Lab3 CUDA Basic

Oct, 2025 Parallel Programming

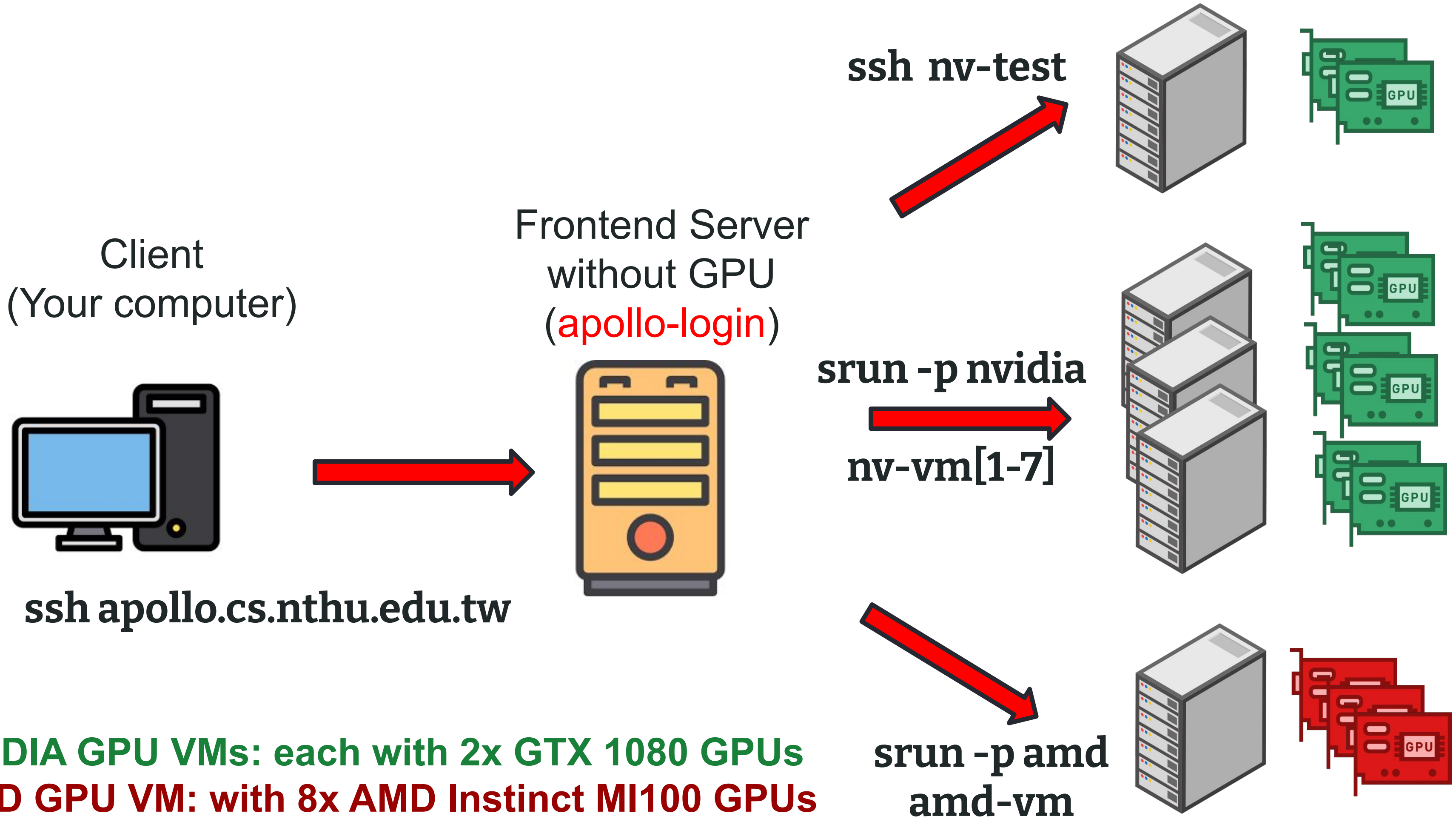
Overview

- Platform guide
- Tools
- Assignment

Platform Guide

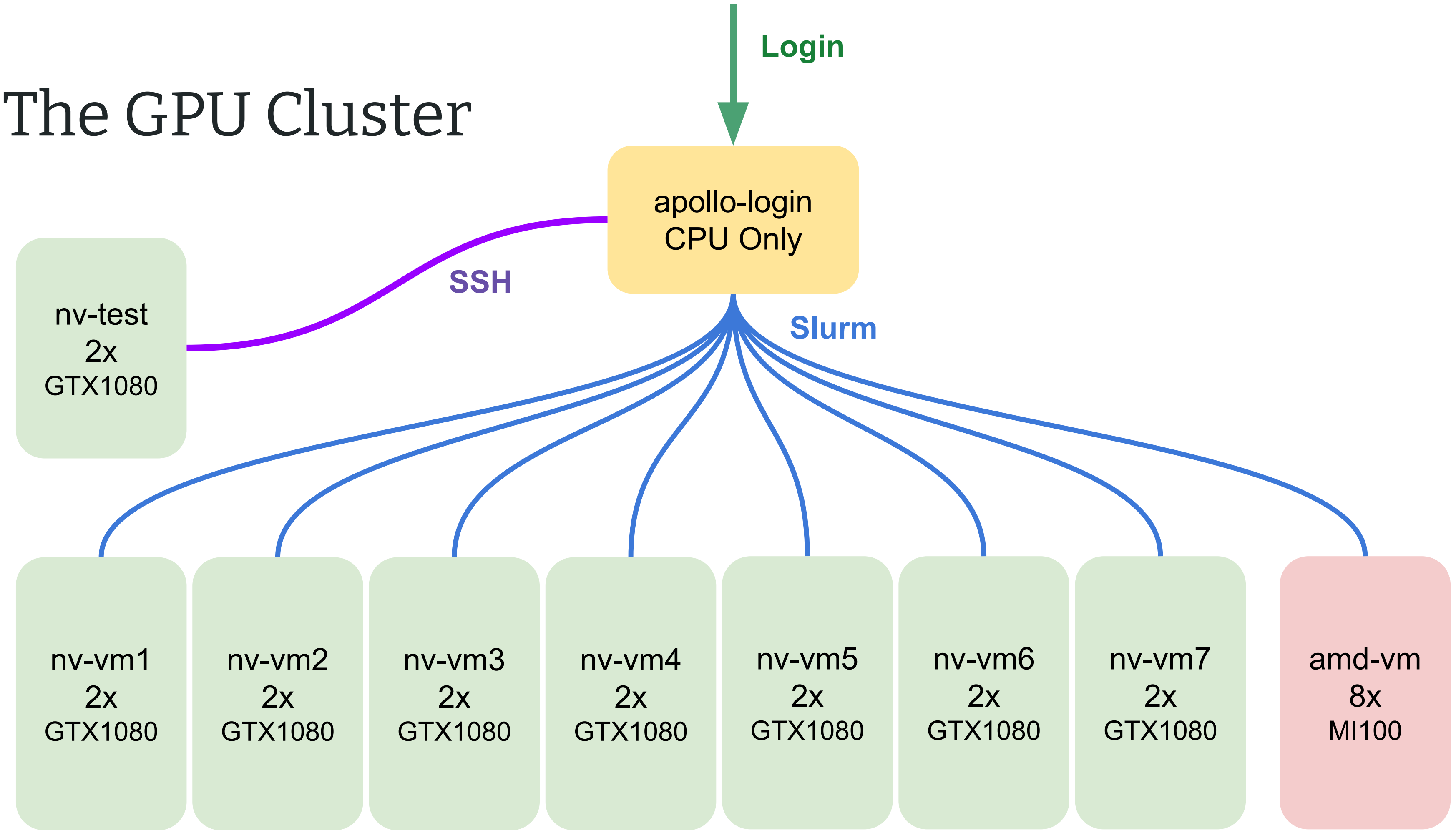
The GPU Cluster

- Host: **`apollo.cs.nthu.edu.tw`**
- Account & Password: Same as apollo origo



8 NVIDIA GPU VMs: each with 2x GTX 1080 GPUs
2 AMD GPU VM: with 8x AMD Instinct MI100 GPUs

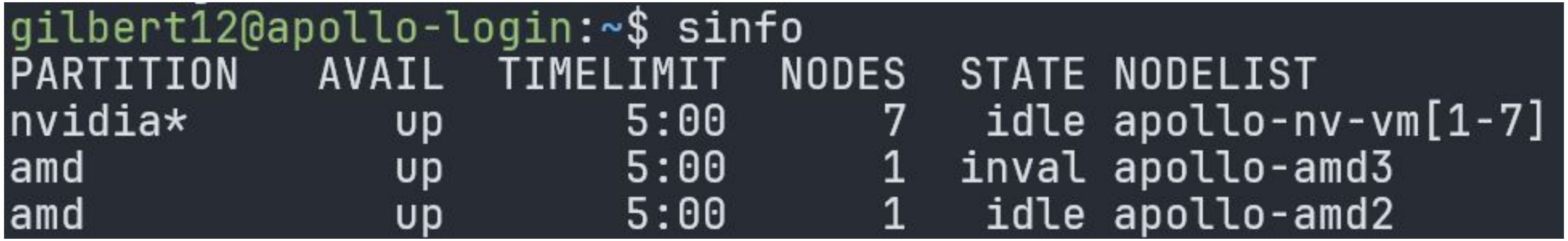
The GPU Cluster



Job Scheduler

❖ Slurm

❖ Partitions: **nvidia** for NVIDIA GPUs (default), **amd** for AMD GPUs



```
