A report about assessment of GOODLAB

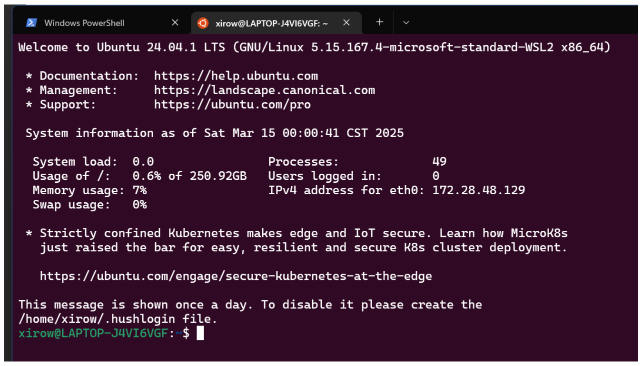
Liwei Huang

I. Task one:

***Index Terms*—Config & Kernel Development**

A. Environment config:

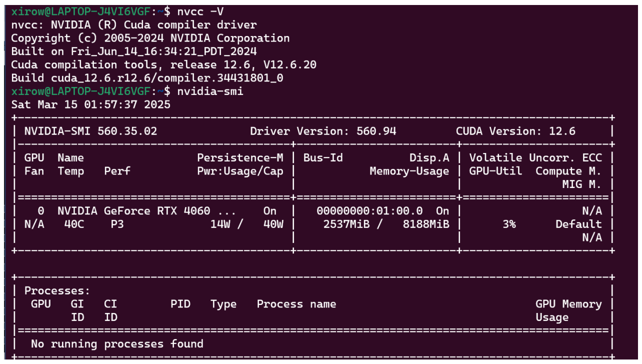
1) : Use WSL2 to deploy Ubuntu
One of the advantages of using WSL2 is The Ubuntu could connect directly the graphic card of ur Windows host!



```
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sat Mar 15 00:00:41 CST 2025

  System load:  0.0               Processes:             49
  Usage of /:   0.6% of 250.92GB  Users logged in:       0
  Memory usage: 7%                IPv4 address for eth0: 172.28.48.129
  Swap usage:   0%

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

This message is shown once a day. To disable it please create the
/home/xirow/.hushlogin file.
xirow@LAPTOP-J4VI6VGF:~$
```

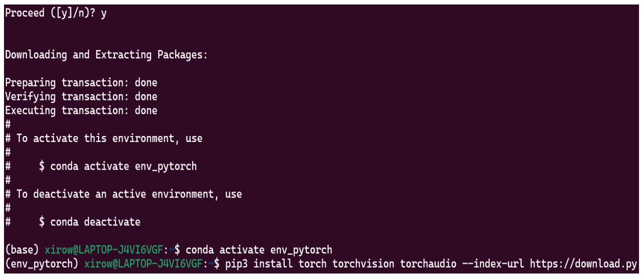
2) : Deploy CUDA Toolkit and Pytorch
search the method of installing CUDA and PyTorch in Ubuntu and configure the environment.

Firstly, employing the CUDA:



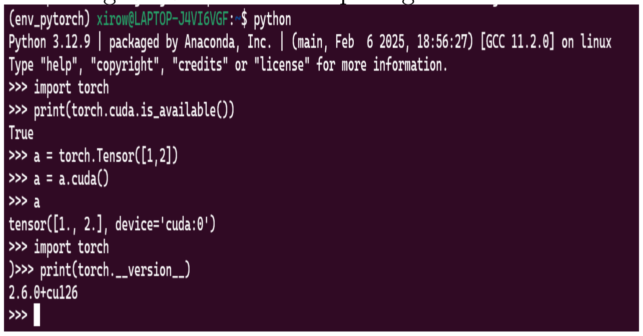
```
xirow@LAPTOP-J4VI6VGF:~$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Fri_Jun_14_16:34:21_PDT_2024
Cuda compilation tools, release 12.6, V12.6.20
Build cuda_12.6.r12.6/compiler.34431801_0
xirow@LAPTOP-J4VI6VGF:~$ nvidia-smi
Sat Mar 15 01:57:37 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 560.35.02              Driver Version: 560.94         CUDA Version: 12.6     |
|-----------------------------------------+------------------------+----------------------+
| 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  NVIDIA GeForce RTX 4060 ...    On  |   00000000:01:00.0  On |                  N/A |
| N/A   40C    P3             14W /   40W |    2537MiB /   8188MiB |      3%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

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

Secondly, configuring the conda environment. I decide to use it to carry my python and pytorch environment for its advantages: 1.isolating dependencies 2.simplifying manage of dependencies 3.Supporting Cross-platform



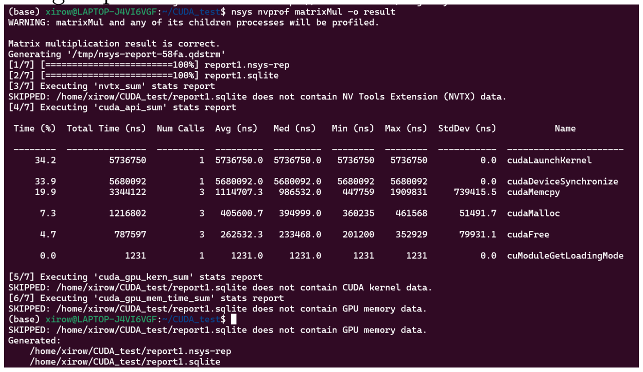
```
Proceed ([y]/n)? y


Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate env_pytorch
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) xirow@LAPTOP-J4VI6VGF:~$ conda activate env_pytorch
(env_pytorch) xirow@LAPTOP-J4VI6VGF:~$ pip3 install torch torchvision torchaudio --index-url https://download.py
```

Then installing the PyTorch Frame:
Detecting GPU and the computing of cuda!



```
(env_pytorch) xirow@LAPTOP-J4VI6VGF:~$ python
Python 3.12.9 | packaged by Anaconda, Inc. | (main, Feb  6 2025, 18:56:27) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.cuda.is_available())
True
>>> a = torch.Tensor([1,2])
>>> a = a.cuda()
>>> a
tensor([1., 2.], device='cuda:0')
>>> import torch
)>>> print(torch.__version__)
2.6.0+cu126
>>>
```

3) : GPU computing power
Verify the availability of the GPU computing power by using nvprof.



```
(base) xirow@LAPTOP-J4VI6VGF:~/CUDA_test$ nsys nvprof matrixMul -o result
WARNING: matrixMul and any of its children processes will be profiled.

Matrix multiplication result is correct.
Generating '/tmp/nsys-report-58fa.qdstrm'
[1/7] [========================100%] report1.nsys-rep
[2/7] [========================100%] report1.sqlite
[3/7] Executing 'nvtx_sum' stats report
SKIPPED: /home/xirow/CUDA_test/report1.sqlite does not contain NV Tools Extension (NVTX) data.
[4/7] Executing 'cuda_api_sum' stats report

 Time (%)  Total Time (ns)  Num Calls  Avg (ns)   Med (ns)   Min (ns)  Max (ns)  StdDev (ns)           Name

 --------  ---------------  ---------  ---------  ---------  --------  --------  -----------  ----------------------
     34.2          5736750          1  5736750.0  5736750.0   5736750   5736750          0.0  cudaLaunchKernel

     33.9          5680092          1  5680092.0  5680092.0   5680092   5680092          0.0  cudaDeviceSynchronize
     19.9          3344122          3  1114707.3   986532.0    447759   1909831     739415.5  cudaMemcpy

      7.3          1216802          3   405600.7   394999.0    360235    461568      51491.7  cudaMalloc

      4.7           787597          3   262532.3   233468.0    201200    352929      79931.1  cudaFree

      0.0             1231          1     1231.0     1231.0      1231      1231          0.0  cuModuleGetLoadingMode

