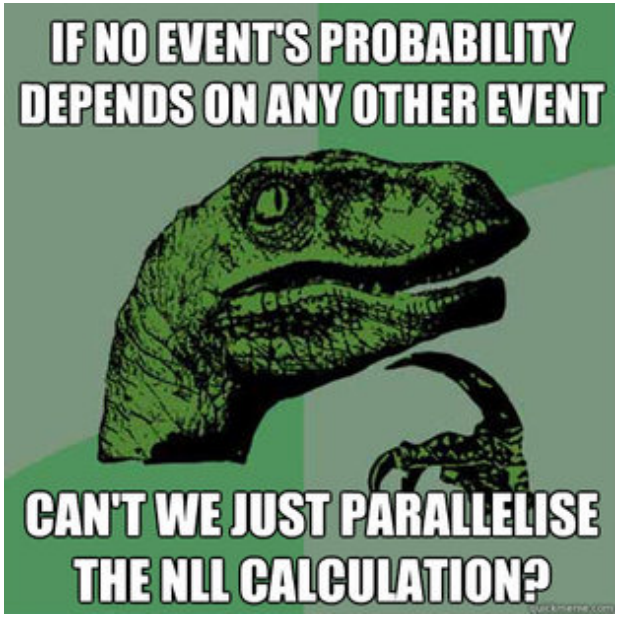
GooFit: Massively parallel function evaluation

- Maximum-likelihood fits to determine physics parameters have execution time at least quadratic in number of fit parameters, and linear in number of events.
- Current data sets are two orders of magnitude larger than those from previous experiments!
- Physics models are also growing in complexity.
- When fits take more than a day to complete, analysis becomes very unwieldy.
- GooFit speeds up fits by exporting the function evaluation onto a massively-parallel GPU with hundreds of processors.
- User-level code designed to be similar to RooFit.

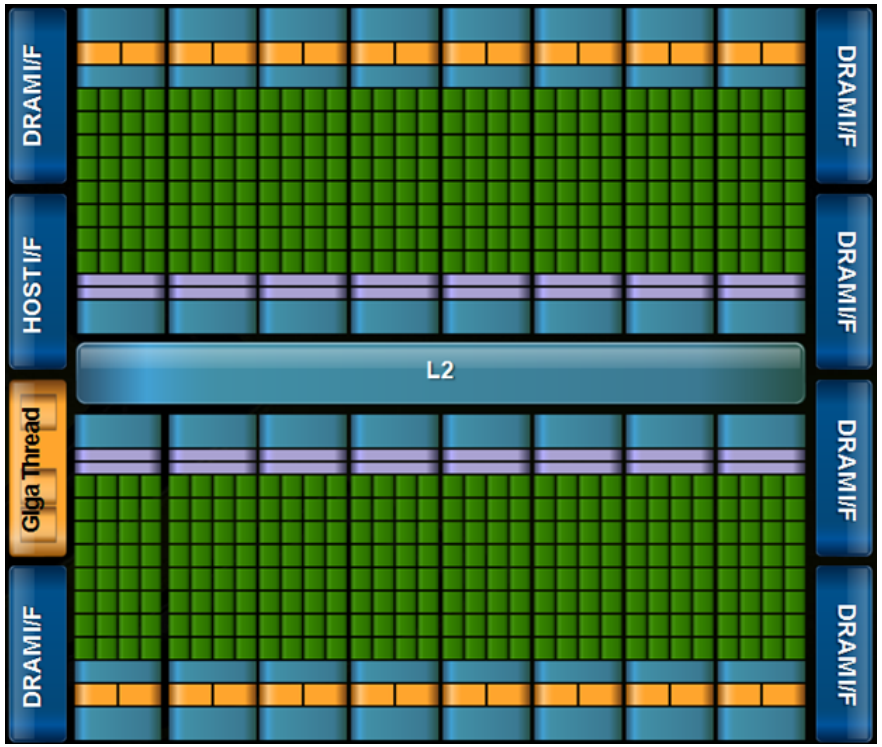


Updates this year

- All parallel kernel launches now made using Thrust library;
- Tested on laptops with nVidia 650M GPUs as well as on workstations with C2050/C2070 boards;
- Works on CPUs under OpenMP.

