NVIDIA DEVELOPER TOOLS: NEW CAPABILITIES IN CUDA 9.X

Jeff Larkin, CAAR Workshop March 2018
CUDA TOOLS
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

NVDISASM, CUOBJDUMP

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
NEW TOOLS FEATURES IN CUDA 9
SUPPORT FOR VOLTA ARCHITECTURE

Support for GPUs with Compute Capability 7.0
CUDA-GDB
New features post CUDA 9

- GPU core dump generation is supported on Volta-MPS
- Reading lightweight GPU core dump files is supported

$ time CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1 CUDA_ENABLE_LIGHTWEIGHT_COREDUMP=1 ./simple_cuda_program

Before: 9.85s user 12.33s system 173% cpu 12.794 total

After: 0.41s user 1.36s system 75% cpu 2.350 total
CUDA-MEMCHECK

- Support for host API functions with pitch parameter.
- *Initial support for the Cooperative Groups programming model.*
- Support for shared memory atomic instructions.
- *Support for detecting invalid accesses to global memory on Pascal and later architectures that extend beyond the end of an allocation.*
- Support for limiting the numbers of errors printed by cuda-memcheck.
- Racecheck analysis reports are assigned a severity level.
- Default print level changed from INFO to WARN.
- *A new command line option to report deprecated instructions even when they are used in safe execution paths. (post CUDA 9)*
CUDA VISUAL PROFILER

Enhancements in CUDA 9.0

Unified Memory

NVLink

Multi-hop remote profiling

Tracing and profiling of Cooperative Kernel launches

PC sampling
VISUAL PROFILER
Segment mode timeline

Segment mode interval
Heat map for CPU page faults
VISUAL PROFILER

CPU Page Fault Source Correlation

Selected interval

Source location
**CPU PAGE FAULT SOURCE CORRELATION**

**Unguided Analysis**

- Option to collect Unified Memory information

**Summary of all CPU page faults**

<table>
<thead>
<tr>
<th>CPU page faults</th>
<th>Source location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td><a href="mailto:main@jacobi.cs">main@jacobi.cs</a>:130</td>
</tr>
<tr>
<td>1001</td>
<td><a href="mailto:main@jacobi.cs">main@jacobi.cs</a>:130</td>
</tr>
<tr>
<td>4</td>
<td>Unknown</td>
</tr>
<tr>
<td>2</td>
<td>Unknown</td>
</tr>
<tr>
<td>1</td>
<td><a href="mailto:_Z4initPFS_iis_1@jacobi.cs">_Z4initPFS_iis_1@jacobi.cs</a>:85</td>
</tr>
<tr>
<td>1</td>
<td>Unknown</td>
</tr>
</tbody>
</table>
VISUAL PROFILER

CPU Page Fault Source Correlation

Source line causing CPU page fault:
```
float * A;
float * a_new;
float * weights;

cudaCall(cudamallocManaged(64u, nx*ny*sizeof(float)));
cudaCall(cudamallocManaged(64u, nx*ny*sizeof(float)));
cudaCall(cudamallocManaged(64u, nx*ny*sizeof(float)));

init(a, a_new, nx, ny, weights, n_weights);
cudaEvent_t start, stop;
cudaCall(cudaeventCreate(64start));
cudaCall(cudaeventCreate(64stop));
cudaCall(cudadevicesynchronize());
cudaCall(cudaeventRecord(start));

while (iter <= iter_max) {
    int iter = 0;
    for (int ix = 0; ix < nx; ++ix) {
        for (int iy = 0; iy < ny; ++iy) {
            a[0*nx+ix] = a[(ny-2)*nx+ix];
            a[(ny-1)*nx+ix] = a[1*nx+ix];
        }
    }
    cudaCall(cudadevicesynchronize());
    cudaCall(cudaeventRecord(stop));
    cudaCall(cudadevicesynchronize());
    cudaCall(cudaeventRecord(stop));
}
```
VISUAL PROFILER - NEW UNIFIED MEMORY EVENTS

Page throttling, Memory thrashing, Remote map
VISUAL PROFILER

Filter and Analyze

Filtered intervals
int threadsPerBlock = 256;
int numBlocks = (length + threadsPerBlock - 1) / threadsPerBlock;

kernel<<<numBlocks, threadsPerBlock>>>(A, B, C, length);

int threadsPerBlock = 256;
int numBlocks = (length + threadsPerBlock - 1) / threadsPerBlock;

cudaMemAdvise(A, size, cudaMemAdviseSetReadMostly, 0);
cudaMemAdvise(B, size, cudaMemAdviseSetReadMostly, 0);

