MASSIVELY PARALLEL DATA PROCESSING USING CUDA

Introduction to Concepts and Coding

Tony Rogerson, SQL Server MVP
@tonyrogerson | http://sqlblogcasts.com/blogs/tonyrogerson
http://www.linkedin.com/in/tonyrogerson
tonyrogerson@torver.net
Who am I?

• Started out as a Developer on the mainframe (1986) – PL/1, CICS, DB2, Application System, System W
• Moved into Client/Server first with Oracle then SQL Server 4.21a in 1993
• Freelance SQL Server consultant
• MSc Business Intelligence (University of Dundee)
• Fellow British Computer Society
• Interested in Data
  • NoSQL
  • BigData
  • Relational
  • Parallel Data Processing
DISCLAIMER
Agenda

• Why the Interest in using CUDA and GPGPU?
• Parallelism concepts
  • CPU, GPU cores
  • RBAR or Set processing
  • SISD, SIMD, SIMT, MIMD
• Throughput – Realities!
• CUDA (Fermi) Architecture
  • Key Components – Streaming Multiprocessor, GigaThread Engine, Warp, Cores
• Coding Concepts
  • Nvidia SDK and toolkit
  • CUDAfy.NET
• Summary and Futures
INTEREST IN PARALLEL DATA PROCESSING

Because it makes perfect sense…
Analytics from Commodity Kit

- SSD based storage – GiBytes per second on random read access now reality
- Dealing with BigData (volume and velocity) at a realistic price point
- CPU cores are expanding horizontally but not getting any quicker (well, no jumps) – physics!!
Disk V SSD comparison

6 x 15Krpm drives RAID 0

<table>
<thead>
<tr>
<th>Queue Depth</th>
<th>Worker Threads</th>
<th>MiB/sec</th>
<th>Latency Avg (ms)</th>
<th>CPU %</th>
<th>IO'ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td>276.43</td>
<td>0.45</td>
<td>14.27</td>
<td>35383</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>328.79</td>
<td>0.75</td>
<td>19.29</td>
<td>42085</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>338.12</td>
<td>1.47</td>
<td>28.79</td>
<td>43279</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>346.49</td>
<td>2.88</td>
<td>33.2</td>
<td>44351</td>
</tr>
</tbody>
</table>

Single OCZ IBIS (PCIe)

Sequential Workload

<table>
<thead>
<tr>
<th>Queue Depth</th>
<th>Worker Threads</th>
<th>MiB/sec</th>
<th>Latency Avg (ms)</th>
<th>CPU %</th>
<th>IO'ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td>407.94</td>
<td>0.3</td>
<td>13.27</td>
<td>52216</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>600.79</td>
<td>0.41</td>
<td>28.17</td>
<td>76901</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>576.78</td>
<td>0.86</td>
<td>27.97</td>
<td>73828</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>560</td>
<td>1.78</td>
<td>30.65</td>
<td>71680</td>
</tr>
</tbody>
</table>

Random Workload

<table>
<thead>
<tr>
<th>Queue Depth</th>
<th>Worker Threads</th>
<th>MiB/sec</th>
<th>Latency Avg (ms)</th>
<th>CPU %</th>
<th>IO'ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td>16.16</td>
<td>7.72</td>
<td>4.06</td>
<td>2068</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>21.66</td>
<td>11.52</td>
<td>4.85</td>
<td>2772</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>26.91</td>
<td>18.49</td>
<td>6.82</td>
<td>3445</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>31.98</td>
<td>30.92</td>
<td>7.04</td>
<td>4093</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue Depth</th>
<th>Worker Threads</th>
<th>MiB/sec</th>
<th>Latency Avg (ms)</th>
<th>CPU %</th>
<th>IO'ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td>358.88</td>
<td>0.34</td>
<td>36.04</td>
<td>45937</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>427.75</td>
<td>0.58</td>
<td>40.44</td>
<td>54752</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>417.48</td>
<td>1.19</td>
<td>40.23</td>
<td>53437</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>424.45</td>
<td>2.35</td>
<td>41.55</td>
<td>54329</td>
</tr>
</tbody>
</table>

Till now (and SSD) we just can’t get data into memory quick enough
SSD Performance

