

SC'19 Tutorial

Tools and Techniques for Programming GPUs

Portable Performance Using Autotuning and the BONSAI Project

Jakub Kurzak

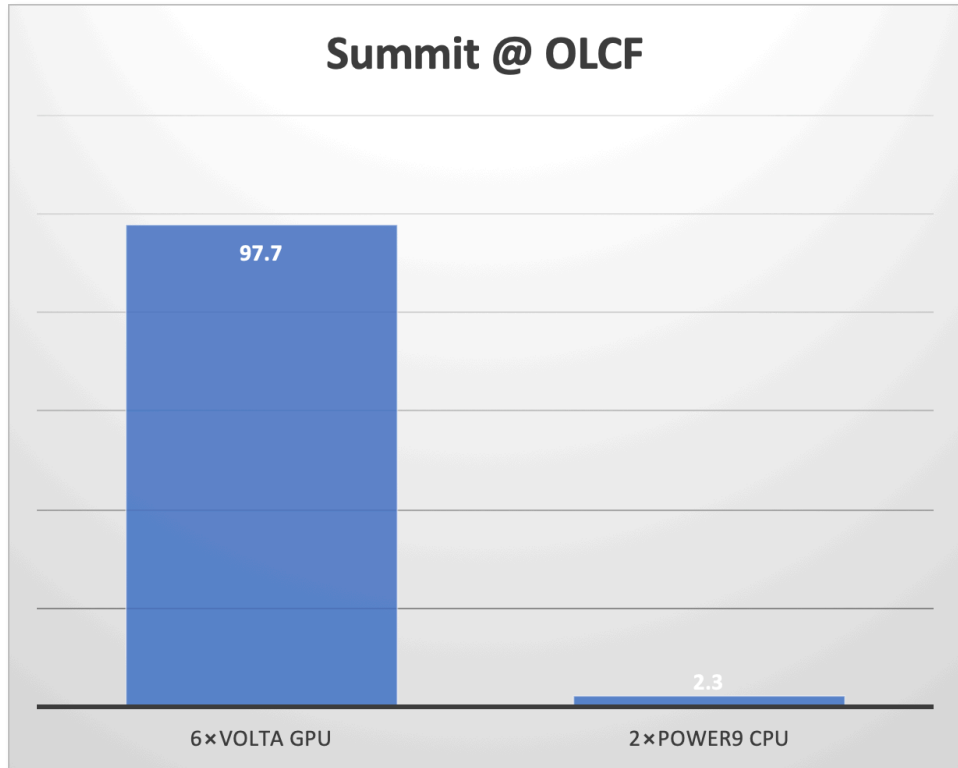
Innovative Computing Laboratory

University of Tennessee



motivation

overwhelming power of GPUs



- **97.7%** of performance in GPUs
- **2.3%** of performance in CPUs



autotuning

- implement a tunable kernel
 - compile-time parameters
 - runtime parameters
- generate the search space
- prune the search space
 - hardware constraints
 - software constraints
 - performance heuristics
- run a sweep and pick the best kernel
 - the fastest
 - the most energy efficient?

hardware

GTX 1060
6GB



- 1280 CUDA cores
- 1503 MHz clock
- **3855** GFLOPS peak (single precision)
- CUDA 10.1
- GCC 8.3

matrix multiplication

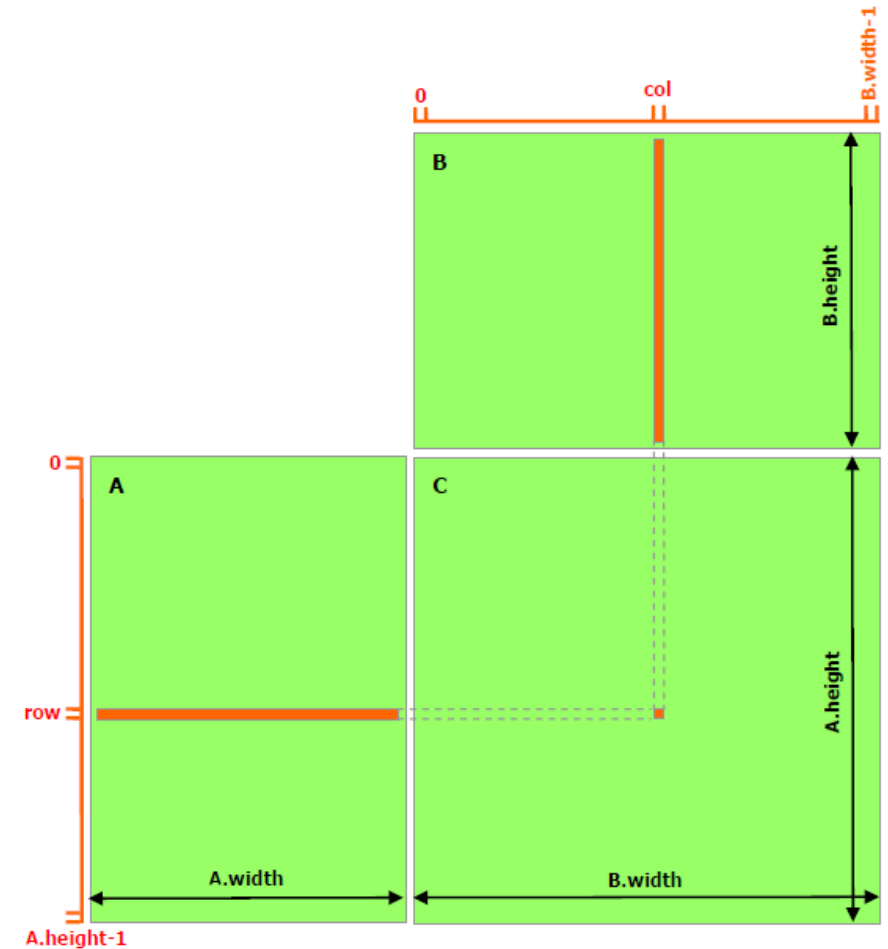
CUDA Programming Guide

```
for (int j = 0; j < n; ++j)
    for (int i = 0; i < m; ++i)
        for (int l = 0; l < k; ++l)
            C(i, j) = alpha * A(i, l) * B(l, j) + beta * C(i, j)
```

For each element of C,
compute the inner product of a row of A and a column of B.

CUDA C Programming Guide, v10.1, August 2019

Figure 9. Matrix Multiplication without Shared Memory



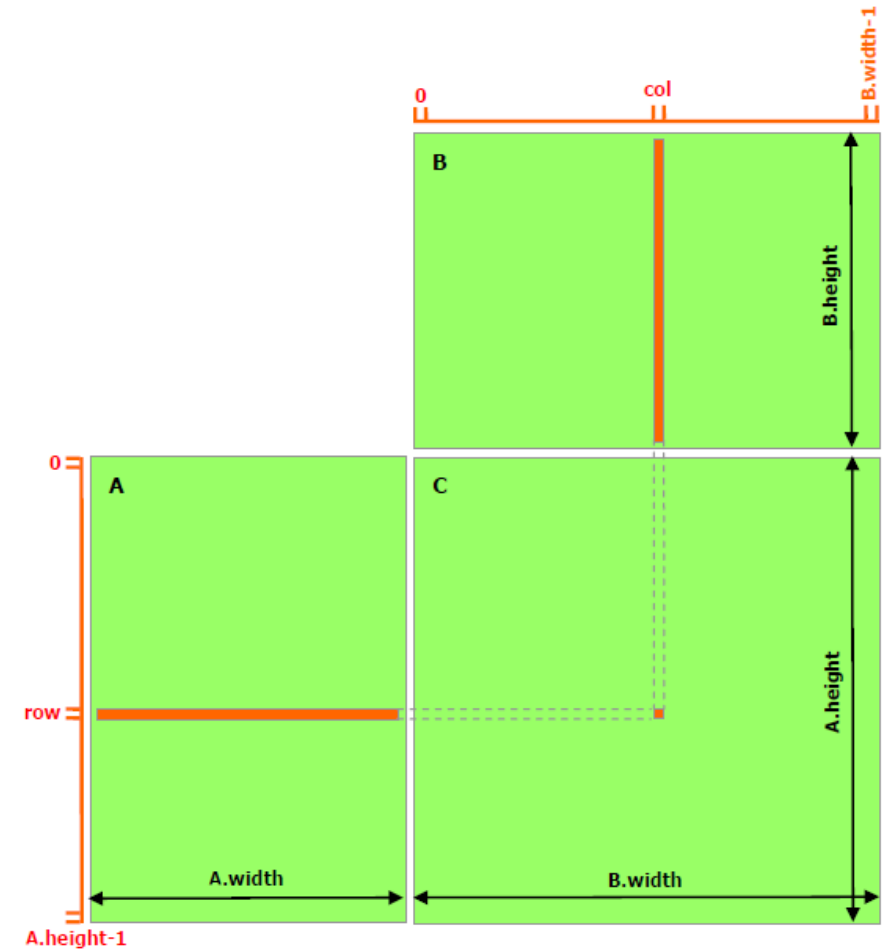
matrix multiplication

CUDA Programming Guide

```
typedef struct {  
    int width;  
    int height;  
    float* elements;  
} Matrix;
```

```
__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)  
{  
    float Cvalue = 0;  
    int row = blockIdx.y * blockDim.y + threadIdx.y;  
    int col = blockIdx.x * blockDim.x + threadIdx.x;  
    for (int e = 0; e < A.width; ++e)  
        Cvalue += A.elements[row * A.width + e]  
                 * B.elements[e * B.width + col];  
    C.elements[row * C.width + col] = Cvalue;  
}
```

```
#define BLOCK_SIZE 16  
void MatMul(const Matrix A, const Matrix B, Matrix C)  
{  
    // ...  
    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);  
    dim3 dimGrid(B.width / dimBlock.x, A.height / dimBlock.y);  
    MatMulKernel<<<dimGrid, dimBlock>>>>(d_A, d_B, d_C);  
    // ...  
}
```



matrix multiplication

CUDA Programming Guide

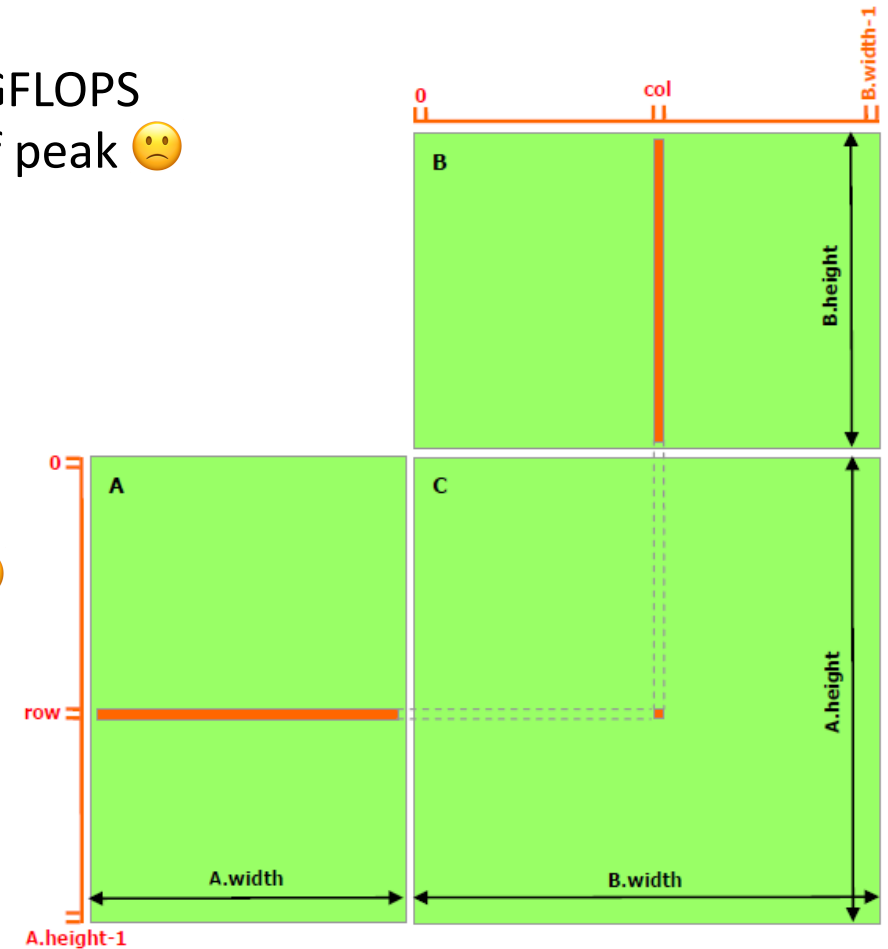
```
typedef struct {  
    int width;  
    int height;  
    float* elements;  
} Matrix;
```

```
__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)  
{  
    float Cvalue = 0;  
    int row = blockIdx.y * blockDim.y + threadIdx.y;  
    int col = blockIdx.x * blockDim.x + threadIdx.x;  
    for (int e = 0; e < A.width; ++e)  
        Cvalue += A.elements[row * A.width + e]  
                 * B.elements[e * B.width + col];  
    C.elements[row * C.width + col] = Cvalue;  
}
```

```
#define BLOCK_SIZE 16  
void MatMul(const Matrix A, const Matrix B, Matrix C)  
{  
    // ...  
    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);  
    dim3 dimGrid(B.width / dimBlock.x, A.height / dimBlock.y);  
    MatMulKernel<<<dimGrid, dimBlock>>>>(d_A, d_B, d_C);  
    // ...  
}
```

- **123 GFLOPS**
- 3% of peak 😞

tunable? 🤔



matrix multiplication

CUDA Programming Guide

```
// #define BLOCK_SIZE 16
void MatMul(const Matrix A, const Matrix B, Matrix C)
{
    // ...
    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
    dim3 dimGrid(B.width / dimBlock.x, A.height / dimBlock.y);
    MatMulKernel<<<dimGrid, dimBlock>>>>(d_A, d_B, d_C);
    // ...
}
```

```
> nvcc -o matmul matmul.cu -DBLOCK_SIZE=8
> ./matmul 8000

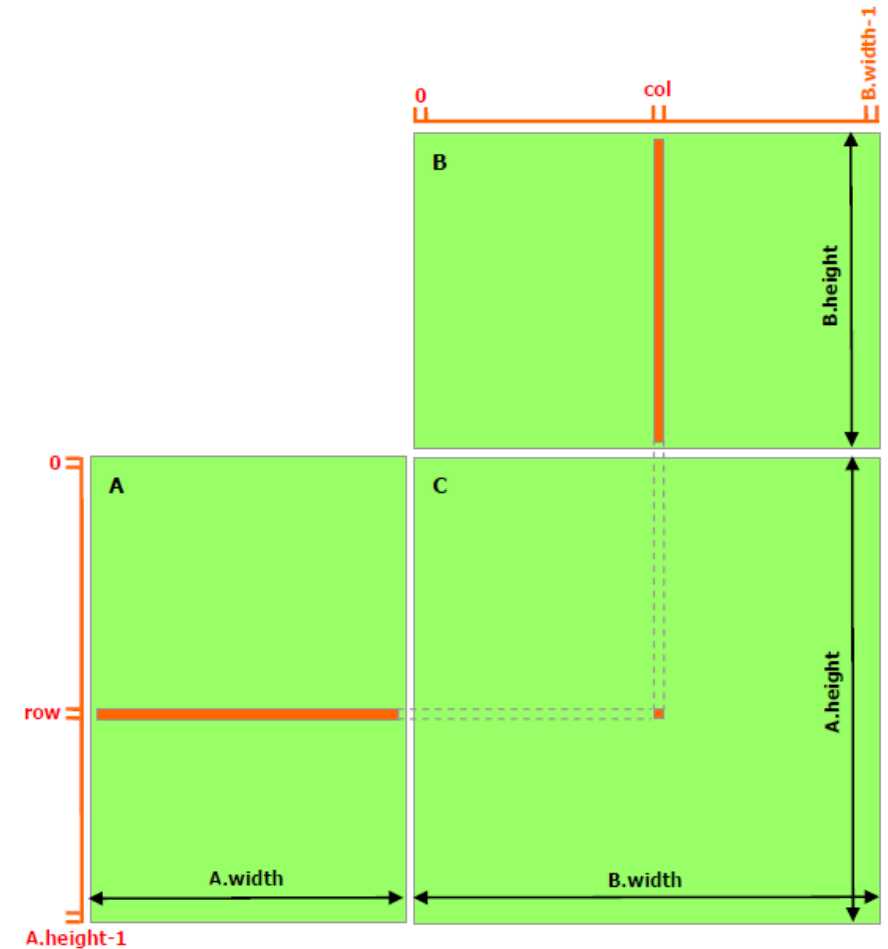
> nvcc -o matmul matmul.cu -DBLOCK_SIZE=16
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=24
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=32
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=40
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=48
> ./matmul 8000
```



matrix multiplication

CUDA Programming Guide

```
> nvcc -o matmul matmul.cu -DBLOCK_SIZE=8
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=16
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=24
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=32
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=40
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=48
> ./matmul 8000
```

```
dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
```

100 GFLOPS

124 GFLOPS

120 GFLOPS

170 GFLOPS

- 37% faster 😊
- 4.4% of peak 😞

invalid configuration argument – runtime error 🤔

invalid configuration argument – runtime error 🤔

$\text{BLOCK_SIZE} \times \text{BLOCK_SIZE} > 1024$

```
std::cout << cudaGetErrorString(cudaPeekAtLastError()) << std::endl;
```

