Partial Wave Analysis using Graphics Cards

Niklaus Berger
IHEP Beijing

Hadron 2011, München
The (computational) problem with partial wave analysis

\[ \log \mathcal{L} \sum \sum \log(\sum \mathbf{V} \mathbf{V}^* \mathbf{A}_a(\Omega)\mathbf{A}_a^*(\Omega)) - \sum \log(\mathbf{V} \mathbf{V}^* (\frac{1}{N_{\text{MC}}^\text{rec}} \sum \mathbf{A}_a(\Omega)\mathbf{A}_a^*(\Omega))) \]

A complex calculation (repeated many times over)

lots of statistics at Babar, Belle, BES III, Compass, GlueX, Panda etc.

something potentially very slow
Four years ago...

- I moved to IHEP Beijing
- All I remembered about partial waves was an unpleasant theory exam
- People at IHEP were worried about a $\times 100$ increase in statistics
- I did not know about partial waves, but new how to do things fast
- I happened to have just read a magazine article about computing on graphics processors
Partial Wave Analysis as a Computational Problem

Splits into subtasks:
- Building a model
- Determining model parameters through a fit to the data
- Judge fit results

Iterate until satisfied

Tightly coupled with the physicist:
look at plots, adjust model and input parameters
From Model to Likelihood

Intensity (number of events) at a phase-space point $\Omega$

$$I(\Omega) = \left| \sum_\alpha V_\alpha A_\alpha (\Omega) \right|^2$$

Sum over partial waves

Decay amplitudes: Resonance and angular structure

Production amplitudes: Complex fit parameters
From Model to Likelihood

Intensity (number of events) at a phase-space point $\Omega$

$$I(\Omega) = \left| \sum_{\alpha} V_{\alpha} A_{\alpha}(\Omega) \right|^2$$

Sum over partial waves

Decay amplitudes: Resonance and angular structure

Production amplitudes: Complex fit parameters

Likelihood, given $n$ data points at $\Omega_i$

$$L \propto \prod_{i=1}^{n} \frac{I(\Omega_i)}{\eta(\Omega) I(\Omega) d\Omega}$$

Product over data events

Normalisation integral over phase space

Detection efficiency
From Model to Likelihood

Likelihood, given \( n \) data points at \( \Omega_i \)

Product over data events

Normalisation integral over phase space

Log likelihood

\[
\log \mathcal{L} \propto \sum_{i=1}^{n} \log \left( \sum_{\alpha, \alpha'} V_\alpha V^{*}_{\alpha'} A_\alpha (\Omega_i) A^{*}_{\alpha'} (\Omega_i) \right) - \sum_{\alpha, \alpha'} \left( V_\alpha V^{*}_{\alpha'} \left( \frac{1}{N_{\text{MC}}^{\text{gen}}} \sum_{i=1}^{N_{\text{MC}}^{\text{rec}}} A_\alpha (\Omega_i) A^{*}_{\alpha'} (\Omega_i) \right) \right)
\]

Sum over data events

Sum over partial waves
From Model to Likelihood: Fixed Amplitudes

\[ L \propto \prod_{i=1}^{n} \int \eta(\Omega) I(\Omega) d\Omega \]

Normalisation integral over phase space

Product over data events

Detection efficiency

Log likelihood

\[ \log L \propto \sum_{i=1}^{n} \log \left( \sum_{\alpha, \alpha'} V_\alpha V^{*}_{\alpha'} A(\Omega_i) A^{*}_{\alpha'}(\Omega_i) \right) - \sum_{\alpha, \alpha'} \left( V_\alpha V^{*}_{\alpha'} \left( \frac{1}{N^{\text{gen}}_{\text{MC}}} \sum_{i=1}^{N^{\text{rec}}_{\text{MC}}} A(\Omega_i) A^{*}_{\alpha'}(\Omega_i) \right) \right) \]

Sum over data events

Sum over partial waves

Independent of fit parameters: precalculate; memory \( O(N_{\text{event}} \times N_{\text{wave}}^2) \)

Independent of fit parameters: precalculate

Normalisation integral as a sum over MC events

Summing only reconstructed events takes into account detection efficiency

Computationally intensive: \( O(N_{\text{iteration}} \times N_{\text{event}} \times N_{\text{wave}}^2) \)

Likelihood, given \( n \) data points at \( \Omega_i \)
Going parallel!

