README

December 6, 2023

0.1 Exercise 4 - Distributed Quicksort

1 Introduction

In this exercise, you will implement a stable version of parallel distributed quicksort across MPI processes to sort data common to all processes. Additionally you will implement it also using GPUs. The exercise consists of three subtasks:

1. Implement the function `quicksort` in `src/quicksort.cu`. Your task here is to implement a single threaded stable version of quicksort.
2. Extend your implementation to be able to use multiple MPI processes. You need to implement `quicksort_distributed` in `src/quicksort_distributed.cu` (you can and are encouraged to re-use your solution from subtask 1.). Now multiple processes hold a copy of the array and each process should end with the same sorted array. Remember to use multiple MPI tasks (-n should be larger than 1).
3. Extend your implementation to work on GPUs. Start first by implementing `quicksort` in `src/quicksort_gpu.cu`. Then extend the GPU implementation to multiple GPUs by implementing `quicksort_distributed` in `src/quicksort_distributed_gpu.cu` (you are allowed to and are highly encouraged to re-use your previous solutions for this). In this exercise you should allocate one process per device (for example `--ntasks-per-node=4 --nodes=2 --gres=gpu:teslap100:4`).

Note: The performance of your implementation is not graded and to get full points, you only need to ensure your implementations give the correct results. However, you should use the hardware relatively efficiently: an implementation that uses a single CUDA thread in task 3, or a single process in tasks 2 or 3, will receive 0 points. Additionally, try to use CUDA-aware MPI instead of first moving data to the CPU and then communicating in task 3.

1.1 Returnables

The solutions should be returned in a single zip file named .zip, for example 12345.zip. The archive should contain at least `src/quicksort.cu`, `src/quicksort_distributed.cu`, `src/quicksort_gpu.cu` and `src/quicksort_distributed_gpu.cu`. You create the archive with the command zip .zip -r src/ (note the -r flag: the archive must contain the src directory).

1.2 Getting started

Run the following commands to get started:

```
module load gcc/11.3.0 cmake/3.26.3 openmpi/4.1.5
cd pps-example-codes/sheet4
mkdir build && cd build
cmake .. && make -j
cd .. && mkdir yourrundir && cd yourrundir
sbatch ../job_scripts/qs_XX.sh
```

Four binaries should appear corresponding to tasks 1, 2, and 3: `quicksort-serial`, `quicksort-distributed`, `quicksort-gpu` and `quicksort-distributed-gpu`. > See Triton user guide for information on how to queue a batch job. There are also some batch scripts in `sheet4/job-scripts` to get you started.

1.3 Hints

Only the functions `quicksort` and `quicksort_distributed` are used for grading. You can add additional helper functions as needed.

If your implementation is very slow, recall how GPUs differ from CPUs (lecture slides). Do not try to create a for loop inside the kernel that loops over all n elements. In addition, you should strive to avoid branch divergence and ensure that the workload is distributed evenly across the stream processors (CUDA cores).

For doing a stable quicksort the prefix sum algorithm is helpful (both when doing it in serial and in parallel). For an introduction to parallel prescan and hints on optimization techniques see NVIDIA's article.