GENERAL PURPOSE GPU COMPUTING IN PARTIAL WAVE ANALYSIS

Hrayr Matevosyan - NTC, Indiana University
COMPUTATIONAL CHALLENGES IN PWA

Rapid Increase in Available Data

- BES III (2 months): 200.0
- BES II: 51.0
- CLEO-C: 27.0
- BES I: 8.8
- DM 2: 14.0
- MARK III: 0.3
- Crystal Ball: 2.2
- MARK II: 1.3
- MARK I: 0.3
- Number of Events (Millions)
Rapid Increase in Available Data

- GLUE-X Projections:

**MC Studies of γp → ηπ₀p**
(Blaże Leverington)

<table>
<thead>
<tr>
<th>Hadronic BG Rate</th>
<th>Signal Rate</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Cross Section (µb)</th>
<th>Rate (kHz)</th>
<th>Number of Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>124 µb</td>
<td>1.55 kHz</td>
<td>(9.37 \times 10^8) day⁻¹</td>
</tr>
<tr>
<td>0.5 µb</td>
<td>6.3 kHz</td>
<td>(3.81 \times 10^6) week⁻¹</td>
</tr>
<tr>
<td>0.5 µb × 4% × 40%</td>
<td>6.3 kHz</td>
<td>(6.3 \times 10^7) year⁻¹</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MC Data Record Size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>124 µb</td>
</tr>
<tr>
<td>6.3 kHz</td>
</tr>
<tr>
<td>(4.37 \times 10^3) day⁻¹</td>
</tr>
<tr>
<td>(6.99 \times 10^1) week⁻¹</td>
</tr>
<tr>
<td>(3.50 \times 10^1) year⁻¹</td>
</tr>
</tbody>
</table>

Table 2: Expected event rates for various production cross sections in photo-production. The \(σ\) column is the cross section. The rate is the equivalent experimental data rate given a beam rate of \(10^7γ/s\) and the 30cm target. A beam rate of \(10^8γ/s\) will increase the magnitudes of the events and data sets by one order.

With \(O(1)\) second/event for MC, 29 years!
Fitting and Plotting:

- Intensity for a Single Event:
  \[ I(\Omega) = \sum_{\alpha} \left| \sum_{\beta} V_{\alpha,\beta} A_{\alpha,\beta}(\Omega) \right|^2 \]

- Unbinned Data Fit - Minimize Log Likelihood:
  \[ -2 \ln(L) = -2 \sum_{\text{Data}} \ln(I(\Omega_i)) + 2 \sum_{\text{MC}} I(\Omega_i) \]

Log-Intensity Sum Recalculated on Each Fit Iteration

Intensity Calculated Only Once

More Sophisticated Amplitudes Will Have non-factorable Fit Parameters: Amplitudes Need Recalculation!
GPU ACCELERATED PWA

IU AmpTool

GPU PWA

ATI STREAM TECHNOLOGY
GPU ACCELERATED PWA

IU AmpTool
User-Oriented

- Analysis Framework Independent:
  - User supplies data and defines amplitudes from 4-vector level
  - The Framework drives the fit, uses fitted parameters to plot data and Intensity-weighted Monte-Carlo.
GPU ACCELERATED PWA

IU AmpTool

GPU Integration

- Turn on by a simple compile-time switch
- The framework completely manages the GPU initialization/memory/function calls.
- All GPU-enabled amplitudes are automatically called, the rest are evaluated on CPU. (for data and MC - norm integral and plotting)
- Log-intensity sum is calculated on the GPU.
- Integration with MPI allows using multiple machines with GPUs
WHY BOTHER?

- Massively parallel architecture
- Large advantage in performance over CPU
- A sound and rapidly developing CUDA (Compute Unite Device Architecture) toolkit as an extension of C language (in future C++, Java, Fortran, etc.)
WHY BOTHER?

- Massively parallel architecture

- Large advantage in performance over CPU

- A sound and rapidly developing CUDA (Compute Unite Device Architecture) toolkit as an extension of C language (in future C++, Java, Fortran, etc.)
WHY BOTHER?

- Massively parallel architecture

- Large advantage in performance over CPU

- A sound and rapidly developing CUDA (Compute Unite Device Architecture) toolkit as an extension of C language (in future C++, Java, Fortran, etc.)
### Why Bother?