gilbert12@apollo-login:~$ sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
nvidia*        up       5:00      7   idle apollo-nv-vm[1-7]
amd            up       5:00      1  inval apollo-amd3
amd            up       5:00      1   idle apollo-amd2
```

❖ Limitations

- ➢ 1GPU or 2 GPUs per Job
- ➢ 2 CPU cores per GPU (i.e. 1 GPU -> 2 cores, 2 GPUs -> 4 cores)
- ➢ 2 Jobs per User
- ➢ Wall time: 5 minutes

Instructions to compile a CUDA program

- Load cuda first!
  - `module load cuda`
- Compile
  - `nvcc -arch=sm_61 [other options] <inputfile>`
  - e.g.,

    ```
    nvcc cuda_code.cu -o cuda_executable
    ```
  - `sm_61` is Compute Capability 6.1, which is what GTX 1080 supports
    - If you are using your own GPU, find your GPU's compute capability here

- If you have a Makefile, simply
  - `make`

Instructions to run a CUDA program

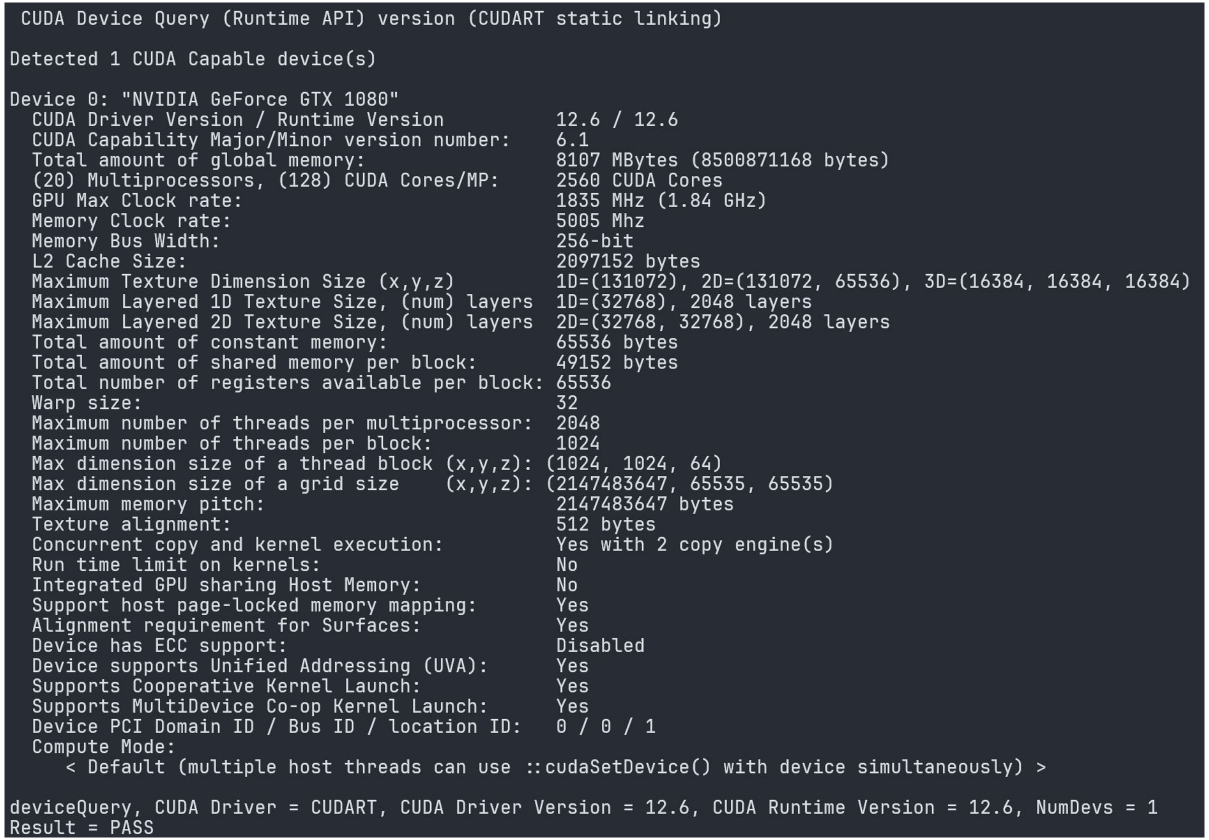
- apollo-nv-test
  - **`ssh apollo-nv-test`** or **`ssh nv-test`**
  - If you want to specify which GPU to use.
    - `export CUDA_VISIBLE_DEVICES=<gpu id>`
    - eg. `export CUDA_VISIBLE_DEVICES=1`
    - eg. `export CUDA_VISIBLE_DEVICES=0,1`

