Parallelization and Legacy code:

a preliminary work on EvtGen

Scenario

Wishing to reuse part of the code written by the BaBar collaboration, there are two main questions that require an answer:

- Is it possible to run in parallel BaBar code (legacy code)? If this is the case, what kind of performances can be expected?
- What type of parallelization can be done on this code with the minimum impact?

Trying to run EvtGen in parallel provided some usefull information

EvtGen

EvtGen is: «…an event generator designed for the simulation of the physics of B decays.» (http://www.slac.stanford.edu/~lange/EvtGen)

Some characteristics:

- Can run in «standalone» mode (without the BaBar Framework)
- It's written in C++ and interfaced with Fortrans event generators
- It depends on legacy code written in Fortran (Pythia, Photos) and C++ (CERNLib, CLHEP)

Simple Event Generation


```
[…]

EvtRandomEngine* MyRandomEngine;
MyRandomEngine = new EvtCLHEPRandomEngine();

double xyzt = 0.0;
HepLorentzVector t_init(xyzt,xyzt,xyzt,xyzt);

EvtGen* myGenerator = new EvtGen("DECAY.DEC","evt.pdl",MyRandomEngine);

EvtVector4R p_init(EvtPDL::getMass(EvtPDL::getId("Upsilon(4S)")),0.0,0.0,0.0);

Event  = EvtParticleFactory::particleFactory(EvtPDL::getId("Upsilon(4S)"), p_init);
Event->setVectorSpinDensity();
TheGenerator->generateEvent(Event, t_init);

[…]
```

Initialize Random number generator

Set initial conditions

Generate one Event

The TBB Approach

Parallelization of a for cycle is done defining a BodyObject as follow:

```
Class BodyObject
{
        private:
                <Thread Pool Private Data>
        public:
                BodyObject(…);

                void operator()(const blocked_range<size_t>&Range) const
                {
                        <Thread Private Data>
                        for (size_t i = Range.begin(); i != Range.end(); ++i)
                        {
                                <Something to be executed in parallel>
                        }
                }
};
```

Parallel decay – Single Generator

A first try was done parallelizing the decay phase of the generation process as follow:

- The body object is initialized with the initial conditions (x,y,z,t) and the Generator to employ
- A vector of events is generated by the functor

```
void operator()(const blocked_range<unsigned long>& Range) const {
        for (unsigned long i = Range.begin(); i < Range.end(); ++i)
        {
                EvtVector4R p_init(EvtPDL::getMass(EvtPDL::getId("Upsilon(4S)")),
                                   0.0,0.0,0.0);

                (*EventVector)[i] = EvtParticleFactory::particleFactory(
                                   EvtPDL::getId("Upsilon(4S)"), p_init);
                (*EventVector)[i]->setVectorSpinDensity();
                EventGenerator->generateEvent((*EventVector)[i], *t_init);
        }
}
```

First Try – Results

A single generator per thread pool is a bottleneck

EvtGen itself must be «parallelized» in some way but:

- There is a large use of static classes and properties
- Data produced in the Fortran part of the code is passed through «Common blocks» (memory shared by code of the same program unit)

A Body Object with a local Event Generator cannot work!

A different parallelization pattern has to be used.

Profiling [1/2]

Profiling a serial execution of the code to produce some thousands of events, we get:

```
 %   cumulative   self              calls   self     total
24.75      4.02     4.02 91182679      0.00     0.00
          EvtBtoXsgammaFermiUtil::FermiExpFunc(…)
12.52      6.05     2.03 67624537      0.00     0.00
          EvtBtoXsgammaKagan::DeltaFermiFunc(…)
 10.85     7.81     1.76 1217901394     0.00     0.00
          std::vector<double, std::allocator<double> >::operator[](…) const
  7.24     8.98     1.18 67624537      0.00     0.00
          EvtBtoXsgammaKagan::Delta(…)
  6.04     9.96     0.98 91182550      0.00     0.00
          EvtBtoXsgammaKagan::FermiFunc(…)
  5.24    10.81     0.85 88725714      0.00     0.00
          EvtItgThreeCoeffFcn::myFunction(…) const
[…]
```

More than $^2/_3$ of the time is spent doing math (Integrating Fermi and Delta Functions)

Profiling [2/2]

- Profiling gave us that Hadronic Mass Spectra computation is the most time consuming procedure
- There's room for a performance increase if we parallelize function integration

How to parallelize legacy code?

