EvtGen status and plans

M. Kreps for EvtGen team

Physics Department

WARWICK

Reminder of EvtGen

WARWICK

- EvtGen is package specialised in decays of heavy flavour hadrons
- Originally written by A. Ryd and D. Lange for use at Babar
- About 20 years old code and code style/design reflects that

- EvtGen internally works by expressing amplitudes squared and using accept-reject method to generate random events
- Some of the design decision in original code are not best in todays view
  - Effectively only one instance of EvtGen was possible up to now (will come to this later)
- Has few dependencies (listing main ones):
  - Photos, Tauola, Pythia8
  - HepMC

Code modernisation

WARWICK

- In the code improvements within LHCb G. Raven made some modernisation:
  - remove redundant explicit constructor and destructors, prefer implicitly generated ones instead
  - improve memory management (fixing a few memory leaks as side effect)
    - use std::unique_ptr instead of raw pointers with explicit delete
    - prefer std::vector over raw heap allocated array
    - prefer objects on the stack in case their lifetime is limited
    - if a class has non-trivial copy constructor, implement a consistent assignment operator
- This is all fresh from past few days
  - We need to test everything and port to our general repository
    - From historical reasons LHCb is using own copy in LHCb repository, which will be removed in near future

Code modernisation

- Lot of changes inside code with no change in functionality
- Mainly cleans up code
- Will be available in near future

New

```
void EvtHQET2::initProbMax(){

  EvtId parnum,mesnum,lnum,nunum;

  parnum = getParentId();
  mesnum = getDaug(0);
  lnum = getDaug(1);
  nunum = getDaug(2);

  double mymaxprob = calcamp->CalcMaxProb(parnum,mesnum,
                          lnum,nunum,hqetffmodel.get());

  setProbMax(mymaxprob);

}


void EvtHQET2::init(){

  checkNDaug(3);

  //We expect the parent to be a scalar
  //and the daughters to be X lepton neutrino
  checkSpinParent(EvtSpinType::SCALAR);

  checkSpinDaughter(1,EvtSpinType::DIRAC);
  checkSpinDaughter(2,EvtSpinType::NEUTRINO);

  EvtSpinType::spintype d1type = EvtPDL::getSpinType(getDaug(0));
  if ( d1type==EvtSpinType::SCALAR) {

    if ( getNArg()==2 ) {

      hqetffmodel = std::make_unique< EvtHQET2FF >(getArg(0),getArg(1));
      calcamp = std::make_unique< EvtSemiLeptonicScalarAmp >();

    } else if ( getNArg()==3 ) {

      hqetffmodel = std::make_unique< EvtHQET2FF >(getArg(0),getArg(1),getArg(2));
      calcamp = std::make_unique< EvtSemiLeptonicScalarAmp >();

    } else {

      report(ERROR,"EvtGen") << "HQET2 model for scalar meson daughters needs 2 arguments for norma
      ::abort();
    }

  } else if ( d1type==EvtSpinType::VECTOR) {

    if ( getNArg()==4 ) {
```

Old

```
void EvtHQET2::initProbMax(){

  EvtId parnum,mesnum,lnum,nunum;

  parnum = getParentId();
  mesnum = getDaug(0);
  lnum = getDaug(1);
  nunum = getDaug(2);

  double mymaxprob = calcamp->CalcMaxProb(parnum,mesnum,
                          lnum,nunum,hqetffmodel);

  setProbMax(mymaxprob);

}


void EvtHQET2::init(){

  checkNDaug(3);

  //We expect the parent to be a scalar
  //and the daughters to be X lepton neutrino
  checkSpinParent(EvtSpinType::SCALAR);

  checkSpinDaughter(1,EvtSpinType::DIRAC);
  checkSpinDaughter(2,EvtSpinType::NEUTRINO);

  EvtSpinType::spintype d1type = EvtPDL::getSpinType(getDaug(0));
  if ( d1type==EvtSpinType::SCALAR) {

    if ( getNArg()==2 ) {

      hqetffmodel = new EvtHQET2FF(getArg(0),getArg(1));
      calcamp = new EvtSemiLeptonicScalarAmp;

    } else if ( getNArg()==3 ) {

      hqetffmodel = new EvtHQET2FF(getArg(0),getArg(1),getArg(2));
      calcamp = new EvtSemiLeptonicScalarAmp;

    } else {

      report(ERROR,"EvtGen") << "HQET2 model for scalar meson daughters needs 2 arguments for norm
      ::abort();
    }

  } else if ( d1type==EvtSpinType::VECTOR) {

    if ( getNArg()==4 ) {
```