- Almost all our hardware is now parallel
- Almost all our software is not
- Almost all our problems are trivially parallel (events!)
- The solution to speed problems is obvious...
How to do parallel?

Grid
- Almost infinite power
- Very limited inter-process communication
- Very long latency

Farm/Cluster
- Lots of power
- Some inter-process communication
- Long latency (Network & Scheduling)

Multi-core CPU
- Finite power
- Very fast inter-process communication
- Almost no latency

Graphics Processor
- Almost infinite floating-point power
- Fast communication with CPU
- Short latency
PWA is embarrassingly parallel:

- Exactly the same (relatively simple) calculation for each event
- Every event has its own data, only fit parameters are shared
- Use parallel hardware and make use of Single Instruction - Multiple Data (SIMD) capabilities
- Very strong here: Graphics processors (GPUs): Cheap and powerful hardware
Programming for the GPU is less straightforward than for the CPU

- Early days: Use graphics interface (OpenGL) - translate problem to drawing a picture
- Vendor low-level frameworks: Nvidida CUDA and ATI CAL
- Vendor higher level framework: Brook+
- Independent commercial software: RapidMind
- Emerging standard: OpenCL
ATI Brook+

We started with using ATI Brook+

• Was the first to provide **double precision**
• Hardware with best **performance/price**
• Very **clean programming model, narrow interface**

Had all of the early adopter problems

• Lots of bugs and limitations
• Small user base
• Mediocre support
• Uncertain future

Now discontinued by AMD/ATI, we switched to OpenCL
OpenCL is a vendor- and hardware independent standard for parallel computing (in principle...)

- Gives you lots of detailed control and optimization options...
- ... at the cost of a very low level, hardware driver like interface
- No type safety, optimization depends on machine type
- For embarrassingly parallel tasks: use some higher level abstraction
GPUPWA at BES III

GPUPWA is our running framework
- Just done transition to OpenCL
- GPU based tensor manipulation
- Management of partial waves
- GPU based normalisation integrals
- GPU based likelihoods
- GPU based analytic gradients
- Interface to ROOT::Minuit2 fitters
- Projections and plots using ROOT

See: http://gpupwa.sourceforge.net
Performance (Brook+)

We use a toy model $J/\psi \rightarrow \gamma K^+K^-$ analysis for all performance studies.

Using an Intel Core 2 Quad 2.4 GHz workstation with 2 GB of RAM and an ATI Radeon 4870 GPU with 512 MB of RAM for measurements.

![Graph showing performance comparison between FORTRAN, GPUPWA on CPU, and GPUPWA on GPU. The graph illustrates a ×150 speedup for GPU computations.]
Performance (OpenCL)
Performance (CPU/GPU)

![Graph showing performance comparison between Fortran, OpenCL, Brook+, and OpenCL CPU for different numbers of events. The graph plots time per iteration in seconds against the number of events.]
Calculation on GPUs using Nvidias CUDA (also on a cluster)

- Need more than hundred-fold parallel tasks: amplitude calculation at event level
- Some cost for copying data to and from GPU
- Small fraction of code (large, expensive loops) ported to GPU
- Coding/debugging somewhat challenging

Using a cluster with message passing interface (MPI)

- High-level inter-process communication; “easy” to code and debug
- Perform likelihood calculation in parallel; each node with a subset of data and MC
- Use Open MPI implementation of MPI2 (www.open-mpi.org)
- Scales well over multiple cores, with fast network also over small cluster

Indiana framework (Cleo-c, BES III and GlueX)

Following a presentation by M. Shepherd; work done by M. Shepherd, R. Mitchell and H. Matevosyan, Indiana University
Speed benchmarks

