Awkward + CCCL

Presentation by: Maksym Naumchyk

Supervisor: Ianna Osborne

Thanks to  team for help and support!

How does it work?

CUDA kernels

```
template <typename T, typename C, typename U>
__global__ void
awkward_reduce_sum_a(
    T* toptr,
    const C* fromptr,
    const U* parents,
    int64_t lenparents,
    int64_t outlength,
    T* temp,
    uint64_t invocation_index,
    uint64_t* err_code) {
  if (err_code[0] == NO_ERROR) {
    int64_t thread_id = blockIdx.x * blockDim.x + threadIdx.x;

    if (thread_id < outlength) {
      toptr[thread_id] = 0;
    }
  }
}
```

```
template <typename T, typename C, typename U>
__global__ void
awkward_reduce_sum_b(
    T* toptr,
    const C* fromptr,
    const U* parents,
    int64_t lenparents,
    int64_t outlength,
    T* temp,
    uint64_t invocation_index,
    uint64_t* err_code) {
  if (err_code[0] == NO_ERROR) {
    int64_t idx = threadIdx.x;
    int64_t thread_id = blockIdx.x * blockDim.x + idx;

    if (thread_id < lenparents) {
      temp[thread_id] = fromptr[thread_id];
    }
    __syncthreads();
```

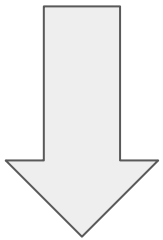
and more …

The whole sum() operation using CUDA kernels:
https://github.com/scikit-hep/awkward/blob/main/src/awkward/_connect/cuda/cuda_kernels/awkward_reduce_sum.cu

CCCL

We can feed ListOffsetArrays directly to the CCCL algorithms:

```
<ListOffsetArray len='6'>
    <offsets><Index dtype='int64' len='7'>
        [0 1 3 5 8 8 9]
    </Index></offsets>
    <content><NumpyArray dtype='int64' len='9'>[1 2 3 4 5 6 7 8 9]</NumpyArray></content>
</ListOffsetArray>
```



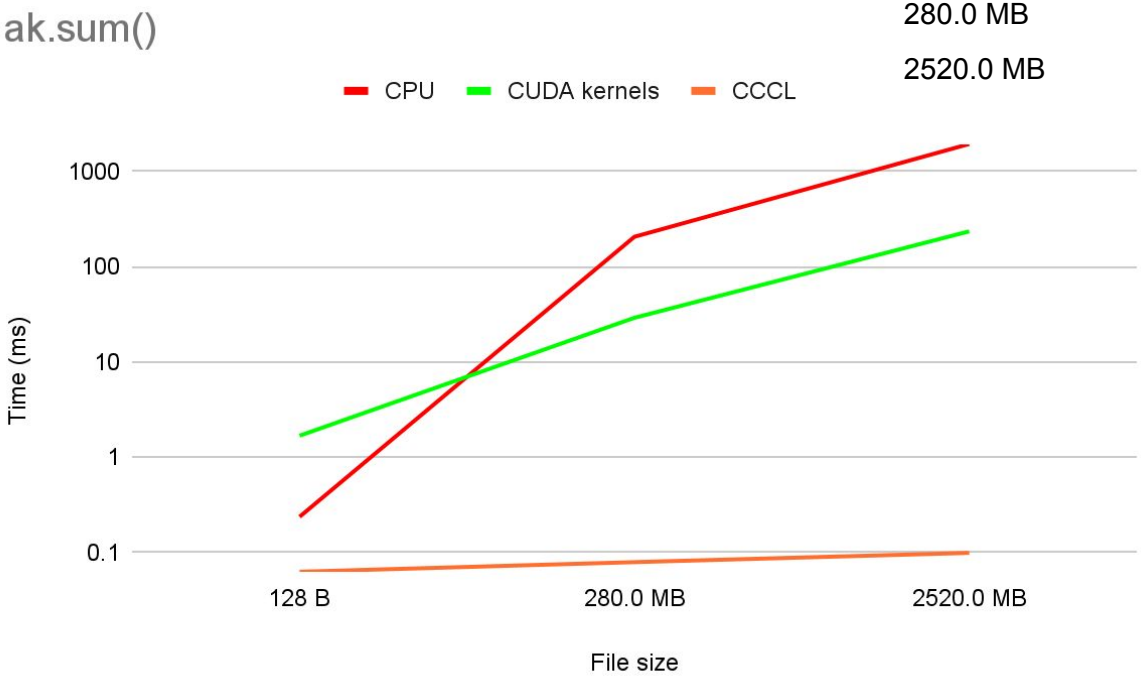
```
segmented_reduce(
    input_data, output, start_offsets, end_offsets, sum_operation, h_initial, n_segments
)
```

CCCL intro tutorial: https://github.com/maxymnaumchyk/AGC-Demo-Day-11/blob/main/code/CCCL%20introduction.ipynb

Why do we need this?

