Mergesort with CUDA

25.02.2020

Architecture

Mapping Cuda calls to Hardware

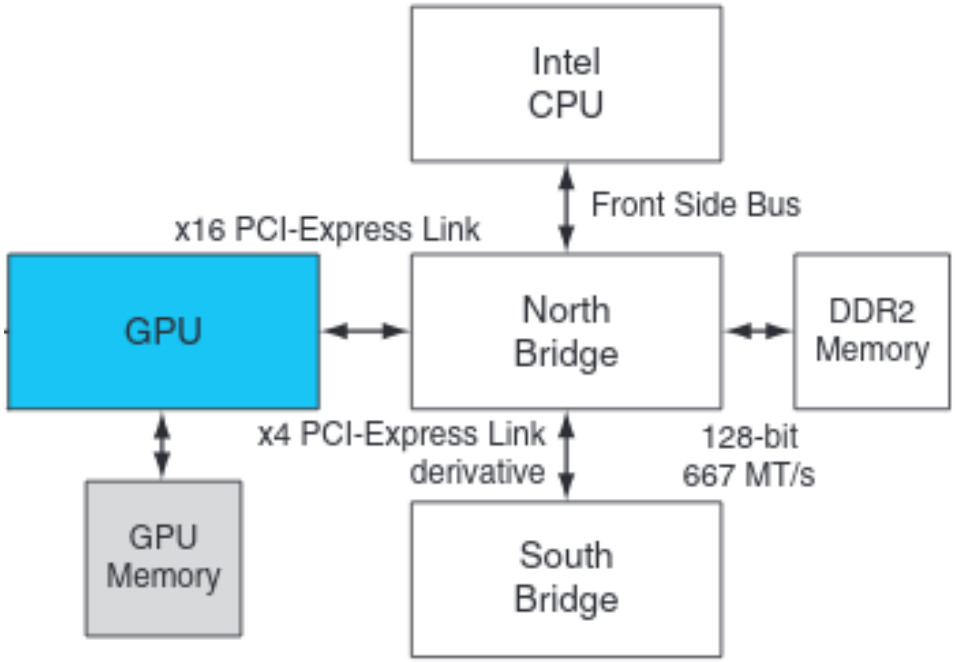
Merge

Memory Hirachy of CUDA

Merging with local memory

Architecture

GPU and CPU uses different memories



Init:

```
1 T* gpup;
2 int sz = 10*sizeof(int);
3 cudaMalloc((void**)&gpup, sz);
```

Memory Transfer:

```
1 cudaMemcpy(gpup, cpup, nBytes, cudaMemcpyHostToDevice);
```

Ideas for Memory Management

```
1  class Storage {
2     public:
3      explicit Storage(const std::vector<T>&);
4
5     private:
6      std::vector<T> _data;
7      T* _cpu_pointer;
8      T* _gpu_pointer;
9      void initialize_gpu_memory();
10 };
```

Memory pool, takes ownership

Initializes the gpu memory as copy

Pointers for cpu/gpu locations

Lazy Memory Sync

```
1  class Storage {
2      public:
3       T* cpu_pointer();
4       T* gpu_pointer();
5       const T* cpu_pointer_const();
6       const T* gpu_pointer_const();
7
8      private:
9       std::string head; \\ head = {CPU, GPU, SYNC}
10      void sync_to_cpu();
11      void sync_to_gpu();
12 };
```

Example:

After Initialization: `head = SYNC`

Access, `gpu_pointer_const()`: `head` remains

Access `cpu_pointer()`: `head = CPU`

Acess `gpu_pointer()`: `sync_to_cpu(); head = SYNC`

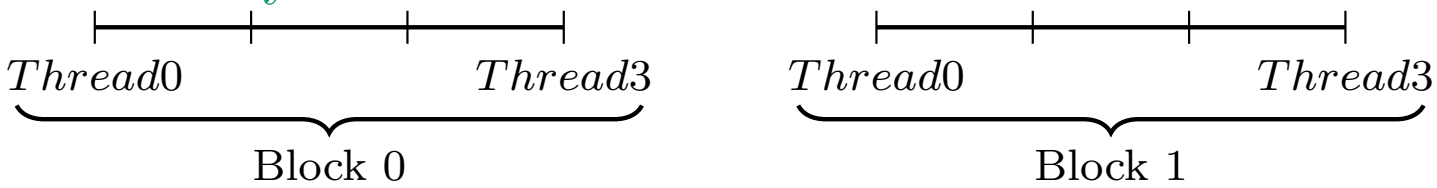
Mapping Cuda calls to Hardware

Launching CUDA threads

Cuda Program

```
dim3 Grid(2)
dim3 Block(4)
add_kernel<<<Grid, Block>>>(...)
```

