

# CUB—CUDA UNBOUND

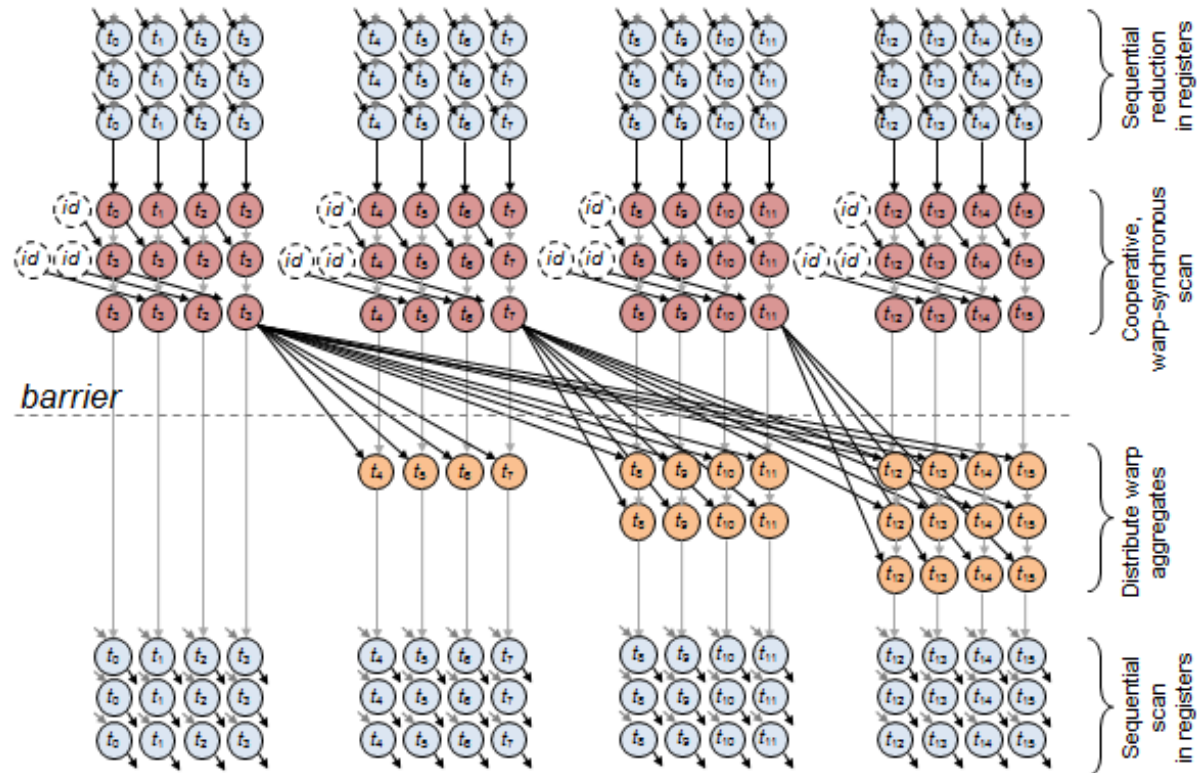
25. APRIL 2018 | JAN H. MEINKE

# WHAT IS CUB?

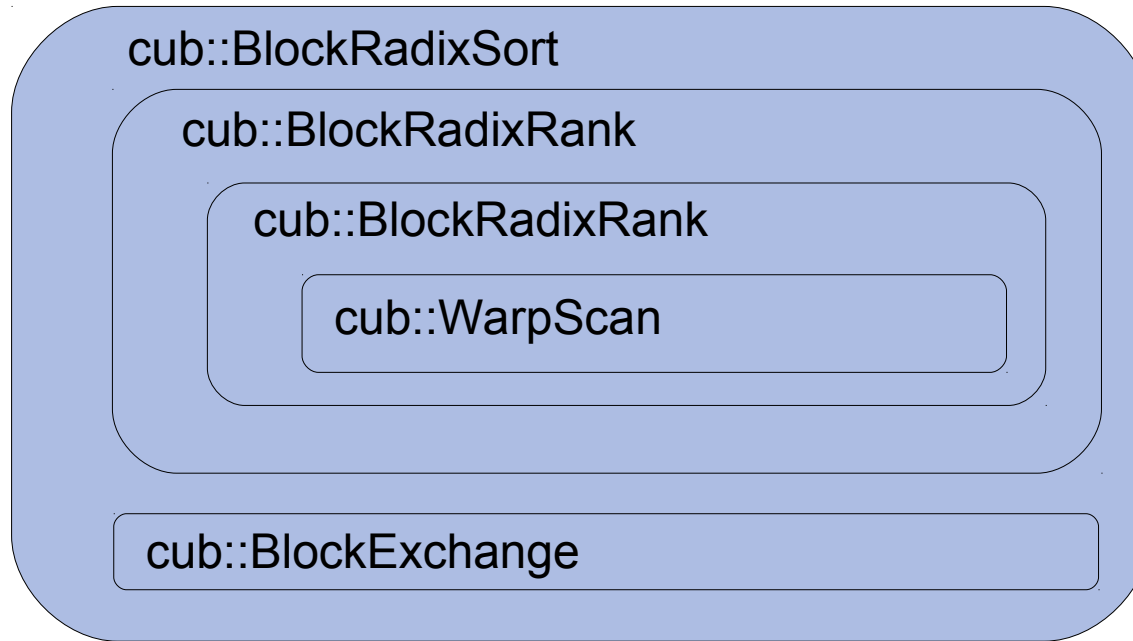
- A design model for collective kernel-level primitives
  - How do I write collective primitives?
  - How do I deal with memory?
  - How do I make them tunable?
- A library of collective primitives
  - BlockLoad, BlockReduce, BlockRadixSort, ...
- A library of global primitives
  - DeviceReduce, DeviceHistogram, DeviceRadixSort, ...