• Enterprise is lagging far behind commodity SSD performance – far behind!
• Remove the IO bottleneck
• Introduce the CPU bottleneck
• Introduce the Memory bottleneck
THROUGHPUT REALITIES

How many GiBytes per second?
PCIe

- Measured in Lanes (x8, x16)
- PCIe 2.0 lane is approx 500MiBytes per second
- PCIe 3.0 lane is approx 1GiByte per second
- PCIe is more prevalent and replacing PCIe 2.0
- Host to Device transfer rate is GiBytes per second – still quicker than storage (even SSD)
- On device (the GPGPU) throughput is astonishing
Bandwidth (bandwidthtest.exe)

GeForce GTX 590
(2 GPU Devices in one)

PCle 2.0  x16

2 x 512 CUDA cores
328GiB/s Peak bandwidth

Device 0: GeForce GTX 590
Quick Mode

Host to Device Bandwidth, 1 Device(s), Paged memory Transfer Size (Bytes) Bandwidth (MB/s)
33554432 3895.2

Device to Host Bandwidth, 1 Device(s), Paged memory Transfer Size (Bytes) Bandwidth (MB/s)
33554432 4123.2

Device to Device Bandwidth, 1 Device(s) Transfer Size (Bytes) Bandwidth (MB/s)
33554432 139551.8
Host memory speed

- Not quite the speed of the GPU!
- Ok – DDR3 PC3-12800 is up to 12.8GiBytes/sec
CPU / GPU Parallelism

• CPU designed for:
  • Data Caching
  • Flow Control
  • Executing single thread as fast as possible
  • Has benefits like conditional prediction
  • Cores are a heavy piece of kit!

• GPU design for:
  • Graphics (Intensive Data-parallel computations)
  • Same Program (kernel) executed on many data items (in parallel)
  • Cores are light weight
Coding for Parallelism

• .NET
  • Data Parallel Task
  • Parallel LINQ
  • System.Threading

• SQL Server
  • Service Broker
  • Multiple Connections

• Other
  • Countless ways…..
Task Parallelism

- Multiple disconnected steps done asynchronously
  - Different queries executing on different threads
  - Code branches that can operate independently
Data Parallelism

- Same instruction operating on same data structure
- Single Instruction Multiple Data (in parallel, in sequence – no code path divergence)
- Single Instruction Multiple Thread (in parallel, can have code divergence but causes issues – discussed in threading later on)
- CUDA is SIMT
Data Partitioning (Streaming)

- Partition data into (equal) streams
- Unbalanced loading
- Choose partition column carefully (row-id or from data?)
DEMO: SQL Server Parallelism
Gather Synch Issues

- Partition Streams finish at different times
- Gather step waits until all streams are complete
- Core affected by other processes switched into work (context switching overhead)
- Skew on Partition Stream extends overall time to process
- Imagine sync issues on 1024 or even 3,000 core-threads!
Kernel Code branches

- Single Kernel works group of CUDA cores – discussed later
- No Branch:
  - Memory access optimised (contiguous read)
- Branch
  - None optimised and performance suffers
- Issue sync (cuda call: __synchthreads())
CUDA ARCHITECTURE

Nvidia’s Fermi
Terminology overview

• CUDA core
  *lightweight core that executes a thread*

• Kernel
  *the piece of code that runs*

• L1 – L3 -> DRAM
  *Multi-level cache, 1 is closet to processor and has lowest latency, DRAM has highest latency*

• SP – Streaming Processor (CUDA core)

• SM – Streaming Multiprocessor (Group of CUDA cores)
Architecture of a CPU

3rd Generation Intel® Core™ Processor: 22nm Process

- Processor Graphics
- Core
- Core
- Core
- Core
- Shared L3 Cache**
- Memory Controller I/O
- System Agent & Memory Controller Including DMI, Display and Misc. I/O

New architecture with shared cache delivering more performance and energy efficiency

Quad Core die with Intel® HD Graphics 4000 shown above
- Transistor count: 1.4Billion
- Die size: 160mm²

