CUB—CUDA UNBOUND

5 JUNE 2024 | JAN H. MEINKE

JÜLICH
Forschungszentrum
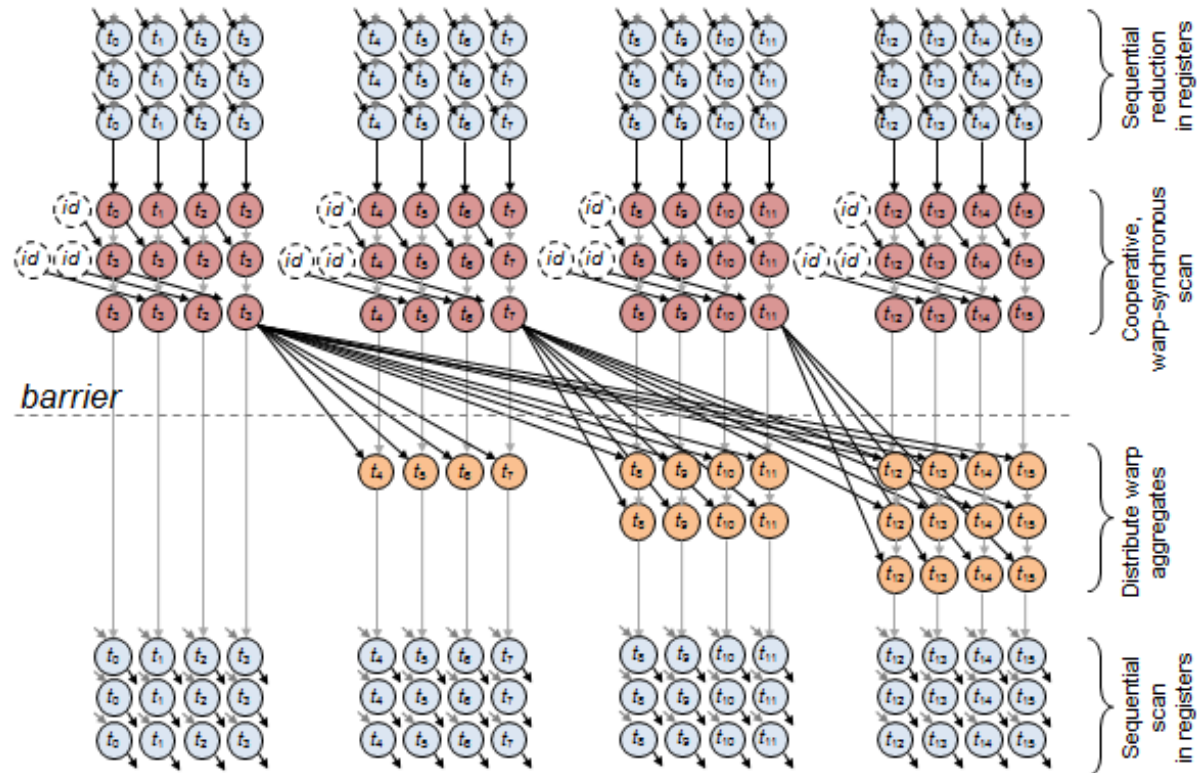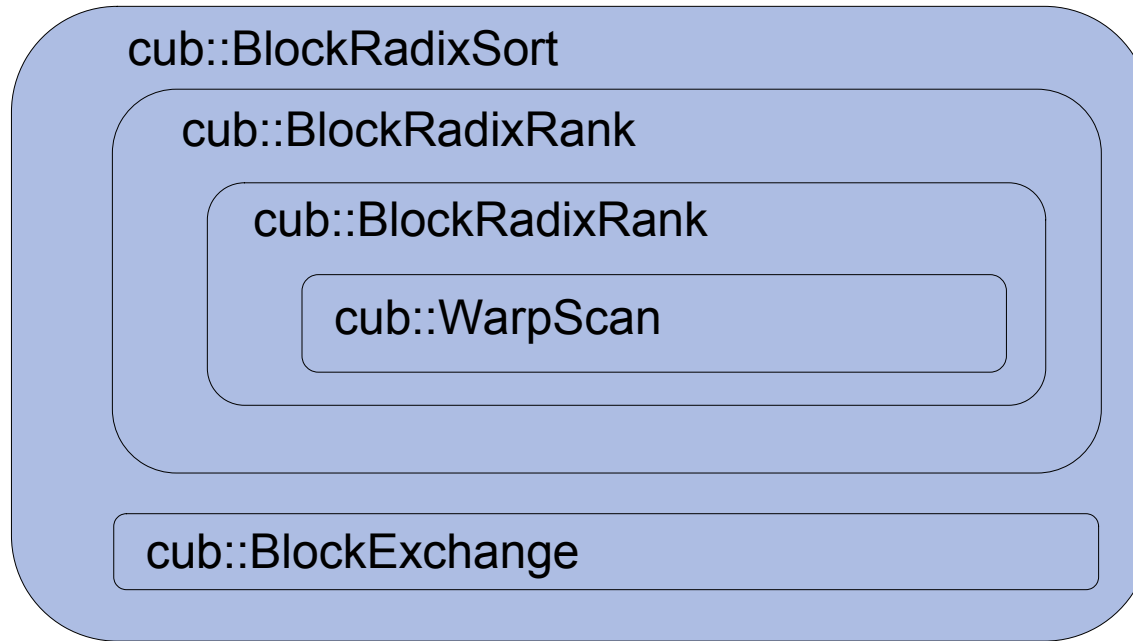
WHAT IS CUB?

