NEW DEVELOPER TOOLS FEATURES IN CUDA 8.0

Sanjiv Satoor
CUDA TOOLS
NVIDIA® NSIGHT™

Homogeneous application development for CPU+GPU compute platforms

CUDA-Aware Editor

CUDA Debugger

CUDA Profiler

CPU+GPU

eclipse

Visual Studio
CUDA TOOLS

VISUAL PROFILER
- Trace CUDA activities
- Profile CUDA kernels
- Correlate performance instrumentation with source code
- Expert-guided performance analysis

NVPROF
- Collect Performance events and metrics

GPU LIBRARY ADVISOR
- Detect CUDA library optimization opportunities

CUDA-MEMCHECK
- Detect out-of-bounds and misaligned memory accesses
- Detect race condition in memory accesses
- Detect uninitialized global memory accesses
- Detect incorrect GPU thread synchronization

CUDA-GDB
- Debug CUDA kernels with CLI
- Debug CPU and GPU code
- CPU and GPU core dump support

NVDISASM, CUOBJDUMP
NEW TOOLS FEATURES IN CUDA 8.0
SUPPORT FOR PASCAL ARCHITECTURE

ALL TOOLS

Support for GPUs with Compute Capability 6.0, 6.1 and 6.2
DEPENDENCY ANALYSIS

Visual Profiler, nvprof

Provide insight into application-level performance limiters
Expose dependencies between activities according to the programming model
Identify waiting time due to inter-stream dependencies
Highlight activities on the critical application runtime path
Supports CUDA (Linux/Mac/Windows) and POSIX threads (Linux/Mac)
DEPENDENCY ANALYSIS IN VISUAL PROFILER
UNIFIED MEMORY PROFILING

Requirements: Pascal (GP100) and Linux-64, Windows TCC in the future

Data transfers to/from GPU with accurate timestamps (event-based)

Supported counters:
CPU and GPU page faults, H2D data migration, D2H data migration

New activity record with timestamps, transfer size, virtual page base address, devices involved
UNIFIED MEMORY PROFILING

Visual profiler - 8.0 unified memory timeline
UNIFIED MEMORY PROFILING ANALYSIS

### Results

#### Unified Memory Analysis
The following table shows the top virtual addresses which have maximum data migration size:

<table>
<thead>
<tr>
<th>Virtual address</th>
<th>HtoD migration size</th>
<th>DtoH migration size</th>
<th>CPU page faults</th>
<th>GPU page faults</th>
<th>Migration throughput</th>
<th>Average faults per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x900709000</td>
<td>20.48 kB</td>
<td>1.012 MB</td>
<td>0</td>
<td>4</td>
<td>67.736 MB/s</td>
<td>37145.6631</td>
</tr>
<tr>
<td>0x900b08000</td>
<td>1.016 MB</td>
<td>4.096 kB</td>
<td>2</td>
<td>0</td>
<td>96.593 MB/s</td>
<td>0.00</td>
</tr>
<tr>
<td>0x900108000</td>
<td>1.016 MB</td>
<td>4.096 kB</td>
<td>2</td>
<td>1</td>
<td>78.726 MB/s</td>
<td>12180.5725</td>
</tr>
<tr>
<td>0x900907000</td>
<td>1.02 MB</td>
<td>0 B</td>
<td>6</td>
<td>6</td>
<td>81.543 MB/s</td>
<td>568367.9704</td>
</tr>
<tr>
<td>0x900309000</td>
<td>4.096 kB</td>
<td>1.012 MB</td>
<td>0</td>
<td>0</td>
<td>177.101 MB/s</td>
<td>0.00</td>
</tr>
<tr>
<td>0x902109000</td>
<td>4.096 kB</td>
<td>1.012 MB</td>
<td>0</td>
<td>4</td>
<td>215.42 MB/s</td>
<td>69839.1378</td>
</tr>
<tr>
<td>0x901909000</td>
<td>0 B</td>
<td>1.012 MB</td>
<td>0</td>
<td>26</td>
<td>5.444 GB/s</td>
<td>242854.0509</td>
</tr>
<tr>
<td>0x900909000</td>
<td>0 B</td>
<td>1.012 MB</td>
<td>0</td>
<td>0</td>
<td>5.251 GB/s</td>
<td>0.00</td>
</tr>
<tr>
<td>0x901b09000</td>
<td>0 B</td>
<td>1.012 MB</td>
<td>0</td>
<td>14</td>
<td>5.456 GB/s</td>
<td>68297.36</td>
</tr>
<tr>
<td>0x900b09000</td>
<td>0 B</td>
<td>1.012 MB</td>
<td>0</td>
<td>0</td>
<td>5.109 GB/s</td>
<td>0.00</td>
</tr>
</tbody>
</table>

