CERN openlab III

Short GPU/multi-core “slide-show” from CHEP-2010

Sverre Jarp, CERN IT
2 November 2010
Presentations/BOFs

- **Plenary:**
  - The experiment offline systems after one year (R.Jones)
  - How to harness the performance potential of current *Multi-Core CPUs and GPUs* (S.Jarp)
  - Computing *Paths to the Future* (R.Goff/DELL)

- **Parallel:**
  - Multicore-aware Applications in CMS
  - Parallelizing Atlas Reconstruction and Simulation: Issues and Optimization Solutions for Scaling on Multi- and Many-CPU Platforms
  - Multi-threaded Event Reconstruction with JANA
  - Track Finding in a High-Rate Time Projection Chamber Using GPUs
  - Fast Parallel Tracking Algorithm for the Muon System and Transition Radiation Detector of the CBM Experiment at FAIR
  - Real Time Pixel Data Reduction with GPUs And Other HEP GPU Applications
  - Algorithm Acceleration from GPGPUs for the ATLAS Upgrade
  - Maximum Likelihood Fits on Graphics Processing Units
  - Partial Wave Analysis on Graphics Processing Units
  - Many-Core Scalability of the Online Event Reconstruction in the CBM Experiment

- **BOF 3:**
  - GPUs: High Performance Co-Processors
Plenary on Monday by Roger Jones (ATLAS)

“The experiment offline systems after one year”

Future Challenges

- We assume we can use growth in CPU
  - But this implies changing architectures
  - And handle the data throughput
- Experiments already working to deal with multi cores
  - Many cores and GPGPUs are down the line
- We need to use them or be very clear why we cannot

Related developments

- IO challenges being (partly) addressed by fast merging
- Re-write of Gaudi with stronger memory model planned
- Down the line, we may need to parallize the code
  - This could be either for many-core processors or for Graphical Processing Units – but the development might address both
    - GPUs having big success & cost savings in other fields
    - Harder for us to use, but funders will continue to ask
    - We need the R&D to know which path to take
- Developments require O(3 years) to implement
- This includes Geant4 – architectural review this year
Plenary on Monday by S.Jarp

“How to harness the performance potential of current Multi-Core CPUs and GPUs”

Today:
Seven dimensions of **multiplicative** performance

- **First three dimensions:**
  - Pipelined execution units
  - Large superscalar design
  - Wide vector width (SIMD)

- **Next dimension is a “pseudo” dimension:**
  - Hardware multithreading

- **Last three dimensions:**
  - Multiple cores
  - Multiple sockets
  - Multiple compute nodes

What are the multi-core options?

There is a discussion in the community about the best way(s) forward:

1) Stay with event-level parallelism (and entirely independent processes)
   - Assume that the necessary memory remains affordable
   - Or rely on tools, such as KSM, to help share pages

2) Rely on forking:
   - Start the first process; Run through the first “event”
   - Fork N other processes
   - Rely on the OS to do “copy on write”, in case pages are modified

3) Move to a fully multi-threaded paradigm
   - Still using coarse-grained (event-level) parallelism
   - But, watch out for increased complexity
Plenary on Monday by S.Jarp

“How to harness the performance potential of current Multi-Core CPUs and GPUs”

Today:
Seven dimensions of **multiplicative** performance

- **First three dimensions:**
  - Pipelined execution units
  - Large superscalar design
  - Wide vector width (SIMD)

- **Next dimension is a “pseudo” dimension:**
  - Hardware multithreading

- **Last three dimensions:**
  - Multiple cores
  - Multiple sockets
  - Multiple compute nodes

**Shortlist**

1) Broad Programming Talent
2) Holistic View with a clear split: Prepare to compute – Compute
3) Controlled Memory Usage
4) C++ for Performance
5) Best-of-breed Tools
Preliminary note on Tuesday by Roger Goff/DELL

- Plenary “vendor” session:

Computing Paths to the Future

Roger Goff
Dell Global CERN/LHC Technologist
+1 970 672 1252 | Roger_Goff@dell.com

Final Takeaways

1. CPU cores are not getting faster.
2. Co-processors are here to stay.
3. Heterogeneous processors are inevitable.
4. Preparing applications for extreme parallelism will enable users to get the most out of future systems.
“Parallelizing Atlas reconstruction and simulation on multi-core platforms”
“Parallelizing Atlas reconstruction and simulation on multi-core platforms”

