Extending Awkward Functions to GPU

Manasvi Goyal [1,2], Jim Pivarski [2], Ianna Osborne [2]

[1] Harvard University *[2] Princeton University*

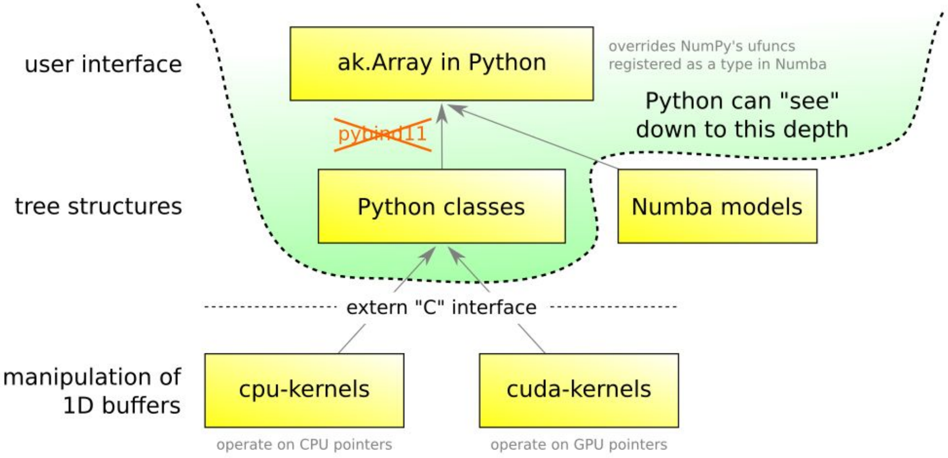
Support for this work was provided by NSF cooperative agreements OAC-1836650 and PHY-2323298 (IRIS-HEP) and OAC-2103945 (Awkward Array).

Implementing CUDA Kernels

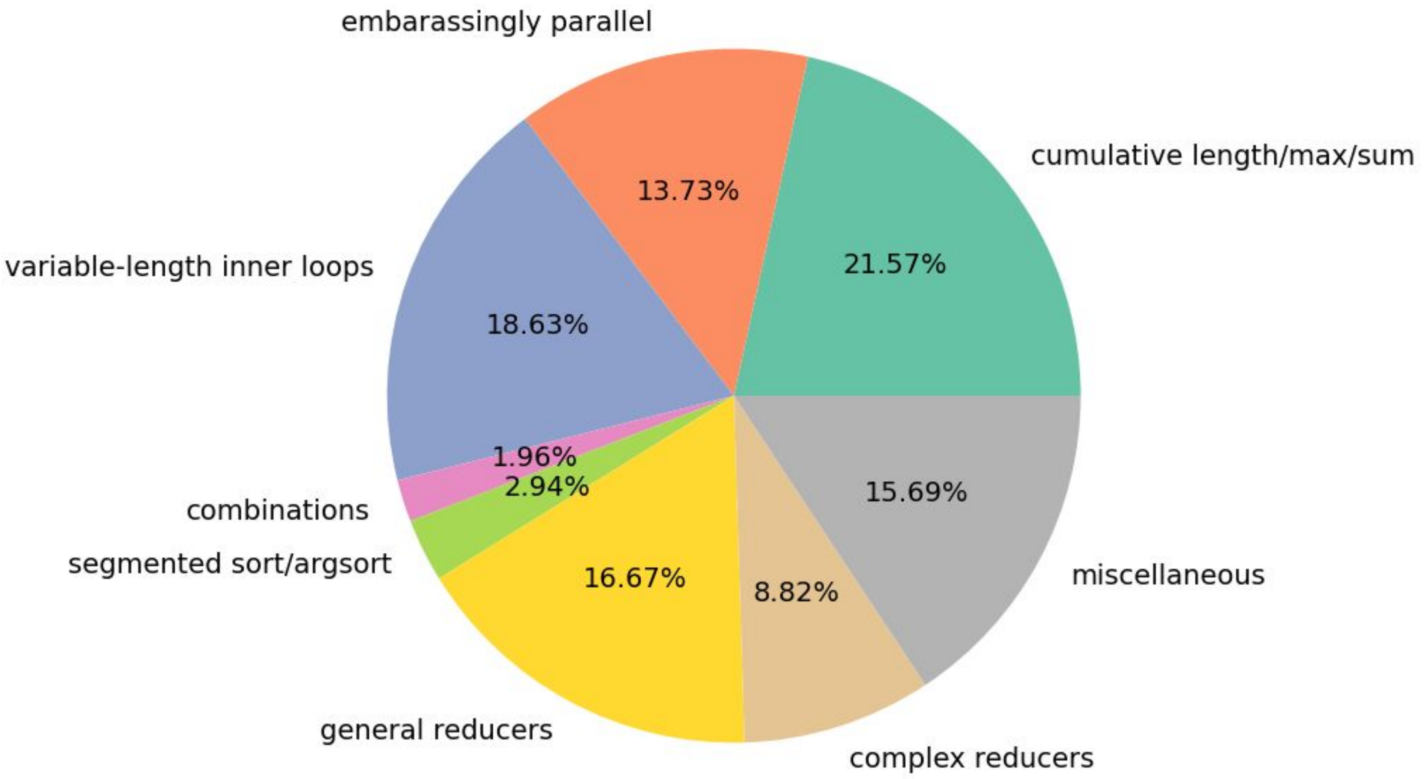
- Infrastructure connecting CUDA kernels to Awkward operations already set up.
- CuPy is used to handle the higher level functions.
- Removed obsolete kernels leftover from Awkward 1.x.
- Overall 144 CPU kernels.
- 42 of the embarrassingly parallel kernels already automatically converted.
- Convert remaining trickier 102 CPU-bound algorithms into CUDA algorithms.
- Fixing errors in existing CUDA kernels.

