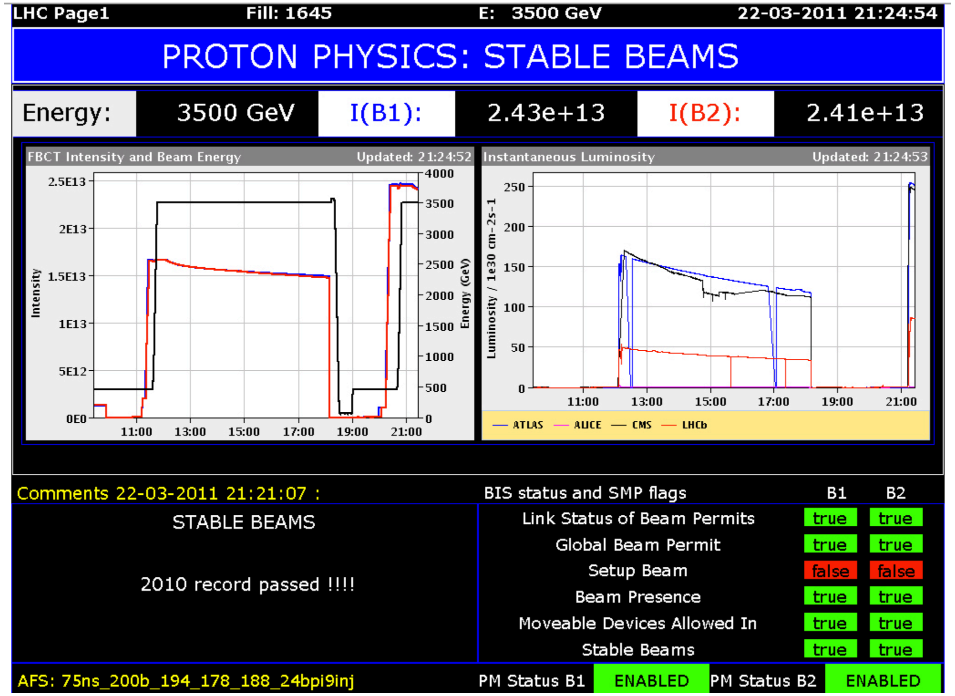
Evaluation of Likelihood Functions for Data Analysis on GPUs

Sverre Jarp, *Alfio Lazzaro*, Julien Leduc,
Andrzej Nowak, Felice Pantaleo
European Organization for Nuclear Research (CERN), Geneva, Switzerland

The 12th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, Anchorage (Alaska), USA
May 20th, 2011

