
GPU Tools
I’m on the GPU, now what?
NVIDIA

CUDA-Memcheck

- You're hitting an error or getting wrong results, try cuda-memcheck first.
  - Reports OOB memory accesses
  - Reports errors from CUDA calls
  - https://developer.nvidia.com/cuda-memcheck
- Works with CUDA and OpenACC

```
$ aprun cuda-memcheck app.exe
```

CUDA-memcheck Output


```
========= CUDA-MEMCHECK
0.000000
========= Invalid __global__ read of size 4
=========     at 0x00000098 in saxpy$ck_L5_2
=========     by thread (0,0,0) in block (0,0,0)
=========     Address 0xb00c00000 is out of bounds
=========     Device Frame:<1 frames were hidden>
=========     Saved host backtrace up to driver entry point at kernel launch time
=========     Host Frame:<9 frames were hidden>
=========     Host Frame:/opt/cray/nvidia//default/lib64/libcuda.so.1 (cuLaunchKernel +
0x3ae) [0xc863e]
=========     Host Frame:/opt/cray/cce/8.1.7/craylibs/x86-64/libcrayacc.so.0
(__cray_acc_hw_start_kernel + 0x1072) [0x1b0a6]
=========     Host Frame:/opt/cray/cce/8.1.7/craylibs/x86-64/libcrayacc.so.0 [0x7c47]
=========     Host Frame:/opt/cray/cce/8.1.7/craylibs/x86-64/libcrayacc.so.0
(cray_start_acc_kernel + 0x114) [0x807e]
=========     Host Frame:./a.out [0xf01]
=========     Host Frame:./a.out [0xd81]
=========     Host Frame:/lib64/libc.so.6 (__libc_start_main + 0xe6) [0x1ec36]
=========     Host Frame:./a.out [0xac9]
========= ERROR SUMMARY: 3 errors
Application 219996 resources: utime ~6s, stime ~1s
```

Compiler Profiling Variables

- The Cray compiler provides automatic instrumentation when `CRAY_ACC_DEBUG=<1,2,3>` at runtime

```
ACC: Initialize CUDA
ACC: Get Device 0
ACC: Create Context
ACC: Set Thread Context
ACC: Start transfer 2 items from saxpy.c:17
ACC:       allocate, copy to acc 'x' (4194304 bytes)
ACC:       allocate, copy to acc 'y' (4194304 bytes)
ACC: End transfer (to acc 8388608 bytes, to host 0 bytes)
ACC: Execute kernel saxpy$ck_L17_1 blocks:8192 threads:128
async(auto) from saxpy.c:17
ACC: Wait async(auto) from saxpy.c:18
ACC: Start transfer 2 items from saxpy.c:18
ACC:       free 'x' (4194304 bytes)
ACC:       copy to host, free 'y' (4194304 bytes)
ACC: End transfer (to acc 0 bytes, to host 4194304 bytes)
```

Compiler Profiling Variables

- The PGI compiler provides automatic instrumentation when `PGI_ACC_TIME=1` at runtime

```
Accelerator Kernel Timing data
/home/jlarkin/kernels/saxpy/saxpy.c
  saxpy  NVIDIA  devicenum=0
        time(us): 3,256
        11: data copyin reached 2 times
             device time(us): total=1,619 max=892 min=727 avg=809
        11: kernel launched 1 times
            grid: [4096]  block: [256]
             device time(us): total=714 max=714 min=714 avg=714
            elapsed time(us): total=724 max=724 min=724 avg=724
        15: data copyout reached 1 times
             device time(us): total=923 max=923 min=923 avg=923
```

CUDA Profiler (nvprof)

- At its most basic, nvprof will instrument your application and provide information about all CUDA-related activity.
- It's also possible to use nvprof to gather data for the CUDA Visual Profiler for viewing on your machine.