Awkward 2.x Architecture

- Due to this, awkward has no direct dependency on cuda
- Introduces an indirection as we move from the upper to the lower layers.
- Awkward functions can be used on GPU with just pip install awkward

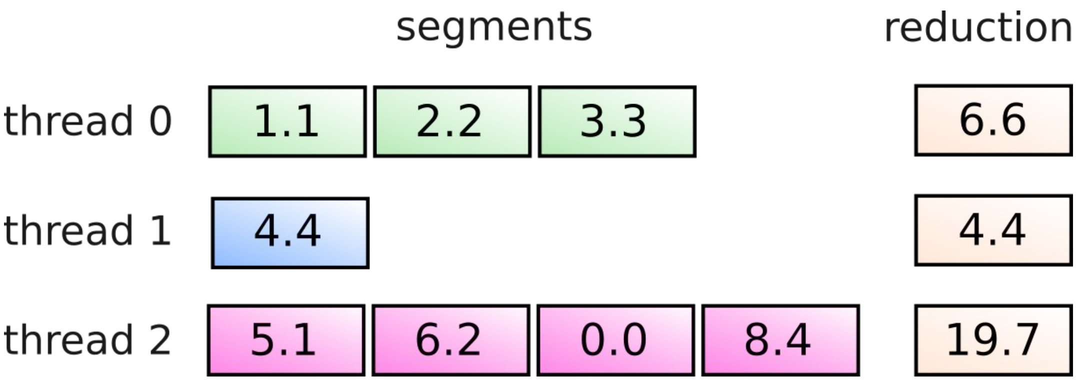


Categorizing CUDA Kernels



Reducer Kernels - Naive Approach

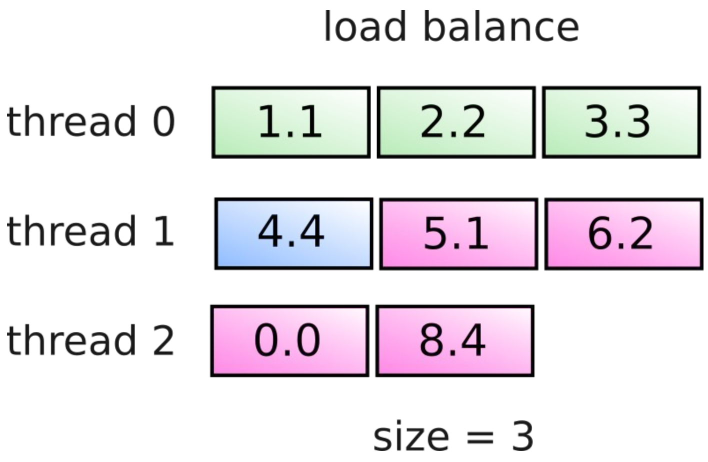
- Assigns one thread to each segment.
- Creates load imbalance.
- Each lane in a warp must wait for the entire warp to finish before returning.
- Threads assigned to long segments stall neighboring threads.



	segments				reduction
thread 0	1.1	2.2	3.3		6.6
thread 1	4.4				4.4
thread 2	5.1	6.2	0.0	8.4	19.7

Reducer Kernels - Load Balanced Approach

- Assign a fixed number of elements to each thread and sequentially accumulate consecutive elements.
- Store partial reduction as carry-out values when the last element of a segment is encountered and clear the accumulator.
- Cooperatively reduce the carry-out values and add them in the partial reductions.

