High Performance Computing

2023 Fall

Lab 9. Application Optimization

Yongzhuo Ma

October 14, 2023

Chapter 1

GPU DGEMM

In this chapter, I will design a C script of DGEMM on NVIDIA GPU.

1.1 Introduction to the Environment

1.1.1 Host Machine

Item	Value
OS version	Windows 11 22H2 (22621.2428)
CPU	Intel Core i9-12900KS
CPU Frequency	8xP 5.5 GHz, 4xE 5.2 GHz
Physical CPU Cores	16
Logical CPU Cores	24
Memory	128 GB
Memory Bandwidth	76.8 GB/s
GPU0	NVIDIA RTX A6000
GPU1	NVIDIA RTX A6000
VRAM	96 GB
VRAM Bandwidth	768 GB/s
Architecture	x86_64

1.1.2 Virtual Machine

Item	Value
Virtualization	WSL VT-x
OS version	Ubuntu 22.04.3 LTS
gcc version	11.4.0

1.1.3 Dependencies

Item	Version
CUDA	12.2.2
nvidia cuda toolkit	11.5.1

1.2 Prepare relevant Environment

DGEMM on GPU is based on CUDA. CUDA has the ability to manage VRAM, and do different types of calculation.

1.2.1 Disable nouveau

Nouveau will cause problems when installing CUDA, it is highly possible to crash the gnome desktop. However, if the linux system is not packed with GUI, this step could be left out.
To disable we can run these in terminal:

```
sudo vi /etc/modprobe.d/blacklist.conf
//Add disable message on the end:
blacklist nouveau
options nouveau modeset=0 //:wq
sudo update-initramfs -u
sudo reboot
lsmod | grep nouveau
```

Nothing should be returned.

1.2.2 Update & Download Packages

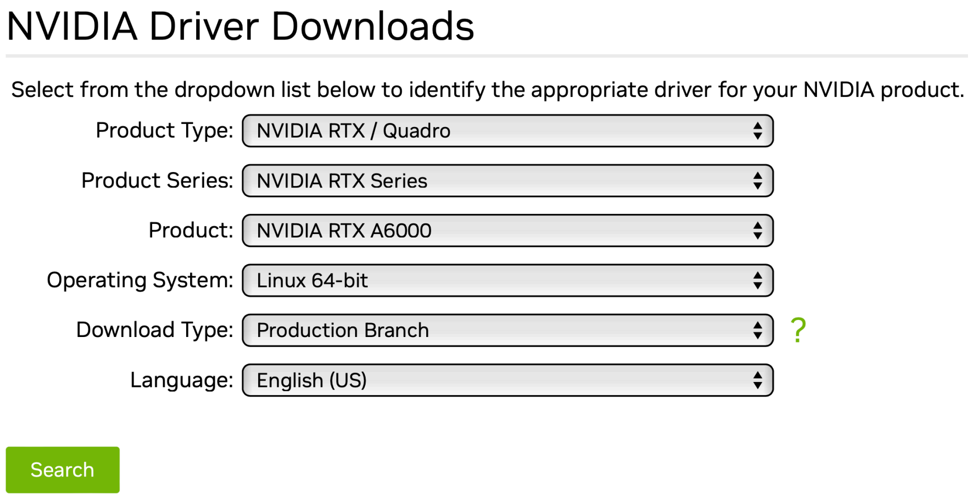
This step aims at avoiding compatible problems.

```
sudo apt update
sudo apt install build-essential
```

1.2.3 GPU driver runfile

According to different hardware, it is possible to download GPU driver runfile from NVIDIA. In linux, we can use wget to download the correct runfile.

```
wget https://us.download.nvidia.com/XFree86/Linux-
  x86_64/535.113.01/NVIDIA-Linux-x86_64-535.113.01.run
```


NVIDIA Driver Downloads

Select from the dropdown list below to identify the appropriate driver for your NVIDIA product.

Product Type:	NVIDIA RTX / Quadro
Product Series:	NVIDIA RTX Series
Product:	NVIDIA RTX A6000
Operating System:	Linux 64-bit
Download Type:	Production Branch
Language:	English (US)

1.2.4 CUDA runfile

According to different hardware, it is possible to download CUDA runfile from NVIDIA Developer website. In linux, we can use wget to download the runfile.

```
wget https://developer.download.nvidia.com/compute/cuda
  /12.2.2/local_installers/cuda_12.2.2_535.104.05_linux.run
```

1.3 Installation

1.3.1 Install driver and CUDA

```
sudo sh NVIDIA-Linux-x86_64-535.113.01.run
sudo sh cuda_12.2.2_535.104.05_linux.run
```