Performance improvement!

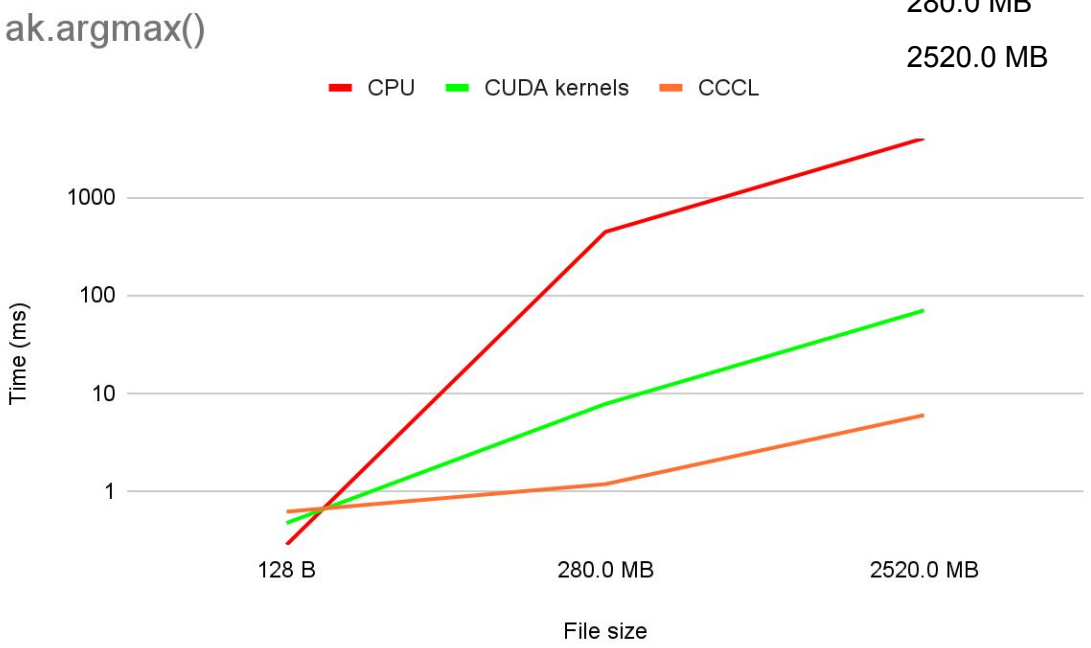
ak.sum() File size	CPU	CUDA kernels	CCCL
128 B	0.238 ms	1.69 ms	0.063 ms
280.0 MB	207 ms	29.2 ms	0.0795 ms
2520.0 MB	1950 ms	236 ms	0.0994 ms



Jupyter notebook for sum(): https://github.com/scikit-hep/awkward/pull/3762

Performance improvement!

ak.argmax() File size	CPU	CUDA kernels	CCCL (using unary_transform)
128 B	0.284 ms	0.474 ms	0.618 ms
280.0 MB	442 ms	7.77 ms	1.18 ms
2520.0 MB	4000 ms	70.0 ms	5.97 ms



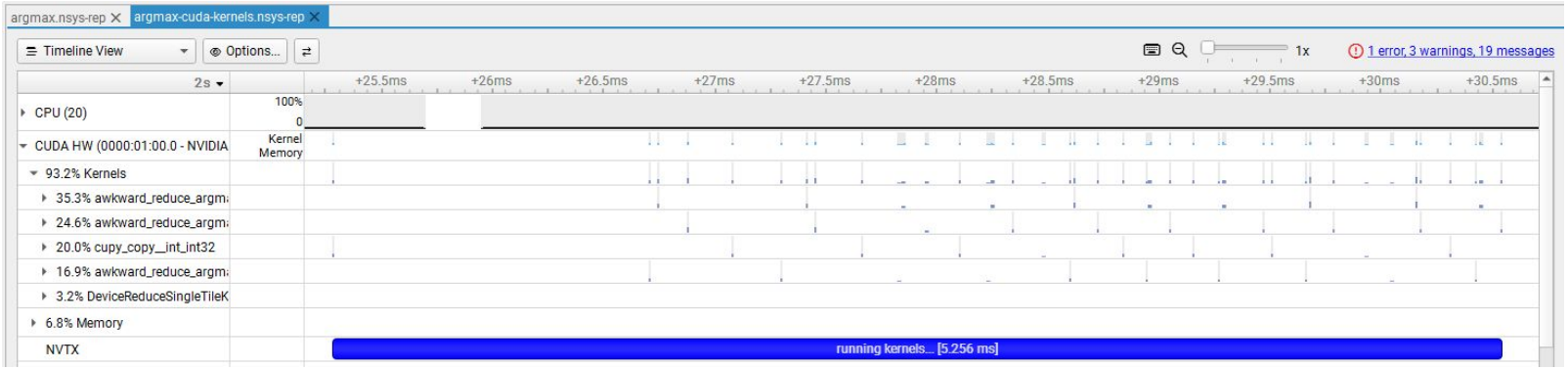
Even more improvements!

