Experience and perspective in parallelization of data analysis

Alfio Lazzaro
Openlab/CERN

SuperB Computing Workshop @ Ferrara
March 9th-12th, 2010

Data analysis

- Targets (2 main areas)
  - Events selection: set of cuts applied on discriminant variables
  - Signal/Background discrimination and parameters extraction
- Techniques
  - Cuts Optimization (Bump Hunter), Fisher Discriminant, Neural Networks (NN), Boosted Decision Trees (BDT)…
  - Maximum Likelihood (ML) fits
- Complexity
  - Simple 1D fit / Cut&Count analyses
  - Multivariate analyses for signal/background discrimination
  - Angular/Dalitz Plot analyses
  - Time Dependent analyses

A bit of history: first measurements

- I jointed Babar in 2000 (undergraduate)
  - First analyses
  - Several packages in Fortran/C/C++
  - A lot of work spent for making comparisons: improvements in the code, but a lot of redundancies (due to several groups making the same analysis)…
- Simple analyses (first measurements), with low efficiency and low systematics
  - Gain confidence in the techniques
  - Try to optimize without using Monte Carlo events, to reduce possible systematic effects

A bit of history: next measurements

- After the first analyses, we increase their complexity
  - Higher efficiency
  - Higher background suppression
  - Better exploitation of the data sample
    - Particular interesting when comparing results with Belle
      - Smaller errors, even with less data than Belle
- Higher complexity requires advanced packages
  - Flexibility, performance, simplicity
- Packages for Multivariate analysis (MVA):
  - StatPatternRecognition (SPR), for example used in the Babar PID: NN, BDT
  - TMVA, integrated inside ROOT: NN, BDT, Fisher Discriminant, …
  - RooFit, integrated inside ROOT: ML fits

Examples

- The charmonium analysis for sin 2β measurement
  - Loose cuts to increase the efficiency
  - Better variables for signal/background discrimination
  - Mode decay channels together: simultaneous fits
- Measurement of sin 2β/sin 2α in charmless decays
  - Same considerations of charmonium analyses, but higher background contamination
  - Dalitz plot analyses
  - Complex variables for signal/background discrimination, including eventual correlation effects between discriminating variables

SuperB case

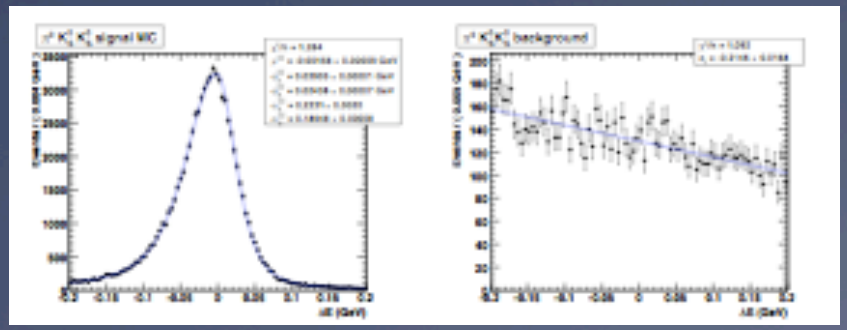
- Same considerations for SuperB analyses:
  - Easy analyses in the first period
  - More complex analyses later
- However: naively we can extrapolate the Babar analyses (same techniques) in case of SuperB statistics
  - At least 2 orders of magnitude more data expected
  - Analyses can suffer some limitations due to approximations not considered in Babar (e.g. correlation between variables), requiring better treatment
  - Increase in statistics can help in some cases (for example using binned fits instead of unbinned), but in general I don't expect that the complexity will increase linearly with the increase of data sample…
- Most analyses will be *impossible* to do because of CPU-time limitations
  - Currently running in hours; in SuperB will be weeks!
  - Increase in CPU performance will NOT help if we use the same algorithms
  - Clearly we can simply the analysis somehow (tighter cuts), but it is not always what we desire…