(c.f. Duane Merrill's talk at GTC)

# COLLECTIVE PARALLEL PROGRAMMING IS HARD

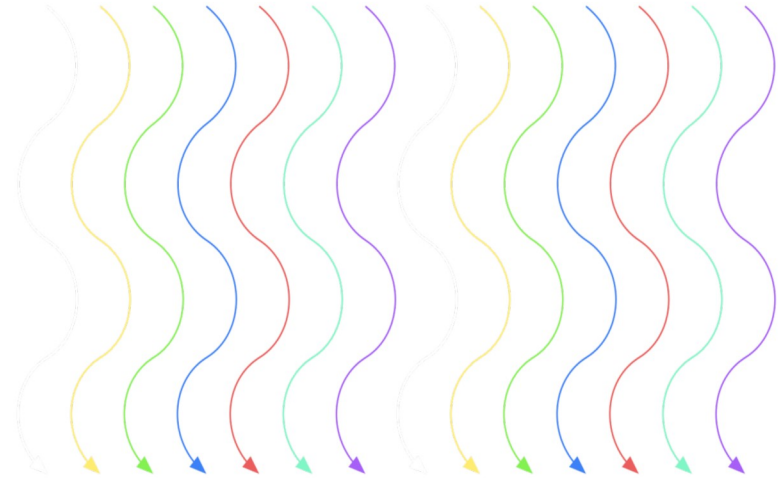


# REDUCE! REUSE! RECYCLE!



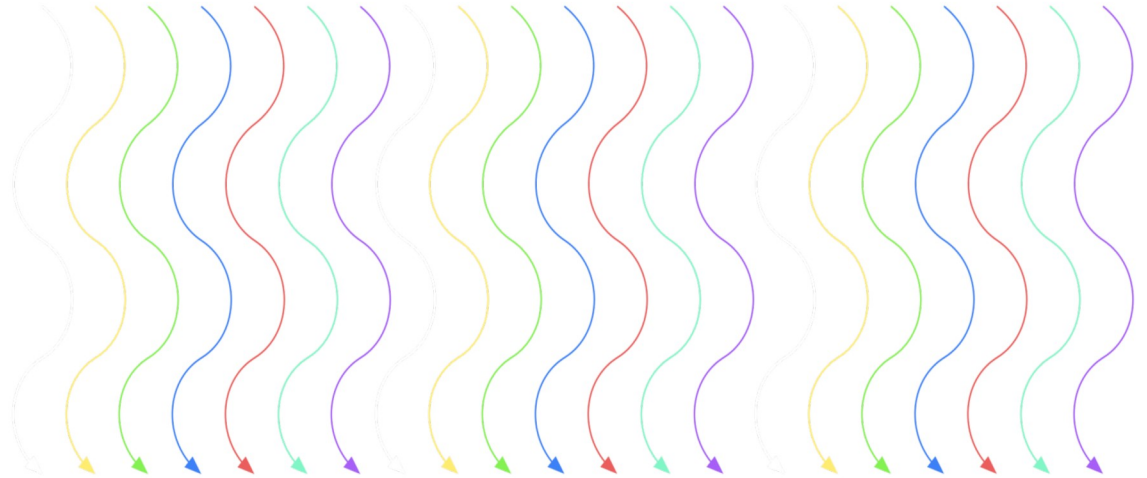
# MAKE IT TUNABLE

- Adjust parallelism
- Adjust grain size



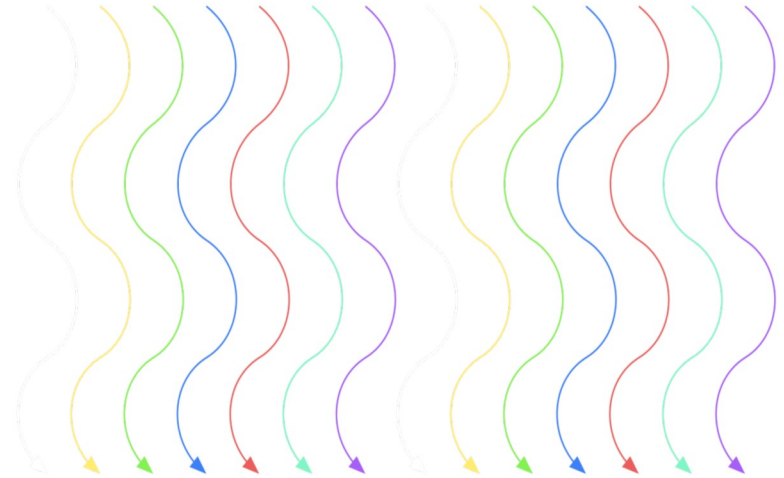
# MAKE IT TUNABLE

- Adjust parallelism
- Adjust grain size



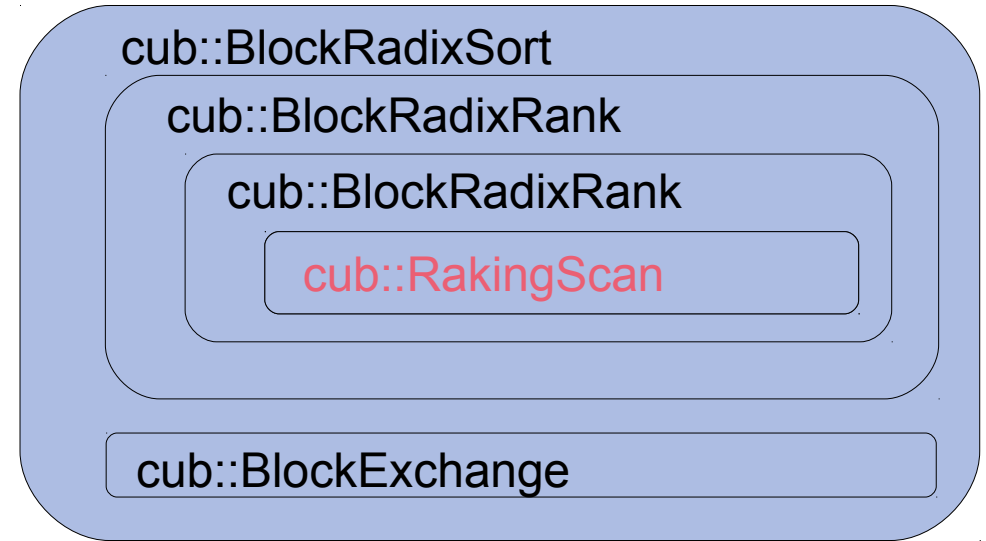
# MAKE IT TUNABLE

- Adjust parallelism
- Adjust grain size



# SWAP OUT COMPONENTS

- Replace inner algorithm easily
- Performance depends on GPU
- Performance depends on the rest of the kernel





# AN EXAMPLE

```
#include <cub/cub.cuh>

__global__ void ExampleKernel(...) {

    // Specialize the template for double precision and 128 threads
    typedef cub::BlockReduce<double, 128> BlockReduceT;

    ...
}
```

# AN EXAMPLE

```
#include <cub/cub.cuh>

__global__ void ExampleKernel(...) {

    // Specialize the template for double precision and 128 threads
    typedef cub::BlockReduce<double, 128> BlockReduceT;

    // Declare shared storage
    __shared__ typename BlockReduceT::TempStorage temp_storage;

    double items[4];

    ...
}
```

# AN EXAMPLE

```
#include <cub/cub.cuh>

__global__ void ExampleKernel(...) {

    // Specialize the template for double precision and 128 threads
    typedef cub::BlockReduce<double, 128> BlockReduceT;

    // Declare shared storage
    __shared__ typename BlockReduceT::TempStorage temp_storage;

    double items[4];

    // Instantiate an instance of BlockReduceT
    double result = BlockReduceT(temp_storage).Sum(items);

    ...
}
```

