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

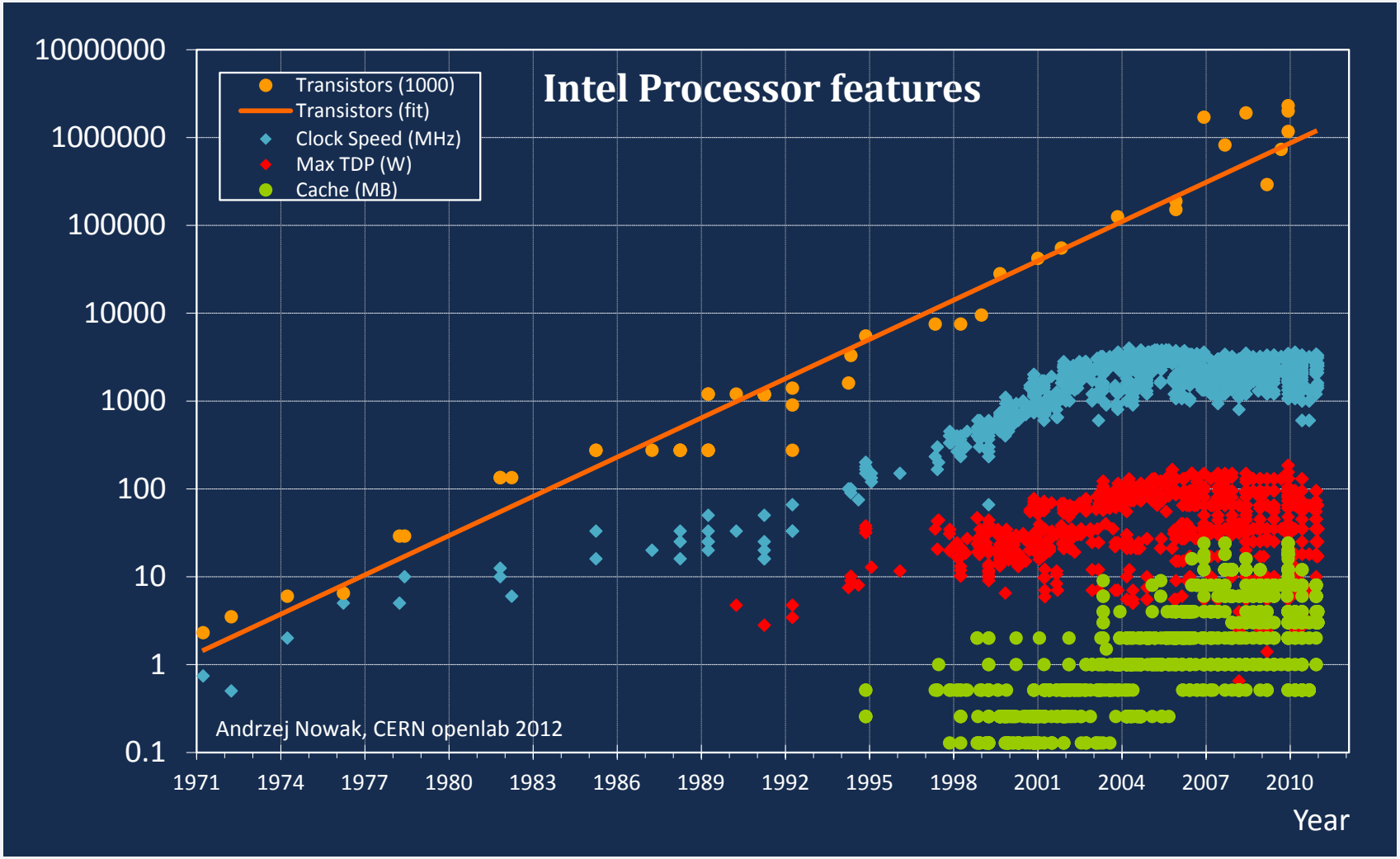
Camogli, June 21st, 2012

Outline

Outline

Current Trends in Computing Technology

Moore's Law — Exponentially Growing Number of Transistors



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 $\Longrightarrow$ 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 $\Longrightarrow$ longevity issues
- Tradeoffs between performance and flexibility/programmability
- New programming models (*OpenCL, Thrust, Ocelot, ...*) try to minimize these effects

