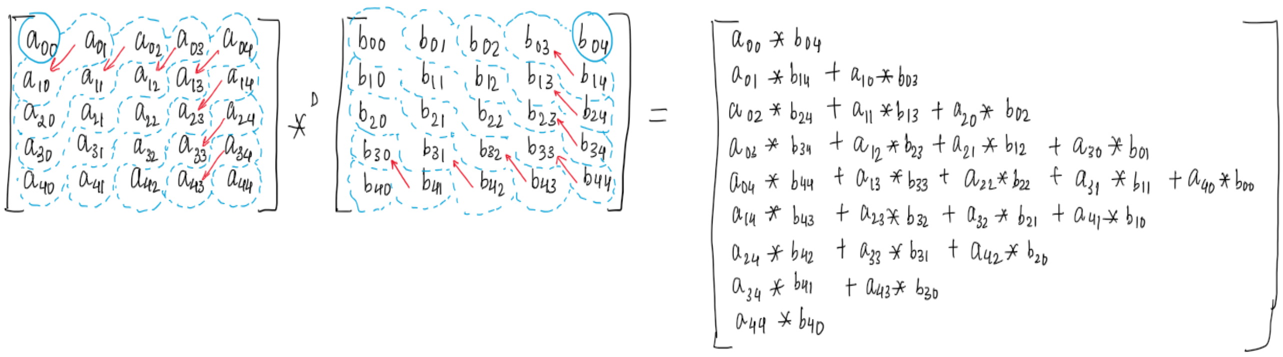
Programming assignment: Part 1
Performance optimizing Diagonal matrix multiplication (DMM)

Overview: In this assignment, your task is to implement and performance optimize "diagonal matrix multiplication (DMM)" of two n*n matrices. A sample, unoptimized single-threaded program is provided to get you started quickly. Part 1 of the assignment would be to (1) performance optimize the single-threaded programs by analyzing relevant hardware performance counter values, and (2) then parallelize the program to utilize multiple threads (using pthreads). In the second part of the programming assignment (details will be provided later), you would be asked to use CUDA to accelerate the same program on the GPU.

You will have to do the programming assignment **individually**.

Diagonal Matrix Multiplication (DMM): The sample algorithm is pictorially depicted below (a sample single-threaded unoptimized implementation is provided via gitlab—see the bottom of this document):

Input: Two n*n matrices. n = 2^k, where k is a natural number.
Output: A vector of length 2n-1



There are three major activities in this assignment, divided into two parts (PartA, PartB):

1. **[PartA-I] Optimize single-threaded DMM (CPU):** In this part, you will hand optimize the single-threaded implementation provided on gitlab. Tips: Identify performance bottlenecks in your code using hardware performance counters (e.g., via perf) and optimize the source accordingly.
   Relevant file: header/single_thread.h
2. **[PartA-II] Implement and optimize multi-threaded DMM (CPU):** In this part, you will implement a multi-threaded version of DMM. Test the scalability of your implementation by varying the number of threads and optimize it to achieve maximum scalability. In an ideal implementation, will achieve a speedup of 2x if the number of cores is increased by a factor of 2, and so on). It is mandatory to use "pthreads" for multi-threaded implementation. Do NOT use any other custom libraries.
   Relevant file: header/multi_thread.h
3. **[PartB] Implement and optimize DMM in CUDA (GPU):** In the final part, you are required to implement DMM for a GPU using CUDA. Details of this part will be available shortly.
   Relevant file: TBD

Deliverables: Submission will be in two parts—one for the CPU-based implementations (this will include both threaded and multi-threaded implementations), and the other part for GPU-based implementation (later). For each part, you are required to submit a report along with your code. Your report should contain details such as the runtime of unoptimized code and the effect of each of your optimizations on the runtime over different input size (say, n = 4k, 8k, and 16k). You should also include details such as how you identified the bottlenecks in the code (e.g., your analysis via hardware performance counters), what optimizations you have implemented, and how it helped ameliorate the bottlenecks you identified. For the multi-threaded implementation, your report must include an evaluation of the scalability of your implementation (e.g., how does the runtime change with different thread count), along with important details of your implementation.

Platform to evaluate your work: You will be given access to a server. You will receive an email from department's system admin about the account and the information on the server. However, you do not necessarily need to use that server. You can use any system you wish to as long as you do the necessary optimizations and report your analysis. For evaluation, we will use a *different* server and different input set anyway.

What/how to submit: Clone the gitlab repository on your machine and follow its README.md for instructions on how to compile and run the programs. All your code has to be implemented in "header/". DO NOT MODIFY main.cpp.
You need to submit your assignment as a zip file named as per the last five digits of your SR number (e.g., 15964.zip).
It should contain two directories – i.e., "headers/" that would contain your implementation and "reports/". Name your reports as "reports/Report_PartA.pdf" and "reports/Report_PartB.pdf".
We will create an assignment on gardescope for you to upload code.

Deadlines:
PartA: 15th January 2021, 11:59 PM
PartB: 31st January 2021, 11:59 PM

Link to GitLab repository:
https://gitlab.com/akkamath/hpca-assignment-2020-2021

Tips: A little bit about the perf tool and hardware performance counters: Modern processors come with an extensive set of performance monitoring counter. It is also called the performance monitoring unit or PMU. These counters typically count occurrences of specific events, e.g., cache miss, TLB miss, etc. Tools like Linux's perf tools provide users with an easy-to-use command-line interface to program and measure these performance counters. There are other tools like Likwid that could do the same or even more.

To see what all performance counters are exposed through perf tool, you can run the following command

```
$ perf list
```

For measuring the number of L1-cache-load misses (from user side) for a command say, sleep 1 (sleeps for 1 second) you can use the following command

```
$ perf stat -e L1-dcache-load-misses:u sleep 1
```

There are tons of information and tutorials online on perf. A few examples are https://perf.wiki.kernel.org/index.php/Tutorial and http://www.brendangregg.com/perf.html

Performance measurement tips: Real hardware numbers are not deterministic. It changes from run to run. So we strongly suggest that run each measurement at least 5 times and report the mean (and standard deviation if you wish to).