- TBB allows fine grained management of threads and tasks, but requires a complete rewrite of the code to become «TBB compliant»
- OpenMP give less freedom to the programmer but can be easily injected inside existent code

Implementation with OpenMP

EvtBtoXsgammaKagan::computeHadronicMass is the EvtGen procedure that calculate Hadronic Mass Spectra

It contains a quite long setup followed by a for cycle where the Branching Fraction is calculated: this is the section that have to be parallelized.

How to proceed with OpenMP parallelization?

- Identifying objects dependencies
- Creating separated objects local to each thread
- Reducing the result of the loop

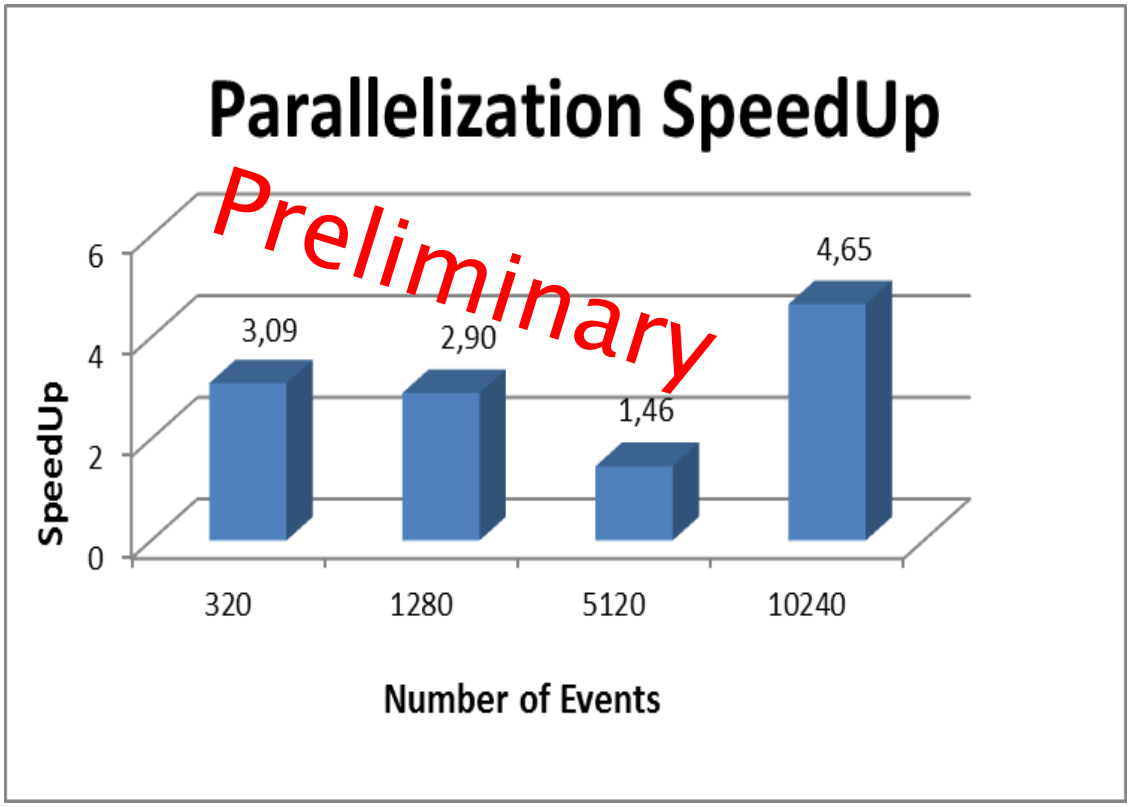
Performances [1/2]

Comparison between serial execution and the OpenMP implementation

Measurements were done on a single Intel Xeon E5630 system (4 cores, 2 HT per core), with 12GB of RAM

The parallelization pattern increases legacy code performances

Note:
Plotted measures are correlated!



