Nvidia Parallel Nsight & Visual Profiler

Seminar Software Profiling
Johannes Henning
What is CUDA

• Library/Runtime environment for GPGPU Programming
• NVidias alternative to OpenCL
• Provides constructs to formulate data-parallel algorithms
  • Which can efficiently be run on a GPU
• Basic idea:
  • One kernel (function) is run massively in parallel (multiple thousands)
  • Each executing kernel is a lightweight thread
  • Threads have IDs on which they depend for data partitioning
GPU Architecture

- **Block**
- **Warp**
- **Thread**

Software Profiling: Nvidia Nsight & Visual Profiler

Johannes Henning

25.06.13
Why Nsight?

• Common Problems with programming CUDA include:
  • Concurrency problems
  • Memory locality
  • Suboptimal utilization

• Normal debugging/optimizing workflow:
  • Modify kernel
  • Launch kernel and measure the execution time

• From debuggers point of view the kernel is a black box
  • Kernel returns or crashes
  • Far from optimal; Need for better tool support

So*ware Profiling: Nvidia Nsight & Visual Profiler

Johannes Henning
What is Nsight?

• IDE for debugging, profiling and optimizing CUDA Applications
• Integrated into Visual Studio or Eclipse
• Introduced in 2009
• Originally named „Nexus“
• Meanwhile on version 3.0
Nsight in Visual Studio

- NVidia Maximus debugging
- CUDA debugging
- OpenGL and DirectX debugging
- Profiling/Tracing
CUDA Debugging

CUDA Kernel

HPI Hasso
CUDA Debugging

```c

// global void

template <int BLOCK_SIZE> __global__ void

matrixMulCUDA(float *C, float *A, float *B, int wa, int wb)

{

    // Block index
    int bx = blockIdx.x;
    int by = blockIdx.y;

    // Thread index
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Index of the first sub-matrix of A processed by the block
    int abegin = wa * BLOCK_SIZE * by;

When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.
```

### CUDA Info

<table>
<thead>
<tr>
<th>Current</th>
<th>CUContext</th>
<th>Grid ID</th>
<th>blockIdx</th>
<th>Warp Index</th>
<th>threadIdx</th>
<th>Status</th>
<th>Source Line</th>
<th>Lanes</th>
<th>Freeze State</th>
<th>At Barrier</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x025766</td>
<td>2</td>
<td>( 0, 0, 0)</td>
<td>0</td>
<td>( 0, 0, 0)</td>
<td>Breakpoint</td>
<td>54</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0x025766</td>
<td>2</td>
<td>( 0, 0, 0)</td>
<td>1</td>
<td>( 0, 2, 0)</td>
<td>Barrier</td>
<td>94</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0x025766</td>
<td>2</td>
<td>( 0, 0, 0)</td>
<td>2</td>
<td>( 0, 4, 0)</td>
<td>Barrier</td>
<td>94</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0x025766</td>
<td>2</td>
<td>( 0, 0, 0)</td>
<td>3</td>
<td>( 0, 5, 0)</td>
<td>Barrier</td>
<td>94</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0x025766</td>
<td>2</td>
<td>( 0, 0, 0)</td>
<td>4</td>
<td>( 0, 6, 0)</td>
<td>Barrier</td>
<td>94</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0x025766</td>
<td>2</td>
<td>( 0, 0, 0)</td>
<td>5</td>
<td>( 0, 10, 0)</td>
<td>Barrier</td>
<td>94</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0x025766</td>
<td>2</td>
<td>( 0, 0, 0)</td>
<td>6</td>
<td>( 0, 12, 0)</td>
<td>Barrier</td>
<td>94</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0x025766</td>
<td>2</td>
<td>( 0, 0, 0)</td>
<td>7</td>
<td>( 0, 14, 0)</td>
<td>Barrier</td>
<td>94</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
```
**CUDA Debugging**

```cpp
// Example CUDA code snippet for debugging

template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *A, float *B, int WA, int WB)
{
    // Block index
    int bx = blockIdx.x;
    int by = blockIdx.y;

