Tips and Tricks for Getting the Most Out of GPU-accelerated Clusters

Rolf vandeVaart, NVIDIA
Overview

- Highlight cluster aware tools
- Highlight NVIDIA provided tools
  - Debugging
  - Profiling
  - Memory Checking
Debugger - Third Party Tools Are Great

- Allinea DDT debugger

- GTC Presentation
  - S3059 - Debugging CUDA Applications with Allinea DDT

- Totalview

- GTC Presentations
  - S3177 - Porting Legacy Code of a Large, Complex Brain Simulation to a GPU-enabled Cluster
cuda-gdb with MPI Application

- You can use cuda-gdb just like gdb with the same tricks
- For smaller applications, just launch xterms and cuda-gdb

> mpirun -np 2 xterm -e cuda-gdb a.out
CUDA 5.0 and forward have the ability to attach to a running process

```c
if (rank == 2) {
    int i = 0;
    printf("rank %d: pid %d on %s ready for attach\n", rank, getpid(), name);
    while (0 == i) {
        sleep(5);
    }
}
```

```
[rolfv]$ mpirun -np 2 -host c0-2,c0-4 connectivity
rank 2: pid 20060 on c0-4 ready for attach
[rolfv]$ ssh c0-4
[rolf@c0-4]$ cuda-gdb --pid 20060
```
CUDA_DEVICE_WAITS_ON_EXCEPTION

The application encountered a device error and CUDA DEVICE WAITS ON EXCEPTION is set. You can now attach a debugger to the application (PID 22279) for inspection.

Program received signal CUDA EXCEPTION 10, Device Illegal Address.
[Switching focus to CUDA kernel 1, grid 1, block (0,0,0), thread (0,0,0), device 0, sm 2, warp 0, lane 0]

0x0000000000000000 in MemoryBoundDeviceMemory<<<(1024,1,1),(512,1,1)>> (A=0x06300000, B=0x06700000, C=0x00b00000, N=524288, worldRank=0)
at program1.cu:80
80 C[i] = C[i] + A[i] + B[i];

cuda-gdb

- New with CUDA 5.5 is that we get the hostname also with the exception message.

- Also new with 5.5 is the ability to debug with multiple GPUs on a node. Need to disable lockfile.

- mpirun -np 2 -xterm cuda-gdb -cuda-use-lockfile=0 a.out
Cluster Aware Profiling

- We work with various parallel profilers
  - VampirTrace
  - Tau
- Also CUDA specific profiling information part of score-p
- These tools along with various Visual tools are good for discovering MPI issues as well as basic CUDA performance inhibitors
Profiling

- Two methods of collecting profiling information via command line
  - nvprof
  - command line profiling
- How to use them with MPI is documented in Profiler User Guide.
- Key is to get all the profiling output in unique files
nvprof helper script

- # Open MPI
- if [ ! -z ${OMPI_COMM_WORLD_RANK} ] ; then
-   rank=${OMPI_COMM_WORLD_RANK}
- fi
- # MVAPICH
- if [ ! -z ${MV2_COMM_WORLD_RANK} ] ; then
-   rank=${MV2_COMM_WORLD_RANK}
- fi

- # Set the nvprof command and arguments.
- NVPROF="nvprof --output-profile $outfile.$rank nvprof_args"
- exec $NVPROF $*
Profiling - nvprof

[rolf]$ mpirun -np 4 -host c0-0,c0-2,c0-3,c0-4 nvprof-script.sh --print-api-trace -c . ./program1

-------- NVPROF is profiling program1...
-------- Command: program1
-------- NVPROF is profiling program1...
-------- Command: program1
-------- NVPROF is profiling program1...
-------- Command: program1
-------- NVPROF is profiling program1...
-------- Command: program1

rank 1: Device Name: Tesla M2050
rank 3: Device Name: Tesla M2070
rank 0: Device Name: Tesla M2050
rank 2: Device Name: Tesla M2090

-------- Generated result file: program1.nvprof-out.1
-------- Generated result file: program1.nvprof-out.2
-------- Generated result file: program1.nvprof-out.0
-------- Generated result file: program1.nvprof-out.3