- Massively parallel architecture
- Large advantage in performance over CPU
- A sound and rapidly developing CUDA (Compute Unite Device Architecture) toolkit as an extension of C language (in future C++, Java, Fortran, etc.)
GPU FEATURES

- Rapid advancements in GPUs driven by the gaming industry: ~$57 billion by 09!
- Cost-effective
  - ~ $350 for 1 TFlops
- Power-effective
  - < 200 Watts
- Scalable
  - Up to 4 cards in a MB
CUDA INITIATIVE

- The market for GPGPU in HPC is taking off!
- Support for double-precision floating point numbers starting GT 200 architecture.
- Dedicated TESLA boards: 4GB of DDR3 memory, GT200 GPU with 240 computing cores.
- Workstations and turn-key clusters available from various vendors.
CUDA INITIATIVE

- The market for GPGPU in HPC is taking off!
- Support for double-precision floating point numbers starting GT 200 architecture.
- Dedicated TESLA boards: 4GB of DDR3 memory, GT200 GPU with 240 computing cores.
- Workstations and turn-key clusters available from various vendors
  - Some of rapidly extending CUDA features:
    - Fast floating point math functions
    - A large list of special math functions
    - FFT for CUDA
    - BLAS for CUDA
    - Fast linear interpolation using textures
CUDA in HPC

Some examples from various fields:

- Lattice QCD
- Image Processing / Tomography / Video Editing
- Molecular Dynamics / Computational Fluid Dynamics
- MATLAB Accelerator Engines (Matrices/Graphics)
- Computational Chemistry
- Computational Finance
Caveats

GPU architecture dictates:

- Demands massively-parallel tasks for optimal performance, to hide memory access latencies, etc.
- Requires extensive optimization at programming level: programmer has to worry about optimal memory access patterns/availability, hardware limitations, adopt effective parallel algorithms.
- Favors arithmetically intensive problems.
- Weak* performance on doubles: only 1 DP core for every 8 SP
CUDA PROGRAMMING MODEL

- Identify parallelizable parts of the code and port them to GPU

- In GPU cores are divided into several multi processors (MP). Each MP has 8 scalar processor (SP) cores.

- Compute threads are divided into Blocks. All the blocks constitute the Grid of threads.

- All threads in blocks are executed at once - assigned to a single MP, scheduling is done automatically by the GPU driver.

- Max blocks/threads per MP: 8 / 1024!

- Each MP has 16KB registers and 16KB shared memory - this limits number of concurrently executed Blocks per MP.

- All processes access FAST, read-only CONSTANT memory.

- Block structure provides Automatic Scalability, provided by CUDA driver.
CUDA Programming Model

- Identify **parallelizable** parts of the code and port them to **GPU**.
- In GPU cores are divided into several multi-processors (MP). Each MP has 8 scalar processor (SP) cores.
- Compute threads are divided into **Blocks**. All the blocks constitute the **Grid** of threads.
- All threads in blocks are executed at once - assigned to a single MP, scheduling is done automatically by the **GPU driver**.
- Max blocks/threads per MP: $8 / 1024$!
- Each MP has **16KB registers** and **16KB shared memory** - this limits number of concurrently executed Blocks per MP.
- All processes access **FAST**, read-only **CONSTANT** memory.
- Block structure provides **Automatic Scalability**, provided by CUDA driver.
CUDA Programming Model

- Identify **parallelizable** parts of the code and port them to **GPU**
- In GPU cores are divided into several multi processors (MP). Each MP has 8 scalar processor (SP) cores.
- Compute threads are divided into **Blocks**. All the blocks constitute the **Grid** of threads.
- All threads in blocks are executed at once - assigned to a single MP, scheduling is done automatically by the **GPU driver**.
- Max blocks/threads per MP: 8 / 1024 !
- Each MP has **16KB registers** and **16KB shared memory** - this limits number of concurrently executed Blocks per MP.
- All processes access **FAST, read-only CONSTANT memory**.
- Block structure provides **Automatic Scalability**, provided by CUDA driver.
CUDA PROGRAMMING MODEL

- Identify **parallelizable** parts of the code and port them to GPU.

- In GPU cores are divided into several multi processors (MP). Each MP has 8 scalar processor (SP) cores.

- Compute threads are divided into **Blocks**. All the blocks constitute the **Grid** of threads.

- All threads in blocks are executed at once - assigned to a single MP, scheduling is done automatically by the **GPU driver**.

- Max blocks/threads per MP: 8 / 1024!

