

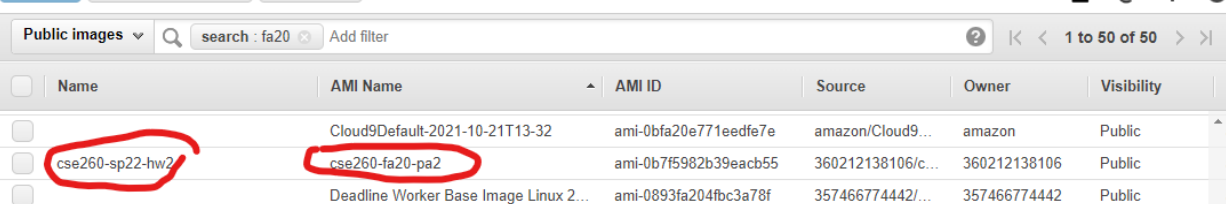
# Assignment #2 - 9th November : 11:59 pm

## Checkpoint due - 26th October : 11:59 pm

NOTE - as with assignment #1 you will work in a team of 2. If you need to do this assignment alone, you may ask for a waiver. You should not have the same partner as in Assignment #1 (for exceptional cases, such as fulltime work, etc, please contact the course staff).

### Part 1. The Assignment

In this assignment, you will optimize matrix multiplication for NVIDIA's Turing GPU. You'll measure and report observed performance on AWS g4dn.xlarge. **\*IMPORTANT (UP ON OUR SOAPBOX)\* - DO NOT LEAVE YOUR INSTANCE RUNNING. IDLE INSTANCES ARE CHARGED AT THE SAME RATE AS ONES THAT ARE ACTIVELY COMPUTING. GET USED TO EDITING OFFLINE and FIRE UP THE INSTANCE WHEN YOU NEED TO RUN SOMETHING. YOU CAN EASILY EXHAUST YOUR AWS CREDITS IF YOU LEAVE THE INSTANCE RUNNING\* (getting off the soapbox now).** The AMI name for creating the g4dn.xlarge instance on Amazon EC2 is cse260-sp22-hw2 (note that says sp22 not fa22).



<input type="checkbox"/>	Name	AMI Name	AMI ID	Source	Owner	Visibility
<input type="checkbox"/>		Cloud9Default-2021-10-21T13-32	ami-0bfa20e771eedfe7e	amazon/Cloud9...	amazon	Public
<input type="checkbox"/>	<b>cse260-sp22-hw2</b>	<b>cse260-fa20-pa2</b>	ami-0b7f5982b39eacb55	360212138106/c...	360212138106	Public
<input type="checkbox"/>		Deadline Worker Base Image Linux 2...	ami-0893fa204fbc3a78f	357466774442/...	357466774442	Public

The assignment includes a report (5-10 pages long, including figures), which is an important part of the assignment. The requirements for your report are described later in this document.

Team name convention: pa2-userID-userID

## Part 2. Getting the Code

As in assignment #1, we have provided some starter code. Enter this URL and git hub classroom will create a repo for you populated with stater code:

<https://classroom.github.com/a/tSC0xUfi>

You must **not** have the same partner as in assignment #1. Similar to assignment #1, the assignment will be graded based on 4 factors: performance attained by your implementation, correctness, well-displayed results and comments, and explanation and insight into how you realized your reported performance. In determining your grade for the assignment, explanations and insight will be emphasized somewhat more heavily than performance itself, unless there was little improvement in performance over the provided naive code.

## Part 3. What to do

1. **Optimize the code to use on-chip memory.** You can improve performance significantly by buffering frequently accessed data in fast on-chip shared memory (Be sure to run your code without the -l flag to ensure that it favors shared memory over cache). Modify the code to use shared memory, but try other performance programming techniques to improve performance still further. Ensure that your data accesses are coalescing.
2. Your performance goal is ~1900-2100 Gigafllops/second (GF/sec) on a matrix of size 1024x1024, ~2500-3000 GF/sec 2048x2048 and about 2700-3200 or more on 16384 x 16384 (single precision). (See grading below). To realize your performance goals, explore the optimizations described in any/all of the following references:

[CUTLASS: Fast Linear Algebra in Cuda C++](#) [1], - describes CUTLASS at a high level and can achieve very high performance. You may refer to the CUTLASS code for

inspiration but DO NOT COPY. COPYING CUTLASS CODE is an academic integrity violation.