** Cache is shared across all 4 cores and processor graphics
Architecture of CUDA (Fermi)

- 512 CUDA Cores per GPU (1536 Kepler)
- GeForce GTX 590 (dual GPU on same card – 1024 CUDA cores), GTX 690 – 3072 CUDA cores
- SP’s in a SM vary – 32/48 (Fermi), 192 (Kepler)
- Note: General principles remain same (Fermi-Kepler)

http://www.nvidia.co.uk/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf
Streaming Multiprocessor

- Streaming Processor (core) isn’t as “heavy” as a CPU core – context switching done elsewhere
- Kernel (your code) runs on a group of cores (Warp or Half Warp)
- 16 SM’s on a Fermi GPU, depends on GPU implementation, compute factor
- 2 GPU’s in the GTX 590
Threading

- CPU threads
  - Heavy weight – context switching
  - Low level of concurrent threads
- GPGPU threads
  - Light weight
Hierarchy of Thread Groups

- **Grid** has one or more **Blocks**
- A **Block** runs on a **Streaming Multiprocessor** on the **GPU**
- **Block** split up into one or more **Warps**

[Diagram of thread groups and their hierarchy]
Warps

- Group of 32 parallel threads (half, quarter)
- Run the same Kernel
- Start at same program address (Single Instruction Multiple Thread (SIMT))
- Differs from SIMD because threads are independent within the Warp – conditional branching.
- Remember the Gather problem!
GigaThread  Thread Scheduler

- Schedules “thread blocks” to various “streaming multiprocessors”.
Host based Memory

- NUMA (memory bank per socket)
  - Latency increases on accessing remote processor
- Processor cache (LLC1 – LLC3)
- Speed
  - DDR3 – 6.4GiB/s – 24GiB/s
  - DDR4 – Too new, not widely adopted yet
Memory Model

- Memory is GDDR5 (192.4GiBytes per second)
- Memory has different latency and speed – Each SM has
  - Local/Shared Memory (64KiB configured 48/16 between shared memory and Local L1 cache)
  - Global Memory
CODING CONCEPTS
Heterogeneous System Architecture

- Hybrid approach using CPU and GPU
- Use the correct processor for the correct task
- Serial work loads – CPU
- Data Parallel work loads – GPU
- Future – will be done automatically for us
- Now – we do the coding and tell it where to run
Design Cycle: APOD

• Analyse
  • Look at parts of app of highest Execution Time
  • Check for suitability to run in parallel (any workflow dependency chains to prevent it?)

• Parallelise
  • Review code to see if and how the code can be GPU-optimised
  • Simply add preprocessor directives?
  • Re-code?
  • Library available to replicate?

• Optimise
  • Review steps and methods taken to parallelise – fine tune/rework

• Deploy

Re: CUDA C Best Practices Guide
Compilation Workflow

- Host and Device code separated
  - Device Code into PTX (Parallel Thread Execution)
- Just In Time compilation
- Nvidia Device Driver does Compilation
- Compilation is cached so first invocation is slow
Windows Services and CUDA

• Doesn’t work Windows 7+ (services run in Windows Session 0)

• Drivers need a user context to run (logged on session)

• TCC (Tesla Computer Cluster) driver – only Tesla series cards Tesla K, M, C – not GeForce!
  http://www.nvidia.co.uk/page/software-for-tesla-products.html

• … so, no windows service then!
CUDA Toolkit

- When using Visual Studio – use 2010, doesn’t work in 2012 yet
- Adjust PATH so that the Nvidia Compiler is in the path
  - `C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 4.2\C\bin\win64\Release;`
  - `C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin;`
- Nsight for Debugging
- White papers, documentation galore!
- Samples galore (NVIDIA CUDA C SDK Code Samples)
- CUDA.NET seems defunct
- CUDAfy.NET is alive and kicking
DEMO

- RadixSort (SortOnGPUExample)
  - Note: can I get this to work in 64 bit can I hell!
- Nsight (MatrixMulDrv_vs2010)
- CUDAfy.NET (CudafyByExample)
SUMMARY AND FUTURES

Unto where do we lead?
Good Reads

Very big subject!

• OpenGL is AMD’s equiv
• CUDA is suppose to be compatible
• Two good books:
  • Programming Massively Parallel Processors (David B. Kirk, Wen-mei W. Hwu)
  • Rob Farber CUDA Application Design and Development
Thanks for listening!

• PUB!