# AN EXAMPLE

```
__global__ void ExampleKernel(const double* in, double* out){  
  
    // Specialize the template for double precision and 128 threads w/ 4 items per thread  
    typedef cub::BlockLoad<double, 128, 4> BlockLoadT;  
    // Specialize the template for double precision and 128 threads  
    typedef cub::BlockReduce<double, 128> BlockReduceT;  
    // Declare shared storage  
  
    __shared__ union {  
        typename BlockLoadT::TempStorage load;  
        typename BlockReduceT::TempStorage reduce;  
    } temp_storage;  
  
    double items[4];  
  
    BlockLoadT(temp_storage.load).Load(in, items);  
    __syncthreads();  
  
    // Instantiate an instance of BlockReduceT  
    double result = BlockReduceT(temp_storage.reduce).Sum(items);  
  
    if (threadIdx.x == 0) *out = result;  
}
```

```
int main(){  
    ...  
    ExampleKernel<<<1, 128>>>>(d_gpu, result_gpu);  
    ...  
}
```

# AN EXAMPLE

```
template <typename T>
__global__ void ExampleKernel(const T* in, T* out){

    // Specialize the template for double precision and 128 threads w/ 4 items per thread
    typedef cub::BlockLoad<T, 1024, 4> BlockLoadT;
    // Specialize the template for double precision and 128 threads
    typedef cub::BlockReduce<T, 1024> BlockReduceT;
    // Declare shared storage

    __shared__ union {
        typename BlockLoadT::TempStorage load;
        typename BlockReduceT::TempStorage reduce;
    } temp_storage;

    T items[4];

    BlockLoadT(temp_storage.load).Load(in, items);
    __syncthreads();

    // Instantiate an instance of BlockReduceT
    T result = BlockReduceT(temp_storage.reduce).Sum(items);

    if (threadIdx.x == 0) *out = result;
}
```

```
int main(){
    ...
    ExampleKernel<<<1, 1024>>>>(d_gpu, result_gpu);
    ...
}
```

# AN EXAMPLE

```
template <int BLOCK_THREADS, int ITEMS_PER_THREAD, typename T>
__global__ void ExampleKernel(const T* in, T* out){

    // Specialize the template for double precision and BLOCK_THREADS threads w/ ITEMS_PER_THREAD items per thread
    typedef cub::BlockLoad<T, BLOCK_THREADS, ITEMS_PER_THREAD> BlockLoadT;
    // Specialize the template for double precision and BLOCK_THREADS threads
    typedef cub::BlockReduce<T, BLOCK_THREADS> BlockReduceT;
    // Declare shared storage

    __shared__ union {
        typename BlockLoadT::TempStorage load;
        typename BlockReduceT::TempStorage reduce;
    } temp_storage;

    T items[ITEMS_PER_THREAD];

    BlockLoadT(temp_storage.load).Load(in, items);
    __syncthreads();

    // Instantiate an instance of BlockReduceT
    T result = BlockReduceT(temp_storage.reduce).Sum(items);

    if (threadIdx.x == 0) *out = result;
}
```

```
int main(){
    ...
    ExampleKernel<1024, 4><<<1, 1024>>>>(d_gpu,
    result_gpu);
    ...
}
```

# AN EXAMPLE

```
template <int BLOCK_THREADS, int ITEMS_PER_THREAD, cub::BlockLoadAlgorithm LOAD_ALGO,
         cub::BlockReduceAlgorithm REDUCE_ALGO, typename T>
__global__ void ExampleKernel(const T* in, T* out){

    // Specialize the template for double precision and BLOCK_THREADS threads w/ ITEMS_PER_THREAD items per thread
    typedef cub::BlockLoad<T, BLOCK_THREADS, ITEMS_PER_THREAD, LOAD_ALGO> BlockLoadT;
    // Specialize the template for double precision and BLOCK_THREADS threads
    typedef cub::BlockReduce<T, BLOCK_THREADS> BlockReduceT;
    // Declare shared storage
    __shared__ union {
        typename BlockLoadT::TempStorage load;
        typename BlockReduceT::TempStorage reduce;
    } temp_storage;

    T items[ITEMS_PER_THREAD];

    BlockLoadT(temp_storage.load).Load(in, items);
    __syncthreads();

    // Instantiate an instance of BlockReduceT
    T result = BlockReduceT(temp_storage.reduce).Sum(items);
    if (threadIdx.x == 0) *out = result;
}
```

```
int main(){
    ...
    ExampleKernel<1024, 4,
                  cub::BLOCK_LOAD_TRANSPOSE,
                  cub::BLOCK_REDUCE_RAKING>
                  <<<1, 1024>>>(d_gpu, result_gpu);
    ...
}
```

# RESOURCES

- Duane Merrill's talks at GTC. Go to <http://on-demand-gtc.gputechconf.com/> and search for “Duane Merrill” or “CUB”.
- The CUB web page at <http://nvlabs.github.io/cub/>.