- 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)

JÜLICH
Forschungszentrum

COLLECTIVE PARALLEL PROGRAMMING IS HARD

REDUCE! REUSE! RECYCLE!

cub::BlockRadixSort
  cub::BlockRadixRank
    cub::BlockRadixRank
      cub::WarpScan

  cub::BlockExchange

JÜLICH
Forschungszentrum

MAKE IT TUNABLE

- Adjust parallelism
- Adjust grain size

JÜLICH
Forschungszentrum
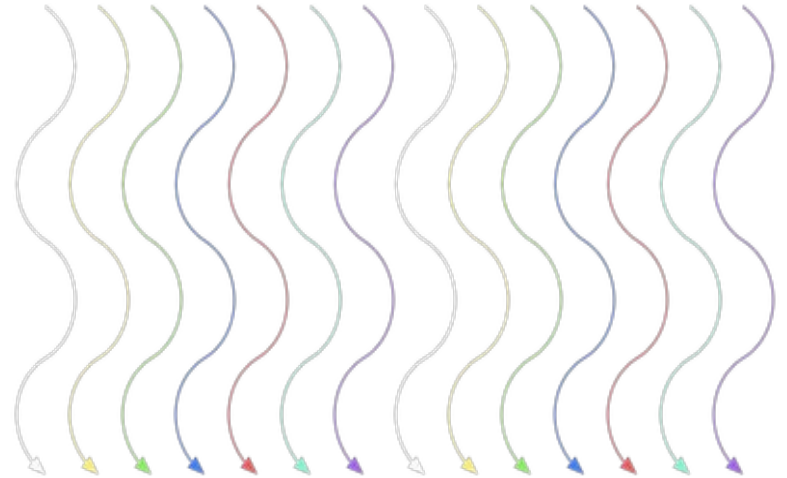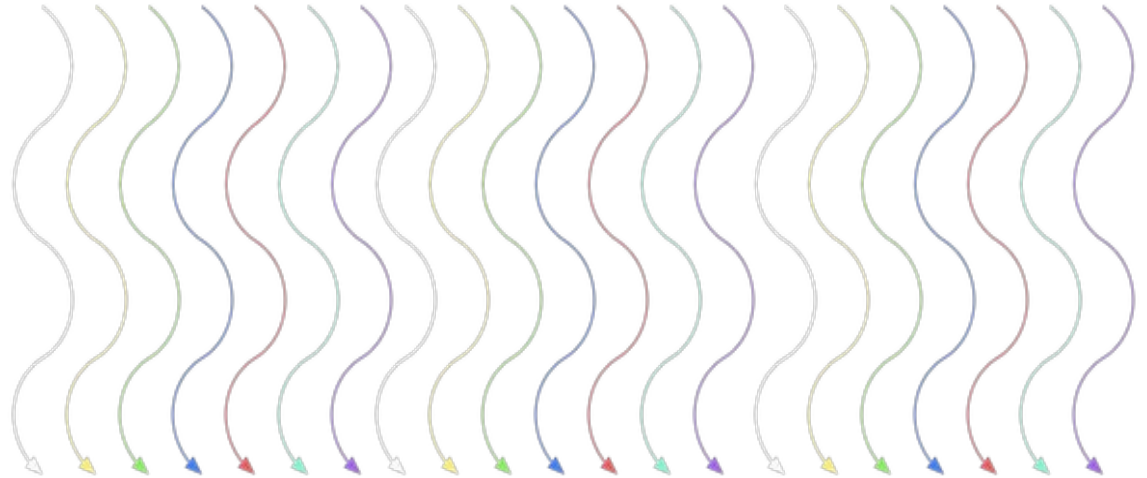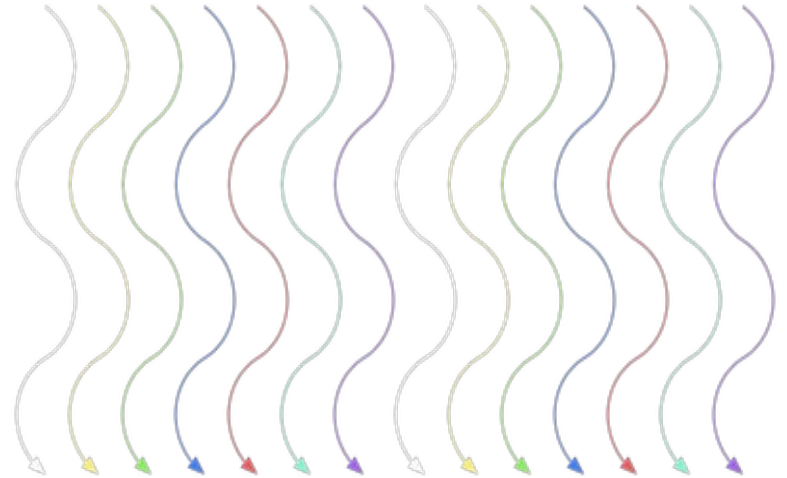
MAKE IT TUNABLE

- Adjust parallelism
- Adjust grain size

MAKE IT TUNABLE

- Adjust parallelism
- Adjust grain size

JÜLICH
Forschungszentrum

SWAP OUT COMPONENTS

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

cub::BlockRadixSort

cub::BlockRadixRank

cub::BlockRadixRank

cub::RakingScan

cub::BlockExchange

JÜLICH
Forschungszentrum

AN EXAMPLE

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

__global__ void ExampleKernel(...){

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


    ...
}
```

JÜLICH
Forschungszentrum

AN EXAMPLE

```cpp
#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];

    ...
}
```

JÜLICH
Forschungszentrum

AN EXAMPLE

```cpp
#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);

    ...
}
```

JÜLICH
Forschungszentrum

AN EXAMPLE

```cpp
__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;
}
```

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

JÜLICH
Forschungszentrum

AN EXAMPLE

```cpp
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;
}
```

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

JÜLICH
Forschungszentrum

AN EXAMPLE

```cpp
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;
}
```

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

JÜLICH
Forschungszentrum

Mitglied der Helmholtz-Gemeinschaft

AN EXAMPLE

```cpp
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;
}
```

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

JÜLICH
Forschungszentrum

Mitglied der Helmholtz-Gemeinschaft

RESOURCES

- Duane Merrill's talk at GTC
  https://on-demand.gputechconf.com/gtc/2014/presentations/S4566-cub-collective-software-primitives.pdf

- The CUB web page at http://nvlabs.github.io/cub/.

- CUB is part of the CUDA C++ Core Libraries.

JÜLICH
Forschungszentrum