Find us on github!

The A4 library

liba4.net

Physics data processing with Google Protocol Buffers

Johannes Ebke[LMU] & Peter Waller[UL]

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

UNIVERSITY OF LIVERPOOL

Goals of the A4 library:

- Fast processing of HEP experimental data
- Powerful analysis tools - easy to use correctly
- Automatic metadata: no bookkeeping required

Why Google Protocol Buffers?

- **fast serialization** and deserialization of **structured data**
- separation of data structure definition and code
- simple, extendable message definition language
- code generators for many programming langauges
- thread safe
- forward and backward binary compatibility
- definition available at runtime
- very **well tested** and widely used

Protocol Buffer definition

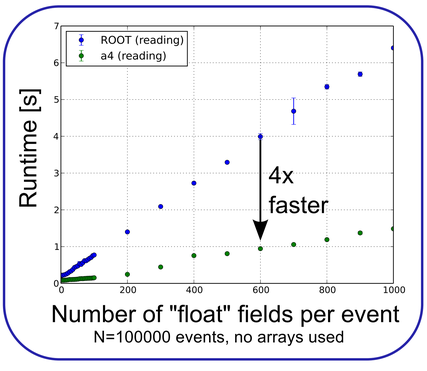
```
message Lepton {
    optional double pt = 1;
    optional double eta = 2;
    optional double phi = 3;
    optional int32 charge = 4;
}
message PhysicsEvent {
    optional int32 run_number = 1;
    optional int32 event_number = 2;
    repeated Lepton electrons = 5;
    repeated Lepton muons = 6;
}
```

A4 use at the LHC

A4 is currently being used to process ~**1 billion** ATLAS events per run on a local cluster. Results from the histograms, cutflows and systematic uncertainties evaluated using A4 have contributed to published ATLAS results.

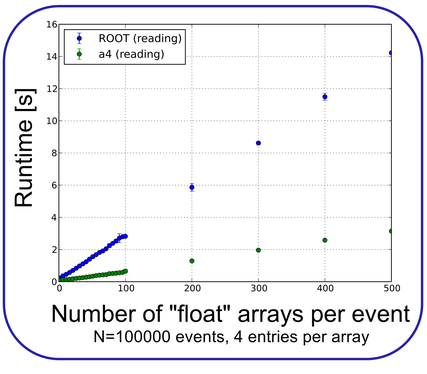
Benchmarking A4

- flat ntuple-like synthetic events with random entries
- entries can be **numbers** or variable-length **arrays**
- processing = adding up all entries
- comparison: ROOT 5.32 TTree



Number of "float" fields per event

N=100000 events, no arrays used



Number of "float" arrays per event

N=100000 events, 4 entries per array

The components of the A4 library

A4 File

- stores **arbitrary** protobuf messages, e.g. events
- stores message descriptors for all contained messages
- transparent **compression**
- stores **metadata** messages for groups of events
- can be added together using `cat`, with metadata still correctly available
- supports streaming mode with no seeks in the input
- preserves protobuf speed

Processing

This module provides a **processor** base class. Users can insert a **process** function which receives protobuf **events**, which can be analyzed, modified and **written**.
The **metadata** is automatically added to any output, and is also available in place, so **no manual bookkeeping** is required when processing large amounts of data in multiple files.
A program using this module can be compiled into an executable, with extensible **command-line options**. **Multithreading** is also managed automatically for you.
Cutflows and Histograms are also saved automatically and with metadata.

I/O library

- C++ and Python libraries
- direct dCache & RFIO access
- a4dump & a4info print **human-readable** representations of messages and **metadata**
- assists in **multithreading**
- manages all message descriptors

Channels

The call **channel("mu")** in a processor triggers a re-run of the whole analysis on this event, but all histograms will have the prefix **"channel/mu/"**. Only in this run the channel("mu") call will return true.

ROOT conversion

A4 files with histograms or events can be converted to ROOT files, and ROOT TTrees can be converted to A4 files.

Histogram Store

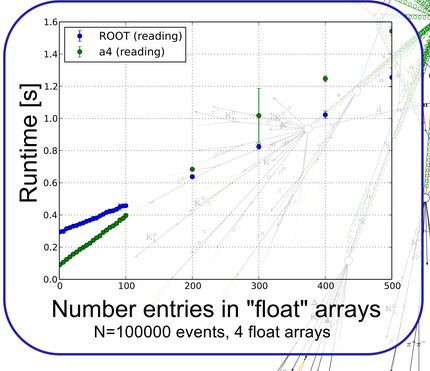
The histogram store enables creating, initializing, filling and saving histograms on one short line of code:
S.T<H1>("h")(5,0,5).fill(n_jets);
The store obejct "S" can be a subdirectory, and also stores event weights.
All this is **very fast**, since no strings are involved.
Ask for details if you are interested!

A4 File

Histograms are stored in A4 files together with the metadata

Systematics

Similar to **channels**, the function **systematics("JES")** returns true in a special run, but this time "jes" has to be added on the command line. All histograms will also have the prefix **"systematic/JES"**.



Number entries in "float" arrays

N=100000 events, 4 float arrays

Benchmark Results:

- The A4 library is competitive in terms of speed, in many cases up to 4x faster than ROOT
- No significant nonlinear behavior has been observed
- Handling of large arrays in events could be improved

MCViz: Visualization of HEP Monte Carlo

The background image of this poster has been generated using **MCViz**, another project of the authors. It can convert HepMC, LHE and Pythia events into beautiful graphs in a variety of styles.
It is available at **mcviz.net**

A4 Processor Example: Skim, slim and thin

```
class SkimSlimThin : public ProcessorOf<Event, MetaData> {
  public: void process(const Event& event) {
    // Cut on at least two muons (skim)
    if (event.size_muons() < 2) return;
    // Remove some fields (slim)
    Event e = event; e.clear_tracks();
    foreach(auto& m, *e.mutable_muons()) {
      // Smear muons before writing them
      if (metadata().simulation()) smear(m);
      m.clear_id_hits(); // Remove hits from muons (thin)
    }
    write(e); // Write modified event to output file
  }
};
```

Summary and Outlook

- The A4 library, while still experimental, can be used successfully for fast HEP data analysis
- Almost no manual bookkeeping is necessary due to the use and automatic handling of metadata
- Histograms and cutflows can be added quickly, and powerful tools to structure the analysis are available
- Use and collaboration is invited, visit us at **liba4.net**