Thread Layout:

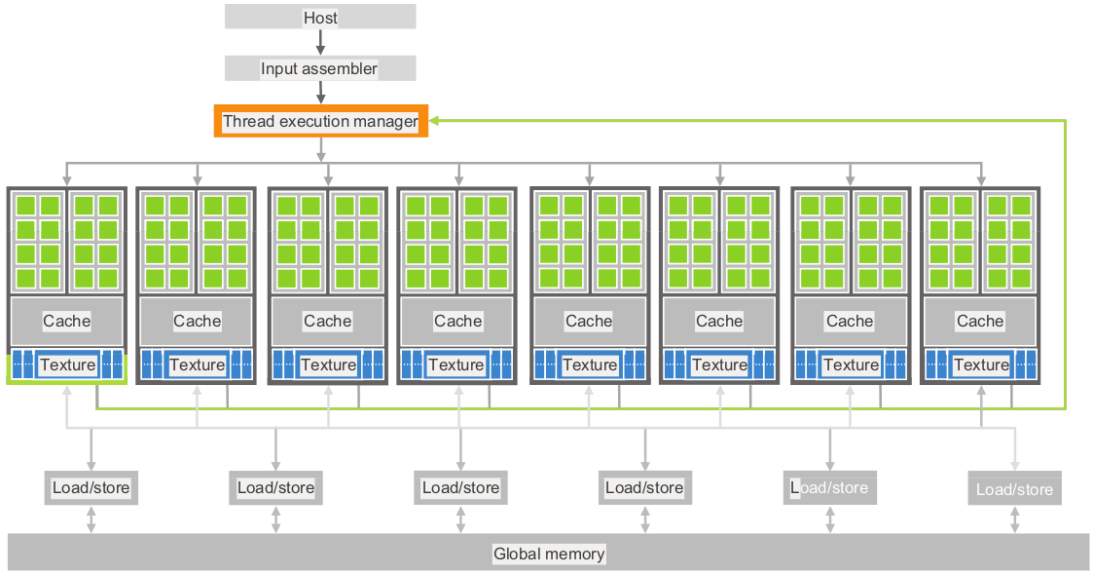


Addition:

```
__global__
add_kernel(float* A, float* B, float* C, int n) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n) {
        C[i] = A[i] + B[i];
    }
}
```

Software/Hardware connection

Programmer: Specifies number of blocks and threads

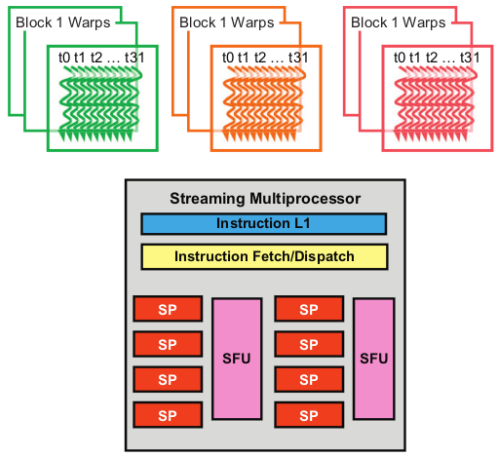


Dispatch:

1. Blocks to SM: Block 1 → SM1, Block 2 → SM2 ...

Software/Hardware connection

Programmer: Specifies number of blocks and threads

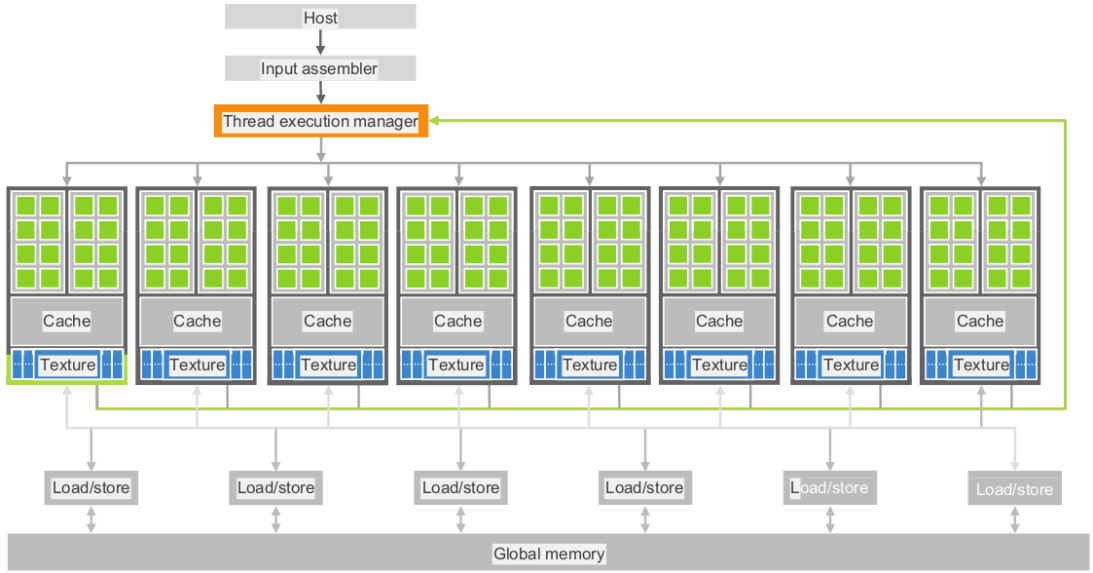


Dispatch:

1. Blocks to SM: Block 1 $\rightarrow$ SM1, Block 2 $\rightarrow$ SM2 …
2. Grouping: Wrappend threads are allocated

Software/Hardware connection

Programmer: Specifies number of blocks and threads



Dispatch:

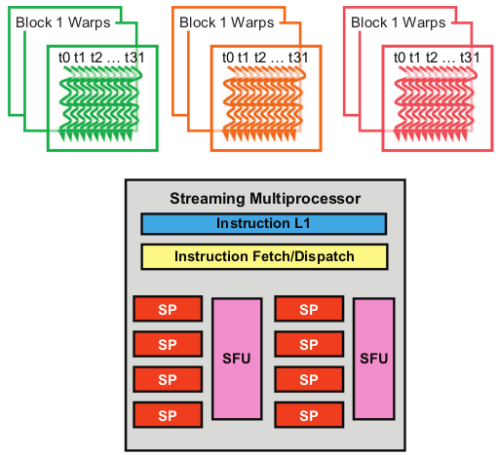
1. Blocks to SM: Block 1 → SM1, Block 2 → SM2 ...
2. Grouping: Wrappend threads are allocated

Stengths:

Quantity: No branch prediction/ chaches, just cores

Software/Hardware connection

Programmer: Specifies number of blocks and threads



Dispatch:

1. Blocks to SM: Block 1 $\rightarrow$ SM1, Block 2 $\rightarrow$ SM2 …
2. Grouping: Wrappend threads are allocated

Stengths:

Quantity: No branch prediction/ chaches, just cores

Latency Hiding: If wrap 1 stalls, scheduler starts wrap 2

Merge

Basic Merge Operation

$$A = 5 \quad 7 \quad 8 \quad 9 \quad 12 \quad 14 \quad 15 \quad 16$$

$$B = 1 \quad 2 \quad 3 \quad 4 \quad 6 \quad 10 \quad 11 \quad 13$$

$$C = ? \quad ? \quad ? \quad ? \quad ? \quad ? \quad ? \quad \ldots \quad ?$$

```
1  void merge(T* a, T* b, T* c, int sz_a, int sz_b) {
2      int i = 0, j = 0, k = 0;
3      while (k < sz_a + sz_b) {
4          if (i == sz_a)
5              c[k++] = b[j++];
6          else if (j == sz_b)
7              c[k++] = a[i++];
8          else if (a[i] <= b[j])
9              c[k++] = a[i++];
10         else
11             c[k++] = b[j++];
12     }
13 }
```

Merge

How to split A and B to spwan many threads?

Example:

$$A = 0 \quad 0 \quad 0 \quad 0$$

$$B = 1 \quad 1 \quad 1 \quad 1$$

$$C = ? \quad ? \quad ? \quad ? \quad ? \quad ? \quad ? \quad ?$$

How to split A and B to spwan many threads?

Example:

$$A = 0 \quad 0 \quad 0 \quad 0$$
$$B = 1 \quad 1 \quad 1 \quad 1$$
$$C = ? \quad ? \quad ? \quad ? \quad ? \quad ? \quad ? \quad ?$$

Naive: 2 Threads, half A and B

How to split A and B to spwan many threads?