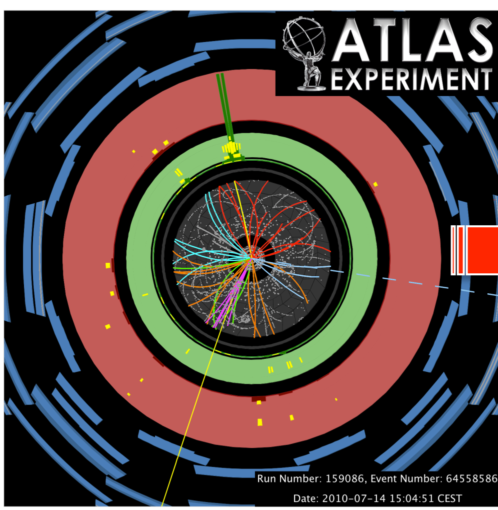
LHC activities

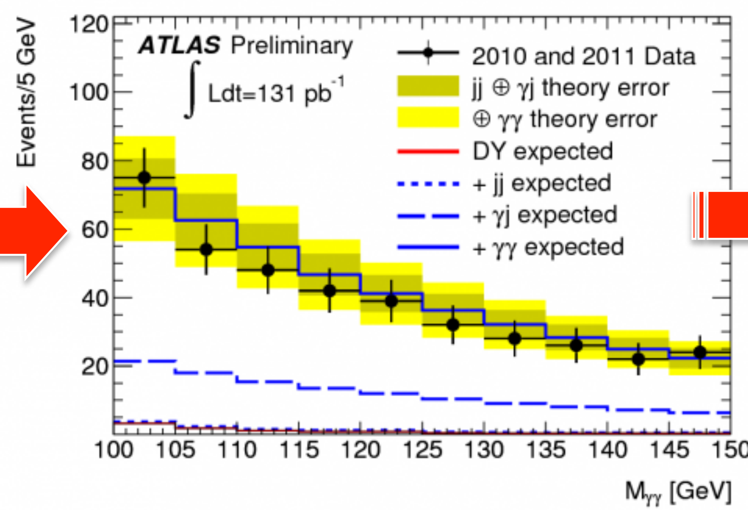
- The Large Hadron Collider (LHC) at CERN started its activity in 2009 with collisions of protons @ 3.5 TeV per beam
  - The highest energy reached in a particle accelerator for smashing protons
  - Operations will run through to the end of 2012, with a short technical stop at the end of 2011

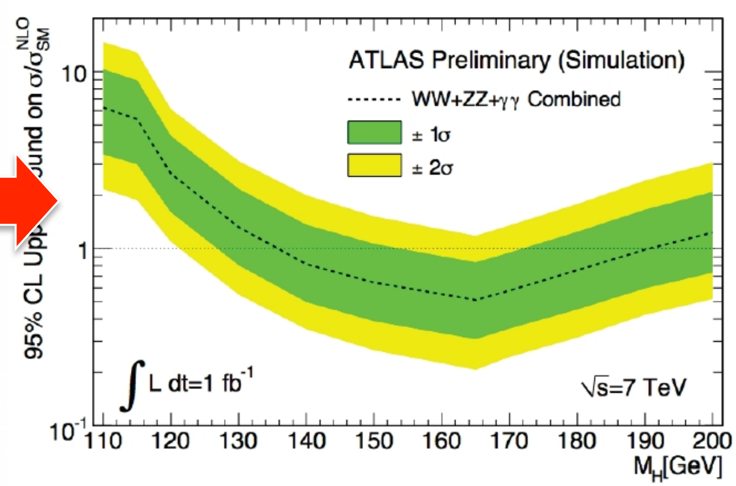


- 4 big experiments collecting the results of the collisions
  - > 10^7 collisions per seconds
  - About 200 collisions (events) recorded per second per experiment: ~300 MB/s (~3 PB/year)

- Huge quantity of data collected, but most of events are due to well-know physics processes
  - New physics effects expected in a tiny fraction of the total events: few tens
- Crucial to have a good discrimination between interesting (signal) events and the rest (background)
  - Data analysis techniques play a crucial role in this “war”







Likelihood-based techniques

- Data are a collection of independent events
  - an event consists of the measurement of a set of variables (energies, masses, spatial and angular variables...) recorded in a brief span of time by the physics detectors
- Introducing the concept of probability $\mathcal{P}$ (= Probability Density Function, PDF) for a given event to be signal or background, we can combine this information for all events in the *likelihood function*

$$\mathcal{L} = \prod_{i=1}^{N} \mathcal{P}(\hat{x}_i|\hat{\theta})$$

N number of events
$\hat{x}_i$ set of variables for the event i
$\hat{\theta}$ set of parameters

- Several data analysis techniques requires the evaluation of $\mathcal{L}$ to discriminate signal versus background events

Maximum Likelihood Fits

- It allows to estimate free parameters over a data sample, by minimizing the corresponding Negative Log-Likelihood (NLL) function

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

s species, i.e. signals and backgrounds
n_j number of events belonging to the species j

