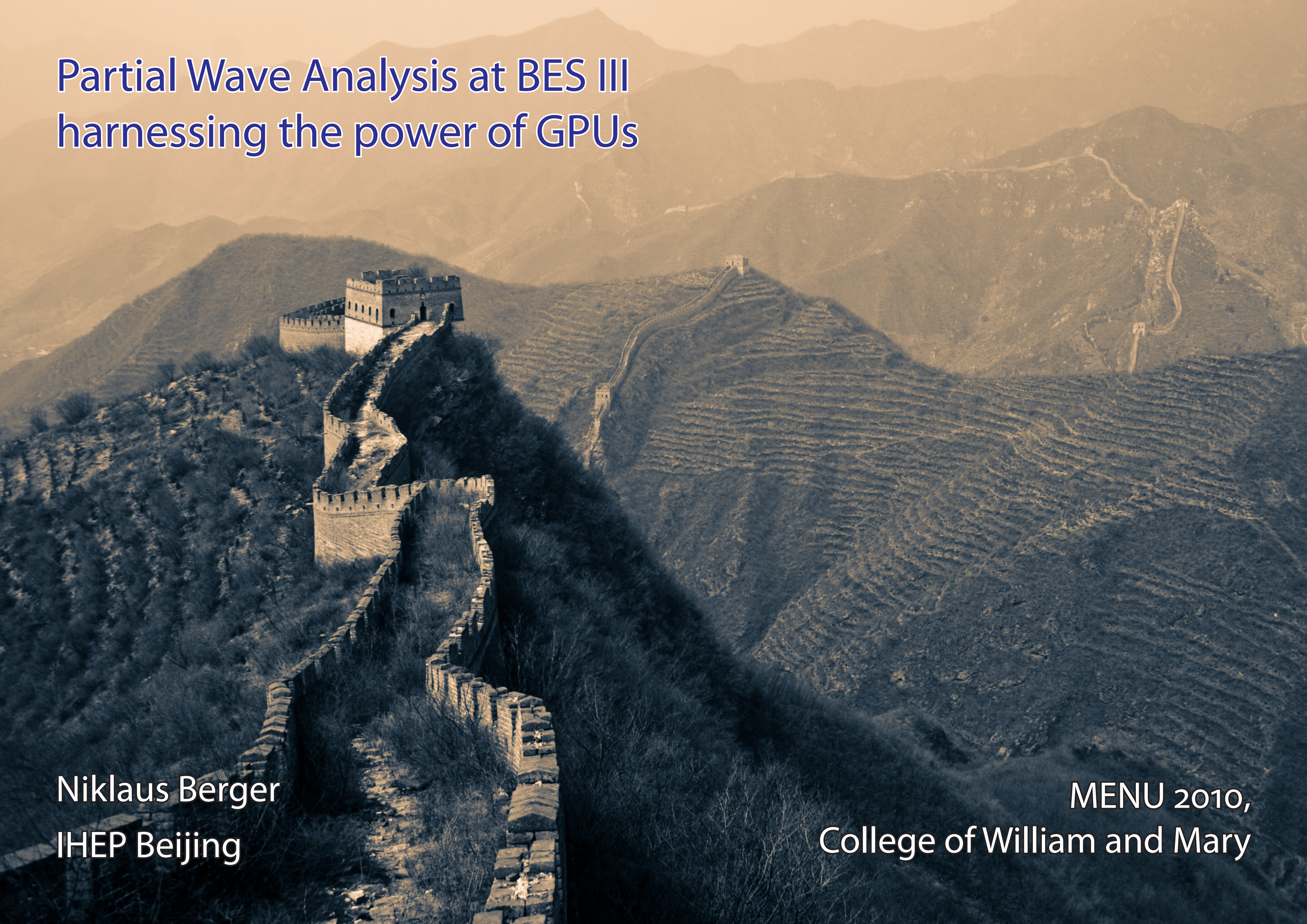


Partial Wave Analysis at BES III harnessing the power of GPUs

Niklaus Berger
IHEP Beijing

MENU 2010,
College of William and Mary

Overview

- Partial Wave Analysis at BES III
- PWA as a computational problem
- Graphics Processing Units (GPUs)
- PWA on GPUs
- Some performance numbers
- Some open questions