    // Thread index
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Index of the first sub-matrix of A processed by the block
    if (bx == 0 && tx == 15 && ty == 0) {
        int scope[100];
        int aBegin = WA * BLOCK_SIZE * by;
    }
}
```

---

### CUDA WarpWatch 1

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>1</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>2</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>3</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>4</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>5</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>6</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>7</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>8</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>9</td>
<td>Could not resolve name 'scope'.</td>
</tr>
<tr>
<td>10</td>
<td>Could not resolve name 'scope'.</td>
</tr>
</tbody>
</table>

### CUDA Info 1

<table>
<thead>
<tr>
<th>CUcontext</th>
<th>Grid ID</th>
<th>blockIdx</th>
<th>Warp Index</th>
<th>threadIdx</th>
<th>Status</th>
<th>Source Line</th>
<th>Lanes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(0, 0, 0)</td>
<td>0</td>
<td>(0, 0, 0)</td>
<td>Breakpoint</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(0, 0, 0)</td>
<td>1</td>
<td>(0, 2, 0)</td>
<td>Barrier</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(0, 0, 0)</td>
<td>2</td>
<td>(0, 4, 0)</td>
<td>Barrier</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(0, 0, 0)</td>
<td>3</td>
<td>(0, 6, 0)</td>
<td>Barrier</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(0, 0, 0)</td>
<td>4</td>
<td>(0, 8, 0)</td>
<td>Barrier</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(0, 0, 0)</td>
<td>5</td>
<td>(0, 10, 0)</td>
<td>Barrier</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(0, 0, 0)</td>
<td>6</td>
<td>(0, 12, 0)</td>
<td>Barrier</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(0, 0, 0)</td>
<td>7</td>
<td>(0, 14, 0)</td>
<td>Barrier</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(1, 0, 0)</td>
<td>0</td>
<td>(0, 0, 0)</td>
<td>None</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>0x0002F6</td>
<td>2</td>
<td>(1, 0, 0)</td>
<td>1</td>
<td>(0, 2, 0)</td>
<td>None</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

---

HPI Hasso Plattner Institut
Nsight in Visual Studio

- NVidia Maximus debugging
- CUDA debugging
- OpenGL and DirectX debugging
- Profiling/Tracing
Graphics Debugging
Performance Analysis

Software Profiling: Nvidia Nsight & Visual Profiler

Johannes Henning
Performance Analysis

Software Profiling: Nvidia Nsight & Visual Profiler

Johannes Henning

25.06.13
Performance Analysis

CUDA Summary

Top Device Functions By Total Time

- Session Summary
- Timeline
- Activity

[Data captured]

<table>
<thead>
<tr>
<th>Name</th>
<th>Launches</th>
<th>Device %</th>
<th>Total (us)</th>
<th>Min (us)</th>
<th>Avg (us)</th>
<th>Max (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>collideD</td>
<td>175</td>
<td>55.64</td>
<td>18,943,360</td>
<td>53,832,142</td>
<td>109,496,191</td>
<td>149,836,883</td>
</tr>
<tr>
<td>launch_closure_by_value&lt;for_each_num&gt;</td>
<td>174</td>
<td>0.81</td>
<td>276,461,538</td>
<td>1,454,078</td>
<td>1,586,974</td>
<td>1,852,029</td>
</tr>
<tr>
<td>reorder=DateAndFindCellStartID</td>
<td>173</td>
<td>0.21</td>
<td>71,723,822</td>
<td>368,927</td>
<td>414,569</td>
<td>429,696</td>
</tr>
<tr>
<td>ScanScatterDigits &lt;uint, uint, int&gt;</td>
<td>174</td>
<td>0.18</td>
<td>61,275,354</td>
<td>329,152</td>
<td>352,157</td>
<td>363,584</td>
</tr>
<tr>
<td>ScanScatterDigits &lt;uint, uint, int&gt;</td>
<td>174</td>
<td>0.16</td>
<td>56,046,927</td>
<td>312,800</td>
<td>322,120</td>
<td>330,496</td>
</tr>
<tr>
<td>ScanScatterDigits &lt;uint, uint, int&gt;</td>
<td>174</td>
<td>0.16</td>
<td>54,896,318</td>
<td>290,006</td>
<td>315,496</td>
<td>356,512</td>
</tr>
<tr>
