Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

CUB – CUDA unbound

April 27, 2016 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

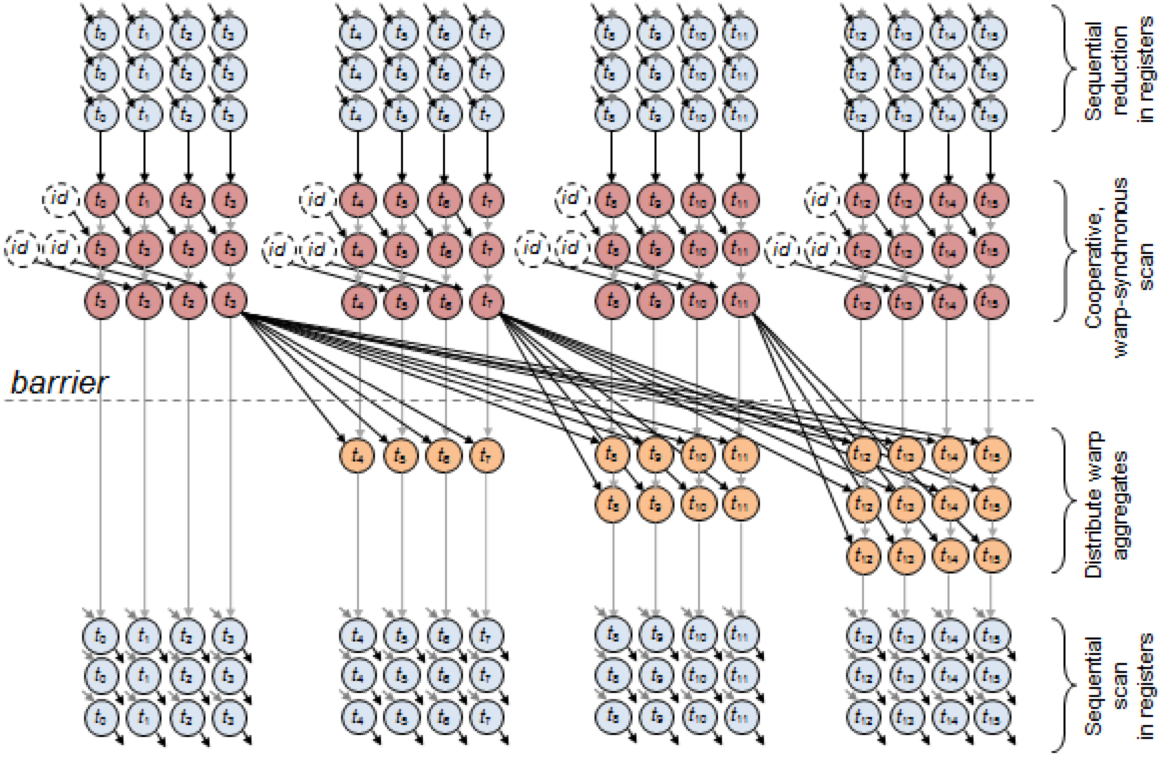
  BlockLoad, BlockReduce, BlockRadixSort, …

