Sorting Optimization

Adam Todd

Table of Contents

Background

- Sorting is an essential process for many applications

- Data Analytics
- E-Commerce
- Database querying

Background

- Not all sorting algorithms lend nicely to multithreading
  - Bubble Sort
  - Insertion Sort
- To be efficient, we must use algorithms that are parallelizable
  - Merge sort
  - Quick Sort
- For top level performance, we must go beyond this into algorithms that are extremely parallelizable, and utilize GPU parallelism
  - Radix Sort
  - Bitonic Sort

Experiment

To determine the effectiveness of multithreading on sorting algorithms, I implemented multiple sorting algorithms with different levels of parallelization.

- Single Threaded Merge Sort
- Multithreaded Merge Sort
- Radix Sort utilizing CUDA

Size	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000
merge	0.000015	0.000131	0.001607	0.023406	**0.285668**	3.694614	47.471729	**595.80066**	
mt_merge	0.127111	0.135648	0.13781	0.226952	**0.36465**	2.345182	27.087753	**333.03562**	
radix	0.154003	0.14	0.141004	0.151	0.161	0.155999	0.240002	0.936044	60.263844
native	0.000004	0.000011	0.00011	0.001164	0.01812	0.363503	**4.99329**	57.720335	**837.217**
cupy native	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	**0.006**	**0.042**	**2.2886**

Experimental Data

Analysis

- As expected, GPU level parallelism performed extremely well
- The already optimized python sort implementation was 38000% slower then the cupy sort for 1 billion item lists
- Even my own implementation of radix sort significantly outperformed the python implementation
- Significant overhead for small sizes, 4 million % slowdown for sorting 10 item list

Analysis

- Out of memory issues…
  - Less optimal in place algorithm might be optimal depending on resource availability or speed requirements
- Power Consumption
  - GPU focused algorithms take a lot more power

Conclusions

- For smaller list sizes, overhead typically worsens performance
- GPU parallelism can greatly improve performance
- Best algorithm heavily depends on specific use case, more than just best Big O
- When possible, keep data pre-sorted