Mergesort with CUDA

25.02.2020

Architecture

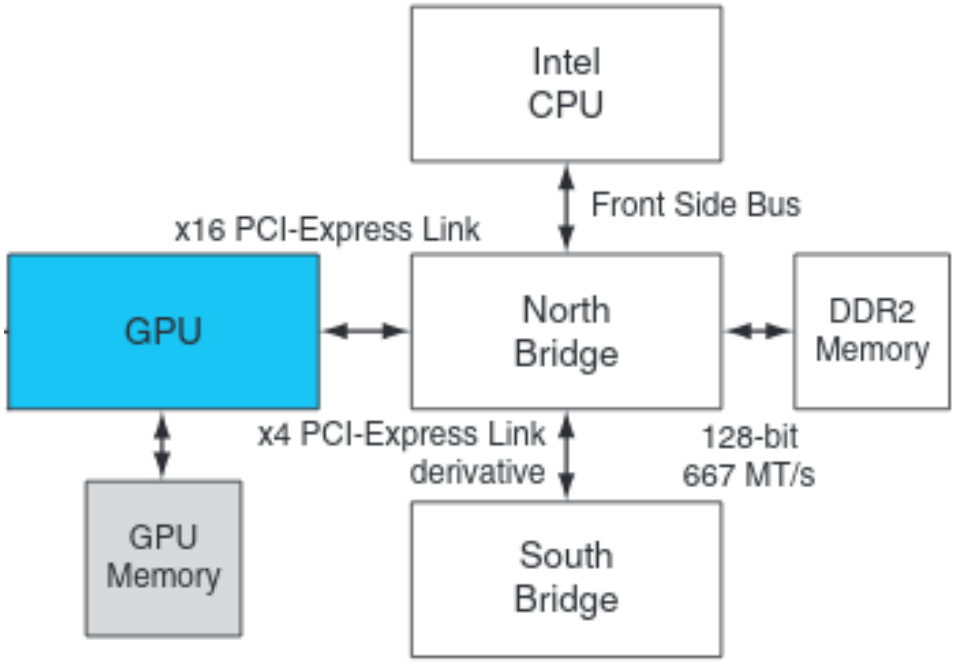
Merge

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`

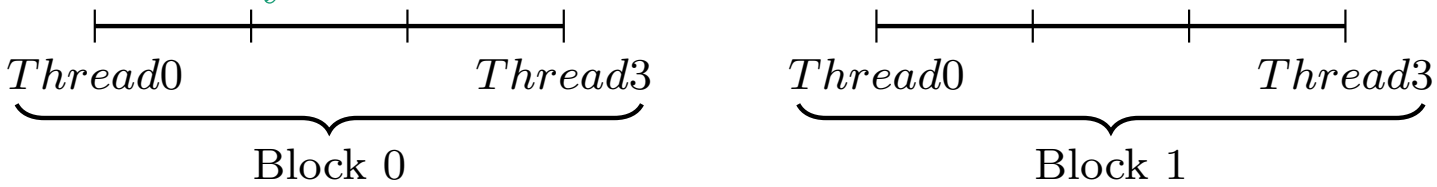
Mapping Cuda calls to Hardware

Launching CUDA threads

Cuda Program

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

Thread Layout:

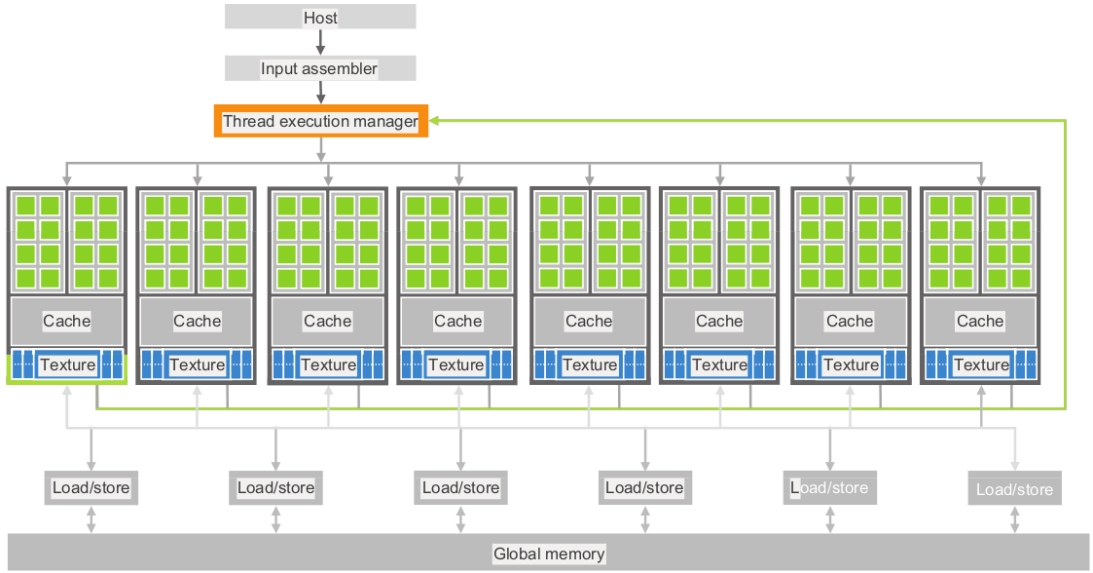


Addition:

```
1  __global__
2  add_kernel(float* A, float* B, float* C, int n) {
3      int i = blockDim.x * blockIdx.x + threadIdx.x;
4      if (i < n) {
5          C[i] = A[i] + B[i];
6      }
7  }
```

Software/Hardware connection

Programmer: Specifies number of blocks and threads

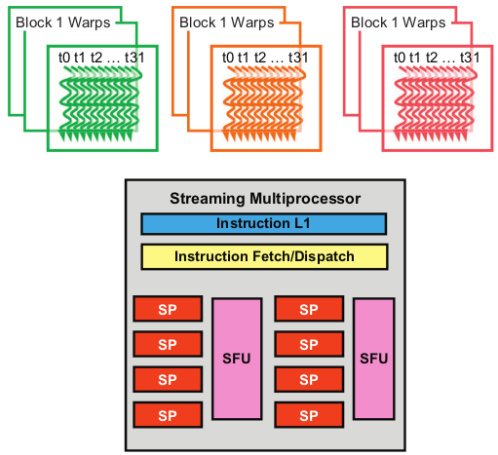


Dispatch:

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

Software/Hardware connection

Programmer: Specifies number of blocks and threads

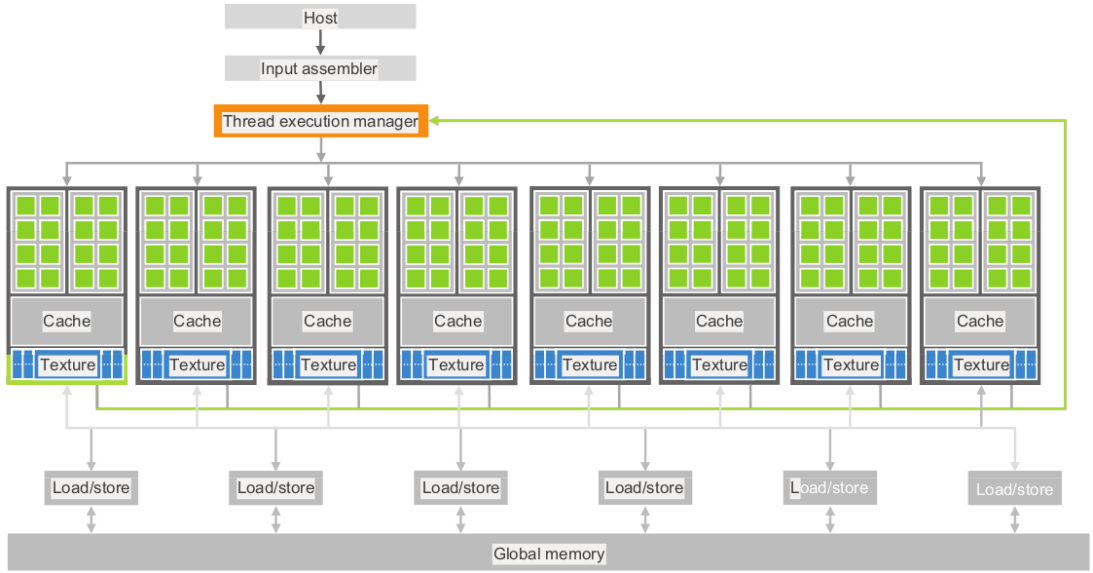


Dispatch:

