Awkward Array

Accelerating Awkward Array Builders

Manasvi Goyal
IRIS-HEP Fellow
Delhi Technological University

Ianna Osborne
IRIS-HEP Mentor
Princeton University

Jim Pivarski
IRIS-HEP Mentor
Princeton University

Background

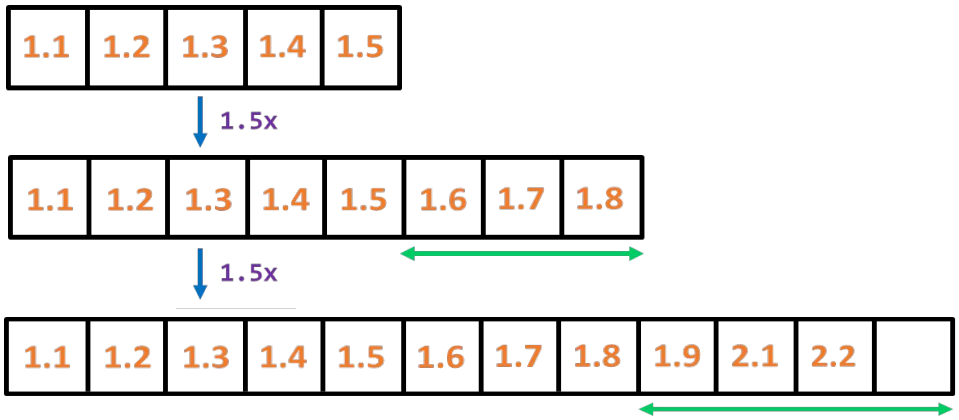
- Awkward Array is a library for nested, variable-sized data, including arbitrary-length lists, records, mixed types, and missing data, to manipulate JSON-like data using *NumPy-like idioms*.
- Awkward Arrays provide a more concise, faster and memory efficient alternative to the equivalent Python expression.
- My project concentrates on accelerating the performance of the Awkward Array Builders by preventing unnecessary memory copies, optimised allocation of memory and improving the speeds by specifying the Array Builder type in advance.

Awkward Bridge between Python and C++

- LayoutBuilder will be a templated static C++ code, implemented entirely in header files, and easily separable from the rest of the Awkward C++ codebase.
- It will now use GrowableBuffer, which will also be header-only.
- This will allow using Awkward Arrays in an external project without linking to the awkward libraries.
- This will facilitate dynamically generating LayoutBuilder from strings in Python and then compiling it with Cling.

