HPA Final Project

Sorting Algorithms

Chaithanya Gadiyam

Instructor: Stephen Moskal
TA: Stavan Satish Karia

Professor: Sonia Lopez Alarcon

Abstract

This project implements data-independent sorting algorithms such as rank sort and odd even sort on both CPU and GPU. The main objective is achieve good speedups by efficient GPU implementation. Both CPU and GPU implementations of rank sort and odd even sort are run for different data sizes ranging from 256 to 131072. And results including CPU execution time, GPU execution time and speedup are tabulated for rank sort and odd even sort.

Design Methodology

Sorting is a computational building block of fundamental importance and is one of the most widely studied algorithmic problems. Although implementing sorting algorithms on the CPU is relatively straightforward, sorting on the GPU is less easily implemented because the GPU is effectively a highly parallel single-instruction, multiple-data (SIMD) architecture. Given that the GPU can outperform the CPU both for memory-bound and compute-bound algorithms, finding ways to sort efficiently on the GPU is important. Furthermore, because reading back data from the GPU to the CPU to perform operations such as sorting is inefficient, sorting the data on the GPU is preferable. Sorting algorithms can be divided into two categories: data-driven ones and data-independent ones. Data-independent sorting algorithms are more efficient on GPU, as they do not change their processing according to the current key value, they always take the same processing path. Two such sorting algorithms are rank sort and odd even sort, which are chosen in this project to be implemented on both CPU and GPU.

Based on the size of the data, an array of numbers of float type in the range 0 to 1 are generated randomly. These random set of numbers are used to sort. For rank sort, the number of numbers that are smaller than each selected number is counted and this count provides the position of selected number in sorted list; that is, its “rank”. First a[0] is read and compared with each of the other numbers, a[1] … a[n-1], recording the number of numbers less than a[0].Suppose this number is x. This is the index of the location in the final sorted list. The number a[0] is copied into the final sorted list b[0] … b[n-1], at location b[x]. Actions repeated with the other numbers. Hence using the rank of each number in the array, sorted array is formed. For the GPU implementation of rank sort, a one dimensional block of 512 threads is used to obtain optimal performance, as each stream multiprocessor can contain a maximum of 1536 threads and the provided configuration would assign three blocks per stream multiprocessor. Here the number of threads launched by the kernel is equal to total number of elements and each thread calculates the rank of one element and the position of that element in the sorted array.

Odd Even sort is based on the Bubble Sort technique of comparing two numbers and switching them if the first is greater than the second, to achieve a left to right ascending ordering. Odd Even sort operates in two alternating phases, odd phase and even phase. In each pass, elements at odd-numbered positions perform a comparison check based on bubble-sort, after which elements at even numbered positions do the same. For an array of size n, sorted array is formed after n passes. For the GPU implementation of odd even sort, a one dimensional block of 512 threads is used to obtain optimal performance. The number of threads launched by the kernel is equal to half of the total number of elements and each thread compares the corresponding element in the array with the element next to it, depending on even phase or odd phase. The odd even sort kernel is called twice for each pass, one for even phase and one for odd phase. Since for a data set with size of n, n passes are required, kernel is called a total of 2n times. Synchronization is done after completion of each kernel operation to ensure proper sequence of operations. And the final sorted array is copied from device to host after n passes. The execution times of CPU implementation and GPU implementation along with speedups are calculated for both rank sort and odd even sort.

Results and Analysis

The program is run on data sizes ranging from 2^8 to 2^16 and the CPU execution time and GPU execution time of both rank sort and odd even sort for each data size are listed in table 1 and table 2 respectively. Also graphs of speedup versus data size for rank sort and odd even sort are plotted and shown in figure 1 and figure 2 respectively. From the results, it can inferred that the speedups of both rank sort algorithm and odd even sort algorithm improve with increase in data size. Also rank sort offer better speedup than odd even sort for a given data size. This is partly because of higher CPU execution time of rank sort when compared to odd even sort. But GPU execution times of odd even sort are much higher than that of rank sort. The major reason for this the divergence caused by odd even sort algorithm which is built on bubble sort technique, as threads take different path based on the result of comparison even though dependency is avoided using even phase and odd phase. But considering the results of odd even sort alone, it can be observed that the speedup improves with increase in total number of elements. Figure 3 shows the sample console output for data size of 131072.

Data Size	CPU Execution time	GPU Execution time	Speedup
256	0	0	0
512	0	0	0
1024	3.2	0	0
2048	18.8	3.2	5.875
4096	78	3.2	24.375
8192	302.6	9.4	32.1915
16384	1214.4	28	43.3714
32768	5009.6	99.8	50.1964
65536	20729.8	365.6	56.7008
131072	92744.6	1454.8	63.7508