Partial Wave Analysis

- Light hadron spectroscopy is messy, even (or especially) with large statistics
- Extracting resonance parameters mostly requires partial wave analysis
- People will not always agree on what good PWA is
- Tensions between the desirable (theorists) and the computationally feasible (experimentalists)
- PWA as a computational problem

Partial Wave Analysis as a Computational Problem

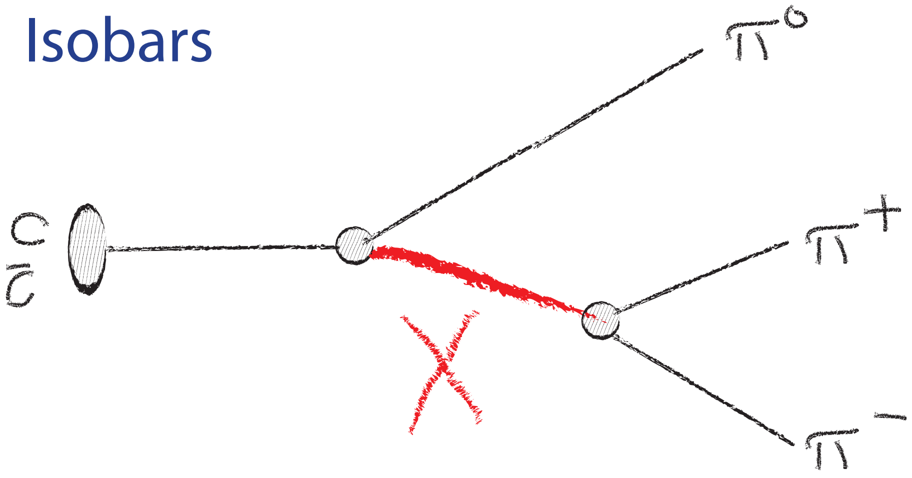
Splits into subtasks:

- Building a model
- Determining model parameters through a fit to the data
- Judge fit results

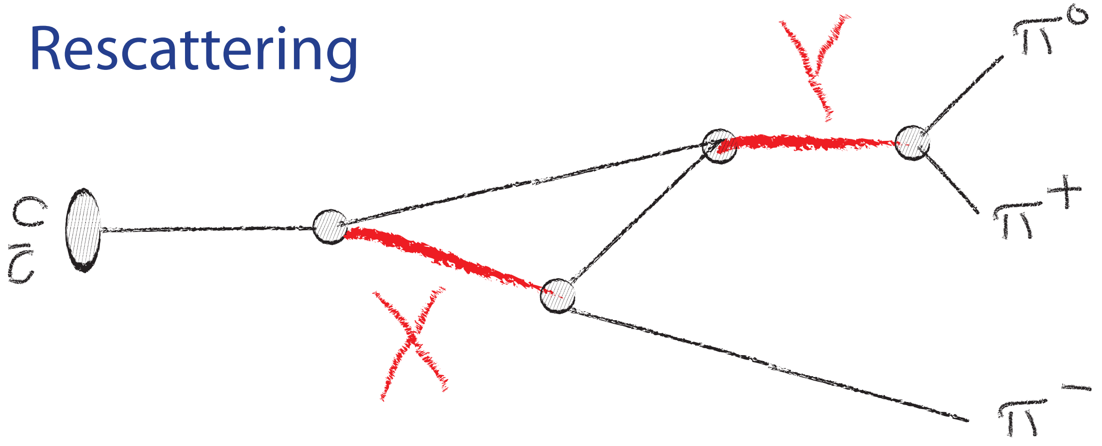
Iterate until satisfied

Tightly coupled with the physicist:
look at plots, adjust model and input parameters