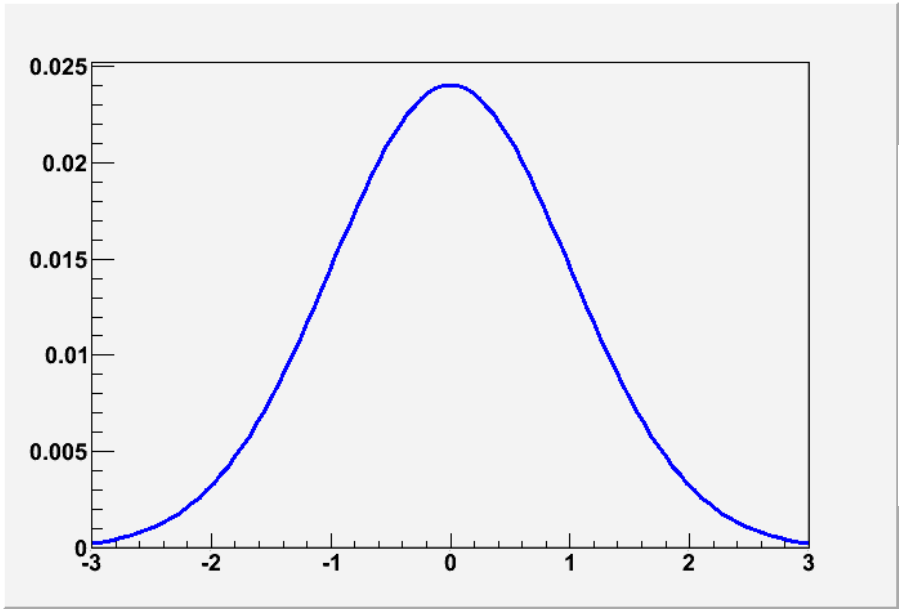
- The procedure of minimization can require several evaluation of the NLL
  - Depending on the complexity of the function, the number of variables, the number of free parameters, and the number of events, the entire procedure can require long execution time
  - Mandatory to speed-up the execution

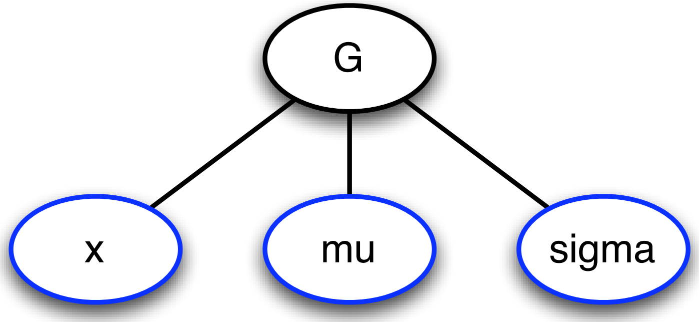
Examples

- In most cases PDFs can be factorized as product of the n PDFs of each variable (i.e. case of uncorrelated variables)

$$\mathcal{P}_j(\hat{x}_i|\hat{\theta}_j) = \prod_{v=1}^{n} \mathcal{P}_j^v(x_i^v|\hat{\theta}_j)$$

Gaussian
$G(x|\mu,\sigma)$



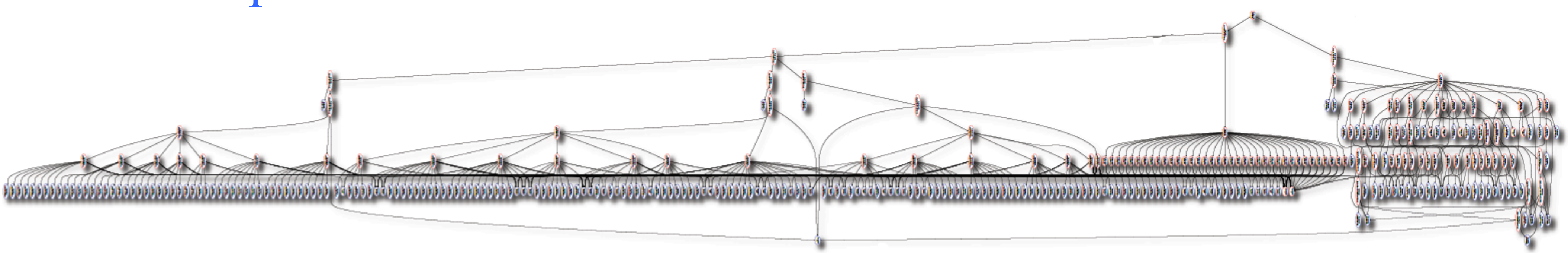


Examples

- In most cases PDFs can be factorized as product of the n PDFs of each variable (i.e. case of uncorrelated variables)