Current Trends in Computing Technology

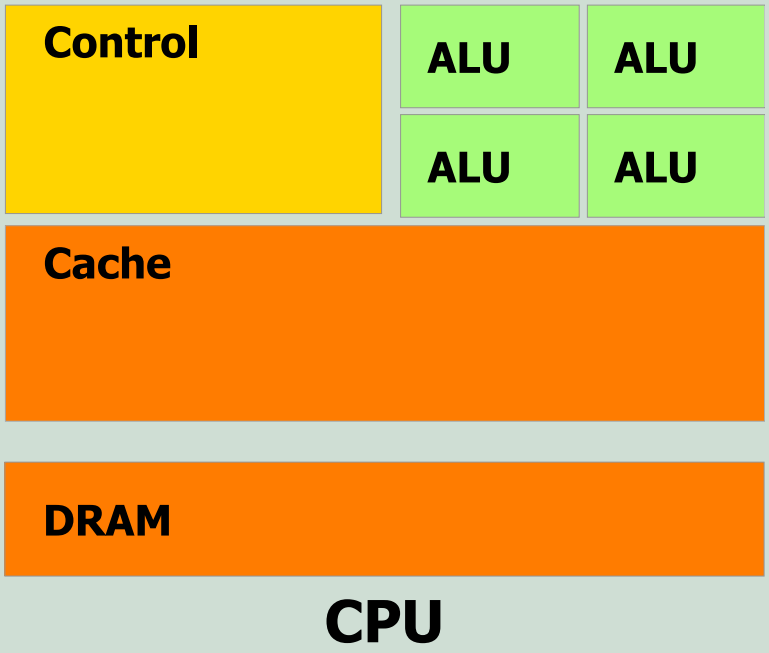
Change of software-development paradigm

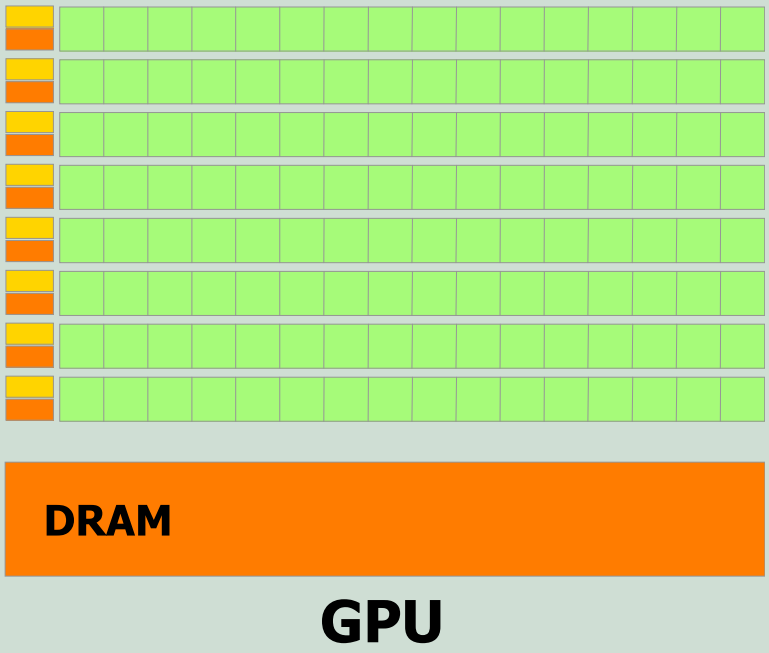
- **Merge two design philosophies**
  1. Design software independent of hardware
  2. Optimize software for specific hardware
- Hardware is a moving target $\Longrightarrow$ longevity issues
- Tradeoffs between performance and flexibility/programmability
- New programming models (*OpenCL, Thrust, Ocelot,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

Outline

Why GPGPU?

Central Processing Unit vs. Graphics Processing Unit





- Large fraction of die devoted to data caching and flow control
- Most part of die devoted to data processing

Why GPGPU? Price per GFLOPS!

Central Processing Unit	vs.	Graphics Processing Unit
• General-purpose; not optimized for specific task		• Optimized for highly parallelizable problems
Intel Core i7 3960X		**nvidia GeForce GTX 680**
• 6 cores @ 3.3 GHz		• 1536 cores @ 1 GHz
• Max. processing power		• Max. processing power
– Single precision: 158 GFLOPS		– Single precision: 3000 GFLOPS
– Double precision: 79 GFLOPS		– Double precision: 125 GFLOPS
• Memory bandwidth 51 GByte/sec		• Memory bandwidth 190 GByte/sec
• Price: $ 850,-		• Price: $ 500,-

Why GPGPU? Price per GFLOPS!

Central Processing Unit vs. Graphics Processing Unit

- General-purpose; not optimized for specific task

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,-

- Optimized for highly parallelizable problems

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!

Central Processing Unit	vs.	Graphics Processing Unit

- General-purpose; not optimized for specific task

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,-

- Optimized for highly parallelizable problems

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,-

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 ≈ 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 ≈ 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 ≈ 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 $\approx 1.5\,\mathrm{y}$
- Rapidly evolving software environment

Outline

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 $\mathcal{O}(10^4 \ldots 10^5)$
- 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 $\mathcal{O}(10^4 \ldots 10^5)$
- 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)

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

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

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

Unbinned likelihood fit

Likelihood computation

$$\mathcal{L}(x) = \prod_{i=1}^{N_{\text{events}}} P(x;\tau_i) \quad \Longrightarrow \quad \ln \mathcal{L}(x) = \sum_{i=1}^{N_{\text{events}}} \ln P(x;\tau_i)$$

- 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

$$\mathcal{L}(x) = \prod_{i=1}^{N_{\text{events}}} P(x;\tau_i) \quad \Longrightarrow \quad \ln \mathcal{L}(x) = \sum_{i=1}^{N_{\text{events}}} \ln P(x;\tau_i)$$

- 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

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`

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`

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`