- Tested with a $\gamma p \rightarrow \pi^+\pi^+\pi^-\pi^-$ analysis with 5 $\pi^+\pi^+\pi^-$ resonances and one floating Breit-Wigner mass
- Amplitudes and log likelihoods are done on the GPU(s), the rest on the CPU(s)
- CPU parallelization handled by MPI

Preliminary conclusions:
- MPI parallelization is efficient
- It is difficult to use the full power of GPUs

<table>
<thead>
<tr>
<th>Fit Configuration</th>
<th>Time per fit iteration (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single CPU</td>
<td>268</td>
</tr>
<tr>
<td>Single CPU + 1 GPU</td>
<td>47</td>
</tr>
<tr>
<td>CPU Master + 4 (CPU + GPU)</td>
<td>14</td>
</tr>
<tr>
<td>CPU Master + 11 CPU Workers</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Amplitude</th>
<th>CPU</th>
<th>GPU*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breit-Wigner</td>
<td>800</td>
<td>8</td>
</tr>
<tr>
<td>Ang. Dist. (D-functions)</td>
<td>15,000</td>
<td>87</td>
</tr>
</tbody>
</table>

* includes time to copy result from GPU memory
Multi-CPU scaling

- MPI allows very efficient parallelization of likelihood computation
- Only parameters and partial sums need to be exchanged between nodes
- User never needs to write MPI calls - all taken care of behind the scenes
- Fast and easy solution for multi-core systems

\[\text{Time for Fit Convergence [seconds]}\]

\[\text{Number of Worker Nodes in Fit}\]

(MPI-Enabled Fit)

\(1/N\) Scaling from 1 CPU Fit

(*Tested on a 12-core machine*)
• Same fit with one change: Compute $\pi$ in the Breit-Wigner using the first $n$ terms of the arctan Taylor-expansion
• Now the fit time is dominated by the computational complexity of the amplitude
• More compute intensive amplitudes, i.e. more sophisticated models, are an excellent match for GPU accelerated fitting
AmpTools

- Independent of the experiment and the particular physics process the amplitude analysis fit (i.e. construction of the likelihood) is pretty much the same
- This suggests it is possible to write a general software package that does all the “heavy lifting” — especially regarding parallel computing
- The user provides code for two types of C++ objects:
  - A recipe for calculating amplitudes, e.g., Breit-Wigner function -- no built-in physics!
  - A mechanism to read data into the framework
- The user specifies how many amplitudes, what types, arguments, free parameters, etc., via a configuration file (limits recompiling between fits)
- Library has been used/developed at Indiana U. over the past several years -- has provided a unified approach for several analyses the group is working on
- They are now trying to make available for general use: amptools.sourceforge.net (although, at this stage, documentation/examples are under development)
Speed is not the problem...

• We are fast enough, if we actually use our hardware

• This requires some work (which is however well invested...)

• This requires moving beyond FORTRAN (to some sort of C...)

• This will allow us to focus on the real problems...
Fitting in the dark...

In partial wave analysis, we perform fits with 20 (40, 60, more...) free parameters:

- We will never know, whether we found the global minimum.
- We can tell if a wave-set is “sufficient”, but can we know it is “right”?
- Can we even judge the goodness of fit? (“Badness” is easy...)
- We know that there must be multiple solutions...
- There is detector resolution

On the technical side:

- Could we get minimisers working with complex numbers?
- Could we get more control over the minimizers?
- Could we get a high level language building on OpenCL?
Which results will be believed?

• However “wrong” the analysis, people will usually believe quantum numbers if there is a bump in the mass spectrum

• However “right” the analysis, people will usually not believe in a new resonance if there is no bump, especially if it is exotic
Summary

• PWA profits from massively parallel computing on GPUs
• We have created a software framework to harness this power - speedups of two orders of magnitude
• User base at BES is growing, development continues
• OpenCL (and beyond) is the way to go
• Interesting work also ongoing at Indiana University - including multiple nodes via MPI and here in Munich

• PWA has fundamental problems because of fits with too(?) many free parameters
• With GlueX (JLAB) and PANDA (FAIR), big new PWA facilities are on the horizon — what can we do?