The two areas

- Event selection well performed in parallel using PROOF (data parallelism)
  - Still large dataset, good data parallelism
  - Few examples of PROOF usage in Babar, largely used by Alice
- Fitting procedures (or similar techniques for results extractions) require a different approach: algorithm parallelism
  - Small samples, intensive CPU-time algorithms
  - Few examples on the market, still a lot to do
- Parallelization is mandatory in a many-cores era
- Different approaches
  - In the follow I will talk about techniques for results extractions, mainly ML fits

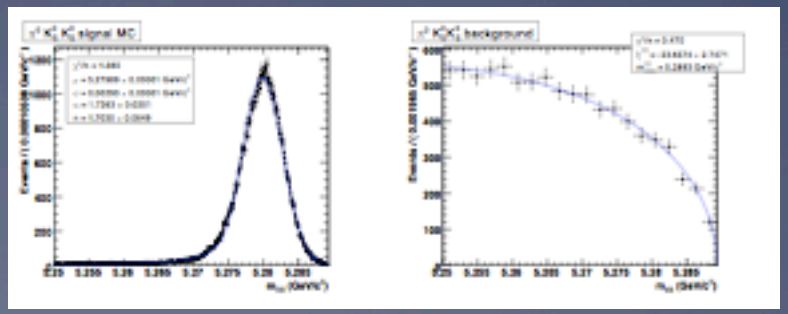
Maximum likelihood fit

- First selection of the sample using loose cuts
  - Reduce background in the final sample keeping high the signal efficiency
- Identify variables with small correlations and good discrimination signal/background (components)
  - Parameterization of the PDFs using control samples (based on MC events or sideband data)
  - Examples: 2 variables, 2 components (generally we have more variables and components)



Signal Background

Signal Background



Maximum likelihood fit

- In Maximum Likelihood fits we have to maximize the likelihood function

$$\mathcal{L} = \frac{e^{-\sum_{j=1}^{s} n_j}}{N!} \prod_{i=1}^{N} \sum_{j=1}^{s} n_j \mathcal{P}_j^i.$$

j species (signals, backgrounds)
n_j number of events for specie j
P_j probability
N number total of events to fit

- In general we minimize the Negative Log-Likelihood Function

$$-\ln \mathcal{L} \equiv NLL = \ln\left(\sum_{j=1}^{s} n_j\right) - \sum_{i=1}^{N} \left(\ln \sum_{j=1}^{s} n_j \mathcal{P}_j^i\right)$$

- The minimization is performed as function of free parameters: n_j number of events, parameters of P_j

Minimization

- The most largely used algorithm for minimization is Minuit (F. James, 1972)
- MINUIT uses the gradient of the function to find local minimum (MIGRAD), requiring
- The calculation of the gradient of the function for each free parameter, naively

$$\left.\frac{\partial NLL}{\partial \hat{\theta}}\right|_{\hat{\theta}_0} \approx \frac{NLL(\hat{\theta}_0 + \hat{\mathrm{d}}) - NLL(\hat{\theta}_0 - \hat{\mathrm{d}})}{2\hat{\mathrm{d}}}$$

2 per derivative

- The calculation of the covariance matrix of the free parameters (which means the second order derivatives)
- The minimization is done in several steps moving in the Newton direction: each step requires the calculation of the gradient

Minimization

- In case of NLL function, it requires the calculation of the function for each free parameter in each minimization step
  1. Many free parameters means slow calculation
  2. Remember the definition of *NLL*

$$NLL = \ln\left(\sum_{j=1}^{s} n_j\right) - \sum_{i=1}^{N}\left(\ln \sum_{j=1}^{s} n_j \mathcal{P}_j^i\right)$$