Table 1. Rank sort results

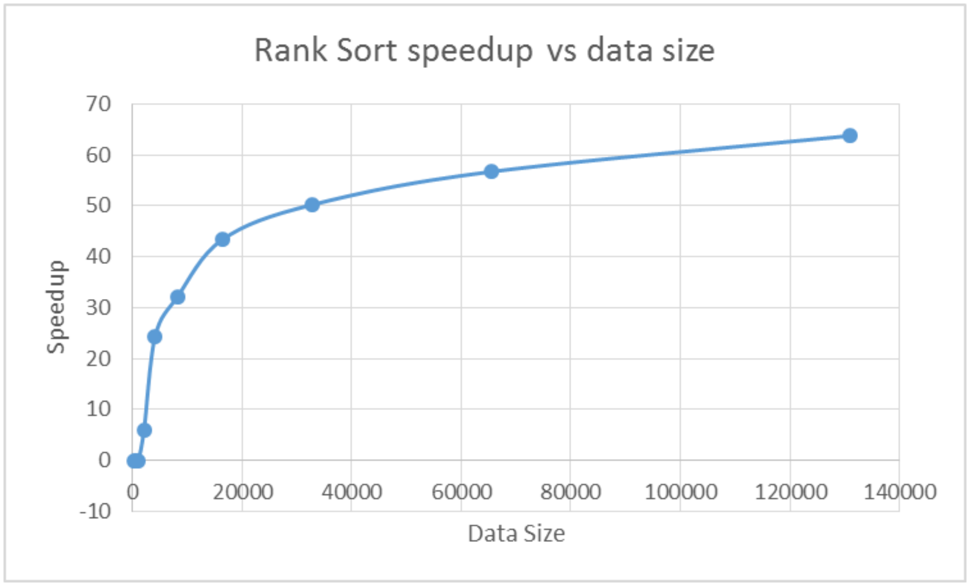


Figure 1. Plot of rank sort speedup versus data size

Data Size	CPU Execution time	GPU Execution time	Speedup
256	0	18.8	0
512	0	34.2	0
1024	3	65.6	0.0457
2048	12.4	153	0.081
4096	47	327.6	0.1434
8192	193.4	585.6	0.33026
16384	783	1176.4	0.66559
32768	3191.8	2444.4	1.30576
65536	12996.4	6496.8	2.00043
131072	51472.6	13255.2	3.8832

Table 2. Odd even sort results

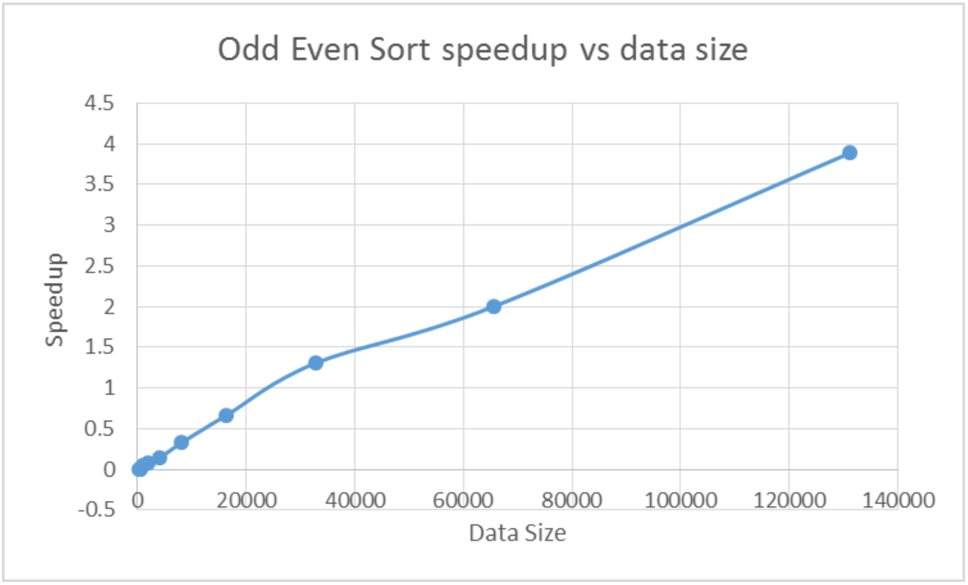
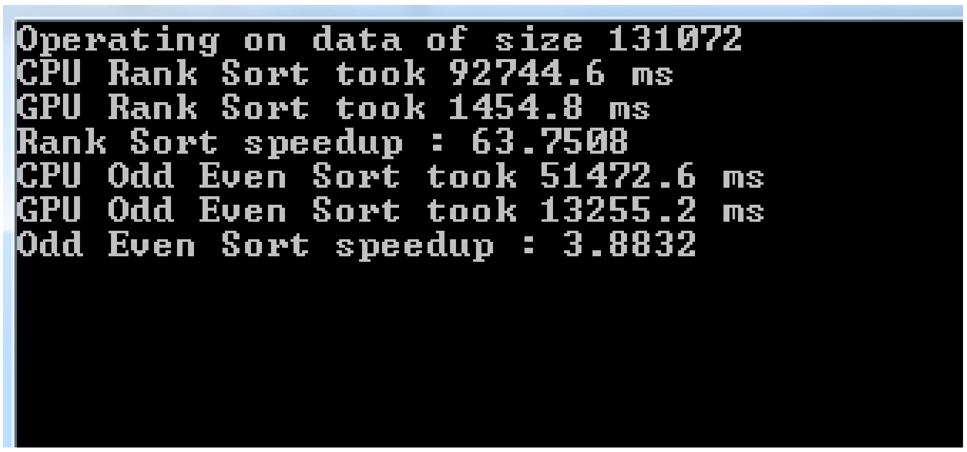


Figure 2. Plot of odd even sort speedup versus data size



```
Operating on data of size 131072
CPU Rank Sort took 92744.6 ms
GPU Rank Sort took 1454.8 ms
Rank Sort speedup : 63.7508
CPU Odd Even Sort took 51472.6 ms
GPU Odd Even Sort took 13255.2 ms
Odd Even Sort speedup : 3.8832
```

Figure 3. Sample Console output

Conclusion

A program with both CPU and GPU implementations of rank sort and odd even algorithms are implemented successfully with results showing decent speedups for the two sorting algorithms. Comparing the performance of two algorithms, it can be said that rank sort performs better than odd even sort on GPU, whereas odd even sort gives better performance on CPU when compared to rank sort. And both algorithms give better speedups for larger data sizes. For rank sort algorithm, usage of constant memory would improve the performance, but with a limit of 64kB constant memory, it does not work for data sizes greater than 16k.