[5/7] Executing 'cuda_gpu_kern_sum' stats report
SKIPPED: /home/xirow/CUDA_test/report1.sqlite does not contain CUDA kernel data.
[6/7] Executing 'cuda_gpu_mem_time_sum' stats report
SKIPPED: /home/xirow/CUDA_test/report1.sqlite does not contain GPU memory data.
(base) xirow@LAPTOP-J4VI6VGF:~/CUDA_test$
SKIPPED: /home/xirow/CUDA_test/report1.sqlite does not contain GPU memory data.
Generated:
    /home/xirow/CUDA_test/report1.nsys-rep
    /home/xirow/CUDA_test/report1.sqlite
```

B. Exploit CUDA kernel:

1) : To pave the way for exploiting kernel

Firstly
CUDA program:
I had a general understanding of cuda programming through study a course of it, which made me have a moderate command of cuda, such as: kernel function,memory model of cuda,thread-block-warp, resource allocation,error check...

Secondly
LayerNorm
LayerNorm are always broken apart into two parts which is normalization and then by layers:
When training for neural net,we often use LayNorm to avoid gradient disappearance or gradient explosion. LayerNorm could encapsulate those values produced by neurons within a much small range,typically centered around zero. What this allow for is much more stable training as we actually perform a gradient step!
And every neuron in every layer is normalized such that all the activation will have a center like zero or standard deviation of one!
The following is the formula to achieve LayerNorm:

$$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ is a small value to avoid zero appearing in denominator simplifying:

$$\hat{x_i} = \frac{x_i - \mu}{\sigma}$$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

2) : process of exploiting kernel

Firstly
Using vector float4 to access memory:
Through reading four floats one times to reduce times of accessing memory so that it can improve efficiency. It also takes advantage of the power of parallel computing better!

Secondly
Using Warp Reduce:
It is such an efficient optimizing technology which can reduce the computing complexity.Because a cuda core computes in parallel in warps which contain 32 threads.So using some instructions to make the thread execute computing tasks simultaneously.In that case,We can apply Warp Reduce to every adding calculation!

Thirdly
Supporting Dynamic shapes automatic adaptation:
Calculating the size of grid and block dynamically. Parameterize the kernel using templates or macros. Dynamic Parallelism.

Fourthly
Realizing Shared-memory Double Buffering system to avoid Bank Conflict:
The core idea of it is using two shared memory areas,one of which is used to read another is used to write. So that when a buffer is being used, another can load and store to improve efficiency!

The whole code is on my github repository!

II. Task two:

***Index Terms*—Compare & Analyze the performance of operator**

A. Calculating TFLOPS:

1) : definition:
”Tera Floating Point Operations Per Second”
TFLOPS is a vital index of assessing the performance of computing device when disposing complex calculating tasks, especially in deep learning field where vast calculations happen.

B. Global Load/Store Efficiency:

1) : definition:
Global Load/Store Efficiency measures how efficiently a CUDA kernel accesses global memory. It is an important metric in performance tuning because global memory accesses are significantly slower than shared memory or registers

C. Register/shared memory usage optimization:

1) : definition:
Registers are the fastest memory on a GPU, located inside each Streaming Multiprocessor (SM).
Shared memory is faster than global memory and shared among threads in a block.

D. Acceleration Ratio of different scales:

1) : definition:
The Acceleration Ratio depends on various factors, including the specific application, workload characteristics, and how effectively the software utilizes the GPU’s architecture.

III. Problems & Solutions

A.

FIRST of all,
I had no idea about all of the terms in the Task.pdf,which really troubled me! However,with my determination and ambition for new knowledge,I decided to get a general understanding of these new terms firstly to have a overview of my following tasks. Then I used search engine to accomplish above-mentioned things.

B. TaskOne

PB1: Before the kernel development, I prefered to have a rough command of CUDA and programming on it through bilibili and csdn.

PB2: However,when I decided to use the "nvprof" instruction,I finded out that it was replaced by some new tools NVIDIA Nsight Systems and NVIDIA Nsight Compute. When I struggled to install and configure NVIDIA Nsight Systems and Compute, its all-English UI really troubled me. So I have to learn how to use this tool to verify the availability of GPU computing power.
It will take some time,but I am interested in it!
Solution to PB2: When I read the help document of Nsys ,I was glad to find out that the instruction "nvprof" could still use by using nsys nvprof!

PB3: when I used the Nsight System to analyze the cuda program,it will always report an error when I use nsys-ui which confused me a lot!!
Solution to PB3: Through reading documents and referring information,I realized that error could be ignored which would not impact my analysis on Nsight!

PB4 and Solution: optimizing the cuda kernel was so difficult which made me have to turn to AI and numerous videos to understand them step by step statement by statement.And I could not take a good command of them within a short days!

C. TaskTwo

PB1 and Solution: I haven't learn the proper nouns of operator performance, so I need to take a general comprehension of them.

PB2: I couldn't find the TFLOPS data and some relevant data on Nsight Compute although I had used Nsight to analyze my CUDA kernel program.

PB3: I tried to use python to write test code that contains native PyTorch LayerNorm and custom CUDA kernel But some unknown and difficult errors really confused me, and I think all of those confusion and errors couldn't be resolved in a short period.