- apollo-nv-vm[1-7]
  - Slurm
  - Access gpus with the flag: `--gres=gpu:<number of gpu>`
    - eg. `srun -n 1 --gres=gpu:1 ./executable`
    - eg. `srun -n 1 --gres=gpu:2 ./executable`
  - If two GPUs are requested, they will be on the same node.

Practice

- In this practice, try to run the **`deviceQuery`**
- Steps:
  - `cp -r /home/pp25/share/deviceQuery $HOME`
  - `cd $HOME/deviceQuery`
  - `nvcc deviceQuery.cpp -o deviceQuery`
- Run it
  - on apollo-nv-test
  - with Slurm scheduler on apollo-nv-vm[1-7]
- How many CUDA cores on NVIDIA GTX 1080?

Device Query - Result



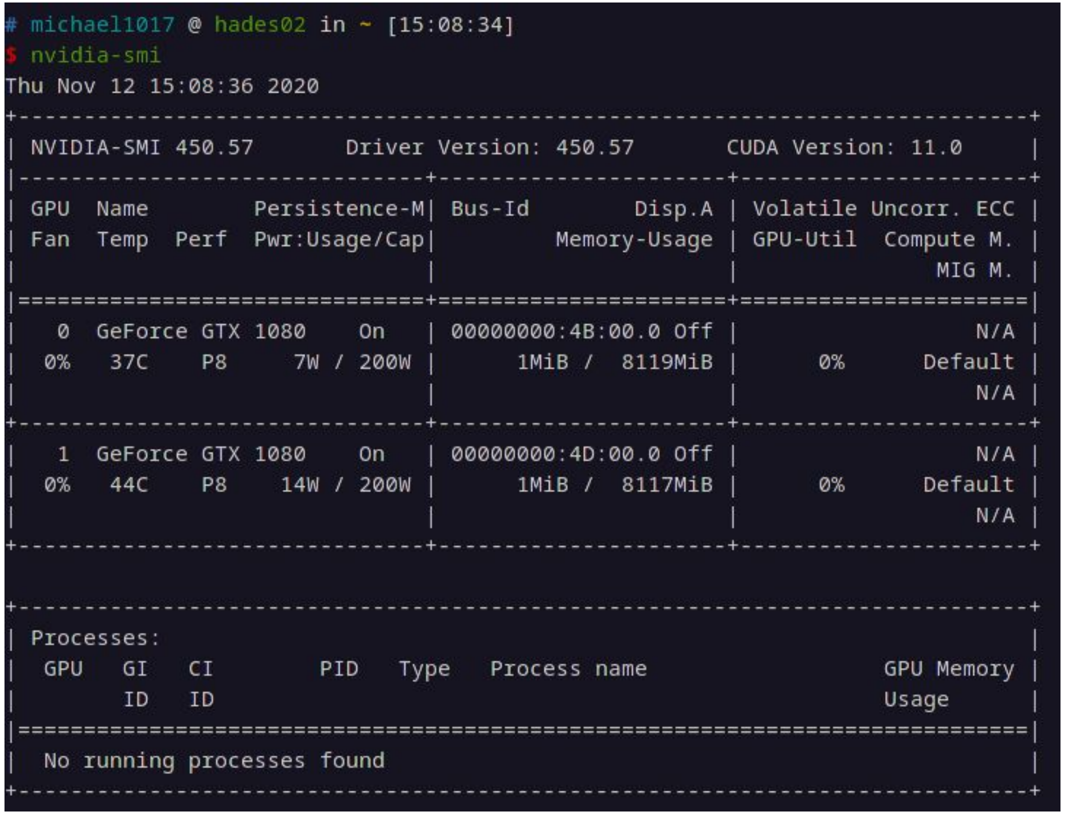
```
 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce GTX 1080"
  CUDA Driver Version / Runtime Version          12.6 / 12.6
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 8107 MBytes (8500871168 bytes)
  (20) Multiprocessors, (128) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1835 MHz (1.84 GHz)
  Memory Clock rate:                             5005 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 0 / 1
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 12.6, CUDA Runtime Version = 12.6, NumDevs = 1
Result = PASS
```