CUDA Programming Guide

Figure 10. Matrix Multiplication with Shared Memory



matrix multiplication

CUDA Programming Guide

```
#define BLOCK_SIZE 16
__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)
{
    int blockRow = blockIdx.y;
    int blockCol = blockIdx.x;

    float Cvalue = 0;

    int row = threadIdx.y;
    int col = threadIdx.x;

    for (int m = 0; m < (A.width / BLOCK_SIZE); ++m) {
        Matrix Asub = GetSubMatrix(A, blockRow, m);
        Matrix Bsub = GetSubMatrix(B, m, blockCol);

        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

        As[row][col] = GetElement(Asub, row, col);
        Bs[row][col] = GetElement(Bsub, row, col);

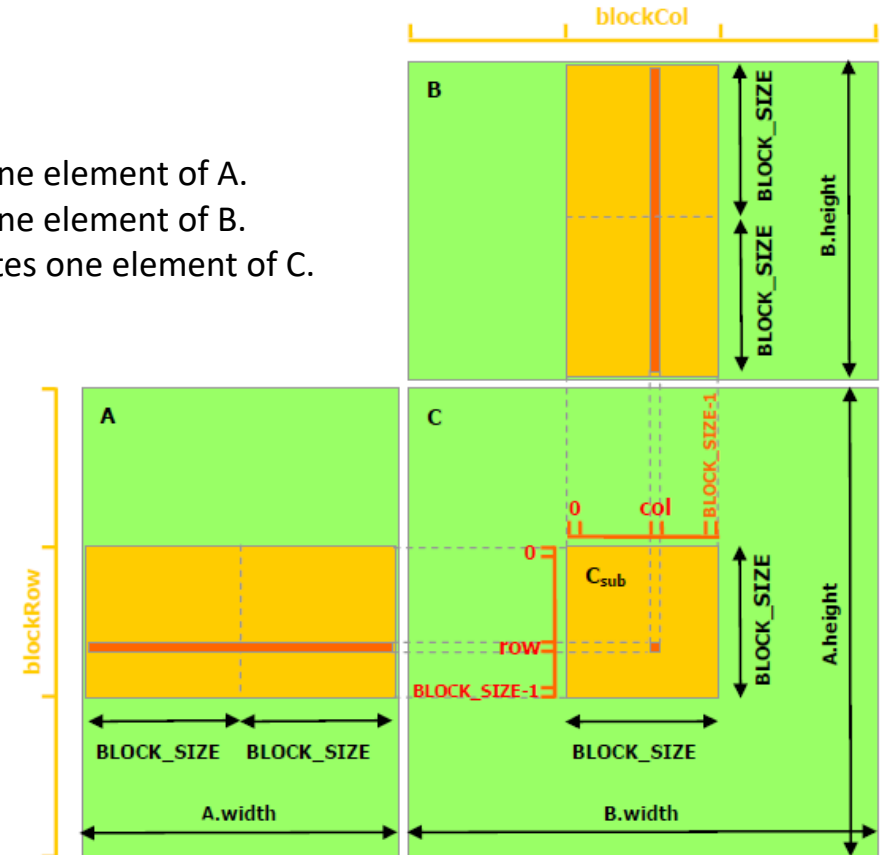
        __syncthreads();

        for (int e = 0; e < BLOCK_SIZE; ++e)
            Cvalue += As[row][e] * Bs[e][col];

        __syncthreads();
    }

    Matrix Csub = GetSubMatrix(C, blockRow, blockCol);
    SetElement(Csub, row, col, Cvalue);
}
```

- Each thread loads one element of A.
- Each thread loads one element of B.
- Each thread computes one element of C.



matrix multiplication

CUDA Programming Guide

```
#define BLOCK_SIZE 16
__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)
{
    int blockRow = blockIdx.y;
    int blockCol = blockIdx.x;

    float Cvalue = 0;

    int row = threadIdx.y;
    int col = threadIdx.x;

    for (int m = 0; m < (A.width / BLOCK_SIZE); ++m) {
        Matrix Asub = GetSubMatrix(A, blockRow, m);
        Matrix Bsub = GetSubMatrix(B, m, blockCol);

        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

        As[row][col] = GetElement(Asub, row, col);
        Bs[row][col] = GetElement(Bsub, row, col);

        __syncthreads();

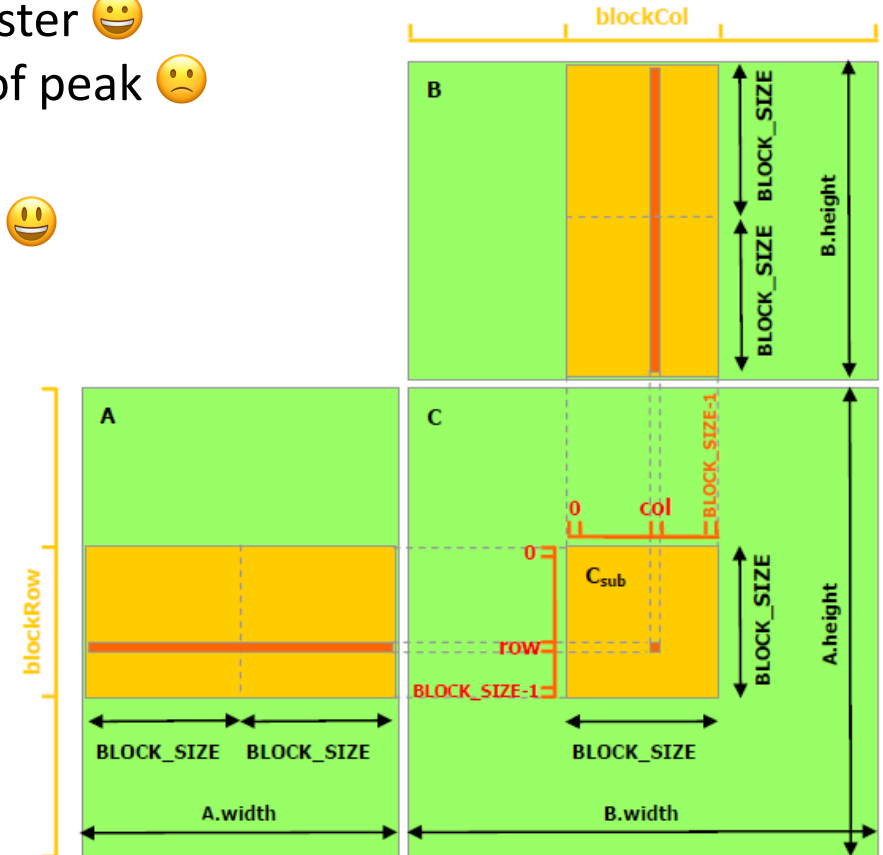
        for (int e = 0; e < BLOCK_SIZE; ++e)
            Cvalue += As[row][e] * Bs[e][col];

        __syncthreads();
    }

    Matrix Csub = GetSubMatrix(C, blockRow, blockCol);
    SetElement(Csub, row, col, Cvalue);
}
```

- 500 GFLOPS
- 4× faster 😊
- 13% of peak 😞

tunable! 😊



matrix multiplication

CUDA Programming Guide

```
// #define BLOCK_SIZE 16
__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)
{
    int blockRow = blockIdx.y;
    int blockCol = blockIdx.x;

    float Cvalue = 0;

    int row = threadIdx.y;
    int col = threadIdx.x;
```

```
> nvcc -o matmul matmul.cu -DBLOCK_SIZE=16
> ./matmul 8000

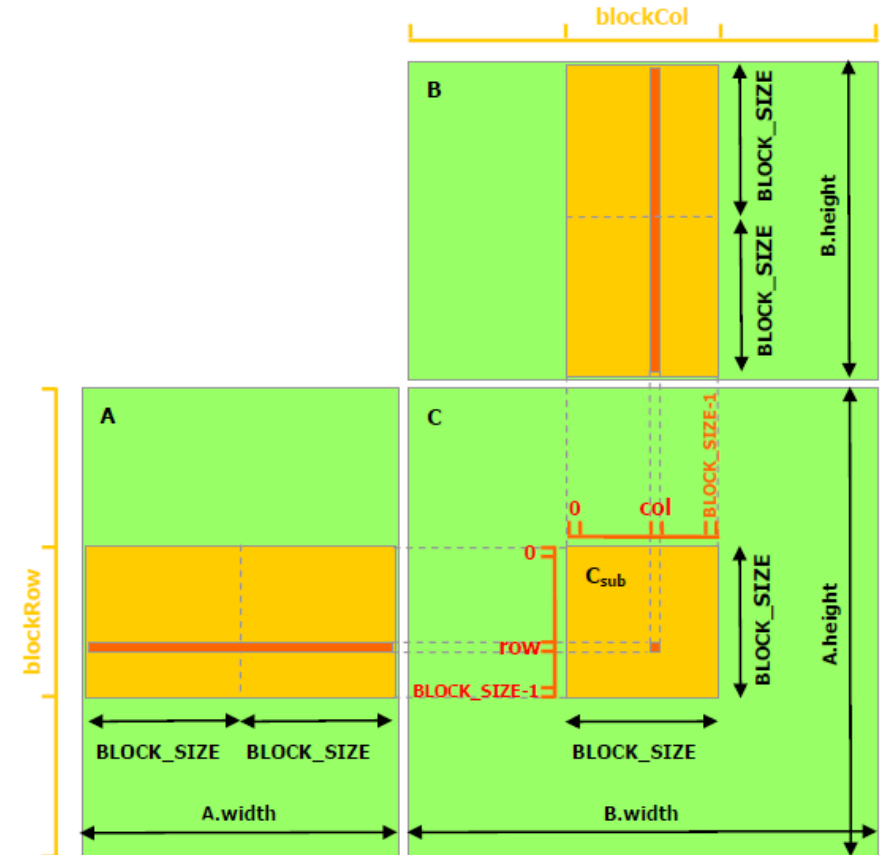
> nvcc -o matmul matmul.cu -DBLOCK_SIZE=32
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=48
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=64
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=80
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=96
> ./matmul 8000
```



matrix multiplication

CUDA Programming Guide

```
> nvcc -o matmul matmul.cu -DBLOCK_SIZE=16
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=32
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=48
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=64
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=80
> ./matmul 8000

> nvcc -o matmul matmul.cu -DBLOCK_SIZE=96
> ./matmul 8000
```

```
__shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
__shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
```

500 GFLOPS

640 GFLOPS

- 28% faster 😐
- 16% of peak 😞

invalid configuration argument – runtime error 🤖

invalid configuration argument – runtime error 🤖

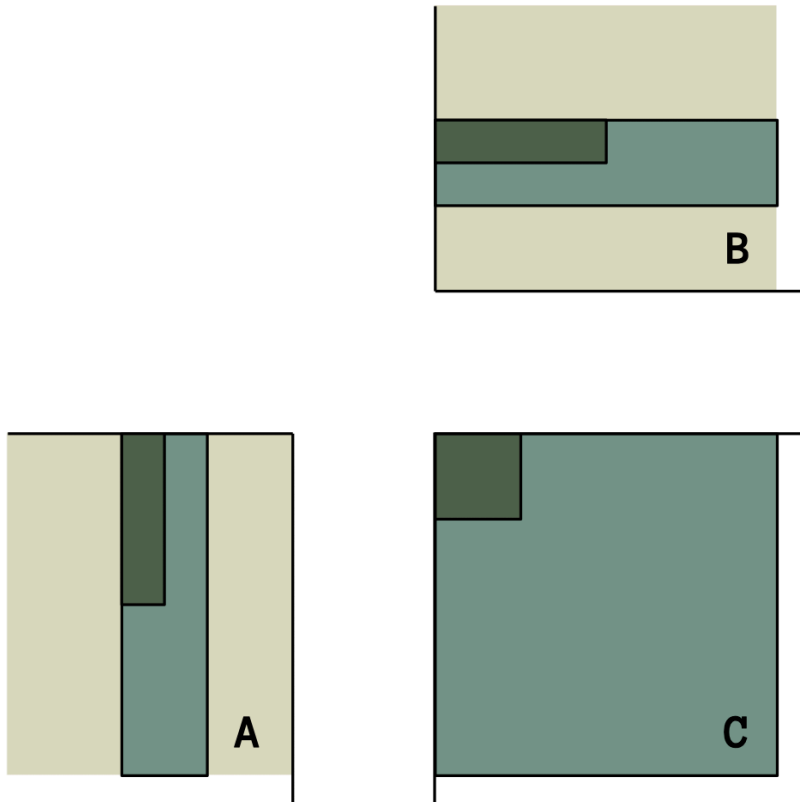
uses too much shared data – compilation error 🤖

uses too much shared data – compilation error 🤖

$2 \times \text{BLOCK_SIZE} \times \text{BLOCK_SIZE} \times \text{sizeof}(\text{float}) > 49152$

matrix multiplication

BONSAI sgemm example



similarly

- reading A and B to shared memory
- computing C in registers

but

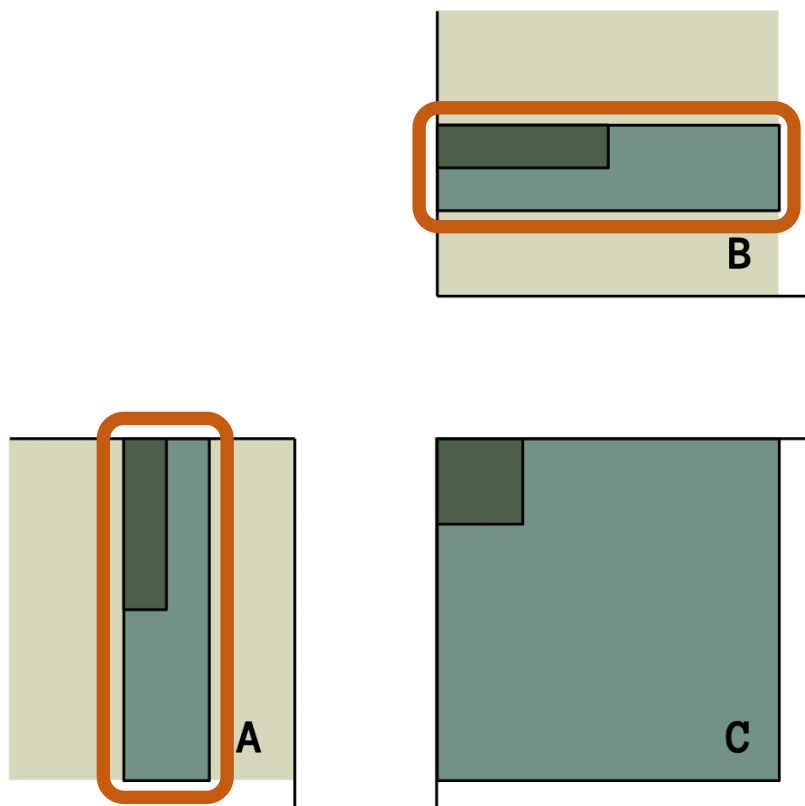
- Each thread can compute many elements of C.
- Blocks of A, B, and C can be rectangular.
- Thread block can be rectangular.
- Threads can be arranged differently for:
 - reading A,
 - reading B,
 - computing C.

one screen of code (~70 lines)

<https://bitbucket.org/icl/bonsai/src/default/examples/sgemm/kernel.cu>

matrix multiplication

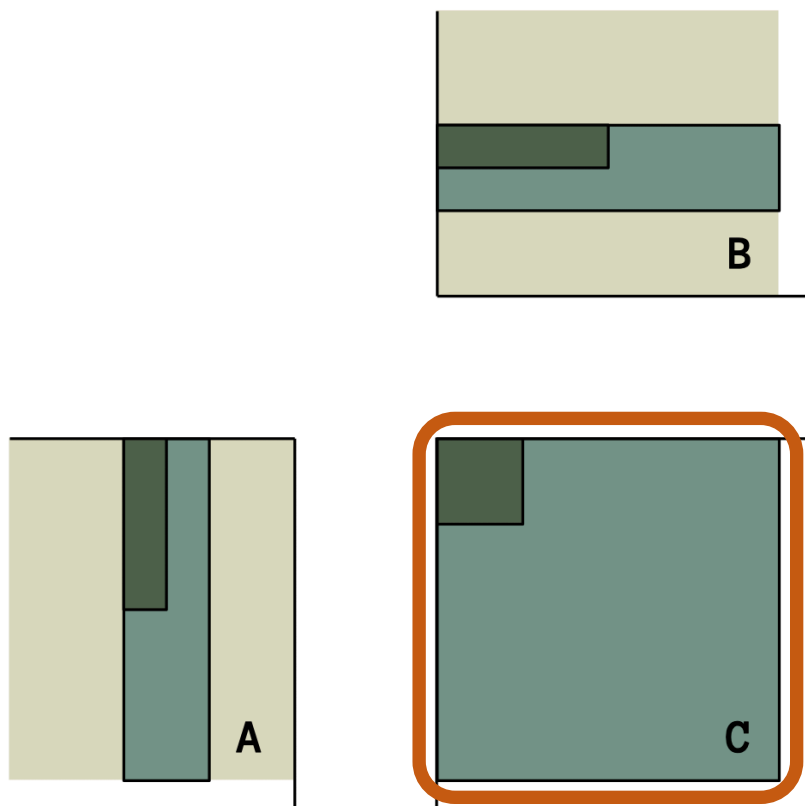
BONSAI sgemm example



Blocks A and B go through shared memory.