SC '08 paper by Volkov and Demmel: [\*Benchmarking GPUs to tune dense linear algebra\*](#) [2]. Since the implementation discussed in the paper was designed for a pre-Kepler device, the required optimizations (or the details, i.e. tuning parameters) will differ for Kepler.

Volkov's presentation at GTC 2010 [http://www.nvidia.com/content/GTC-2010/pdfs/2238\\_GTC2010.pdf](http://www.nvidia.com/content/GTC-2010/pdfs/2238_GTC2010.pdf) [3] describes increasing parallelism without increasing thread count. You may consider the idea of multiple outputs/thread in 2D (Volkov only expands in one dimension while the idea may apply to 2 dimensions).

3. **Verify correctness.** To build confidence in the correctness of your code, test it against different combinations of matrix size  $n$  and thread block geometry. The provided verification code, which resides in `genMatrix.cpp`, will enable you to establish correctness. Note, the driver code expects your kernel to compute  $C = A * B$  and not  $C += A * B$  as in assignment 1. So repetitive calls to the kernel will all produce the same result. As in PA1, there are several utilities in `genMatrix` and options in `cmdLine.cpp` which will help you debug for correctness.

## Part 4. The Report and Grading

You must follow the guidelines for a well-written and organized report. Full credit will be awarded only if the guidelines mentioned in the Report format document are followed.

### Performance (30 pts)

Your implementation will be graded on its performance at the following matrix sizes on single-precision FP numbers running on T4

n=256, n=512, n=1024, n=2048

You will provide us with a thread block geometry for this performance test in OPTIONS.TXT in the format

bx=\_\_\_ by=\_\_\_

The performance score for T4 is determined according to the following table. This is a general numbers rubric and liable to change. We may also test against higher matrix sizes, so please make sure that your kernel isnt optimized just for these values.

	Performance	Points
n=256	>250	1
	>400	5
n=512	>800	1
	>1300	5
n=1024	>2000	1
	>2700	5
n=2048	>2200	1
	>3000	5
-----		

Correctness (10 pts)

Report (60)

## Part 5. Requirements

- Results must be reported on AWS g4dn.xlarge instance running CUDA 11.6
- Your code must correctly handle the case where the thread blocking factor does not divide  $n$  evenly, without a significant loss of performance. More generally, it should deliver robust performance over a range of values of  $n$ . By "robust" we mean that while performance may be higher or lower for certain values of  $n$ , the peaks and dips should be the exception and not the rule. (note: we expect that performance will improve at larger matrix sizes).
- You must use single-precision floating-point numbers (32 bits).
- You must use an  $O(N^3)$  algorithm for matrix-multiply. Do not reduce the operation count with Strassen's or a similar algorithm.
- Your code modifications should be made to just four files:
  - `mytypes.h` - you may define TILE and block relationships,
  - `mmpy_kernel.cu` - the CUDA kernel that implements matrix multiplication
  - `setGrid.cu` - where you can set the block and grid configuration.
  - `OPTIONS.TXT` - where you can set the block and grid configuration.
  - **NOTE:** You should be able to meet our performance goals by modifying just these 3 files. We expect only these three files to be modified and they should work with the rest of the starter code. However, if you would like to explore higher performance you could do that with a customized `mmpy.cu` (with a different name in the final submission) and use that for parts of your report as needed. Please state that you have a custom version of another file in the report.

## Part 6. General Information

### Environment Setup

Run the following commands once after you create/launch the instance. This command will set the clock rate at which the GPU will run and make sure that the clock does not vary and give varying performance.

For g4dn.xlarge (T4):

**`sudo nvidia-smi -ac 5001,1590`**

## The Starter Code

The starter code is provided via a GitHub repository. It is a variation of the naive code from Hwu and Kirk. The naive version will perform very poorly. As discussed in lecture, it is memory bandwidth bound since it makes all accesses through global memory. Your implementation will run significantly faster. The provided code reports various information about your device (see `utils.cu:ReportDevice()`).