Isobars



Rescattering



From Model to Likelihood

Decay amplitudes:
Resonance and angular structure

Intensity (number of events) at a phase-space point Ω

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

Sum over partial waves

Production amplitudes:
Complex fit parameters

From Model to Likelihood

Decay amplitudes:
Resonance and angular structure

Intensity (number of events) at a phase-space point Ω

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

Sum over partial waves

Production amplitudes:
Complex fit parameters

Likelihood,
given n data points at Ω_i

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

Normalisation integral over phase space

Product over data events

Detection efficiency

From Model to Likelihood

Likelihood,
given n data points at Ω_i

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

Normalisation integral over phase space

Product over data events

Detection efficiency

Log likelihood

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

Sum over data events

Sum over partial waves

From Model to Likelihood: Fixed Amplitudes

Likelihood, given n data points at Ω_i

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

Normalisation integral over phase space

Product over data events

Detection efficiency

Log likelihood

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

Independent of fit parameters: precalculate

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

Sum over data events

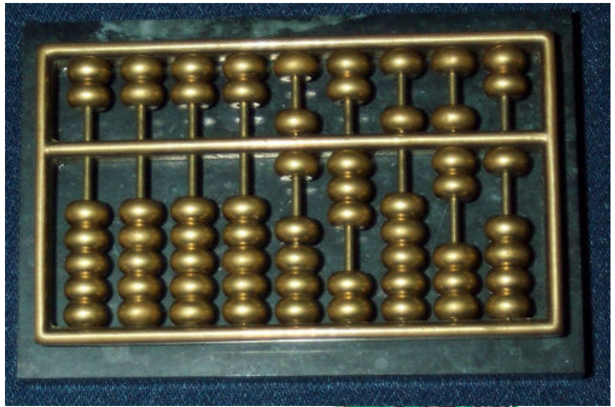
Sum over partial waves

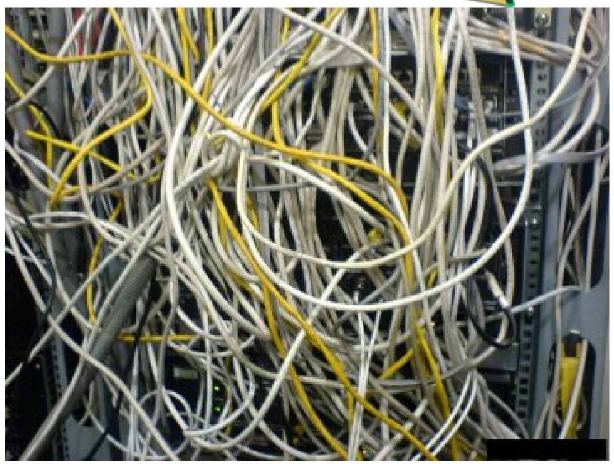
Computationally intensive: $\mathcal{O}(N_{iteration} \times N_{event} \times N_{wave}^2)$

Normalisation integral as a sum over MC events
Summing only reconstructed events takes into account detection efficiency

Speed Limits

- As long as the lookup tables fit into memory, calculation speed in the sum is limiting
- For large data sets, memory will be the limiting factor, if you are not very clever about caching
- When doing parallel computing (cluster or graphics card) at some point transfers become limiting