Tools

nvidia-smi

- NVIDIA System Management Interface program
- You can query details about
  - gpu type
  - gpu utilization
  - memory usage
  - temperature
  - clock rate
  - ...

nvidia-smi example



```
# michael1017 @ hades02 in ~ [15:08:34]
$ nvidia-smi
Thu Nov 12 15:08:36 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.57       Driver Version: 450.57       CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce GTX 1080    On   | 00000000:4B:00.0 Off |                  N/A |
|  0%   37C    P8     7W / 200W |      1MiB /  8119MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 1080    On   | 00000000:4D:00.0 Off |                  N/A |
|  0%   44C    P8    14W / 200W |      1MiB /  8117MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+


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

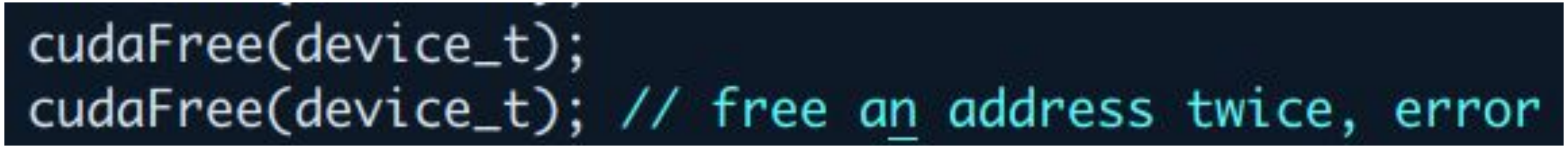
compute-sanitizer

- Compute Sanitizer is a functional correctness checking suite included in the CUDA toolkit. This suite contains multiple tools that can perform different type of checks.

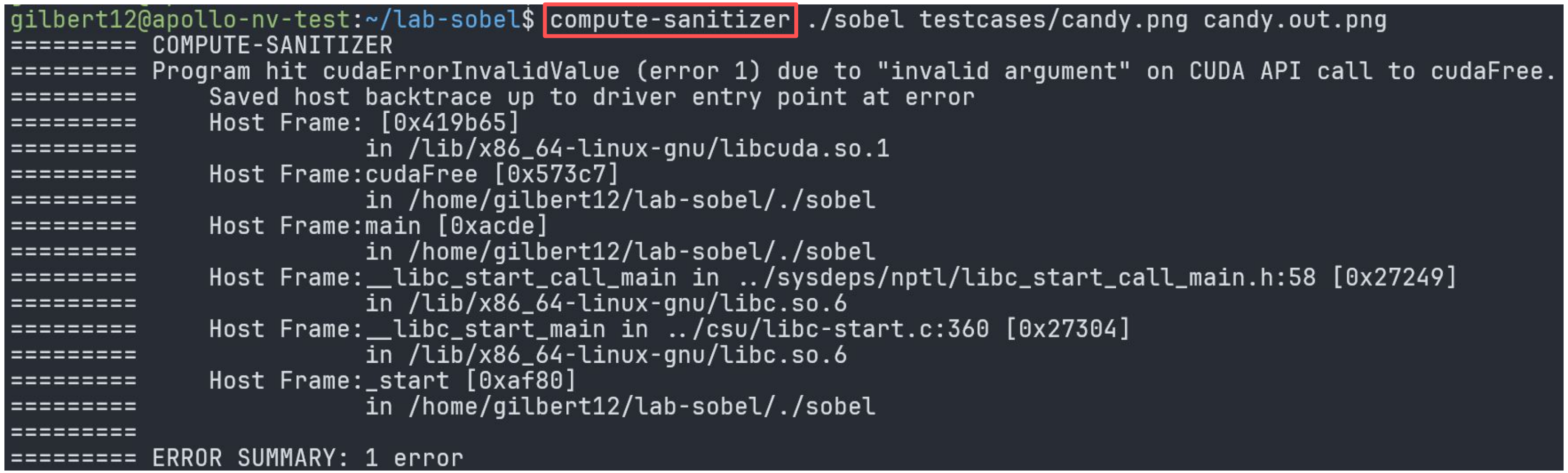
- Tutorial
  - Compute-sanitizer
  - Error-type

- Module
  - module load nvhpc-nompi/24.9

compute-sanitizer



