Experiences on using GPU accelerators for data analysis in ROOT/RooFit

Sverre Jarp, Alfio Lazzaro, Julien Leduc,
Yngve Sneen Lindal, Andrzej Nowak
European Organization for Nuclear Research (CERN), Geneva, Switzerland

Workshop on Future Computing in Particle Physics, e-Science Institute,
Edinburgh (UK)
June 15th–17th, 2011

Tiny OpenCL intro

OpenCL device abstractions

- Different hardware/SDKs/drivers are represented by different «platform» objects
- A platform object can have a range of devices (of course, if you have them physically)

An example

```
cl_platform platform;
cl_device device;
cl_context context;
cl_command_queue queue;
cl_int status;

clGetPlatformIDs(1, &platform, NULL);
clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
context = clCreateContext(NULL, 1, &device, NULL, NULL, &status);
queue = clCreateCommandQueue(context, device, 0, &status);
```

Tiny OpenCL intro

❑ *Declaring* a computational kernel

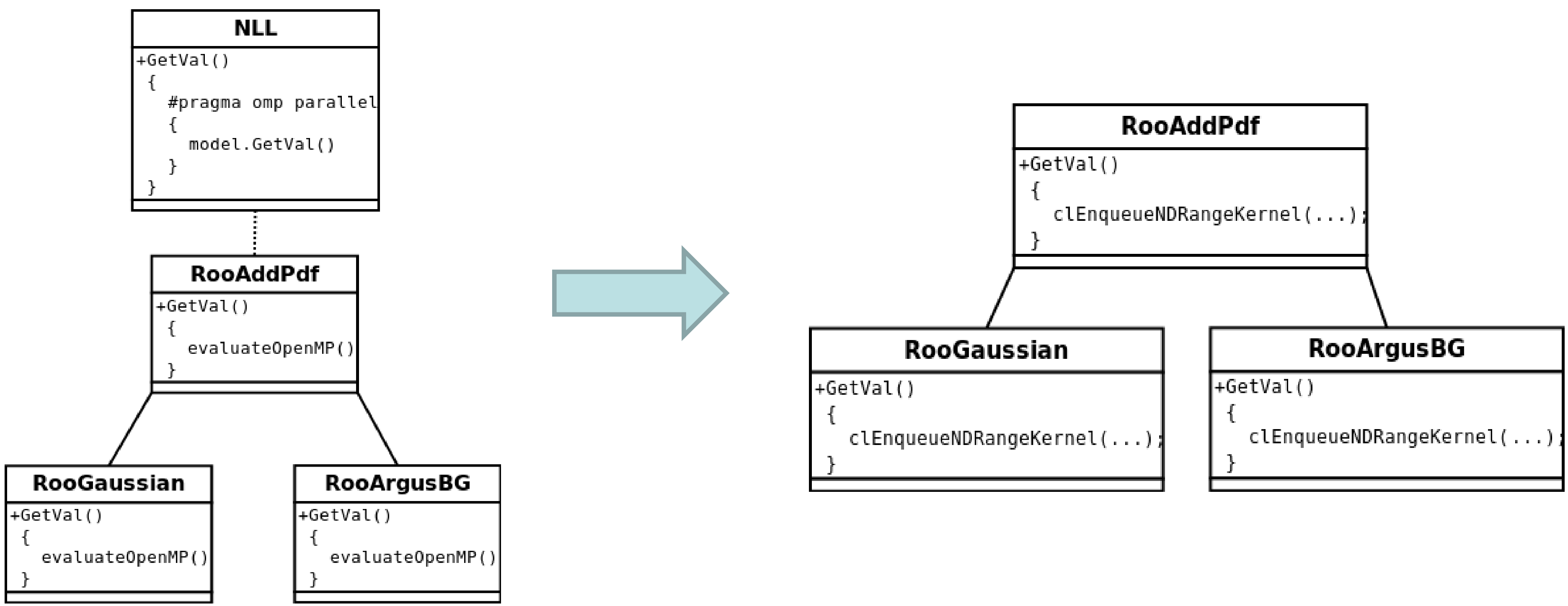
```
__kernel void evaluatePdfGaussian(__const double mu, __const double sigma, __global const double *data,
__global double *results, __const int N)
{
   int i = get_global_id(0);
   if (i >= N) return;
   double x = data[i];
   double temp = (x-mu)/sigma;
   temp *= temp;
   results[i] = exp(-0.5*temp);
}
```