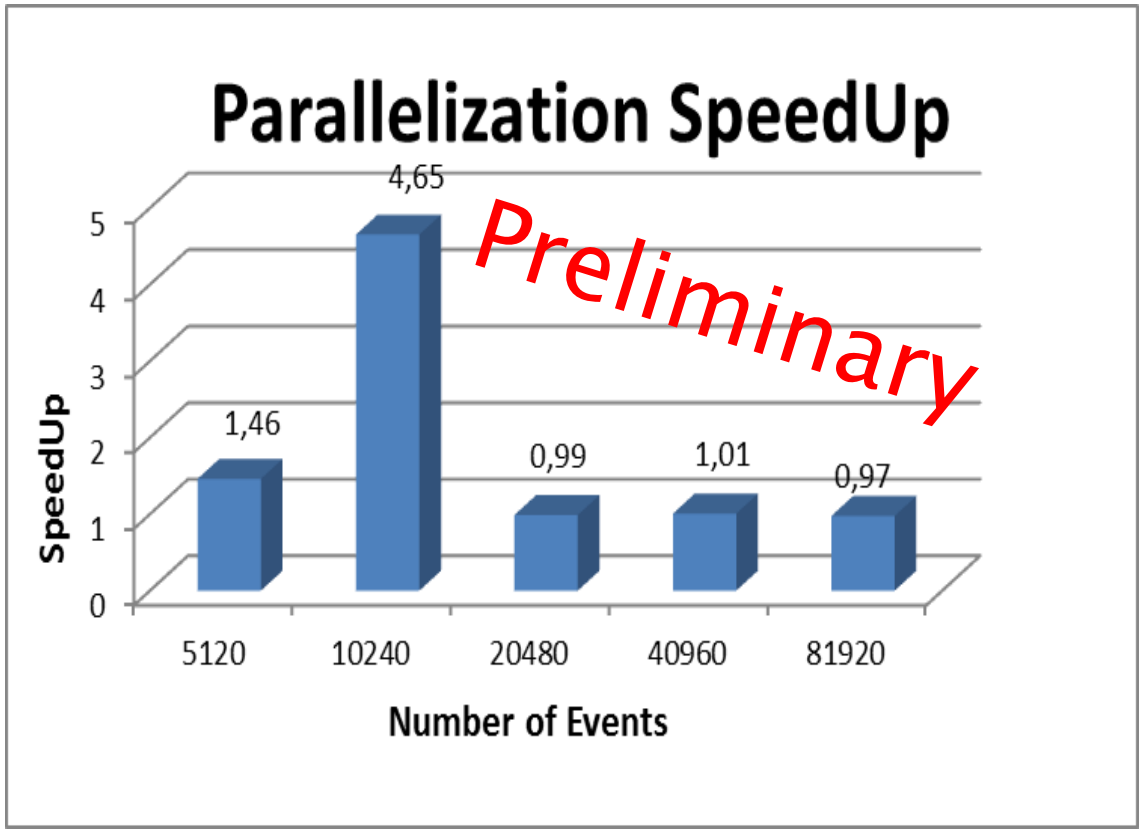
Performances [2/2]

Comparison between serial execution and the OpenMP implementation

Moving beyond the setup used to profile the application, the SpeedUp quickly falls down.

Other parts of the code become dominant in CPU usage

Note:
Plotted measures are correlated!



Conclusions [1/2]

EvtGen parallelization provided some usefull information:

- Legacy Fortran code can be executed in a parallel environment like OpenMP or TBB
- TBB cannot be profitably used in modules like EvtGen (static classes/properties) if we don't want to rewrite major part of code
- OpenMP can be employed to paralelize sections of legacy code, with minor modification
- OpenMP solution can provide a quite good performance gain

Conclusions [2/2]

EvtGen example also suggest a pattern that can be adopted to parallelize legacy modules:

- Identify a set of tipical use cases
- Profile the module on those cases
- Identify the most time consuming part of the code
- Parallelize it via OpenMP

Unfortunately, this add a new line of work to the Framework R&D activities.

Thanks
For your attention!