matrix multiplication

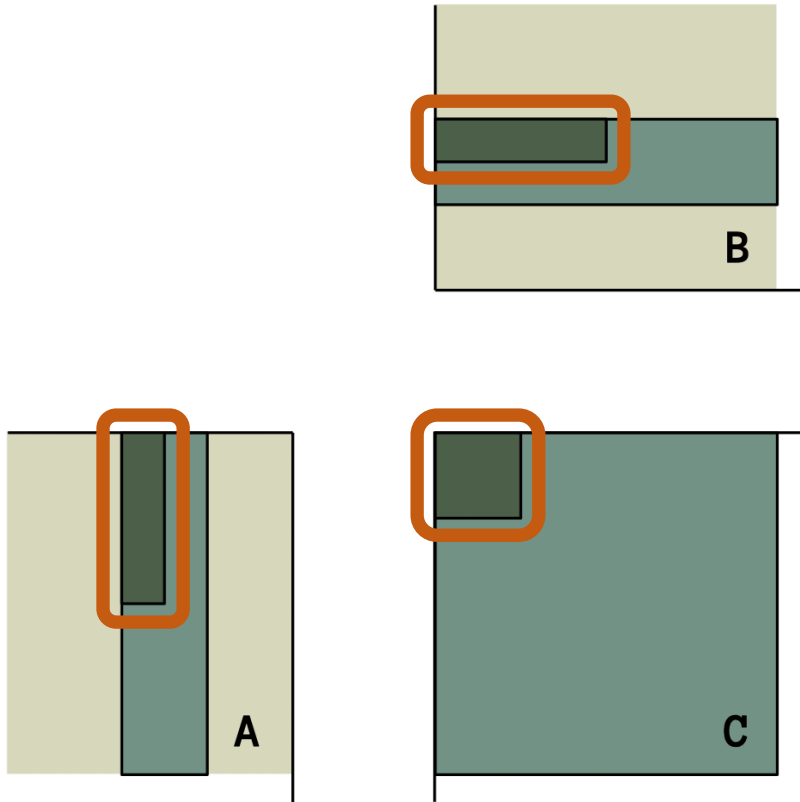
BONSAI sgemm example



Block of C is computed in registers.

matrix multiplication

BONSAI sgemm example



Threads can be arranged differently for:

- reading A,
- reading B,
- computing C.

one screen of code (~70 lines)

<https://bitbucket.org/icl/bonsai/src/default/examples/sgemm/kernel.cu>

```
for (int kk = 0; kk < K; kk += BLK_K)
{
    #pragma unroll
    for (int n = 0; n < BLK_K; n += DIM_N_A)
        #pragma unroll
        for (int m = 0; m < BLK_M; m += DIM_M_A)
            sA[n+idyA][m+idxA] = __ldg(&A[coord_A + (n*lda+m)]);

    #pragma unroll
    for (int n = 0; n < BLK_N; n += DIM_N_B)
        #pragma unroll
        for (int m = 0; m < BLK_K; m += DIM_M_B)
            sB[n+idyB][m+idxB] = __ldg(&B[coord_B + (n*ldb+m)]);

    __syncthreads();

    #pragma unroll
    for (int k = 0; k < BLK_K; ++k)
        #pragma unroll
        for (int n = 0; n < BLK_N/DIM_N; ++n)
            #pragma unroll
            for (int m = 0; m < BLK_M/DIM_M; ++m)
                rC[n][m] += sA[k][m*DIM_M+idx] * sB[n*DIM_N+idy][k];

    __syncthreads();

    coord_A += BLK_K*lda;
    coord_B += BLK_K;
}
```



```
__global__
void SgemmKernel(int M, int N, int K,
                 float alpha, float *A, int lda,
                 float *B, int ldb,
                 float beta, float *C, int ldc)
{
    int idx = threadIdx.x; // thread's m position in C
    int idy = threadIdx.y; // thread's n position in C
    int idt = DIM_M*idy + idx; // thread's number
    int blx = blockIdx.x; // block's m position
    int bly = blockIdx.y; // block's n position
    int idxA = idt % DIM_M_A; // thread's m position for loading A
    int idyA = idt / DIM_M_A; // thread's n position for loading A
    int idxB = idt % DIM_M_B; // thread's m position for loading B
    int idyB = idt / DIM_M_B; // thread's n position for loading B

    __shared__ float sA[BLK_K][BLK_M]; // shared memory for A
    __shared__ float sB[BLK_N][BLK_K]; // shared memory for B
    float rC[BLK_N/DIM_N][BLK_M/DIM_M]; // registers for C

    int coord_A = (blx*BLK_M + idyA*lda) + idxA;
    int coord_B = (bly*BLK_N*ldb + idyB*ldb) + idxB;

    #pragma unroll
    for (int n = 0; n < BLK_N/DIM_N; ++n)
        #pragma unroll
        for (int m = 0; m < BLK_M/DIM_M; ++m)
            rC[n][m] = 0.0;

    #pragma unroll
    for (int n = 0; n < BLK_N/DIM_N; ++n) {
        int coord_dCn = bly*BLK_N + n*DIM_N+idy;
        #pragma unroll
        for (int m = 0; m < BLK_M/DIM_M; ++m) {
            int offsC = coord_dCn*ldc + blx*BLK_M;
            C[offsC + m*DIM_M+idx] =
                alpha*rC[n][m] + beta*C[offsC + m*DIM_M+idx];
        }
    }
}
```

one screen of code (~70 lines)

<https://bitbucket.org/icl/bonsai/src/default/examples/sgemm/kernel.cu>

```
for (int kk = 0; kk < K; kk += BLK_K)
{
    for (int n = 0; n < BLK_K; n += DIM_N_A)
        for (int m = 0; m < BLK_M; m += DIM_M_A)
            sA[n+idyA][m+idxA] = __ldg(&A[coord_A + (n*lda+m)]);

    for (int n = 0; n < BLK_N; n += DIM_N_B)
        for (int m = 0; m < BLK_K; m += DIM_M_B)
            sB[n+idyB][m+idxB] = __ldg(&B[coord_B + (n*ldb+m)]);

    __syncthreads();

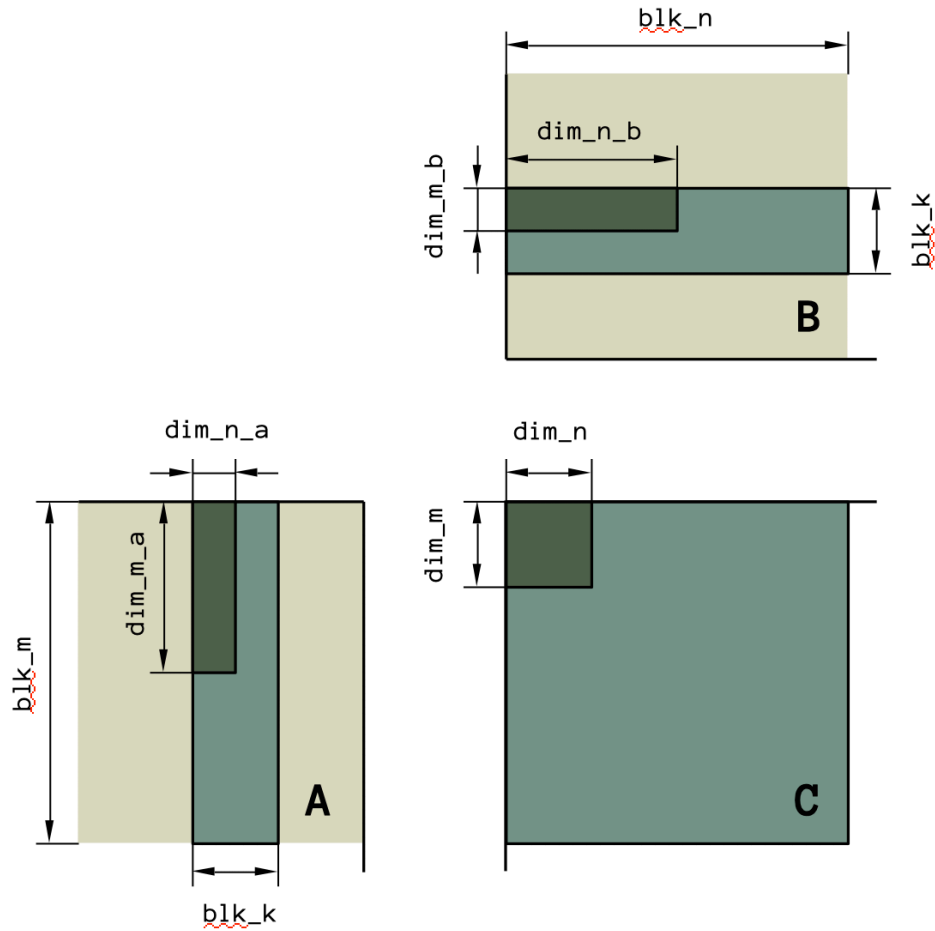
    for (int k = 0; k < BLK_K; ++k)
        for (int n = 0; n < BLK_N/DIM_N; ++n)
            for (int m = 0; m < BLK_M/DIM_M; ++m)
                rC[n][m] += sA[k][m*DIM_M+idx] * sB[n*DIM_N+idy][k];

    __syncthreads();

    coord_A += BLK_K*lda;
    coord_B += BLK_K;
}
```

matrix multiplication

search space



```
for dim_m in range(...):
    for dim_n in range(...):

        for blk_m in range(...):
            for blk_n in range(...):
                for blk_k in range(...):

                    for dim_m_a in range(...):
                        for dim_n_a in range(...):

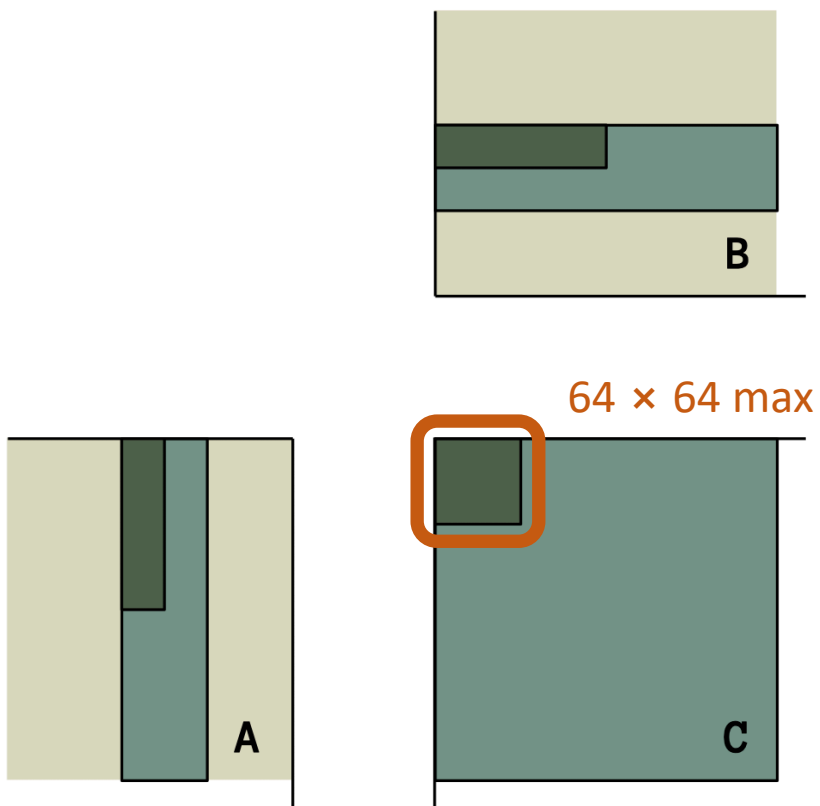
                            for dim_m_b in range(...):
                                for dim_n_b in range(...):
```

one screen of code (~70 lines)

<https://bitbucket.org/icl/bonsai/src/default/examples/sgemm/generator.py>

matrix multiplication

search space

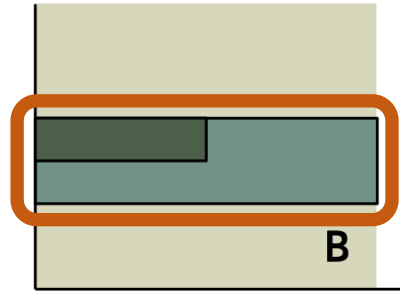


```
for dim_m in range(4, 64+1, 4):  
    for dim_n in range(4, 64+1, 4):
```

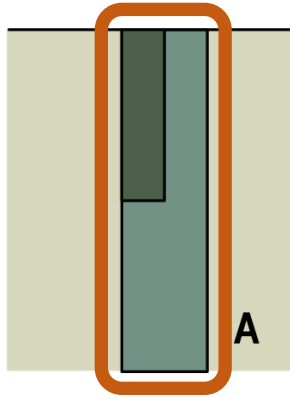
matrix multiplication

search space

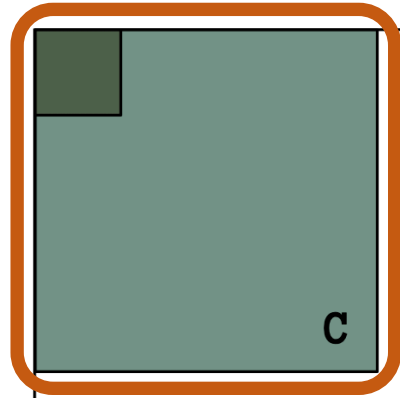
64 × 128 max



128 × 64 max



128 × 128 max

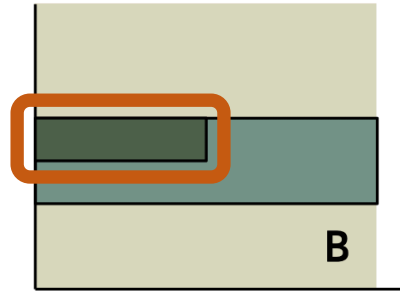


```
for dim_m in range(4, 64+1, 4):  
    for dim_n in range(4, 64+1, 4):  
  
        for blk_m in range(dim_m, 128+1, dim_m):  
            for blk_n in range(dim_n, 128+1, dim_n):  
                for blk_k in range(4, 64+1, 4):
```

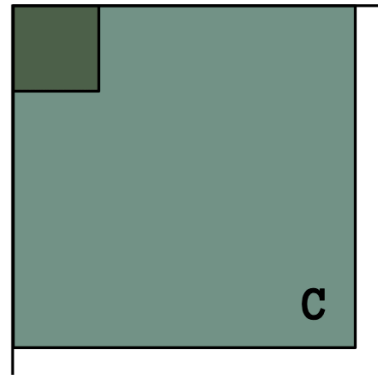
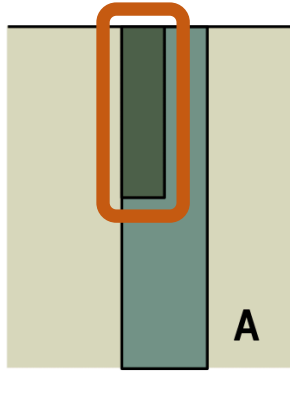

matrix multiplication

search space

64 × 128 max



128 × 64 max

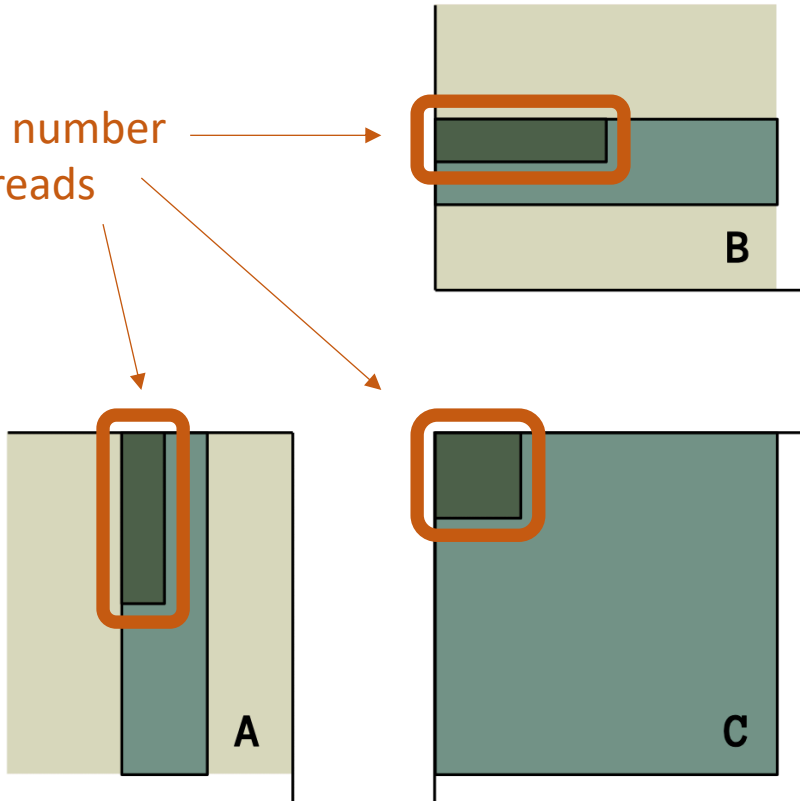


```
for dim_m in range(4, 64+1, 4):  
    for dim_n in range(4, 64+1, 4):  
  
        for blk_m in range(dim_m, 128+1, dim_m):  
            for blk_n in range(dim_n, 128+1, dim_n):  
                for blk_k in range(4, 64+1, 4):  
  
                    for dim_m_a in range(4, blk_m+1, 4):  
                        for dim_n_a in range(4, 64+1, 4):  
  
                            for dim_m_b in range(4, 64+1, 4):  
                                for dim_n_b in range(4, blk_n+1, 4):
```