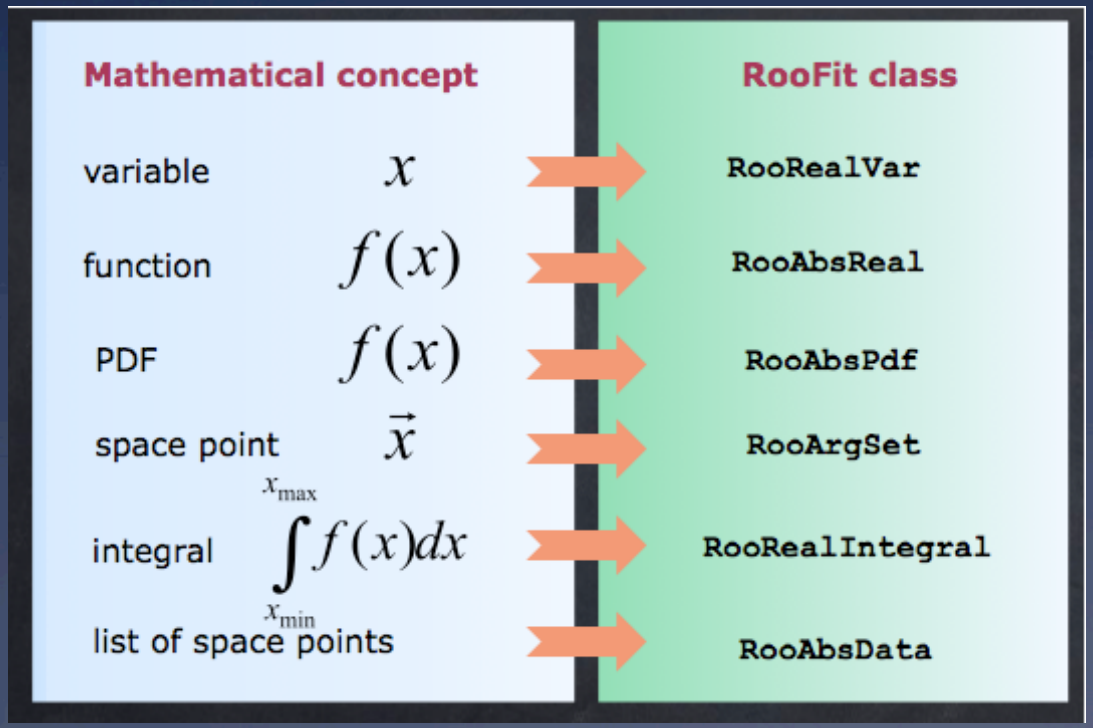
  - The computational cost scales with the *N* number of events in the input sample
  3. Note, also, that P_j needs to be normalized (calculation of the integral) for each iteration, which can be a very slow procedure if we don't have an analytical function
  4. Further complexity introduced by convolutions of the PDFs
- Complex fits take several hours (or days)! (example with 2M events, 4 variables, ~100 free parameters, like Dalitz plot analysis of *D* mesons)
  - Usually you have to run several fits for your tests