ak.sort() with cccl

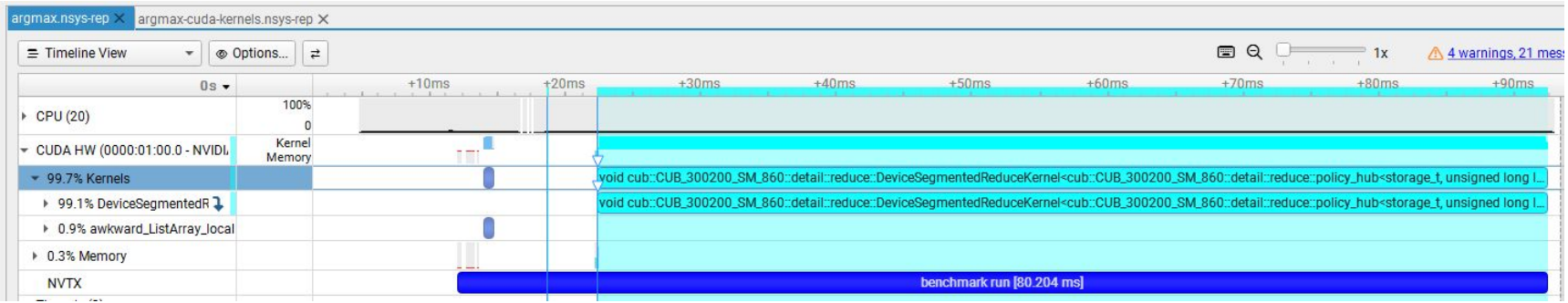
ak.combinations() using cp.searchsorted

Python script for argmax(): https://github.com/scikit-hep/awkward/pull/3800

Kernels fusion



A lot of small CUDA kernels (previously with awkward) vs →



one big kernel fused with CCCL

Going a step further

Introducing (new) kernels

```
# select only electrons with pt > 40
selected_electrons = events.electrons[events.electrons.pt > 40.0]
```

```
def cond_muon(x):
    return (x[0] > 20.0) & (abs(x[1]) < 2.4)
selected_muons = filter_lists(events.muons, cond_muon)
```

```
def filter_lists(array, cond):
    it, meta = awkward_to_cccl_iterator(array)
    in_segments = meta["offsets"]
    out_array = empty_like(array)
    it_out, meta_out = awkward_to_cccl_iterator(out_array)
    out_segments = meta_out["offsets"]
    num_items = meta["count"]
    segmented_select(
        it,
        in_segments,
        it_out,
        out_segments,
        cond,
        num_items
    )
    return reconstruct_with_offsets(out_array, out_segments)
```

LHC data analysis playground by Peter: https://github.com/scikit-hep/awkward/pull/3734/files

Playground using CCCL by Ashwin: https://github.com/scikit-hep/awkward/blob/main/studies/cccl/playground.py

[Prototype] Intermediate Representation (IR)

```
selected_muons = filter_lists(events.muons, cond_muon)
two_muons = select_lists(
    selected_muons, (list_sizes(selected_muons) == 2).astype('int8'))
```

```
class IsEqual(IRNode):
    def lower(self, a, b):
        return a == b
selected_muons = Filter(events.muons, cond_muon)
two_muons = SelectLists(
    selected_muons, (IsEqual(ListSizes(selected_muons),2)))
```

Playground using CCCL and IR> https://github.com/scikit-hep/awkward/pull/3792

[Prototype] IR wrapper

```
arr = ak.Array([[1, 2, 3], [4, 5], [6, 7, 8, 9]], backend="cuda")

# Create lazy wrapper
lazy_arr = ak.cuda.lazy(arr)

# Build computation graph without executing
transformed = lazy_arr * 2 + 1
result = transformed.filter(lazy_arr > 3)

# Execute the computation
print("Computed result:")
print(result.compute())
```

```
Computed result:
[[], [9, 11], [13, 15, 17, 19]]
```

Github issue on this prototype: https://github.com/scikit-hep/awkward/pull/3792
Feel free to discuss it there!

Thank *you* for attention!

Useful links:
NVIDIA CCCL: https://nvidia.github.io/cccl/python/
Ashwin's presentation on CCCL for HEP: https://indico.cern.ch/event/1566263/contributions/6733091/
[youtube recording]