Argonne NATIONAL LABORATORY

ProMC file format

S. Chekanov
HEP/ANL

March 29, 2016
Thomas Jefferson National Accelerator Facility

ProMC file format

S.C., E.May, K. Strand, P. Van Gemmeren, Comp. Physics Comm. 185 (2014), 2629

- "Archive" data format to keep MC events:
  - Event records, NLO,original logfiles, PDG tables etc.
- 30% smaller files than existing formats after compression

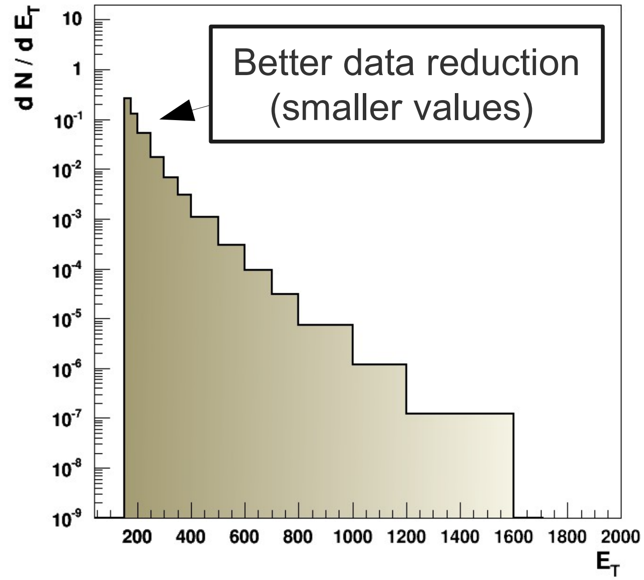
Number of used bytes depends on values.
Small values use small number of bytes

Google's Protocol buffers — protobuf
Protocol Buffers - Google's data interchange format

- Effective file size reduction for pile-up events
  - Particles with small momenta → less bytes used
- Installed on Mira (BlueGene/Q).
- Supports C++/Java/Python
- Separate events can be streamed over the Internet:
  - similar to avi frames (video streaming)

http://atlaswww.hep.anl.gov/asc/promc/

8-bytes (int64) → varint



compression strength keeping precision of constant

Benchmarks for EVGEN files

ProMC files:
- 12 times smaller than HEPMC
- 30% smaller than ROOT

~30% faster to process (C++/Java)

File sizes for 10,000 $t\bar{t}$ events for pp at LHC

File format	File Size (MB)	C++ (sec)	CPython (sec)	Java (sec)	Jython (sec)
ProMC	307	15.8	980	11.7 (12.1 +JVM startup)	33.3 (35 +JVM startup)
ROOT	423	20.4	66.7 (PyROOT)	-	-
LHEF	2472	84.7	30.4	9.0 (9.6 +JVM startup)	-
HEPMC	2740	175.1	-	-	-
LHEF (gzip)	712	-	-	-	-
LHEF (bzip2)	552	-	-	-	-
LHEF (lzma)	513	-	-	-	-
HEPMC (gzip)	1021			-	-
HEPMC (bzip2)	837			-	-
HEPMC (lzma)	802	-	-	-	-

ASCII text files
(after compression)

Table 1. Benchmark tests for reading files with 10,000 ttbar events stored in different file formats. For each test, the memory cache on Linux was cleared. In case of C++, the benchmark program reads complete event records using appropriate libraries. CPython code for ProMC file is implemented in pure CPython and does not use C++ binding (unlike PyROOT that uses C++ libraries). In case of LHEF files. JAVA and CPYTHON benchmarks only parse lines and tokenize the strings, without attempting to build an event record, therefore, such benchmarks may not be accurate while comparing with ProMC and ROOT.

https://atlaswww.hep.anl.gov/asc/wikidoc/doku.php?id=asc:promc:introduction

Google's ProtocolBuffers

https://developers.google.com/protocol-buffers/

Protocol Buffers

Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

HOME GUIDES REFERENCE SUPPORT

Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args|
john.writeTo(output);
```

```
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

What are protocol buffers?

