,  
    const SCALAR alpha,  
    const TYPE *A,  
    const int lda,  
    const TYPE *B,  
    const int ldb,  
    const SCALAR beta,  
    TYPE *C,  
    const int ldc)
```

# BLAS GEMM

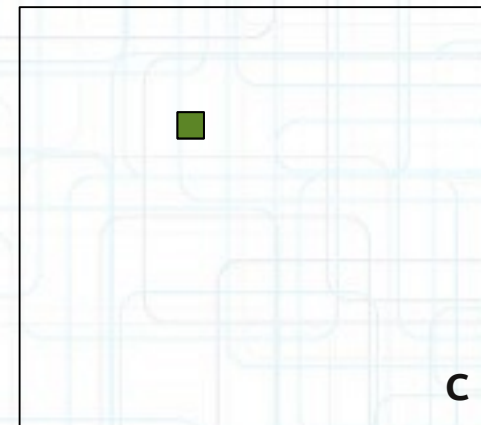
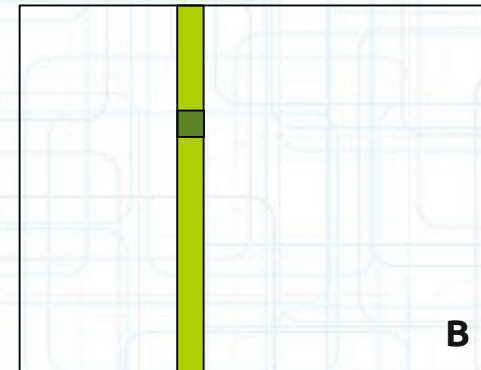
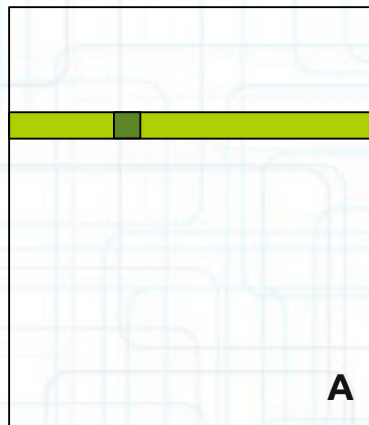
## Canonical Form

**FOR** each element

**FOR** each element

**FOR** each element

$$C = \alpha A \times B + \beta C$$



# BLAS GEMM

## Tiling

**FOR** each tile

**FOR** each tile

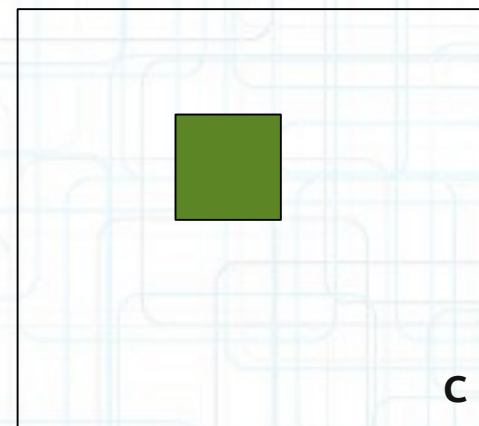