The starter code has runtime options to enter the thread block geometry via `-x` and `-y` flags. These will work for the naive kernel but may not work for your implementation. In your optimized code, these flags are not used. Instead, you should compile in the thread block size via the make parameters. For example, to make a 16x8 thread-block do:

**`make -C build_T4 bx=16 by=8`**

When changing the block geometry, be sure to do a "make clean" before recompiling. The make options `bx` and `by` set compile-time constants `BLOCKDIM_X` and `BLOCKDIM_Y`. You may want to derive other constants, such as `BLOCKTILE_N` and `BLOCKTILE_M` (the region computing by a thread block of `BLOCKDIM_X` and `BLOCKDIM_Y`). Modify `setGrid.cu` accordingly. You may also choose to introduce a third block dimension and you can specify this as a compile-time constant or as a default value in `mytypes.h`.

The code uses a define called `_FTYPE_` which is declared as single point floating.

To run cublas

```
make -C build_T4 cublastest=1
```

**NOTE:** We will be using OPTIONS.TXT (as in assignment #1), to take in the options for -bx and -by. Once you figure out your optimal block size, put the -bx and -by options in the OPTIONS.txt file. You should also put any compile-time options you want us to use here. (see Makefile for VARIABLES to use). You may override NVCCFLAGS if you wish, but you will have to include all the NVCCFLAGS that the Makefile normally sets up for you.

**NOTE:** If you wish to specify runtime options, such as the -D flag to set shared memory config for this kernel to 8B, you may specify this in an OPTIONS\_RUNTIME.TXT file. As an example, we expect to run: `./mmpy `cat src_todo_T4/OPTIONS_RUNTIME.txt` -n 256` to run a matrix of size 256 with your flags specified in OPTIONS\_RUNTIME.TXT. The cat command will replace whatever is in the txt file (so to run with -D flag your file will only contain -D in the first line).

## Code Development

You may build the starter code in any Cuda environment but we support AWS with cuda 11.1.

## Taking Timings

The driver program's "-r" flag enables you to set the repetition number for a kernel. Adjust the repetition count so that your program runs for about 5 seconds. Try a value of 50 to start and adjust accordingly.

Use nvprof to help gain insight on how to speed up your program.

The provided code has a wrapper around gettimeofday(). For any given combination of n and optimization (e.g. thread block size), take at least 3 timings and note the minimum (and maximum). It uses cudaThreadSynchronize() to sync the host code with the GPU kernel when taking timings.

You may see variations in timing. The K80 GPU uses DVFS, so the clock frequency may vary with the GPU temperature (note: you should set the GPU to its highest clock rate, but if it overheats, it may slow down)..

You can also use the Cuda timers by compiling with the following instead of wall clock time measured by the host with:

```
make -C build_T4 cuda_timer=1
```

Don't forget to make clean before changing this flag. See Nvidia CUDA documentation for more information on GPU timers.

## Profiling

CUDA provides profiling tools, which are discussed in the [Profiler's User's Guide](#).

The command-line tool is called nvprof. Look at the instructions for the Command line profiler in § 3 of the [Profiler User's Guide](#).

Using nvprof , you can collect performance statistics. Nvprof --query-metrics will list all the metrics. Pay particular attention to shared\_replay\_overhead, shared\_efficiency, shared\_load\_throughput, sm\_efficiency. Achieving high shared memory efficiency is a key to raising Q. We provided a script called run\_nvprof.sh which you may want to modify.

```
nvprof --system-profiling on ./mmpy -v -n 2048
```

will tell you the average GPU frequency when running your kernel. You can use this to calculate what peak performance is (use the average clock frequency).

On aws:

newer tools( I found these a little clunky to use, but these are replacing nvprof).

using nvidia nsight:

<https://docs.nvidia.com/nsight-compute/2022.2/NsightComputeCli/index.html#command-line-options-profile>



compute your code with `debug=1` (which will set the `-g` and `-G` flags)

Collect stats:

```
sudo /usr/local/cuda-11/bin/ncu -o profile --page source --print-source cuda --target-processes all  
./mmpy -r 10 -n 512
```

view with

nsight compute

<https://docs.nvidia.com/nsight-compute/NsightCompute/index.html>

`ncs-ui profile.ncu-rep`

Note: you can install nvidia tools on your local computer (even if you don't have a GPU) and view the profiles that are collected on aws.

Terminology - blocks and tiles.