- Each MP has **16KB registers** and **16KB shared memory** - this **limits** number of concurrently executed Blocks per MP.

- All processes access **FAST**, read-only **CONSTANT** memory.

- Block structure provides **Automatic Scalability**, provided by CUDA driver.
CUDA Programming Model

• Identify **parallelizable** parts of the code and port them to GPU.

• In GPU cores are divided into several multi processors (MP). Each MP has 8 scalar processor (SP) cores.

• Compute threads are divided into **Blocks**. All the blocks constitute the **Grid** of threads.

• All threads in blocks are executed at once - assigned to a single MP, scheduling is done automatically by the GPU driver.

• Max blocks/threads per MP: 8 / 1024 !

• Each MP has 16KB registers and 16KB shared memory - this limits number of concurrently executed Blocks per MP.

• All processes access **FAST, read-only** CONSTANT memory.

• Block structure provides **Automatic Scalability**, provided by CUDA driver.
- Global memory access has large latency (~100 x slower than local and shared).

- All the processes in a warp should access global memory within 128B for a single transfer read/write.

- Registers and Shared memory are a scarce resource.

- Shared memory access should also be arranged to avoid bank conflicts.

Left: random float memory access within a 64B segment, resulting in one memory transaction.
Center: misaligned float memory access, resulting in one transaction.
Right: misaligned float memory access, resulting in two transactions.
CUDA FOR C

- Is a superset of C programming language. Additional, very limited set of GPU primitives. GPU functions (kernels) can only call other GPU functions (no system calls, yet).
- Supports some C++ constructs. Full support for the announced, next-gen cards.
- The compiler - nvcc and GPU driver available for most common flavors of OSes - Linux/Mac/Windows. CUDA code is universal across the platforms.
- Comes with toolkit, manual and SDK - easy learning curve.
- GDB debugger for Linux, VS integrated NEXUS debugger for Windows.
- Rapidly Evolving!
Partial Wave Analysis seems to be a ideal candidate for GPU computing!

- The quantities of interest are additive for each event - Trivially Parallelizable!

- Amplitudes calculations grow in arithmetical complexity.

- Maximum likelihood fits involve a large number of fitting iterations - only a few numbers transferred to GPU per computationally intensive calculation!

- Large number of events leverage GPU’s multi-core, multi-thread capabilities.
GPU DRIVING CODE

- Setting and Querying the GPU:

```c
///////CUDA_INITIALIZATION
cudaSetDevice(0);
//Query the device
cudaDeviceProp devProp;
cudaGetDeviceProperties( &devProp, 0 );
```

- Allocating memory on GPU and copying data to it:

```c
int m_iVArrSize=sizeof(GDouble)*m_iN Amps;
//Device memory
cudaMalloc((void**) &m_pfDevAmpRe, m_iAmpArrSize);
//Memory transfer CPU -> GPU
cudaMemcpy(m_pfDevAmpRe, m_pfAmpRe, m_iAmpArrSize, cudaMemcpyHostToDevice);
```

- Calling function on GPU:

```c
void GPU_ExecCalcAmp(dim3 dimGrid, dim3 dimBlock, GDouble* pfDevData, GDouble* pcDevAmp, int iNEvents, int iNParticles, GDouble* pfPars, int iNPars)
{
    //casting Amp array to WCUComplex for 8 or 16 bit write operation of both real and complex parts at once
    ampcalc_kernel<<< dimGrid, dimBlock >>>( pfDevData, (WCUComplex*) pcDevAmp, iNEvents, iNParticles);
}
```
CUDA AMPLITUDE

CUDA Amplitude Template in IU AmpTool:

```c
__global__ void
ampcalc_kernel(GDouble* pfDevData, WCUComplex* pcDevAmp, int iNEvents, int iNParticles)
{
    int iEvent = threadIdx.x+GPU_BLOCK_SIZE_X*threadIdx.y+
                 (blockIdx.x+blockIdx.y*gridDim.x)*GPU_BLOCK_SIZE_Y;

    WCUComplex m_cS;
    WCUComplex m_cSij[3];

    GDouble dV1[4];
    GDouble dV2[4];
    GDouble dV3[4];

    dV1[0]=pfDevData[iEvent];
    dV1[1]=pfDevData[iNEvents+iEvent];
    dV1[2]=pfDevData[2*iNEvents+iEvent];
    dV1[3]=pfDevData[3*iNEvents+iEvent];

    ...

    pcDevAmp[iEvent]=cAmp;
}
```

Here goes the code for the event number “iEvent”
FEASIBLE TEST PLATFORM?

