

Winter 2020

# CS 133 Lab 1

**CPU w/ OpenMP: General Matrix Multiplication (GEMM)**

**Due Date : 1/23/20 10:00AM**

## Description

Your task is to parallelize general matrix multiplication (GEMM) with OpenMP based on the sequential implementation we have provided. Specifically, for matrix multiplication  $C = A \times B$ , you will need to implement two functions using OpenMP:

```
void GemmParallel(const float a[kI][kK], const float b[kK][kJ], float c[kI][kJ]);  
void GemmParallelBlocked(const float a[kI][kK], const float b[kK][kJ], float  
c[kI][kJ]);
```

**GemmParallel:** A parallel version of the given matrix multiplication routine using OpenMP pragmas. Please note that some loop permutation might be needed to enable more concurrency.

**GemmParallelBlocked:** Blocked matrix multiplication with OpenMP. You should select the block size with the best performance when you submit this lab. You are welcome to add any other optimization to further increase the performance. Please describe every optimization you performed in the report.

Tip

- We will use m5.2xlarge instances for grading.
- You can assume the matrix sizes are multiples of 1024 (e.g.  $1024 \times 2048 * 2048 \times 4096 = 1024 \times 4096$ )

## Preparation

### Create an AWS Instance

Please refer to the tutorial slides and create an m5.2xlarge instance with a Ubuntu 18.04 AMI. Please use your AWS educate classroom account for this lab.

### Run Baseline GEMM

We have prepared the host code for you at [GitHub](https://github.com/UCLA-VAST/cs-133-20w). Log in to your instance and run the following commands:

```
git clone https://github.com/UCLA-VAST/cs-133-20w -o upstream  
cd cs-133-20w/lab1
```

```
./setup.sh
make gemm
./gemm
```

It should run without error and finish in a few seconds.

### Tips

- To resume a session in case you lose your connection, you can run `screen` after login.
- You can recover your session with `screen -DRR` if you lost your ssh connection.
- You should **stop** your instance if you are going back and resume your work in a few hours or days. Your data will be preserved but you will be charged for the [EBS storage](#) for \$0.10 per GB per month (with default settings).
- You should **terminate** your instance if you are not going to back and resume your work in days or weeks. **Data on the instance will be lost.**
- You are recommended to use **private** repos provided by [GitHub](#). **Do not put your code in a public repo.**

## Create Your Own GEMM with OpenMP

If you have successfully launched an instance and run the baseline, you can start to create your own GEMM kernel. The provided code will generate the test data and verify your results against a baseline fast GEMM implementation.

Your first task is to implement a parallel version of GEMM. You can start with the sequential version provided in `gemm.cpp`. You should edit `omp.cpp` for this task.

Your second task is to implement a parallel version of blocked GEMM. You should continue working on your results from the first task. You should edit `omp-blocked.cpp` for this task.

### Tips

- To check in your code to a **private** GitHub repo, [create a repo](#) first.

```
git branch -m upstream
git checkout -b master
git add omp.cpp omp-blocked.cpp
git commit -m "lab1: first version" # change commit message accordingly
# please replace the URL with your own URL
git remote add origin git@github.com:YourGitHubUserName/your-repo-name.git
git push -u origin master
```

- You are recommended to `git add` and `git commit` often so that you can keep track of the history and revert whenever necessary.
- If you move to a new instance, just `git clone` your repo.
- Run `make test` to re-compile and test your code.
- You can run the sequential GEMM by `make gemm && ./gemm sequential`. Be aware that this is very slow.
- If `make test` fails, it means your code produces the wrong result.
- ***Make sure your code produces correct results!***

# Submission

You need to report the performance results of your OpenMP implementation on an m5.2xlarge instance. Please express your performance in GFlops and the speedup compared with the sequential version. In particular, you need to submit a brief report which summarizes:

- The results comparing the sequential version with the two parallel version on three different problem sizes ( $1024^3$ ,  $2048^3$ , and  $4096^3$ ). If you get significantly different speedup numbers for the different sizes, please explain why.
- omp-blocked only : For the problem size  $4096^3$ , please quantify the impact of each optimization, including the block size selection and any other optimization you performed.
- omp-blocked only: For the problem size  $4096^3$ , please report the scalability of your optimized code with different number of threads, including 1, 2, 4, etc, and discuss the result. (hint: please find out how many threads are supported on m5.2xlarge).
- A discussion of your results.
- *Optional*: The challenges you faced, and how you overcame them.

You will need to submit your optimized kernel code. Please do not modify or submit the host code. Please submit to CCLE. Please verify the correctness of your kernel before submission.

Your final submission should be a tarball which contains and only contains the following files:

<Your UID>.tar.gz

└ <Your UID>

└ omp.cpp

└ omp-blocked.cpp

└ lab1-report.pdf

File lab1-report.pdf must be in PDF format. You should make the tarball by copying your lab1-report.pdf to the lab1 directory and running `make tar UID=<Your UID>`. If you made the tarball in other ways, you should put it in the lab1 directory and check by running `make check UID=<Your UID>`.

## Grading Policy

### Submission Format (10%)

Points will be deducted if your submission does not comply with the requirement. In case of missing reports, missing codes, or compilation error, you might receive 0 for this category.

### Correctness (50%)

Please check the correctness of your parallel and parallel-blocked implementation with different problem sizes, including but not limited to  $1024^3$ ,  $2048^3$ , and  $4096^3$ .

## Performance (25%)

Your performance will be evaluated based on the performance of problem size 4096<sup>3</sup>. The performance point will be added only if you have the correct result, so please prioritize the correctness over performance. Your performance will be evaluated based on the ranges of throughput (GFlops). We will set five ranges after evaluating all submissions and assign the points as follows (Ranges A+ and A++ will be defined after all the submissions are made):

- Range A++, better than Range A+ performance: 25 points + 5 points (bonus)
- Range A+, better than Range A performance: 25 points + 3 points (bonus)
- Range A GFlops [92, 121): 25 points
- Range B GFlops [63, 92): 20 points
- Range C GFlops [34, 63): 15 points
- Range D GFlops [5, 34): 10 points
- Speed up lower than range D [0, 5): 5 points
- Slowdown: 0 points

## Report (15%)

Points may be deducted if your report misses any of the sections described above.

## Academic Integrity

All work is to be done individually, and any sources of help are to be explicitly cited. Any instance of academic dishonesty will be promptly reported to the Office of the Dean of Students. Academic dishonesty, includes, but is not limited to, cheating, fabrication, plagiarism, copying code from other students or from the internet or facilitating academic misconduct. We'll use automated software to identify similar sections between different student programming assignments or against popular Internet sources.

Students are not allowed to post the lab solutions on public websites (including Github).

Late policy: Late submission will be accepted for **one day** with a 10% penalty. No late submission will be accepted after that (i.e., work that is more than two days late will lose all the points).