The following table shows the summary of unified memory migrations and page faults for the application:

<table>
<thead>
<tr>
<th>Total HtoD migration size</th>
<th>Total DtoH migration size</th>
<th>Total CPU Page faults</th>
<th>Total GPU Page faults</th>
<th>Total different pages</th>
<th>The virtual address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>37.753 MB</td>
<td>33.559 MB</td>
<td>614</td>
<td>20868</td>
<td>1797</td>
<td>0x902c1d000-0x90087000</td>
</tr>
</tbody>
</table>
NVLink Visualization

Unguided Analysis

Visual Profiler

Option to collect NVLink information

Topology

Static properties

Runtime values

Achieved throughput

The following NVLink topology diagram shows logical NVLink connections between GPUs and CPU. The logical NVLink can contain one or more physical links. For a logical NVLink between two devices A and B, the average receive throughput for device A is shown close to device A and the average receive throughput for device B is shown close to device B. Receive throughput of device A is the same as transmit throughput of device B.

The tables on the right-hand side show the properties, bandwidth utilization and throughput of each logical NVLink.

<table>
<thead>
<tr>
<th>Logical NVLink Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU0&lt;-&gt;GPU1</td>
</tr>
<tr>
<td>GPU0&lt;-&gt;GPU2</td>
</tr>
<tr>
<td>GPU1&lt;-&gt;GPU0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logical NVLink Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU0&lt;-&gt;GPU1</td>
</tr>
<tr>
<td>GPU0&lt;-&gt;GPU2</td>
</tr>
<tr>
<td>GPU1&lt;-&gt;GPU0</td>
</tr>
<tr>
<td>GPU1&lt;-&gt;GPU2</td>
</tr>
<tr>
<td>GPU1&lt;-&gt;CPU0</td>
</tr>
</tbody>
</table>
**NVLINK VISUALIZATION**

Visual Profiler on DGX1

---

**NVLink Analysis**

The following NVLink topology diagram shows logical NVLink connections between GPUs and CPUs. A logical NVLink can contain one or more physical links. When two devices A and B are connected by an NVLink, the receive throughput of device A is same as the transmit throughput of device B. The tables on right hand side show the properties for each logical NVLink.

* NVLink utilization may vary in accuracy, because any activity within the sampling period is treated as active, even though most of that period could be idle.

---

### Logical NVLink Properties

<table>
<thead>
<tr>
<th>Logical NVLink</th>
<th>PeakBandwidth</th>
<th>PhysicalNVLinks</th>
<th>PeerAccess</th>
<th>SystemAccess</th>
<th>PeerAtomic</th>
<th>SystemAtomic</th>
<th>Utilization %</th>
<th>IdleTime %</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU0 → GPU1</td>
<td>40 GB/s</td>
<td>1</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>...</td>
<td>40 GB/s</td>
<td>1</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>0</td>
<td>51</td>
</tr>
</tbody>
</table>

### Logical NVLink Throughput

<table>
<thead>
<tr>
<th>Logical NVLink</th>
<th>Avg Throughput</th>
<th>Min Throughput</th>
<th>Max Throughput</th>
<th>Min Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU0 → GPU1</td>
<td>20.519 MB/s</td>
<td>16.925 GB/s</td>
<td>2.580 GB/s</td>
<td>2.580 GB/s</td>
</tr>
<tr>
<td>...</td>
<td>20.519 MB/s</td>
<td>16.925 GB/s</td>
<td>2.580 GB/s</td>
<td>2.580 GB/s</td>
</tr>
</tbody>
</table>

---

**NVIDIA**
CPU PROFILING
Visual Profiler

Profiles the process of interest
Collect CPU sampling data alongside the GPU data you are used to
Sample at a specific frequency
Function & library symbols listed
Analyze the data per-thread from different perspectives (top-down, bottom-up or flat)
# CPU PROFILING IN VISUAL PROFILER