Example:

$$A = 0 \quad 0 \quad 0 \quad 0$$

$$B = 1 \quad 1 \quad 1 \quad 1$$

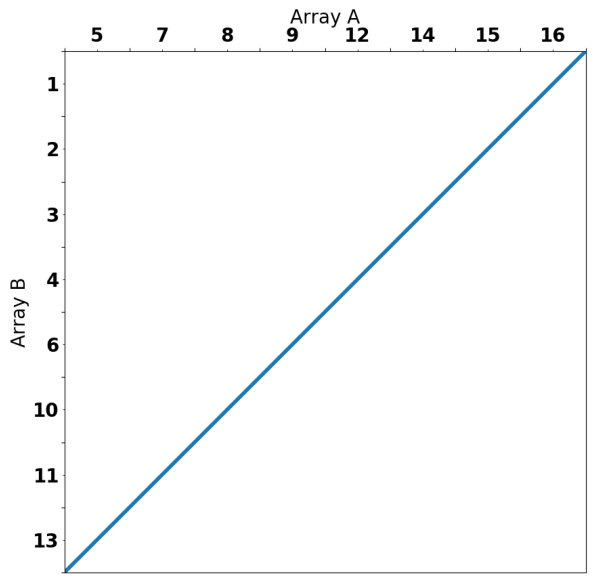
$$C = ? \quad ? \quad ? \quad ? \quad ? \quad ? \quad ? \quad ?$$

Naive: 2 Threads, half A and B Result:

Instead: How to split arrays

NEED TO CITE PAPER!

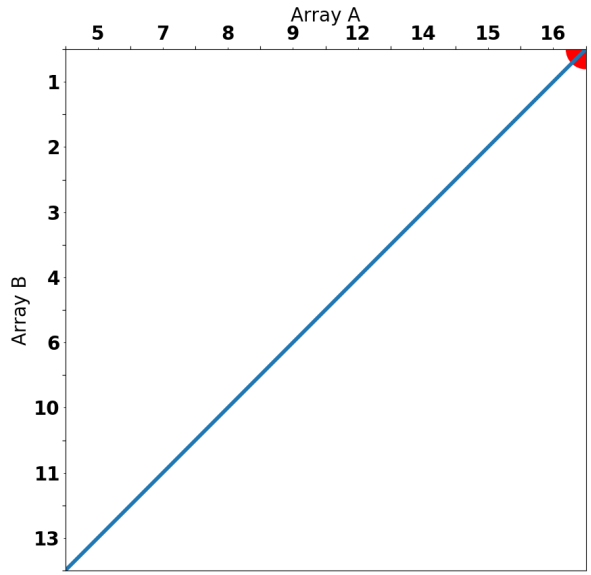
How to split two arrays in equal chuncks?



Instead: How to split arrays

NEED TO CITE PAPER!

One feasible split: Array A to Thread 1, Array B to

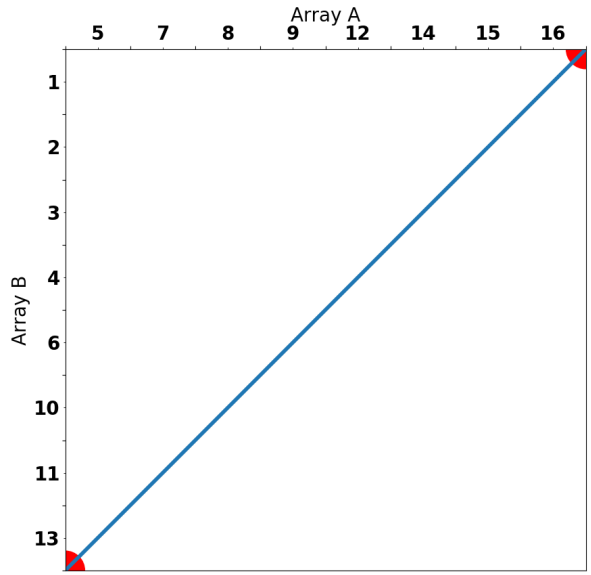


Thread 2

Instead: How to split arrays

NEED TO CITE PAPER!

Another feasible split: Array B to Thread 1, Array A

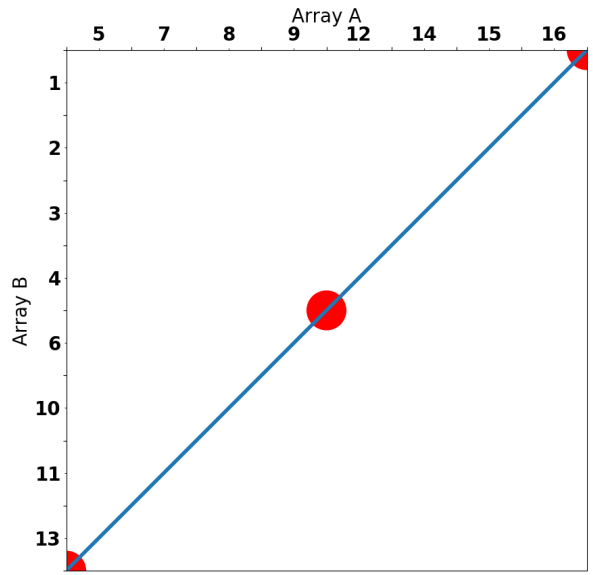


to Thread 2

Instead: How to split arrays

NEED TO CITE PAPER!