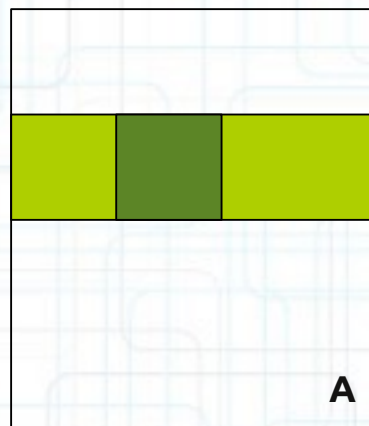
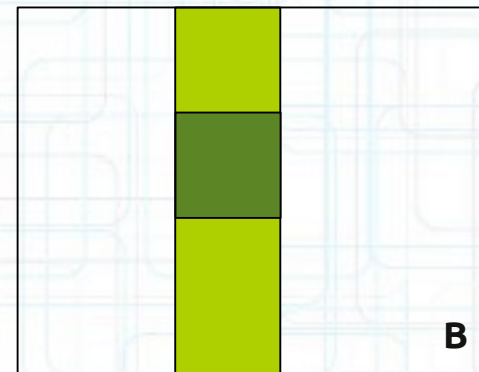
**FOR** each tile

**FOR** each element

**FOR** each element

**FOR** each element

$$C = \alpha A \times B + \beta C$$



# BLAS GEMM

## Unrolling

**FOR** each tile  
**FOR** each tile  
**FOR** each tile

**FOR** each element  
**FOR** each element  
**FOR** each element

$$C = \alpha A \times B + \beta C$$

$$C = \alpha A \times B + \beta C$$

$$C = \alpha A \times B + \beta C$$

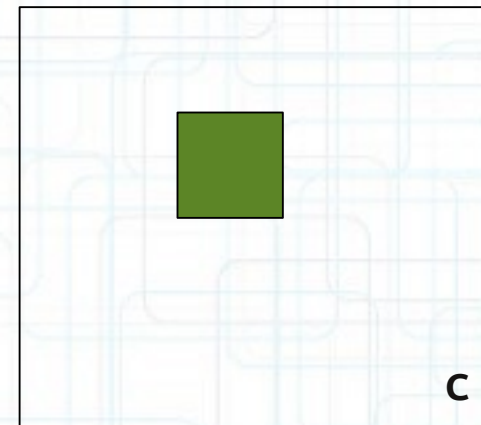
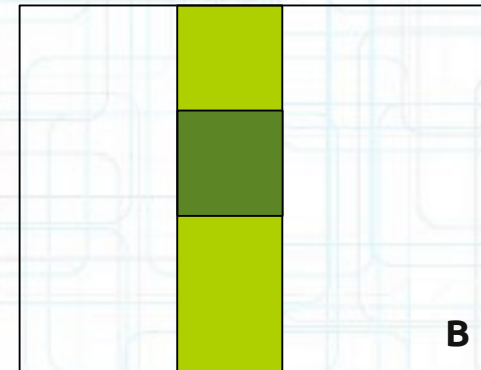