matrix multiplication

pruning constraints

same number
of threads



```
for dim_m in range(4, 64+1, 4):
    for dim_n in range(4, 64+1, 4):

        for blk_m in range(dim_m, 128+1, dim_m):
            for blk_n in range(dim_n, 128+1, dim_n):
                for blk_k in range(4, 64+1, 4):

                    for dim_m_a in range(4, blk_m+1, 4):
                        for dim_n_a in range(4, 64+1, 4):

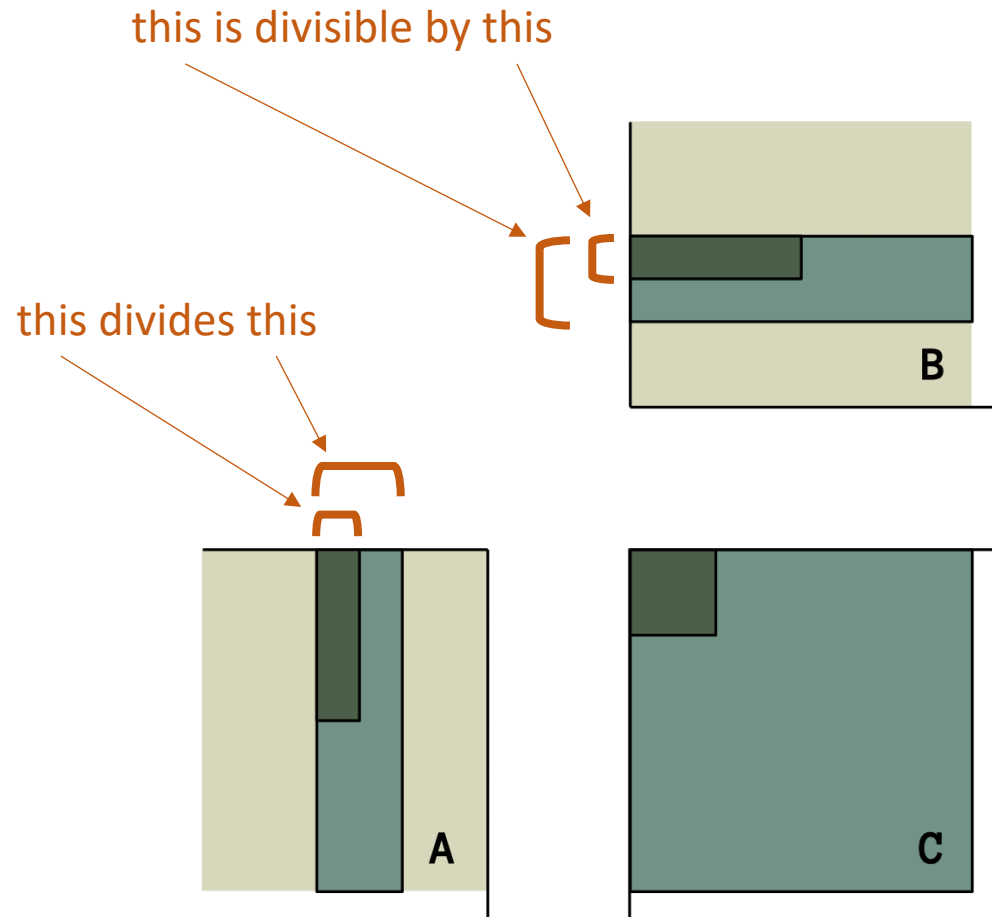
                            for dim_m_b in range(4, 64+1, 4):
                                for dim_n_b in range(4, blk_n+1, 4):
```

```
    if dim_m_a*dim_n_a != dim_m*dim_n:
        continue
```

```
    if dim_m_b*dim_n_b != dim_m*dim_n:
        continue
```

matrix multiplication

pruning constraints



```
for dim_m in range(4, 64+1, 4):
    for dim_n in range(4, 64+1, 4):

        for blk_m in range(dim_m, 128+1, dim_m):
            for blk_n in range(dim_n, 128+1, dim_n):
                for blk_k in range(4, 64+1, 4):

                    for dim_m_a in range(4, blk_m+1, 4):
                        for dim_n_a in range(4, 64+1, 4):

                            for dim_m_b in range(4, 64+1, 4):
                                for dim_n_b in range(4, blk_n+1, 4):
```

```
        if dim_m_a*dim_n_a != dim_m*dim_n:
            continue
```

```
        if dim_m_b*dim_n_b != dim_m*dim_n:
            continue
```

```
        if blk_k%dim_n_a != 0:
            continue
```

```
        if blk_k%dim_m_b != 0:
            continue
```

matrix multiplication

pruning constraints

```
for dim_m in range(4, 64+1, 4):
    for dim_n in range(4, 64+1, 4):

        for blk_m in range(dim_m, 128+1, dim_m):
            for blk_n in range(dim_n, 128+1, dim_n):
                for blk_k in range(4, 64+1, 4):

                    for dim_m_a in range(4, blk_m+1, 4):
                        for dim_n_a in range(4, 64+1, 4):

                            for dim_m_b in range(4, 64+1, 4):
                                for dim_n_b in range(4, blk_n+1, 4):
```

```
if dim_m_a*dim_n_a != dim_m*dim_n:
    continue

if dim_m_b*dim_n_b != dim_m*dim_n:
    continue
```

```
if blk_k%dim_n_a != 0:
    continue

if blk_k%dim_m_b != 0:
    continue
```

388,567 configurations

matrix multiplication

pruning constraints

388,567

```
if threads_per_block % 32 != 0:  
    continue
```

number of threads divisible by warp size

315,058

matrix multiplication

pruning constraints

388,567

```
if threads_per_block % 32 != 0:  
    continue
```

number of threads divisible by warp size

315,058

```
if threads_per_block > 1024:  
    continue
```

not exceeding max threads per block

runtime error
invalid configuration argument

310,270

matrix multiplication

pruning constraints

388,567

```
if threads_per_block % 32 != 0:  
    continue
```

number of threads divisible by warp size

315,058

```
if threads_per_block > 1024:  
    continue
```

not exceeding max threads per block

runtime error
invalid configuration argument

310,270

```
if shmem_per_block > 49152:  
    continue
```

not exceeding max shared memory

compilation error
uses too much shared data

297,690

matrix multiplication

pruning constraints

388,567

```
if threads_per_block % 32 != 0:  
    continue
```

number of threads divisible by warp size

315,058

```
if threads_per_block > 1024:  
    continue
```

not exceeding max threads per block

runtime error
invalid configuration argument

310,270

```
if shmem_per_block > 49152:  
    continue
```

not exceeding max shared memory

compilation error
uses too much shared data

297,690

```
if regs_per_thread > 255:  
    continue
```

not exceeding max registers per thread (heuristic)

291,207

matrix multiplication

pruning constraints

388,567

```
if threads_per_block % 32 != 0:  
    continue
```

number of threads divisible by warp size

315,058

```
if threads_per_block > 1024:  
    continue
```

not exceeding max threads per block

runtime error
invalid configuration argument

310,270

```
if shmem_per_block > 49152:  
    continue
```

not exceeding max shared memory

compilation error
uses too much shared data

297,690

```
if regs_per_thread > 255:  
    continue
```

not exceeding max registers per thread (heuristic)

291,207

```
if regs_per_block > 65536:  
    continue
```

not exceeding max registers per thread block (heuristic)

291,207

matrix multiplication

pruning constraints

291,207

```
if threads_per_block < 256:  
    continue
```

at least 256 threads – good occupancy (heuristic)

79,532

matrix multiplication

pruning constraints

291,207

```
if threads_per_block < 256:  
    continue
```

at least 256 threads – good occupancy (heuristic)

79,532

```
if fmas_per_load < 64:  
    continue
```

at least 64 FMAs per load from DRAM – great compute intensity
(heuristic)

528

one screen of code (~70 lines)

<https://bitbucket.org/icl/bonsai/src/default/examples/sgemm/generator.py>

matrix multiplication

running the sweep

- **528** configurations
- compile and run each one
- 8000 × 8000 matrices A, B, C
- repeat each run 5 times

matrix multiplication

running the sweep

- **528** configurations
- compile and run each one
- 8000×8000 matrices A, B, C
- repeat each run 5 times

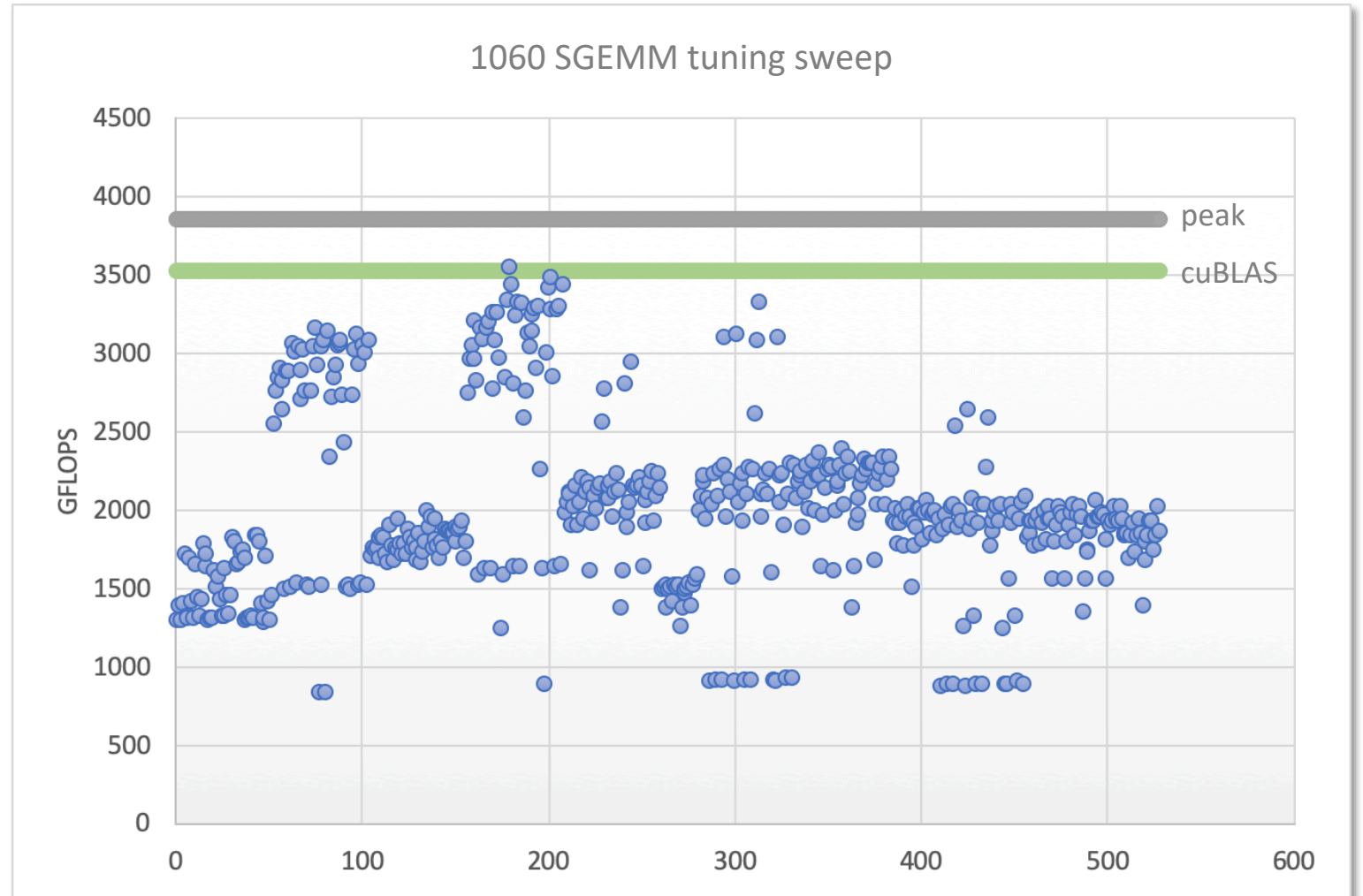
- **44** min and **16** sec



matrix multiplication

sweep results

- 528 configurations
- compile run each one
- 8000 × 8000 matrices A, B, C
- repeat each run 5 times
- **3560** GFLOPS max
- 3530 GFLOPS for cuBLAS
- 3855 GFLOPS peak
- matching cuBLAS
- 92% of peak

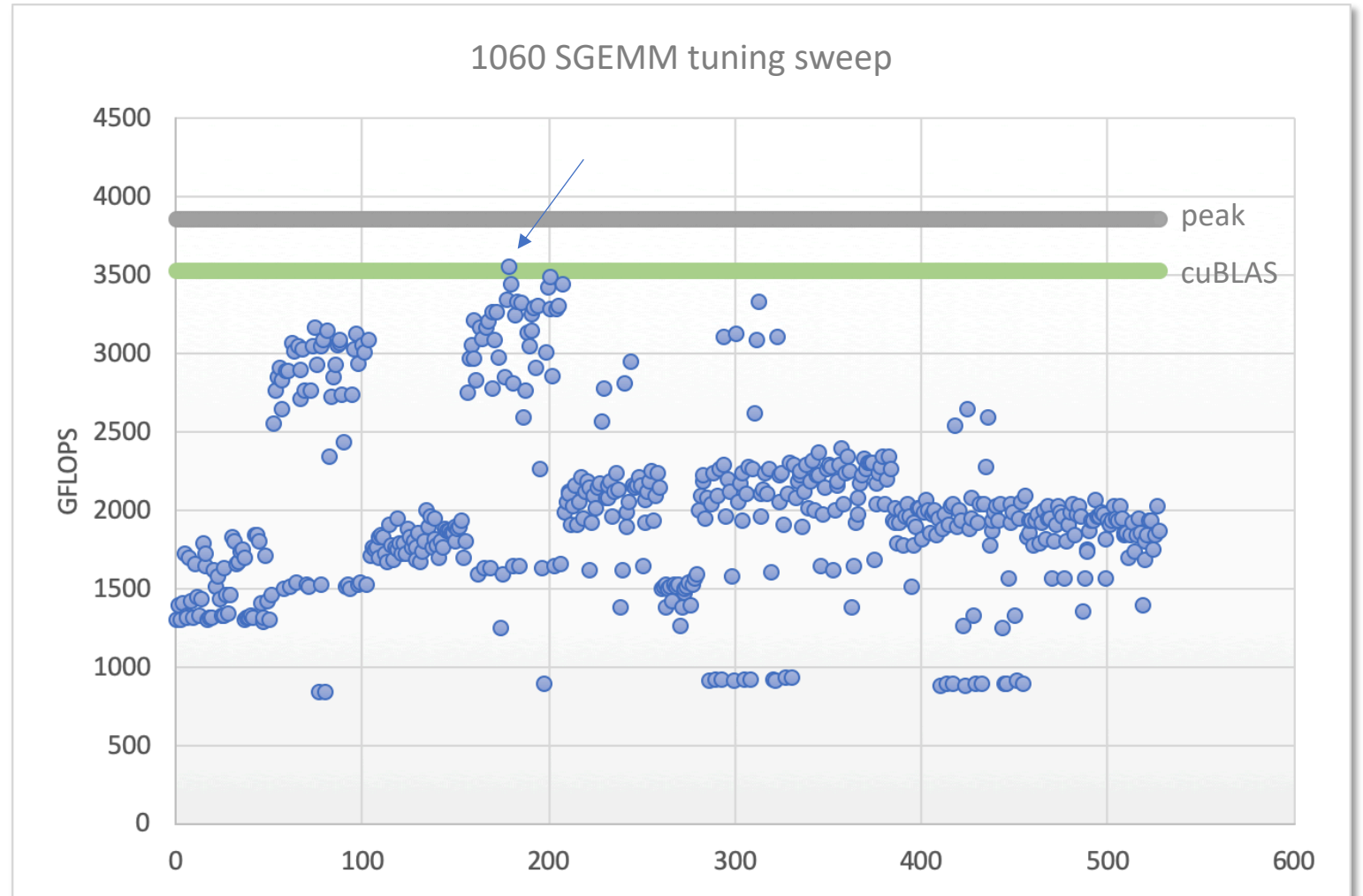


matrix multiplication

sweep results

- 128×128 – block of C
- 128×24 – block of A
- 24×128 – block of B

- 16×16 – computing C
- 32×8 – reading A
- 8×32 – reading B



motivation

parallel tuning



Summit @ ORNL

- **27,648 GPUs**

We write codes for massively parallel machines.

