CS542200
Parallel Programming
Lab 2: CUDA

CHEN-CHUN CHEN

2016/12/08

Outline

1. Platform Guide
2. Programming Guide
3. Tools
4. Reference

Outline

1. Platform Guide
2. Programming Guide
3. Tools
4. Reference

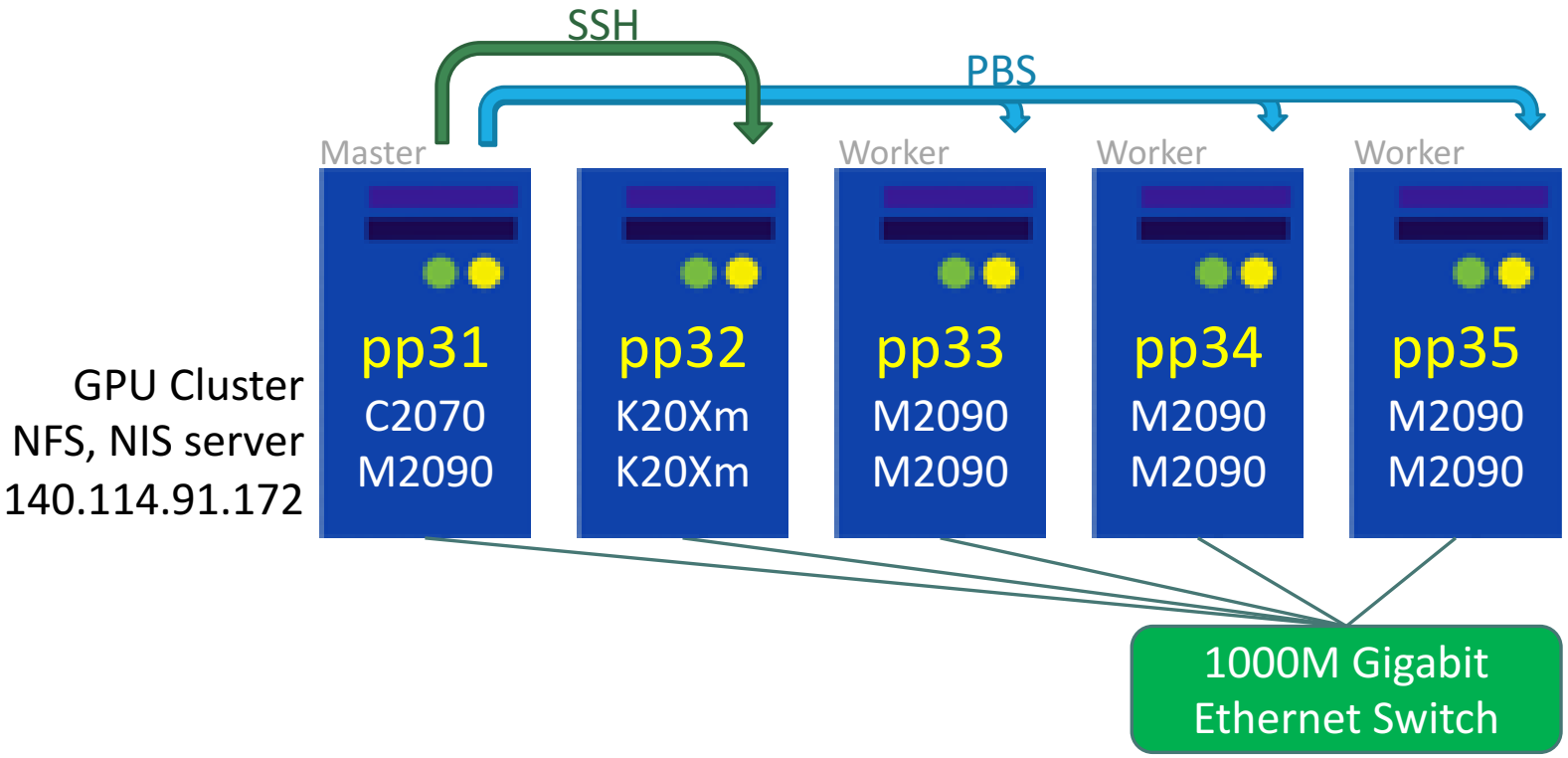
The GPU Cluster

Host: 140.114.91.172

Account: same as pp01

Password: same as pp01(Dec. 3rd)