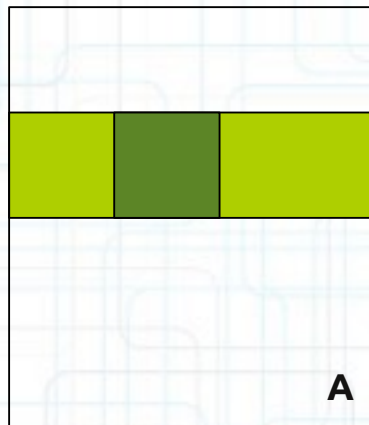
$$C = \alpha A \times B + \beta C$$

$$C = \alpha A \times B + \beta C$$

$$C = \alpha A \times B + \beta C$$

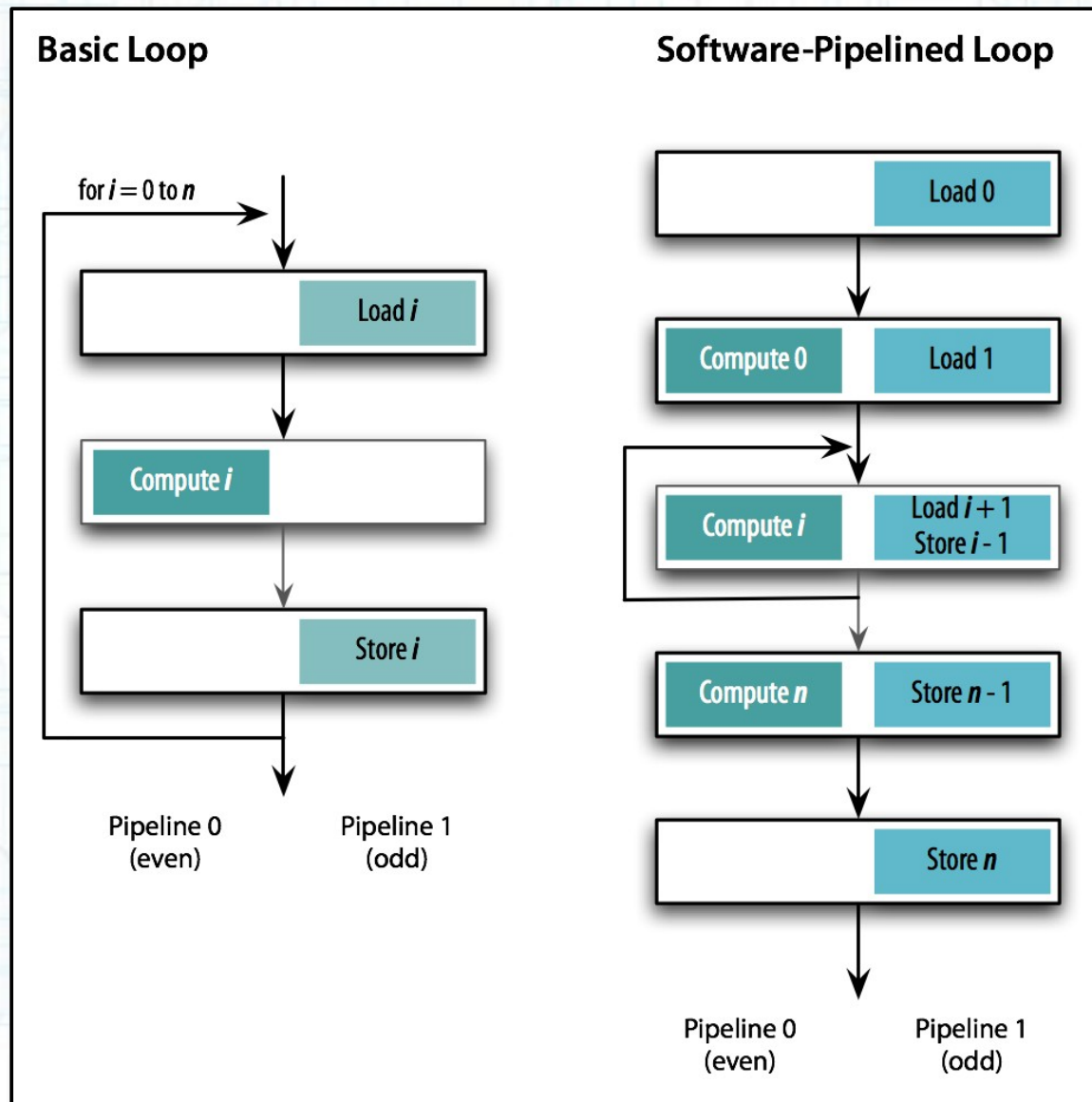
$$C = \alpha A \times B + \beta C$$

$$C = \alpha A \times B + \beta C$$



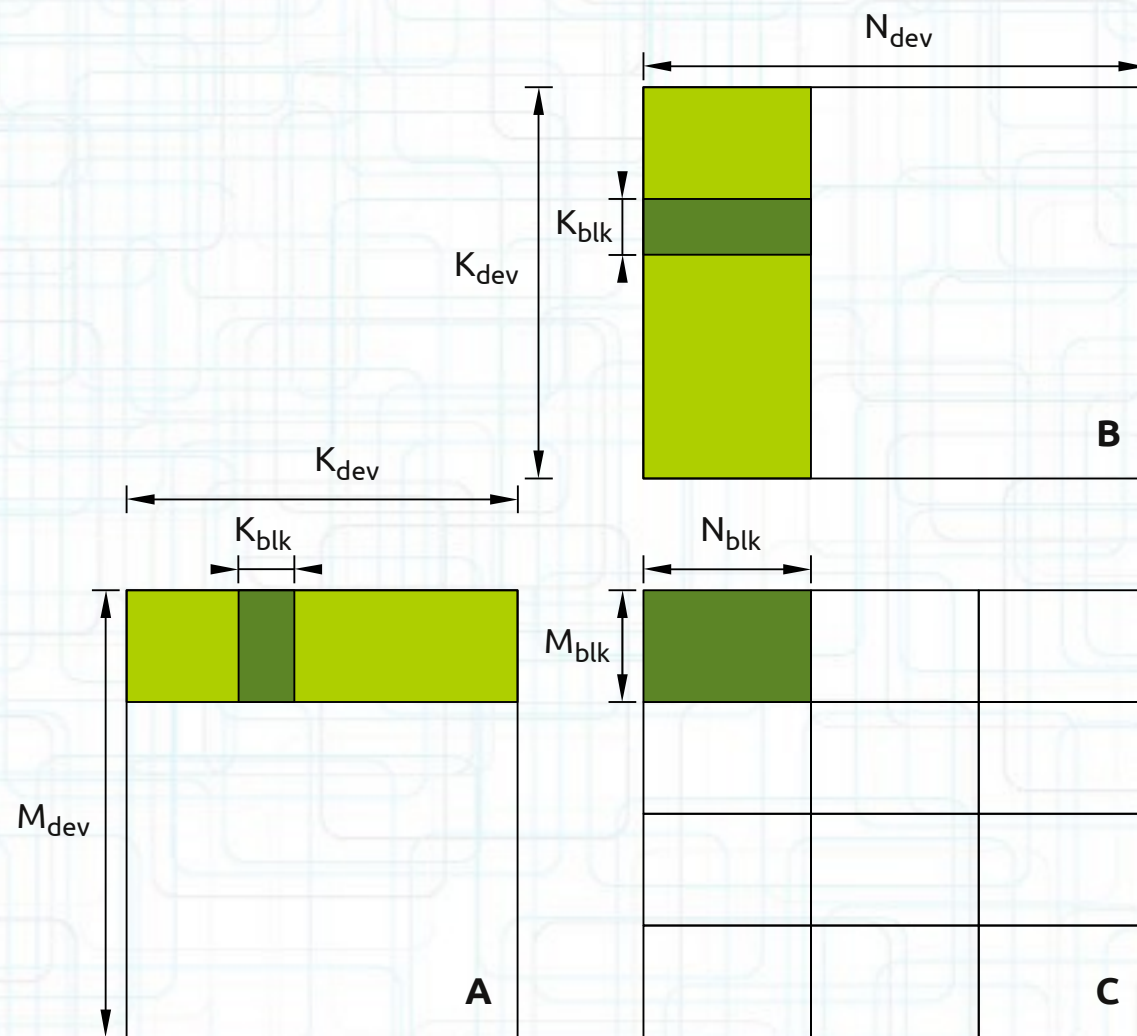
# BLAS GEMM

## Double-Buffering



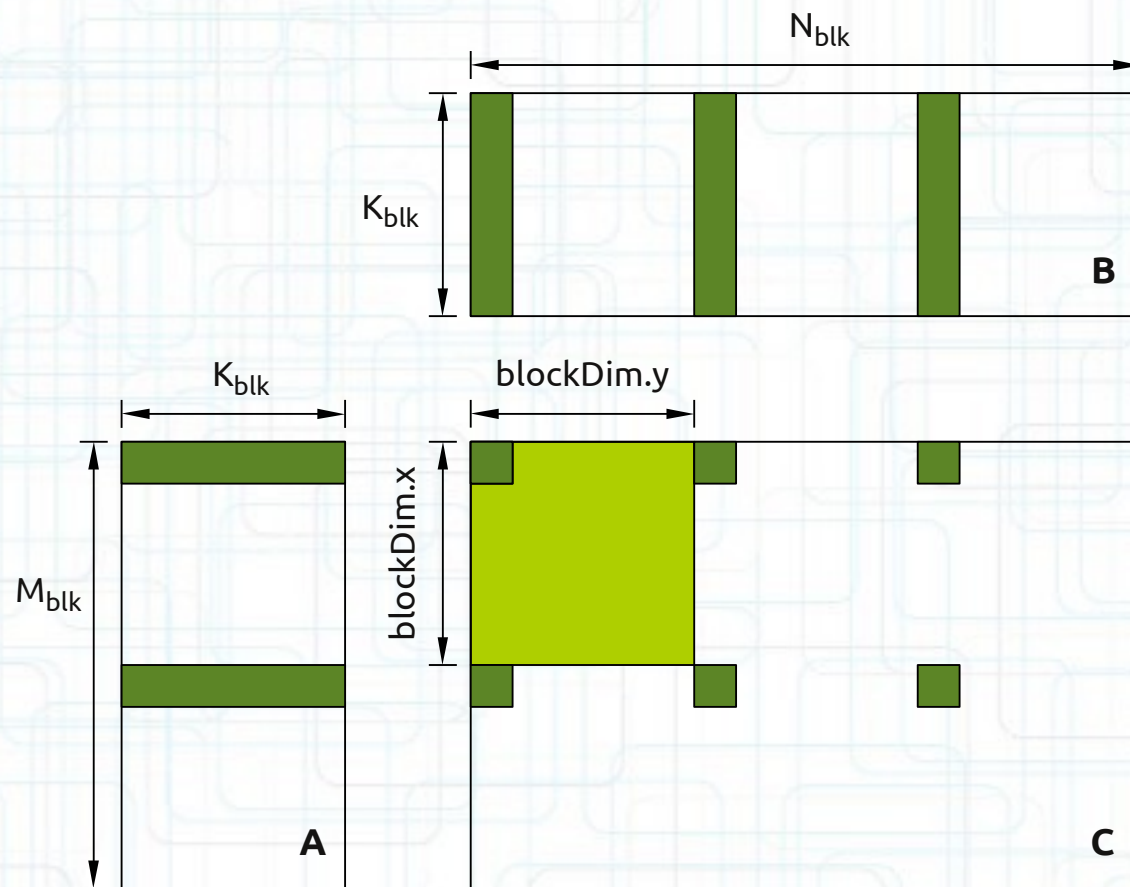
# CUDA GEMM

## Device Level



# CUDA GEMM

## Thread-Block Level

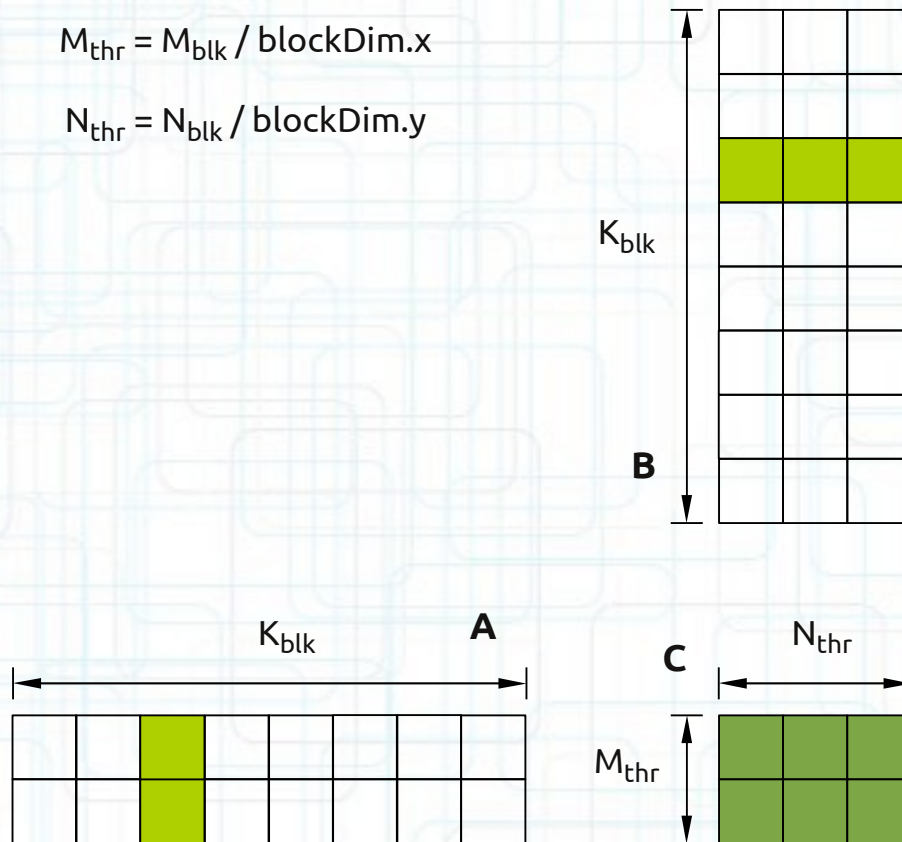