Towards multithreading

WARWICK

- Multithreading is next step
- Strictly, when we generate 100k or more events, it does not matter too much whether multithreading is used
- Main reason for multithreading is memory management
  - Number of cores per CPU grows faster than memory available
  - Experience tells that when we try to parallelise code, changes to memory alignments improve code performance also in single thread mode
- LHCb future simulation framework is multithreaded and so we need EvtGen version which can be safely used
- Plan is to allow processing n events in n threads (each event being processed in single thread)
- We do not think it is worth of spending time to work out exact details to parallelise on smaller units

Towards multithreading

- Random number engine is one of the main issues
- If we want to have reproducible results, each thread needs its own random number engine
- Will require significant redesign of the code
- Use functional approach, where we pass random number engine around
- Plan is to get first prototype with few decay models for development

```
while (s == 0.0)
  {
  xbox = EvtRandom::Flat(smin, smax);
  ybox = EvtRandom::Flat(_dGdsProbMax);
  double prob= _calcprob->dGdsProb(mb, _ms, ml, xbox);
  if ( !(prob>=0.0) && !(prob<=0.0)) {
    //      report(INFO,"EvtGen") << "nan from dGdsProb
  }
  if ( ybox < prob ) s=xbox;
  }
```

```
class EvtRandomEngine;

class EvtRandom{

public:

  static double Flat();
  static double Flat(double max);
  static double Flat(double min, double max);

  //generate unit Gaussian
  static double Gaussian();

  static double random();

  //This class does not take ownership of the random engine;
  //the caller needs to make sure that the engine is not
  //destroyed.
  static void setRandomEngine(EvtRandomEngine* randomEngine);

private:

  static EvtRandomEngine* _randomEngine;

};
```

Towards multithreading

WARWICK

- Two other major singletons are table of particle properties and decay table
  - In my view, it was wrong decision but there are probably not many uses cases where this would be limiting factor
  - In first multithread safe version we plan to keep those global objects used by all threads
    - Once they are initialised at start they do not change
- There are some more static variables in various places, which we will have to review but they can be probably handled easily
- Question will be in dependencies
  - Pythia8 should be thread safe
  - Photos and Tauola is not clear to
  - We use root in testing but there we do not worry too much