- For a Start:
  GeForce 8400 GT ~ $30

Or Even a Laptop’s NVIDIA Card.
FEASIBLE TEST PLATFORM?

THE BEAST

- **Specs:**
  - **CPU:** Core i7 920 - 2.66 GHz Quad-Core
  - **Memory:** 6 GB DDR3 1600 MHz
  - **Overall Cost:** ~ $1600
  - **Expandable:** up to 3 GPUs!
  - **OS:** Fedora 10 x86_64

- **CUDA GPU:**
  - EVGA GTX 285 SSC ~ 1 TFlops!
  - 1584 MHz core clock
  - 240 Processing Cores

- **GPU MEMORY:**
  - 1024 MB, 512 bit DDR3
  - 169.3 GB/s Memory Bandwidth
  - GPU to RAM: ~ 5.6 GB/s measured!
**Is it Working?  Yes !!!**

\[ L = \sum_{n \in \mathbb{N}_{Events}} \log \left( \sum_{i=1, j \leq i}^{i \leq N_{Amps}} V_i V_j^* A_i(n) A_j^*(n) \right) \]

### Standalone Benchmarks:

<table>
<thead>
<tr>
<th>Events</th>
<th>Amps</th>
<th>CPU</th>
<th>GPU</th>
<th>GPU/CPU x</th>
</tr>
</thead>
<tbody>
<tr>
<td>(~1.*10^5)</td>
<td>40</td>
<td>38.5s/500</td>
<td>29.1s/5000</td>
<td>13</td>
</tr>
<tr>
<td>(~2.*10^5)</td>
<td>20</td>
<td>21.4s/500</td>
<td>11.5s/5000</td>
<td>18.5</td>
</tr>
<tr>
<td>(~2.*10^5)</td>
<td>8</td>
<td>5.3s/500</td>
<td>2.1s/5000</td>
<td>25</td>
</tr>
<tr>
<td>(~2*10^6)</td>
<td>8</td>
<td>53s/500</td>
<td>19.7/5000</td>
<td>27</td>
</tr>
<tr>
<td>(~1.*10^7)</td>
<td>4</td>
<td>29.5s/100</td>
<td>5.78s/1000</td>
<td>51</td>
</tr>
<tr>
<td>(~1.*10^7)</td>
<td>1</td>
<td>17.3s/100</td>
<td>1.85s/1000</td>
<td>95</td>
</tr>
<tr>
<td>(~2.*10^7)</td>
<td>2</td>
<td>45.8s/100</td>
<td>4.56s/1000</td>
<td>100.8</td>
</tr>
</tbody>
</table>

1 core, **SSE VECTORIZED!**
**Some More Benchmarks**

GPU timings include GPU-> CPU memory transfer time, C++ Compiler - > ICC 11 with Auto-Vectorization!

**Amps Calc Time: sec/Iteration** (with AmpFactor->Amp transform on CPU)

<table>
<thead>
<tr>
<th>Events</th>
<th>CPU - 1 Amp</th>
<th>GPU - 1 Amp</th>
<th>CPU - 3 Amps</th>
<th>GPU - 3 Amps</th>
</tr>
</thead>
<tbody>
<tr>
<td>2581</td>
<td>5.1*10^{-3}</td>
<td>1.3*10^{-4}</td>
<td>0.015</td>
<td>3*10^{-4}</td>
</tr>
<tr>
<td>~1.*10^6</td>
<td>2</td>
<td>0.012</td>
<td>6</td>
<td>0.037</td>
</tr>
<tr>
<td>~8.*10^6</td>
<td>16</td>
<td>0.097</td>
<td>48</td>
<td>0.29</td>
</tr>
</tbody>
</table>

**Log-Like Sum: sec/Iteration**

<table>
<thead>
<tr>
<th>Events</th>
<th>CPU - 1 Amp</th>
<th>GPU - 1 Amp</th>
<th>CPU - 3 Amps</th>
<th>GPU - 3 Amps</th>
</tr>
</thead>
<tbody>
<tr>
<td>2581</td>
<td>1.1*10^{-4}</td>
<td>6.8*10^{-5}</td>
<td>1.8*10^{-4}</td>
<td>6.8*10^{-5}</td>
</tr>
<tr>
<td>~1.*10^6</td>
<td>0.049</td>
<td>2.5*10^{-4}</td>
<td>0.074</td>
<td>4.6*10^{-4}</td>
</tr>
<tr>
<td>~8.*10^6</td>
<td>0.37</td>
<td>1.5*10^{-3}</td>
<td>0.59</td>
<td>3.2*10^{-3}</td>
</tr>
</tbody>
</table>
What's Ahead?