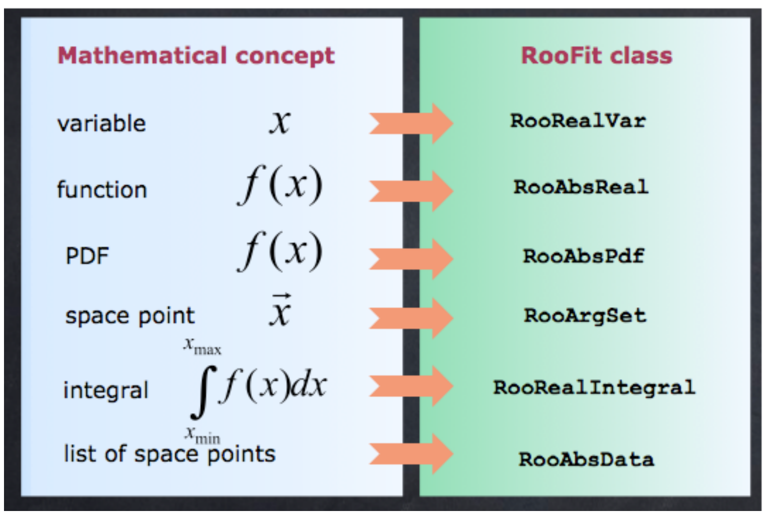
$$\mathcal{P}_j(\hat{x}_i|\hat{\theta}_j) = \prod_{v=1}^{n} \mathcal{P}_j^v(x_i^v|\hat{\theta}_j)$$

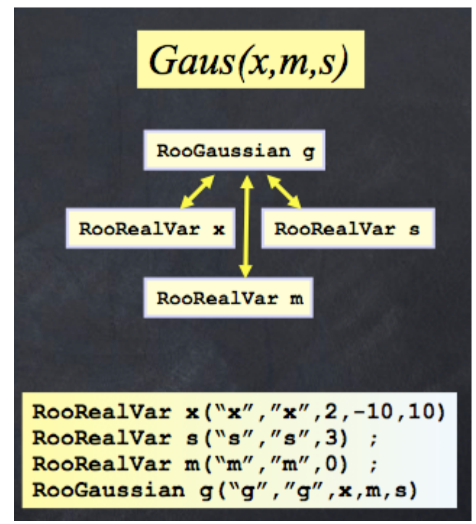
Combined Atlas & CMS Higgs analysis:
12 variables
50 free parameters



Building models: RooFit

- RooFit is communely used in High Energy Physics experiments to define the likelihood functions (W. Verkerke and D. Kirkby)
  - Details at http://root.cern.ch/drupal/content/roofit
  - Mathematical concepts are represented as C++ objects





- On top of RooFit developed another package for advanced data analysis techniques, RooStats
  - Limits and intervals on Higgs mass and New Physics effects