Target platforms

- Thrust library backends: CUDA or OpenMP.
- CUDA backend requires nVidia GPU and `nvcc` compiler. Tested on Fermi, Tesla, Kepler generations.
- Originally developed for workstation GPUs - also works on laptop onboards!
- OpenMP backend runs on ordinary CPUs - instant parallelism.
- Thrust library hides kernel launch (CUDA) or `#pragma omp` within templated `transform` method. Compile-time switch specifies backend.



Hello, GooFit: Simplest possible example

```
int main (int argc, char** argv) {
  // Variable class stores name, upper and lower limit, optionally
  // number of bins and current error
  Variable* xvar = new Variable("xvar", -5, 5);
  xvar->numbins = 10000;

  // Generate data
  TRandom donram(42); // Borrowed from ROOT
  UnbinnedDataSet data(xvar); // Stores events
  for (int i = 0; i < 10000; ++i) {
    fptype val = donram.Gaus(0.2, 1.1);
    if (fabs(val) > 5) {--i; continue;}
    data.addEvent(val);
  }

  // Create PDF
  Variable* mean = new Variable("mean", 0, 0.1, -10, 10);
  Variable* sigm = new Variable("sigm", 1, 0.1, 0.5, 1.5);
  // FooPdf classes are PDF objects.
  GaussianPdf gauss("gauss", xvar, mean, sigm);
  gauss.setData(&data);

  // FitManager is glue between MINUIT and GPU.
  FitManager fitter(&gauss);
  fitter.fit();
}
```

Existing PDF classes

- **Simple PDFs: Argus function, correlated Gaussian, Crystal Ball, exponential, Gaussian, Johnson SU, polynomial, relativistic Breit-Wigner, scaled Gaussian, smoothed histogram, step function, Voigtian.**
- **Composites:**
  - **Sum, $f_1 A(\vec{x}) + (1 - f_1) B(\vec{x})$.**
  - **Product, $A(\vec{x}) \times B(\vec{x})$.**
  - **Composition, $A(B(x))$ (only one dimension).**
  - **Convolution, $\int_{t_1}^{t_2} A(x - t) * B(t) \mathrm{d}t$.**
  - **Map,**

$$F(x) = \begin{cases} A(x) & \text{if } x \in [x_0, x_1) \\ B(x) & \text{if } x \in [x_1, x_2) \\ \cdots & \\ Z(x) & \text{if } x \in [x_{N-1}, x_N] \end{cases}$$

- **Specialised mixing PDFs: Coherent amplitude sum, incoherent sum, truth resolution, three-Gaussian resolution, Dalitz-plot region veto, threshold damping function.**

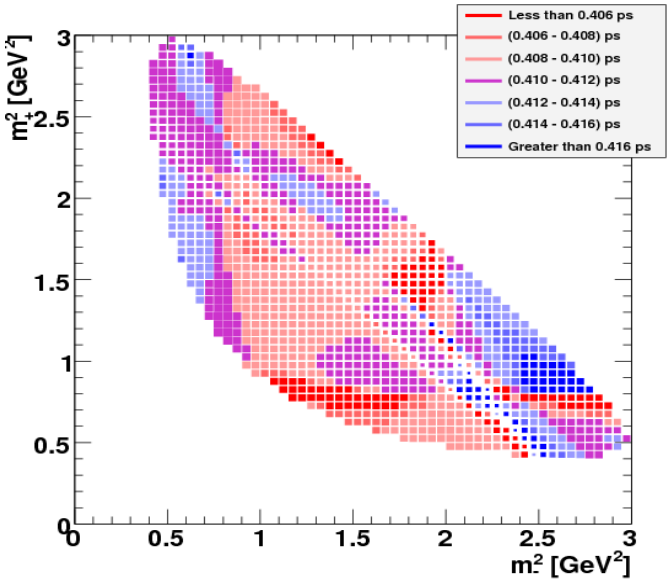
Advanced users: New PDFs

- **Backend code follows two patterns: Function signatures for Thrust, variable-lookup for arbitrary parameters.**
- **To write a new PDF, some CUDA code is necessary:**