Protocol buffers are Google's language-neutral, platform-neutral, extensible

Pick your favourite language

How do I start?

1. Download and install the protocol buffer compiler

used at Google for storing / interchanging structured information.

Google's ProtocolBuffers

https://developers.google.com/protocol-buffers/

- ProMC is a method to organize and stream (write/read) ProtocolBuffers messages.
- Uses “zip” to combine entries (“lossless”)
- One massage is one “event”
- Streaming is done using:
  - **zipios** library (not maintained) uses zip32 (~65k files)
  - **libzip** library (zip64)
    - no 65k limit, but heavy memory usage
- The choice “zip32 vs zip64” is done during file creation
  - Java always write zip64 entries!
- There is a “distiller” program “prom2promc” to convert zip32 to zip64

Why ProMC is self-describing format?

ProtocolBuggers interface description language

- Unzip a ProMC file: **unzip <file.promc>**:
  - version
  - description
  - header
  - 0
  - 1
  - 2
  - 3
  - ..
  - ProMCDescription.proto
  - ProMCHeader.proto
  - ProMC.proto
  - ProMCStat.proto
  - logfile.txt

Events in compact, binary wire format. Uses “varints” to compact int64

ASCII data layout file describing event structure using platform-neutral Prottocol-Buffer syntax (Google)

```
message ProMCEvent {

  // Event information
  message Event {
        optional int32 Number = 1;      // Number
        optional int32 Process_ID = 2;  // ID unique signal process id
        optional int32 MPI        = 3;  // MPI number of multi parton inter
        optional int32 ID1        = 4;  // ID1 flavour code of first parton
        optional int32 ID2        = 5;  // ID2 flavour code of second parto
        optional float PDF1       = 6;  // PDF1 PDF (id1, x1, Q)
        optional float PDF2       = 7;  // PDF2 PDF (id2, x2, Q)
        optional float X1         = 8;  // X1 fraction of beam momentum car
        optional float X2         = 9;  // X2 fraction of beam momentum car
        optional float Scale_PDF  = 10; // Scale PDF Q-scale used in evalua
        optional float Alpha_QED  = 11; // AlphaQED QED coupling, see hep-p
        optional float Scale      = 12; // energy scale, see hep-ph/0109068
        optional float Alpha_QCD  = 13; // QCD coupling, see hep-ph/0109068
        optional double Weight    = 14; // event weight
  }

 // Generator (truth) particles
 message Particles {
   repeated uint32  id=1          [packed=true]; // ID in the generator
   repeated sint32  pdg_id=2      [packed=true]; // unique integer ID speci
   repeated uint32  status=3      [packed=true]; // integer specifying the
   repeated uint64  mass=4        [packed=true]; // mass
   repeated sint64  Px=5          [packed=true]; // pX
   repeated sint64  Py=6          [packed=true]; // pY
   repeated sint64  Pz=7          [packed=true]; // pZ
   repeated uint32  mother1=8     [packed=true]; // first mother
   repeated uint32  mother2=9     [packed=true]; // second mother
   repeated uint32  daughter1=10  [packed=true]; // first daughter
   repeated uint32  daughter2=11  [packed=true]; // second daughter
   repeated sint32  barcode=12    [packed=true]; // barcode if used
   repeated sint32  X=13          [packed=true]; // vertex X position
   repeated sint32  Y=14          [packed=true]; // vertex Y position
   repeated sint32  Z=15          [packed=true]; // vertex Z position
   repeated uint32  T=16          [packed=true]; // time
   repeated uint64  weight=17     [packed=true]; // particle weight
   repeated sint32  charge=18     [packed=true]; // Charge
   repeated sint64  energy=19     [packed=true]; // Energy
 }
```

If you know “ProMC.proto” template, you can generate analysis code (C++/Java/Python) for reading / writing binary entries inside ProMC

