CS410 Assignment 4
Matrix Multiplication on GPU

Date released: Mar/28/2024.
Due date: Apr/7/2024 (11:59 pm IST).

There are two parts to this assignment. In the first part of this assignment, you will implement the widely used **matrix multiplication** algorithm for execution on GPUs using CUDA-C**.** In the second part, you will implement the K42 variant of the MCS lock protocol.

Part 1: Implement Matrix Multiplication using CUDA

You will implement and experiment with the algorithm based on the inner product (`ijk` loop). The following code shows the `'ijk'` loop for computing `C=C+A*B`:

```
C = zeros(n,n);
for i = 1 to n
      for j = 1 to n
            for k = 1 to n
                  C(i,j) = C(i,j) + A(i,k) * B(k,j);
```

You can assume that the input matrix sizes are `nxn`.

You will have to implement the parallel matrix multiplication algorithm using CUDA-C. Call this `matmul.cu`. Your parallelization strategy should have each thread compute one element of the C matrix. Each thread should load a row of matrix A from memory, a column of matrix B from memory, compute an element of matrix C and store the result in memory. As part of this implementation, you will have to:

a. Allocate and free global memory on GPU
b. Copy data from CPU to GPU global memory and from GPU global memory to CPU.
c. Launch the kernel on the GPU
d. Free the device and host memory

Your program should accept one argument, n, which indicates the square matrix size. Initialize matrices with random single-precision floating point numbers within a range -1.0 to 1.0. Also, you should implement a function for computing matrix multiplication on the host (you can use the 'ijk' ordering or any other ordering.). This function must be used to validate your results obtained from the device.

Experiments:
When you compile your code using nvcc, make sure to use -O3 optimization. All of the performance measurements should be obtained using the executable produced with this

optimization. Vary the size of your matrices from n=1024, 2048, 4096, 8192. Measure the runtime (separately for data transfer time and device computation time). Plot runtime vs. n.

Report:
Write in the report how your program works. Explain how your program exploits parallelism. This report should NOT include your program, though it may include one or more figures containing *pseudo-code* that sketches key elements of the parallelization strategy.

Show the speedup vs. n plot, include a discussion explaining why you see the speedup that you see. Your response must touch upon SIMD parallelism in addition to other strategies that you adopt.

Part 2: Implement K42 MCS Lock
K42 is a version of MCS that avoids the need to pass a queue node as argument. In the code here: https://www.cs.rochester.edu/research/synchronization/pseudocode/ss.html#K42 notice how "procedure acquire_lock (L : ^lnode)" and "procedure release_lock (L : ^lnode)" only take a Lock pointer and do not need the caller to pass a qnode (compare it with MCS list based queue lock (https://www.cs.rochester.edu/research/synchronization/pseudocode/ss.html#mcs).

1. Carefully understand the code for the K42 MCS lock.
2. Familiarize yourself with c++11 atomics.
   https://en.cppreference.com/w/cpp/atomic/atomic
3. Implement the K42 version of the MCS lock in C++11.
   We have given a scaffolding code that uses default c++ mutex. You need implement the following two functions correctly using the K42 algorithm. The Acquire() and Release() functions can be modified to take a single "lnode" argument to meet the K42 signature.

```
inline void Acquire() {
   // Implement using your K42 Lock.
   mtx.lock();
}

inline void Release() {
   // Implement using your K42 Lock.
   mtx.unlock();
}
```

The provided Makefile offers the following:

`make run MAXTHREADS=8 MAXTIME=10`
Runs the program and prints the throughput (locks / sec) for 1, 2, … 8 threads running each setting for 10 seconds.

`make run_check MAXTHREADS=8 MAXTIME=10`
Runs the same but also checks for some basic correctness.

4. Make sure your K42 lock passes the correctness for the maximum number of CPU cores on the machine (On ParamUtkarsh nodes, it is 40) and setting `MAXTHREADS=40 MAXTIME=30`
5. Plot a graph of NUM_CORES to LOCKS/SEC with `MAXTHREADS=40 MAXTIME=30`. Make sure you collect this data without the CHECK_CORRECTNESS mode (i.e., `make run`)
6. Include a report describing your leanings from implementing the K42 lock.

Execution Environment

You will use the CDAC Param Utkarsh cluster and run your code. You can login using the following command:

ssh <username>@paramutkarsh.cdac.in -p 4422

The Param Utkarsh user manual is shared here. The credentials have been shared with you earlier. If you have trouble logging in, please let the instructors know immediately.
You may do the project alone or in a group of two students.
You may discuss ideas but do not share the code with another team.

Distribution and collection of assignments will be managed using **GitHub Classroom**. You can accept the assignment at the following URL: https://classroom.github.com/a/Q4dI-szt

When you accept the assignment, it will provide you with a clone of a GitHub repository that contains this problem statement only.
.

Submitting your assignment

Your assignment should be submitted in three parts.

- One or more source files containing your different versions of the program. Your submission must edit the given Makefile to include the following targets:
  - `team`: prints the team members' names on the terminal.
  - `part1`: builds related code and runs experiments mentioned in experiments section of part 1.
- A report about your program in PDF format. Guidelines for your report: Make sure that you address all the requirements specified in the "Experiments" section. Don't forget to add Axis headers, legends, figure title, discussion corresponding to a figure, and averaging from multiple runs.
- Data from your experiments. Include a file part1.log that contains consolidated data from experiment related to part 1. This is the output that is written to a file after your job completes.

You will submit your assignment by committing your source files, Makefile, and report to your GitHub Classroom repository. If you have trouble using GitHub and come up against the deadline for submitting your assignment, email the instructor a tar file containing your source code and your report.

Click on the link shared above. This will create a repository in your GitHub account.

1. Clone the repository into your local development environment (on DGX master node) using the **git clone** command.
2. Add all assignment related files that you want to submit to your GitHub local repository using the **git add** command.
3. Save the changes using the **git commit** command
4. Upload the changes to GitHub using the **git push** command
5. Release your changes by first tagging your commit on the local environment using the **git tag -a cs410assignment3 -m "submitting assignment 3"**.
6. Next, push the tag to GitHub with the help of the following command: **git push --tags** \\
   If you want to make changes after you have submitted (repeat the above steps from 1 to 5 and apply modified commands shown below in place of step 4):
   git tag -a -f git tag -a cs410assignment3 -m "submitting assignment 3"
   git push -f --tags

Grading criteria

- 20% for part 1 and 50% for part 2. This includes correctness of the code, completeness (whether your code includes all the requirements), performance, validation code, and logs.
- 30% report (containing both part 1 and part 2). Your grade on the report will consider the quality of the writing (grammar, sentence structure), whether all of the required elements are included, and the clarity of your explanations. Make sure that you have performed all of the experiments requested above and provided the information requested about them in your report. If you haven't, you will lose points!

Using SLURM on ParamUtkarsh to submit GPU jobs

The queue that you should use to submit the jobs is v100. There is another queue a100 available. You may use it if the nodes in that queue are available. See sample script below to know how to do this.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --gres=gpu:1
#SBATCH --error=job.%J.err
#SBATCH --output=job.%J.out
#SBATCH --time=00:01:00
#SBATCH --partition=gpu
module load cuda/10.1
```

```
cd /home/iitdh10/testgpu/11_3_2024
nvcc demo1_spmd.cu -o /scratch/iitdh10/demo1
/scratch/iitdh10/demo1
```