**Event Level Parallelism with AthenaMP**

- **Input Files**
- **First Events**
- **Init**
- **OS-fork**
- **Worker 0**: Events: [0, 4, 8,...,96]
- **Worker 1**: Events: [1, 5, 9,...,97]
- **Worker 2**: Events: [2, 6, 10,...,98]
- **Worker 3**: Events: [3, 7, 11,...,99]
- **Output Files**
- **Output tmp files**
- **Output tmp files**
- **Output tmp files**
- **Merge**
- **End**
- **Maximize the shared memory**

**Issues with Large OOP Code Bases**

- Function calls result in added instructions
  - Call and return
  - Runtime address resolution (trampolines) required for position independent code/ shared object cross invocations
    - Indirect branches can be more costly
  - Freeing & restoring registers for local use
  - Setting and reading function arguments
- Virtual function calls (function pointers) increase indirect call instructions and associated pointer loads
  - Virtual functions can't be inlined!
- Atlas code has 2500 shared libraries!
Why Bother?

HEP processing is naturally parallelizable
We have billions of events
Each event can be processed independently

Memory is becoming a limitation
Historically GB/US$ increases at the same rate as number of transistors in a CPU
http://www.jcmit.com/memoryprice.htm
Funding levels are not guaranteed to stay this high
We can afford 2GB/core now but may not in the future
Opportunistic use of grid sites improves if we lower our memory requirements
Not all grid sites have 2GB/core
Technical limitations on connecting many cores to shared system memory

Multi-core aware applications can improve memory sharing
Threading
All threads share the same address space but have to worry about concurrent usage
Forking
Each child process gets its own address space
Untouched memory setup by the parent is shared between the child processes

Memory Sharing

Multi-core Aware Applications in CMS
“Multithreaded event reconstruction with JANA”

Multi-threading

- Each thread has a complete set of factories making it capable of completely reconstructing a single event.
- Factories only work with other factories in the same thread eliminating the need for expensive mutex locking within the factories.
- All events are seen by all Event Processors (multiple processors can exist in a program).

Testing on a 48-core “Magny Cours”

Event reconstruction using 48 processing threads on a CPU with 48 cores generally scales quite well.

Eventually, an I/O limit will be encountered.

Memory Usage vs. time while repeatedly running the 35 thread test. The marked area indicates the test where the program ran slower.

- Occasionally some problems with inexplicably lower rates.
- Program appears to simply run slower while not operating any differently.
- Unclear if this is due to hardware or Linux kernel.
“Track finding in a high-rate time projection Chamber using GPUs”

Hardware setup:
- CPU: Intel Core™2 Quad Q8400 2.66 GHz (single thread)
- GPU: NVIDIA GTX 280 (1GB)

Execution time comparison:
- Time (5 Tracks): GPU 55.36, CPU 60.66
- Time (10 Tracks): GPU 2.97, CPU 3.27

18.4x
Parallel PWA on GPU

- Events are independent - calculate terms in the sum in parallel
  - Use many PCs
  - Use parallel hardware and make use of Single Instruction - Multiple Data (SIMD) capabilities
  - Very strong here: Graphics processors (GPUs): Cheap and powerful hardware

- PWA is embarassingly parallel:
  - Exactly the same (relatively simple) calculation for each event
  - Every event has its own data, or fit parameters are shared
  - Ideal for GPU implementation
  - True for many HEP applications

Performance

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 sums on CPU, and GPUPWA sums on GPU with speedup of 150.
“HLT (and other things) with GPUs”

Belle II High Level Trigger

Can we use GPUs?

CPU/GPU Results

- Floats are faster as expected
- Copy from/to CPU to/from GPU not included (but is significant)
- Cholesky decomposition cannot saturate GPU
- Cannot do $24576 \times 24576$ (not enough memory on one C2050)

Report from CHEP2010 – S.Jarp
Inflation

• Inflation started $10^{-36}$ sec. after the birth of the universe and lasted for $10^{-34}$ sec.

• During that period, the universe expanded by an order of $e^{60}$, from Plank scale to a meter (our observable universe).

• Inflation was caused by a particle (field) of energy scale of $10^{16}$ GeV.

• **Cosmic Microwave Background Radiation** is the probe to measure its energy scale.
  - Let me use the parameter “$r$” to represent it.