Another split (seen before): Thread 1 gets half of A



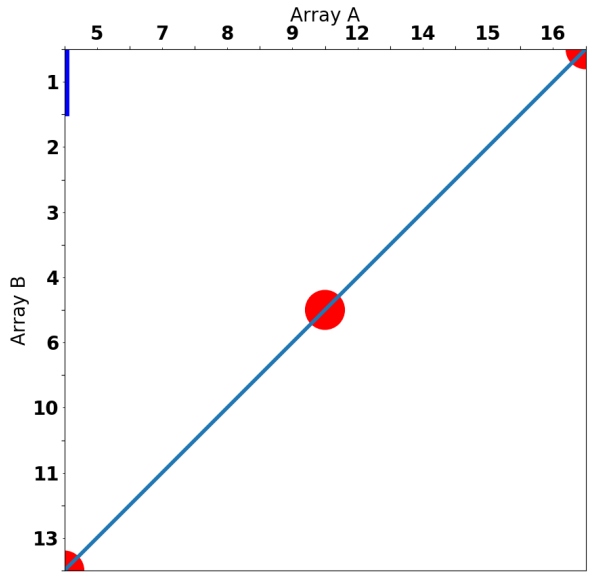
and B

Summary: All allocations along vertical line split work equally!

Instead: How to split arrays

NEED TO CITE PAPER!

Mergepath: Define the optimal split: One Elemet of B



Summary: All allocations along vertical line split work equally!

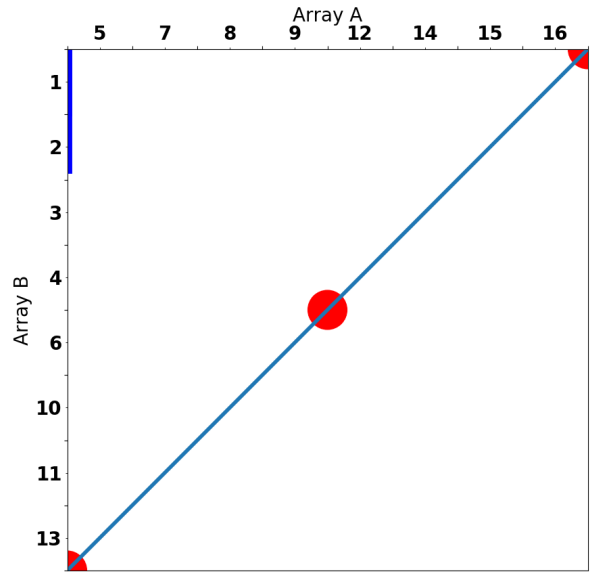
Mergpath: Vertical move: pick array B; horizontal move: Array

Merge

Instead: How to split arrays

NEED TO CITE PAPER!

Mergepath: Define the optimal split: Another Elemet

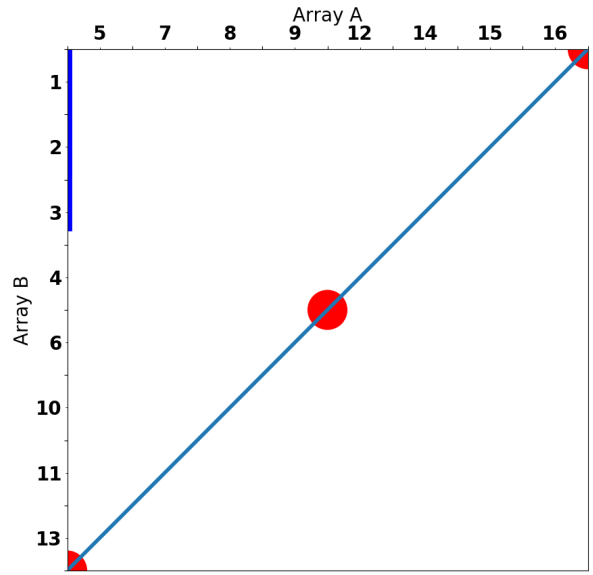


of B

Instead: How to split arrays

NEED TO CITE PAPER!

