Universiteit Leiden

Third Assignment Report

High Performance Computing I

2019/2020

João Francisco Veríssimo Dias Esteves
s2679663

December 21, 2019

1 Introduction

This assignment implements parallel versions of the Bitonic Sort algorithm across different nodes using OpenMPI, in which one version runs the local sorting workloads in the CPU and the other runs them in the GPU via CUDA. In the GPU's case it also takes advantage of the PSRS algorithm when there isn't enough VRAM available, which also runs the Bitonic Sort algorithm in chunks as detailed in section 2.1.

2 Implementation

Bitonic sort was implemented for both CPU and GPU. In the GPU, thanks to the algorithm's data-independent nature, each step consist of launching threads on the GPU where each thread is responsible for 1 operation of compare-and-swap. It should be noted the algorithm requires an input array with a size that is a power of 2.

2.1 Parallel Sort by Regular Sampling

The Titan X (Maxwell generation) GPU's available on the DAS-5 supercomputer only have 12GB of VRAM, which isn't enough to fit the dataset of 16 billion 32-bit integers. Therefore, Parallel Sort by Regular Sampling (PSRS) was used to sequentially sort chunks of too big datasets in the GPU's.

This work's implementation follows the specification in [1]. In the original algorithm, multiple processors process different chunks of the array and then merge them. In this assignment, each node is already occupied with its own part of the array, which is still too big. So, instead of giving the multiple chunks to different processors because they're already busy, they are copied one-by-one to VRAM, sorted with bitonic sort and copied back to host memory. The merge process that follows has the same principle, being sequentially ordered in chunks with bitonic sort by the GPU.

Since bitonic sort requires an array with a number of elements that is a power of 2, the merge process of the PSRS had to be slightly adapted. After rearranging the partitions to form new chunks, these get a varying size. So, the maximum value is found for each, and it is repeated in each new chunk as many times as required until their length becomes a power of 2. Then, when copying the result to the final array, these extra dummy values are ignored. Thus, bitonic sort can be applied to any array of arbitrary length.

The implementation of PSRS has been successful, as it's possible to sort an array with 2^{32} 32-bit integers which takes a space of $4 * 2^{32} = 4 * 4\text{G} = 16\text{GB}$, larger than the Titan X's 12GB of VRAM.

2.2 Limitations

From datasets of 2^{32} and larger, the sort is successful but only 0's are displayed on the output. It should be noted the local array of each node is sorted correctly. The problem lies with OpenMPI's MPI_Gather() which cannot aggreggate so many elements, as it's limited to the *int* datatype. Still, this is a minor problem in the assignment's context, as an inspection to the local arrays proves that PSRS is working as intended.

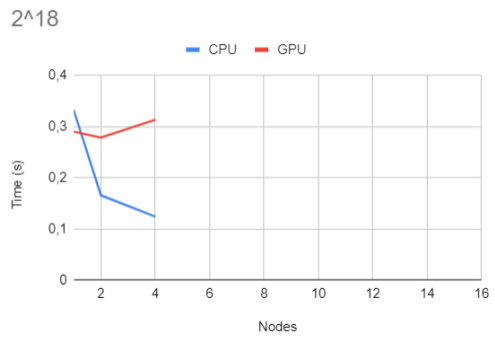
The intended dataset of 2^{34} could not be tested because there isn't enough system RAM available, so instead 2^{32} is used as the biggest dataset.

