Accelerating PWA Fits
Exploiting Current Trends in Computing Technology

Boris Grube
CERN
on leave of absence from
Physik-Department E18
Technische Universität München

Workshop on
New Partial-Wave Analysis Tools
for Next-Generation Hadron-Spectroscopy Experiments
Camogli, June 21st, 2012
Outline

1. Current trends in computing technology
2. Why GPGPU?
3. The computational challenges of PWA
Outline

1. Current trends in computing technology
2. Why GPGPU?
3. The computational challenges of PWA
Current Trends in Computing Technology

Moore’s Law — Exponentially Growing Number of Transistors

![Graph showing exponential growth of transistors over time](image-url)
Current Trends in Computing Technology

Moore’s Law — Exponentially Growing Number of Transistors

**Clock speed stagnates**
- Limited by power consumption
- **Times of free speedup are over**

**Performance increases along multiple dimensions**
- Larger caches
- More powerful vector units
  - Single Instruction, Multiple Data (SIMD)
  - E.g. Advanced Vector Extension (AVX)
    - 256 bit register size
    - Parallel processing of 4 double-precision floating point values
- Growing number of cores per CPU
  - Multithreading/processing
  - Growth tapering of
    - Limited by interconnection (number of pins/pads)
Current Trends in Computing Technology

Moore’s Law — Exponentially Growing Number of Transistors

Clock speed stagnates

- Limited by power consumption
- Times of free speedup are over

Performance increases along multiple dimensions

- Larger caches
- More powerful vector units
  - Single Instruction, Multiple Data (SIMD)
  - E.g. Advanced Vector Extension (AVX)
    - 256 bit register size
    - Parallel processing of 4 double-precision floating point values
- Growing number of cores per CPU
  - Multithreading/processing
  - Growth tapering of
    - Limited by interconnection (number of pins/pads)
Current Trends in Computing Technology

Moore’s Law — Exponentially Growing Number of Transistors

Clock speed stagnates

- Limited by power consumption
- **Times of free speedup are over**

Performance increases along multiple dimensions

- Larger **caches**
- More powerful **vector units**
  - **Single Instruction, Multiple Data (SIMD)**
  - E.g. **Advanced Vector Extension (AVX)**
    - 256 bit register size
    - Parallel processing of 4 double-precision floating point values
- **Growing number of cores per CPU**
  - Multithreading/processing
  - Growth tapering of
    - Limited by interconnection (number of pins/pads)
Current Trends in Computing Technology

Moore’s Law — Exponentially Growing Number of Transistors

Clock speed stagnates

- Limited by power consumption
- **Times of free speedup are over**

Performance increases along multiple dimensions

- Larger **caches**
- More powerful **vector units**
  - Single Instruction, Multiple Data (SIMD)
  - E.g. Advanced Vector Extension (AVX)
    - 256 bit register size
    - Parallel processing of 4 double-precision floating point values
- Growing **number of cores** per CPU
  - Multithreading/processing
  - Growth tapering of
    - Limited by interconnection (number of pins/pads)
Current Trends in Computing Technology
Moore’s Law — Exponentially Growing Number of Transistors

Performance increases along multiple dimensions (cont.)

- Growing number of CPU sockets in systems
  - Limited by reliability, networking, and cost
  - Efficiency decreases with system size
- Hybrid computing using more specialized hardware
  - General-purpose computing on graphics processing units (GPGPU)
  - Reconfigurable logic devices (FPGA)
  - Digital signal processors (DSP)
  - …

Overall trend

- From multicore to manycore
- From parallel to massively-parallel computing
Current Trends in Computing Technology
Moore’s Law — Exponentially Growing Number of Transistors

Performance increases along multiple dimensions (cont.)

- Growing **number of CPU sockets** in systems
  - Limited by reliability, networking, and cost
  - Efficiency decreases with system size
- **Hybrid computing** using more specialized hardware
  - **General-purpose computing on graphics processing units (GPGPU)**
  - Reconfigurable logic devices (FPGA)
  - Digital signal processors (DSP)
  - …

Overall trend

- From multicore to **manycore**
- From parallel to **massively-parallel computing**
Current Trends in Computing Technology

Moore’s Law — Exponentially Growing Number of Transistors

Performance increases along multiple dimensions (cont.)

- Growing number of CPU sockets in systems
  - Limited by reliability, networking, and cost
  - Efficiency decreases with system size
- Hybrid computing using more specialized hardware
  - General-purpose computing on graphics processing units (GPGPU)
  - Reconfigurable logic devices (FPGA)
  - Digital signal processors (DSP)
  - ...