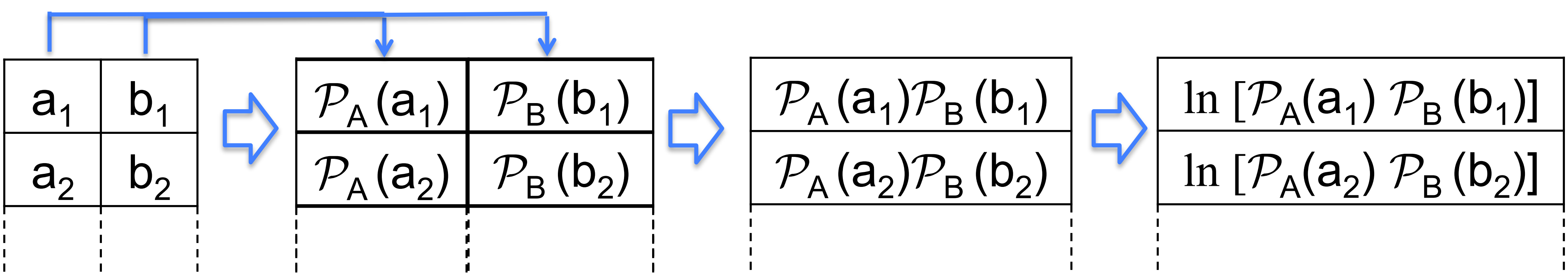
Caveats

- We developed a new algorithm for the likelihood function evaluation to be added in RooFit
  - We don't replace the current RooFit algorithm, which is used for results checking
  - Very chaotic situation: users can implement any kind of model
  - No need to change the user code to use the new implementation
- The new algorithm is optimized to run on the CPU
  - Auto-vectorization by the Intel compiler
  - Parallelization using OpenMP
  - Used as reference for the GPU implementation: "fair" comparison
- All data in the calculation are in double precision floating point numbers
- We target is to use commodity systems (e.g. laptops or desktops), easily accessible to data analysts

Algorithm and parallelization

1. Read all events and store in arrays in memory
2. For each PDF make the calculation on all events
   - Corresponding array of results is produced for each PDF
   - Evaluation of the function inside the local PDF (drawback: require to handle arrays of temporary results: 1 value per each event and PDF)
3. Combine the arrays of results (composite PDFs)
4. Loop over the final array of results to calculate NLL (final reduction)

Ex: $\mathcal{P} = \mathcal{P}_A(a_i)\ \mathcal{P}_B(b_i)$



Parallelization splitting calculation of each PDF over the events (data parallelism) and over the independent PDFs (task parallelism)

OpenMP parallelization

```
// Inline method for the Gaussian PDF calculation,
// defined inside the class RooGaussian
inline double evaluateLocal(const double x,
                            const double mu,
                            const double sigma) const
{
   return std::exp(-0.5*std::pow((x-mu)/sigma,2));
}

// Virtual method for the calculation of the
// Gaussian PDF on a single event
// (this is the original RooFit algorithm)
virtual double evaluate() const
{
   return evaluateLocal(x,mu,sigma);
}


// Virtual method for the calculation of the
// Gaussian PDF on all events
// (new implemented algorithm)
virtual bool evaluate(const RooAbsData& data)
{
   // retrive the data array of values for the variable
   const double *dataArray = data.GetDataArray(x.arg());
   // check if there is an array for the variable
   if (dataArray==0)
     return false;

   // retrive the number of events
   int nEvents = data.GetEntries();
   // retrive the array for the partial results
   double *resultsArray = GetResultsArray();
   double m_mu = mu;
   double m_sigma = sigma;

   // loop over the events to calculate the Gaussian
#pragma omp parallel for
   for (int idx = 0; idx<nEvents; ++idx) {
     resultsArray[idx] = evaluateLocal(dataArray[idx],
                                       m_mu,m_sigma);
   }

   return true;
}
```

- Only data parallelism
- Take benefit from the code optimizations
  - Inlining of the functions
  - Data organized in C arrays, perfect for vectorization
- Sequential algorithm runs 4.5x faster than the original RooFit implementation
  - 1.8x from SSE vectorization (additional 12% using AVX on Intel Sandy Bridge)
- Very easy parallelization with OpenMP
- Final reduction executed in parallel, using double-double summation algorithm to reduce rounding effects

Complex Model Test

$$n_a[f_{1,a}G_{1,a}(x) + (1 - f_{1,a})G_{2,a}(x)]AG_{1,a}(y)AG_{2,a}(z) + n_bG_{1,b}(x)BW_{1,b}(y)G_{2,b}(z) + n_cAR_{1,c}(x)P_{1,c}(y)P_{2,c}(z) + n_dP_{1,d}(x)G_{1,d}(y)AG_{1,d}(z)$$