❑ *Executing* a computational kernel

```
//Assume we have the required arguments and a kernel object for the Gaussian kernel above
clSetKernelArg(evaluatePdfGaussian, 0, sizeof(float), (void*)&mu);
clSetKernelArg(evaluatePdfGaussian, 1, sizeof(float), (void*)&sigma);
clSetKernelArg(evaluatePdfGaussian, 2, sizeof(cl_mem), (void*)&data);
clSetKernelArg(evaluatePdfGaussian, 3, sizeof(cl_mem), (void*)&results);
clSetKernelArg(evaluatePdfGaussian, 4, sizeof(int), (void*)&N);
size_t workGroupSize = 128; //e.g.
size_t numWorkGroups = N % workGroupSize == 0 ? N/workGroupSize : N/workGroupSize + 1;
size_t total = workGroupSize * numWorkGroups;
clEnqueueNDRangeKernel(queue, evaluatePdfGaussian, 1, NULL, &total, &workGroupSize, 0, NULL, NULL);
```

GPU Implementation (OpenCL)

- With OpenMP, each thread can evaluate the tree top-down directly in fully parallel. Using a GPU requires an explicit call to a kernel inside each PDF (see 2nd illustration), suggesting lower parallel efficiency.



- Leads to larger serial fraction, many kernel calls and in general, stalls
- Data is uploaded once, in the beginning of the run.

GPU Implementation (OpenCL)

- Parallel block-wise reduction is used. Improves the speedup significantly (uses GPU shared mem)
- Double precision and general accuracy requirements prevents using native transcendental units and also limits performance in general (GPUs are made for single-precision primarily)
- Not memory-bound (on an Nvidia GTX470, at least) since we're doing expensive computations, so texture cache has no effect
- Straight-forward implementation. No possibility to use e.g. shared memory (except for reduction). But this is also beneficial from a user perspective

Downsides

- **Introduces more expressive code when setting up environment and e.g. calling kernels.**
- **Duplication of code since we now use an OpenCL compiler in addition to the C/C++ compiler**
- **May be necessary to explicitly program with vector types to exploit performance on AMD cards (we have not tested this yet).**
- **We have also tried OpenCL for CPUs. Our experiences:**
  - Have to use vector types to achieve vectorization. But even then AMDs OpenCL compiler does not vectorize transcendentals for instance
  - To obtain performant code, it is necessary to do more work per OpenCL thread. Like doing work by hand instead of making a computer do it…
  - Talked to Intel OpenCL guru today, he says that this is *not* the case with Intels implementation
  - It would of course be nice to have one unified programming model for any device, but that seems like somewhat of a silver bullet so far…