# CUDA GEMM

## Thread Level

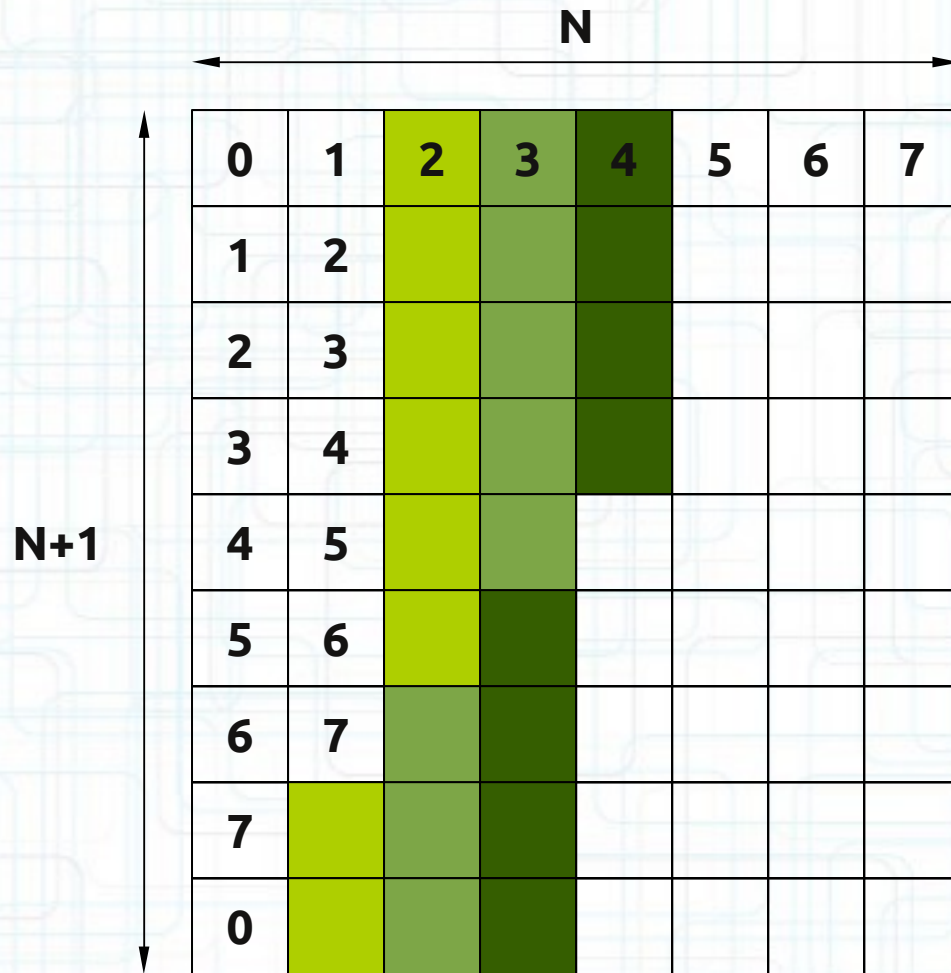
$$M_{\text{thr}} = M_{\text{blk}} / \text{blockDim.x}$$

$$N_{\text{thr}} = N_{\text{blk}} / \text{blockDim.y}$$



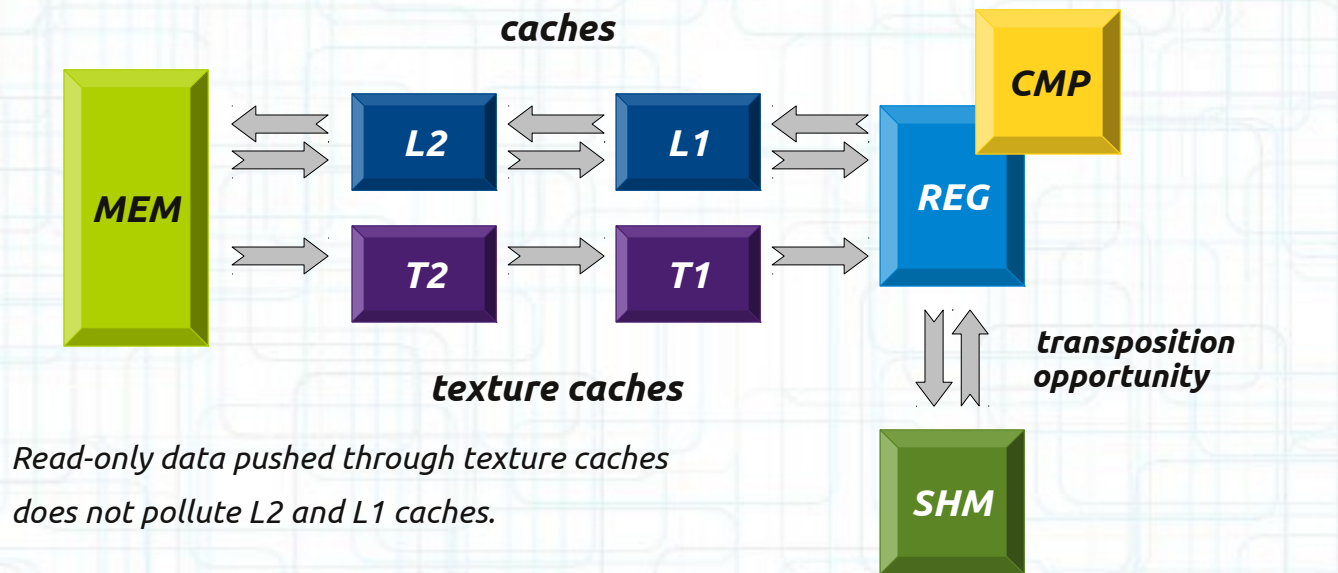
# CUDA GEMM

## Shared Memory Skewing



# CUDA GEMM

## Data Flow



device memory



shared memory



registers

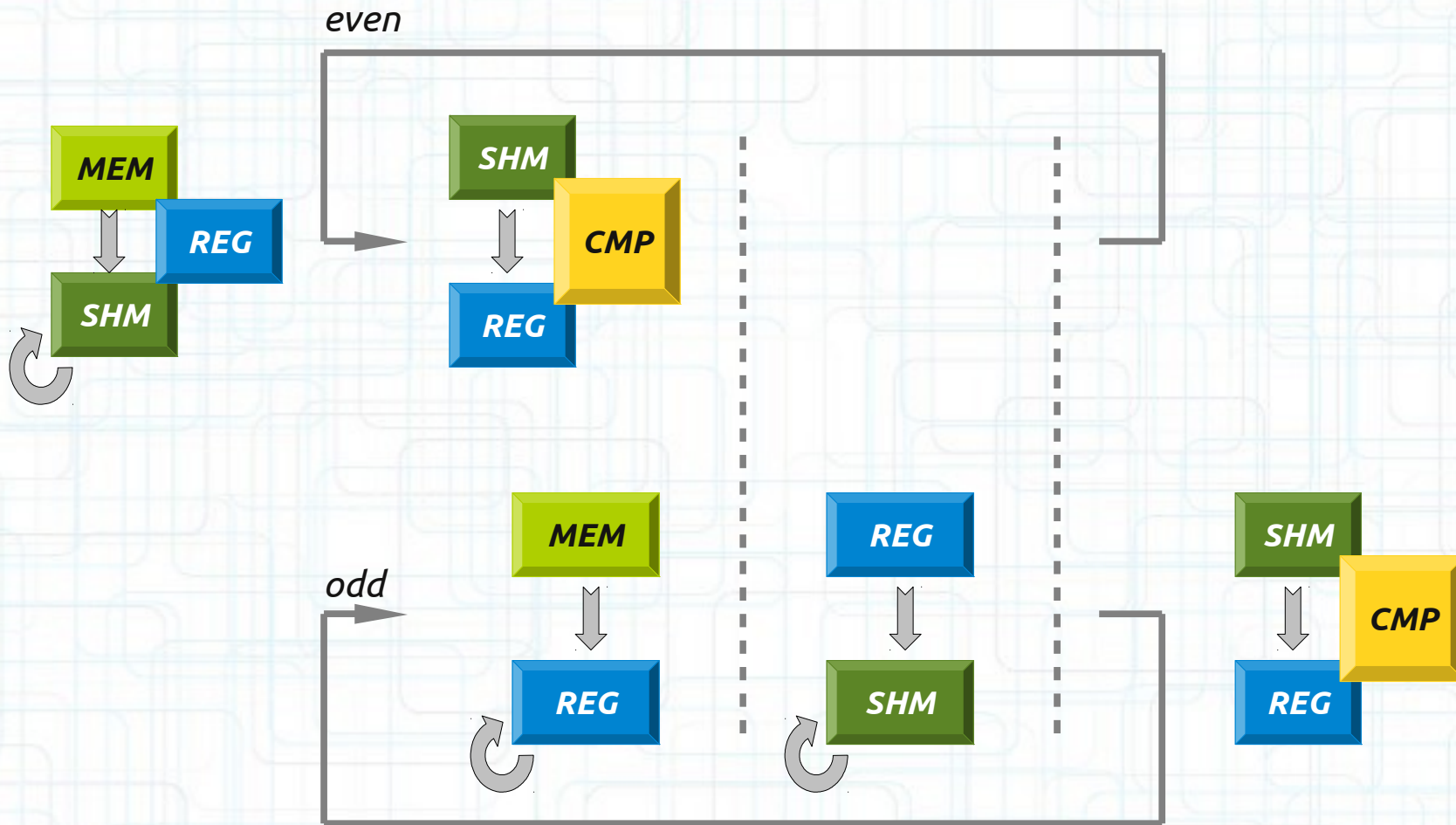


compute



# CUDA GEMM

## Double-Buffering



device memory



shared memory



registers



compute



syncthreads



transposition opportunity



# Texture Reads Double Precision Example