Mergepath: Define the optimal split: Another Elemet

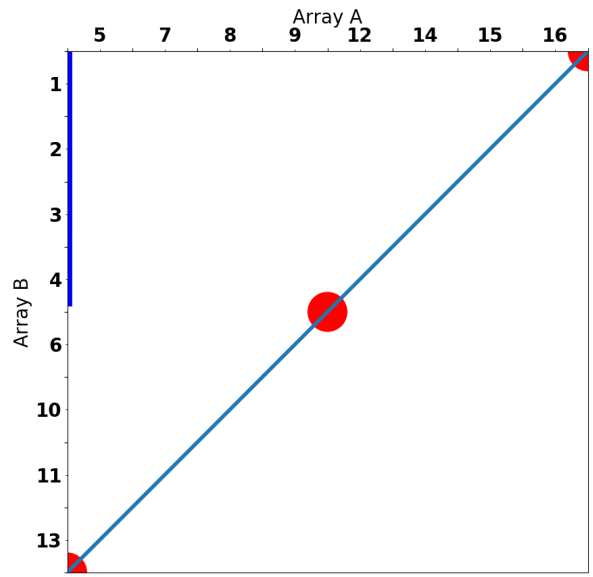


of B

Instead: How to split arrays

NEED TO CITE PAPER!

Mergepath: Define the optimal split: Another Elemet

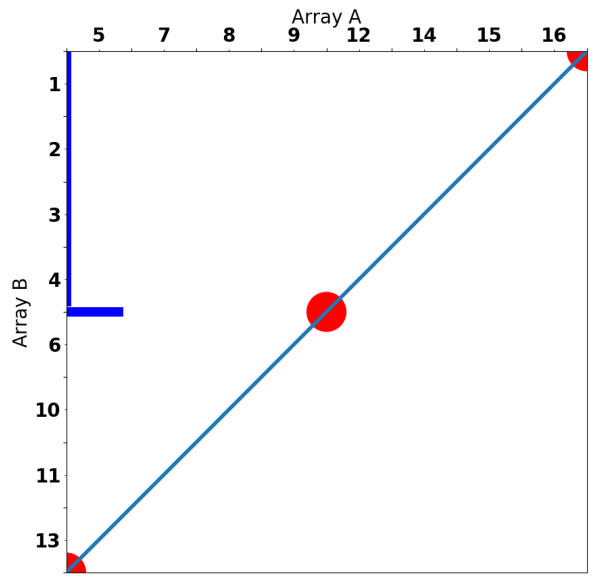


of B

Instead: How to split arrays

NEED TO CITE PAPER!

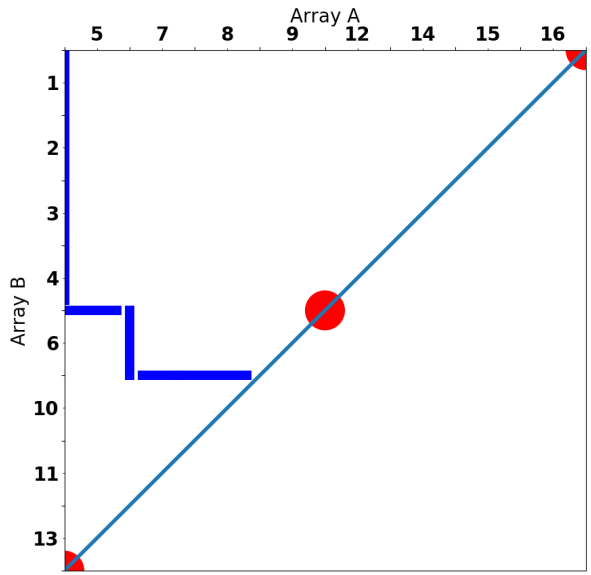
Mergepath: Define the optimal split: Pick first elment



of A

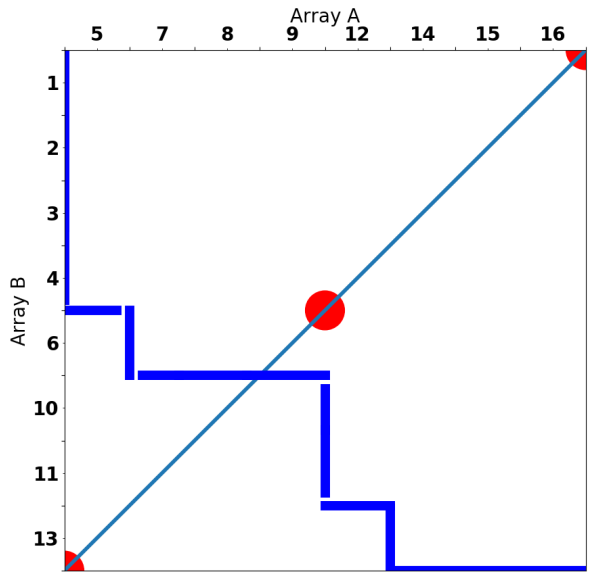
Instead: How to split arrays

NEED TO CITE PAPER!



