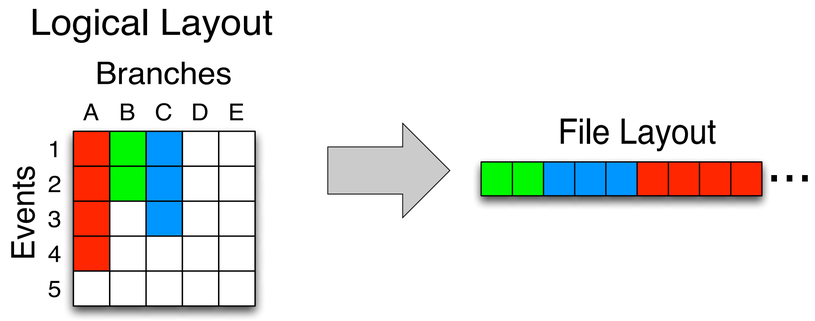
Speeding HEP Analysis with ROOT Bulk I/O

Brian Bockelman, Zhe Zhang, Oksana Shadura, *University of Nebraska-Lincoln*

Introduction

ROOT IO is an incredibly flexible format.

It can easily store the complex objects that correspond to the experiment's data.

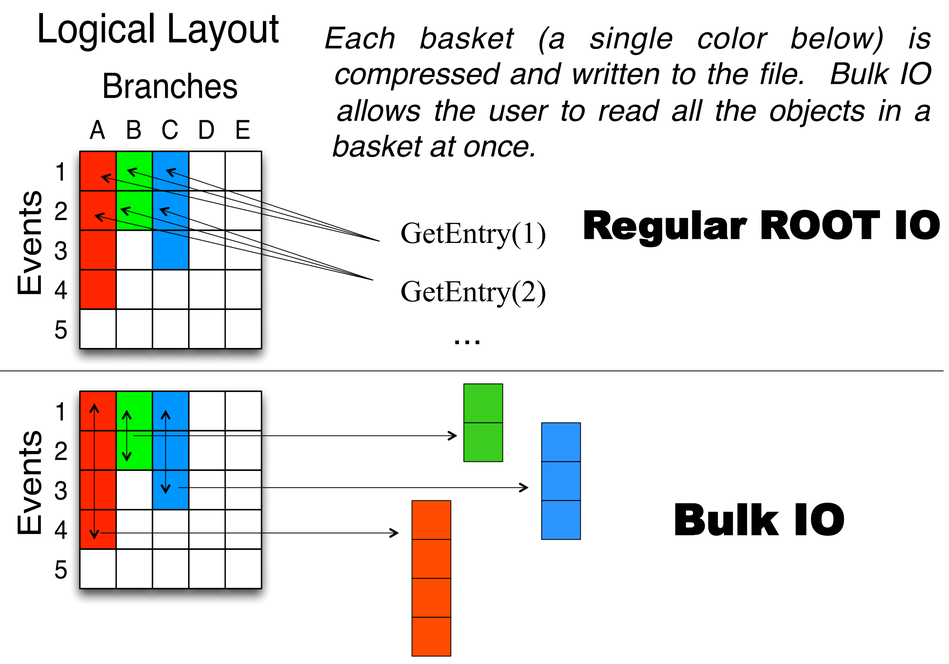


High Overheads for Simple Objects

Considering a TBranch that consists floats, reading a float object involves in:

- Virtual Calls
- Function pointer calls
- Bounds checking, error condition checking, etc.

ROOT IO vs Bulk IO: An Illustration



Each basket (a single color below) is compressed and written to the file. Bulk IO allows the user to read all the objects in a basket at once.

TTreeReader with Bulk IO

TTreeReader Interface

An interface for an user to access simple object (primitives, arrays, etc.).

Sample Code

```
1    TTreeReader myReader("T", hfile);
2    TTreeReaderValue<float> myF(myReader, "myFloat");
3    Long64_t idx = 0;
4    Float_t sum = 1;
5    while (myReader.Next()) {
6       sum += *myF;
7    }
```

TTreeReaderFast

- **myReader.Next()** is inlined by compiler, avoiding function calls.
- ***myF** would invoke the appropriate deserialization code.

RDataSource with Bulk IO

RDataFrame (RDF)

Interface design was inspired by other dataframe APIs such as pandas and Spark DataFrames.

RDataSource

We integrate Bulk IO into RDataFrame through RDataSource (RDS) which defines an API for RDF to read arbitrary data formats.