```
#ifdef TEXTURE_2D
```

```
static __device__  
double tex_fetch(texture<int2, 2> tex_ref, int coord_x, int coord_y)  
{  
    int2 v = tex2D(tex_ref, coord_x, coord_y);  
    return __hiloint2double(v.y, v.x);  
}
```

```
#else
```

```
#ifdef TEXTURE_1D
```

```
static __device__  
double tex_fetch(texture<int2> tex_ref, int coord)  
{  
    int2 v = tex1Dfetch(tex_ref, coord);  
    return __hiloint2double(v.y, v.x);  
}
```

```
#endif
```

```
#endif
```

```
////////////////////////////////////
```

```
#ifdef TEXTURE_2D
```

```
#define fetch(A, m, n) tex_fetch(tex_ref_##A, coord_##A##x+m, coord_##A##y+n)
```

```
#else
```

```
#ifdef TEXTURE_1D
```

```
#define fetch(A, m, n) tex_fetch(tex_ref_##A, coord_##A + n*LD##A+m)
```

```
#else
```

```
#define fetch(A, m, n) ofs_d##A[n*LD##A+m]
```

```
#endif
```

```
#endif
```

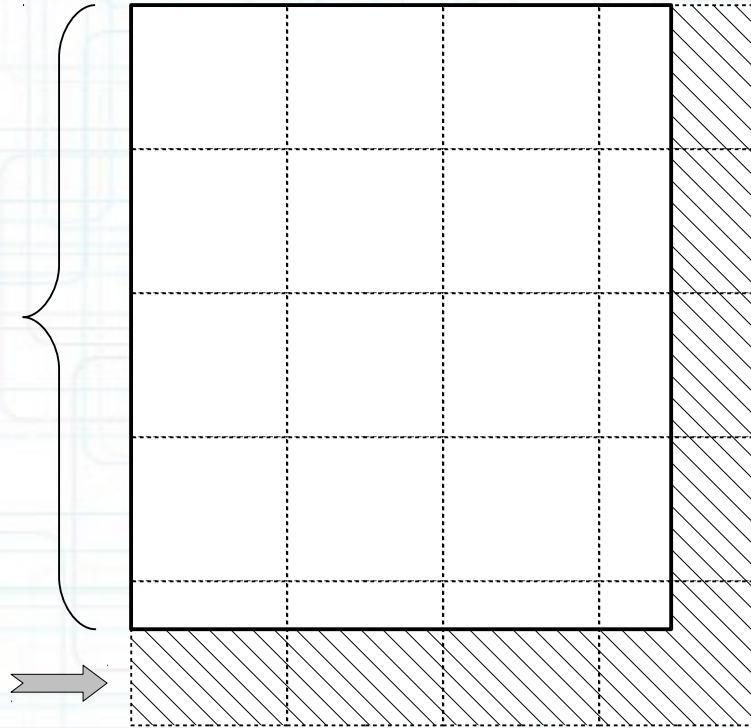
# Texture Reads

## Cleanup using clamping

*maps outside  
of the matrix*



*matrix = texture*



*maps to an unintentional  
location within the matrix*

- border blocks processed like full blocks
- A & B – texture clamping to avoid access violations
- C – conditional store

# Multiple Precisions < cublas.h >