The GPU Cluster



Job Scheduler

Master: pp31

Worker: pp33, pp34, pp35

Resource Manager: TORQUE-5.1.1.2

Scheduler: TORQUE-5.1.1.2

- Max nodes = 2
- Max walltime = 30 minute
- Max jobs queuable at any time = 4
- Max jobs runnable at any time = 2

Priority: FIFO

```
#PBS -N CUDA_JOB
#PBS -r n
#PBS -l nodes=1:ppn=1:gpus=1:exclusive_process
#PBS -l walltime=00:01:00

cd $PBS_O_WORKDIR
./executable args
```

Password-less SSH

Execute the following script to generate key and go through all nodes:

- `$ /home/pp2016/shared/auto_keygen.sh`

Outline

1. Platform Guide
2. Programming Guide
3. Tools
4. Reference

Compile & run

Compile

- `nvcc [options] input_file`
- Example: `nvcc -o executable code.cu`

Run

- `./executable [args]`

For more options, please see `nvcc --help`

Steps to follow

1. Initialize CUDA device
2. Allocate memory in device & put sequential code into kernel function
3. Relabel index variables with combinations of `threadIdx, blockIdx, blockDim, gridDim`
4. Optimizations (requires great deal of effort!)

How to measure kernel execution time?

cudaEventCreate(): Init timer

cudaEventDestroy(): Destroy timer

cudaEventRecord(): Set timer

cudaEventSynchronize(): Sync timer after each kernel call

cudaEventElapsedTime(): Returns the elapsed time in milliseconds

How to measure kernel execution time?

```
cudaEvent_t start, stop;
float time;

cudaEventCreate (&start);
cudaEventCreate (&stop);

cudaEventRecord (start, 0);
kernel <<< grid, threads >>> (d_in, d_out);
cudaEventRecord (stop, 0);
cudaEventSynchronize (stop);

cudaEventElapsedTime (&time, start, stop);
fprintf (stderr, "%lf\n", time);

cudaEventDestroy (start);
cudaEventDestroy (stop);
```

Outline

1. Platform Guide
2. Programming Guide
3. Tools
4. Reference

nvidia-smi

Purpose: Query and modify GPU's state

You can query details about

- device type
- clock rate
- temperature
- power
- memory
- …

nvidia-smi: example

```
[root@pp31 ~]# nvidia-smi
Sun Dec  4 15:49:05 2016
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 367.48                 Driver Version: 367.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla C2070         Off  | 0000:03:00.0     Off |                  Off |
| 30%   52C    P0    N/A /  N/A |      0MiB /  6066MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M2090         Off  | 0000:06:00.0     Off |                  Off |
| N/A   N/A    P0    75W /  N/A |      0MiB /  6066MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

nvidia-smi: example

```
[root@pp31 ~]# nvidia-smi -q -d CLOCK
==============NVSMI LOG==============
Timestamp                           : Sun Dec  4 15:53:09 2016
Driver Version                      : 367.48
Attached GPUs                       : 2
GPU 0000:03:00.0
    Clocks
        Graphics                    : 573 MHz
        SM                          : 1147 MHz
        Memory                      : 1494 MHz
        Video                       : 540 MHz
    Applications Clocks
        Graphics                    : N/A
        Memory                      : N/A
    Default Applications Clocks
        Graphics                    : N/A
        Memory                      : N/A
    Max Clocks
        Graphics                    : 573 MHz
        SM                          : 1147 MHz
        Memory                      : 1494 MHz
        Video                       : 540 MHz
```

cuda-memcheck

This tool checks memory errors of your program, and it also reports hardware exceptions encountered by the GPU.

These errors may not cause program to crash, but they could result in unexpected program behavior and memory misusage.

cuda-memcheck

Some erroneous code

