UPDATES FOR CUDA 9.2

**NVPROF**

Many New Metrics:
- Tensor Core Metrics
- L2 Metrics
- Memory Instructions Per Load/Store

Display PCIe Topology

View Trace and Profile in combined output (-t trace)

**VISUAL PROFILER**

Summary View for Memory Hierarchy

Improved Handling of Segments for UVM Data on the Timeline
UPDATES FOR CUDA 10.0

Added tracing support for Turing

New kernel profiler - Nsight Compute (supports Turing)

OpenMP profiling

Tracing support for CUDA kernels, memcpy and memset nodes launched by a CUDA Graph

Support for version 3 NVIDIA Tools Extension API (NVTX) (This is a header-only implementation)
CUDA VISUAL PROFILER

Overview of key features

Kernel profile - memory hierarchy view
Unified Memory
NVLink
PC sampling
OpenACC/OpenMP Profiling
NVTX
NVIDIA’S VISUAL PROFILER (NVVP)

Guided System

1. CUDA Application Analysis
2. Performance-Critical Kernels
3. Compute, Bandwidth, or Latency Bound
   - The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "Step10_cudac_kernel" is most likely limited by compute.
   - If you modify the kernel you need to re-run your application to update this analysis.

Timeline

Analysis

Stall Reasons
- execute dependency
- instruction fetch
- data request
- synchronization
- other
DATA MOVEMENT IN VISUAL PROFILER
UVM IN VISUAL PROFILER
KERNEL PROFILE

Memory hierarchy view

- Texture: 0.00 Req, 0.00 B
- Local: 138.34 kReq, 9.77 KB, 9.64 KB
- Global: 53.59 kReq, 5.20 KB, 1.95 KB
- Shared: 47.14 kReq, 4.35 KB, 7.13 KB
- Unified Cache: 2.61 MB, 4.77 MB
- L2 Cache: 10.00 B, 22.00 B, 4.41 MB, 3.19 MB
VISUAL PROFILER
Segment mode timeline

Segment mode interval  Heat map for CPU page faults
VISUAL PROFILER
Switch to non-segment view

- Uncheck
- Select settings view
- Select this tab
- Load data within a specific time range
VISUAL PROFILER
Non-segmented mode timeline
VISUAL PROFILER

CPU Page Fault Source Correlation

Selected interval

Source location
VISUAL PROFILER
CPU Page Fault Source Correlation

Source line causing CPU page fault

```cpp
float *a;
float *a_new;
float *weights;

CUDA_CALL(cudaMallocManaged(a, nx*ny* sizeof(float)));
CUDA_CALL(cudaMallocManaged(a_new, nx*ny* sizeof(float)));
CUDA_CALL(cudaMallocManaged(6*weights, a_weights* sizeof(float)));

init(a, a_new, nx, ny, weights, n_weights);
cudaEvent_t start, stop;
CUDA_CALL(cudaEventCreate(&start));
CUDA_CALL(cudaEventCreate(&stop));
CUDA_CALL(cudaEventSynchronize());
CUDA_CALL(cudaEventRecord(start));
PUSH RANGE("while loop",0)
int iter = 0;
while (iter <= iter_max)
{
PUSH RANGE("Jacobi step",1)
Jacobi_iteration<<<dim3(nx/32,ny/4),dim3(32,4)>>>(a_new, a, nx, ny, weights);
CUDA_CALL(cudaGetLastError());
CUDA_CALL(cudaDeviceSynchronize());
POP RANGE
std::swap(a, a_new);
PUSH RANGE("periodic boundary conditions",2)
// Apply periodic boundary conditions
for (int ix = 0; ix < nx; ++ix)
{
    a[*mx+ix] = a[(ny-2)*nx+ix];
a[(ny-1)*nx+ix] = a[ix];
}
POP RANGE
if ( B == iter%100)
{
    std::cout << iter << std::endl;
    iter++;
}
CUDA_CALL(cudaEventRecord(stop));
CUDA_CALL(cudaDeviceSynchronize());
POP RANGE
```
VISUAL PROFILER - NEW UNIFIED MEMORY EVENTS

Page throttling, Memory thrashing, Remote map
VISUAL PROFILER
Filter and Analyze

Filtered intervals
VISUAL PROFILER

NVLINK visualization

Unguided Analysis

Option to collect NVLink information

Version

Color codes for NVLink

Topology

Selected NVLink

Static properties

Runtime values

Unguided Analysis

Option to collect NVLink information

Version

Color codes for NVLink

Topology

Selected NVLink

Static properties

Runtime values
VISUAL PROFILER

NVLink events on timeline

NVLink Events on Timeline

MemCpy API

Color Coding of NVLink Events
VISUAL PROFILER
Multi-hop remote profiling - Application Profiling

1. Select custom script, then create a remote session as usual

2. Application transparently runs on compute node and profiling data is displayed in the Visual Profiler
CPU SAMPLING

- CPU profile is gathered by periodically sampling the state of each thread in the running application.
- The CPU details view summarizes the samples collected into a call-tree, listing the number of samples (or amount of time) that was recorded in each function.
## VISUAL PROFILER