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:

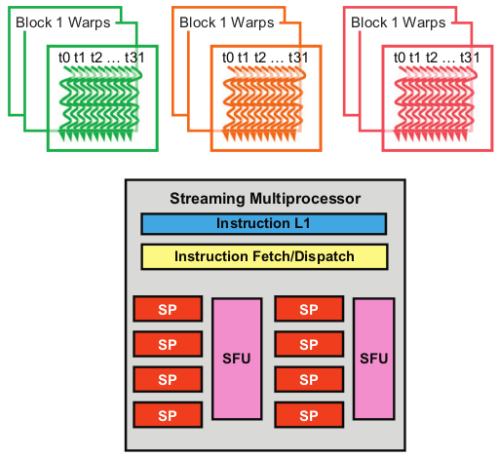
1. Blocks to SM: Block 1 → SM1, Block 2 → SM2 ...
2. Grouping: Wrappend threads are allocated

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

$$A = 578912141516$$

$$B = 12346101113$$

$$C = ????????????????$$

```
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 }
```

How to spwan to many threads?

Naive: 2 Threads, half A and B

Example:

$$A = 0000$$

$$B = 1111$$

$$C = ????????$$

$$A = \underbrace{00}_{\text{Thread 1}} \mid \underbrace{00}_{\text{Thread 2}}$$

$$B = \underbrace{11}_{\text{Thread 1}} \mid \underbrace{11}_{\text{Thread 2}}$$

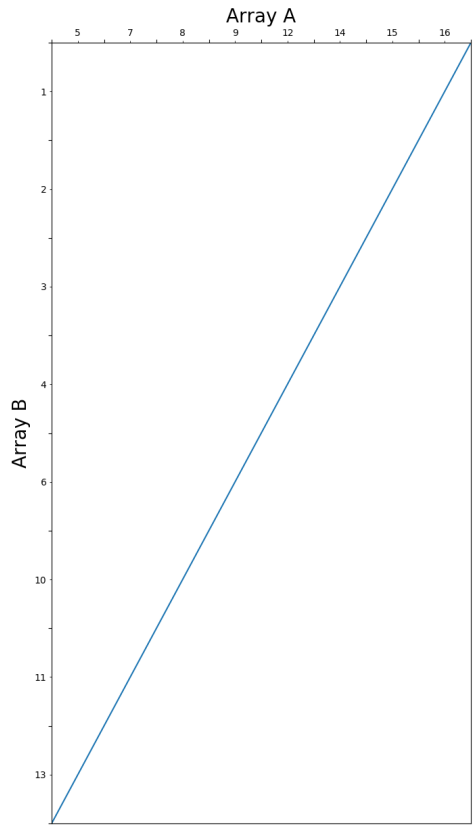
$$C = \underbrace{????}_{\text{Thread 1}} \mid \underbrace{????}_{\text{Thread 2}}$$

Result:

$$C = \underbrace{0011}_{\text{Thread 1}} \mid \underbrace{0011}_{\text{Thread 2}}$$

