CSCE 435 Fall 2023

Assignment 3: Bitonic Sort using CUDA

Instructions to compile and execute the bitonic sort CUDA code:

1. Upload the starter code files to your scratch directory after logging into grace portal.
2. Open the current directory in the terminal using the “Open in terminal” option on the top.
3. Authenticate using your net id’s password and duo-2 factor authentication.
4. Initialize the cmake build:
    a. **$. build.sh**
5. If further changes are made to the code, use make to re-build:
    a. **$ make**
6. Run the batch file using the following command, giving number of threads and number of values:
    a. **sbatch bitonic.grace_job <t> <v>**
7. After a job is complete, you’ll be able to see the output in the output file corresponding to your jobid in the same directory as the source code.
   (You’ll be able to find out whether a recent job has been completed or not by going to: grace dashboard > jobs > active jobs)

Important: In the code, the number of values that are to be sorted (NUM_VALS) = number of threads (THREADS) x number of blocks (BLOCKS). So, if you had 2^9 threads and 2^15 blocks that should equal to 2^24 numbers.

Assignment:

Summary

- Measure the time taken by the cudaMemCpy function for transferring data (both, host to device, and device to host) and plot **time taken vs number of threads** (64, 128, 512, and 1024) for 3 sizes of NUM_VALS (2^16, 2^20, and 2^24). **[30 points]**
    - You can plot both of the memcpy times (host to device, and device to host) on the same graph. There will be 3 plots for 3 different NUM_VALS.
    - Make sure you also change BLOCKS when you're changing THREADS so as to keep NUM_VALS constant.

- Measure the time taken by the kernel (bitonic_sort_step) to execute (Hint: use cudaDeviceSynchronize()) and plot **time taken vs number of threads** (64, 128, 512, and 1024) for 3 sizes of NUM_VALS (2^16, 2^20, and 2^24). **[30 points]**
    - There will be 3 plots in total for 3 different NUM_VALS.

- Calculate the effective bandwidth of the kernel (in GB/s) for different numbers of threads (64, 128, 512, and 1024) keeping NUM_VALS = 2^24. **[30 points]**
    - For reference: https://developer.nvidia.com/blog/how-implement-performance-metrics-cuda-cc/

- Write down your observations on the variation in runtimes depending on number of threads, block size, and the total number of values. **[10 points]**
- Use Thicket to generate plots **[+10 points]**

Part 1 – Caliper

Use CALI_MARK_BEGIN(name) and CALI_MARK_END(name) region annotations to time the different parts detailed in the summary. The names that should be used for each region are already defined at the top of the file.

Part 2 – CUDA

1. Implement timers – time each region detailed in the summary. This can be done by using the CUDA events API, which is an alternative to using any built-in C++ timers. Some necessary elements:
    a. cudaEvent_t
    b. cudaEventElapsedTime()
    c. cudaEventRecord()
    d. cudaEventCreate()
    e. cudaEventSynchronize()
2. Calculate the effective bandwidth.

Note: Corresponding timers from part 1 and 2 should be equivalent or almost equal in value.

Upload a .zip file on canvas containing:

- **A pdf with your answers**

- **Your code changes**
- **YOUR-NETID.zip of your .cali files**