OpenCL?

- Oriented towards consumer market.
- Still in early stages of development - lots of
- Easy Transition CUDA -> OpenCL
- Slow to evolve:
OpenCL?

- Oriented towards consumer market.
- Still in early stages of development - lots of...
- Easy Transition CUDA -> OpenCL

NO, Thanks for Now!
What's Ahead?

FERMI

Consumer Cards Coming Early 2010!

Price ~ $400-$500

<table>
<thead>
<tr>
<th>GPU</th>
<th>G80</th>
<th>GT200</th>
<th>Fermi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transistors</td>
<td>681 million</td>
<td>1.4 billion</td>
<td>3.0 billion</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>128</td>
<td>240</td>
<td>512</td>
</tr>
<tr>
<td>Double Precision Floating Point Capability</td>
<td>None</td>
<td>30 FMA ops / clock</td>
<td>256 FMA ops /clock</td>
</tr>
<tr>
<td>Single Precision Floating Point Capability</td>
<td>128 MAD ops / clock</td>
<td>240 MAD ops / clock</td>
<td>512 FMA ops /clock</td>
</tr>
<tr>
<td>Warp schedulers (per SM)</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Special Function Units (SFUs) / SM</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Shared Memory (per SM)</td>
<td>16 KB</td>
<td>16 KB</td>
<td>Configurable 48 KB or 16 KB</td>
</tr>
<tr>
<td>L1 Cache (per SM)</td>
<td>None</td>
<td>None</td>
<td>Configurable 16 KB or 48 KB</td>
</tr>
<tr>
<td>L2 Cache (per SM)</td>
<td>None</td>
<td>None</td>
<td>768 KB</td>
</tr>
<tr>
<td>ECC Memory Support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Concurrent Kernels</td>
<td>No</td>
<td>No</td>
<td>Up to 16</td>
</tr>
<tr>
<td>Load/Store Address Width</td>
<td>32-bit</td>
<td>32-bit</td>
<td>64-bit</td>
</tr>
</tbody>
</table>
GT300 Improvements:

- Support for the next generation IEEE 754-2008 double precision floating point standard
- Up to 1 terabyte of faster (GDDR5) memory
- 64-bit virtual address space (possibly unified CPU-GPU memory mapping)
- ECC (error correcting codes) - not available on consumer cards
- Multi-level cache hierarchy with L1 and L2 caches
- Support for the C++ programming language
- System calls and recursive functions
- Concurrent kernel execution, fast context switching, 10x faster atomic instructions
- More cores, 1:2 ratio for double-precision cores.
GT300 Improvements:

- Support for the next generation IEEE 754-2008 double precision floating point standard
- Up to 1 terabyte of faster (GDDR5) memory
- 64-bit virtual address space (possibly unified CPU-GPU memory mapping)
- ECC (error correcting codes) - not available on consumer cards
- Multi-level cache hierarchy with L1 and L2 caches
- Support for the C++ programming language
- System calls and recursive functions
- Concurrent kernel execution, fast context switching, 10x faster atomic instructions
- More cores, 1:2 ratio for double-precision cores.
Just Announced

Tesla C2050 & C2070 GPU Computing Processors
- Single GPU PCI-Express Gen-2 cards for workstation configurations
- Up to 3GB and 6GB (respectively) on-board GDDR5 memory
- Double precision performance in the range of 520GFlops - 630 GFlops
- Price - $2,499 and $3,999

Tesla S2050 & S2070 GPU Computing Systems
- Four Tesla GPUs in a 1U system product for cluster and datacenter deployments
- Up to 12 GB and 24 GB (respectively) total on board GDDR5 memory
- Double precision performance in the range of 2.1 TFlops - 2.5 TFlops
- Price - $12,995 and $18,995
PWA is becoming increasingly computationally intensive due to massive amount of new data and use of more sophisticated amplitudes.

GPGPU brings a computational power of a cluster to a desktop computer.

GPU acceleration in IU AmpTools PWA framework proved to be highly successful.