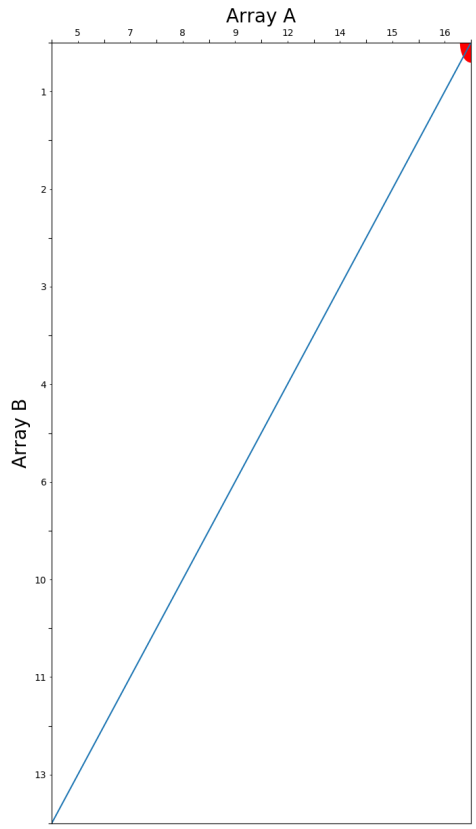
Merge

How to allocated work?



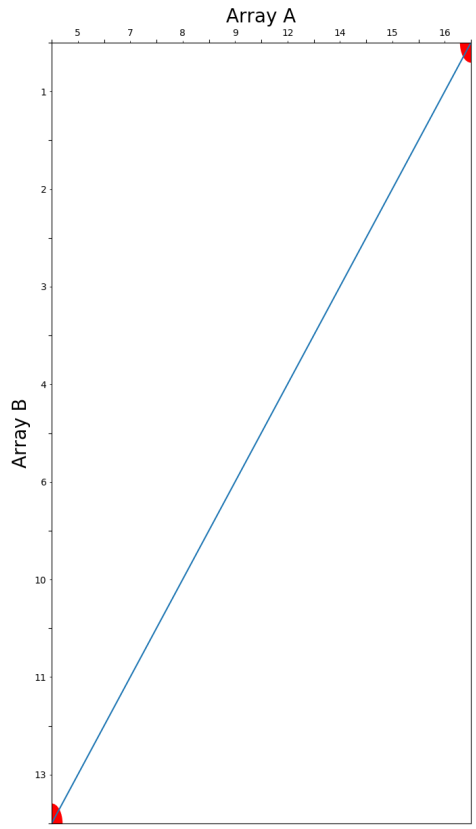
Merge

How to allocated work?



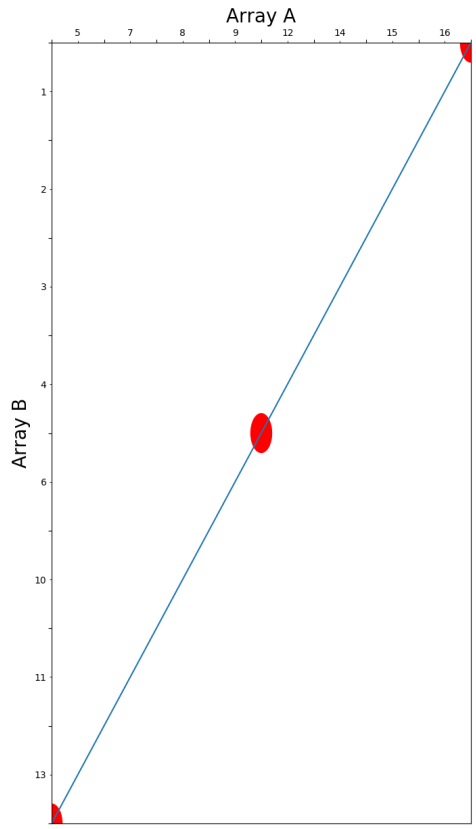
Merge

How to allocated work?