Overall trend

- From multicore to manycore
- From parallel to massively-parallel computing
How to Make Your Programs Run Faster?

Put the additional transistors to use!

- Utilize vector units in CPUs
  - Parallelize software so that it runs on multiprocessor systems
    - Well-established programming models (OpenMP, Intel TBB, OpenMPI, …)
  - Exploit potential of hybrid computing
    - GPGPU most promising
      - Hardware-specific programming models (CUDA, OpenCL)
    - Potential game changer: Intel’s Many Integrated Core (MIC) architecture (Xeon Phi)
How to Make Your Programs Run Faster?

Put the additional transistors to use!

- Utilize **vector units** in CPUs
- Parallelize software so that it runs on **multiprocessor systems**
  - Well-established programming models (*OpenMP*, *Intel TBB*, *OpenMPI*, …)
- Exploit potential of **hybrid computing**
  - **GPGPU** most promising
    - Hardware-specific programming models (*CUDA*, *OpenCL* )
  - **Potential game changer**: Intel’s Many Integrated Core (MIC) architecture (Xeon Phi)
How to Make Your Programs Run Faster?

Put the additional transistors to use!

- Utilize **vector units** in CPUs
- Parallelize software so that it runs on **multiprocessor systems**
  - Well-established programming models (*OpenMP*, *Intel TBB*, *OpenMPI*, …)
- Exploit potential of **hybrid computing**
  - **GPGPU most promising**
    - Hardware-specific programming models (*CUDA*, *OpenCL*)
  - **Potential game changer:** Intel’s Many Integrated Core (MIC) architecture (Xeon Phi)
How to Make Your Programs Run Faster?

Beware of I/O bottlenecks

- Memory, disk, and network bandwidth grow with lower rate
- Amount of transferred bytes per FLOP decreases
- Optimize memory access patterns

E.g. Intel Core i7 3960X

- Cache miss: 167 clock cycles latency to fetch data from RAM
- During the same time the AVX unit could perform
  - 668 double-precision floating point multiplications
  - 12 ... 32 double-precision floating point divisions
How to Make Your Programs Run Faster?

Beware of I/O bottlenecks

- Memory, disk, and network bandwidth grow with lower rate
- Amount of transferred bytes per FLOP decreases
- Optimize memory access patterns
- E.g. Intel Core i7 3960X
  - Cache miss: 167 clock cycles latency to fetch data from RAM
  - During the same time the AVX unit could perform
    - 668 double-precision floating point multiplications
    - 12 … 32 double-precision floating point divisions
Current Trends in Computing Technology

Change of software-development paradigm

- **Merge two design philosophies**
  1. Design software *independent of hardware*
  2. Optimize software *for specific hardware*

- Hardware is a moving target $\Rightarrow$ longevity issues
- Tradeoffs between *performance* and *flexibility/programmability*
- New programming models (*OpenCL, Thrust, Ocelot, …*) try to minimize these effects
Current Trends in Computing Technology

Change of software-development paradigm

- Merge two design philosophies
  1. Design software **independent of hardware**
  2. Optimize software **for specific hardware**

- Hardware is a moving target \(\Rightarrow\) **longevity issues**

- Tradeoffs between **performance and flexibility/programmability**

- New programming models (*OpenCL, Thrust, Ocelot, …*) try to minimize these effects
Current Trends in Computing Technology

Change of software-development paradigm

- **Merge two design philosophies**
  1. Design software independent of hardware
  2. Optimize software for specific hardware
- Hardware is a moving target $\Rightarrow$ longevity issues
- Tradeoffs between performance and flexibility/programmability
- New programming models (*OpenCL*, *Thrust*, *Ocelot*, ...) try to minimize these effects
Outline

1. Current trends in computing technology
2. Why GPGPU?
3. The computational challenges of PWA
## Why GPGPU?

<table>
<thead>
<tr>
<th>Central Processing Unit</th>
<th>vs.</th>
<th>Graphics Processing Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Control</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td><strong>Cache</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DRAM</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large fraction of die</td>
<td></td>
<td></td>
</tr>
<tr>
<td>devoted to data caching</td>
<td></td>
<td>Most part of die devoted to</td>
</tr>
<tr>
<td>and flow control</td>
<td></td>
<td>data processing</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Why GPGPU? Price per GFLOPS!