Event structure. Examples

"HEPMC" truth record using for Snowmass13

```
option java_outer_classname = "ProMC";

message ProMCEvent {

  // Event information
  message Event {
        optional int32 Number = 1;        // Number
        optional int32 Process_ID = 2;    // ID unique signal process id
        optional int32 MPI       = 3;     // MPI number of multi parton interactions
        optional int32 ID1       = 4;     // ID1 flavour code of first parton
        optional int32 ID2       = 5;     // ID2 flavour code of second parton
        optional float PDF1      = 6;     // PDF1 PDF (id1, x1, Q)
        optional float PDF2      = 7;     // PDF2 PDF (id2, x2, Q)
        optional float X1        = 8;     // X1 fraction of beam momentum carried by first parton ("beam side")
        optional float X2        = 9;     // X2 fraction of beam momentum carried by second parton ("target side"
        optional float Scale_PDF = 10;    // Scale PDF Q-scale used in evaluation of PDF's (in GeV) |
        optional float Alpha_QED = 11;    // AlphaQED QED coupling, see hep-ph/0109068
        optional float Scale     = 12;    // Scale   energy scale, see hep-ph/0109068
        optional float Alpha_QCD = 13;    // QCD coupling, see hep-ph/0109068
        optional double Weight   = 14;    // event weight
  }

 // Generator (truth) particles
 message Particles {
   repeated uint32  id=1          [packed=true]; // ID in the generator
   repeated sint32  pdg_id=2      [packed=true]; // unique integer ID specifying the particle type
   repeated uint32  status=3      [packed=true]; // integer specifying the particle's status (i.e. decayed or n
   repeated uint64  mass=4        [packed=true]; // mass
   repeated sint64  Px=5          [packed=true]; // pX
   repeated sint64  Py=6          [packed=true]; // pY
   repeated sint64  Pz=7          [packed=true]; // pZ
   repeated uint32  mother1=8     [packed=true]; // first mother
   repeated uint32  mother2=9     [packed=true]; // second mother
   repeated uint32  daughter1=10  [packed=true]; // first daughter
   repeated uint32  daughter2=11  [packed=true]; // second daughter
   repeated sint32  barcode=12    [packed=true]; // barcode if used
   repeated sint32  X=13          [packed=true]; // vertex X position
   repeated sint32  Y=14          [packed=true]; // vertex Y position
   repeated sint32  Z=15          [packed=true]; // vertex Z position
   repeated uint32  T=16          [packed=true]; // time
   repeated uint64  weight=17     [packed=true]; // particle weight
   repeated sint32  charge=18     [packed=true]; // Charge
 }

   // even record for this event
   optional Event      event = 1;          // information on event
   optional Particles  particles = 2;      // information on generator-level particles

}
```

Example of reconstructed event record for Snowmass13 (Delphes3)

```
// GenJet
 message GenJets {
      repeated uint64  PT=1            [packed=true]; // PT
      repeated sint64  Eta=2           [packed=true]; // Eta
      repeated sint64  Phi=3           [packed=true]; // Phi
      repeated uint64  Mass=4          [packed=true]; // Mass
      repeated sint32  Btag=5          [packed=true]; // BTag
      repeated sint32  TauTag=6        [packed=true]; // TauTag
      repeated sint32  Charge=7        [packed=true]; // Charge
      repeated uint32  DeltaEta=8      [packed=true]; // Delta Eta
      repeated uint32  DeltaPhi=9      [packed=true]; // Delta Phi
      repeated sint32  HadOverEem=10    [packed=true]; // Ehad/Eem
 }

 // Electrons
 message Electrons {
      repeated uint64  PT=1            [packed=true]; // PT
      repeated sint64  Eta=2           [packed=true]; // Eta
      repeated sint64  Phi=3           [packed=true]; // Phi
      repeated sint32  Charge=4        [packed=true]; // Charge
 }

// Reconstructed Muons
 message Muons {
      repeated uint64  PT=1            [packed=true]; // PT
      repeated sint64  Eta=2           [packed=true]; // Eta
      repeated sint64  Phi=3           [packed=true]; // Phi
      repeated sint32  Charge=4        [packed=true]; // Charge
 }

// Charged Tracks
 message Tracks {
      repeated uint64  PT=1            [packed=true]; // PT
      repeated sint64  Eta=2           [packed=true]; // Eta
      repeated sint64  Phi=3           [packed=true]; // Phi
      repeated sint32  Charge=4        [packed=true]; // Charge
      repeated sint32  X=5             [packed=true]; // vertex X position
      repeated sint32  Y=6             [packed=true]; // vertex Y position
      repeated sint32  Z=7             [packed=true]; // vertex Z position
      repeated sint32  XOuter=8        [packed=true]; // XOuter
      repeated sint32  YOuter=9        [packed=true]; // YOuter
      repeated sint32  ZOuter=10       [packed=true]; // ZOuter
      repeated sint32  EtaOuter=11     [packed=true]; // EtaOuter
      repeated sint32  PhiOuter=12     [packed=true]; // PhiOuter
 }
```