```
        cudaFree (d_data);
        cudaFree (d_data); // error
    return 0;
}
```

Error summary

```
[user0@gpucluster0 shared]$ cuda-memcheck sobel candy.bmp
========= CUDA-MEMCHECK
========= Program hit cudaErrorInvalidDevicePointer (error 17) due to "invalid
device pointer" on CUDA API call to cudaFree.
=========     Saved host backtrace up to driver entry point at error
=========     Host Frame:/lib64/libcuda.so.1 [0x2e4263]
=========     Host Frame:sobel [0x3dcb6]
=========     Host Frame:sobel [0x27b1]
=========     Host Frame:/lib64/libc.so.6 (__libc_start_main + 0xf5) [0x21af5]
=========     Host Frame:sobel [0x287d]
=========
========= ERROR SUMMARY: 1 error
```

cuda-memcheck error types

Name	Description	Location	Precision
Memory access error	Errors due to out of bounds or misaligned accesses to memory by a global, local, shared or global atomic access.	Device	Precise
Hardware exception	Errors that are reported by the hardware error reporting mechanism.	Device	Imprecise
Malloc/Free errors	Errors that occur due to incorrect use ofmalloc()/free() in CUDA kernels.	Device	Precise
CUDA API errors	Reported when a CUDA API call in the application returns a failure.	Host	Precise
cudaMalloc memory leaks	Allocations of device memory using cudaMalloc()that have not been freed by the application.	Host	Precise
Device Heap Memory Leaks	Allocations of device memory using malloc() in device code that have not been freed by the application.	Device	Imprecise

cuda-gdb

Similar to GDB

A tool provides developers with a mechanism for debugging CUDA application running on actual hardware.

For more details, please refer to **cuda-debugging-tools.pdf**

cuda-gdb: print/set variables

Print variable

```
(cuda-gdb) print total
$1 = 11.1110363
```

Reassign value to variable

```
(cuda-gdb) print total = 31.1095
$2 = 31.109499
```

cuda-gdb: breakpoint

by kernel name

```
(cuda-gdb) break sobel_Kernel
```

by file & line number

```
(cuda-gdb) break test.cu:149
```

by address

```
(cuda-gdb) break 0x4e15f73
```

cuda-gdb: execution control

Launch application (with arguments)

```
(cuda-gdb) run arg1 arg2
```

Resume execution

```
(cuda-gdb) continue
```

Kill the program

```
(cuda-gdb) kill
```

cuda-gdb: execution control

Interrupt the program

- Ctrl + C

Single stepping

	At source level	At assembly level
Over function calls	next	nexti
Into function calls	step	stepi

nvprof

A CUDA profiler

Provides feedback to optimize CUDA programs

--metrics <METRIC_NAME> to measure specific metrics

--events <EVENT_NAME> to record specific events

-o <FILE> to save result to a file

-i <FILE> to read result from a file

nvvp

nvprof’s GUI counterpart

easier to use

CUDA Profiling with MPI

Assume we run with 2 MPI tasks