<table>
<thead>
<tr>
<th>Central Processing Unit</th>
<th>vs.</th>
<th>Graphics Processing Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General-purpose</strong>; not optimized for specific task</td>
<td><strong>Optimized for highly parallelizable problems</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Intel Core i7 3960X
- 6 cores @ 3.3 GHz
- Max. processing power
  - Single precision: 158 GFLOPS
  - Double precision: 79 GFLOPS
- Memory bandwidth: 51 GByte/sec
- Price: $850,-

### nvidia GeForce GTX 680
- 1536 cores @ 1 GHz
- Max. processing power
  - Single precision: 3000 GFLOPS
  - Double precision: 125 GFLOPS
- Memory bandwidth: 190 GByte/sec
- Price: $500,-
Why GPGPU? Price per GFLOPS!

<table>
<thead>
<tr>
<th>Central Processing Unit</th>
<th>vs.</th>
<th>Graphics Processing Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General-purpose; not</strong></td>
<td></td>
<td><strong>Optimized for highly</strong></td>
</tr>
<tr>
<td><strong>optimized for specific task</strong></td>
<td></td>
<td><strong>parallelizable problems</strong></td>
</tr>
<tr>
<td><strong>Intel Core i7 3960X</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 cores @ 3.3 GHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max. processing power</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single precision:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>158 GFLOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double precision:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>79 GFLOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51 GByte/sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price: $ 850,-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>nvidia GeForce GTX 680</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1536 cores @ 1 GHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max. processing power</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single precision:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000 GFLOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double precision:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>125 GFLOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>190 GByte/sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price: $ 500,-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Why GPGPU? Price per GFLOPS!

<table>
<thead>
<tr>
<th>Central Processing Unit</th>
<th>vs.</th>
<th>Graphics Processing Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General-purpose; not optimized for specific task</strong></td>
<td><strong>Optimized for highly parallelizable problems</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Intel Core i7 3960X</strong></td>
<td><strong>nvidia GeForce GTX 680</strong></td>
<td></td>
</tr>
<tr>
<td>6 cores @ 3.3 GHz</td>
<td>1536 cores @ 1 GHz</td>
<td></td>
</tr>
<tr>
<td>Max. processing power</td>
<td>Max. processing power</td>
<td></td>
</tr>
<tr>
<td>- Single precision: 158 GFLOPS</td>
<td>- Single precision: 3000 GFLOPS</td>
<td></td>
</tr>
<tr>
<td>- Double precision: 79 GFLOPS</td>
<td>- Double precision: 125 GFLOPS</td>
<td></td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>Memory bandwidth</td>
<td></td>
</tr>
<tr>
<td>51 GByte/sec</td>
<td>190 GByte/sec</td>
<td></td>
</tr>
<tr>
<td>Price: $850,-</td>
<td>Price: $500,-</td>
<td></td>
</tr>
</tbody>
</table>
GPGPU Programming Models

Up to now mostly via low-level interfaces

- Open Computing Language (*OpenCL*; vendor-independent)
- Compute Unified Device Architecture (*CUDA*; nvidia only)

Recent developments

- Higher-level interfaces
  - *CUDA* supports some C++ features (simple objects, templates)
  - *Thrust* : CUDA-based C++ template library
- Libraries for heterogeneous computing (CPU + GPU)
  - *OpenCL* : GPUs and x86 multicore CPUs
  - *Thrust* : nvidia GPUs and x86 multicore CPUs
  - *Ocelot* : runs *CUDA* code on AMD GPUs and x86 multicore CPUs

GPGPU landscape is very dynamic

- New hardware generation with every $\approx 1.5$ y
- Rapidly evolving software environment
GPGPU Programming Models

Up to now mostly via low-level interfaces

- Open Computing Language (OpenCL; vendor-independent)
- Compute Unified Device Architecture (CUDA; nvidia only)

Recent developments

- Higher-level interfaces
  - CUDA supports some C++ features (simple objects, templates)
  - Thrust: CUDA-based C++ template library

- Libraries for heterogeneous computing (CPU + GPU)
  - OpenCL: GPUs and x86 multicore CPUs
  - Thrust: nvidia GPUs and x86 multicore CPUs
  - Ocelot: runs CUDA code on AMD GPUs and x86 multicore CPUs

GPGPU landscape is very dynamic

- New hardware generation with every $\approx 1.5$ y
- Rapidly evolving software environment
GPGPU Programming Models

Up to now mostly via low-level interfaces

- Open Computing Language (*OpenCL*; vendor-independent)
- Compute Unified Device Architecture (*CUDA*; nvidia only)

Recent developments

- **Higher-level interfaces**
  - *CUDA* supports some C++ features (simple objects, templates)
  - *Thrust*: CUDA-based C++ template library