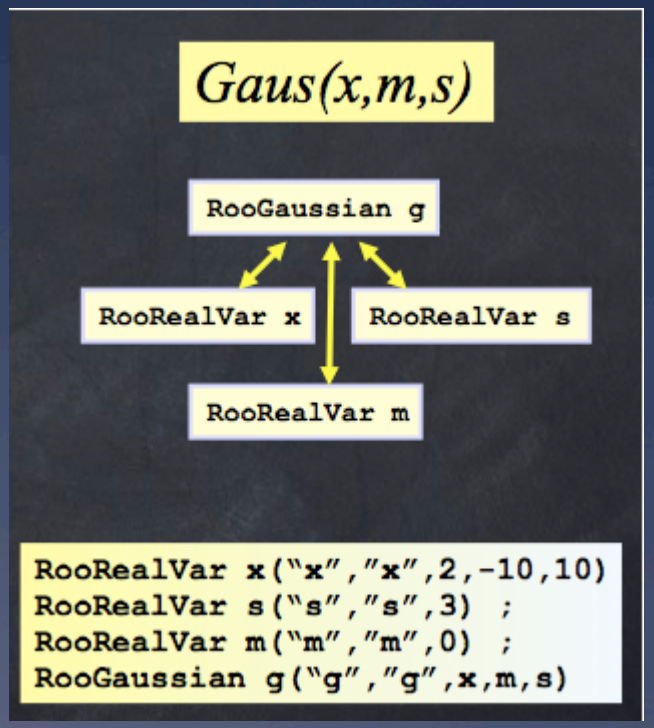
RooFit

- The most used analysis techniques is based on Maximum Likelihood fits
- Wouter Verkerke and David Kirkby developed a set of classes for ML fit, called RooFit
  - Based on ROOT classes (in C++)
  - Core classes for probability density functions (PDFs) integration, event generation, likelihood function definition, convolution, data representation and visualization, binned and unbinned fits
  - Set of several PDFs (Gaussian, Polynomials, Argus function…), with classes to combine them (Sum, Product, Simultaneous PDFs)
- Several programs developed on top of RooFit for a better configuration
  - High flexibility, easy to use
- Since 2006 RooFit is inside ROOT
  - Good support by Wouter
  - A lot of expertise in several experiments (Babar, Belle, CDF, D0, LHC experiments, …)
  - It is the biggest package inside ROOT!

RooFit

- Mathematical concepts are represented as C++ objects





- Provides a factory to auto-generates objects from a math-like language

```
"Gaussian::g(x[2,-10,10],m[0],s[3])"
```

- Chosen as base package for RooStats, advanced statistical tool used in LHC experiments

Babar fitting package survey

- In 2005 there was a survey on the available fitting tools inside the experiment:
  - Several meetings with presentation from the authors
  - The survey was based on analyses targeted to ICHEP 2004, Moriond 2005 and Lepton-Photon 2005
  - Statistics:
    - PAW and Minuit: 18%
    - ROOT: 22%
    - RooFit: 30%
    - Multi-purpose fitter (majority are based on RooFit): 30%
  - A update in 2007 (see http://www.slac.stanford.edu/BFROOT/www/doc/Workshops/2007/BaBar_RooFit/Agenda.html) showed that the fraction of analysis using RooFit is higher
- I think that concentrating on a single package avoid to reinvent a wheel each time a new analysis is started
  - RooFit now is very mature
  - **In the following I will use RooFit as the standard package for ML fits**

Why focus on RooFit

- In data analysis there is not a general common framework (like reconstruction) for different analysts'
  - In general everybody wants the "power" to obtain the final results, i.e. his own version of data analysis code
- This means a "plethora" of programs
  - Not always based on the same base-code (different languages, Matlab, different algorithms…)
- Advantage: possible to make comparisons to spot bugs out
- Disadvantage: "sometimes" all the versions are not well optimized
  - Last year Babar sent a request to do a parallel version of a fitting code to ROOT people. The programs had taken about 1 hour. After few optimizations (not parallelization) now it takes 3 minutes…
  - You can image the possible scenario if we move to parallel version of the code that are, by definition, more difficult to develop and debug…
- Concentrate the efforts in one single package can make life easier…

RooFit/Minuit Parallelization

- RooFit implements the possibility to split the likelihood calculation over different threads
  - Likelihood calculation is done on sub-samples
  - Then the results are collected and summed
  - You gain a lot using multi-cores architecture over large data samples, scaling almost with a factor proportional to the number of threads
- However, if you have a lot of free parameters, the bottleneck become the minimization procedure
  - Split the derivative calculation over several MPI processes
  - Possible to apply an hybrid parallelization of likelihood and minimization using a Cartesian topology (see my CHEP09 proceeding, to be published on …)
    - Improve the scalability for case with large number of parameters and large samples
- Code already inside ROOT (since 5.26), based on Minuit2 (the OO version of Minuit)