```
cudaFree(device_t);
cudaFree(device_t); // free an address twice, error
```



```
gilbert12@apollo-nv-test:~/lab-sobel$ compute-sanitizer ./sobel testcases/candy.png candy.out.png
========= COMPUTE-SANITIZER
========= Program hit cudaErrorInvalidValue (error 1) due to "invalid argument" on CUDA API call to cudaFree.
=========     Saved host backtrace up to driver entry point at error
=========     Host Frame: [0x419b65]
=========                in /lib/x86_64-linux-gnu/libcuda.so.1
=========     Host Frame:cudaFree [0x573c7]
=========                in /home/gilbert12/lab-sobel/./sobel
=========     Host Frame:main [0xacde]
=========                in /home/gilbert12/lab-sobel/./sobel
=========     Host Frame:__libc_start_call_main in ../sysdeps/nptl/libc_start_call_main.h:58 [0x27249]
=========                in /lib/x86_64-linux-gnu/libc.so.6
=========     Host Frame:__libc_start_main in ../csu/libc-start.c:360 [0x27304]
=========                in /lib/x86_64-linux-gnu/libc.so.6
=========     Host Frame:_start [0xaf80]
=========                in /home/gilbert12/lab-sobel/./sobel
=========
========= ERROR SUMMARY: 1 error
```

cuda-gdb

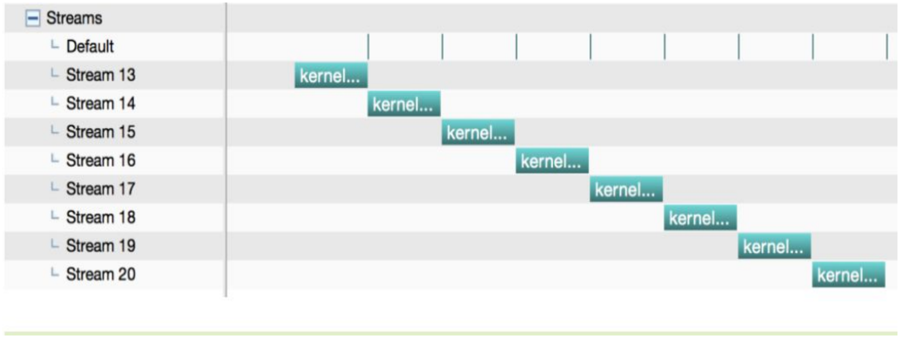
- cuda-gdb tutorial
- Module
  - module load nvhpc-nompi/24.9

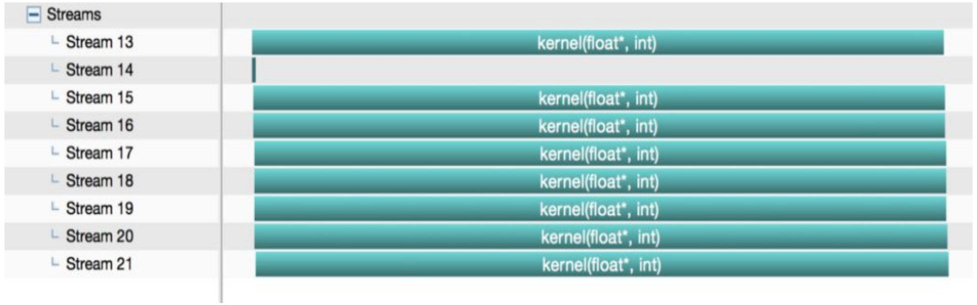
nvprof

- nvprof provide you feedback about how to optimize CUDA programs
  - `nvprof <CUDA executable>`
  - `-o <FILE>` to save result to a file
  - `-i <FILE>` to read result from a file

nvvp

- nvvp-tutorial
- GUI version of nvprof
- Useful for the stream optimization
  - Timeline





nvvp is useful for checking the concurrency of stream

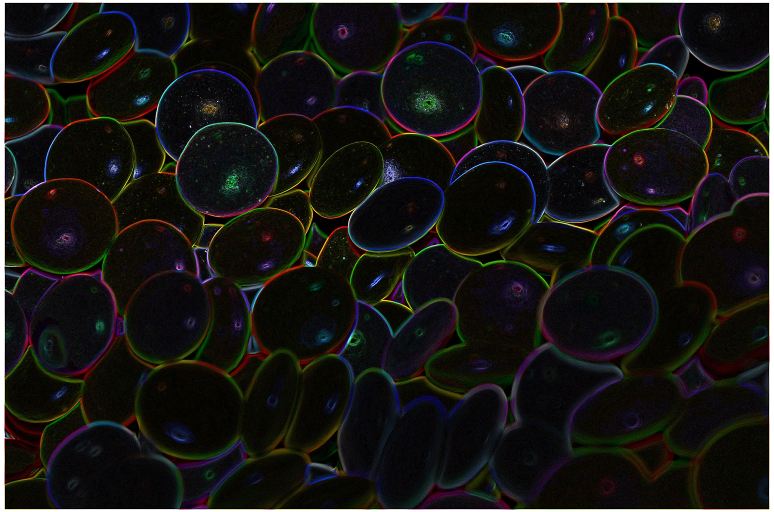
NSight Systems (nsys)

- nsys-tutorial
- Module
  - module load nvhpc-nompi/24.9
- Usage
  - nsys profile -t cuda …