<td>ScanScatterDigits &lt;uint, uint, int&gt;</td>
<td>174</td>
<td>0.16</td>
<td>53,525,404</td>
<td>303,986</td>
<td>309,341</td>
<td>313,728</td>
</tr>
<tr>
<td>ScanScatterDigits &lt;uint, uint, int&gt;</td>
<td>174</td>
<td>0.16</td>
<td>53,247,624</td>
<td>298,176</td>
<td>306,021</td>
<td>318,624</td>
</tr>
<tr>
<td>collHasID</td>
<td>174</td>
<td>0.14</td>
<td>48,932,335</td>
<td>299,848</td>
<td>277,722</td>
<td>294,304</td>
</tr>
<tr>
<td>SrsfScanSpline(�vo&lt;;)</td>
<td>1392</td>
<td>0.09</td>
<td>50,976,799</td>
<td>20,960</td>
<td>22,253</td>
<td>26,240</td>
</tr>
</tbody>
</table>

Software Profiling: Nvidia Nsight & Visual Profiler

Johannes Henning
Performance Analysis

25.06.13
Software Profiling: Nvidia Nsight & Visual Profiler
Johannes Henning
Performance Analysis

Software Profiling: Nvidia Nsight & Visual Profiler

Johannes Henning
• Lots of data available
• Lot of background knowledge needed to make use of it
• Finding performance problems can be cumbersome
• Profiling more lightweight, but same data sources
What is the Visual Profiler?

• Comes with the CUDA Toolkit
• Profiling support similar to Nsight
• Main differences to Nsight:
  • Cross platform, lightweight
  • Results are linked to CUDA best practices
  • Less filterable/customizable
• Think ktrace for CUDA
Visual Profiler

Data Transfer Between Host and Device

The peak theoretical bandwidth between the device memory and the GPU is much higher (177.6 GB/s on the NVIDIA Tesla M2090, for example) than the peak theoretical bandwidth between host memory and device memory (8 GB/s on the PCIe ×16 Gen2). Hence, for best overall application performance, it is important to minimize data transfer between the host and the device, even if that means running kernels on the GPU that do not demonstrate any speedup compared with running them on the host CPU.

Note: High Priority: Minimize data transfer between the host and the device, even if it means running some kernels on the device that do not show performance gains when compared with running them on the host CPU.

Intermediate data structures should be created in device memory, operated on by the device, and destroyed without ever being mapped by the host or copied to host memory.

Also, because of the overhead associated with each transfer, batching many small transfers into one larger transfer performs significantly better than making each transfer separately.

Finally, higher bandwidth between the host and the device is achieved when using page-locked (or pinned) memory, as discussed in the CUDA C Programming Guide and Pinned Memory of this document.

Pinned Memory

Page-locked or pinned memory transfers attain the highest bandwidth between the host and the device. On PCIe ×16 Gen2 cards, for example, pinned memory can attain greater than 5 GB/s transfer rates.

Pinned memory is allocated using the cuMemAlloc() functions in the Runtime API. The bandwidthTest.cu program in the NVIDIA GPU Computing SDK shows how to use these functions as well as how to measure memory transfer performance.

Pinned memory should not be overused. Excessive use can reduce overall system performance because pinned memory is a scarce resource. How much is too much is difficult to tell in advance, so as with all optimizations, test the applications and the systems they run on for optimal performance parameters.

Asynchronous and Overlapping Transfers with Computation
Summary

- Nsight: debugging and deep tracing
- Visual profiler: easy access and quick tips
What I did not talk about

- Graphics Debugging/Profiling: DirectX 10, DirectX 11, and OpenGL 4.2
- Nsight support for Dynamic Parallelism
- Nvprof: commandline profiler
- Memory Checking
- Freeze/Thaw
- CUDA Source View
- ...
Sources

- nvidia.com/nexus
- developer.nvidia.com/nsight
- developer.nvidia.com/nvidia-visual-profiler

</Talk>
CUDA Source View

Software Profiling: Nvidia Nsight & Visual Profiler

Johannes Henning