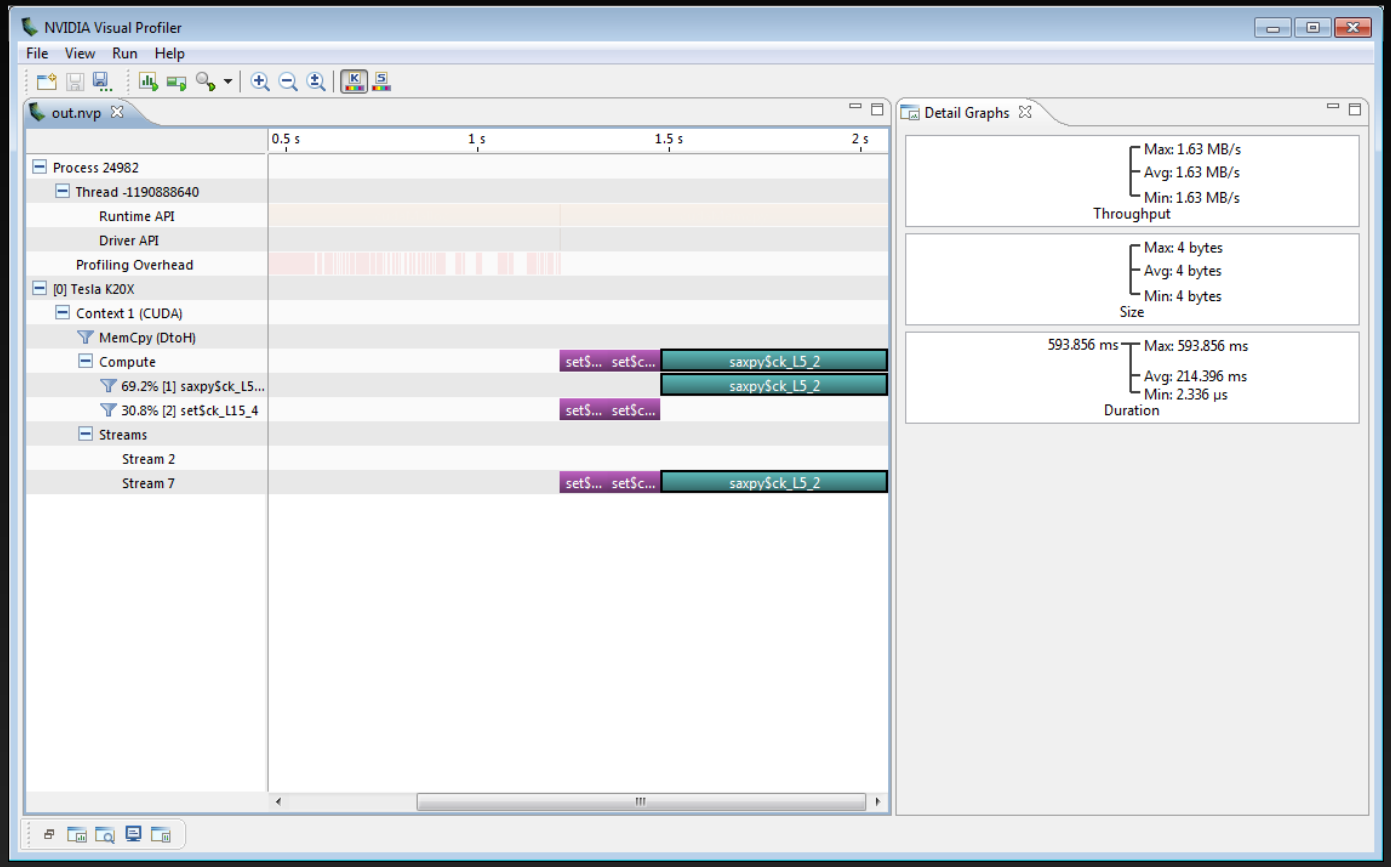
- NOTE: On Cray XK7, it's necessary to set the environment variable below to gather data.

```
export PMI_NO_FORK=1
setenv PMI_NO_FORK 1
```

NVProf Basic Output


```
$ aprun nvprof ./a.out
======== NVPROF is profiling a.out...
======== Command: a.out
2.000000
======== Profiling result:
 Time(%)      Time   Calls       Avg       Min       Max
Name
   70.20  594.27ms       1  594.27ms  594.27ms  594.27ms
saxpy$ck_L5_2
   29.80  252.26ms       2  126.13ms  126.13ms  126.13ms
set$ck_L15_4
    0.00    2.34us       1    2.34us    2.34us    2.34us
[CUDA memcpy DtoH]
```

Nvidia Visual Profiler



Instrument on compute node with: `aprun nvprof –o out.nvp a.out`
Then import into Visual Profiler on your local machine to analyze.

NVProf XK7 Trick

- When running a MPI app, all processes will write to the same file, but try this trick to get 1 per node:

```
#!/bin/bash
# USAGE: Add between aprun options and executable
# For Example: aprun -n 16 -N 1 ./foo arg1 arg2
# Becomes: aprun -n 16 -N 1 ./nvprof.sh ./foo arg1 arg2

# Give each *node* a separate file
LOG=profile_$(hostname).nvp

# Stripe each profile file by 1 to share the load on large runs
lfs setstripe -c 1 $LOG

# Execute the provided command.
exec nvprof -o $LOG $*
```

- Explanation: this script intercepts the call to your executable, determines a unique filename based on the compute node, and calls nvprof.

See `nvprof.sh` in the workshop github repo.

CUDA Command-Line Profiler

- Any CUDA or OpenACC program can also get a more detailed profile via the command-line profiler.

```
export COMPUTE_PROFILE=1
```

- Many performance counters are available.

```
export COMPUTE_PROFILE_CONFIG=events.txt
```

- Outputting to CSV allows importing into Visual Profiler

```
export COMPUTE_PROFILE_CSV=1
```

CLI Profiler Trick

- This trick matches the `nvprof` trick for getting a unique log file for each XK7 node.

```
#!/bin/bash
# USAGE: Add between aprun options and executable
# For Example: aprun -n 16 -N 1 ./foo arg1 arg2
# Becomes: aprun -n 16 -N 1 ./profile.sh ./foo arg1 arg2

# Enable command-line profiler
export COMPUTE_PROFILE=1

# Set output to CSV (optional)
export COMPUTE_PROFILE_CSV=1

# Give each *node* a separate file
export COMPUTE_PROFILE_LOG=cuda_profile_$(hostname).log

# Stripe each profile file by 1 to share the load on large runs
lfs setstripe -c 1 $COMPUTE_PROFILE_LOG

# Execute the provided command.
exec $*
```

See `profile.sh` and `profile_csv.sh` in the workshop github repo.