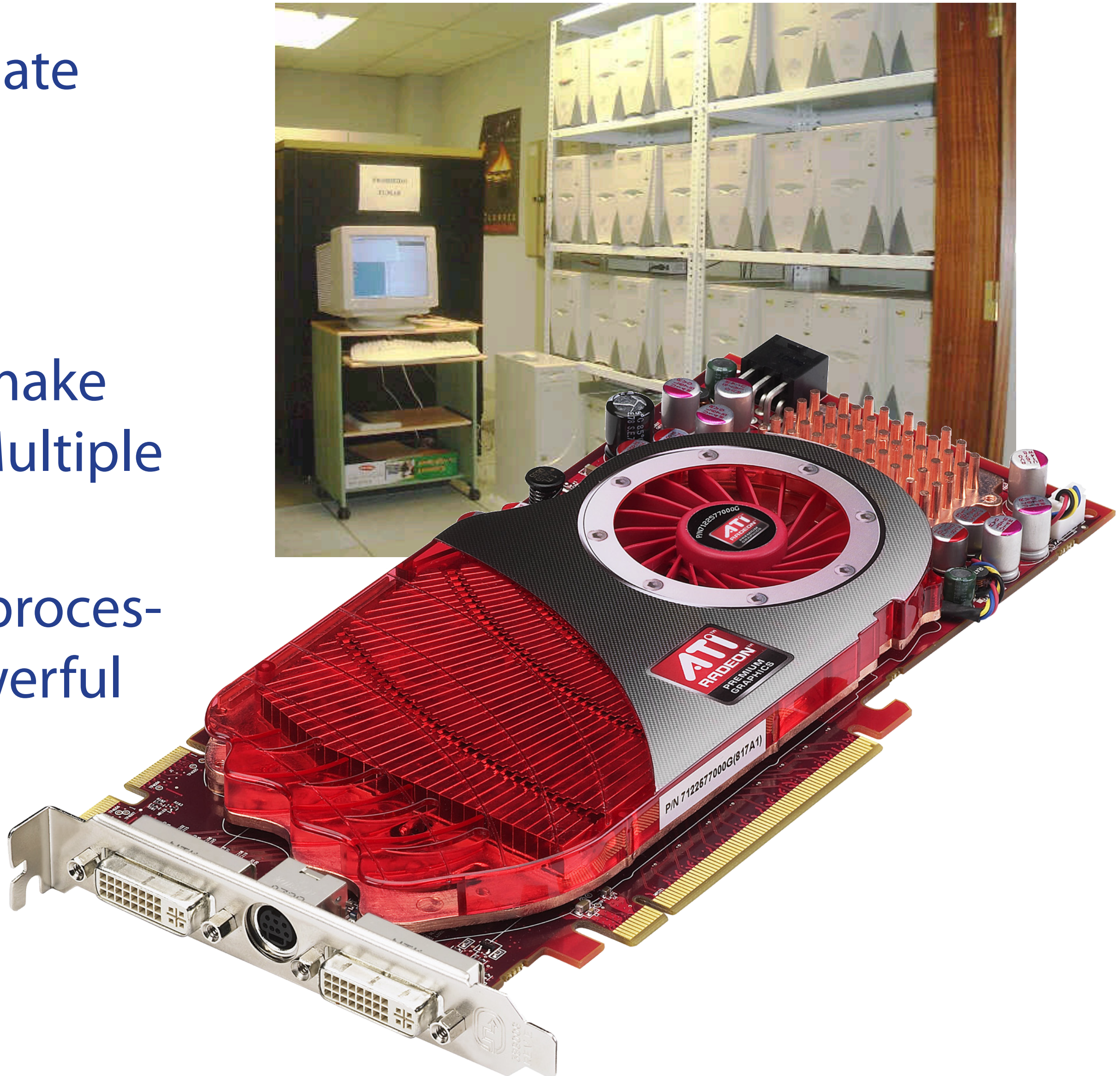




Parallel Computing

Events are independent - calculate terms in the sum in parallel

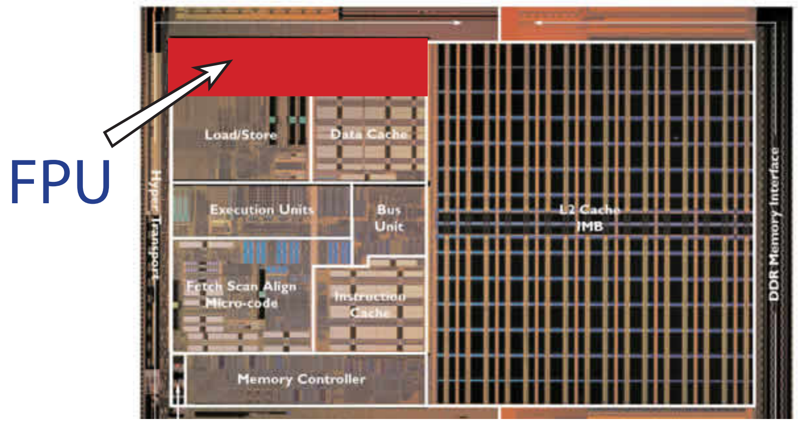
- Use a cluster/farm of PCs - not discussed here
- Use parallel hardware and make use of Single Instruction - Multiple Data (SIMD) capabilities
- Very strong here: Graphics processors (GPUs): Cheap and powerful hardware