Data structures can be very complicated..

Converters (in C++ and Java)

ProMC Commands	Description
hepmc2promc <HEPMC input> <ProMC output> "description"	converts HepMC file to ProMC file
promc2hepmc <ProMC input> <HepMC output>	converts ProMC file to HEPMC file
stdhep2promc <StdHEP input> <ProMC output>	converts StdHEP file to ProMC file
promc2root <ProMC input> <ProMC output>	converts ProMC file to ROOT
promc2stdhep <ProMC input> <STDHEP output>	converts ProMC file to STDHEP
promc2lcio <ProMC input> <LCIO output>	converts ProMC file to LCIO
lhe2promc <LHE input> <ProMC output>	converts LHEF file to ProMC

Many tools: merge split etc..

Using varints to compact data:

Energy	Representation	How many bytes in encoding
0.01 MeV	1	1 bytes
0.1 MeV	10	1 bytes
1 MeV	100	2 bytes
1 GeV	100 000	4 bytes
1 TeV	100 000 000	8 bytes
20 TeV	2000 000 000	8 bytes

Monte Carlo event record have many integers (PID, ID, mother1, mother2, daughter1,2, status code, etc)

They are all small integers → can be represented by 1-2 bytes of varints

Example of reading a ProMC file
(Check $PROMC variable first!)

- See: https://atlaswww.hep.anl.gov/asc/wikidoc/doku.php?id=asc:promc:examples
  - wget http://atlaswww.hep.anl.gov/asc/promc/download/Pythia8.promc
  - promc_proto Pythia8.promc # extracts data layouts into the directory "proto"
  - promc_code # creates C++ analysis code
  - make # compiles C++ code reader.cc
  - ./reader Pythia8.promc # runs the C++ analysis code
  - unzip -p Pythia8.promc logfile.txt # extracts Monte Carlo log file

"promc_code" also creates analysis codes in C++, Java, Python (in directories /src /java /python directories)

"promc_code" rebuilds C++ header files (or Java classes) using input ProtocolBuffers template files

If "template" files are not appended to the file, data cannot be recovered.

Creating a ProMC file

- Look at the example: $PROMC/examples/random
- Make a directory “**proto**” and define templates for your data
- promc_code # creates header files/source codes (in “src”)
- make; ./writer # write your data

Summary

- **ProMC files are well tested since 2013:**
  - **up to 4 GB per file**
  - **up to 200k events/entries (ZIP64 version)**
  - **1.5 billion events stored in HepSim (http://atlaswww.hep.anl.gov/hepsim/) since Snowmass 2013**

- **To be solved:**
  - Two types of ProMC are currently in use: based ZIP (max 65k entries) and ZIP64 types
    - ZIP allows fast streaming of records (**zipios** library), but max number of events is 65k
    - ZIP64 uses libzip, but it accumulates data before streaming
    - How to unify these 2 approaches?
- **We need to fix zipios library for zip64! Geant4 uses this library too!**