Wait for extracting. Do not install the driver in CUDA installation wizard. Just install 3 or 4 CUDA folders.
After Installation, terminal will return the path.

```
1 ===========
2 = Summary =
3 ===========
4 Driver: Not Selected
5 Toolkit: Installed in /usr/local/cuda-12.2/
6 Please make sure that
7 - PATH includes /usr/local/cuda-12.2/bin
8 - LD_LIBRARY_PATH includes /usr/local/cuda-12.2/lib64, or, add
   /usr/local/cuda-12.2/lib64 to /etc/ld.so.conf and run ldconfig as
   root
9 To uninstall the CUDA Toolkit, run cuda-uninstaller in
   /usr/local/cuda-12.2/bin
10 ***WARNING: Incomplete installation! This installation did not
   install the CUDA Driver. A driver of version at least 535.00 is
   required for CUDA 12.2 functionality to work.
11 To install the driver using this installer, run the following command,
   replacing <CudaInstaller> with the name of this run file:
12 sudo <CudaInstaller>.run –silent –driver
13 Logfile is /var/log/cuda-installer.log
```

Mind line 7 and 8. Add two lines to **/.bashrc**
export PATH=$PATH:/usr/local/cuda-12.2/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-12.2/lib64
Then **source /.bashrc**.

1.3.2 Last work

```
nvcc -V
nvidia-smi
```

Using these 2 lines, you will know if they are correctly installed. If installed correctly, install toolkit package.

```
sudo apt install nvidia-cuda-toolkit
```

Now, all preparation has done.

1.4 Code

DGEMM on GPU is like DGEMM using MPI, the difference is that MPI sends the data between one CPU core to another CPU core while CUDA sends the data between CPU and GPU. Similarly, we just need to prepare the memory to be used in GPU, and receive/send the matrices.

This line prepares **a** with space of m*k double number in GPU:

```
cudaMalloc((void**)&a, m*k*sizeof(double));
```

This line collects matrix from **A**(Host) to **a**(GPU):

```
cudaMemcpy(a, A, m*k*sizeof(double),cudaMemcpyHostToDevice);
```

This line frees VRAM of **a**.

```
cudaFree(a);
```

When all matrices have been loaded to GPU, we should design:
how to divide the task to the CUDA cores.
struct dim3 is a data structure used to represent a three-dimensional grid or block configuration for launching kernels on a GPU . It is a structure

defined in the CUDA runtime and API. More detailed:

```
struct dim3 {
unsigned int x, y, z;
};
```

x, y, and z are dimensions in the x, y, and z directions. So we can define blocks and grids using dim3 to perform DGEMM.
LENB is defined as macro 16, representing the size of a block. This value shows best performance based on testing.

```
dim3 blockSize(LENB, LENB);
dim3 gridSize((n + LENB - 1) / LENB, (m + LENB - 1) / LENB);
```

This means:
Each block is LENB×LENB, and we break the matrix into

$$\frac{n + LENB - 1}{LENB} \times \frac{m + LENB - 1}{LENB}$$

blocks to cover all matrix parts.

Finally, we design the multiplication Part

```
__global__ void My_MMult(double *A, double *B, double *C, int m,
  int n, int k) {
int row = blockIdx.y * blockDim.y + threadIdx.y;
int col = blockIdx.x * blockDim.x + threadIdx.x;
if (row < m && col < n) {
    double t = C[row * n + col];
    for (int i = 0; i < k; ++i)
        t += A[row * k + i] * B[i * n + col];
    C[row * n + col] = t;
}
}
```