17 PDFs in total, 3 variables, 4 components, 35 parameters

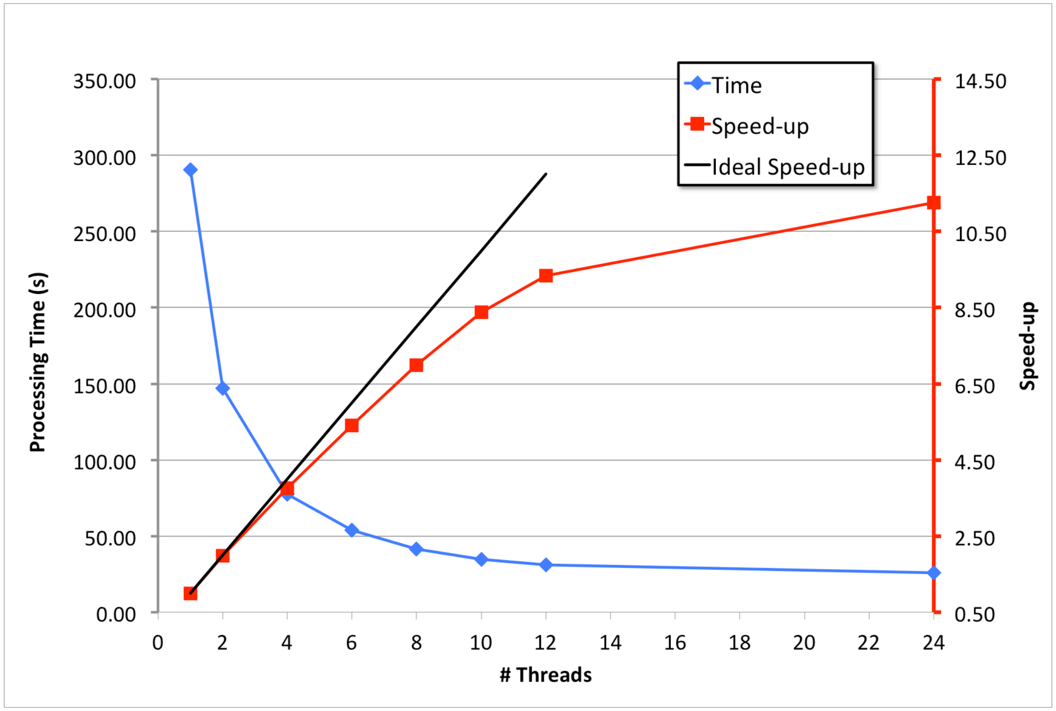
- G: Gaussian
- AG: Asymmetric Gaussian
- BW: Breit-Wigner
- AR: Argus function
- P: Polynomial

40% of the execution time is spent in exp's calculation

Note: all PDFs have analytical normalization integral, i.e. >98% of the sequential portion can be parallelized

Test on CPU in parallel

- Dual socket Intel Westmere-based system: CPU @ 2.67GHz (12 physical cores, 24 hardware threads in total), 10x4096MB DDR3 memory @ 1333MHz
- Linux 64bit, Intel C++ compiler version 12.0.2
- 100,000 events
- Data is shared, i.e. no significant increase in the memory footprint
  - Possibility to use Hyper-threading (about 20% improvement)
- Limited by the sequential part, OpenMP overhead, and memory access to data



GPU Implementation (CUDA)

- Data parallelism: a thread per each event and each PDF
- Task parallelism: running in parallel the kernel for the independent PDFs
  - Require synchronization in case of composite PDFs, using *streams*
- Data is copied on the GPU once (synchronous)
- Results for each PDF are resident only on the GPU
  - Arrays of results are allocated on the global memory once and they are deallocated at the end
    - Minimize CPU ⇔ GPU communication
    - Re-usage of the values in case a PDF doesn't change in consecutive calls
  - Only the final results are copied on the CPU for the final reduction to compute *NLL*, done on the CPU

