Empower. Partner. Lead.

Ohio Supercomputer Center

Multi-level Parallel Fit Algorithms Using MPI and CUDA

SuperB Software R&D Workshop, 8-12 March 2010, Ferrara, Italy

Karen Tomko[1], Adam Simpson[2], Rolf Andreassen[2], Mike Sokoloff[2], Brian Meadows[2]

[1]Ohio Supercomputer Center
Ohio State University
Columbus, Ohio, USA
ktomko@osc.edu

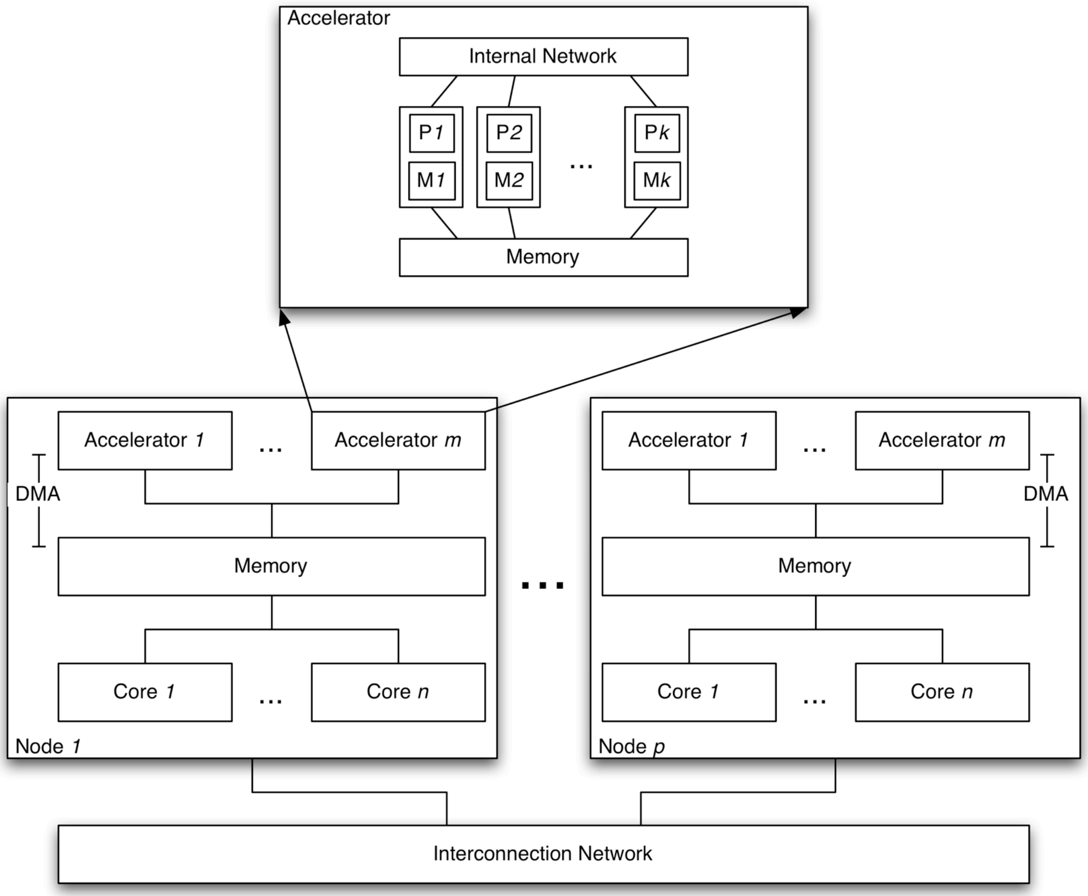
[2]Physics Department
University of Cincinnati
Cincinnati, Ohio, USA

Topics

- Modern heterogeneous computing systems
- Negative Log Likelihood (NLL) examples
  - Simple Gaussian
  - Breit-Wigner convoluted with a Gaussian
- Conclusions