```
#ifndef COMPLEX
#define DOUBLE

#define conj(A)          cuConj(A)
#define add(A, B)       cuCadd(A, B)
#define mul(A, B)       cuCmul(A, B)
#define fma(A, B, C) C = cuCfma(A, B, C)
#define make_FloatingPoint(x, y) make_cuDoubleComplex(x, y);

#else

#define conj(A)          cuConjf(A)
#define add(A, B)       cuCaddf(A, B)
#define mul(A, B)       cuCmulf(A, B)
#define fma(A, B, C) C = cuCfmaf(A, B, C)
#define make_FloatingPoint(x, y) make_cuFloatComplex(x, y);

#endif
#else

#define conj(A)          (A)
#define add(A, B)       (A+B)
#define mul(A, B)       (A*B)
#define fma(A, B, C) C += (A*B)
#define make_FloatingPoint(x, y) (x)

#endif
```

# CUDA GEMM

## Universal Stencil

- All types of input (BLAS standard)
  - single / double
  - real / complex
  - A: NoTrans / Trans / ConjTrans
  - B: NoTrans / Trans / ConjTrans
- Three types of memory reads
  - no textures / 1D textures / 2D textures
- $2 \times (4 + 9) \times 3 = 78$  cases total

# CUDA GEMM

## Universal Stencil

- 9 tuning parameters
  - $M, N, K$  – tiling of  $A, B, C$
  - $X_A, Y_A$  – shape of thread grid for reading  $A$
  - $X_B, Y_B$  – shape of thread grid for reading  $B$
  - $X_C, Y_C$  – shape of thread grid for computing  $C$
- Basically  $256^9$  search space, ca.  $10^{21}$  (sextillion)

# Hardware Specs

## Queryable (no need for probing)

Technical Specifications	Compute Capability				
	1.0	1.1	1.2	1.3	2.x
Maximum x- or y-dimension of a grid of thread blocks	65535				
Maximum number of threads per block	512				1024