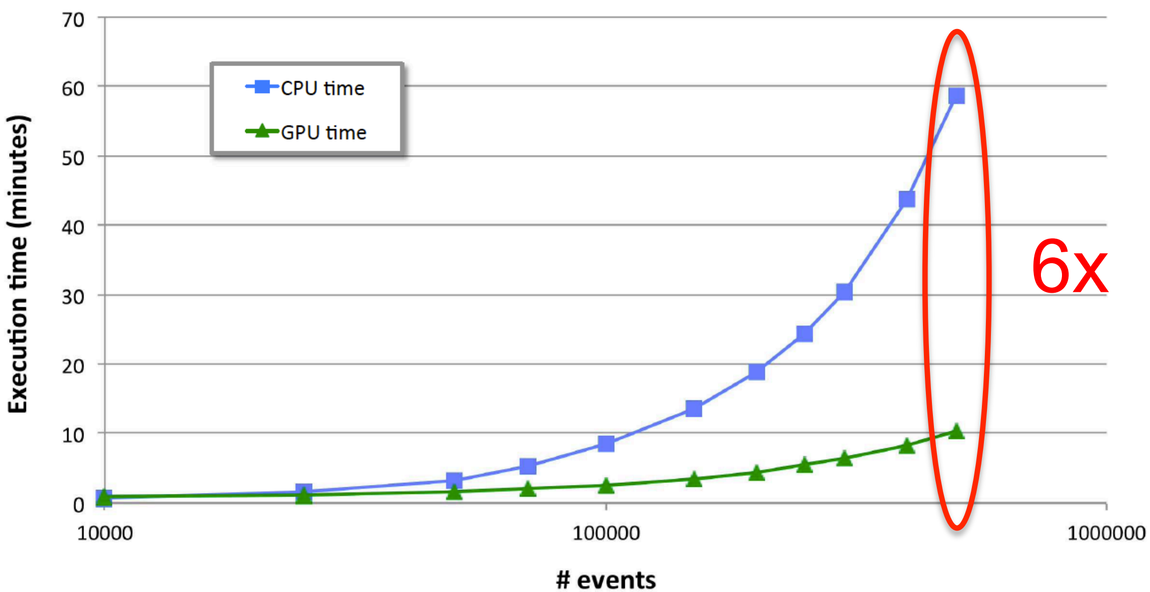
GPU Test environment

- PC (host)
  - CPU: Intel Nehalem @ 3.2GHz: 4 cores – 8 hardware threads
  - Linux 64bit, Intel C++ compiler version 11.1

- GPU: ASUS nVidia GTX470 PCI-e 2.0
  - Commodity card (for gamers)
  - Architecture: GF100 (Fermi)
  - Memory: 1280MB DDR5
  - Core/Memory Clock: 607MHz/837MHz
  - Maximum # of Threads per Block: 1024
  - Number of SMs: 14
  - CUDA Toolkit 3.2
  - Power Consumption 200W
  - Price ~$340

GPU performance

- Device algorithm performance using a linear polynomial PDF and 1,000,000 events
  - 112 GFLOPS (not including communications), about 82% of the peak performance (double precision)
- Comparison using our benchmark model
  - OpenMP runs on the 4 threads for the CPU reference (3.6x speed-up with 500,000 events)



@ 500,000 events:
68% device kernels
21% host execution
11% communications

Conclusion

- Implementation of the algorithm in CUDA required not so drastic changes in the existing RooFit code
  - Up to a factor 6x with respect to OpenMP with 4 threads
  - GPUs behaves better with more events, as expected
- Note that our target is running fits at the user-level on the GPU of small systems (laptops), i.e. with small number of CPU cores and commodity GPU cards
  - Main limitation is the double precision
  - No limitation due to CPU⇔GPU communication
- Soon the code will be released in the standard RooFit (discussion with the authors of the package ongoing)

Future developments

- Implement an OpenCL version
- Concurrent execution on CPU with OpenMP and CUDA/OpenCL on the GPU
- Including MPI for complex models to run on multiple nodes (for data and task parallelism)

CERN openlab

- CERN openlab is the only large-scale structure at CERN for developing industrial R&D partnerships
  - www.cern.ch/openlab-about
- Divided in competence centers
  - HP: wireless networking
  - Intel: advanced hardware and software evaluations and integrations
  - Oracle: database and storage
  - Siemens: automating control systems

