Lab Assignment 6
Due Nov 23, 2023 at 11:59pm

1 Objective

The purpose of this lab is to implement a kernel the performs **an inclusive parallel scan on a 1D list**. The scan operator will be the addition (plus) operator. You should implement the *work-efficient (Brent-Kung)* kernel in lecture 12. Your kernel should be able to handle input lists of arbitrary length. Please refer to the lecture notes (slides 35-38) on the hierarchical parallel scan for input with arbitrary length. To simplify the lab, the students can assume that the input list will be at most of length 2,048 * 65,535 elements (parallel computation performed using only one kernel launch).

The boundary condition can be handled by filling "identity value (0 for sum)" into the shared memory of the last block when the length is not a multiple of the work group size.

2 Instructions

Edit the skeleton code to perform the following:

- Allocate device memory
- Copy host memory to device
- Initialize thread block and kernel grid dimensions
- Invoke CUDA kernel
- Copy results from device to host
- Deallocate device memory
- Implement the work efficient scan routine
- Use shared memory to reduce the number of global memory accesses, handle the boundary conditions when loading input elements into the shared memory
- Write the CUDA kernel

Compile the template with the provided `Makefile`. The executable generated as a result of compilation can be run using the following code:

```
./ListScan_Template -e <expected.raw> -i <input.raw> -o <output.raw>
-t vector
```

where `<expected.raw>` is the expected output, `<input.raw>` is the input dataset, and `<output.raw>` is an optional path to store the results.

`README.md` has details on how to build `libgputk`, `template.cpp` and the dataset generator.

3 What to Turn in

Submit a report that includes the following:

1. How many global memory reads are being performed by your kernel?
2. How many global memory writes are being performed by your kernel?
3. How many times does a single thread block synchronize to reduce its portion of the array to a single value?
4. Suppose that you want to scan using a binary operator that is not commutative. Can you use a parallel scan for that?
5. Is it possible to get different results from running the serial version and parallel version of scan? Explain.
6. Your version of `template.cpp`.
7. The result as a table/graph of kernel execution times for different input data, with the system information where you performed your evaluation. Run your implementation with the input generated by the provided dataset generator. For time measurement, use `gpuTKTime_start` and `gpuTKTime_stop` functions (You can find details in `libgputk/README.md`).