Applied Parallel Computing

parallel-computing.pro

Profiling

Pavel Yakimov

© APC

Profiling tools

- Command-line profiler
- nvprof
  - collecting and view of profiling data from the command-line
- Visual profiler (graphical tool)
  - Standalone application
  - Part of the Nsight EE

Preparing application for profiling

- No changes required to enable profiling
- Increasing the usability:
  - Focused profiling:
    - ✓ `cudaProfilerStart()`
    - ✓ `cudaProfilerStop()`
  - Flush profile data
    - ✓ `cudaDeviceReset()`
  - NVIDIA Tools Extension API (NVTX)
    - ✓ Marking regions
    - ✓ Naming CPU and GPU resources

Visual profiler

- nvvp from the command line
- Graphical program, simple to use
- Enhanced guided profiling in CUDA 5.5
- Features are similar to nvprof
- Results of nvprof can be imported

nvprof

- nvprof and command line profiler are mutually exclusive
- Supports several profiling modes:
  - Summary mode (default)
  - GPU-Trace and API-Trace mode
  - Event summary mode
  - Event trace mode

Summary mode

```
[user@tesla-apc SIMPLECUDA]$ nvprof ./main
======== NVPROF is profiling main...
======== Command: main
======== Profiling result:
 Time(%)  Time  Calls  Avg  Min  Max  Name
 53.11   73.61us  1 73.61us 73.61us  73.61us
  kern(float*, int)
 46.89   64.99us  1 64.99us 64.99us  64.99us
  [CUDA memcpy DtoH]
```

GPU-Trace

```
[user@tesla-apc SIMPLECUDA]$ nvprof --print-gpu-trace ./main
======== NVPROF is profiling main...
======== Command: main
======== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*
SSMem*    DSMem*       Size  Throughput    Device   Context
Stream  Name

195.42ms   71.64us           (1024 1 1)        (98 1 1)        14
 0B        40B          -           -          0          1          2
 kern(float*, int)

195.50ms   64.93us                     -               -         -
 -         -  400.00KB    6.16GB/s          0          1          2
 [CUDA memcpy DtoH]

Regs: Number of registers used per CUDA thread.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
```

API-Trace

```
[user@tesla-apc SIMPLECUDA]$ nvprof --print-api-trace ./main
======== NVPROF is profiling main...
======== Command: main
======== Profiling result:
     Start  Duration  Name
  146.65ms    1.00us  cuDeviceGetCount
  146.66ms       0ns  cuDeviceGet
  146.67ms   63.00us  cuDeviceGetName
  146.73ms   91.00us  cuDeviceTotalMem
  146.82ms    1.00us  cuDeviceGetAttribute
…
  218.00ms    1.00us  cudaSetupArgument
  218.00ms   17.00us  cudaLaunch
  218.02ms  313.00us  cudaMemcpy
  218.33ms  239.00us  cudaDeviceSynchronize
  218.73ms   55.14ms  cudaDeviceReset
```

Event summary

- **nvprof –query-events** to see the list of all events

```
[user@tesla-apc SIMPLECUDA]$ nvprof --events
   global_store_transaction ./main
======== NVPROF is profiling main...
======== Command: main
======== Profiling result:
    Invocations        Avg        Min        Max  Event Name
Device 0
  Kernel: kern(float*, int)
          1     100499     100499     100499
   global_store_transaction
```

Event trace mode

```
[auser@tesla-apc SIMPLECUDA]$ nvprof --events
  global_store_transaction --print-gpu-trace --
  aggregate-mode-off ./main
======== NVPROF is profiling main...
======== Command: main
======== Profiling result:
   Device   Context    Stream, Event Name, Kernel,
  Values
        0         1         2,
  global_store_transaction, kern(float*, int),
  6762 6214 7644 7938
```

Export/Import

- Use nvprof –output-profile filename to generate a result file
- Results can be imported:
  - In nvprof by nvprof –import-profile filename
  - In Visual Profile File->Import nvprof profile

Command line profiler

- Built into CUDA Runtime, active when **`COMPUTE_PROFILE=1`**
- Writes result to **`cuda_profile%d.log`**, where **`%d`** is GPU index, or to file specified by **`COMPUTE_PROFILE_LOG`**
- The set of active profiling counters could be specified in file, which name is recorded in **`COMPUTE_PROFILE_CONFIG`**. One counter per line, lines starting with «**#**» are comments