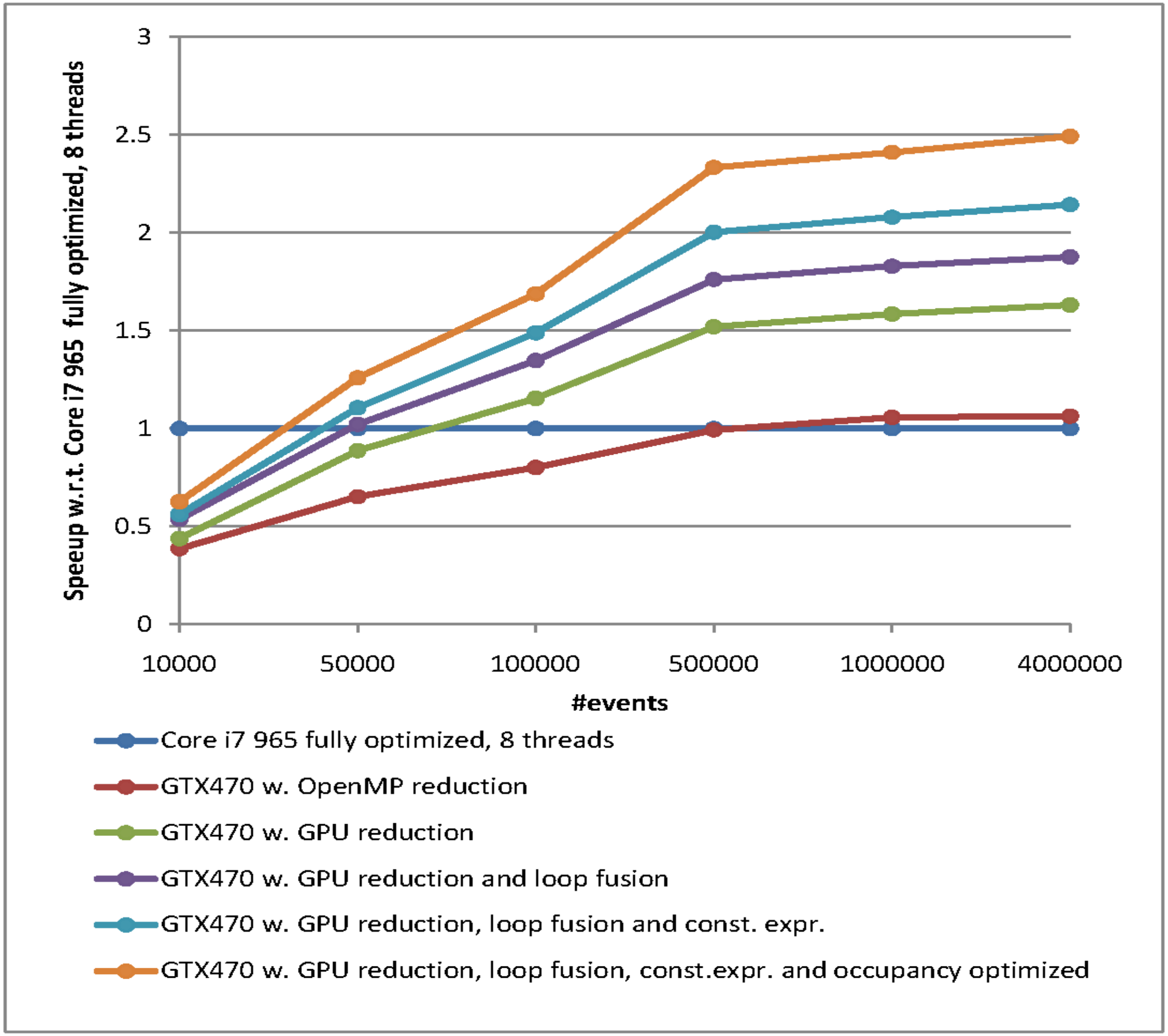
GPU Test environment

- **PC (host)**
  - Desktop system
  - 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
  - Power Consumption 200W
  - Price ~$300 (July 2010)

Performance

- This is not a fair "CPU vs GPU" comparison because of different algorithm



Conclusion

- The two algorithms (OpenMP and OpenCL) can coexist seamlessly in the application
- Up to a factor 2.5x (on our tests) with respect to OpenMP with 8 SMT threads (i7 965 and GTX470). The CPU scalability compared to one core is ~4.6x.
- GPUs behaves better with more events, as expected
- Seems ideal to load-balance, since equally priced products perform comparable
- It is clear that reduction must be done on the GPU to achieve high GPU performance. This reduction is deterministic, which can be a requirement from minimization algorithms
- We have measured the GPU idle percentage to be around 12% in ideal cases, which is not too bad, taking the algorithm into account

Conclusion

- Note that our target is running at the user-level on the GPU of small systems (laptops, desktops), i.e. with small number of CPU cores and commodity GPU cards
  - Comparisons with a GPU Tesla card is more appropriate with a CPU server system, which is not our goal
  - Main limitation is the algorithm and the double precision
  - Small limitation due to CPU⇔GPU communication
- Soon the code will be released in the standard RooFit (discussion with the authors of the package ongoing)

Current/future developments

- Try the code on LHC analyses
- Test vector types on AMD cards to see if they have any performance effect
- Concurrent execution on CPU with OpenMP and GPU with OpenCL