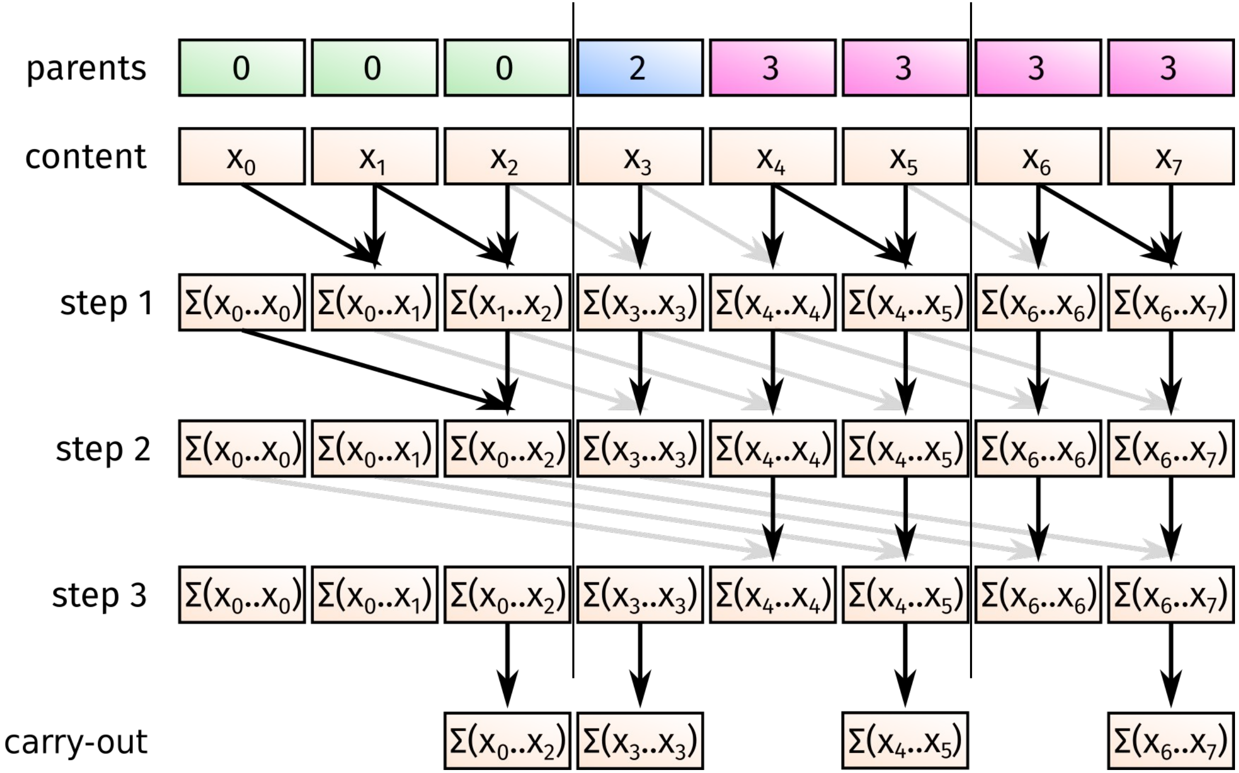


load balance

thread 0	1.1	2.2	3.3
thread 1	4.4	5.1	6.2
thread 2	0.0	8.4	

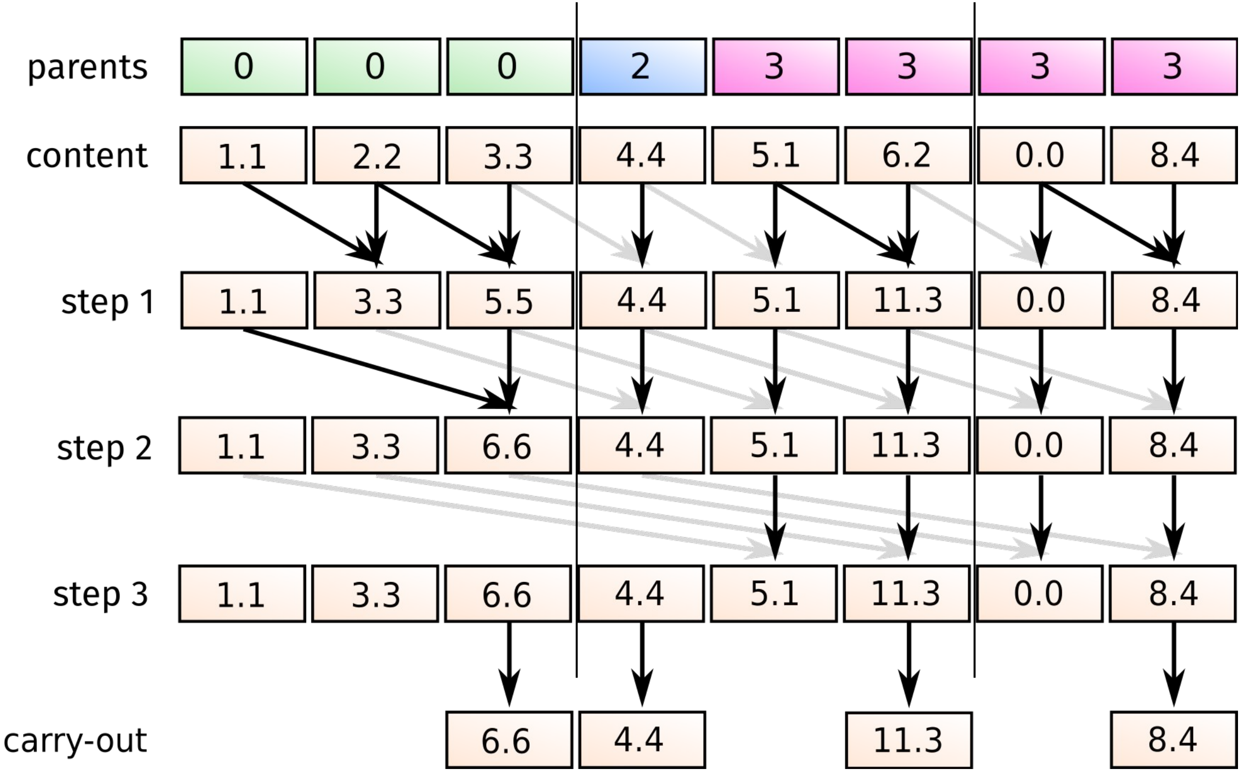
size = 3

Modified Hillis-Steele Algorithm



parents	0	0	0	2	3	3	3	3
content	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
step 1	$\Sigma(x_0..x_0)$	$\Sigma(x_0..x_1)$	$\Sigma(x_1..x_2)$	$\Sigma(x_3..x_3)$	$\Sigma(x_4..x_4)$	$\Sigma(x_4..x_5)$	$\Sigma(x_6..x_6)$	$\Sigma(x_6..x_7)$