[rolf]$
nvprof

- New in CUDA 5.5
- Embed pid in output file name
  - mpirun -np 2 nvprof --output-profile profile.out.%p
- If you want to just save the textual output.
  - mpirun -np 2 nvprof --log-file profile.out.%p
- Collect profile data on all processes that run on a node
  - nvprof --profile-all-processes -o profile.out.%p
nvprof - post process

```
[rolf]$ nvprof -i profile.out.3796
---------- Profiling result:

Time(%)   Time          Calls   Avg   Min   Max   Name
60.53%   68.527ms      1       68.527ms 68.527ms 68.527ms MemoryBoundSystemMemory(int*, int*, int*, int*, int*, int)
34.69%   39.272ms      1       39.272ms 39.272ms 39.272ms MemoryBoundTextureCache(int*, int)
2.35%    2.662ms       2       1.3310ms 1.3032ms 1.3589ms [CUDA memcpy HtoD]
1.08%    1.2185ms      1       1.2185ms 1.2185ms 1.2185ms MemoryBoundL1Cache(int const *, int const *, int*, int)
0.82%    924.16us      1       924.16us 924.16us 924.16us [CUDA memcpyDtoH]
0.29%    328.16us      1       328.16us 328.16us 328.16us MemoryBoundDeviceMemory(int*, int*, int*, int, int)
0.25%    283.49us      1       283.49us 283.49us 283.49us MemoryBoundL2Cache(int*, int*, int*, int)

[rolf]$  
```
NVIDIA Visual Profiler

[Image of NVIDIA Visual Profiler interface showing a detailed analysis of a CUDA kernel execution, including timeline, events, and performance metrics.]
Profiler - command line

- Everything is controlled by environment variables.
  - \( \text{setenv } \) \text{COMPUTE\_PROFILE\_LOG} \text{ cuda\_profile.}d.\%p
  - \( \text{setenv } \) \text{COMPUTE\_CONFIG} ./profile.config
  - \( \text{setenv } \) \text{COMPUTE\_PROFILE} 1
  - \( \text{mpirun -x COMPUTE\_PROFILE\_LOG -x COMPUTE\_CONFIG -x COMPUTE\_PROFILE -np 4 -host c0-0,c0-2,c0-3,c0-4 program1} \)
cuda-memcheck

- Functional correctness checking suite
- mpirun -np 2 cuda-memcheck a.out
cuda-memcheck

- Similar issues to running nvprof
- Use nvprof script and make two substitutions
  
  \textit{nvprof} to \textit{cuda-memcheck}
  
  \textit{--output-profile} to \textit{--save}

- mpirun -np 2 cuda-memcheck-script.sh a.out
cuda-memcheck

[rolf]$ mpirun -np 4 -host c0-0,c0-2,c0-3,c0-4 memcheck-script.sh -o mcheck -c ./program1
======== CUDA-MEMCHECK
======== CUDA-MEMCHECK
======== CUDA-MEMCHECK
======== CUDA-MEMCHECK

rank 3: Device Name: Tesla M2070

[..snip..]

======== ERROR SUMMARY: 2 errors
======== ERROR SUMMARY: 0 errors
======== ERROR SUMMARY: 0 errors
======== ERROR SUMMARY: 0 errors

-----------------------------------------------
While the primary job terminated normally, 1 process returned
a non-zero exit code. Further examination may be required.

[rolf]$
cuda-memcheck

- Read the output using cuda-memcheck. Look for any log files that are non-empty.

```
[rolf]$ ls -lt mcheck*
-rw-rw-r-- 1 rolf rolf 2517 Mar 11 10:07 mcheck.0
-rw-rw-r-- 1 rolf rolf  0 Mar 11 10:07 mcheck.1
-rw-rw-r-- 1 rolf rolf  0 Mar 11 10:07 mcheck.2
-rw-rw-r-- 1 rolf rolf  0 Mar 11 10:07 mcheck.3
[rolf]$ cuda-memcheck --read mcheck.0
======== CUDA-MEMCHECK
======== Invalid _global_ write of size 4
======== at 0x00600038 in /home/rolf/gtc-2013/program1.cu:77:MemoryBoundDeviceMemory(int*, int*, int*, int, int)
======== by thread (0,0,0) in block (0,0,0)
======== Address 0xf02f259fc is out of bounds
======== Saved host backtrace up to driver entry point at kernel launch time
```
Conclusion

- If you can, make use of all the cluster aware debugging and profiling tools. NVIDIA partners with them.

- Otherwise, NVIDIA supplies many tools that can still help you get productivity out of your application.
  - cuda-gdb
  - nvprof, NVIDIA Visual Profiler
  - cuda-memcheck
Questions

- rvandevaart@nvidia.com