- A library of global primitives

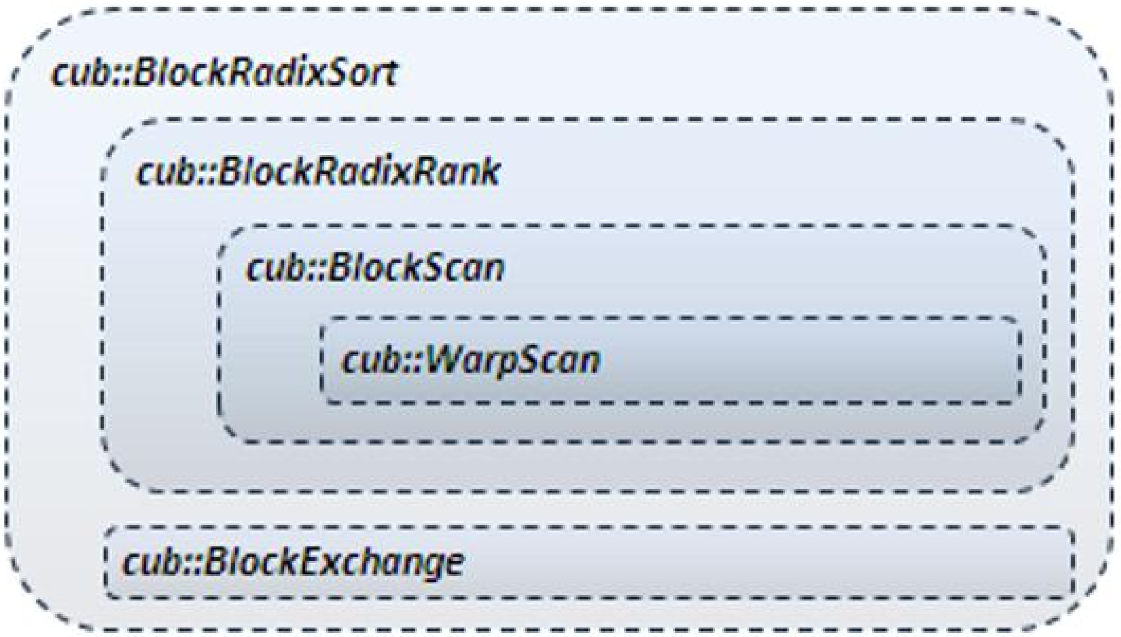
  DeviceReduce, DeviceHistogram, DeviceRadixSort, ...

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

Collective Parallel Programming is Hard

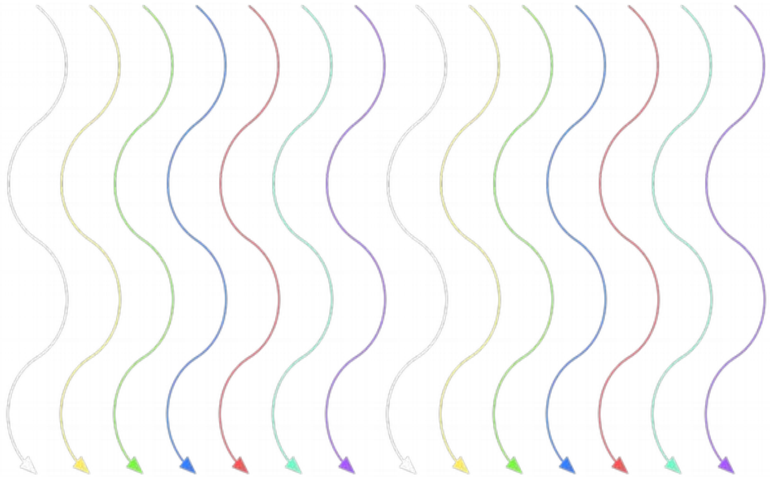

Sequential reduction in registers
Cooperative, warp-synchronous scan
barrier
Distribute warp aggregates
Sequential scan in registers
id
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15}

Reduce! Reuse! Recycle!