- **Libraries for heterogeneous computing (CPU + GPU)**
  - *OpenCL*: GPUs and x86 multicore CPUs
  - *Thrust*: nvidia GPUs and x86 multicore CPUs
  - *Ocelot*: runs *CUDA* code on AMD GPUs and x86 multicore CPUs

GPGPU landscape is very dynamic

- New hardware generation with every \(\approx 1.5\) y
- Rapidly evolving software environment
GPGPU Programming Models

Up to now mostly via low-level interfaces

- Open Computing Language (OpenCL; vendor-independent)
- Compute Unified Device Architecture (CUDA; nvidia only)

Recent developments

- Higher-level interfaces
  - **CUDA** supports some C++ features (simple objects, templates)
  - **Thrust**: CUDA-based C++ template library
- Libraries for heterogeneous computing (CPU + GPU)
  - **OpenCL**: GPUs and x86 multicore CPUs
  - **Thrust**: nvidia GPUs and x86 multicore CPUs
  - **Ocelot**: runs CUDA code on AMD GPUs and x86 multicore CPUs

GPGPU landscape is very dynamic

- New hardware generation with every $\approx 1.5$ y
- Rapidly evolving software environment
1. Current trends in computing technology
2. Why GPGPU?
3. The computational challenges of PWA
The Computational Challenges of PWA

Main challenges

- **Huge data sets** from existing and future experiments
  - BESIII, COMPASS, GlueX, Panda, …
- **Huge model space**
  - Exploration requires many PWA fits \( O(10^4 \ldots 10^5) \)
- **More complex and computationally expensive models**

Two main bottlenecks

1. **Calculation of decay amplitudes** \( A_{\text{wave}}(\tau) \)
   - Need to calculate \( O(100) \) complex amplitudes for each event

2. **Unbinned likelihood fit**
   - Fits with \( O(10^6) \) events

Both problems

- Very floating-point intense
- Easy parallelizable (events are independent)
The Computational Challenges of PWA

Main challenges

- **Huge data sets** from existing and future experiments
  - BESIII, COMPASS, GlueX, Panda, ...
- **Huge model space**
  - Exploration requires many PWA fits $\mathcal{O}(10^4 \ldots 10^5)$
- **More complex and computationally expensive models**

Two main bottlenecks

1. **Calculation of decay amplitudes** $A_{\text{wave}}(\tau)$
   - Need to calculate $\mathcal{O}(100)$ complex amplitudes for each event
2. **Unbinned likelihood fit**
   - Fits with $\mathcal{O}(10^6)$ events

Both problems

- Very floating-point intense
- Easy parallelizable (events are independent)
The Computational Challenges of PWA

Main challenges

- Huge data sets from existing and future experiments
  - BESIII, COMPASS, GlueX, Panda, …
- Huge model space
  - Exploration requires many PWA fits $O(10^4 \ldots 10^5)$
- More complex and computationally expensive models

Two main bottlenecks

1. Calculation of decay amplitudes $A_{\text{wave}}(\tau)$
   - Need to calculate $O(100)$ complex amplitudes for each event
2. Unbinned likelihood fit
   - Fits with $O(10^6)$ events

Both problems

- Very floating-point intense
- Easy parallelizable (events are independent)
Calculation of Decay Amplitude

Simplest case: No free parameters

- Easy to parallelize on multiprocessor systems: parallel jobs
- Implementation on GPGPU more difficult
  - Amplitude code has to be rewritten
  - \textit{GPUPWA} (BESIII): tensor formalism
  - \textit{ROOTPWA} (COMPASS): helicity formalism work in progress

More interesting: Amplitudes contain fit parameters

- Amplitudes have to be recalculated for every fit iteration
- Efficient caching is a must
- Implementation on multiprocessor and GPGPU more challenging
- First steps in this direction: \textit{AmpTools} (Indiana)

Speed limited by floating-point throughput
Calculation of Decay Amplitude

**Simplest case: No free parameters**
- Easy to parallelize on multiprocessor systems: parallel jobs
- Implementation on GPGPU more difficult
  - Amplitude code has to be rewritten
  - GPUPWA (BESIII): tensor formalism
  - ROOTPWA (COMPASS): helicity formalism work in progress

**More interesting: Amplitudes contain fit parameters**
- Amplitudes have to be recalculated for every fit iteration
- Efficient caching is a must
- Implementation on multiprocessor and GPGPU more challenging
- First steps in this direction: AmpTools (Indiana)