<table>
<thead>
<tr>
<th>Event</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bench_staggeredleapfrog2_</code></td>
<td>95.833%</td>
<td>689.695 ms</td>
</tr>
<tr>
<td><code>CCTKiBindingsFortranWrapperBenchADM</code></td>
<td>95.833%</td>
<td>689.695 ms</td>
</tr>
<tr>
<td><code>CCTK_CallFunction</code></td>
<td>95.833%</td>
<td>689.695 ms</td>
</tr>
<tr>
<td><code>__open_nocancel</code></td>
<td>1.389%</td>
<td>9.996 ms</td>
</tr>
<tr>
<td><code>InitialFlat</code></td>
<td>1.389%</td>
<td>9.996 ms</td>
</tr>
<tr>
<td><code>__c_mcopy8</code></td>
<td>1.389%</td>
<td>9.996 ms</td>
</tr>
</tbody>
</table>

```c
!OMP PARALLEL DO
DO 100 J=1,N

Multiple markers at this line
- Generated 6 prefetch instructions for the loop
- Generated vector sse code for the loop
- Generated 5 alternate versions of the loop
- 2 loop-carried redundant expressions removed with 2 operations and 4 arrays
- intensity = 1.93

100 CONTINUE

PERIODIC CONTINUATION
```

![Code snippet from a Fortran program with NVIDIA logo in the corner](image_url)
nvprof:

- Summary of OpenACC activities with their inclusive and exclusive time
- Trace of activities along with CUDA API calls and GPU activities, including CUDA device/ctx/stream info, data transfer sizes, etc.
- Source file/line and kernel name correlation

Visual Profiler:

- Visualization of OpenACC activities, coloring for different types of activities
- Correlation with their compiler-generated API calls and device activities
- Source code view for mapped OpenACC directives
OPENACC PROFILING

OpenAcc->Driver API->Compute correlation

OpenAcc timeline

OpenAcc->Source Code correlation

OpenAcc Properties
New HW feature introduced with Pascal:
Long running kernels can be preempted
Processes using the GPU run without blocking each other (CUDA/Graphics)

Resulting debugger improvements:
Support for fast single GPU debugging
Lower overhead for attaching to running process
DEBUG ON DISPLAY GPU WITH NO RESTRICTIONS
MEMORY CHECKER

cuda-memcheck

Support for non-migratable system-scoped atomics checking on SM 6.x in Memcheck Tool.

Support for reporting fatal CPU-side faults when Unified Memory is enabled in Memcheck Tool.

Support for correctly determining the expected set of threads at a barrier in the presence of exited threads in Synccheck Tool.
NEW TOOLS FEATURES IN CUDA 8.0

PASCAL Architecture support
Profiler enhancements
  Dependency Analysis  CPU Profiling
  Unified Memory profiling  OpenACC profiling
  NVLINK analysis
Debug on display GPU with no restrictions
Memcheck enhancements
FOR MORE INFORMATION ...

CUDA 8 Features Revealed Parallel Forall Blog Post : https://devblogs.nvidia.com/parallelforall/cuda-8-features-revealed/

CUDA Documentation: http://docs.nvidia.com/cuda/

Download CUDA Toolkit 8: https://developer.nvidia.com/cuda-downloads
BACKUP SLIDES
AGENDA

CUDA Tools summary
New Tools Features in CUDA 8.0
CUDA TOOLS


Debug: cuda-gdb, CUDA Debug API

Memcheck: cuda-memcheck

Profile/Trace: CUDA Visual Profiler, nvprof, CUPTI
DEPENDENCY ANALYSIS EXAMPLE

Dependencies between events derived from programming model constraints

Allows to compute wait states and the critical path

cudaLaunch  \(\rightarrow\)  cudaStreamSynchronize

Stalls CPU (waiting time)

Kernel
DEPENDENCY ANALYSIS IN VISUAL PROFILER

New option in the Visual Profiler’s expert system and new modes to display the critical path data on the timeline. Exec dependencies for a selected interval:
UNIFIED VIRTUAL MEMORY

