# Assignment 3 : Bitonic Sort using CUDA

CSCE-435 Fall 2024
Texas A&M University

November 11, 2024

## 1 Compile and execute project

- Upload the starter code to your scratch directory after logging into the grace portal.
- Navigate to the directory that the files were uploaded to after logging into the grace portal.
- Intialize the CMake build using the command:

```
        $ . build.sh
```

- **Whenever** a change is made to the code, run make to re-build:

```
        $ make
```

- Run the batch file by running the following command, making sure to specify the number of threads(**t**) and the number of values(**v**):

```
        $ sbatch bitonic.grace_job <t> <v>
```

- Once the job is completed, you will be able to see the output file named according to the job id.

**Note:** In the code, the number of values to be sorted is calculated as the product of the number of threads and the number of blocks.

`NUM_VALS = THREADS x BLOCKS`

## 2 Assignment

The code provided includes an implementation of Bitonic Sort using CUDA. The task is to measure the time taken for different functions as the number of threads and values are varied.

### Implementation

#### CUDA Timers

Use CUDA timers to mark the following regions

- `cudaMemCpy` function for transferring data from host to device **and** device to host.
- The kernel `bitonic_sort_step`.

Cuda timers can be implemented using CUDA Events API. Some useful elements are

- `cudaEvent_t`
- `cudaEventElapsedTime()`
- `cudaEventRecord()`
- `cudaEventCreate()`
- `cudaEventSynchronize()`

### Effective Bandwidth

Calculate the effective bandwidth of the kernel(in $GB/s$). The following link might be helpful Performance Metrics.

### Caliper

Use `CALI_MARK_BEGIN(name)` and `CALI_MARK_END(name)` region annotations to time for the `cudaMemCpy` function for both host to device **and** device to host.

The time for the **kernel** is automatically measured by Caliper. You can check the times by reading the caliper files using **thicket**.

## Analysis

Make the following plots based on the time measured for the `cudaMemCpy` function (for both host to device **and** device to host), and the **kernel**:

- **Time v/s THREADS**: Vary the number of threads as {**64, 128, 512, 1024**}
- **Time v/s NUM_VALS**: Vary the number of values as {$\mathbf{2^{16}}$, $\mathbf{2^{20}}$, $\mathbf{2^{24}}$}
- **Effective Bandwidth v/s THREADS**: Vary the number of threads as {**64, 128, 512, 1024**}

You can plot both of the `cudaMemCpy` times (host to device, and device to host) on the same graph. There will be **3** plots for 3 different `NUM_VALS`.

**Hint:** When measuring time for the **kernel**, use `cudaDeviceSynchronize()`.

**Note:** Make sure you also change `BLOCKS` when you're changing `THREADS` so as to keep `NUM_VALS` constant.

### Observations

Write down your observations on the variation in runtimes depending on number of threads, block size, and the total number of values. Make sure to explain the reasoning behind observed trends.

### Grading

| **Exercise** | **Points** |
|---|---|
| Correct `cudaMemCpy` time plots | 30 |
| Correct kernel time plots | 30 |
| Correct bandwidth plots | 30 |
| Observations | 10 |

## 3 Submission

Submit a `.zip`file on Canvas containing

- A **pdf** with the plots and your observations.
- Your code changes
- A `.zip` file containing your `.cali` files named **YOUR_NETID.zip**