Maximum x- or y-dimension of a block	512				1024
Maximum z-dimension of a block	64				
Warp size	32				
Maximum number of resident blocks per multiprocessor	8				
Maximum number of resident warps per multiprocessor	24	32		48	
Maximum number of resident threads per multiprocessor	768	1024		1536	
Number of 32-bit registers per multiprocessor	8 K	16 K		32 K	
Maximum amount of shared memory	16 KB				48 KB

# Pruning

## Hardware Constraints

- Rigid (exceeding hardware specs)
  - more threads than **MAX\_THREADS\_PER\_BLOCK**
  - more registers than **MAX\_REGISTERS\_PER\_BLOCK**
  - more shared memory than **MAX\_SHMEM\_PER\_BLOCK**
- Flexible (causing obvious inefficiencies)
  - threads per block not divisible by **WARP\_SIZE**

# Pruning

## Software Constraints

- Rigid (violating implementation correctness)
  - shape of thread block mismatches tiling
- Flexible (causing obvious inefficiencies)
  - **too low occupancy**
  - too big register pressure
  - too big memory pressure

# Occupancy Pruning

## Limited by register usage

$\text{regs\_per\_thread} =$  *estimated usage in the innermost (completely unrolled) loop*

$\text{regs\_per\_block} = \text{regs\_per\_thread} * \text{threads\_per\_block}$

$\text{blocks\_per\_sm} = \text{TOTAL\_REGS\_PER\_SM} / \text{regs\_per\_block}$

*blocks\_per\_sm > MAX\_BLOCKS\_PER\_SM?*

*warps\_per\_sm > MAX\_WARPS\_PER\_SM?*

$\text{threads\_per\_sm} = \text{blocks\_per\_sm} * \text{threads\_per\_block}$