Modern Clusters

Computing model



- OSC's Bale Visualization Cluster
  - 18 nodes populated with:
  - 2 AMD 2.6GHz Dual-Core Opteron CPU's
  - 8 GBs of RAM
  - 750 GB SATA hard disk
  - Infiniband Dual Port HCA card
  - 2 Tesla C1060 cards per node, on 8 of the nodes
- Tesla C1060
  - Global memory: 4GB
  - No. of Multiprocessors: 30
  - Number of cores: 240
  - Constant memory: 64KB
  - Shared memory: 16KB
  - Registers available per block: 16K
  - Clock rate: 1.30 GHz
- Cuda 2.3

Negative Log Likelihood (NLL) Example

TMinuit implementation of the MINUIT maximum likelihood fitting code found in the ROOT package from CERN

Three steps[1]:

1. calculating the negative log likelihood (NLL) for a set of fitting parameters -- a sum over all of the events (or elements in the measurements array)
2. calculating the normalization of the probability density function (PDF) -- a function only of the fitting parameters, whose calculation can be very slow if no analytic expression is available for the integral;
3. the minimization of NLL -- (implemented in MIGRAD) requires a gradient calculation, i.e., the calculation of derivatives with respect to the number of free parameters.

[1] from A. Lazzaro and L. Moneta, MINUIT Package Parallelization and applications using the RooFit Package, Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research (2008).

NLL Hybrid CUDA/MPI Implementation

- Cuda design (Adam Simpson)
  - Fundamental operation on each event

$$f(x, x_0, \sigma) = \log\left(Ae^{B(x-x_0)^2}\right), A = \frac{1}{\sqrt{2\pi}\sigma}, B = \frac{-1}{2\sigma^2}$$

  - Summation over all events based on CUDA reduction example
  - Tree-based reduction algorithm, Log_2 N steps
- The MPI/CUDA hybrid design
  - One GPU device per MPI process
  - Partition measurement array (events) evenly across MPI processes
  - Perform summation on GPU for events local to a process
  - Sum across MPI processes using MPI_AllReduce

Simple Gaussian CUDA/MPI results

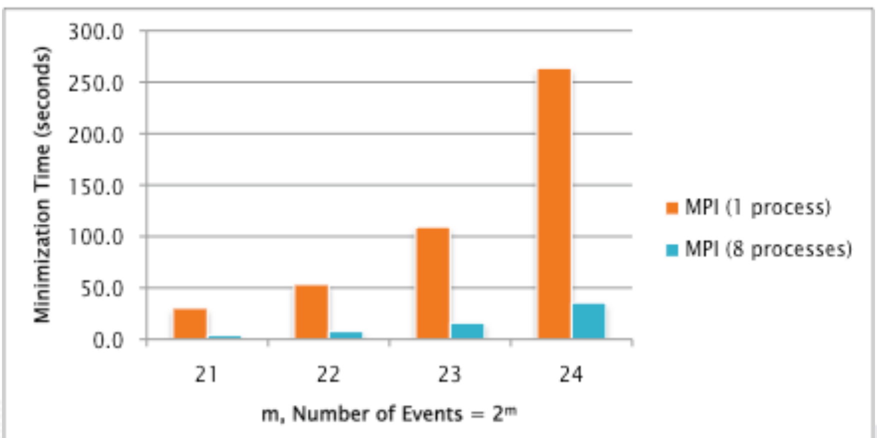
Number of Events	Average time of Minimization (seconds)			
	MPI 1 CPU core	MPI 8 CPU cores	CUDA 1 GPU	CUDA/MPI 8 CPU cores, 8 GPUS
2^{21}	30.5	4.0	0.4	0.1
2^{22}	53.6	7.9	0.8	0.2
2^{23}	109.4	16.1	1.5	0.4
2^{24}	263.8	35.7	3.0	0.7