Current Growable Buffer

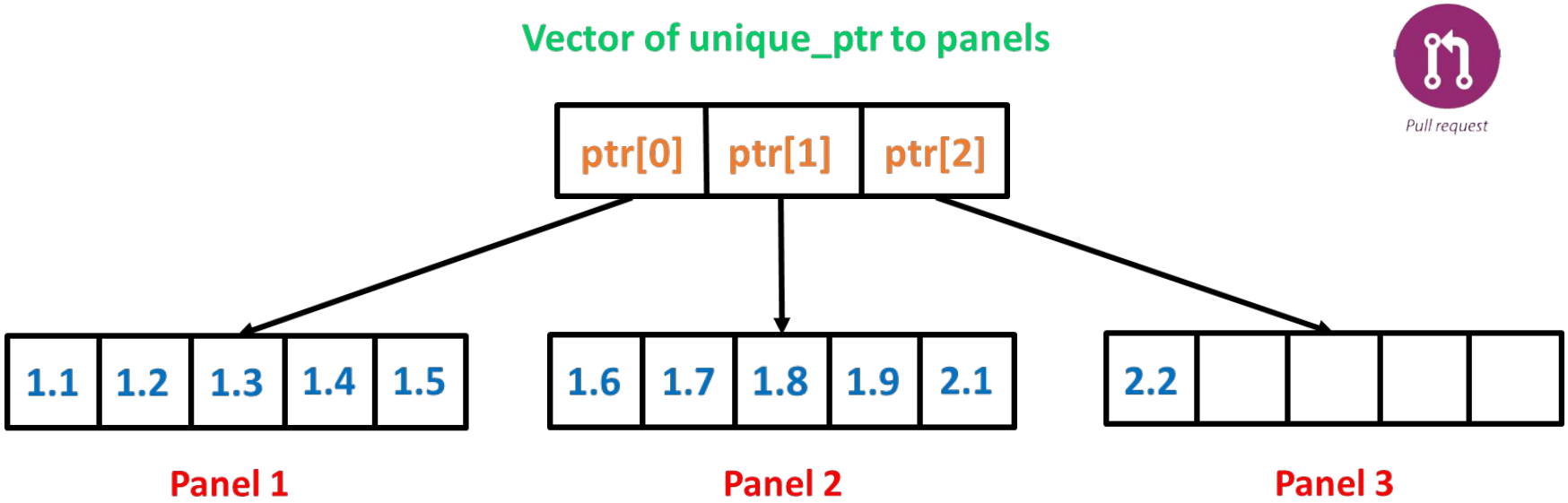
```
size_t reserved_ = 5, resize = 1.5;
double data[11] = {1.1, 1.2, 1.3, 1.4, 1.5, 1.6,
                   1.7, 1.8, 1.9, 2.1, 2.2};
```




```
template <typename T>
void GrowableBuffer<T>::append(T datum) {
  if (length_ == reserved_)
    set_reserved((size_t)ceil(reserved_ * resize));
  ptr_.get()[length_] = datum;
  length_++;
}

template <typename T>
void GrowableBuffer<T>::set_reserved(size_t minreserved) {
  if (minreserved > reserved_) {
    UniquePtr ptr(reinterpret_cast<T*>(awkward_malloc(
    (int64_t)(minreserved * sizeof(T)))));
    memcpy(ptr.get(), ptr_.get(), length_* sizeof(T));
    ptr_ = std::move(ptr);
    reserved_ = minreserved;
  }
}
```

New Growable Buffer with Panels



New Growable Buffer Unit Tests

```
void test_complex() {
  int data_size = 9;
  std::complex<double> data[9] = {{1.1, 0.1}, {2.2, 0.2}, {3.3, 0.3},
                                  {4.4, 0.4}, {5.5, 0.5}, {6.6, 0.6},
                                  {7.7, 0.7}, {8.8, 0.8}, {9.9, 0.9}};
  size_t initial = 3;
  auto buffer = GrowableBuffer<std::complex<double>>::empty(initial);
  for (int64_t i = 0; i < data_size; i++) {
    buffer.append(data[i]);
  }
  std::complex<double>* ptr = new std::complex<double>[data_size];
  buffer.concatenate(ptr);
  for (int64_t at = 0; at < buffer.length(); at++) {
    assert(ptr[at] == data[at]);
  }
}
```

```
void test_float() {
  int data_size = 18;
  float data[18] = {1.1, 1.2, 1.3, 1.4, 1.5, 1.6,
                    1.7, 1.8, 1.9, 2.1, 2.2, 2.3,
                    2.4, 2.5, 2.6, 2.7, 2.8, 2.9};
  size_t initial = 4;
  auto buffer = GrowableBuffer<float>::empty(initial);
  for (int64_t i = 0; i < data_size; i++) {
    buffer.append(data[i]);
  }
  float* ptr = new float[data_size];
  buffer.concatenate(ptr);
  for (int64_t at = 0; at < buffer.length(); at++) {
    assert(ptr[at] == data[at]);
  }
}
```

Future Work

- Develop templated, header-only LayoutBuilder which uses GrowableBuffer.
- Modify ArrayBuilder to accommodate the changes made in GrowableBuffer.
- Add methods to allow the user to add an entire array of data.

```
void append (PRIMITIVE *array, size_t array_size);
```

- Write C++ unit tests for LayoutBuilder and ArrayBuilder.
- Write performance comparison tests.
- Integrating the C++ code with Python.
- Submit an abstract to ACAT 2022.

Awkward Array

Thank You

ManasviGoyal

mg.manasvi@gmail.com

manasvi-goyal-2809