# Occupancy Pruning

## Limited by shared memory usage

$\text{shmem\_per\_block} =$  *as declared in the kernel body*

$\text{blocks\_per\_sm} = \text{TOTAL\_SHMEM\_PER\_SM} / \text{shmem\_per\_block}$

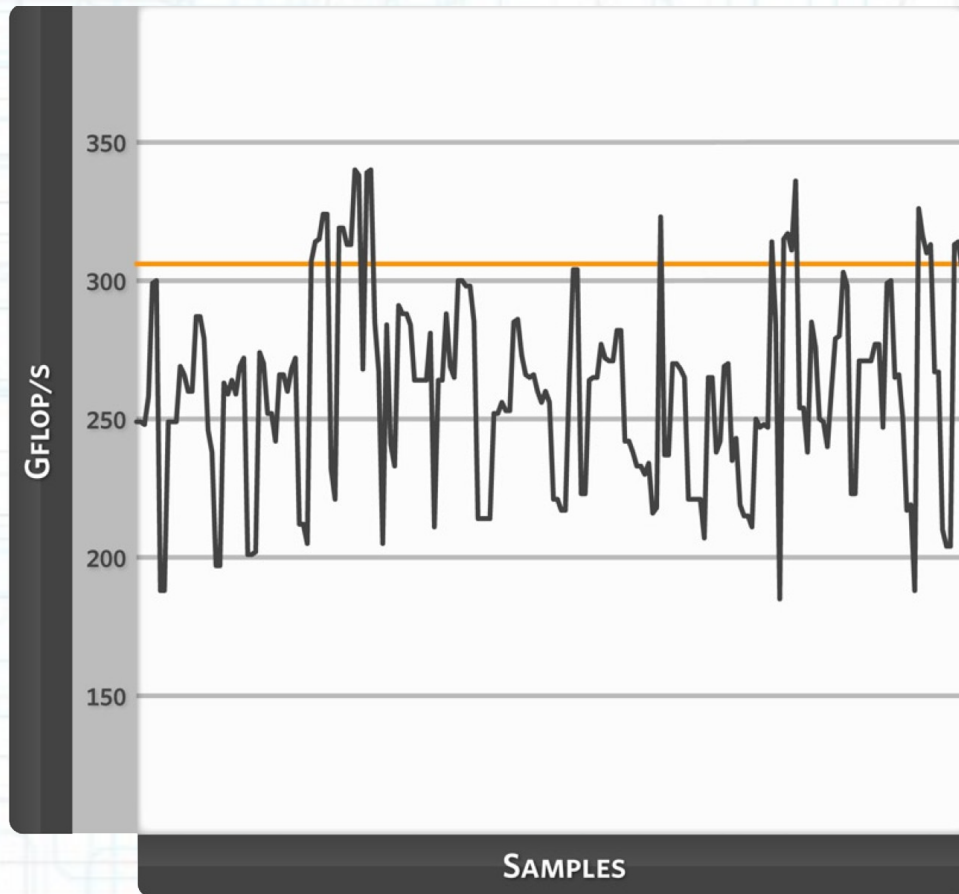
*blocks\_per\_sm > MAX\_BLOCKS\_PER\_SM?*

*warps\_per\_sm > MAX\_WARPS\_PER\_SM?*

$\text{threads\_per\_sm} = \text{blocks\_per\_sm} * \text{threads\_per\_block}$

# ZGEMM

## Autotuning Run



300 GFLOPS → 340 GFLOPS

- no constraints



**1021**

- hard constraints



**99,519**

- soft constraints
  - occupancy  $\geq \frac{1}{3}$  (512)



**492**

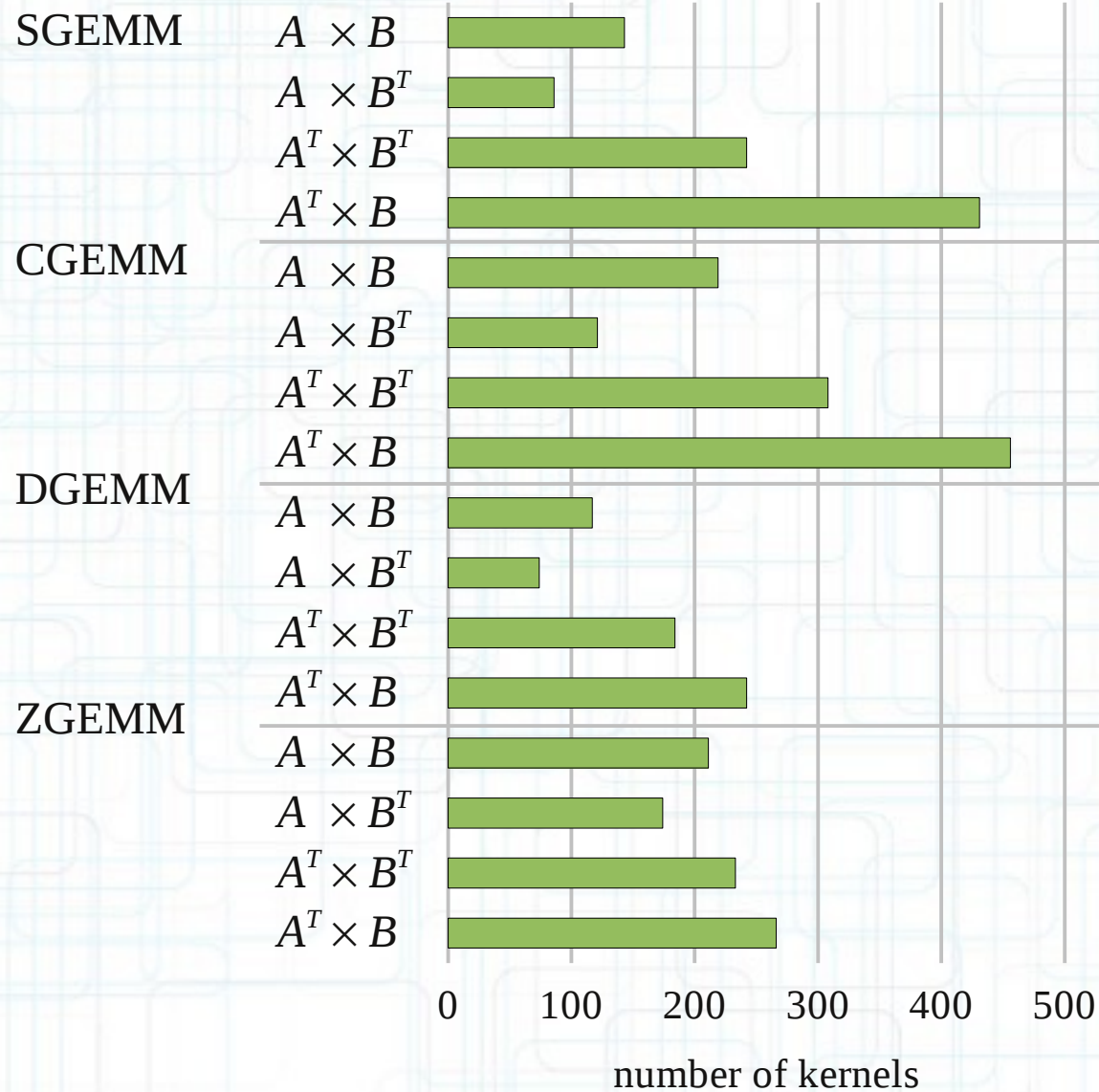
- soft constraints
  - max 2 loads / FMA
  - min 2 blocks / SM



**211**

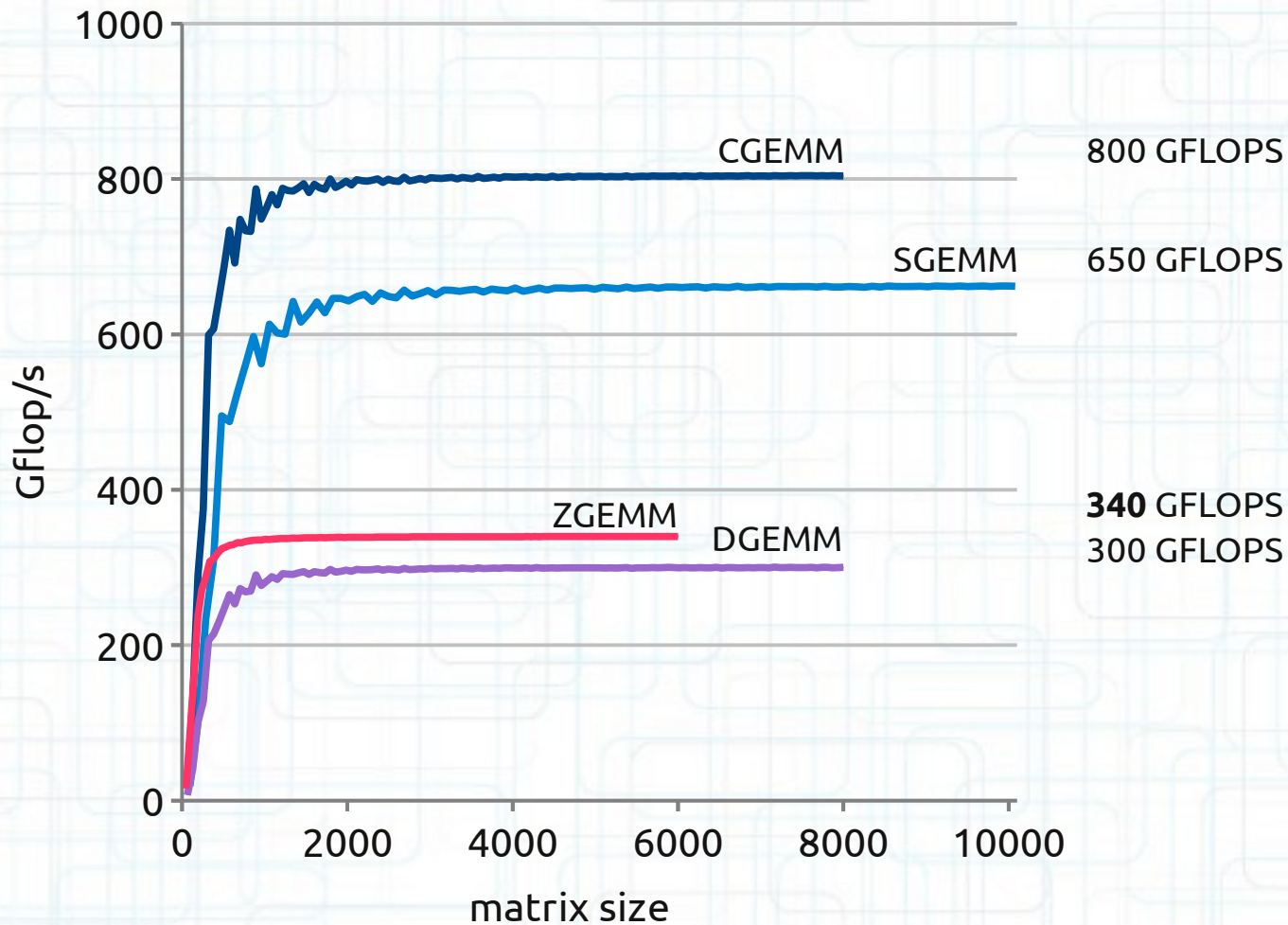
# GEMMs

## All Precisions



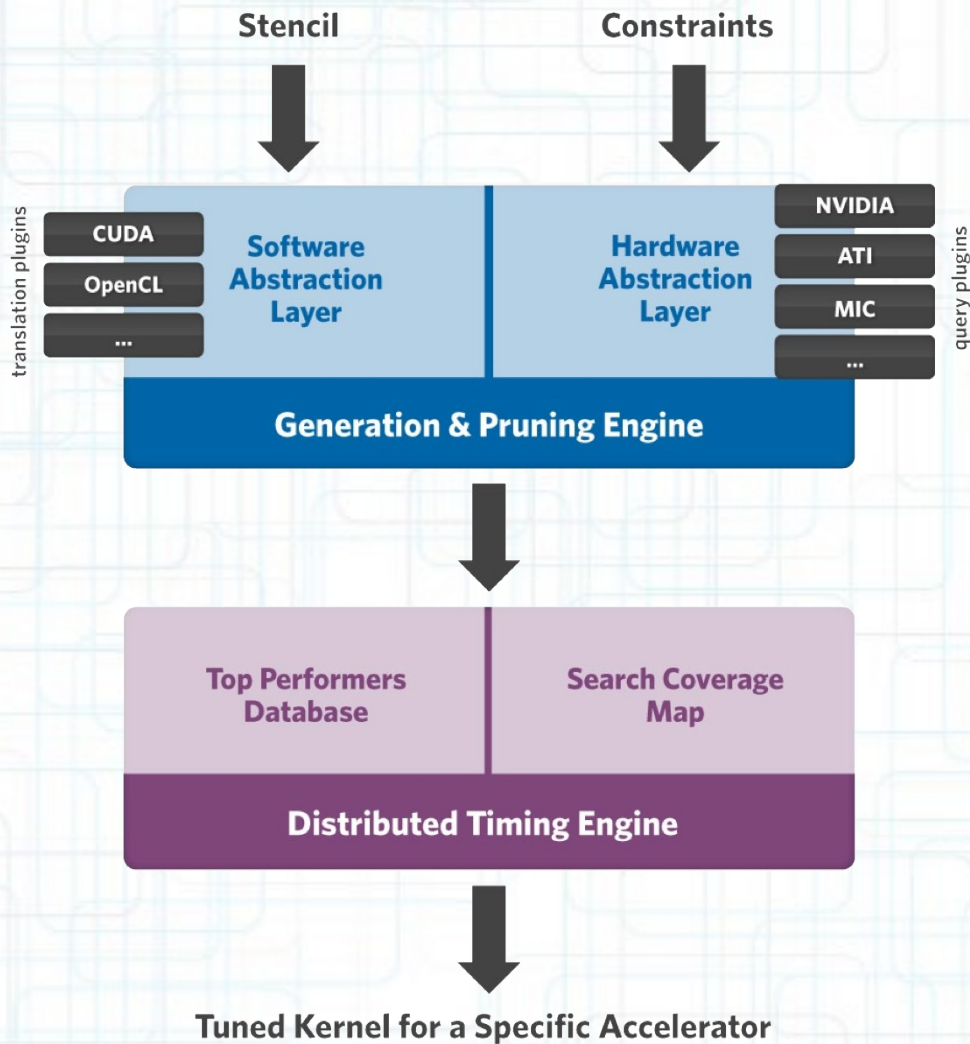
# GEMMs

## All Precisions



# ASTRA

## Automatic Stencil TuneR for Accelerators



# ASTRA

## Making the case for massively parallel tuning

*The Chinese supercomputer (Tianhe-1A), which made it to the top of the TOP500 list in November 2010, has **7168** Fermi GPUs.*

*By using **365** GPUs for **24** hours, an autotuning run can be done that would take a year on a single GPU; 365 GPUs are exactly **5%** of the machine.*

*In order to test the speed of the Chinese machine (and determine its TOP500 ranking)*

*an HPL run was made that took **202 minutes**.*

*If that time was used for an autotuning run, one could tune in parallel the equivalent of **1000 days** of tuning on a single GPU.*

# Q&A

and a vigorous discussion...