Lab3 Assignment

Problem Description

- Edge Detection: Identifying points in a digital image at which the image brightness changes sharply



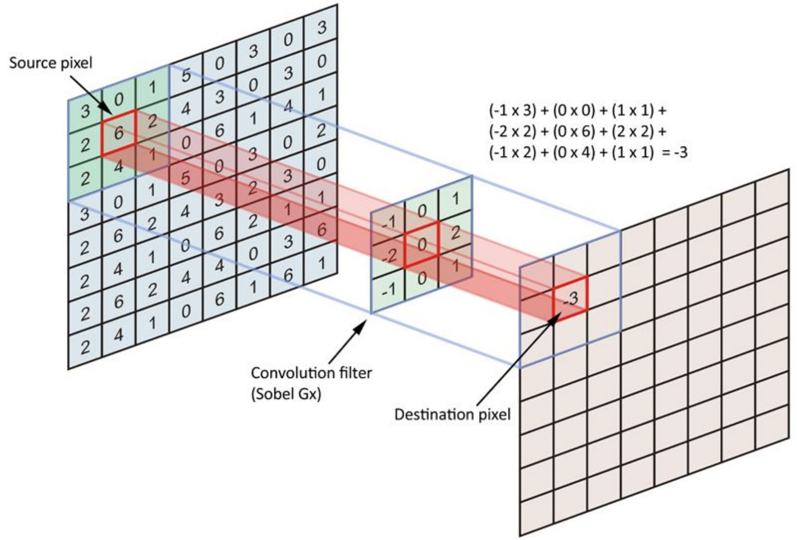
Sobel Operator

- Used in image processing and computer vision, particularly within **edge detection algorithms**.
- Uses two **3x3** **filter matrix gx, gy** which are **convolved with the original image** to calculate approximations of the derivatives - one for horizontal changes, and one for vertical.
- In this lab, we use **5x5 kernels**

$$g_x = \begin{pmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 6 & 12 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{pmatrix}, \qquad g_y = \begin{pmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

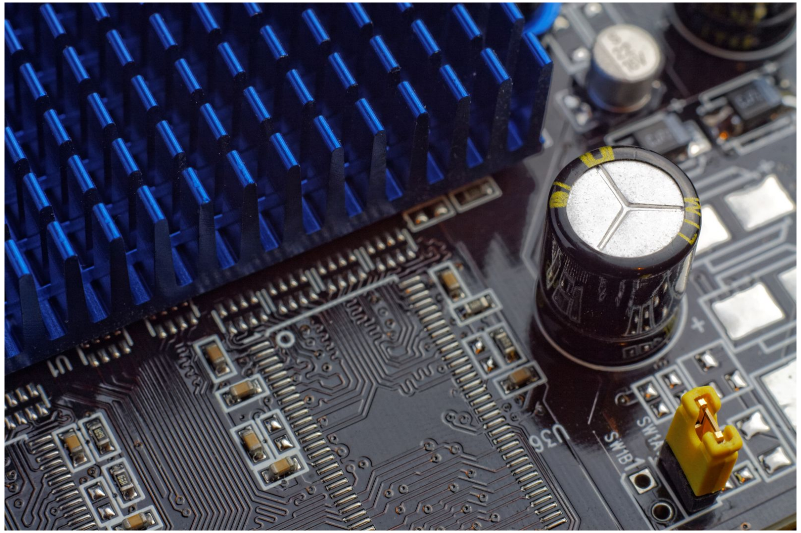
Convolution Calculation

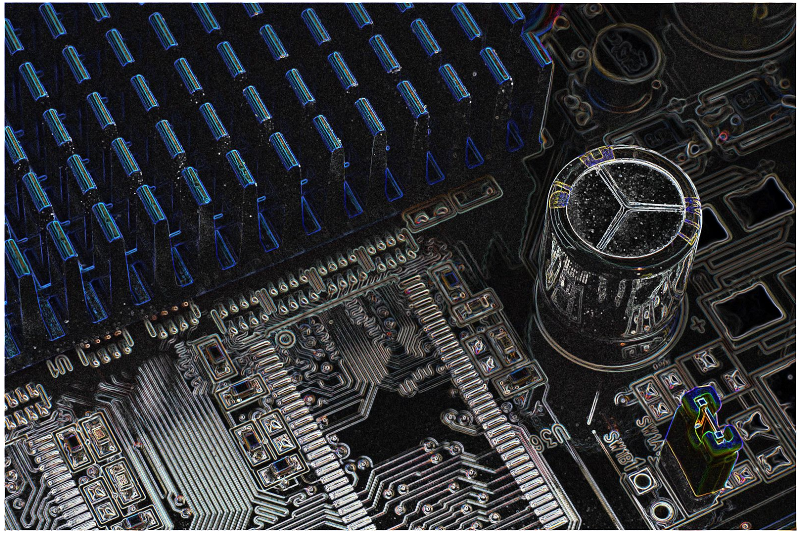
- Iterate through the width and height of the image
- For each pixel, multiply the filter matrix with original image element-wisely and sum them up.


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Credit: https://soubhihadri.medium.com/image-processing-best-practices-c-part-2-c0988b2d3e0c

Sample Result





Preparation

- ❖ TA provided CPU version, Makefile, and hint
- ❖ Files are located at `/home/pp25/share/lab-sobel`
- ❖ Please do not modify the test cases
- ❖ `sobel.cu` is cpu version (you need to rewrite it with cuda!)
- ❖ Follow hints
- ❖ After you finish the code using CUDA, try it on AMD GPU

Workflow

1. cp -r /home/pp25/share/lab-sobel $HOME
2. module load rocm cuda
3. Finish the hints
4. Compile the program : make sobel
5. Run the program : srun --gres=gpu:1 ./sobel testcases/candy.png candy.out.png
6. Check the diff : png-diff testcases/candy.out.png candy.out.png
7. Judge: lab-sobel-judge
8. Scoreboard: sobel

Workflow for AMD GPU (Optional)