Why are we not using massively parallel machines for autotuning?

motivation

parallel tuning

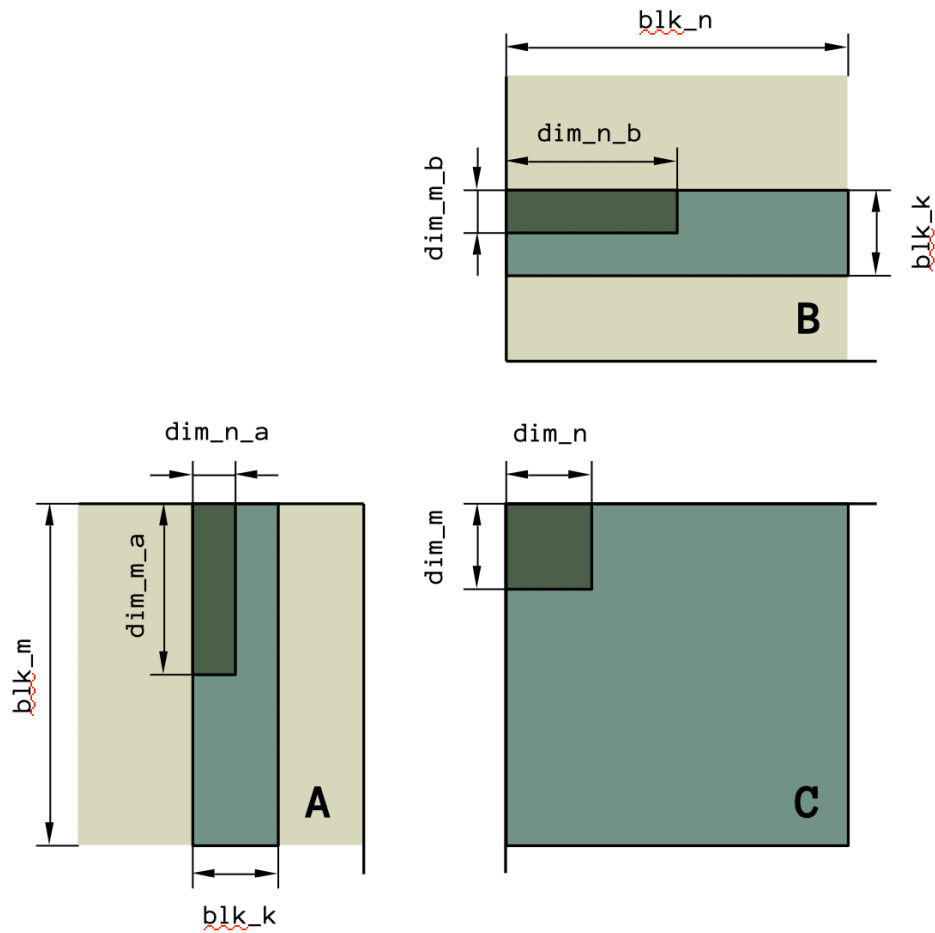


- Summit's HPL run
 - ~9 hours
 - 27,648 GPUs
 - = **248,832** GPU hours

This is equivalent to over **28 years** of using 1 GPU for serial tuning.

motivation

large search spaces



GEMM parameters

- 9 tiling factors
- SIMD / no SIMD
- textures / no textures
- 2 hardware parameters
 - `cudaFuncCache`
 - `cudaSharedMemConfig`
- four precisions
 - single / double
 - real / complex
- A: Trans / NoTrans
- B: Trans / NoTrans
- 3 input dimensions (m, n, k)

motivation

different input sizes

David E. Tanner

Tensile: Auto-tuning GPU Assembly for All Problem Sizes

iWAPT@IPDPS, 2018

	N																											
	16	32	64	112	176	256	352	464	592	736	896	1072	1264	1472	1696	1936	2192	2464	2752	3056	3376	3712	4064	4432	4816	5216	5632	
16	29	62	118	205	314	448	596	735	890	1,024	1,131	1,204	1,401	1,502	1,562	1,625	1,646	1,750	1,754	1,823	1,833	1,887	1,937	1,890	1,976	1,977	2,036	
32	60	116	238	406	615	880	1,137	1,424	1,722	2,006	2,271	2,304	2,628	2,877	2,953	3,120	3,171	3,365	3,452	3,617	3,619	3,654	3,800	3,710	3,848	3,896	3,992	
64	121	236	452	774	1,168	1,635	2,030	2,402	2,785	3,173	3,539	3,625	4,206	4,506	4,723	4,959	4,936	5,146	5,483	5,846	5,644	5,979	6,404	5,869	6,251	6,449	6,510	
112	206	409	790	1,270	1,831	2,365	2,758	3,275	3,590	3,989	4,353	4,398	5,002	5,487	5,422	6,033	5,625	5,791	6,362	6,879	6,652	6,891	7,436	7,058	7,203	7,016	7,458	
176	316	633	1,201	1,848	2,404	3,178	3,457	4,012	4,506	4,695	5,479	5,397	6,223	5,894	6,204	6,908	6,755	7,237	7,001	7,332	7,864	7,630	8,156	7,868	7,704	8,343	8,023	
256	456	878	1,575	2,342	3,065	4,021	4,271	4,847	5,052	5,810	5,780	5,968	6,579	6,898	6,811	7,593	7,095	7,458	7,936	8,597	8,012	8,558	9,211	8,357	8,761	8,387	8,835	
352	604	1,186	2,141	2,971	3,566	4,566	4,720	5,458	5,924	6,001	6,555	6,683	7,363	7,298	7,224	7,899	7,811	7,934	7,886	8,573	8,636	8,741	9,013	8,483	8,534	9,188	8,854	
464	755	1,447	2,489	3,340	4,113	5,100	5,480	5,877	5,810	6,906	7,229	7,075	7,789	7,932	7,927	8,217	8,232	8,403	8,562	8,950	8,526	8,708	9,437	8,882	9,007	8,905	9,378	
592	947	1,711	2,892	3,749	4,510	5,561	6,022	5,942	6,393	7,288	7,512	7,550	7,734	8,013	8,120	8,403	8,677	8,430	8,848	9,118	8,947	8,898	9,573	9,414	9,861	9,461	9,707	
736	1,073	2,018	3,415	4,254	4,966	6,241	6,099	6,967	7,254	7,666	7,913	8,111	8,476	8,693	8,424	9,010	8,796	9,114	9,079	9,678	9,715	9,801	10,293	9,534	9,505	10,182	9,853	
896	1,228	2,225	3,755	4,450	5,394	6,442	6,447	6,794	6,950	7,527	7,974	8,695	8,841	8,860	9,428	9,395	9,785	9,516	9,655	9,758	10,327	10,173	10,338	10,442	10,216	10,456	10,786	
1072	1,304	2,380	3,895	4,616	5,536	6,392	6,786	7,242	7,635	8,173	8,763	8,442	9,037	9,050	9,180	9,259	9,446	9,620	9,836	10,038	9,745	10,054	10,334	10,281	10,016	10,399	10,338	
1264	1,427	2,716	4,490	5,222	6,367	7,212	7,401	7,782	7,755	8,474	9,031	9,012	9,125	9,373	9,506	9,739	9,929	9,614	10,071	10,300	10,133	10,140	11,079	10,544	11,055	10,580	11,158	
1472	1,509	2,924	4,700	5,562	6,147	7,643	7,476	7,889	7,948	8,619	9,981	8,968	9,299	9,585	9,790	9,854	9,897	9,804	10,213	10,185	10,124	10,565	10,824	10,323	10,434	10,564	10,728	
1696	1,608	3,128	4,896	5,594	6,562	7,673	7,503	8,015	8,196	8,409	9,484	9,100	9,521	9,792	9,641	9,605	9,943	9,926	10,056	10,052	10,381	10,304	10,753	10,358	10,234	10,841	10,593	
1936	1,673	3,162	5,085	6,223	6,961	8,097	7,939	8,244	8,373	9,018	9,442	9,245	9,734	9,894	9,617	10,001	10,076	10,313	10,328	10,756	10,301	10,183	10,874	10,575	10,411	10,514	11,049	
2192	1,666	3,291	5,152	6,069	7,078	8,025	7,870	8,285	8,692	8,773	9,876	9,419	9,929	9,894	9,946	10,081	9,841	10,083	10,276	10,515	10,485	10,543	10,970	10,806	10,726	10,663	10,651	
2464	1,746	3,438	5,511	6,385	7,275	8,370	8,098	8,424	8,438	9,103	9,541	9,608	9,661	9,778	9,924	10,247	10,056	10,339	10,607	10,577	10,396	10,549	11,076	11,013	11,005	11,025	11,065	
2752	1,805	3,501	5,703	6,353	7,109	8,311	8,062	8,530	8,825	9,046	9,846	9,765	10,032	10,185	10,053	10,301	10,260	10,617	10,726	10,485	10,647	11,294	11,298	10,608	10,669	10,732	10,868	
3056	1,848	3,623	5,879	6,856	7,724	8,901	8,576	8,932	9,107	9,681	9,918	10,008	10,310	10,268	10,040	10,734	10,520	10,516	10,508	11,474	10,751	11,497	11,552	10,864	11,024	11,207	11,393	
3376	1,836	3,570	5,910	6,907	7,934	8,733	8,671	8,510	8,930	9,739	10,468	9,701	10,125	10,145	10,398	10,296	10,481	10,394	10,669	10,722	10,744	10,801	10,989	11,199	10,775	11,037	11,290	
3712	1,890	3,701	6,048	6,773	7,709	8,632	8,643	8,630	8,760	9,770	10,161	9,957	10,041	10,525	10,275	10,136	10,441	10,462	11,227	11,372	10,682	10,935	11,159	11,420	11,065	11,343	11,700	
4064	1,948	3,824	6,511	7,413	8,252	9,440	8,982	9,380	9,553	10,282	10,555	10,289	11,005	10,828	10,748	10,806	10,915	11,081	11,257	11,483	10,935	11,228	11,521	11,210	11,548	11,328	11,685	
4432	1,938	3,700	6,049	7,225	7,945	9,047	8,473	8,873	9,381	9,555	10,561	10,279	10,599	10,338	10,382	10,563	10,819	11,061	10,643	10,856	11,197	11,554	11,277	11,100	11,480	11,371	11,310	
4816	1,966	3,819	6,334	7,244	7,879	9,041	8,608	9,070	9,868	9,534	10,245	9,994	11,052	10,466	10,211	10,420	10,701	11,044	10,694	11,023	10,786	11,199	11,605	11,480	11,416	11,375	11,396	
5216	1,976	3,920	6,601	7,198	8,310	8,942	9,198	8,960	9,429	10,168	10,722	10,314	10,655	10,538	10,822	10,464	10,578	11,008	10,755	11,146	10,974	11,427	11,338	11,310	11,311	11,364	11,433	
5632	2,034	3,996	6,180	7,169	7,800	9,051	8,528	9,295	9,440	9,765	10,740	10,183	10,998	10,577	10,521	10,906	10,524	10,999	10,797	11,250	11,160	11,654	11,602	11,166	11,244	11,355	11,513	

M	N																											
	16	32	64	112	176	256	352	464	592	736	896	1072	1264	1472	1696	1936	2192	2464	2752	3056	3376	3712	4064	4432	4816	5216	5632	
16	3	3	3	3	3	3	3	3	1	3	1	7	11	26	1	26	1	26	26	18	18	14	18	15	15	1	15	
32	3	3	3	3	3	3	3	3	3	3	1	7	1	18	1	18	1	18	14	18	18	14	18	15	15	29	15	
64	3	3	3	3	3	3	3	12	7	12	7	12	12	14	23	30	12	0	30	30	23	30	30	0	30	5	5	
112	8	3	3	3	3	1	12	23	10	0	28	25	25	28	28	28	25	22	5	5	28	5	5	28	5	28	28	
176	3	3	3	3	23	4	23	13	28	25	25	28	28	28	5	5	28	5	28	28	5	28	5	5	6	6	5	
256	8	3	3	1	26	14	9	10	17	22	22	17	22	5	22	24	22	5	24	24	5	6	6	5	24	5	24	
352	8	3	3	4	4	2	25	28	28	25	5	28	5	28	28	5	5	6	5	5	5	5	5	20	24	27	27	5
464	3	3	1	7	4	10	28	28	30	5	28	28	5	5	5	20	5	5	5	20	24	5	27	24	5	5	24	
592	3	3	7	4	25	25	28	25	30	5	28	28	28	20	5	5	5	5	24	27	5	5	20	24	27	24	20	
736	3	3	4	2	25	28	25	5	5	28	28	28	20	5	5	5	20	24	5	5	24	24	24	27	20	5	27	24
896	11	1	4	2	17	2	21	17	17	28	28	6	5	5	24	24	27	24	24	24	27	24	6	27	24	6	27	
1072	8	1	4	25	28	25	25	28	28	28	6	5	5	5	27	5	5	5	5	24	5	6	24	27	24	6	6	
1264	11	1	4	25	28	28	5	5	28	6	5	5	5	27	5	20	24	5	20	24	24	27	27	27	27	27	27	
1472	11	18	4	28	25	5	25	5	6	5	28	5	27	5	24	27	24	5	6	20	6	24	27	6	6	6	6	
1696	1	1	7	10	25	28	25	5	28	28	24	27	5	24	5	20	24	24	24	27	24	27	6	20	27	27	27	
1936	11	1	17	28	28	5	5	20	5	20	24	5	20	27	20	27	27	24	27	24	27	24	27	27	24	6	27	
2192	1	26	12	25	25	28	5	5	5	24	27	5	24	24	24	6	16	24	27	24	6	27	27	27	27	27	27	
2464	26	26	17	28	5	5	28	5	5	27	24	5	5	5	5	27	6	24	27	20	24	6	27	27	27	27	27	
2752	26	26	28	5	28	28	28	24	5	24	5	5	5	20	24	27	24	24	27	24	24	24	27	27	20	20	27	
3056	18	26	22	28	28	24	5	20	27	24	5	24	24	20	24	27	27	20	24	27	16	27	27	27	27	27	27	
3376	11	11	28	25	5	5	5	5	5	6	27	5	24	16	27	24	24	24	24	6	24	27	27	27	27	27	27	
3712	3	18	22	5	28	6	5	27	5	24	24	24	24	24	24	24	24	24	27	27	27	27	27	27	27	27	27	
4064	18	18	22	5	5	6	20	27	20	27	20	24	27	27	27	27	27	27	27	27	27	27	27	27	27	27	27	
4432	8	15	17	28	5	5	24	24	20	27	27	20	16	16	27	27	27	16	27	27	27	27	27	27	27	27	27	
4816	26	15	22	5	28	28	5	5	27	5	24	24	27	16	24	20	27	16	27	27	27	27	27	27	27	27	27	
5216	11	19	5	28	5	28	27	5	24	20	20	24	20	27	16	27	20	27	27	27	27	27	27	27	27	27	27	
5632	19	19	22	24	5	5	24	24	6	24	27	24	27	24	27	27	27	27	27	27	27	27	27	27	27	27	27	

motivation

GEMM vs convolution

DNN convolution is basically a GEMM with slightly altered address arithmetic.

So, once we have a fast GEMM, we have a fast convolution, right?

```
1 for(int k=0; k<K; k+=16) {
2   __local float* plA = lA + idx*97+idy;
3   __local float* plB = lB + idx*97+idy;
4
5   barrier(CLK_LOCAL_MEM_FENCE);
6
7   //Load next submatrix into local memory
8
9
10
11
12
13
14
15   for(int i=0; i<96; i+=16)
16     plA[i] = A[i];
17   for(int i=0; i<96; i+=16)
18     plB[i] = B[i*ldb];
19
20
21
22   barrier(CLK_LOCAL_MEM_FENCE);
23
24   //Inner computation loop
25   ...
26
27   //Move to next submatrix
28   A += 16*lda;
29   B += 16;
30
31 }
```

```
1 for(int k=0; k<K; k+=16) {
2   __local float* plA = lA + idx*97+idy;
3   __local float* plB = lB + idx*97+idy;
4   int x = p * stride_u + r;
5   int y = q * stride_v + s;
6   barrier(CLK_LOCAL_MEM_FENCE);
7
8   //Load next submatrix into local memory
9   //Check if it's in the padding region
10  if( x < pad_h || x >= H+pad_h ||
11      y < pad_w || y >= W+pad_w )
12    for(int i=0; i<96; i+=16)
13      plA[i] = 0;
14  else
15    for(int i=0; i<96; i+=16)
16      plA[i]=A[index+c*N*H*W+r*N*W+s*N+i];
17  for(int i=0; i<96; i+=16)
18    plB[i] = B[i*ldb];
19
20  //Update indices for next submatrix
21  s += 16; r += s/S; s = s%S;
22  c += r/R; r = r%R;
23  barrier(CLK_LOCAL_MEM_FENCE);
24
25  //Inner computation loop
26  ...
27
28  //Move to next submatrix
29  A += 16*lda;
30  B += 16;
31 }
```

Y. Tsai et al.

Performance-portable autotuning of OpenCL kernels for convolutional layers of deep neural networks

MLHPC@SC, 2016

motivation

GEMM vs convolution

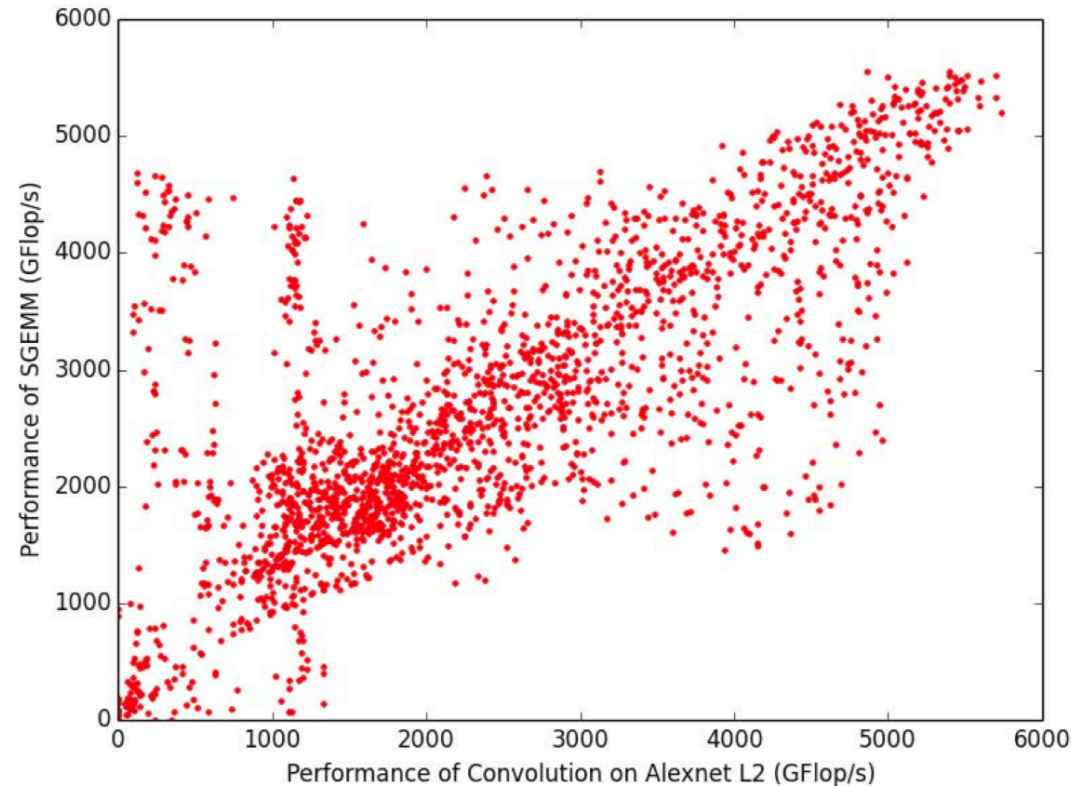
DNN convolution is basically a GEMM with slightly altered address arithmetic.