__global__ means this function will be used as kernel function for GPU.

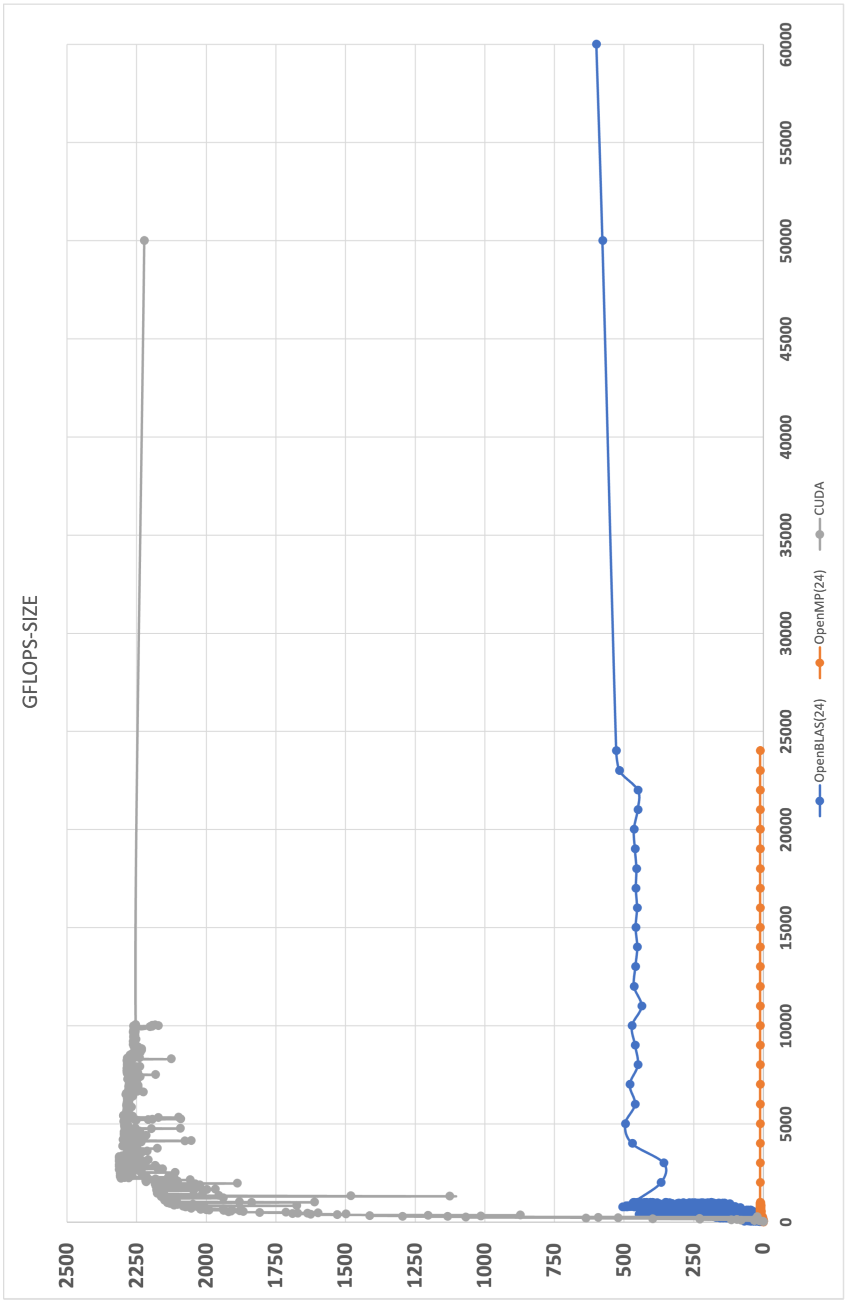
To avoid false sharing, we will have a new variable t to save the results Line 2 and line 3 tell GPU cores where they are now (which grid and which block).

By the way, when getting the time, we should force the program to wait for the end of calculating. We need cudaDeviceSynchronize().

```
double st= TIME();
My_MMult<<<gridSize, blockSize>>>(a, b, c, m, n, k);
cudaDeviceSynchronize();
double et = TIME();
```

Here <<< ... >>> means activating the kernel function using the paremeters '...'.

1.5 Performance Comparison



This graph shows that DGEMM on GPU with CUDA performs extremely efficient, rated at about 2.25 TFlops. while OpenBLAS was rated at 500 to 600 GFlops. 200 times faster than OpenMP (11GFlops). By the way, RTX A6000 has only 64 FP64 cores.
SGEMM on RTX A6000 reaches 68.28TFlops.

Chapter 2

HPL on GPU

In fact, HPL on GPU is similar to HPL on CPU.

2.1 Introduction to the Environment

Host Machine

Item	Value
OS version	Ubuntu 22.04.3 LTS
CPU	Atris Cling G8031
CPU Cores	2048
Memory	512 TB
GPU	8 x 2 x NVIDIA H100 NVL
VRAM	1.46 TB
VRAM Bandwidth	7.80 TB/s

Due to technical reasons, descriptions here are not detailed. This machine may use a different file system to manage file. And the information on the machine is not detailed, so this test can just act as a reference.

2.2 Prepare the Environment

2.2.1 Driver and CUDA

Refer to Page 3 to 6, we still need NVIDIA driver and CUDA.