kernel<<<numBlocks, threadsPerBlock>>>(A, B, C, length);
OPTIMIZED APPLICATION

NoDtoH Migrations and thrashing

Speedup 4x (2.9 vs 12.2)
VISUAL PROFILER

NVLINK visualization

Unguided Analysis

Option to collect NVLink information

Version

Color codes for NVLink

Topology

Selected NVLink

Static properties

Runtime values

Color codes for NVLink

Version

Topology

Selected NVLink

Static properties

Runtime values

Unguided Analysis

Option to collect NVLink information

Version

Topology

Selected NVLink

Static properties

Runtime values
VISUAL PROFILER

NVLink events on timeline - Segment mode
VISUAL PROFILER

Multi-hop remote profiling

Script available on github: https://github.com/NVIDIA/cuda-profiler/tree/master/one_hop_profiling
VISUAL PROFILER
Multi-hop remote profiling - One-Time Setup

1. Configure script on the login node
2. Connect Visual Profiler to the login node
3. Use the custom script option
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
VISUAL PROFILER
CPU Sampling (post CUDA 9)

Percentage of time spent collectively by all threads

Range of time spent across all threads

Selected thread is highlighted in Orange

Bar chart of the amount of time spent by thread
VISUAL PROFILER - PC SAMPLING
Option to select sampling period (post CUDA 9)
VISUAL PROFILER

PC SAMPLING UI

Pie chart for sample distribution for a CUDA function

Sample distribution

Source-Assembly view
MPI PROFILING
MPI PROFILING

nvprof

$ LD_PRELOAD="libnvtx_pmpi.so" mpirun -n 4 nvprof --process-name "MPI Rank %q{PMIX_RANK}" --context-name "MPI Rank %q{PMIX_RANK}" -o timeline.%q{PMIX_RANK}.nvprof ./simpleMPI

Running on 4 nodes

==21977== NVPROF is profiling process 21977, command: ./simpleMPI
==21983== NVPROF is profiling process 21983, command: ./simpleMPI
==21979== NVPROF is profiling process 21979, command: ./simpleMPI
==21982== NVPROF is profiling process 21982, command: ./simpleMPI

<program output>

==21982== Generated result file: timeline.0.nvprof
==21977== Generated result file: timeline.3.nvprof
==21983== Generated result file: timeline.1.nvprof
==21979== Generated result file: timeline.2.nvprof
MPI PROFILING
Importing into the Visual Profiler

1. Open the visual profiler.
2. Select "Import...".
3. Choose "Multiple processes".
4. Browse and select the nvprof profile files.

Profile Files:
- /home/apoorvaj/sw/gpupi/bin/x86_64_LINUX Debug/timeline.1.pdm
- /home/apoorvaj/sw/gpupi/bin/x86_64_LINUX Debug/timeline.2.pdm
- /home/apoorvaj/sw/gpupi/bin/x86_64_LINUX Debug/timeline.3.pdm
MPI PROFILING
Visual Profiler

MPI PROFILING

MPI + NVTX

Manual mode

nvtxEventAttributes_t range = {0};
range.message.ascii = "MPI_Scatter";
nvtxRangePushEx(range);
int result = MPI_Scatter(...);
nvtxRangePop();

Interception mode

1. Auto-generate mpi_interception.so

2. LD_PRELOAD=mpi_interception.so

3. Run your MPI app with nvprof.
   MPI calls will be auto-annotated using NVTX.
MPI PROFILING

Interception

MPI app

```c
int res = MPI_Scatter(...);
```

Interception library (LD_PRELOAD)

```c
int MPI_Scatter(...) {
    nvtxRangePushEx(range);
    int res = PMPI_Scatter(...);
    nvtxRangePop();
    return res;
}
```

MPI library

```c
int PMPI_Scatter(...)
```
FOR MORE INFORMATION …

CUDA 9 Features Revealed Parallel Forall Blog Post :  

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

BACKUP SLIDES
DGX-1V NVLINK TOPOLOGY
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
FILTER AND ANALYZE

1. Select unified memory in the unguided analysis section

2. Select required events and click on ‘Filter and Analyze’

Summary of filtered intervals
FILTER AND ANALYZE

Unfiltered
# CPU PAGE FAULT SOURCE CORRELATION

**Unguided Analysis**

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

---

**Table: CPU page faults and source locations**

<table>
<thead>
<tr>
<th>CPU page faults</th>
<th>Source location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>main@jacobi:cu:130</td>
</tr>
<tr>
<td>1001</td>
<td>main@jacobi:cu:130</td>
</tr>
<tr>
<td>4</td>
<td>Unknown</td>
</tr>
<tr>
<td>2</td>
<td>Unknown</td>
</tr>
<tr>
<td>1</td>
<td>_Z4initPFS_iIS_i@jacobi:cu85</td>
</tr>
<tr>
<td>1</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

(Double-click to open the location in source code)
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.