Speed limited by floating-point throughput
### Calculation of Decay Amplitude

<table>
<thead>
<tr>
<th>Simplest case: <strong>No free parameters</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Easy to parallelize on multiprocessor systems: parallel jobs</td>
</tr>
<tr>
<td>- Implementation on GPGPU more difficult</td>
</tr>
<tr>
<td>- Amplitude code has to be rewritten</td>
</tr>
<tr>
<td>- \textit{GPUPWA} (BESIII): tensor formalism</td>
</tr>
<tr>
<td>- \textit{ROOTPWA} (COMPASS): helicity formalism work in progress</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>More interesting: <strong>Amplitudes contain fit parameters</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Amplitudes have to be recalculated for every fit iteration</td>
</tr>
<tr>
<td>- Efficient caching is a must</td>
</tr>
<tr>
<td>- Implementation on multiprocessor and GPGPU more challenging</td>
</tr>
<tr>
<td>- First steps in this direction: \textit{AmpTools} (Indiana)</td>
</tr>
</tbody>
</table>

**Speed limited by floating-point throughput**
Unbinned likelihood fit

Likelihood computation

\[ \mathcal{L}(x) = \prod_{i=1}^{N_{\text{events}}} P(x; \tau_i) \implies \ln \mathcal{L}(x) = \sum_{i=1}^{N_{\text{events}}} \ln P(x; \tau_i) \]

- Large sum over events is parallelizable in form of tree sum
- Easy to implement on GPGPU
- Somewhat more difficult on multiprocessor systems
  - Data distribution and process steering
  - Implemented in AmpTools (Indiana) using OpenMPI
- Simplest case: No free parameters in decay amplitudes
  - Decay amplitudes are pre-calculated
  - Performance bound by memory-bandwidth
Unbinned likelihood fit

Likelihood computation

\[ \mathcal{L}(x) = \prod_{i=1}^{N_{\text{events}}} P(x; \tau_i) \implies \ln \mathcal{L}(x) = \sum_{i=1}^{N_{\text{events}}} \ln P(x; \tau_i) \]

- Large sum over events is parallelizable in form of tree sum
- Easy to implement on GPGPU
- Somewhat more difficult on multiprocessor systems
  - Data distribution and process steering
  - Implemented in AmpTools (Indiana) using OpenMPI
- Simplest case: No free parameters in decay amplitudes
  - Decay amplitudes are pre-calculated
  - Performance bound by memory-bandwidth
Huge data sets and more elaborate fit models demand more computing power

IT industry moves to massively-parallel architectures
- In principle ideal for PWA analyses
- Leveraging this potential requires development work
  - Optimize critical software parts for specific hardware
- At the moment GPGPUs have best performance/price ratio
  - Speedups between 20 \times and 150 \times seen in real analyses
  - Hard- and software still evolving
  - High-level programming interfaces and libraries for heterogeneous computing started to emerge

Examples for implementations available on the web
- AmpTools: http://sourceforge.net/projects/amptools
- GPUPWA: http://sourceforge.net/projects/gpupwa
- ROOTPWA: http://sourceforge.net/projects/rootpwa
Summary

Conclusions

- Huge data sets and more elaborate fit models demand more computing power
- IT industry moves to massively-parallel architectures
  - In principle ideal for PWA analyses
  - Leveraging this potential requires development work
    - Optimize critical software parts for specific hardware
  - At the moment GPGPUs have best performance/price ratio
    - Speedups between $20 \times$ and $150 \times$ seen in real analyses
    - Hard- and software still evolving
    - High-level programming interfaces and libraries for heterogeneous computing started to emerge
- Examples for implementations available on the web
  - AmpTools: http://sourceforge.net/projects/amptools
  - GPUPWA: http://sourceforge.net/projects/gpupwa
  - ROOTPWA: http://sourceforge.net/projects/rootpwa
Huge data sets and more elaborate fit models demand more computing power

IT industry moves to massively-parallel architectures
- In principle ideal for PWA analyses
- Leveraging this potential requires development work
  - Optimize critical software parts for specific hardware
- At the moment GPGPUs have best performance/price ratio
  - Speedups between $20 \times$ and $150 \times$ seen in real analyses
  - Hard- and software still evolving
  - High-level programming interfaces and libraries for heterogeneous computing started to emerge

Examples for implementations available on the web
- AmpTools: http://sourceforge.net/projects/amptools
- GPUPWA: http://sourceforge.net/projects/gpupwa
- ROOTPWA: http://sourceforge.net/projects/rootpwa