CPUs and GPUs

Modern CPUs serve many purposes; important features:

- Quick reaction to (impredictable) user input
- Task switching (multitask operation systems)
- Some integer arithmetic, some floating point arithmetic in various precisions



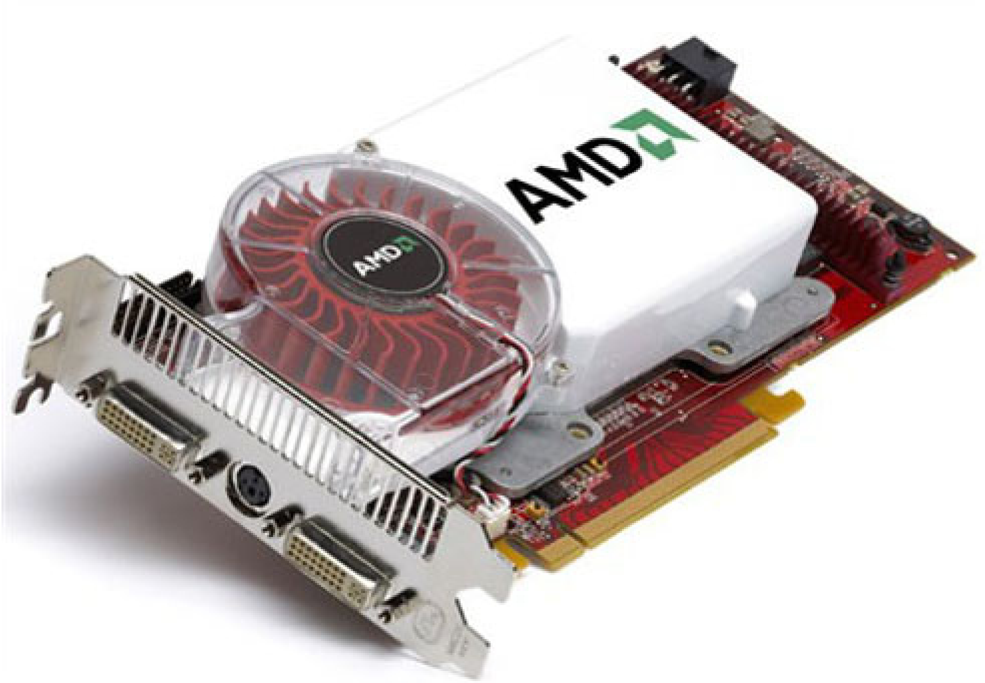
Modern graphics processors are built for one thing: 3D computer games

- Calculate projection of monsters, walls etc. to screen
- Colour each pixel, taking into account light and shades, texture of monster etc.
- Do all this in parallel

Architecture is optimised for operating on many (pixels) 4-tuples (3 colours + transparency) of floating point values - and lookups into large tables - just what we need

The Power of GPUs

- The market for game-PCs is huge and drives hardware
- We use a card with 800 parallel floating point units and > 1 TFlop/s
- This extremely powerful hardware available for < 300 $
- Machine can stand under your desktop (as opposed to a farm)
- In general: processors do not become much faster anymore, they become more parallel - we will have to learn to use this