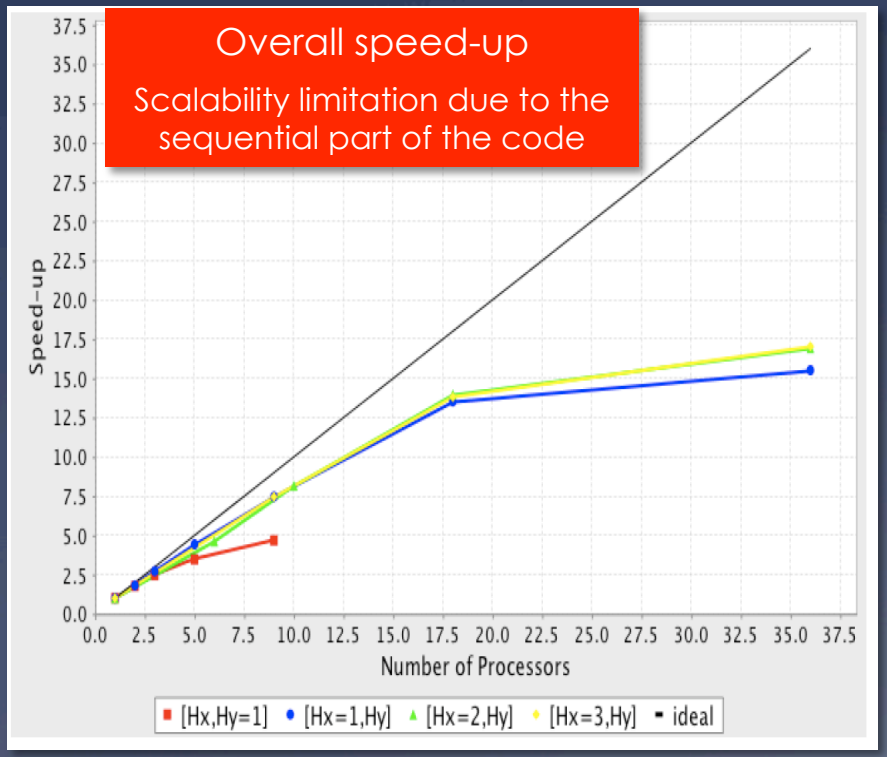
Tests

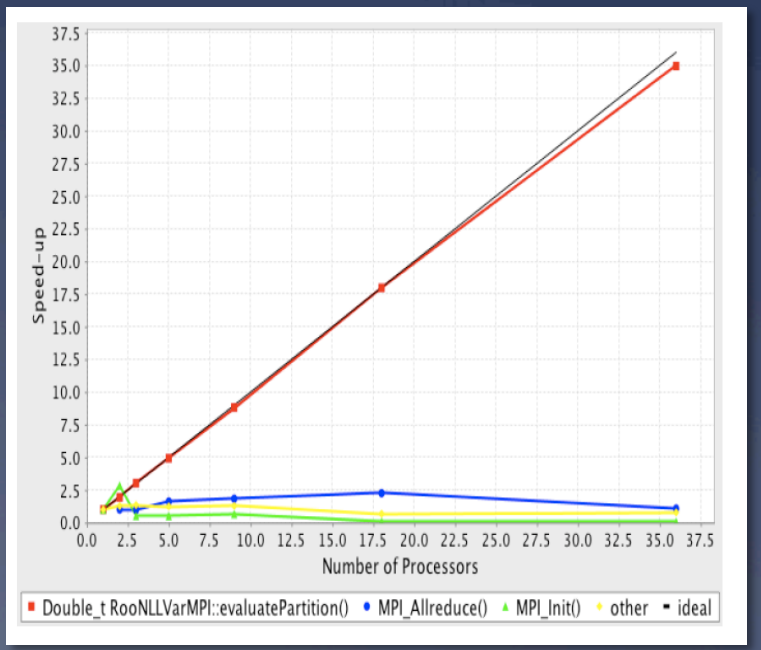
Test @ INFN CNAF cluster, Bologna (Italy)

3 variables, 600K events, 23 free parameters

PDFs per each variable: 2 Gaussians for signal, parabola for background

Sequential execution time (Intel Xeon @ 2.66GHz): **~80 minutes**





`RooNLLVarMPI::evaluatePartition()` does the NLL calculation: excellent scalability