So, once we have a fast GEMM, we have a fast convolution, right?

Not necessarily.

We need to retune.

We may have catastrophic performance degradation.



Y. Tsai et al.

Performance-portable autotuning of OpenCL kernels for convolutional layers of deep neural networks

MLHPC@SC, 2016

motivation

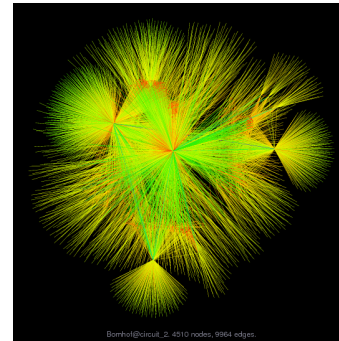
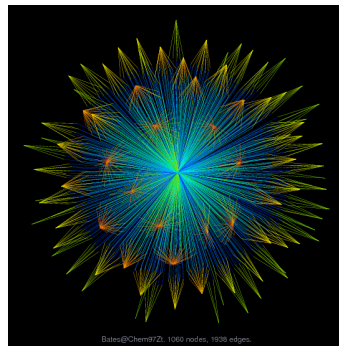
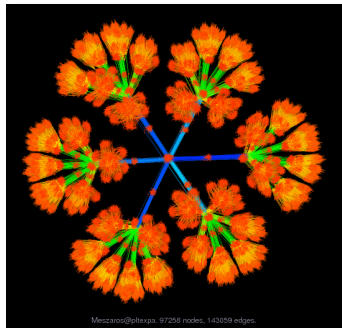
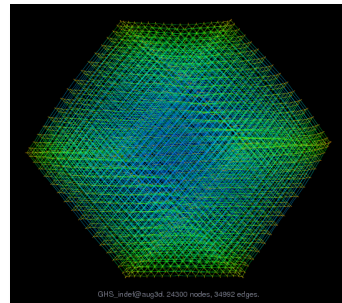
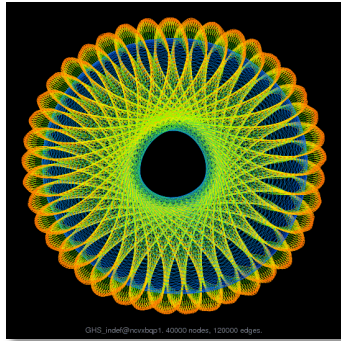
different data structures

SuiteSparse Matrix Collection

Tim Davis @ Texas A&M

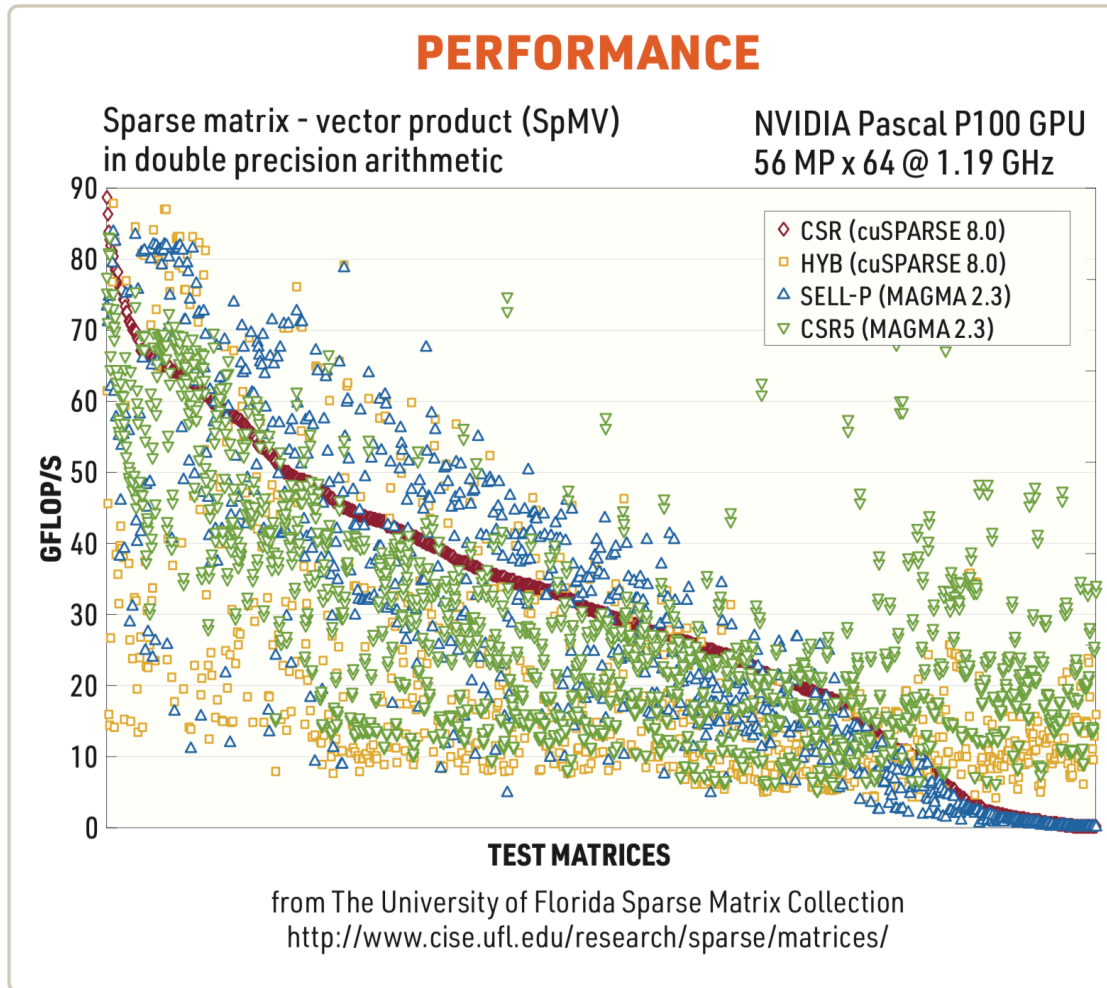
SpMV parameters

- 2,833 matrices
- a dozen storage formats
 - CSC/CSR
 - ELLPACK
 - SELL-C/SELL-P
 - Sell-C-Sigma
 - BCSR
 - DIA
 - COO
 - ...
- multiple precisions
- hardware parameters
- implementation tuning parameters



motivation

different data structures



SpMV parameters

- 2,833 matrices
- a dozen storage formats
 - CSC/CSR
 - ELLPACK
 - SELL-C/SELL-P
 - Sell-C-Sigma
 - BCSR
 - DIA
 - COO
 - ...
- multiple precisions
- hardware parameters
- implementation tuning parameters

motivation

collect hardware counters

Metric Name	Description	Scope
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor	Multi-context
atomic_transactions	Global memory atomic and reduction transactions	Multi-context
atomic_transactions_per_request	Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction	Multi-context
branch_efficiency	Ratio of branch instruction to sum of branch and divergent branch instruction	Multi-context
cf_executed	Number of executed control-flow instructions	Multi-context
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10	Multi-context
cf_issued	Number of issued control-flow instructions	Multi-context
double_precision_fu_utilization	The utilization level of the multiprocessor function units that execute double-precision floating-point instructions on a scale of 0 to 10	Multi-context
dram_read_bytes	Total bytes read from DRAM to L2 cache	Multi-context
dram_read_throughput	Device memory read throughput	Multi-context
dram_read_transactions	Device memory read transactions	Multi-context
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10	Multi-context
dram_write_bytes	Total bytes written from L2 cache to DRAM	Multi-context
dram_write_throughput	Device memory write throughput	Multi-context
dram_write_transactions	Device memory write transactions	Multi-context
eligible_warps_per_cycle	Average number of warps that are eligible to issue per active cycle	Multi-context
flop_count_dp	Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, and multiply-accumulate). Each multiply-accumulate operation contributes 2 to the count.	Multi-context

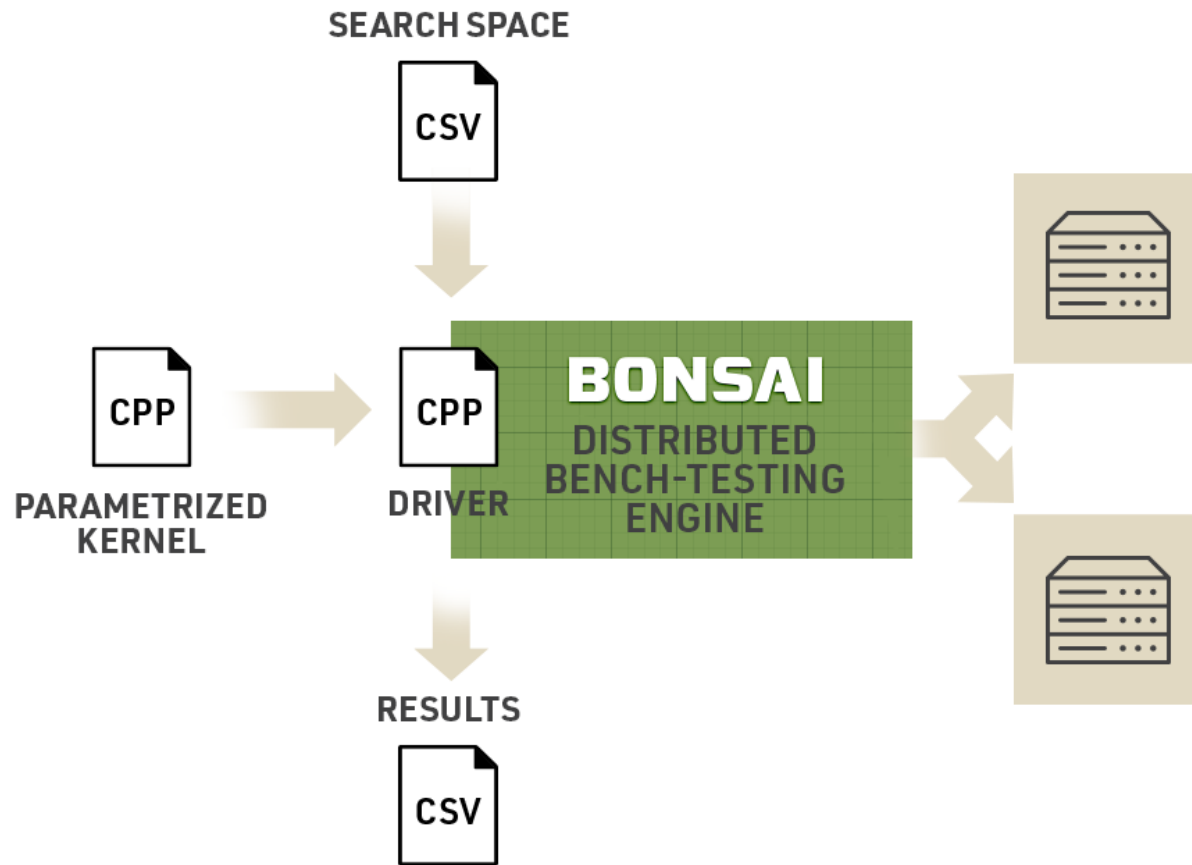
Capability 7.x Metrics

- 62 events
- 176 metrics
- cannot collect all in one launch

Even if we just want the time/GFLOPS, we need multiple runs for statistically significant results.

BONSAI

principles of operation



deploy large sweeps

- distributed memory
- parallel compilation (multicore)
- parallel benchmarking (multi-GPU)

J. Kurzak et al.

Massively Parallel Automated Software Tuning
ICPP, 2019

```

// Thread block size
#define BLOCK_SIZE 16

// Forward declaration of the matrix multiplication kernel
__global__ void MatMulKernel(const Matrix, const Matrix, Matrix);

// Matrix multiplication - Host code
// Matrix dimensions are assumed to be multiples of BLOCK_SIZE
void MatMul(const Matrix A, const Matrix B, Matrix C)
{
    ...

    // Invoke kernel
    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
    dim3 dimGrid(B.width / dimBlock.x, A.height / dimBlock.y);
    MatMulKernel<<<dimGrid, dimBlock>>>>(d_A, d_B, d_C);

    ...
}

// Matrix multiplication kernel called by MatMul()
__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)
{
    ...

    for (int m = 0; m < (A.width / BLOCK_SIZE); ++m) {

        ...

        // Shared memory used to store Asub and Bsub respectively
        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

        // Load Asub and Bsub from device memory to shared memory
        // Each thread loads one element of each sub-matrix
        As[row][col] = GetElement(Asub, row, col);
        Bs[row][col] = GetElement(Bsub, row, col);

        __syncthreads();

        // Multiply Asub and Bsub together
        for (int e = 0; e < BLOCK_SIZE; ++e)
            Cvalue += As[row][e] * Bs[e][col];

        __syncthreads();

    }

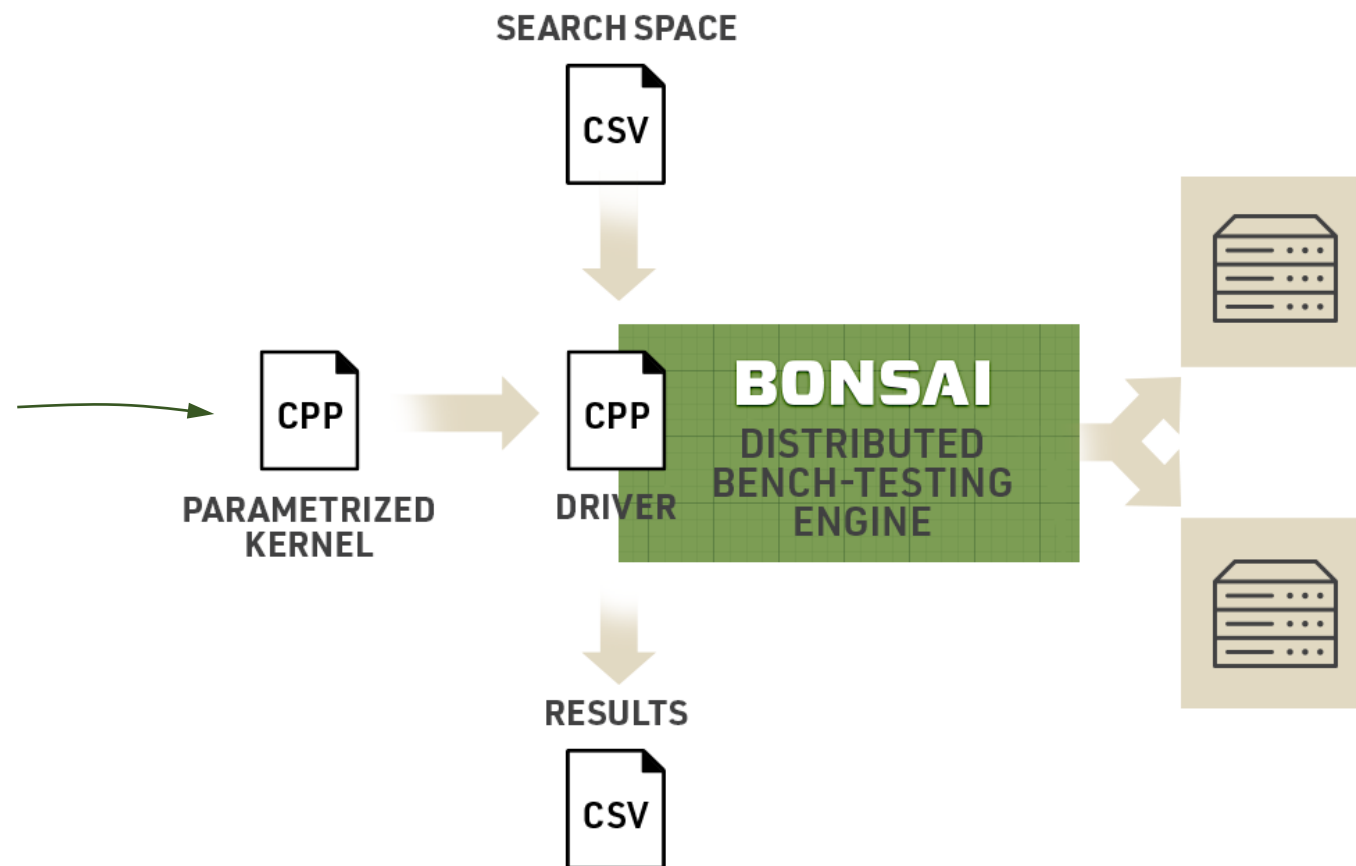
    ...
}