More than 300x over 1 CPU core

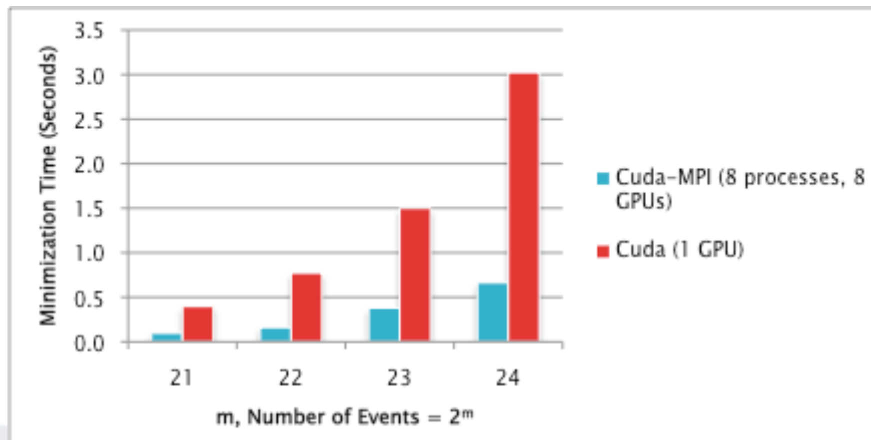
About 90x over 1 CPU core

Execution time comparisons

MPI: 1 vs. 8 CPUs



CUDA: 1 GPU vs. CUDA-MPI 8 GPUs



A more challenging case: Breit-Wigner convoluted with a Gausian (Voigtian)

- Implementation approach is the same
- Voigtian function replaces simple Gaussian
- Breit-Wigner equation: $f(x, x_0, g) = \frac{1}{(x - x_0)^2 + \frac{1}{4} g^2}$
- The convolution: $f(x, x_0, g, \sigma) = \frac{1}{(x - x_0)^2 + \frac{1}{4} g^2} \otimes A e^{B(x - x_0)^2}$, $A = \frac{1}{\sqrt{2\pi}\,\sigma}$, $B = \frac{-1}{2\sigma^2}$
- Free parameters: $x_0 = 2.0$, $g = 0.001$, $\sigma = 0.001$
- Complex error function, Faddeeva_2 from Matpac
- Why more challenging:
  - Code complexity is greater than simple Guassian
  - More computations and complex arithmetic
  - Local coefficient arrays
  - Data dependent branches

Breit-Wigner convoluted with a Gaussian CUDA/MPI results

- Still a work in Progress
  - CUDA and MPI versions running, hybrid CUDA/MPI not yet complete

Number of Events	Average time of Minimization (seconds)			
	MPI 1 CPU core	MPI 8 CPU cores	CUDA 1 GPU	CUDA/MPI 8 CPU cores, 8 GPUS
2^{21}	206.49	70.14	4.65	N/A

About 45x over 1 CPU core

- Problems:
  - High register requirements resulting in low utilization of GPU cores
  - Some divergent branching due to conditional statements in the complex error function

Possible Improvements:

- Look up table based implementation
  - e.g. FastAlgorithm in RooVoigtian
- Parameterize the coefficient arrays
- Investigate ptx code (assembly) to identify problems, try to fix at CUDA level
- Next generation NVIDIA Fermi and CUDA 3.0 and features will help
  - Register pressure relief (32K registers per SM)
  - Branch prediction and predicated instructions can reduce cost of divergent branches

Conclusions:

- Investigated GPU acceleration of HEP software
  - Two levels of parallelism, MPI and CUDA GPU
  - Fit algorithms: Simple Gaussian and Breit-Wigner convoluted with a Gaussian
  - Events processed and summed in parallel
- Results
  - 2 orders of magnitude reduction in runtime demonstrated for simple Gaussian
  - Significant reduction in runtime for Breit-Wigner convoluted with a Gaussian, but only ½ the improvement of the simpler case

Conclusions:

- Next Steps (with Adam Simpson, Rolf Andreassen)
  - Investigate alternatives for improving the Breit-Wigner convoluted with a Gaussian case
  - Try OpenCL for these fit algorithms
  - Develop a fitting package along the lines of RooFit to take advantage of these parallelization schemes
- Outlook with NVIDIA Fermi looks even brighter
  - Faster: 2X single precision, 8.5X double precision
  - ECC memory
  - Support for more of C++