Now: 1600 parallel FPUs
150 GB/s memory bandwidth
< 400$

GPUPWA

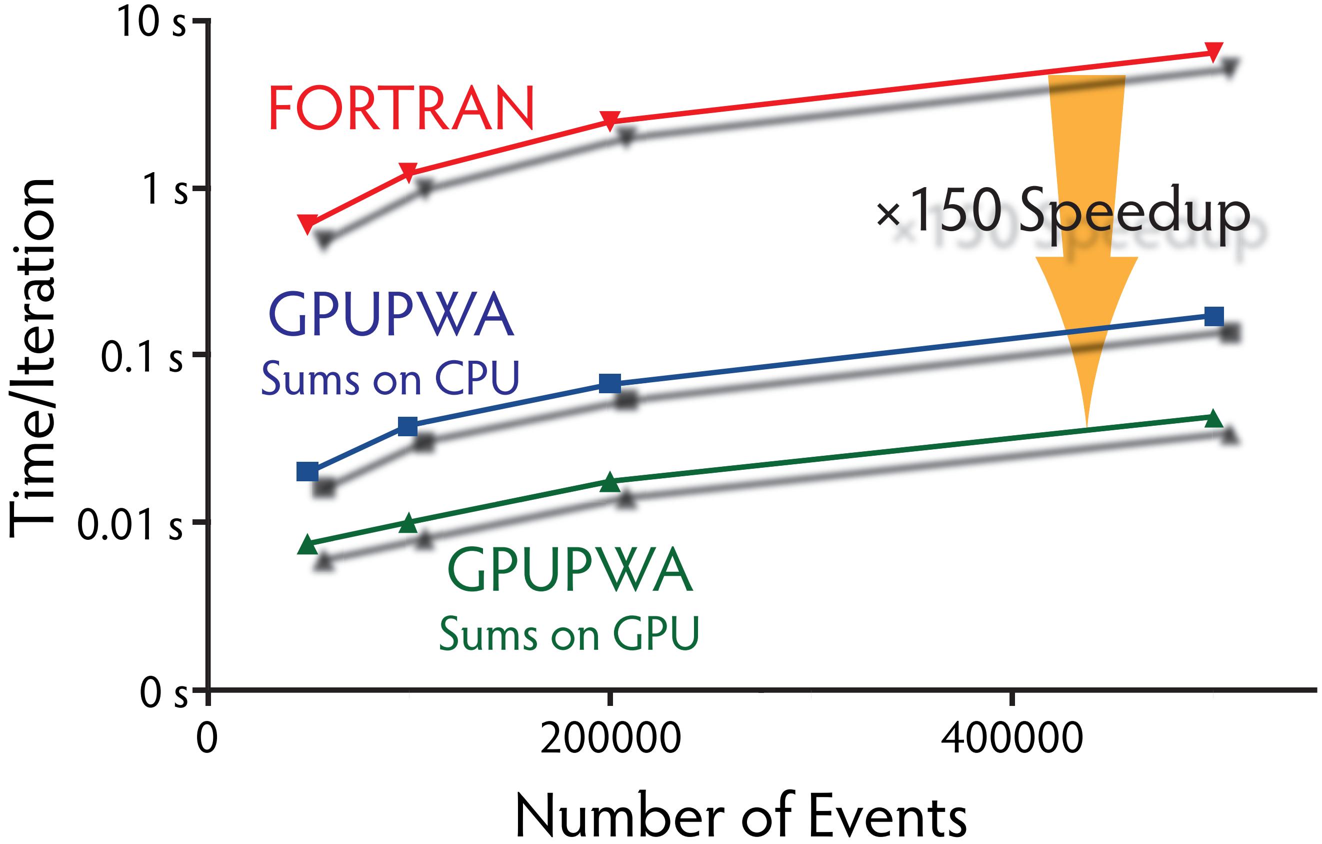
GPUPWA is our running framework

- GPU based tensor manipulation
- Management of partial waves
- GPU based normalistaion integrals
- GPU based likelihoods
- GPU based gradients
- Interface to ROOT::Minuit2 fitters
- Projections and plots using ROOT