Towards multithreading

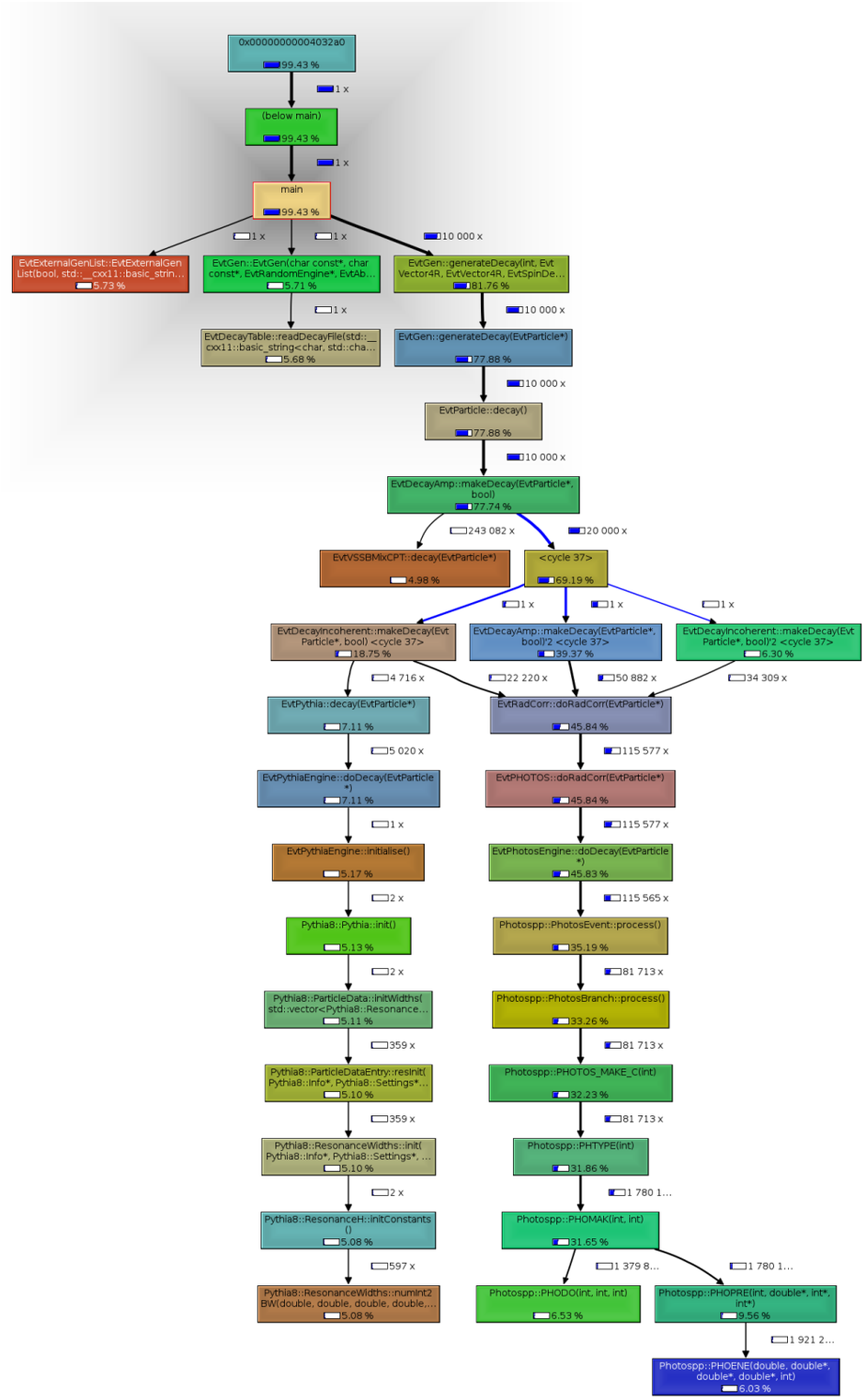
WARWICK

- EvtGen internally uses its own storage
- At the end of event generation result is translated to HepMC event record returned to user
- HepMC is also used in communication with Pythia8, Tauola and Photos
- Currently HepMC v2 is used
- Change to HepMC v3 expected to be simple as it is used only in limited number of places
  - We might have this in month or two
- HepMC v3 is ready for multithreading so this side should be resolved easily

- Will try to get multithread version within next few months
  - Not full functionality but enough to allow people start to include it in their multithread code under development

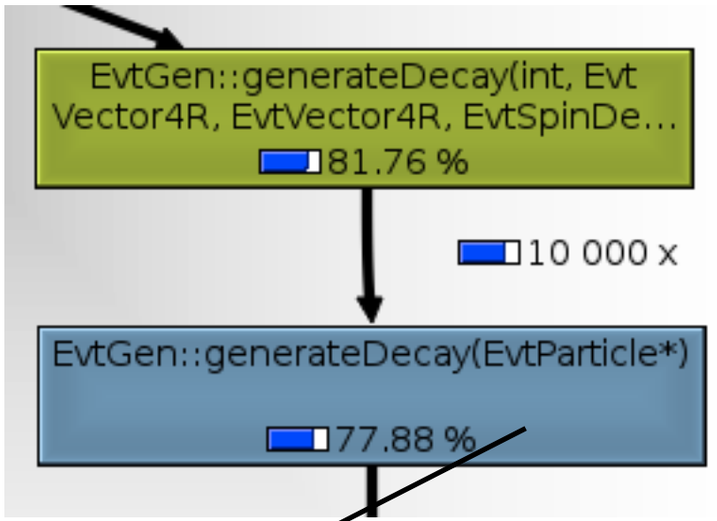
Running through profiler

- Ran generic decays through callgrind
- Main motivation was to help us to see how different bits are connected
- Some interesting information on performance can be easily seen
- We did not had time to fully digest everything it provides