Custom Data Management

System Memory

GPU Memory

Developer View With Unified Memory

Unified Memory

Starting with Kepler and CUDA 6
UNIFIED MEMORY PROFILING

Visual profiler - 8.0 unified memory timeline (interval selection)
RUN GRAPHICS APPLICATIONS WHILE DEBUGGING CUDA APPLICATIONS
RUN MULTIPLE DEBUGGER SESSIONS
**DEPENDENCY ANALYSIS IN NVPROF**

New option to run post-mortem dependency analysis

New option to trace POSIX threads for multi-threaded applications

```plaintext
==22557== Dependency Analysis:
<table>
<thead>
<tr>
<th>Critical path(%)</th>
<th>Critical path</th>
<th>Waiting time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>94.61%</td>
<td>3.942181s</td>
<td>0ns</td>
<td>clock_block(long*, long)</td>
</tr>
<tr>
<td>5.20%</td>
<td>216.857718ms</td>
<td>0ns</td>
<td>cudaMalloc</td>
</tr>
<tr>
<td>0.16%</td>
<td>6.617667ms</td>
<td>0ns</td>
<td>&lt;Other&gt;</td>
</tr>
<tr>
<td>0.01%</td>
<td>293.028000us</td>
<td>0ns</td>
<td>cuDeviceGetAttribute</td>
</tr>
<tr>
<td>0.01%</td>
<td>235.154000us</td>
<td>0ns</td>
<td>cudaGetDeviceProperties</td>
</tr>
<tr>
<td>0.01%</td>
<td>221.116000us</td>
<td>0ns</td>
<td>cudaFree</td>
</tr>
<tr>
<td>0.00%</td>
<td>158.703000us</td>
<td>0ns</td>
<td>cudaStreamCreate</td>
</tr>
<tr>
<td>0.00%</td>
<td>35.252000us</td>
<td>0ns</td>
<td>cudaConfigureCall</td>
</tr>
<tr>
<td>0.00%</td>
<td>35.248000us</td>
<td>0ns</td>
<td>cuDeviceGetName</td>
</tr>
<tr>
<td>0.00%</td>
<td>33.139000us</td>
<td>0ns</td>
<td>cuDeviceTotalMem_v2</td>
</tr>
<tr>
<td>0.00%</td>
<td>20.298000us</td>
<td>0ns</td>
<td>cudaSetupArgument</td>
</tr>
<tr>
<td>0.00%</td>
<td>19.433000us</td>
<td>0ns</td>
<td>cudaGetDevice</td>
</tr>
<tr>
<td>0.00%</td>
<td>0ns 3.942147s</td>
<td>pthread_join</td>
<td></td>
</tr>
<tr>
<td>0.00%</td>
<td>0ns 3.942136s</td>
<td>cudaStreamSynchronize</td>
<td></td>
</tr>
<tr>
<td>0.00%</td>
<td>0ns 1.001459s</td>
<td>pthread_mutex_lock</td>
<td></td>
</tr>
<tr>
<td>0.00%</td>
<td>0ns 980.464357ms</td>
<td>pthread_cond_wait</td>
<td></td>
</tr>
</tbody>
</table>
```
NVLINK ANALYSIS
Visual Profiler

Memcpy P2P
NVLINK CUPTI SUPPORT

New record for Nvlink topology

Information about connected devices and connected ports

Version, peak BW, properties (e.g. peer/sysmem accesses, atomic accesses, ...)

Performance given using metrics: data transmitted/throughput, overhead

Metrics available split according to type of operation (atom/reduction/write/response) as well as accumulated
SOURCE-DISASSEMBLY VIEW ENHANCEMENTS

Visual Profiler

Single integrated view for the different source level analysis results

The Source-Disassembly view contains:

- High level source
- Assembly instructions
- Hotspots at the source level
- Hotspots at the assembly instruction level
- Columns for profiling data aggregated to the source level
- Columns for profiling data collected at the assembly instruction level
SOURCE-DISASSEMBLY VIEW ENHANCEMENTS
MISC IMPROVEMENTS

Profilers/CUPTI

Added support to establish correlation between an external API (such as OpenACC, OpenMP) and CUPTI API activity records

New activity records for CUDA synchronization constructs (ctx sync, CUDA event, ...)

UVM-lite counter profiling support on Mac

Support for FP16 data

Added containers to store the information of events and metrics in the form of activity records