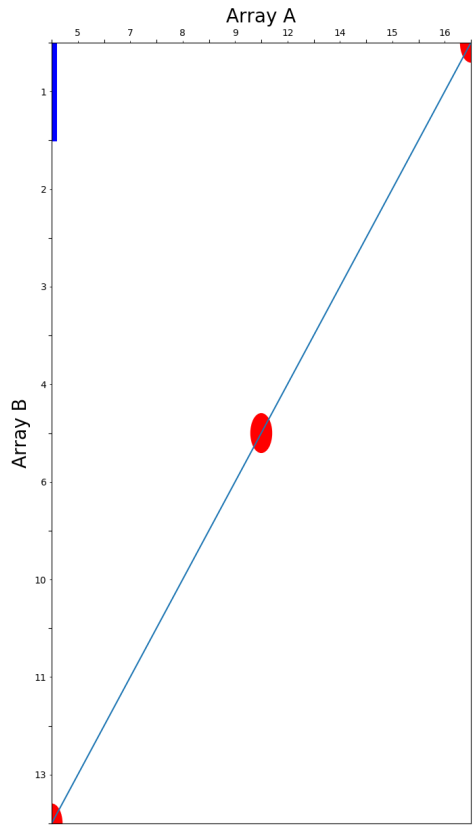
Merge

How to allocated work?



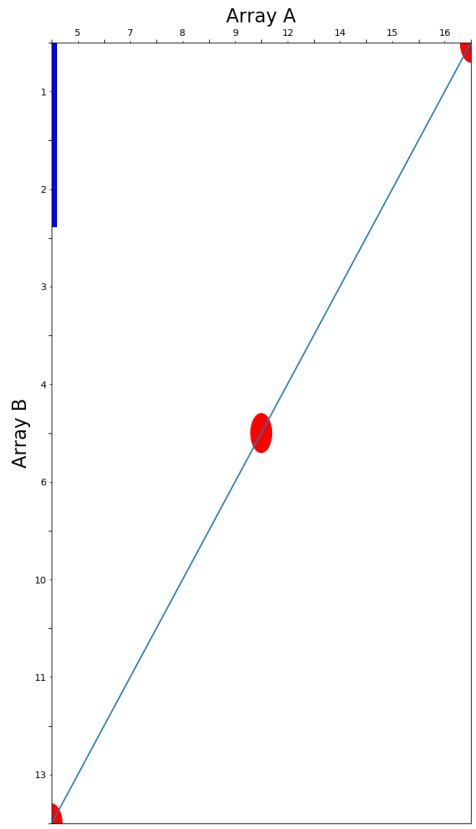
Merge

How to allocated work?



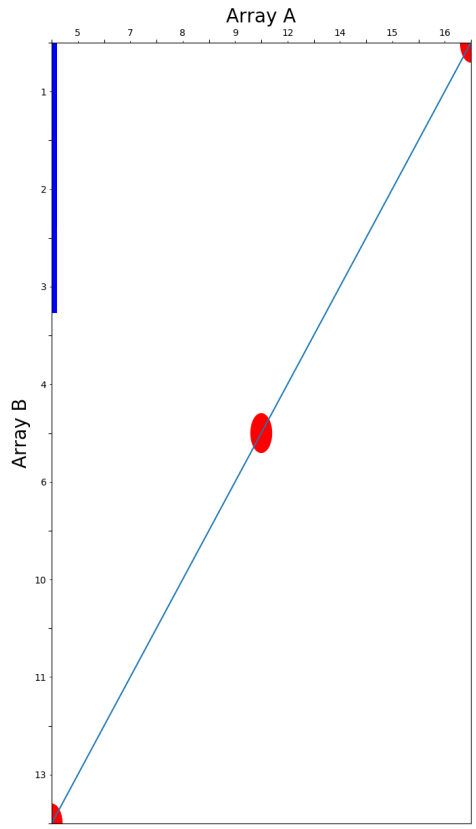
Merge

How to allocated work?



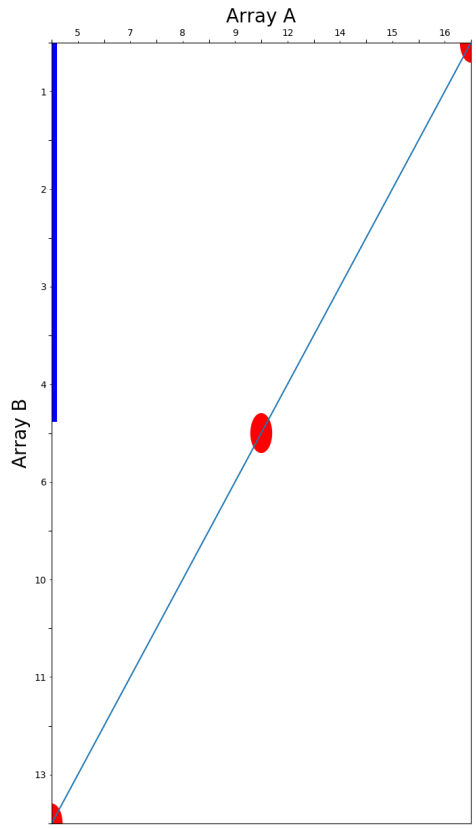
Merge

How to allocated work?