Scalability

- Better scalability in case of complex fits (lot of events, lot of free parameters)
  - Good example of Gustafson's law application
- However there are fits where the main limitations are
  - Integral calculation: once for each step of iteration, depending on the parameters of the PDFs (otherwise cache the integral)
  - Convolutions (based on FFT): once for each step of iteration
- Other limitations are the events generation (MC simulated experiments) and complex fits that require a scan of the NLL around the minimum (due to possible local minimums)
- Note that from some tests we did, rounding problems can occur in Minuit for large samples (over 1 million of events)
- Same considerations can be used for analyses in other experiments, like LHCb analyses

Most problematic case: Dalitz Plot analyses

- Properly take in account all physics phenomena
  - Ideally all analysis with decays with more that 2 daughters final states should do a Dalitz Plot analysis
  - In case of law statistics we can use quasi-two body approximation
    - Not valid anymore in case of |SuperB for most analysis
- Complex analysis: several free parameters, complex functions, 2D PDFs, Time Dependent analyses, Convolutions to take in account detector resolution
- At the moment this is the case of Charm physics in Babar (spectroscopy, D mixing, …)
  - Fit takes weeks (good example: Antimo Palano's analyses)

Future plans

- Improve RooFit
  - Most of the time is spent on data reading (based on TTree)
  - Improve events generation and integral calculation
- Event generation can be parallelized
  - Useful for MC simulated events
  - Integral calculation based on MC method (in particular in case of multidimensional functions)
- Parallel FFT for convolutions
- Minuit doesn't guarantee good scalability and it is particularly sensible to multiple minimum
  - Need to use another algorithm for optimization, such as Genetic Algorithms
    - Genetic Algorithms are highly parallelizable, they can help during the minimization, running on GPUs
  - Run directly Minuit on GPUs (see tomorrow morning talk by Karen Tomko)

MC Experiments: Parallelization

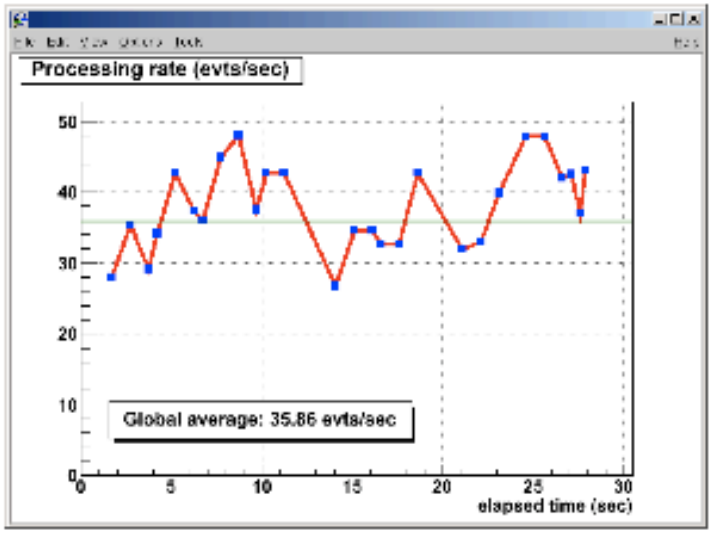
- This is done inside RooFit) using PROOF (implemented by Wouter)
  - Used different instances of the random generator (TRandom3) , setting different seeds for each MC experiment generation
    - Extracts different stream using different seed
    - Allows to reproduce the event generation given the seed
- Random generation based on TRandom3:
  - Based on Mersenne-Twister technique (large period 2**19937-1)
  - Static generator declared inside the class RooRandom (RooFit interface to ROOT event generators)