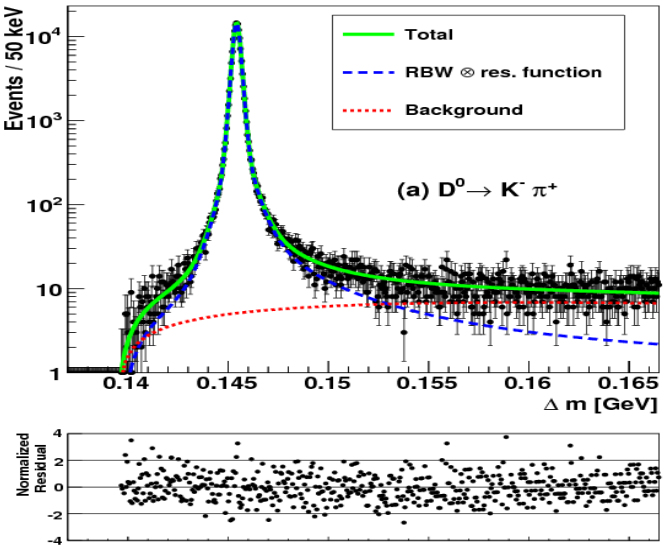
```
__device__ fptype dev_Gaussian (fptype* evt, fptype* p, unsigned int* indices)
  fptype x = evt[indices[2 + indices[0]]];
  fptype mean = p[indices[1]];
  fptype sigma = p[indices[2]];
  return EXP(-0.5*(x-mean)*(x-mean)/(sigma*sigma));
}

__device__ device_function_ptr ptr_to_Gaussian = dev_Gaussian;

__host__ GaussianPdf::GaussianPdf (std::string n,
                                   Variable* _x,
                                   Variable* mean,
                                   Variable* sigma)
  : GooPdf(_x, n)
{
  std::vector<unsigned int> pindices;
  pindices.push_back(registerParameter(mean));
  pindices.push_back(registerParameter(sigma));
  cudaMemcpyFromSymbol((void**) &host_fcn_ptr, ptr_to_Gaussian, sizeof(void*))
  initialise(pindices);
}
```

Realistic problems

- Time-dependent amplitude analysis of $D^0 \to \pi^+\pi^-\pi^0$ ("mixing fit").
  - Unbinned fit in four dimensions.
  - About 40 parameters for the signal.



- Measuring the natural line width of the D^{*+} ("Zach's analysis").
  - Binned fit involving a convolution of a sum of three Gaussians with the relativistic p-wave Breit-Wigner resonance lineshape.
- Both analyses originally written for CPU.