The compile-time `bx` and `by` flags determine the threadblock geometry (`BLOCKDIM_X` and `BLOCKDIM_Y`). The naive code provides a one-to-one correspondence between a thread and one element of `C`. As described in the lecture, you may want each thread to be responsible for more than one data point in `C`. We use the term `BLOCKTILE` to correspond to the area computed by one threadblock. In the naive code, the `BLOCKTILE` geometry and the threadblock geometry are the same. We describe the relationship between `BLOCKTILE` and `BLOCKDIM` in `mytypes.h`. The following would be in `mytypes.h` for the naive code:

```
#define BLOCKTILE_M BLOCKDIM_Y  
#define BLOCKTILE_N BLOCKDIM_X
```

Feel free to redefine these for your kernel.

For your convenience, we have makefile variables `tm`, `tn`, `tk` which you can use to specify the ratio between `BLOCKTILE_M` and `BLOCKDIM_Y`, `BLOCKTILE_N` and `BLOCKDIM_X`, (note the mixing of `M` and `Y` and `N` and `X`) and how you want the `K` dimension to scale (recall that blocks of matrices are `... x K` and `K x ...`). You can choose to use these parameters or ignore as you wish.

## Part 7. Extra Credit

**EXTRA CREDIT points for higher T4 Performance:**

n=512	>=1500	2
n=1024	>=3100	2
n=2048	>=3500	2

maybe more points if you WOW us!

Another task - Coming soon.

## Part 8. Turning it in

When you have completed your assignment, you will submit your final versions of the code. You should be submitting periodically with comments so we may see your progress. The steps for final submission are no different.

### Short Summary:

- Source code -> Github
- Teameval.txt -> Github
- DECLARATION -> Github
- Report -> Github and gradescope

### Preparing the Repo:

1. run *git status* to see what files have changed.
2. use *git add* to stage your changes for commit.
3. use *git commit -m "FINAL SUBMISSION"* to commit your staged files into your local repository. The -m FINAL SUBMISSION lets us know that you intend this to be your final submission. If you submit again before the deadline, we will also consider the last

submission before the deadline as your final (but please also mark that with the FINAL SUBMISSION message).

4. push your changes into GitHub with *git push origin master*. For those more familiar with git, you may also be merging branches at this point. **We want the final code to be on the master branch. Don't forget this step. IF you don't push your changes, we won't see them.**
5. Surf over to your repository or clone a new copy to see if your changes are really there.

Things to watch out for:

- Watch out for (and resolve) merge conflicts before your final push!!

If you and your partner have modified files on multiple computers, you may have merge issues. This happens because your code from one repository is out of sync with your code from another and git doesn't know how to merge the files. MERGE CONFLICTS will prevent your code from compiling so be sure to resolve these before you do your final push.

Don't let these problems bite you in the end. If you add/commit/push often, this is less likely to happen.

- As a team, push just the team repository. You shouldn't have any others.
- Be careful with the 'FINAL SUBMISSION' message. You shouldn't have multiple repositories (but if you do somehow), make sure only the one you want graded as this message
- You can submit multiple FINAL SUBMISSIONS before the deadline. We will grade the latest one before the deadline.
- Late work will be accepted with a 10% deduction per 24 hours up to 72 hours. After 72 hours, we will consider the last submission before the deadline annotated as 'FINAL SUBMISSION' or the last submission before the deadline. (basically no late work after 72 hours).
- DO NOT commit binaries - just source code. PDF and XLS files are okay.

## Required Files for Submission

### Source code

Even though we will be substituting fresh copies for all but the required files, include all code files you worked with in case we need to unravel a problem with the turnin. For this reason, please include your Makefile.

*We expect that only files in src\_todo\_K80 (and src\_todo\_T4) folder have changed.*

### Text documents

Your turnin must include two important documents. **ONLY ONE SUBMISSION PER TEAM.** These documents are required and we will not report a grade for your assignment without them. A 10% penalty will be assessed in cases of missing forms, empty forms, or forms not properly filled out.

1. teameval.txt: A completed self/team evaluation. If in a team, fill out questions related to the division of labor and other aspects of your team.
2. DECLARATION - "signed" electronically by typing in your name(s) by both team members or by you if working alone and date. Signing this document indicates that you did not violate the Academic Integrity policies of cse 260. Make no other changes to this form.