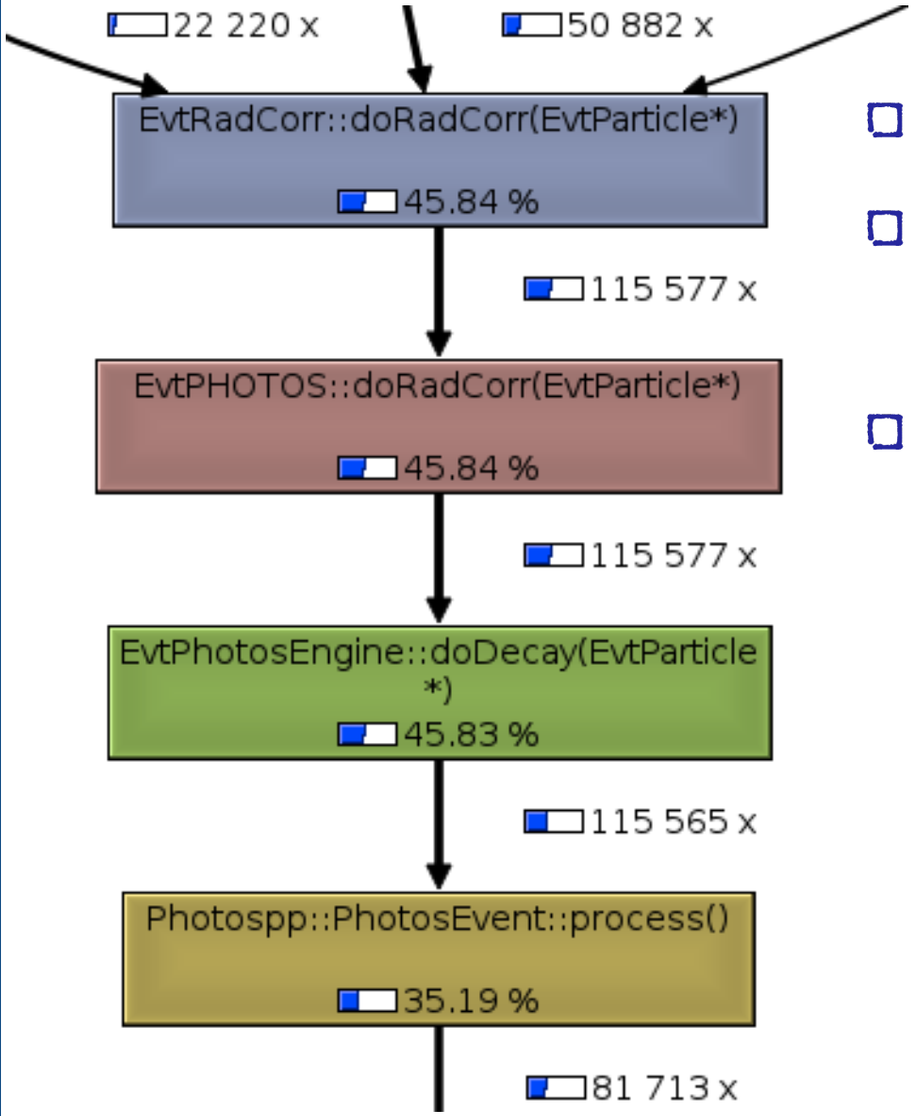
Running through profiler

- Difference between two generateDecay function is creation/deletion of EvtParticle and creating HepMC with result




```
void EvtGen::initExternalGenerators() {

  report(INFO,"EvtGen")<<"Initialising all external generators"<<endl;

  EvtExternalGenFactory* externalGenerators = EvtExternalGenFactory::getInstance();
  externalGenerators->initialiseAllGenerators();

  _initExternalGenerators = true;

  report(INFO,"EvtGen")<<"Done initialising external generators"<<endl;

}

EvtHepMCEvent* EvtGen::generateDecay(int PDGId, EvtVector4R refFrameP4,
             EvtVector4R translation,
             EvtSpinDensity* spinDensity) {

  EvtParticle* theParticle(0);

  if (spinDensity == 0 ){
    theParticle = EvtParticleFactory::particleFactory(EvtPDL::evtIdFromStdHep(PDGId),
                  refFrameP4);
  } else {
    theParticle = EvtParticleFactory::particleFactory(EvtPDL::evtIdFromStdHep(PDGId),
                  refFrameP4, *spinDensity);
  }

  generateDecay(theParticle);
  EvtHepMCEvent* hepMCEvent = new EvtHepMCEvent();
  hepMCEvent->constructEvent(theParticle, translation);

  theParticle->deleteTree();

  return hepMCEvent;

}
```

Running through profiler



- We seem to spend lot of time in Photos
- We did not had time to digest this enough to say whether it is Photos or how we use it
- Clearly significant fraction of time is spent in preparing info to pass to Photos and decode it back
  - We create HepMC event, pass it to Photos and then work through HepMC event to transfer result back to own storage
- Log and tan contribute significant amount
- Lot of memory allocation and deallocation
- We will need some time to investigate things in more detail

Plans

- We hope to move to HepMC v3 within next 2 months
  - Need to see how our dependencies support it
- Probably most tricky for us is what Photos and Tauola allows
  - Tauola is something we could possibly drop in favour of Pythia8
- On scale of about half a year we will try to prepare prototype of multithreading
  - Event is most sensible unit for us
  - What we call event is not necessarily what others call event
  - Our event can be single heavy quark hadron, second hadron can be second event from EvtGen point of view
- At this moment we do not see obvious place where we can gain speed
  - Does not mean it is impossible but that it is probably not easy
  - Some improvements likely from ongoing code modernisation