Results

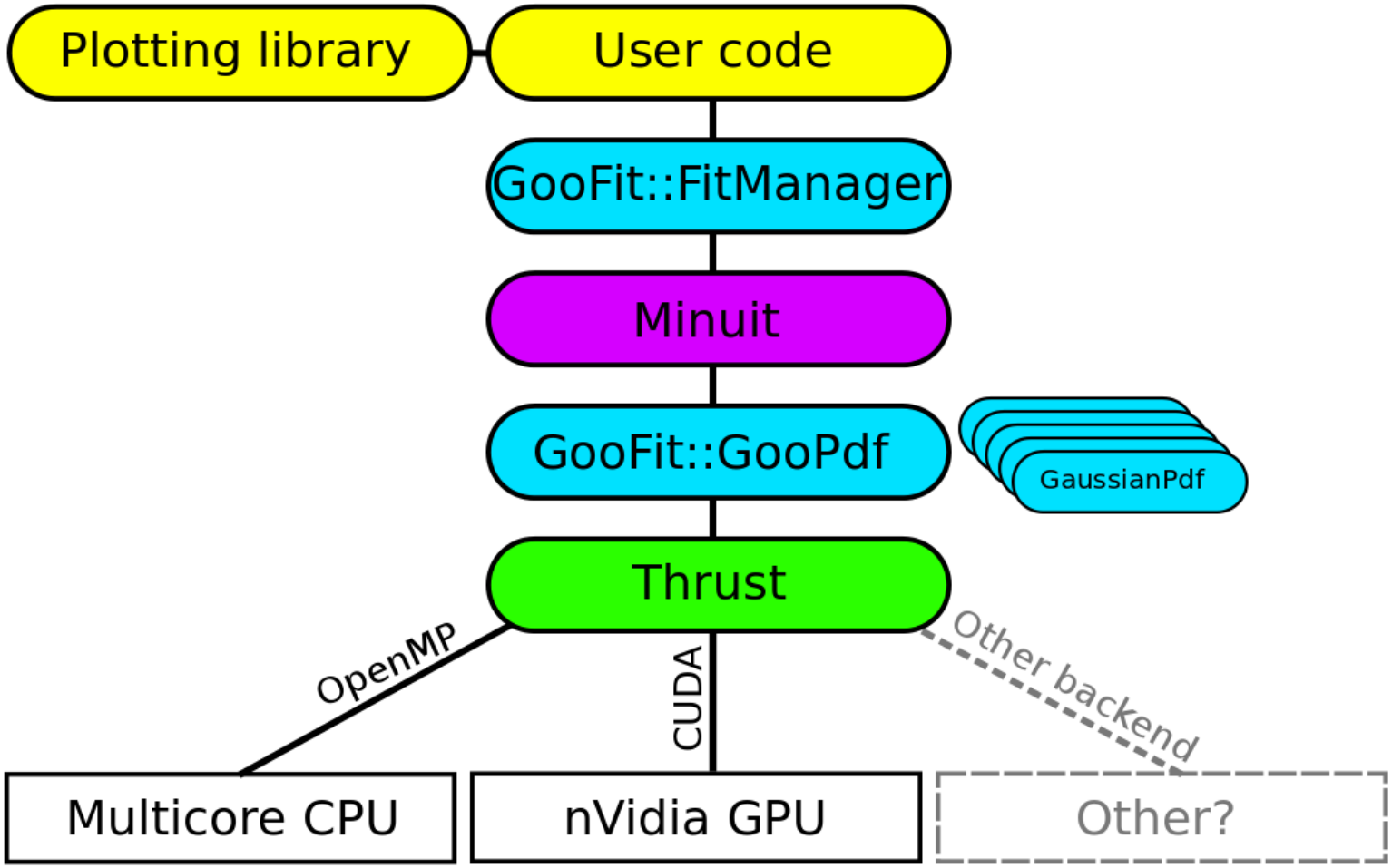
- Several platforms:

Name	Chip	Cores	Clock [GHz]	RAM [Gb]	OS
Cerberus (CPU)	Intel Xeon E5520	8*	2.27	24	Fedora 14
Cerberus (GPU)	nVidia C2050	448	1.15	3	Fedora 14
Starscream (CPU)	Intel i7-3610QM	4*	2.3	8	Ubuntu 12.04
Starscream (GPU)	nVidia 650M	384	0.9	1	Ubuntu 12.04
Oakley	nVidia C2070	448	1.15	6	RedHat 6.3

- Asterisk indicates hyperthreading - two virtual processors per physical core.

	Mixing fit		Zach's fit	
Platform	Time [s]	Speedup	Time [s]	Speedup
Cerberus, 1 core, original code	19489	1.0	438	1.0
Cerberus OMP (1)	3056	6.4	60.6	7.2
Cerberus OMP (2)	1563	12.5	31.0	14.1
Cerberus OMP (4)	809	24.1	18.2	24.1
Cerberus OMP (8)	432	45.1	9.2	47.6
Cerberus OMP (12)	534	36.5	12.2	35.9
Cerberus OMP (16)	326	59.8	6.9	63.5
Cerberus OMP (24)	432	45.1	9.5	46.1
Cerberus C2050	64	304.5	5.8	75.5
Starscream OMP (1)	2042	9.5	37.1	11.8
Starscream OMP (2)	1056	18.5	19.2	22.8
Starscream OMP (4)	562	34.6	10.8	40.6
Starscream OMP (8)	407	47.9	6.9	63.5
Starscream 650M	212	91.9	18.6	23.5
Oakley C2070	54	360.1	5.4	81.1

