UNIVERSITÀ DEGLI STUDI DI MILANO

GPU Computing Project

Bitonic Sort using Numba on Nvidia GPUs

Author:

Manuele Lucchi

08659A

Academic Year 2022/2023

1 Abstract

This documents describe the performances of an existing algorithm, the Bitonic Sort, implemented using Python and Numba, comparing it with a serial approach and CUDA C implementation.

Contents

2 Introduction

The Bitonic Mergesort [1] is an algorithm created in 1968 by Ken Batcher [2]. The elements to choose for the comparison are indipendent from the value (it's not data-dependant) therefore is well suited for parallel processing.

Over the years there are been many implementations using low level languages such as C and even high level such as Java or Python, but regarding the parallel variant, only C/C++ examples can be found on the web (using OpenMP [3] and CUDA C [4]) The CUDA community developed an abstraction layer over the CUDA runtime to be able to execute CUDA kernels from python with little-to-no overhead using Numba [5].

Numba is not a CUDA specific framework, it's a generic parallelization abstraction that allows to compile Python functions into machine code, while integrating with numpy [6] to have access to a powerful tensor manipulation library; but of course its primary target is CUDA.
The scope of the paper is to implement the algorithm using Numba and compare it with different implementations

3 Algorithm

The algorithm is based on the concept of Bitonic Sequence [1], a sequence with $x_0 \leq ... \leq x_k \geq x... \geq x_{n-1}$ for some k, $0 \leq k \leq n$, meaning that there are two subsequences sorted in opposite directions.
By using a sorting network, we can create a Bitonic Sequence from any sequence and then merging them to obtain the final sorted sequence, so the algorithm has two phases.
This algorithm has an asymptotic complexity of $O(nlog(n)^2)$, the same of the odd-even mergesort [7] and shellsort [8]

3.1 Step 1: Bitonic Sort

To create a Bitonic Sequence we need to build a sorting network. This step has $log_2 n$ phases (with the last one being the second step), each one with i stages, where i is the current phase starting from 1. By sorting each pair of elements in the sequence in different directions pairwise (using the so called "comparers"), we obtain a sequence full of bitonic subsequences. We can then at each phase double the size of these subsequences and half their number.
As we can see in picture one, at each stage of a phase the elements in the comparers get closer and in each phase a bitonic subsequence is sorted. This structure is known as butterfly network [9]

3.2 Step 2: Bitonic Merge

The last step is a variation of the first one, where we only have a single bitonic sequence and sort the two subsequences with the comparers oriented in the same direction, resulting in the sorted sequence.

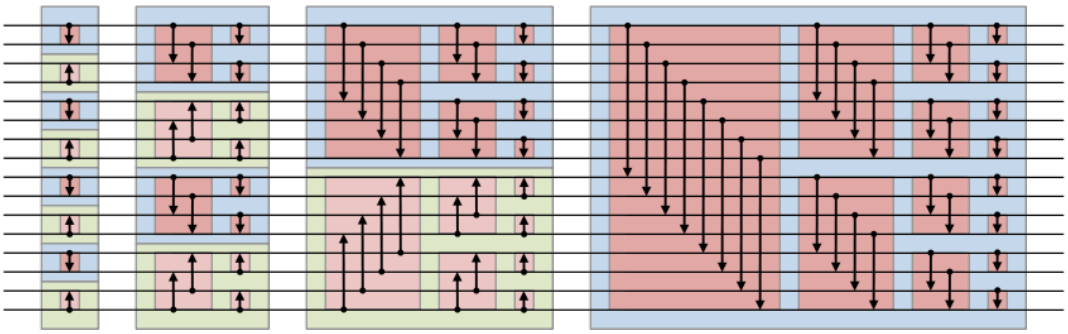

Figure 1: The structure of a bitonic sorting network

4 Implementation

The scope of the implementation is to use Numba to measure its efficiency compared to serial python algorithms and a standard CUDA C++ version.
The other versions are inspired by the course material and by the Wikipedia iterative implementation, with some changes to support a generic sorting direction (and of course, translated to python).