We ask you to test your code on AMD GPU as well.

No need to rewrite the code, just use “Hipify”

1. `module load rocm cuda`
2. Hipify your CUDA code: `hipify-clang sobel.cu`

   Generates `sobel.cu.hip`
3. Inspect the code and learn how HIP works
4. Rename the file to sobel.hip
5. Compile the program

   `make sobel-amd`
6. Run : srun -p amd ./sobel-amd testcases-amd/candy.png candy.out.png
7. Judge: lab-sobel-amd-judge
8. Scoreboard: sobel-amd

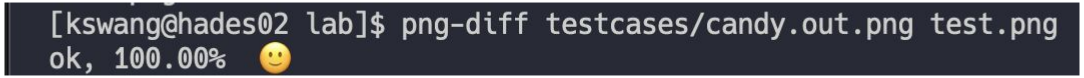
How to run

- apollo-nv-test
  - `./sobel <input> <output>`
  - `CUDA_VISIBLE_DEVICES=0 ./sobel <input> <output>`
- apollo-nv-vm[1-7]
  - `srun -n 1 --gres=gpu:1 ./sobel <input> <output>`

- apollo-amd-vm
  - `srun -p amd -n 1 --gres=gpu:1 ./sobel-hip <input> <output>`

Check the correctness

- `png-diff <result_file> <answer_file>`
  - It verifies the correctness of your output result
  - `result_file` is the output file from your CUDA program.
  - `answer_file` is the provided file for correctness checking.
    - If your input_file is "`~/lab_sobel/testcases/candy.png`", your answer_file is "`~/lab_sobel/testcases/candy.out.png`"



```
[kswang@hades02 lab]$ png-diff testcases/candy.out.png test.png
ok, 100.00% 🙂
```

  - Your code is correct if you see "`ok, 100.00%`"

Hints

- Malloc memory on GPU
- Copy the original image to GPU
- Put filter matrix on device memory (or declare it on device)
- Parallelize the sobel computing
- Copy the results from device to host
- Free unused address

Submission

- Judge will execute your code with single process, single GPU
- Submit your code and Makefile (optional) to eeclass before 11/6 23:59
- Use **lab-sobel-judge** to judge
- Score Board: lab-sobel
- Get started as soon as possible to avoid heavy queueing delay

- sobel.cu
- Makefile (Optional)