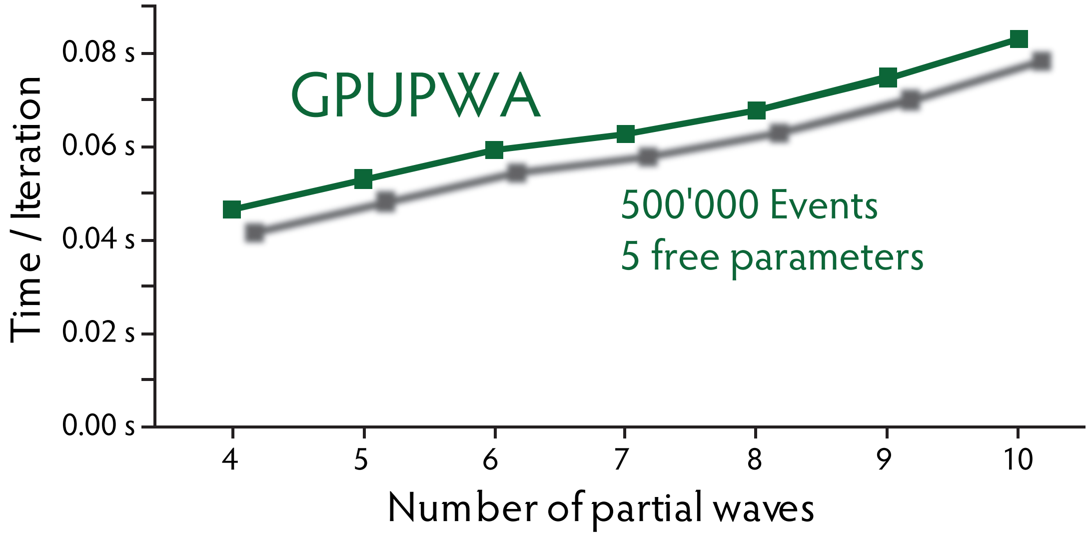
Performance

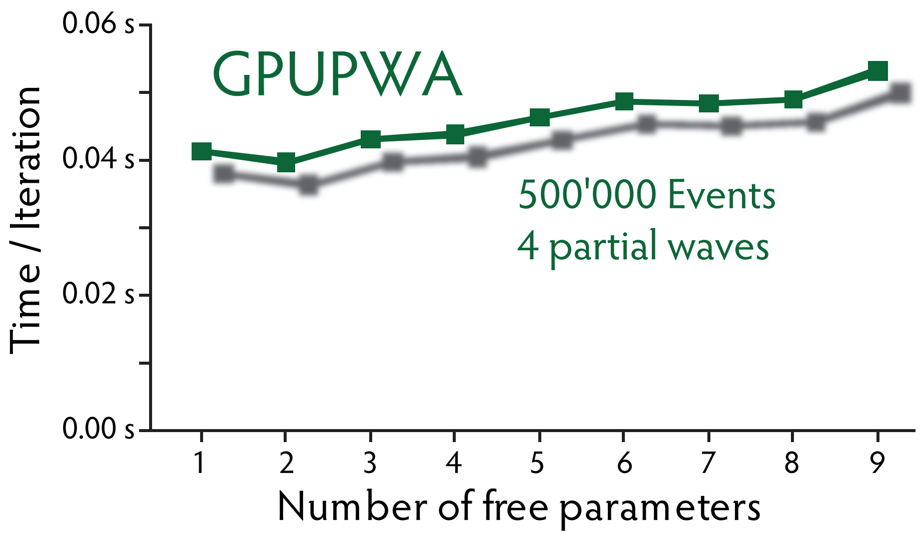
We use a toy model $J/\psi \to \gamma K^+K^-$ analysis for all performance studies

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



Performance





Fitting

Interface to ROOT::Minuit2 for fitting

MINUIT:

- Standard fitter, many tests performed, many iterations
- Allows for free resonance parameters
- Errors can be refined using MINOS

MINUIT with analytical gradients:

- No obvious performance or precision benefit
- Good for debugging

FUMILI:

- Extremely fast fitter, using stripped down version from BES II
- Needs very few iterations
- Finds same minimum as MINUIT
- Will never converge for ill defined problems
- Errors are not usable

Fitting - some questions

How should we parametrize a complex number?