4.1 Sequences of length not power of 2

5 Benchmark and Profiling

Size	CPU Recursive	CPU Iterative	GPU
10	10 ms	10 ms	10 ms
10	10 ms	10 ms	10 ms

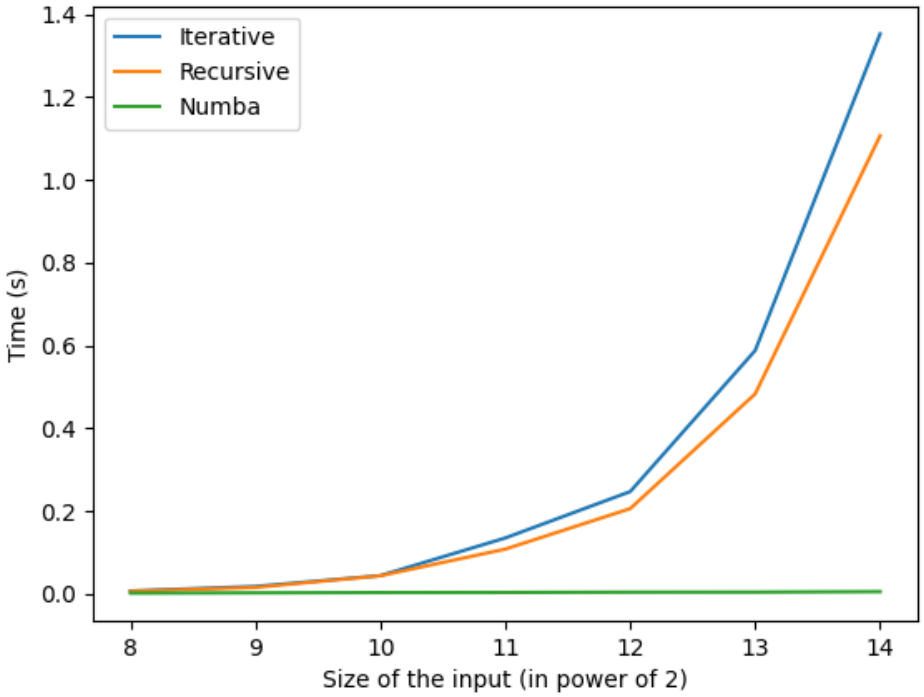

Figure 2: Time comparison between different implementations

Size	Cuda C++	Python Numba
10	10 ms	10 ms
10	10 ms	10 ms

6 Conclusion

References

1. Einstein, A. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik* **322,** 891–921 (1905).

2. Knuth, D. *Knuth: Computers and Typesetting* `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).

3. Knuth, D. *Knuth: Computers and Typesetting* `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).

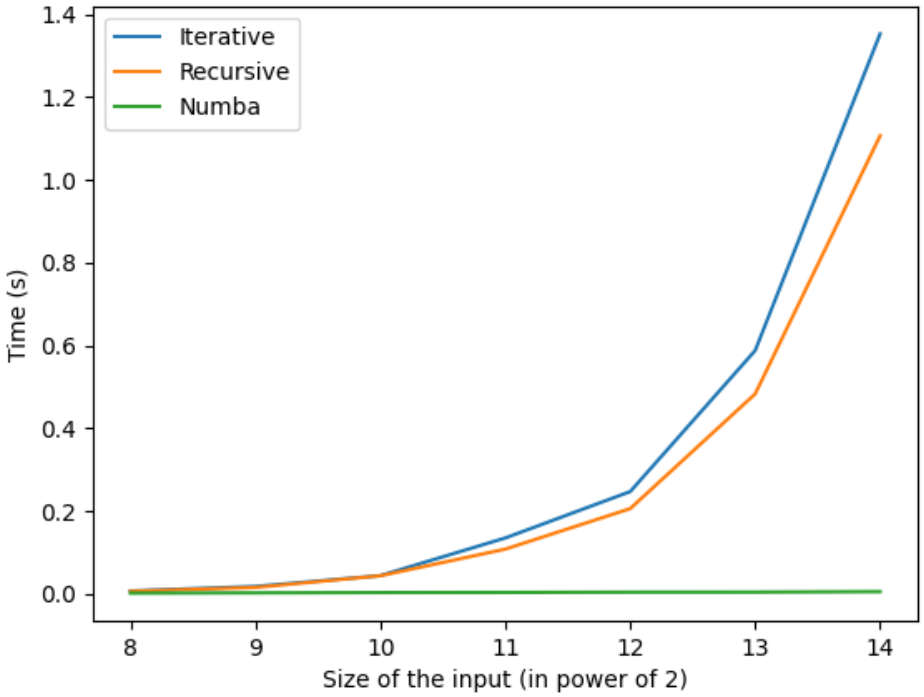

Figure 3: Time comparison between parallel implementations on different platforms

4. Knuth, D. *Knuth: Computers and Typesetting* `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).

5. Knuth, D. *Knuth: Computers and Typesetting* `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).

6. Knuth, D. *Knuth: Computers and Typesetting* `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).

7. Knuth, D. *Knuth: Computers and Typesetting* `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).

8. Knuth, D. *Knuth: Computers and Typesetting* `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).

9. Knuth, D. *Knuth: Computers and Typesetting* `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. (accessed: 01.09.2016).