Architecture



Praise for Thrust

- **Thrust's abstraction of the kernel launch makes it easy to write code that works on multiple backends:**

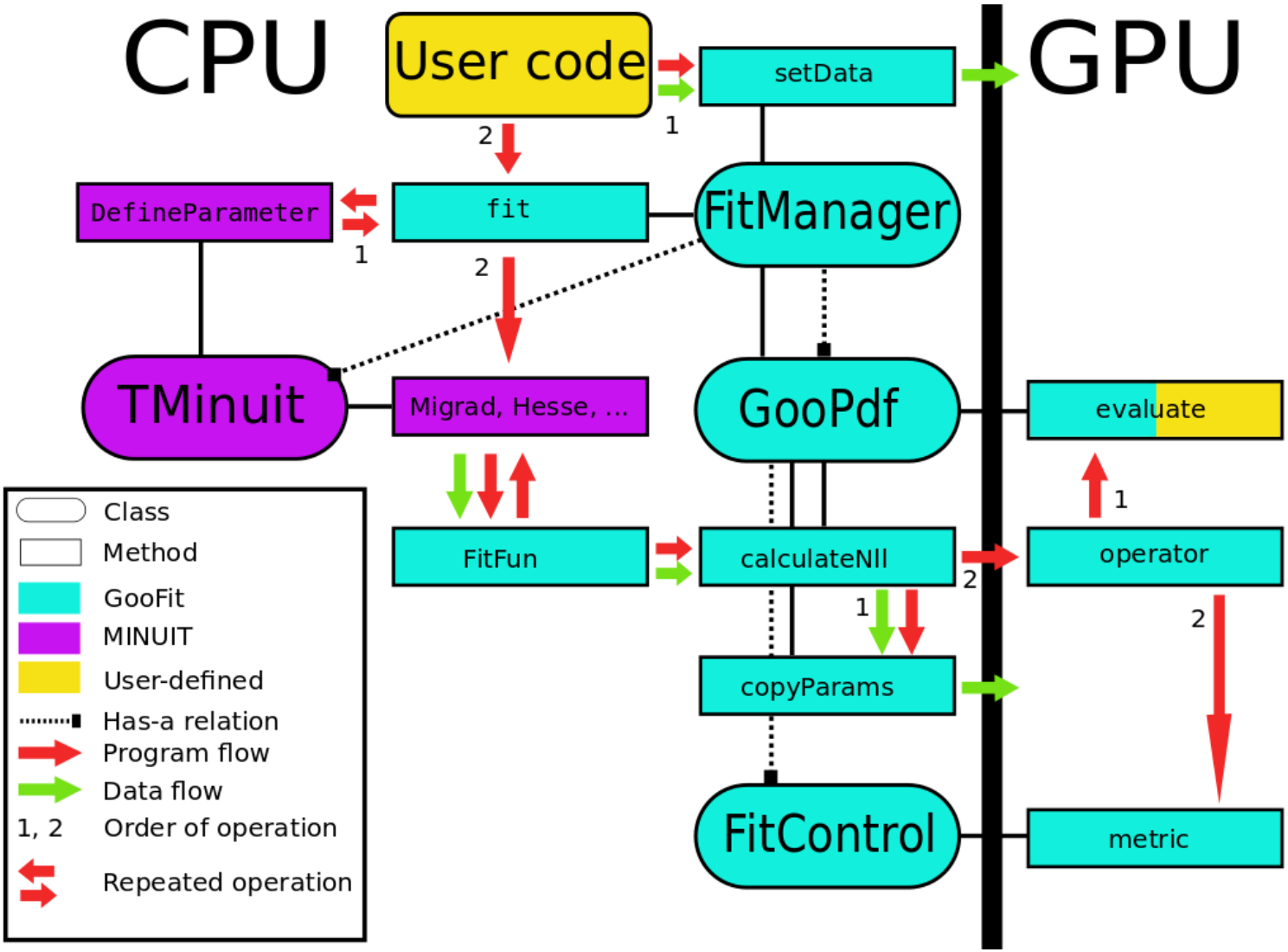
```
MetricTaker evalor(this, getMetricPointer("ptr_to_Prob"));
// transform call does all the launching - just need operator() method
transform(make_zip_iterator(make_tuple(eventIndex, arrayAddress, eventSize)),
          make_zip_iterator(make_tuple(eventIndex + numEntries, arrayAddress,
          results.begin(),
          evalor);


 // ...
EXEC_TARGET fptype
MetricTaker::operator () (tuple<int, fptype*, int> t) const {
```