```
$ mpirun -n 2 nvprof -o
profile.out.%q{PMI_RANK} ./exe
```

This generates **profile.out.0** and **profile.out.1**

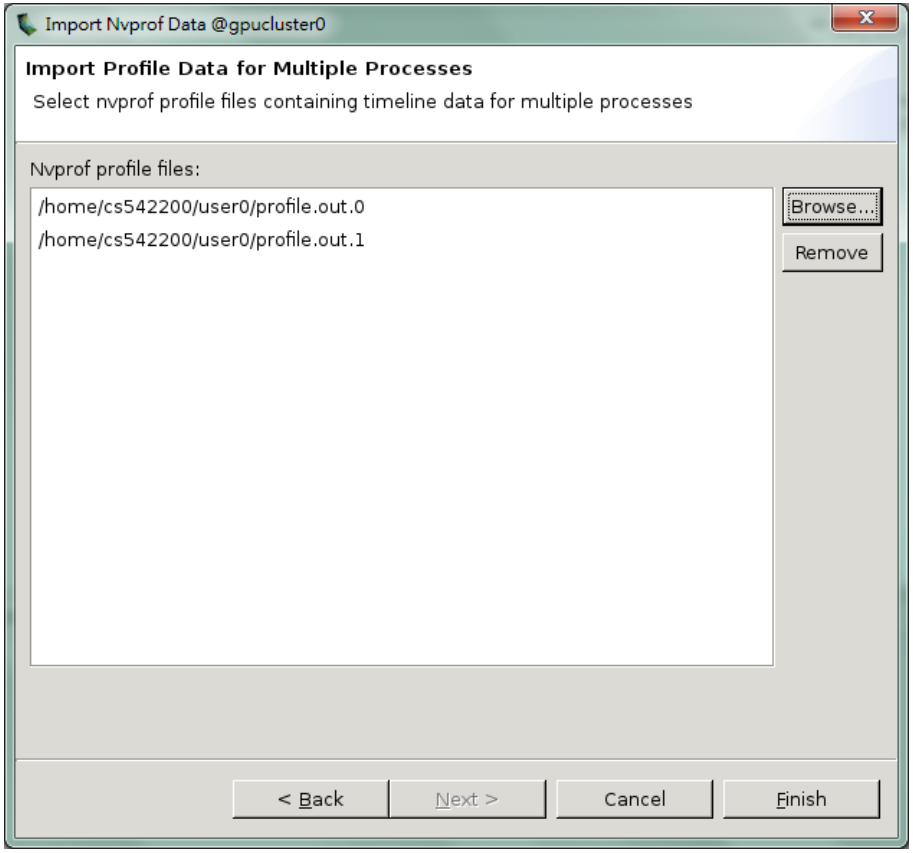
We can use **nvprof** or **nvvp** to further analyze them

CUDA+MPI: nvprof example

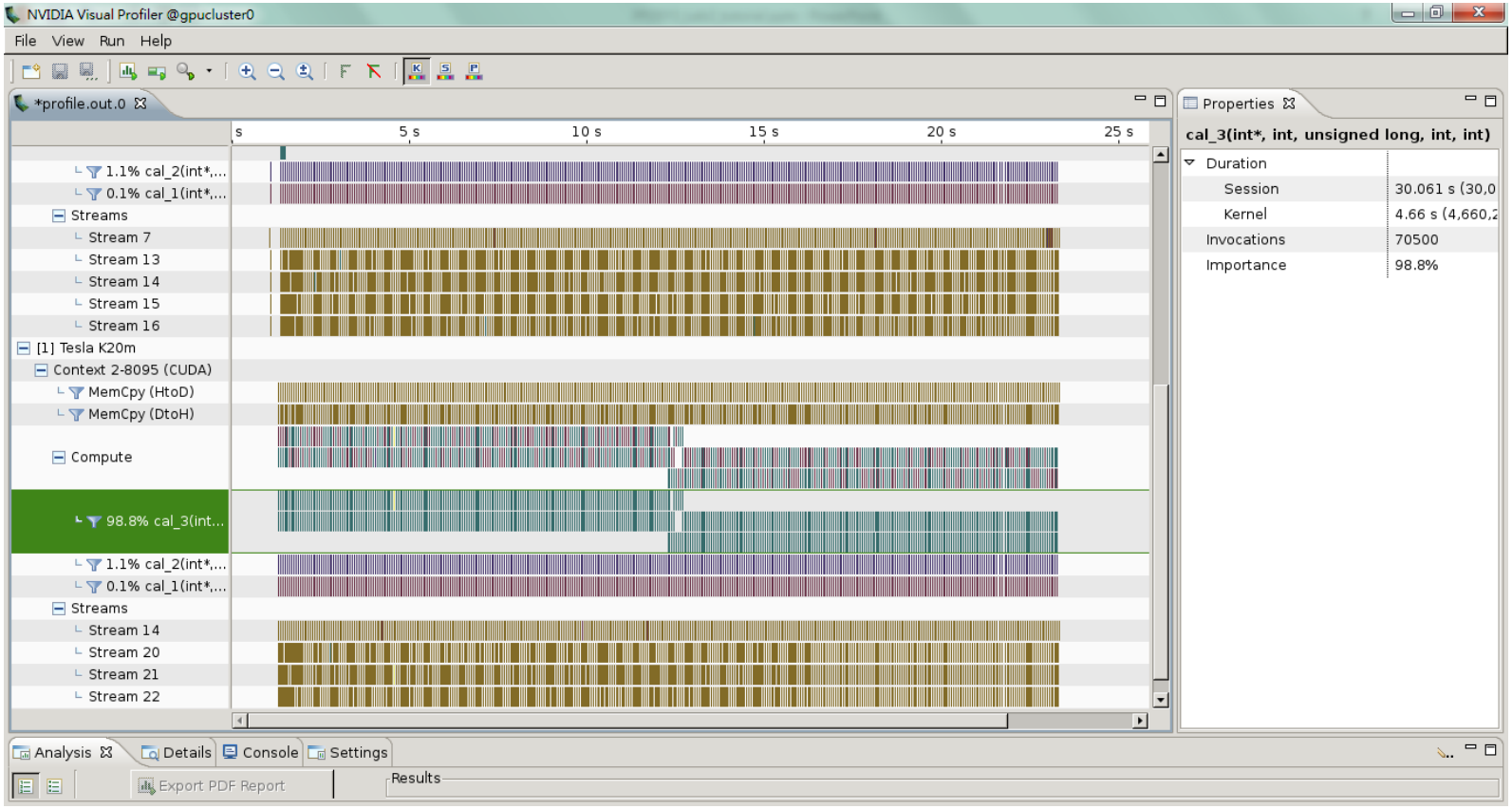
```
[user0@gpucluster0 ~]$ nvprof -i profile.out.0
======== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 34.28%  4.63078s     70125  66.036us  3.3930us  81.699us  cal_3(int*, int, unsigned long, int,
int)
 32.97%  4.45322s       376  11.844ms  11.790ms  23.636ms  [CUDA memcpy HtoD]
 32.32%  4.36655s     70126  62.267us  59.458us  21.517ms  [CUDA memcpy DtoH]
  0.40%  53.702ms       375  143.20us  141.96us  145.54us  cal_2(int*, int, unsigned long, int)
  0.03%  4.1410ms       375  11.042us  10.880us  11.392us  cal_1(int*, int, unsigned long, int)

======== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 47.91%  4.49867s       377  11.933ms  11.849ms  23.724ms  cudaMemcpy2D
 29.17%  2.73946s       375  7.3052ms  2.7430ms  7.6579ms  cudaDeviceSynchronize
  8.89%  834.79ms     70875  11.778us  9.6190us  1.6846ms  cudaLaunch
  7.31%  686.19ms     70125  9.7850us  8.2120us  30.115ms  cudaMemcpy2DAsync
  4.59%  431.25ms         1  431.25ms  431.25ms  431.25ms  cudaHostAlloc
  1.43%  134.59ms    353625     380ns     255ns  571.46us  cudaSetupArgument
  0.37%  34.631ms     70875     488ns     375ns  557.99us  cudaConfigureCall
  0.30%  28.024ms         1  28.024ms  28.024ms  28.024ms  cudaFreeHost
  0.01%  1.0072ms         1  1.0072ms  1.0072ms  1.0072ms  cudaMallocPitch
  0.01%  972.39us       166  5.8570us     554ns  204.95us  cuDeviceGetAttribute
  0.01%  645.85us         4  161.46us  19.732us  524.41us  cudaStreamCreate
  0.00%  291.91us         1  291.91us  291.91us  291.91us  cudaFree
```

CUDA+MPI: nvvp example(1)



CUDA+MPI: nvvp example(2)



Outline

1. Platform Guide
2. Programming Guide
3. Tools
4. Reference

References

NVIDIA CUDA Toolkit Documentation

NVIDIA CUDA Runtime API Documentation

Vyas Venkataraman, “CUDA debugging tools: CUDA-GDB & CUDA-MEMCHECK,” GTC 2014.