---

### CPU/GPU Results

<table>
<thead>
<tr>
<th></th>
<th>6144×6144 (GFLOPS)</th>
<th>12288×12288 (GFLOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (i-7 920) (numerical recipes)</td>
<td>0.61</td>
<td>0.62</td>
</tr>
<tr>
<td>CPU (i-7 <a href="mailto:X980@3.33GHz">X980@3.33GHz</a>) MKL(6core)</td>
<td>68</td>
<td>72</td>
</tr>
<tr>
<td>GTX480 (our CUDA code)</td>
<td>108</td>
<td>115</td>
</tr>
<tr>
<td>C2050 (our CUDA code)</td>
<td>87</td>
<td>91</td>
</tr>
<tr>
<td>C2050 (CULA)</td>
<td>159</td>
<td>190</td>
</tr>
</tbody>
</table>

• Peak performance of C2050 is supposed to be >500 GFLOPS
• GTX480 DP speed is suppressed to $\frac{1}{4}$ of C2050 (but is faster)
• CUDA 3.1 and CULA 2.1 are used
• Floats are faster as expected
• Copy from/to CPU to/from GPU not included (but is significant)
• Cholesky decomposition cannot saturate GPU
• Cannot do 24576×24576 (not enough memory on one C2050)
“Algorithm acceleration from GPGPUs for the ATLAS upgrade”

- Number of GPU related projects at Edinburgh over the summer:
  - **Chris Jones** - “Porting the Z finder algorithm to GPU” (MSc in High Performance Computing)
  - **Maria Rovatsou** - “SIMT design of the High Level Trigger Kalman Fitter” (MSc School of Informatics)
  - **James Henderson** - “An Investigation Into Particles Tracking and Simulation Algorithms using GPUs”

Project reports and source code available at: [ATLAS Edinburgh GPU Computing](#)

- Results for spacepoint pairs show up to 35x speed-up (Fermi).
- Initial results for spacepoint *triplets* also show speed-up.
“Maximum likelihood fits using GPUs”

**Test environment**

- **PCs**
  - CPU: Nehalem @ 3.2GHz: 4 cores – 8 hw-threads
  - OS: SLC5 64bit - GCC 4.3.4
  - ROOT trunk (October 11th, 2010)

- **GPU:** ASUS nVidia GTX470 PCI-e 2.0
  - Commodity card (for gamers)
  - Architecture: GF100 (Fermi)
  - Memory: 1280MB DDR5
  - Core/Memory Clock: 607MHz/837MHz
  - Maximum # of Threads per Block: 1024
  - Number of SMs: 14
  - CUDA Toolkit 3.1 06/2010
  - Developer Driver 256.40
  - Power Consumption 200W
  - Price ~$340

**PDF-event-base: GPU VS OpenMP**

- **Fair comparison**
  - Same algorithm
  - Algorithm on CPU optimized and parallelized (4 threads)
  - CPU does the final sum of the NLL and normalization integral calculations
  - Check that the results are compatible: asymmetry less than $10^{-12}$

- **Speed-up increases with the dimension of the sample, taking benefit from the data streaming on GPU and the integral calculation only on the CPU**
  - ~3x for small samples, up to ~7x for large samples
“Fast track reconstruction of the muon system and transition radiation detector”

Optimization of the algorithm

- Minimize access to global memory
  - Approximation of the 70 MB large magnetic field map
    - 7 degree polynomial in the detector planes was proven the best
  - Simplification of the detector geometry
    - Problem
      - Monte-Carlo geometry consists of 800000 nodes
      - Geometry navigation based on ROOT TGeo
      - Take into account absorbers and staggered Z positions of the stations
    - Solution
      - Create simplified geometry by converting Monte-Carlo geometry
      - Implement fast geometry navigation for the simplified geometry
- Computational optimization of the Kalman Filter
  - From double to float
  - Implicit calculation on non-trivial matrix elements
  - Loop unrolling
  - Branches (if then else . . .) have been eliminated

All these steps are necessary to implement SIMD tracking

Performance of the track fit in MUCH

<table>
<thead>
<tr>
<th>Track fit quality</th>
<th>Residuals</th>
<th>Pulls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X [cm]</td>
<td>Y [cm]</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>0.38</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Speedup of the track fitter