```

-DBLOCK_SIZE

BONSAI

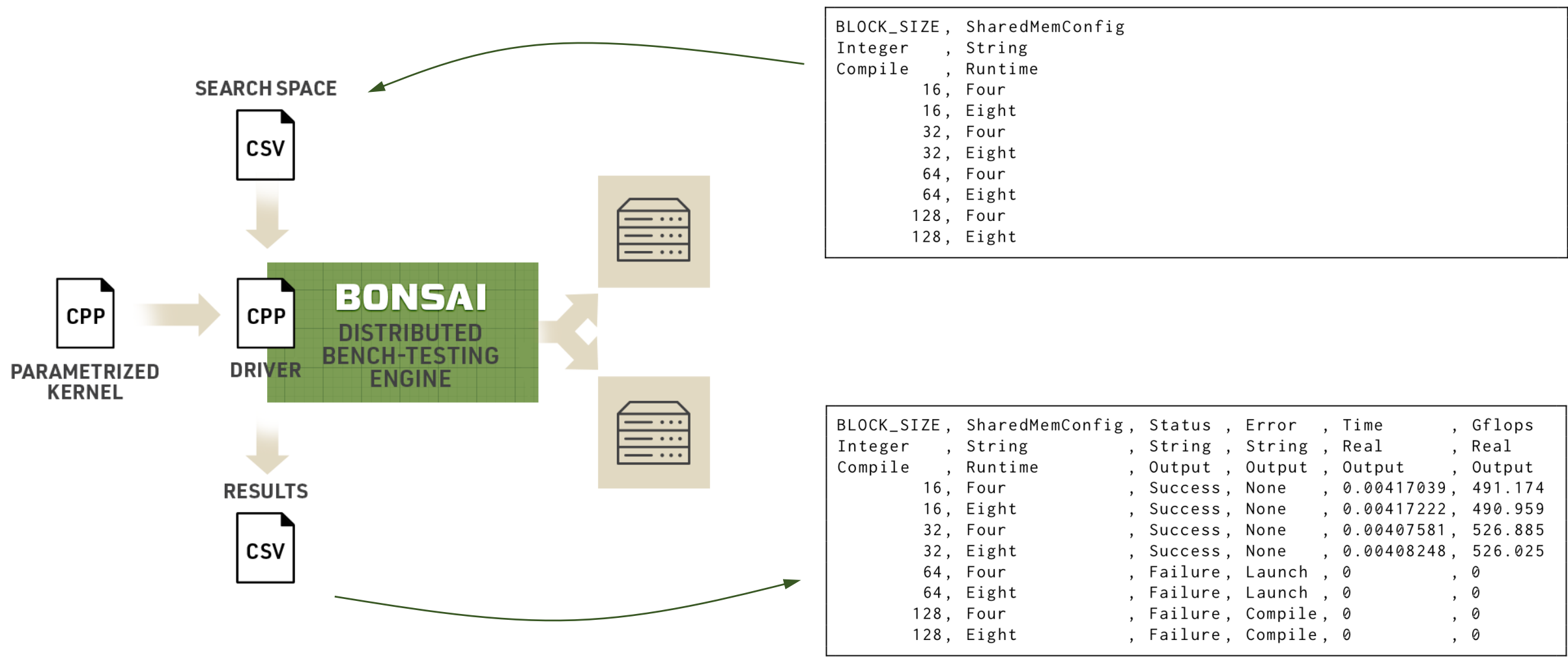
parametrized kernels



<https://bitbucket.org/icl/bonsai/src/default/examples/matmul/>

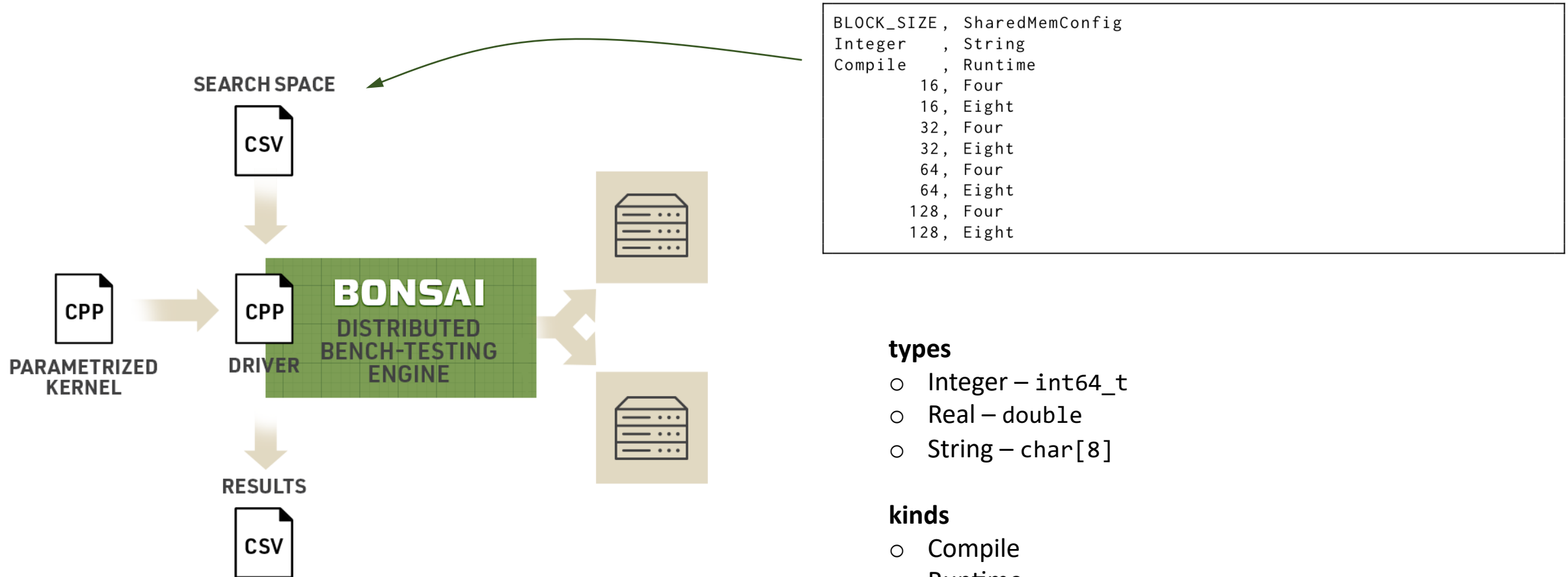
BONSAI

CSV input and output



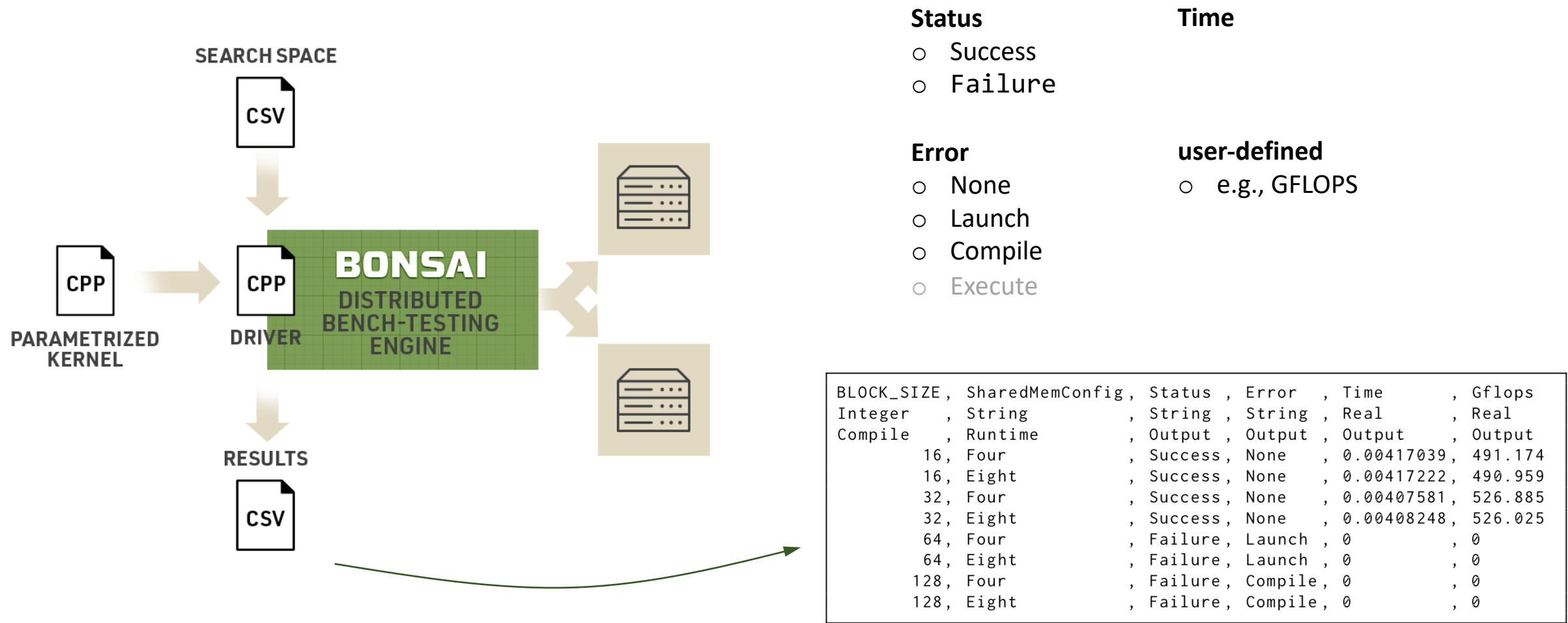
BONSAI

input parameters



BONSAI

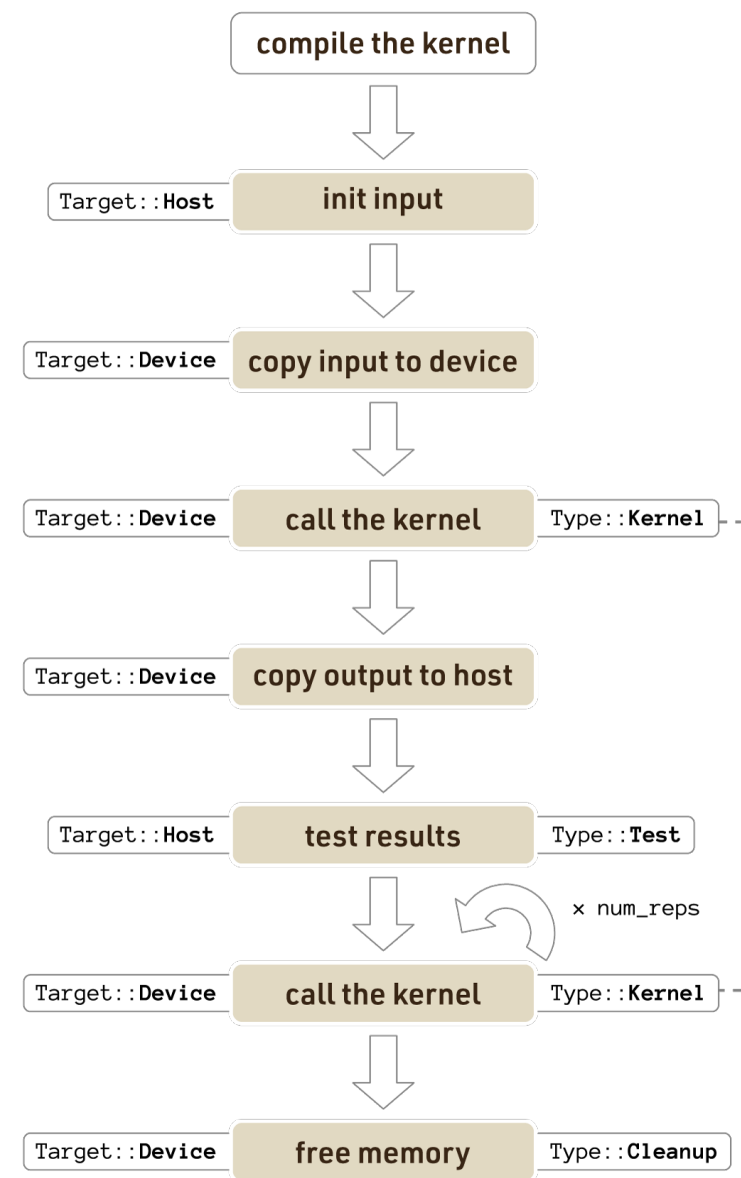
output parameters



BONSAI

processing pipeline

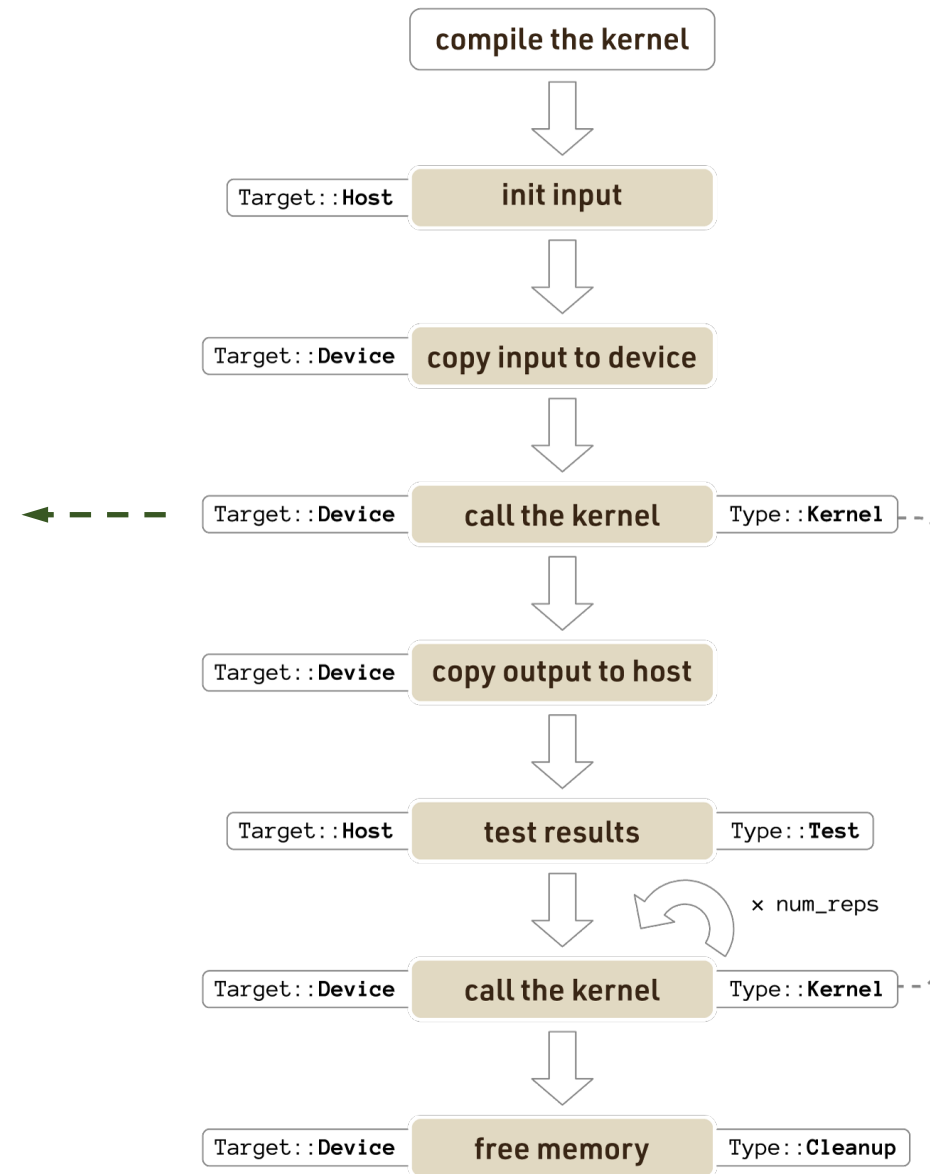
```
try {  
    using namespace bonsai;  
    using Target = Callback::Target;  
    using Type    = Callback::Type;  
  
    bonsai::Sweep sweep(  
        std::string("input.csv"),    // input CSV filename  
        std::string("output.csv"),   // output CSV filename  
        std::string("kernel.cu"),    // kernel source filename  
        std::string("MatMul"),       // kernel function name  
        sizeof(CallbackData),        // size of callback data  
        num_reps,                    // number of repetitions  
        chunk_size,                  // dispatch chunk size  
        std::string("-arch=sm_60")); // extra compilation flags  
  
    sweep.addParameter(std::string("Gflops"), Value::Type::Real);  
  
    sweep.addCallback(init_input, Target::Host);  
    sweep.addCallback(copy_input, Target::Device);  
    sweep.addCallback(call_kernel, Target::Device, Type::Kernel);  
    sweep.addCallback(copy_output, Target::Device);  
    sweep.addCallback(free_mem, Target::Device, Type::Cleanup);  
    sweep.addCallback(test_result, Target::Host, Type::Test);  
  
    sweep.run();  
}  
catch (bonsai::Exception& e) {  
    std::cerr << e.what() << std::endl;  
}
```



runtime parameters

input runtime parameters accessible in the callback:

ICLUT



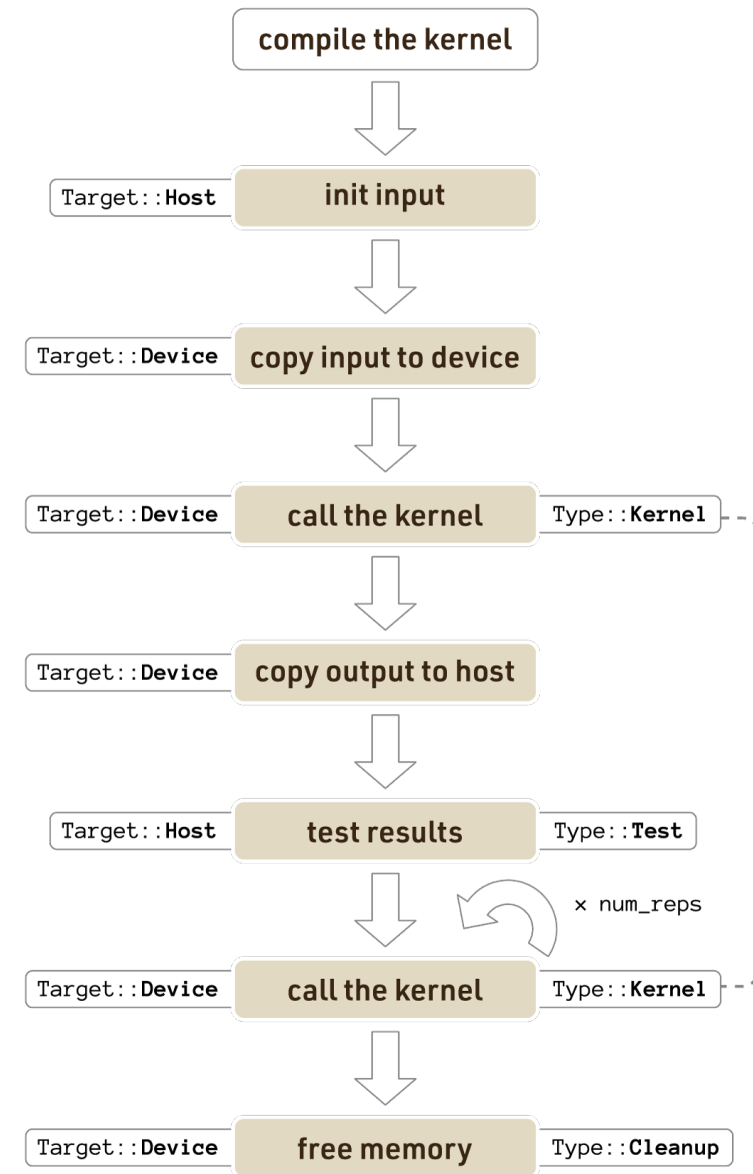
BONSAI

output parameters

output runtime parameters accessible in the callback:

```
kernel_params[std::string("ParameterName")].string  
                                                    .integer  
                                                    .real
```

```
// Performs cleanup tasks (Target::Device, Type::Cleanup)  
bool free_mem(void* callback_data,  
              bonsai::KernelParams& kernel_params,  
              bonsai::Kernel::Func kernel_func,  
              cudaStream_t stream)  
{  
    cudaFree(...);  
    ...  
    free(...);  
    ...  
    double time = kernel_params.at(std::string("Time")).real;  
    double gflops = ... / time / 1000000000.0;  
    kernel_params[std::string("Gflops")].real = gflops;  
    return cudaGetLastError() == cudaSuccess;  
}
```



BONSAI

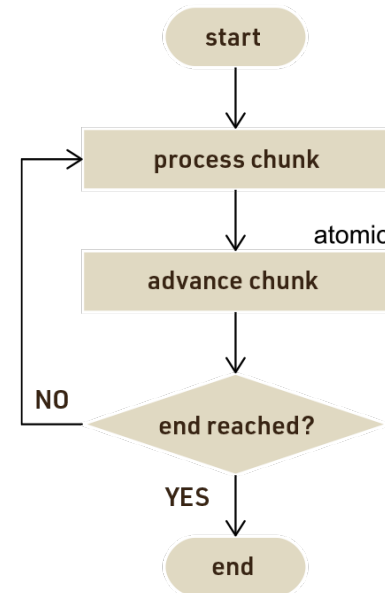
rank 0 processing

basically distributed memory implementation of

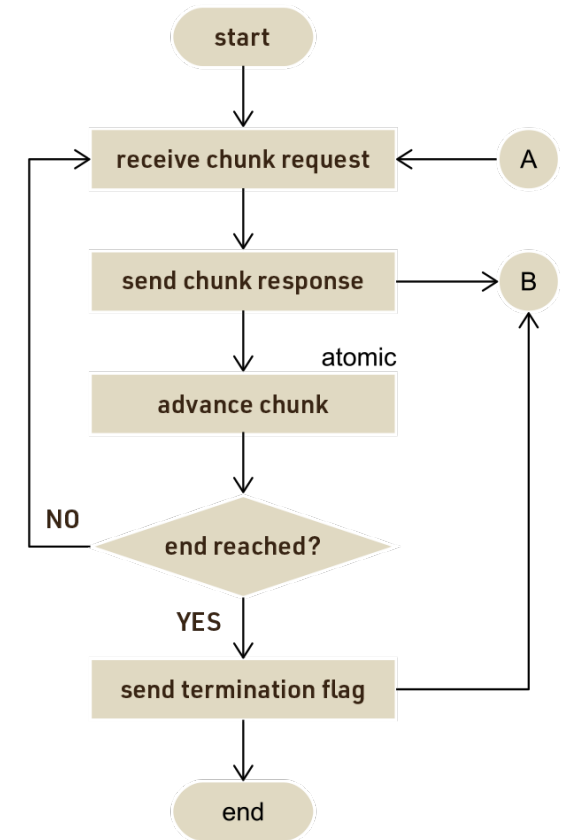
- o `#pragma omp parallel for schedule(dynamic , chunk)`

```
void Sweep::run()
{
    if (mpi_rank_ == mpi_root_) {
        #pragma omp parallel
        {
            if (mpi_size_ > 1) {
                #pragma omp master
                {
                    // server code
                }
            }
            #pragma omp single
            {
                // local processing
            }
        }
    }
    else {
        #pragma omp parallel
        {
            #pragma omp master
            {
                // client code
            }
        }
    }
}
```

MPI root: local processing



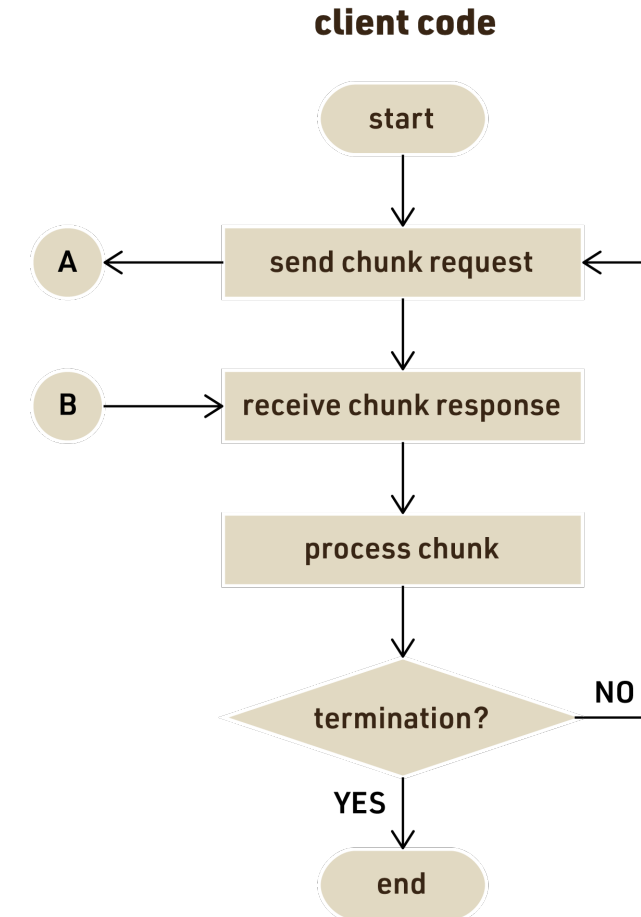
MPI root: server code



BONSAI

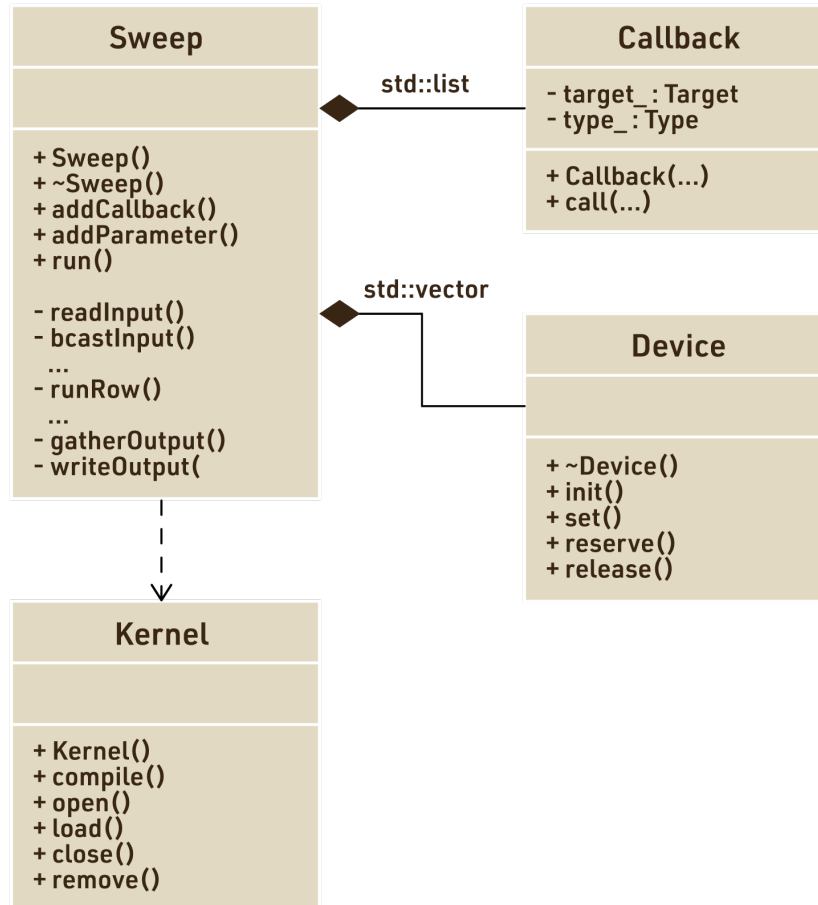
rank != 0 processing

```
void Sweep::run()
{
    if (mpi_rank_ == mpi_root_) {
        #pragma omp parallel
        {
            if (mpi_size_ > 1) {
                #pragma omp master
                {
                    // server code
                }
            }
            #pragma omp single
            {
                // local processing
            }
        }
    }
    else {
        #pragma omp parallel
        {
            #pragma omp master
            {
                // client code
            }
        }
    }
}
```



BONSAI

software



- C++
- header only implementation
- standard library containers
- reasonably commented
- Doxygenated
- <https://bitbucket.org/icl/bonsai/>
- modified BSD license

experiment

GEMM tuning sweep

GTX 1060
6GB



sgemm tuning sweep

○ **44 min and 16 sec**

experiment

GEMM tuning sweep

GTX 1060
6GB



sgemm tuning sweep

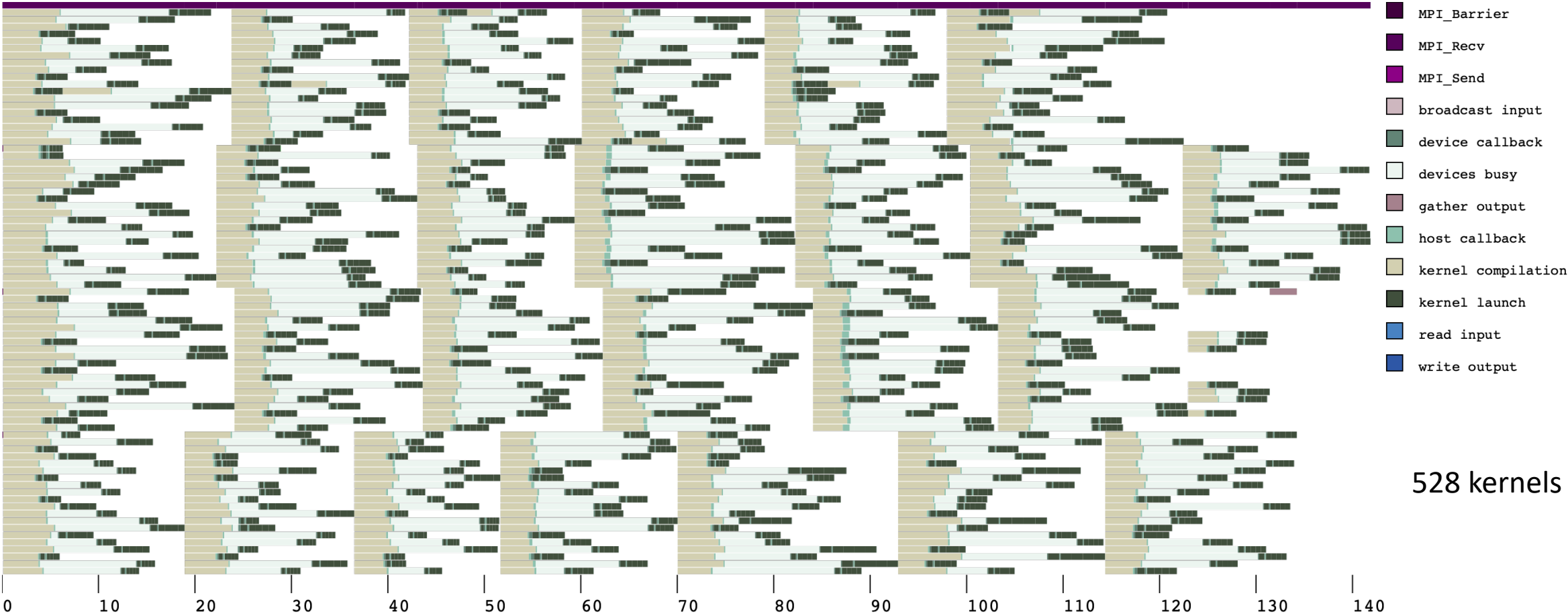
○ **44 min and 16 sec**

16 GTX 1060s



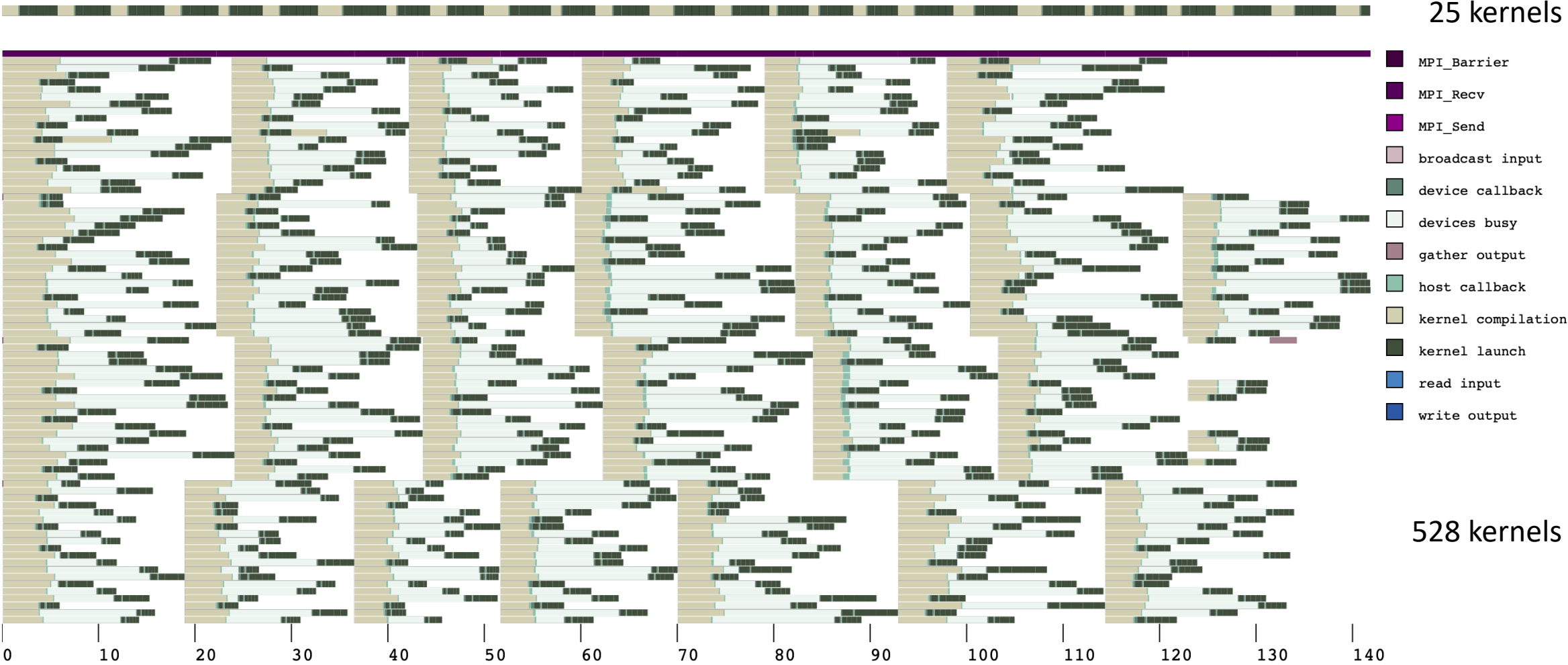
experiment

GEMM tuning sweep



experiment

GEMM tuning sweep



experiment

GEMM tuning sweep

- serial sweep: 45 min
- parallel sweep: 2.5 min
- speedup: 18x

16 GTX 1060s