### CPU Sampling

<table>
<thead>
<tr>
<th>Event</th>
<th>Percentage</th>
<th>Range of Time Spent</th>
<th>Selected Thread Highlighted in Orange</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mp_barrier_tw</td>
<td>32.578%</td>
<td>389.9 ms - 1.2 s</td>
<td>Orange</td>
</tr>
<tr>
<td>x_solve</td>
<td>11.513%</td>
<td>210 ms - 439.9 ms</td>
<td></td>
</tr>
<tr>
<td>compute_rhs</td>
<td>9.063%</td>
<td>210 ms - 359.9 ms</td>
<td></td>
</tr>
<tr>
<td>z_solve</td>
<td>4.776%</td>
<td>90 ms - 210 ms</td>
<td></td>
</tr>
<tr>
<td>y_solve</td>
<td>3.735%</td>
<td>100 ms - 110 ms</td>
<td></td>
</tr>
<tr>
<td>tzeet</td>
<td>0.531%</td>
<td>10 ms - 20 ms</td>
<td></td>
</tr>
</tbody>
</table>
PC SAMPLING

PC sampling feature is available for device with CC >= 5.2

Provides CPU PC sampling parity + additional information for warp states/stalls reasons for GPU kernels

Effective in optimizing large kernels, pinpoints performance bottlenecks at specific lines in source code or assembly instructions

Samples warp states periodically in round robin order over all active warps

No overheads in kernel runtime, CPU overheads to parse the records
VISUAL PROFILER - PC SAMPLING

Option to select sampling period

The actual sampling period will be in 2^n cycles
VISUAL PROFILER

PC SAMPLING UI

Pie chart for sample distribution for a CUDA function

Sample distribution
MULTI-PROCESS PROFILING

When running nvprof with multiple processes, it’s useful to label each process:

```bash
$ nvprof --o timeline_rank%q{OMPI_COMM_WORLD_RANK} \\
    --context-name "MPI Rank %q{OMPI_COMM_WORLD_RANK} \\
    --process-name "MPI Rank %q{OMPI_COMM_WORLD_RANK} \\
    --annotate-mpi openmpi ...
```
MPI PROFILING
Importing into the Visual Profiler
MPI PROFILING

Visual Profiler

Real applications frequently produce too much data to manage.

Profiling can be programmatically toggled:

```c
#include <cuda_profiler_api.h>

cudaProfilerStart();

...

cudaProfilerStop();
```

This can be paired with nvprof:

```
$ nvprof --profile-from-start off ...
```
SELECTIVE PROFILING

When the profiler API still isn’t enough, selectively profile kernels, particularly with performance counters.

$ nvprof --kernels :::::1 --analysis-metrics ...

context:stream:kernel:invocation

Record metrics for only the first invocation of each kernel.
The NVIDIA Tools Extensions (NVTX) allow you to annotate the profile:

```c
#include <nvToolsExt.h> // Link with -lnvToolsExt

nvtxRangePushA("timestep");

timestep();

nvtxRangePop();
```

See https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvtx for more features, including V3 usage.
NVTX IN VISUAL PROFILER

Named Range
EXPORTING DATA

It’s often useful to post-process nvprof data using your favorite tool (Python, Excel, ...):

```bash
$ nvprof --csv --log-file output.csv \  
    -i profile.nvprof
```

It’s often necessary to massage this file before loading into your favorite tool.
OPENACC PROFILING

- OpenAcc->Driver API->Compute correlation
- OpenAcc timeline
- OpenAcc->Source Code correlation
- Properties
OPENMP PROFILING

Information about OpenMP regions using the OpenMP tools interface (OMPT) starting CUDA 10.0

Supported on x86_64 and Power Linux with PGI runtime 18.1+

Supported added in the CUPTI, nvprof and Visual Profiler
OPENMP PROFILING IN NVPROF

nvprof option openmp-profiling to enable/disable the OpenMP profiling, default on

$nvprof openmp-profiling on ./omp-app

<table>
<thead>
<tr>
<th>Type</th>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP (incl):</td>
<td>99.97%</td>
<td>277.10ms</td>
<td>20</td>
<td>13.855ms</td>
<td>13.131ms</td>
<td>18.151ms</td>
<td>omp_parallel</td>
</tr>
<tr>
<td>0.03%</td>
<td>72.728us</td>
<td></td>
<td>19</td>
<td>3.8270us</td>
<td>2.9840us</td>
<td>9.5610us</td>
<td>omp_idle</td>
</tr>
<tr>
<td>0.00%</td>
<td>7.9170us</td>
<td></td>
<td>7</td>
<td>1.1310us</td>
<td>1.0360us</td>
<td>1.5330us</td>
<td>omp_wait_barrier</td>
</tr>
</tbody>
</table>

Option --print-openmp-summary to print a summary of all recorded OpenMP activities
OPENMP PROFILING IN VISUAL PROFILER
# OPENMP PROFILING IN VISUAL PROFILER

## Table View

<table>
<thead>
<tr>
<th>Name</th>
<th>%</th>
<th>Time</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP_Parallel</td>
<td>93.895%</td>
<td>5.4 s</td>
<td>3003</td>
</tr>
<tr>
<td>OMP_Idle</td>
<td>16.619%</td>
<td>0.9 s</td>
<td>3002</td>
</tr>
<tr>
<td>OMP_Wait_barrier</td>
<td>7.528%</td>
<td>0.4 s</td>
<td>3001</td>
</tr>
</tbody>
</table>
PROFILING NVLINK USAGE

Using nvprof+NVVP

Run nvprof multiple times to collect metrics

```bash
jsrun <args> nvprof --output-profile profile.<metric>.%q{OMPI_COMM_WORLD_RANK}\n   --aggregate-mode off --event-collection-mode continuous \n   --metrics <metric> -f
```

Use `--query-metrics` and `--query-events` for full list of metrics (-m) or events (-e)

Combine with an MPI annotated timeline file for full picture
SUMMIT NVLINK TOPOLOGY
CPU PAGE FAULT SOURCE CORRELATION

- Unguided Analysis
- Summary of all CPU page faults
- Option to collect Unified Memory information