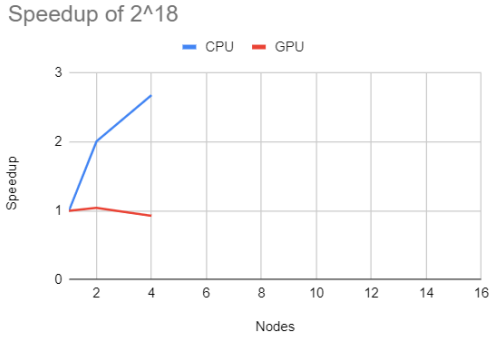
3 Benchmarks

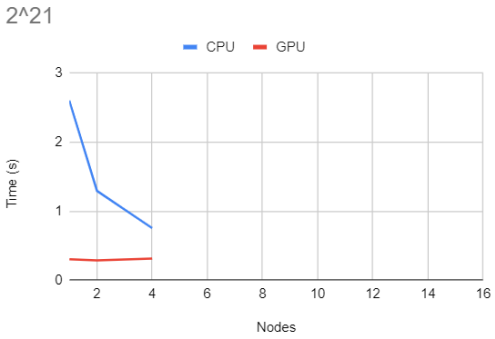
All results obtained from code compiled with OpenMPI and CUDA 10. The times measured didn't take into account the time required to generate the input array. The configurations were a total combination of the following variables:

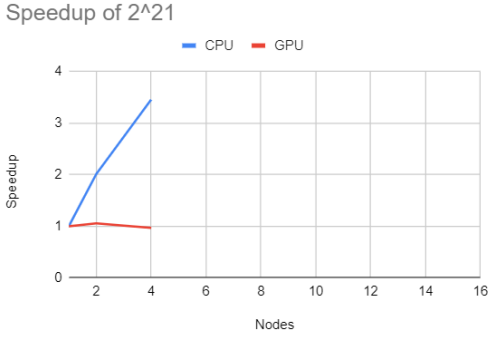
- Nodes: 1, 2, 4, 8, 16
- Execution mode: CPU, GPU
- Base seeds: 0x1234abcd, 0x10203040, 0x40e8c724, 0x79cbba1d, 0xac7bd459
- Dataset size: $2^{18}, 2^{21}, 2^{26}, 2^{32}$

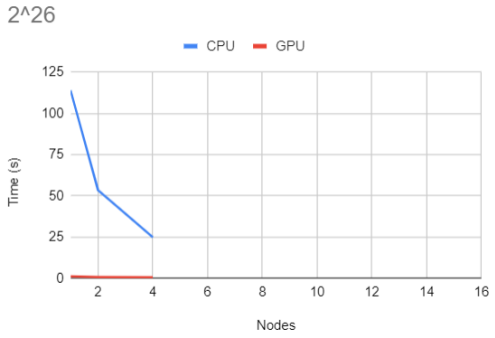
3.1 Results

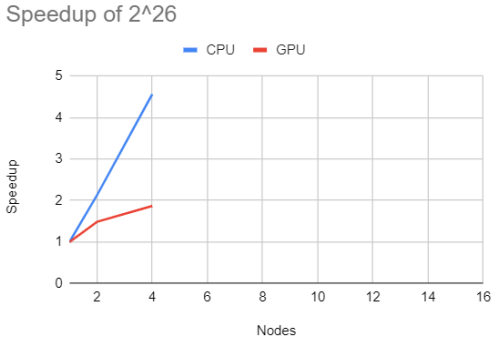












GPU times for 2^{32} with 1 and 4 nodes are respectively 170,533 and 32,967 seconds. This is a 5x speedup from 1 to 4 nodes.

There isn't enough time in the world to test how long the CPU would take with such a large dataset (note that the GPU version takes between 46 and 128 times more time to run than with the 2^{26} dataset), and DAS-5 is far too overloaded to test other configurations at this time.

3.2 Analysis

The CPU version presents a very good speedup, while the GPU version even degrades on the two smaller datasets, only starting to increase on the bigger ones. However, from 2^{21} and larger, the GPU version is far faster, demonstrating well the capabilities of a GPU to process such a large amount of data.

References

[1] X. Li, P. Lu, J. Schaeffer, J. Shillington, P. S. Wong and H. Shi. On the Versatility of Parallel Sorting by Regular Sampling, 1993. Available at `http://www.math-cs.gordon.edu/courses/cps343/doc/psrs.pdf`