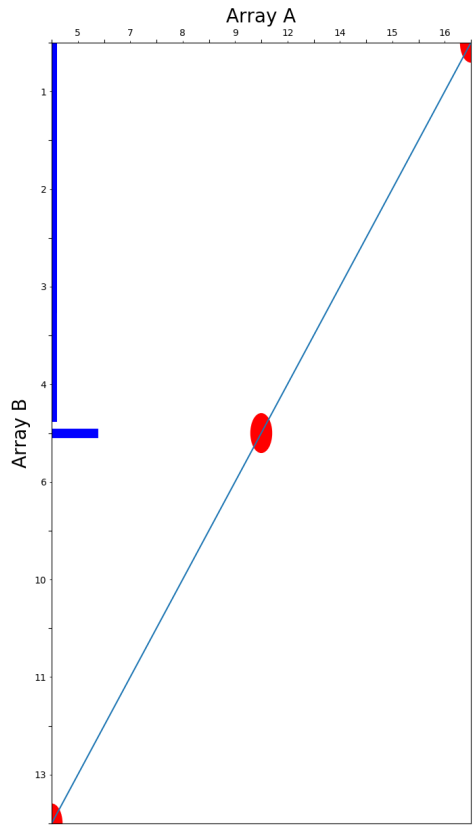
Merge

How to allocated work?



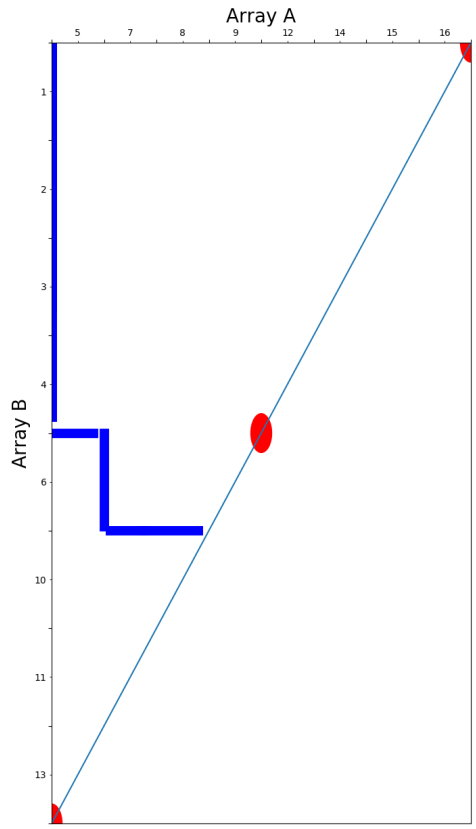
Merge

How to allocated work?



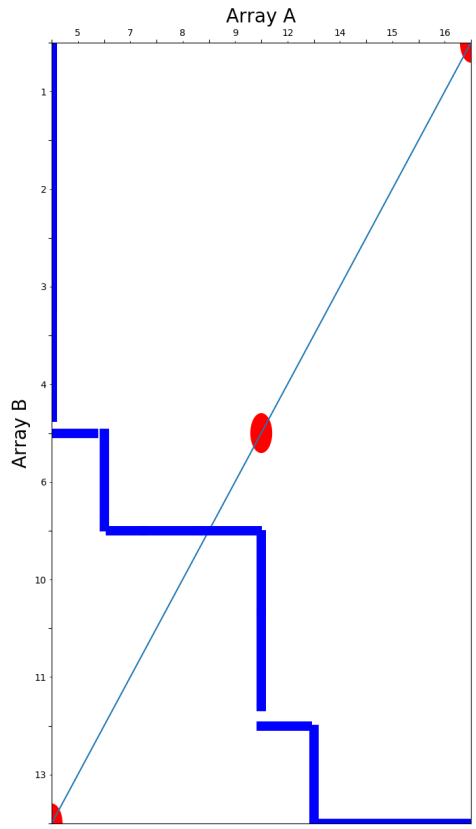
Merge

How to allocated work?