Turbocharging ROOT with Bulk IO

What is Bulk IO?

Bulk IO is a set of techniques and APIs we developed for ROOT that allows the user to deserialize a large set of events at a time.

Why Bulk IO?

For small, simple events the overhead of ROOT library calls is much larger than the cost of deserialization in user-space. By returning an entire basket of serialized objects to the user, the user can deserialize data when needed.

Further improvements can be achieved by returning serialized events to the user and allowing the compiler to inline deserialization in the event loop.

Why Not Bulk IO?

Complex objects involving references or from polymorphic classes require expensive lookups to deserialize data. In these cases, the library overheads are minimal and bulk IO provides little benefit.

API Implementation

Design API

Int_t TBranch::GetBulkEntries(Long64_t entry, TBuffer &user_buf)

Arguments

entry: an entry number that contains a complex event object.
user_buf: an user-defined buffer that stores deserialized basket.

Access to Events

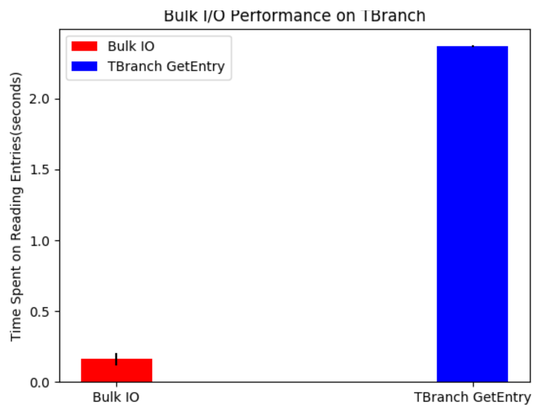
An user can later on access an event in the basket using:
****reinterpret_cast<T*>(user_buf.GetCurrent())[idx]***
where T is the object type.

Performance of TBranch and TTreeReaderFast

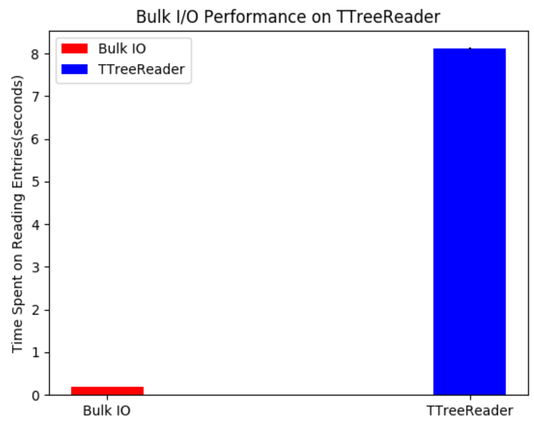
Test Setup:

- Desktop Intel i5 4-Core @ 3.2GHz
- Reading from a TTree with 100 million float values.

10+ times faster !



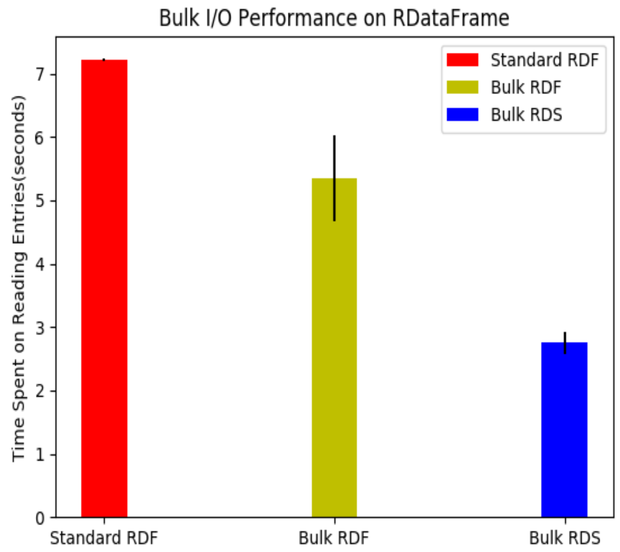
TBranch::GetEntry vs. TBranch::GetBulkEntries



TTreeReader vs. TTreeReaderFast

Performance with RDataSource and RDataFrame

- **RDataFrame introduces overheads on Bulk IO, but Bulk IO still outperforms regular RDF.**
- Using RDataSource directly is 2X faster than regular RDF.



Acknowledgement

This work was supported by the National Science Foundation under Grant ACI-1450323. This research was done using resources provided by the Holland Computing Center of the University of Nebraska.