- **Add a layer of abstraction:**

```
#define MEM_DEVICE __device__
#define MEM_SHARED __shared__
#define MEM_CONSTANT __constant__
#define EXEC_TARGET __device__
#define SYNCH cudaDeviceSynchronize
#define MEMCPY(target, source, count, direction) \
        cudaMemcpy(target, source, count, direction)
 // ...
```

in exchange for targeting multiple backends.

Program flow



Summary

- **GooFit is a working library ready for use in analysis.**
- **Thrust backend allows us to parallelise either for GPU or multicore CPU.**
  - **nVidia GPU supporting CUDA, with compute capability at least 2.0. Includes both heavy-duty workstation boards and onboard laptop chips.**
  - **CPU supporting OpenMP - most modern architectures.**
- **GitHub repository: https://github.com/GooFit.**
  - **Source code at https://github.com/GooFit/GooFit.**
  - **Also includes user manual.**
  - **Tutorials, examples, exercises at https://github.com/GooFit/GooTorial.**
  - **We support users actively! E-mail sokoloff@ucmail.uc.edu.**
  - **We can arrange GPU access for education and R&D.**
- **Further reading:**
  - **"GooFit: A library for massively parallelising maximum-likelihood fits", CHEP 2013 proceedings, http://arxiv.org/abs/1311.1753.**
  - **"Implementation of a Thread-Parallel, GPU-Friendly Function Evaluation Library", R. E. Andreassen *et al.*, *IEEE Access*, Vol. 2, 160-176 (2014) doi:10.1109/ACCESS.2014.2306895.**
  - **"Parallelizing the Standard Algorithms Library, N3408", a Post-C++11 Library Proposal, Jared Hoberock *et al.* (2012) http://cplusplus.github.io/LWG/lwg-proposal-status.html.**

Backup slides

Plotting with GooFit

- **GooFit is a fitting package: No graphics capability is built in.**
- **To make a plot, extract values of function (including component functions) at specified points, then use those values in your plotting library of choice:**

```
Variable* gmean = new Variable("gmean", 0, 1, -10, 10);
Variable* sigma = new Variable("sigma", 1, 0.5, 1.5);
Variable* alpha = new Variable("alpha", -2, -10, 0);
Variable* sFrac = new Variable("sFrac", 0.9, 0.5, 1.0);
GaussianPdf* signal = new GaussianPdf("signal", xvar, mean, sigma);
ExpPdf* backgr = new ExpPdf("backgr", xvar, alpha);
AddPdf* total = new AddPdf("total", sFrac, signal, backgr);

// [...]

UnbinnedDataSet grid(xvar); // To contain points to be plotted
double step = (xvar->upperlimit - xvar->lowerlimit)/xvar->numbins;
for (int i = 0; i < xvar->numbins; ++i) {
  xvar->value = xvar->lowerlimit + (i + 0.5) * step;
  grid.addEvent();
}

total.setData(&grid);
vector<vector<double> > pdfVals; // To store PDf values
total.getCompProbsAtDataPoints(pdfVals);
// pdfVals[0,1,2] now contains values of total, signal, backgr PDFs
```