Merge

How to allocated work?



Merge

Comutation 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  }
```

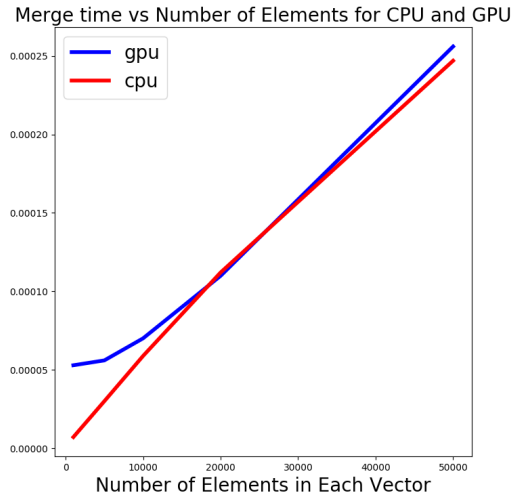
Each tread works on one part

Thread calculates the value A_lower for itself

Calcultes the also the B_lower (Why does that work again?)

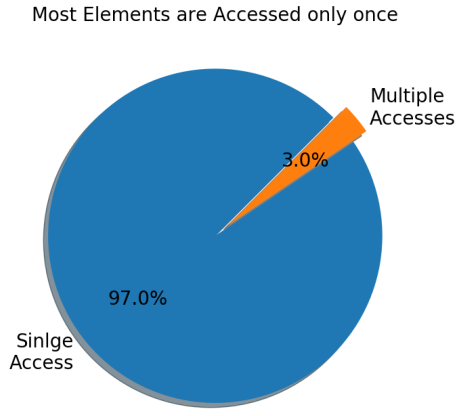
merges the two arrays

Problem: Slow as a Snail



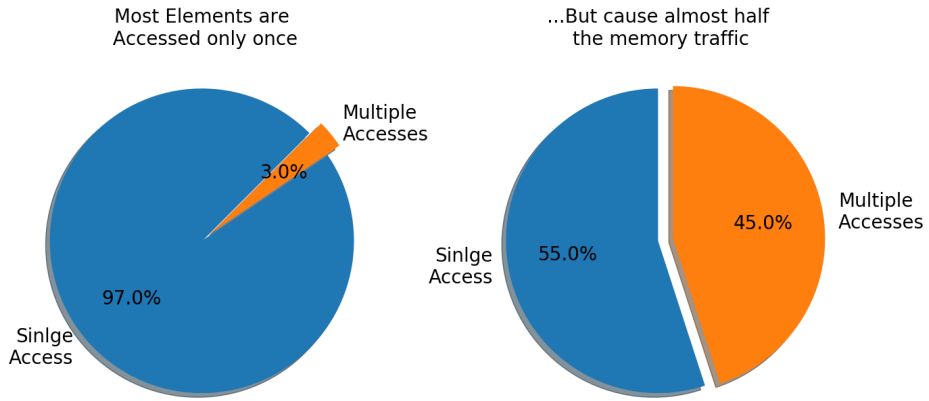
Merge

Reason: So much global meory access



Merge

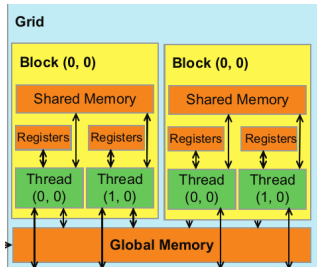
Reason: So much global meory access



Merge

Memory Hirachy of CUDA

The different memories and their sizes



Visibility:

Global Memory: Can be accessed by all threads

Shared Memory: Private for each Block

Registers: Private for each 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

Describe the shared memory

show the plot of the different memories and their relative size on my card

Show the results