CCL Profiler

- Activating global memory loads/stores counters:

```
$ export COMPUTE_PROFILE=1
$ export COMPUTE_PROFILE_CONFIG=cuda_profile.cfg
$ cat cuda_profile.cfg
gld_request
gst_request

$ ./matmul1 2048
n = 2048
time spent executing kernel: 760.51 millseconds
time spent executing cublasSgemm_v2: 54.66
   millseconds
```

CCL Profiler Result

- After application's finished, the result is written into log file:

```
$ cat matmul2.log
# CUDA_PROFILE_LOG_VERSION 2.0
# CUDA_DEVICE 0 Tesla T10 Processor
# CUDA_CONTEXT 1
# TIMESTAMPFACTOR fffff6bc5fba6c00
method,gputime,cputime,occupancy,gld_request,gst_request
method=[ memcpyHtoD ] gputime=[ 3986.144 ] cputime=[ 4119.000 ]
method=[ memcpyHtoD ] gputime=[ 4009.024 ] cputime=[ 4074.000 ]
method=[ _Z7matmul2PfS_iS_ ] gputime=[ 85546.398 ] cputime=[ 85682.000 ]
occupancy=[ 1.000 ] gld_request=[ 1120256 ] gst_request=[ 4376 ]
…
```

Useful counters

- **divergent_branch**: incremented by one if at least one thread diverges via a data dependent conditional branch
- **gld_request**, **gst_request**: number of global data loads and stores
- **sm_cta_launched**: number of threads blocks launched on a multiprocessor
- **gridsize**, **threadblocksize**: number of blocks in a grid, number of threads in a block
- **8 user-defined counters**: might be use to track any custom events in CUDA kernels

- **Available counters depend on compute capability of GPU**

NVIDIA Tools Extension

- C-based API for annotating events, code ranges and resources
- Files:
  - `nvToolsExt.h`
  - `nvToolsExtCuda.h`
  - `nvToolsExtCudaRt.h`
  - `libnvToolsExt.so`

NVTX Markers

- Marker used to describe an instantaneous event

```
nvtxMarkA("My mark");

nvtxEventAttributes_t eventAttrib = {0};
eventAttrib.version = NVTX_VERSION;
eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE;
eventAttrib.colorType = NVTX_COLOR_ARGB;
eventAttrib.color = COLOR_RED;
eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII;
eventAttrib.message.ascii = "my mark with
  attributes";
nvtxMarkEx(&eventAttrib);
```

NVTX Range Start/Stop

- Ranges are used to detail specific time span

```
nvtxRangeId_t id1 = nvtxRangeStartA("My range");
nvtxRangeEnd(id1);

nvtxEventAttributes_t eventAttrib = {0};
eventAttrib.version = NVTX_VERSION;
eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE;
eventAttrib.colorType = NVTX_COLOR_ARGB;
eventAttrib.color = COLOR_BLUE;
eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII;
eventAttrib.message.ascii = "my start/stop range";
nvtxRangeId_t id2 = nvtxRangeStartEx(&eventAttrib);
nvtxRangeEnd(id2);
```

Ranges Push/Pop

```
nvtxRangePushA("outer");
nvtxRangePushA("inner");
nvtxRangePop(); // end "inner" range
nvtxRangePop(); // end "outer" range
```

NVTX Resource naming

```
// Windows
nvtxNameOsThread(GetCurrentThreadId()
  ,"MAIN_THREAD");

// Linux/Mac
nvtxNameOsThread(pthread_self(),
  "MAIN_THREAD");
```

NVTX Resource naming

```
nvtxNameCudaDeviceA(0, "my cuda
  device 0");

cudaStream_t cudastream;
cudaStreamCreate(&cudastream);
nvtxNameCudaStreamA(cudastream, "my
  cuda stream");
```

See also

- Compute_Visual_Profiler_User_guide.pdf
- CUPTI — CUDA Profiling Tools Interface — allows embedding profiler directly into applications
- Visual Profile metrics
- MPI Profiling