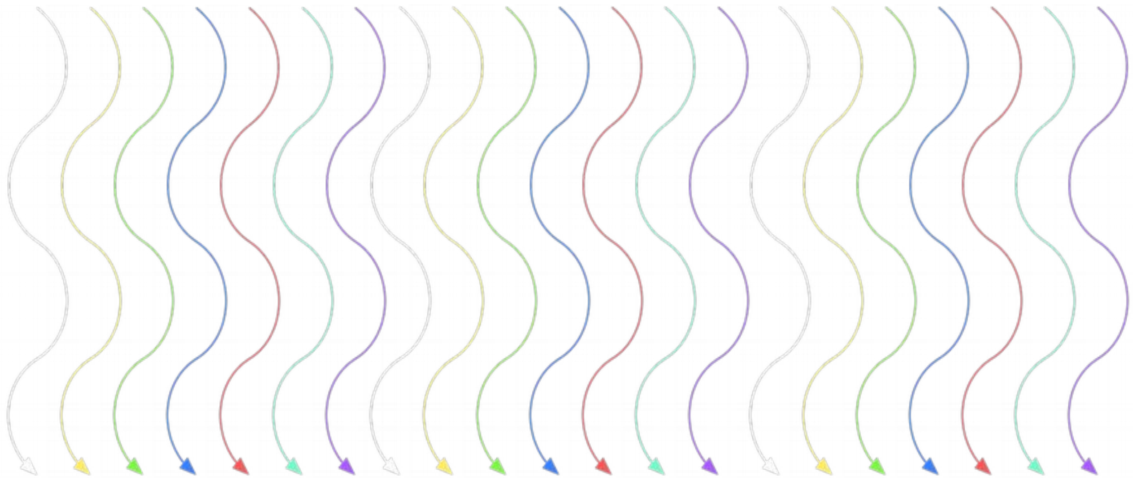
Make It Tunable

- Adjust parallelism
- Adjust grain size



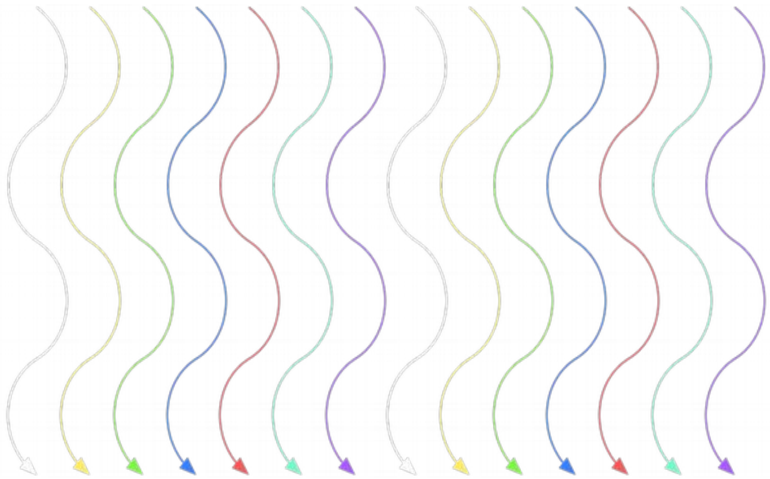
Make It Tunable

- Adjust parallelism
- Adjust grain size



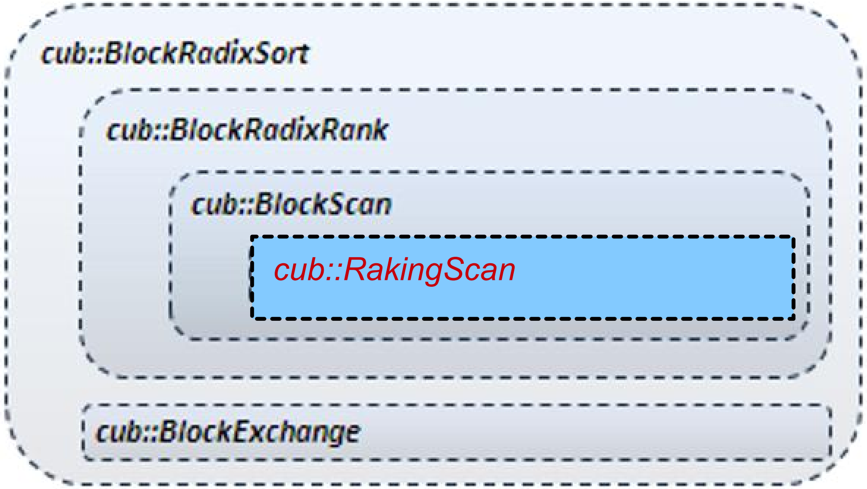
Make It Tunable

- 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<const 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<const 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<const 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/.