CERN openlab

www.cern.ch/openlab

PARTNERS

hp invent

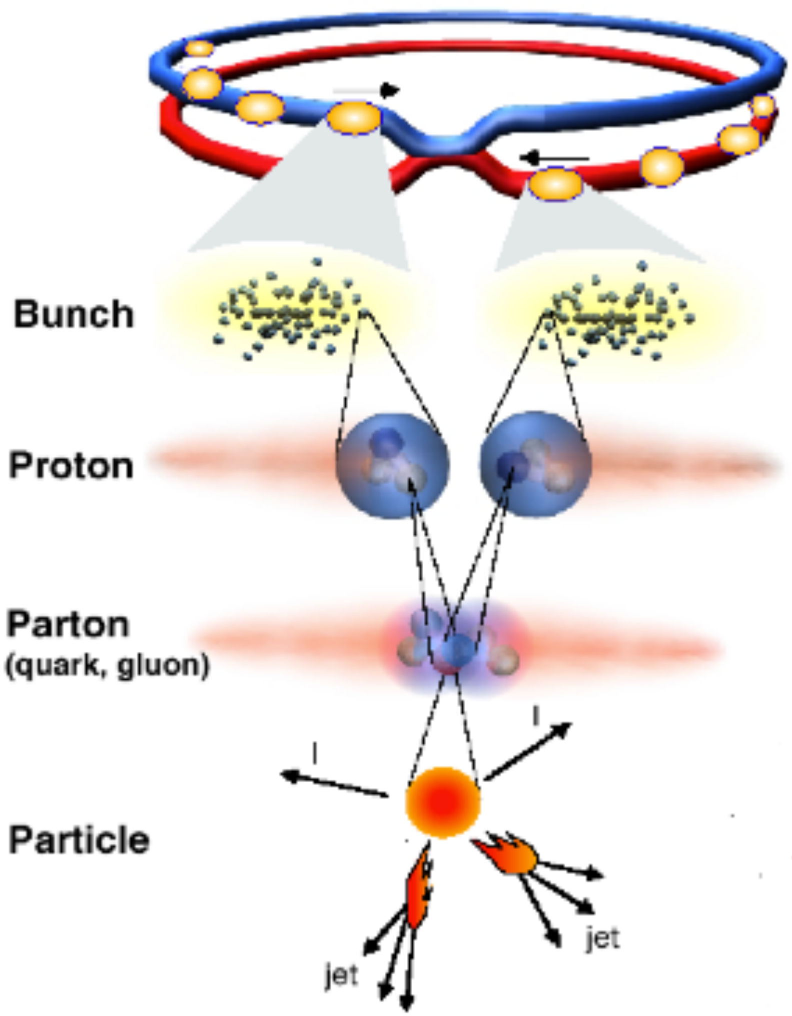
intel

ORACLE

SIEMENS

Backup slides

Data acquisition



- Collisions at LHC
  - Proton-Proton or Pb-Pb
  - 40 MHz crossing rate
  - Collisions $>10^7$ Hz (up to ~50 collisions per bunch crossing)
- Total initial rate: ~1 PB/s
- Several levels of selection of the events (online)
  - Hardware (Level 1), software (Level 2, 3)
  - Final rate for storing: 200 Hz (300 MB/s, ~3 PB/year)

Events are independent: trivial parallelism over the events!

MINUIT

- Numerical minimization of the *NLL* using MINUIT (F. James, Minuit, Function Minimization and Error Analysis, CERN long write-up D506, 1970)
- 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

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

2 function calls per each parameter

  - 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
  - ➪ Several calls to the *NLL*

Likelihood Function calculation in RooFit

1. Read the values of the variables for each event
2. Make the calculation of PDFs for each event
   - Each PDF has a common interface declared inside the class RooAbsPdf with a virtual method which defines the function
   - Automatic calculation of the normalization integrals for each PDF
   - Calculation of composite PDFs: sums, products, extendend PDFs
3. Loop on all events and make the calculation of the *NLL*
   - A *single* loop for *all* events

Parallel execution over the events, with final reduction of the contribution

Variables

Events

	var_1	var_2	…	var_n
1				
2				
…				
N				