<table>
<thead>
<tr>
<th></th>
<th>Time [µs/track]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>1200</td>
<td>-</td>
</tr>
<tr>
<td>Optimization</td>
<td>13</td>
<td>92</td>
</tr>
<tr>
<td>SIMDization</td>
<td>4.4</td>
<td>3</td>
</tr>
<tr>
<td>Multithreading</td>
<td>0.5</td>
<td>8.8</td>
</tr>
<tr>
<td>Final</td>
<td>0.5</td>
<td>2400</td>
</tr>
</tbody>
</table>

Throughput: 2*10^6 tracks/s

Report from CHEP2010 – S. Jarp 17
“Fast track reconstruction of the muon system and transition radiation detector”

**Optimization of the algorithm**

- Minimize access to global memory
  - Approximation of the 70 MB large magnetic field map
    - 7 degree polynomial in the detector planes was proven the best
- Simplification of the detector geometry
  - Problem
    - Monte-Carlo geometry consists of 800000 nodes
    - Geometry navigation based on ROOT TGeo
    - Take into account absorbers and staggered Z positions of the stations
  - Solution
    - Create simplified geometry by converting Monte-Carlo geometry
    - Implement fast geometry navigation for the simplified geometry
- Computational optimization of the Kalman Filter
  - From double to float
  - Implicit calculation on non-trivial matrix elements
  - Loop unrolling
  - Branches (if then else ...) have been eliminated

*All these steps are necessary to implement SIMD tracking*

**Performance of the tracking in MUCH**

**Simulation:**
- 1000 UrQMD events at 25 AGeV Au-Au collisions + 5 \( \mu^+ \) and 5 \( \mu^- \) embedded in each event

<table>
<thead>
<tr>
<th></th>
<th>Initial version</th>
<th>Parallel version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency [%]</td>
<td>94.7</td>
<td>94.0</td>
</tr>
</tbody>
</table>

**Speedup of the track finder**

<table>
<thead>
<tr>
<th></th>
<th>Time [ms/event]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>730</td>
<td>-</td>
</tr>
<tr>
<td>Optimization</td>
<td>7.2</td>
<td>101</td>
</tr>
<tr>
<td>SIMDization</td>
<td>4.8</td>
<td>1.5</td>
</tr>
<tr>
<td>Multithreading</td>
<td>1.5</td>
<td>3.3</td>
</tr>
<tr>
<td>Final</td>
<td>1.5</td>
<td>487</td>
</tr>
</tbody>
</table>

*Computer with 2xCPUs Intel Core i7 (8 cores in total) at 2.67 GHz*
"Many-core scalability of the online reconstruction in CBM"

- Cellular Automaton (CA) as Track Finder

0. Hits (CBM)

1. Segments

2. Counters

3. Track Candidates

4. Tracks

Cellular Automaton:
- local w.r.t. data
- intrinsically parallel
- extremely simple
- very fast

Perfect for many-core CPU/GPU!
“Many-core scalability of the online reconstruction in CBM”

Performance of the KF Track Fit on CPU/GPU Systems

CBM Cellular Automaton Track Finder

Real-time performance on different Intel CPU platforms

Real-time performance on NVIDIA GPU graphic cards

The Kalman Filter algorithm performs at ns level

Highly efficient reconstruction of 150 central collisions per second
BOF by Mohammad Al Turany and Alfio

- GPUs

Cuda Toolkit

- NVCC C compiler
- CUDA FFT and BLAS libraries for the GPU
- CUDA-gdb hardware debugger
- CUDA Visual Profiler
- CUDA runtime driver (also available in the standard NVIDIA GPU driver)
- CUDA programming manual

Summary

- Cuda is an easy to learn and to use tool that allows heterogeneous programming.
- Depending on the use case one can win factors in performance compared to CPU
- Texture memory can be used to solve problems that require lookup tables effectively
- Pinned Memory simplify some problems, gives also better performance.
- With Fermi we are getting towards the end of the distinction between CPUs and GPUs
  - The GPU increasingly taking on the form of a massively parallel co-processor
More and more people are working on multi-core, many-core, and accelerators
- Especially, in the on-line domain
- And, in new or upgraded experiments
- Positive outcome: Code is revisited and made more “C-like”

But, many people forget to do a “fair” comparison:
- GPU code should expose “rich” loops for threading
  - Transfer times must be included
- CPU code should exploit vectors + threads

As usual it is important to perform a comprehensive calculation of
- Throughput/W/CHF

Expect more activity in this domain in the coming months/years

In openlab we will continue to participate actively in realistic and relevant evaluations