2.2.2 HPL pack

Get the standard HPL pack from NVIDIA. You can easily find it online. You can get CUDA Accelerated Linpack, or the container file (but we need to do it ourself, so we can extract that from the container).

https://developer.nvidia.com/computeworks-developer-exclusive-downloads
or
https://catalog.ngc.nvidia.com/orgs/nvidia/containers/hpc-benchmarks

2.2.3 MKI and OpenMPI

We need OpenMPI, Intel MKL Library to perform HPL on GPU. We just need to find the position of the packages and add them to these parameters if needed.

```
MPdir = 'Your MPdir'
MPinc = -I$(MPdir)/include
MPlib = $(MPdir)/lib/libmpi.so
LAdir = 'Your LAdir'
LAlib = 'Your LAlib'
```

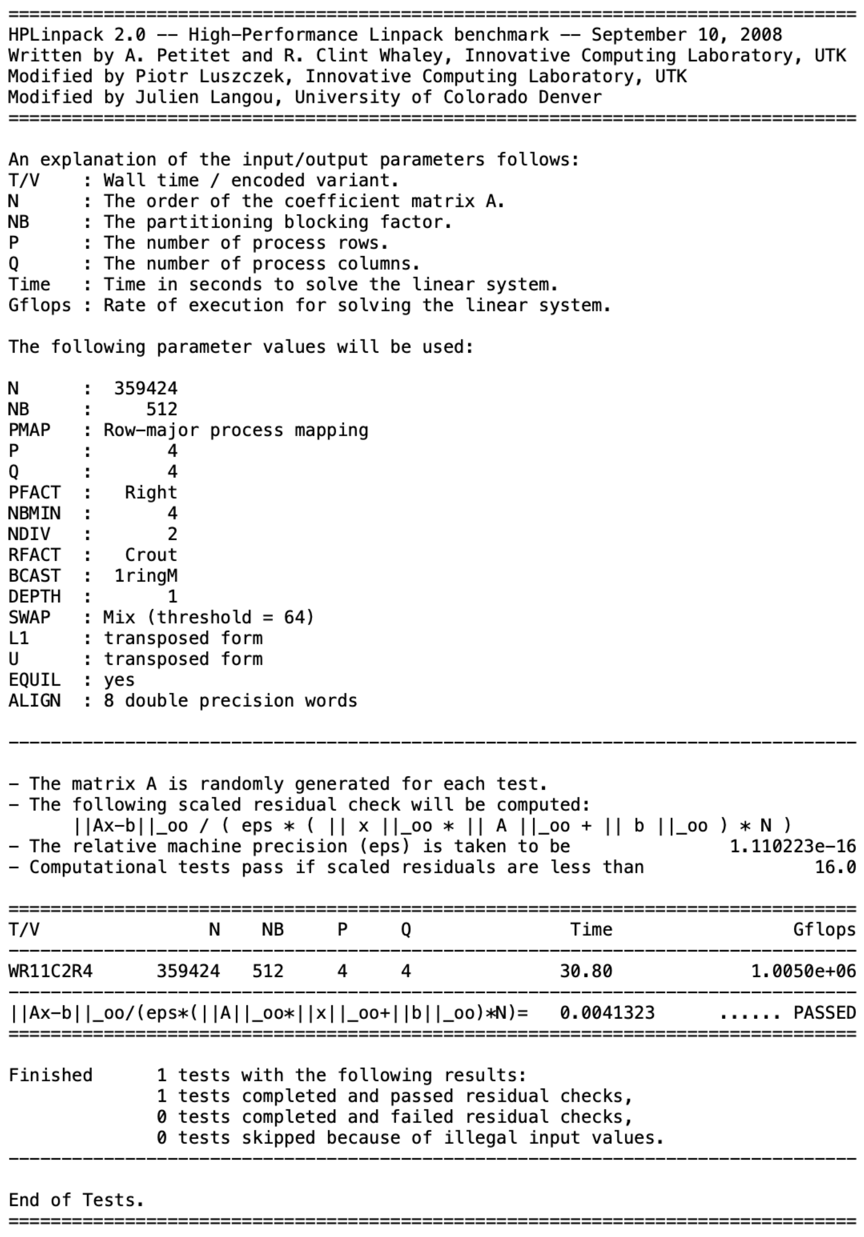
And then we compile HPL just like that on CPU. Tuning parts are almost the same, NB is 512.

Results are below.
The result can be seen as 512.131 TFlops.
By starting Tensor Cores, the performance was 1.005 PFlops.

2.3 Normal

```
================================================================================
HPLinpack 2.0 -- High-Performance Linpack benchmark -- September 10, 2008
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :  359424
NB     :     512
PMAP   : Row-major process mapping
P      :       4
Q      :       4
PFACT  :   Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be               1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0

================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4      359424   512     4     4              60.44              5.1213e+05
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   0.0039872      ...... PASSED
================================================================================

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```

2.4 Tensor cores



```
================================================================================
HPLinpack 2.0 -- High-Performance Linpack benchmark -- September 10, 2008
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :  359424
NB     :     512
PMAP   : Row-major process mapping
P      :       4
Q      :       4
PFACT  :   Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be               1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0

================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4      359424   512     4     4              30.80             1.0050e+06
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   0.0041323      ...... PASSED
================================================================================

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```