step 2	$\Sigma(x_0..x_0)$	$\Sigma(x_0..x_1)$	$\Sigma(x_0..x_2)$	$\Sigma(x_3..x_3)$	$\Sigma(x_4..x_4)$	$\Sigma(x_4..x_5)$	$\Sigma(x_6..x_6)$	$\Sigma(x_6..x_7)$
step 3	$\Sigma(x_0..x_0)$	$\Sigma(x_0..x_1)$	$\Sigma(x_0..x_2)$	$\Sigma(x_3..x_3)$	$\Sigma(x_4..x_4)$	$\Sigma(x_4..x_5)$	$\Sigma(x_6..x_6)$	$\Sigma(x_6..x_7)$
carry-out			$\Sigma(x_0..x_2)$	$\Sigma(x_3..x_3)$		$\Sigma(x_4..x_5)$		$\Sigma(x_6..x_7)$

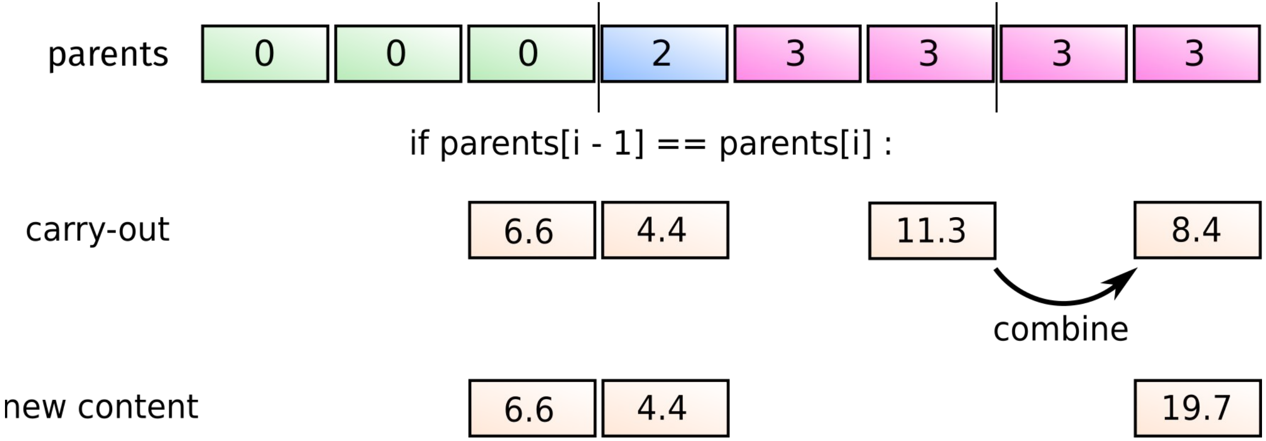
Modified Hillis-Steele Example



parents	0	0	0	2	3	3	3	3
content	1.1	2.2	3.3	4.4	5.1	6.2	0.0	8.4
step 1	1.1	3.3	5.5	4.4	5.1	11.3	0.0	8.4
step 2	1.1	3.3	6.6	4.4	5.1	11.3	0.0	8.4
step 3	1.1	3.3	6.6	4.4	5.1	11.3	0.0	8.4
carry-out			6.6	4.4		11.3		8.4

Across Block Boundary

- Only combine pairs in the same event by checking parents
- Take the last value in each event.