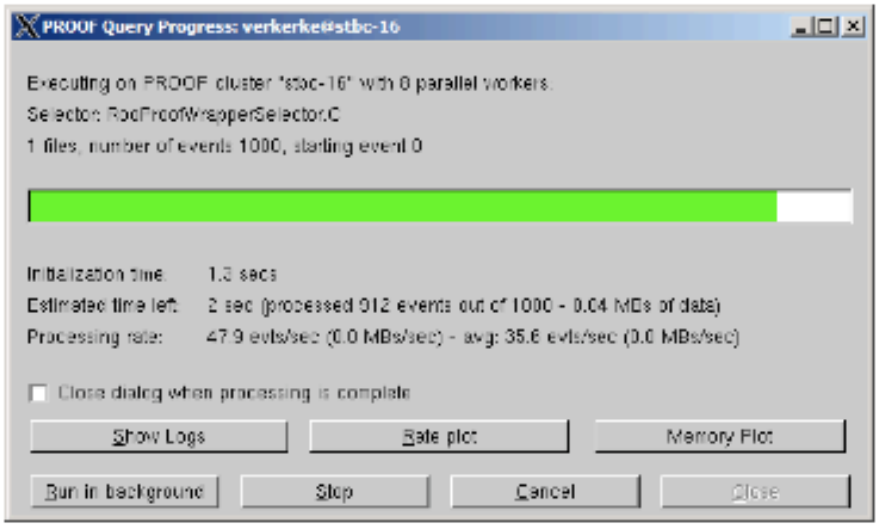
RooFit-PROOF (Wouter's slide)

Demo of parallelization with PROOF-lite

- Example – Factor 8 speed up on a dual-quad core box.
  - Works with out-of-the box ROOT distribution
  - Also: Graceful early termination when users presses 'Stop'

```
RooStudyManager mcs(*w,gfs) ;
mcs.run(1000) ; // inline running
mcs.runProof(1000,"") ; // empty string is PROOF-lite
mcs.prepareBatchInput("default",1000,kTRUE) ;
```





However…

- TRandom3 is not a Parallel Random Generator (PRNG)
  - Using different instances with different seeds can introduce correlations between the streams
  - Eventual effects of correlation should be negligible for the Mersenne Twister of TRandom3, since his period is large
- In any case we need a PRNG in case we are not doing MC-experiments (e.g. integral calculation)
- Working on an implementation using the package Tina RNG (TRNG library)
  - It looks like a very interesting package, with a good documentation and some examples in MPI, OpenMP and TBB
  - Last version: Feb 2010 (so it is still well supported)
  - There is a proposal to include it in the new C++ standard
  - http://trng.berlios.de/

Random Generations

- Comparing sequential TRandom3 and TRNG to generate 100M events:
  - TRandom3: 5.53 seconds
  - TRNG: 5.56 seconds
- Move to parallel version of the code using TRNG in case of accept/reject (in seconds):

1T	8.98
2T	4.53
4T	2.31
8T	1.40

- Good results, working on a better integration in RooFit/ ROOT
  - Note that neither RooFit nor ROOT are thread-safe code…

Conclusions

- Currently efforts are to parallelize the current packages (RooFit, Minuit)
  - Fit and event generation parallelization
- Consider data analysis software as the experiment software framework
  - Concentrate all the efforts in few packages (I would say just one…)
  - At the moment there is not a enormous demand for parallelization
    - AFAIK about 10 groups are working on data analysis parallelization
- Individuate the most time-expensive part and write them as kernel function (plugins) that you can run in parallel inside you current program
  - Not forget optimization!
  - Not easy to run everything in parallel, unless you want to rewrite everything!
  - Run the different kernel exploring all kind of possible parallelization on the market (GPUs, multi-cores, vectorization, …)
  - Use MPI/OpenMP/TBB techniques (new C++ standard can be part of the game)

Conclusions

- In some cases our algorithms are old (like CERNLIB Minuit)
  - Not good algorithms to scale to many-cores
  - Need to think about how we can improve them
- Other data analysis techniques (NN, BDT, …) are good candidate for parallelization
- SuperB is challenging for data analysis software, but we are lucky:
  - Most of the code is already available and we can predict what will be the future scenario
    - Let's start to work! Think Parallel!
  - Common effort for all experiments for data analyses