### Lab Report

This will be a PDF document. The stem part of the file name (before the .pdf) should begin with A2-report and end with your repository name (e.g. A2-report-usingh.pdf) **Check this into both the git repository and gradescope. (one per team!!! Gradescope understands teams and will assign both members of the team the same grade).**

An important part of your report is your *analysis* providing insight into your results. For this reason, if you are running out of time, put more effort into the writeup and your analysis

rather than trying yet another performance enhancement. It can take time to collect, tabulate and plot results, as well as offer a reasonable explanation. We prefer a deeper analysis involving a few program optimizations over a shallower analysis over a larger number of optimizations. Spend more time discussing the optimizations that made a difference and less on those that made less of a difference. Negative results are also especially valued if you carefully document them, *with the underlying causes*, or best guesses if the causes are unclear. Use your github repo to document your development process. For example if you try an experiment, check in that code and push it to your repo so you can cite that in your report. This helps document your development processes and greatly reduces the likelihood of potential academic integrity issues.

**Do not include full code listings** in your writeup as these will be in your GitHub. Take care with how you plot your data, plotting only what's necessary to get your points across. Choose log and linear plots carefully, to take advantage of the plotting space. **You must also include any plotted data in tabular form.** Tabular data may be included electronically (in your GitHub) as a spreadsheet or other document called, e.g. Data.xlsx, Data.txt. Be sure to provide a caption for each plot or table that describes the experiment you conducted, e.g. "strong scaling."

Cite any written works or software as appropriate in the text of the report. Citations should include any software you have used that was written by someone else, or that was written by you for purposes other than this class. In such cases, you must **talk with us** (and any code co-authors) prior to getting approval.

## Warning

**Don't wait until the last minute to attempt a turnin for the first time. Try out the turnin process early on so you are sure that you are familiar with the process, and that all the required additional files are present.**

You will receive 0 points for this assignment if your code does not compile, including cases of non-compilation due to merge conflicts. If you developed your code on another machine you are responsible for proper testing, else you may lose significant credit, possibly all 10 points depending on the circumstances. As mentioned previously, you must ensure that you resolve all of your merge conflicts and that your code compiles and runs correctly IMMEDIATELY BEFORE you push to Github.

For this assignment staff will attempt to intervene manually to fix problems. However, the policy for such manual intervention is as follows. If any manual intervention is required by staff you will lose at least 2 points. Included are: incorrectly naming your Github repository, or misspelling the words FINAL SUBMISSION in the commit message. Lastly, the only files you should be pushing to GitHub are text files: source code and ASCII text files. Do not push large binaries such as executables or media files, as doing so could lead to problems such as losing the ability to push to GitHub. Staff will not intervene under these circumstances and you may be unable to complete your assignment by the deadline. If you commit binaries, you are doing so at your own risk.

## Part 9. FAQ

Be sure to frequently check piazza for any queries related to assignment #2.

## References

[1] [Andrew Kerr, Duane Merrill, Julien Demouth and John Tran , CUTLASS: Fast Linear Algebra in Cuda C++, December 2017](#)

[2] Jianyu Huang, Chenhan Yu, Robert van de Geijn, Implementing Strassen's Algorithm on NVIDIA Volta GPUs - <https://arxiv.org/pdf/1808.07984.pdf>

[3] Nvidia Cutlass github -

[https://github.com/NVIDIA/cutlass/blob/master/media/docs/efficient\\_gemm.md](https://github.com/NVIDIA/cutlass/blob/master/media/docs/efficient_gemm.md)

[3] [Volkov, Demmel, Benchmarking GPUs to Tune Dense Linear Algebra, SC2008](#)

[4] [Volkov, Better Performance at lower Occupancy, GTC2010](#)

Other documents you will be interested in are the following, which may all be accessed via NVIDIA's CUDA Toolkit documentation web pages: (we are using Cuda 11.6 with a GPU capability of 7.5)

[5] Cuda C++ programming guide - <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>

[6] Cuda documentation - <https://docs.nvidia.com/cuda/index.html>

pay special attention to Turing Tuning Guide. You will also find help for cuda-gdb and profilers (nvprof is old but still useful).

[7] AWS Description of T4 - <https://aws.amazon.com/blogs/aws/now-available-ec2-instances-g4-with-nvidia-t4-tensor-core-gpus/>

[8] Jia, Maggioni, Smith, Scarpazza, "Dissecting the NVidia Turing T4 GPU via Microbenchmarking" : <https://arxiv.org/abs/1903.07486>

Profiling and Performance Measurement