Awkward Functions : CPU vs CUDA Backend

```python
array = np.array([0, 1, 2, 3, 4, 5], dtype=np.int64)
content = ak.contents.NumpyArray(array)
offsets = ak.index.Index64(np.array([0, 3, 3, 6], dtype=np.int64))
depth1 = ak.contents.ListOffsetArray(offsets, content)
```

CPU

```python
# depth1 = ak.to_backend(depth1, "cpu")

ak.sum(depth1, axis=-1).to_list()
# [3, 0, 12]

ak.prod(depth1, axis=-1).to_list()
# [0, 1, 60]

ak.max(depth1, axis=-1).to_list()
# [2, None, 5]
```

CUDA

```python
depth1 = ak.to_backend(depth1, "cuda")

ak.sum(depth1, axis=-1).to_list()
# [3, 0, 12]

ak.prod(depth1, axis=-1).to_list()
# [0, 1, 60]

ak.max(depth1, axis=-1).to_list()
# [2, None, 5]
```

Testing of Kernels

- Modifying Python test generation scripts
  - To add custom unit tests in Python generated from a JSON file of test cases.
  - To fix generation of tests for kernels containing pointer-to-pointer.

- Adding integration tests for each Awkward function on the CUDA backend.

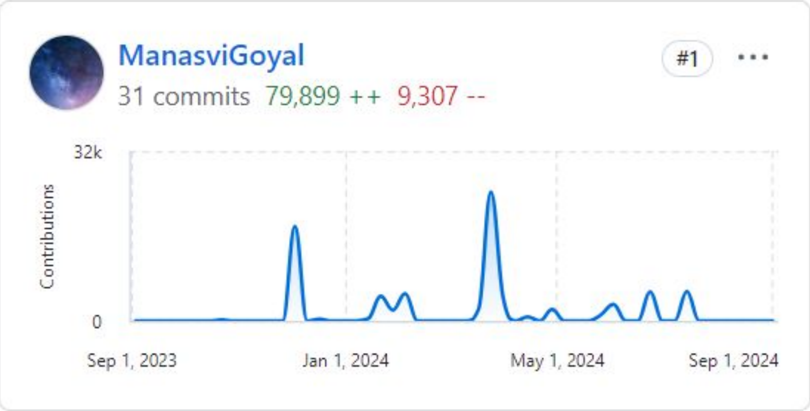
```python
def test_0115_generic_reducer_operation_count_max_1():
    content = ak.contents.NumpyArray(
        np.array([1.1, 2.2, 3.3, 0.0, 2.2, 0.0, 0.0, 2.2, 0.0, 4.4])
    )
    offsets = ak.index.Index64(np.array([0, 3, 6, 10], dtype=np.int64))
    cpu_depth1 = ak.contents.ListOffsetArray(offsets3, content2)
    cuda_depth1 = ak.to_backend(cpu_depth1, "cuda", highlevel=False)

    assert to_list(ak.max(cpu_depth1, -1, highlevel=False)) == [3.3, 2.2, 4.4]
    assert to_list(ak.max(cuda_depth1, -1, highlevel=False)) == [3.3, 2.2, 4.4]
```

Miscellaneous Tasks

- Adding kernel specifications of remaining Python kernels.
- Fixed some CPU kernels issues (memory access, nomenclature etc.)
- Adding some missing kernels for boolean type.
- Removing obsolete functions and dead code.

Summary

- Almost all major CUDA kernels implemented.
- Enhanced test coverage for CPU, CUDA and Python kernels.
- Users can now use Awkward functions in GPU to do their analysis.



ManasviGoyal

31 commits 79,899 ++ 9,307 --