Instead: How to split arrays

NEED TO CITE PAPER!



Computation Procedure

```
1  __global__
2  void paralleMerge(int* a, int sz_a, int* b, int sz_b,
3                    int* c, int length)
4  {
5      int diag = threadIdx.x * length;
6      int a_start = mergepath(a, sz_a, b, sz_b, diag);
7      int b_start = diag - a_start;
8      merge(a, a_start, sz_a, b, b_start, sz_b, c, diag, leng
9  }
```

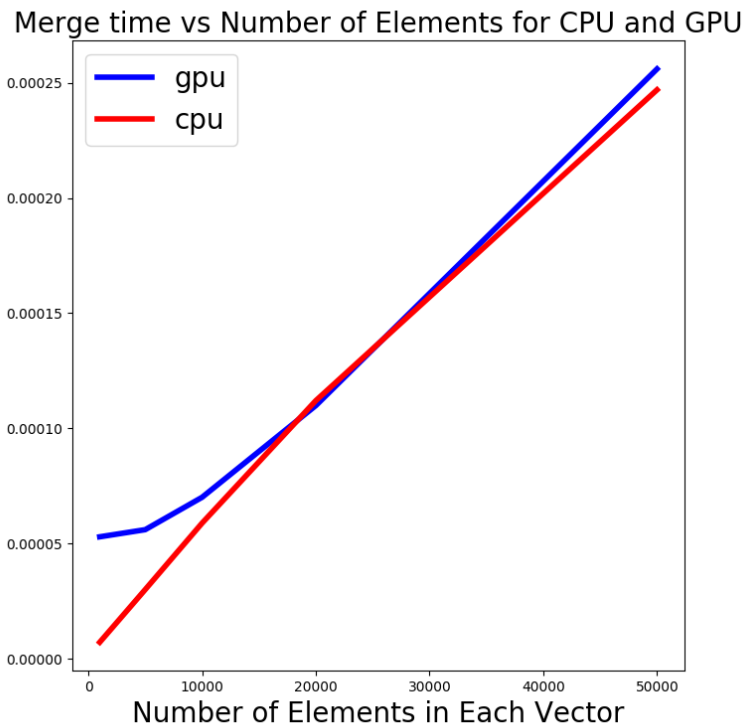
Each tread works on one part

Thread calculates $`A_lower`$ `mergepath`

Obtain `B_lower` as difference

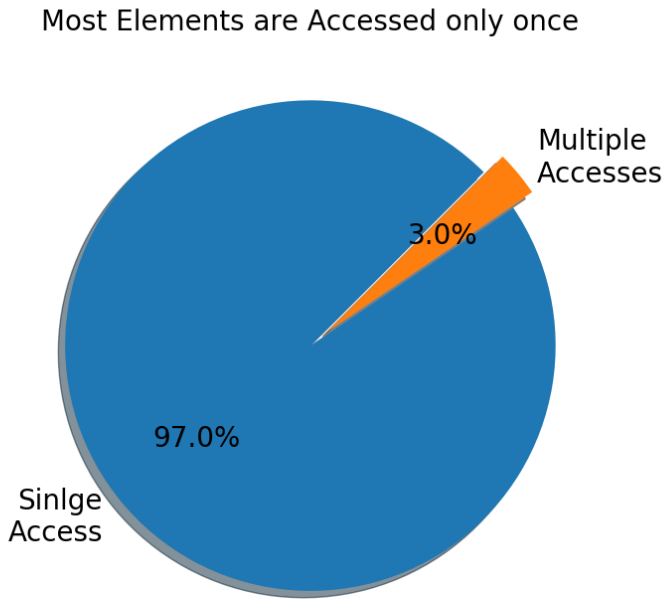
merges two sub-arrays

Problem: Slow as a Snail



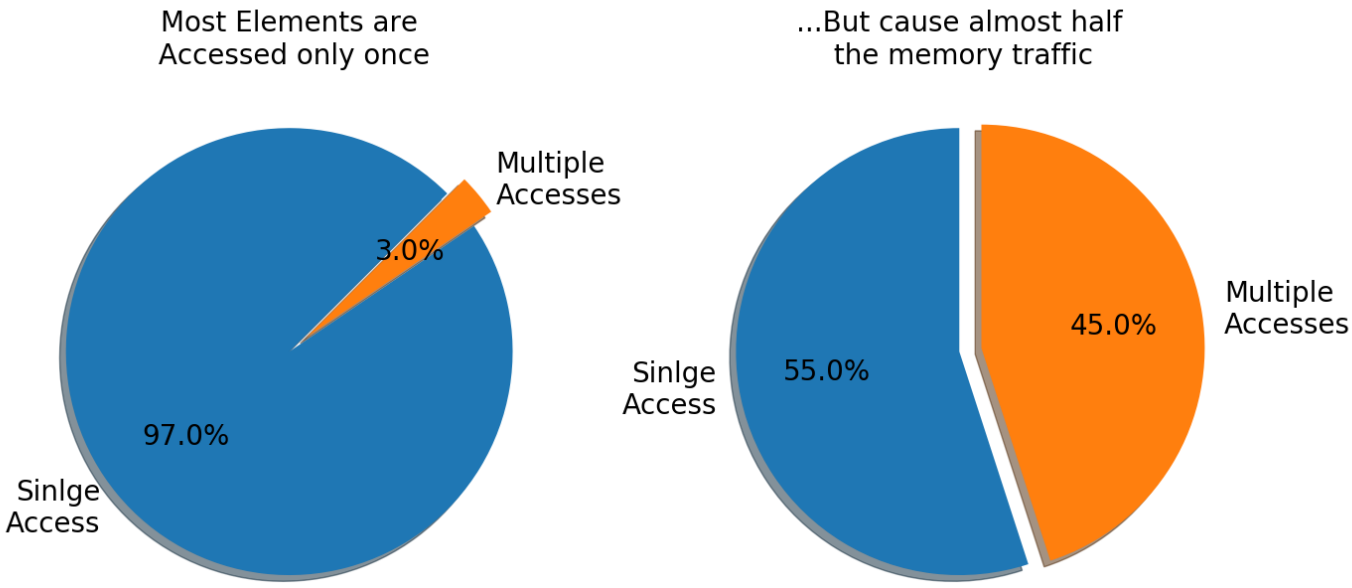
Merge

Reason: So much global meory access



Merge

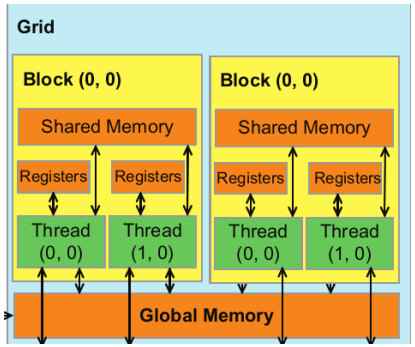
Reason: So much global meory access



Merge

Memory Hirachy of CUDA

The different memories and their sizes



Visibility:

- Global Memory: Accessed by all threads
- Shared Memory: Private for Block
- Registers: Private for Thread

Size:

- Global Memory: 2 GB
- Shared Memory: Total: 192 KB, per block: 48 KB

Latency:

- Global Memory: 8 GB/s
- Shared Memory: 80 GB/s

Merging with local memory

General Idea

Problem: Block shared memory: 48 Kb -> need to make blocks small enough

Solution:

- Determine large but small enough chuncks of arrays to load into blocks
- Load into shared memeory
- Each thread determines its range (as before), from shared memory
- Each thread merges subset (as before)

```
__global__
void paralleMerge(const int* a, int sz_a, const int* b, int
                  int* c, int* boundaries, int length,
                  int size_shared) {
    extern __shared__ int shared[];
    __shared__ int block_ranges[4];
    ranges(block_ranges, sz_a, sz_b, boundaries);
    loadtodevice(a, sz_a, b, sz_b, block_ranges, shared);
    int diag = threadIdx.x * length;
    if (diag < block_ranges[2] + block_ranges[3]) {
        int a_start =
            mergepath(shared, block_ranges[2], &shared[bloc
                      block_ranges[3], diag);
        int b_start = diag - a_start;
        merge(shared, a_start, block_ranges[2], &shared[blo
              b_start, block_ranges[3], c, diag + blockIdx
              length);
    }
}
```

Merging with local memory

Show the results