- Cartesian:
  No trouble with bounds,
  possibly large correlations,
  painful for phase constraints

- Polar:
  Correlations ok,
  more „physical“,
  periodicity problem,
  undetermined parameter if
  amplitude zero

A Minuit extension for periodic parameters would be a blessing...

Fitting - some questions

> With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.
>
> ATTRIBUTED TO JOHN VON NEUMANN BY ENRICO FERMI

How many free parameters are sensible?

We can easily build models with >20 partial waves - 80+ free parameters

Does this make sense?

How do the fitters handle this? (Minuit manual: „...up to around 20...“)

What could constrain the models?

How do we choose which waves to include?

What PWA results do we believe?

- Determining quantum numbers for „bumps“ visible in mass spectra is well accepted
- Claims for „resonances“ not seen in mass spectra tend to be contested

- Leakage
- Background treatment
- Choice of waveset
- Parametrisation of resonances

Conclusions (Part I)

- PWA profits from massively parallel computing
- GPUs are currently the cheapest and most parallel HW available for the task
- We have created a software framework to harness this power - speedups of two orders of magnitude
- The software is in use at BES III
- User base is growing, development continues

Conclusions (Part II)

- Computing power is not the limiting factor in PWA
- If there are calculable models, we have the power to go beyond the isobar model
- Using todays computing power requires work
- Close collaboration between experiment, theory and programmers needed to make best use of our beautiful data

Additional Material/Backup

Accessing the Power of GPUs

Programming for the GPU is less straightforward than for the CPU

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

ATI Brook+

We use 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

Will switch to OpenGL as soon as I get back to Beijing, as an implementation for ATI HW is now available

Tensor Manipulation

Use C++ operator overloading for easy writing of covariant tensor code
(Can be taken from Zou and Bugg/Dulat and Zou)
Example: Amplitude for $J/\psi \rightarrow \gamma f_2 \rightarrow \gamma K^+K^-$ (without resonance shape)

$$U^{\mu\nu}_{(\gamma f_2)2} = g^{\mu\nu} p^{\alpha}_{\psi} p^{\beta}_{\psi} \tilde{t}^{(f_2)}_{\alpha\beta} B_2(Q_{\Psi\gamma f_2})$$

```
...
GPUMetricTensor & g = *new GPUMetricTensor();
GPUStreamVector & f2 = k_plus + k_minus;
GPUOrbitalTensors f2orbitals(f2, k_plus, k_minus);
GPUOrbitalTensors psiorbitals(psi, gamma, x);
GPUStreamTensor2 & t_f2 = f2orbitals.Spin2OrbitalTensor();
GPUStreamScalar & B2 = psiorbitals.Barrier2();
GPUStreamTensor2 & U_f2 = g * (psi%psi)¦t_f2 * B2;
...
```

B.-S. Zou, D.V. Bugg, „Covariant tensor formalism for partial-wave analyses of ψ decay to mesons", EPJ A 16, 537, 2003.
S. Dulat, B.-S. Zou, „Covariant tensor formalism for partial wave analyses of ψ decays into γB anti-B, γγV and $\psi(2S) \rightarrow \gamma\chi_{c,0,1,2}$ with $\chi_{c,0,1,2} \rightarrow K$ anti-K $\pi^+\pi^-$ and $2\pi^+2\pi^-$", EPJ A26, 125, 2005.

Tensor manipulation behind the scenes

- In GPUPWA, all tensors are function objects
- Upon creation, relations to other tensors/input files are stored, no calculations performed
- () operators will perform actual calculations - all event loops implicit in GPU parallelism
- Intelligent caching mechanism keeps intermediate results until they are no longer needed
- User C++ code as generic as possible
- Brook+ tensor code for GPU not generic at all
- Interface through template specialisation: Catch missing GPU implementations at compile- rather than at runtime

Plotting

GPUPWA also produces nice plots using ROOT

- Every scalar can be plotted
- Summed, per wave and interfernce terms separately
- Output to PS, PNG and ROOT files