Project Proposal

Parallel Sorting Algorithms in Python

CS550 – Advanced Operating Systems

Feb 6, 2023

Zhuolin Fu

A20501576

Rakesh Abbireddy

A20525389

PROJECT GUIDE

MENG TANG

1. Introduction

Sorting algorithms are widely used in many programs, and with the growth of big data, the need for efficient algorithms to handle large datasets has increased. Parallel sorting algorithms enables efficient processing of large amounts of data by dividing the sorting task into smaller sub-tasks that can be executed simultaneously on different nodes in a distributed system. The main challenges in parallelizing sorting algorithms include Synchronization and coordination between computing nodes, limited memory for concurrent burst of usage and load balancing.

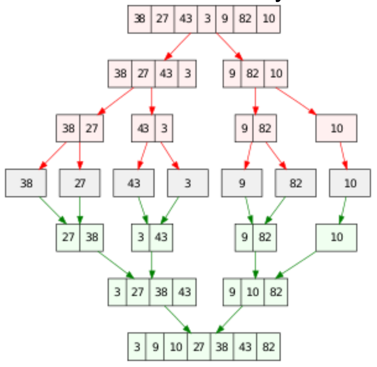
Python is a popular choice for data processing and parallel computing. It has several favorable features: Large ecosystem of libraries including Dask [1], high Interoperability that makes it easy to implement and read and Community support that constantly provides improvement and new features.

Dask is designed to scale Python computations from a single machine to a cluster of machines, making it well suited for parallel computing in a computer cluster environment. With little effort, we can take advantage of the processing power of multiple machines in a highly parallel and efficient manner.

This project aims to implement a baseline parallel sorting algorithm and a more advanced one by leveraging both Dask and cleverly designed algorithms, and to evaluate their performance against each other.

2. Background Information

Merge sort is an efficient sorting algorithm that follows divide-and-conquer paradigm. The input array is divided into sufficiently small subarrays. Each subarray is sorted using a simple subsort. Then the sorted subarrays are merged into the final sorted array using a linear scan. See figure on the right for an example.



Single-core merge sort is not a satisfying solution for distributed systems, where data can be enormous and distributed. Parallel sorting algorithms can divide an array into smaller sub-arrays and sort each sub-array in parallel, making use of resources from multiple nodes. This allows a faster sorting process, as multiple processors are working in parallel.

3. Problem Statement

The aim of this project is to implement a parallel sorting algorithm in Python using the Dask library, and evaluate its performance against a sequential sorting algorithm. The algorithm can be any sorting algorithm, including bubble sort, insertion sort, merge sort, quick sort, or any other algorithm that students choose to implement. The students should justify their choice of algorithm, including the advantages and limitations of the algorithm, and the reasons for choosing it over other algorithms.

4. Related Work

Parallel merge sort algorithms are developed based on different computation models, including early circuit-based model and Parallel Random Access Machine (PRAM). In circuit-based model, sorting operations are performed by manipulating the values on these wires using logical operation. The use of this model has declined due to more powerful hardwares.There are 3 substypes for PRAM paradigm: Concurrent Read, Concurrent Write (CRCW), Concurrent Read, Exclusive Write (CREW) and Exclusive Read, Exclusive Write (EREW).

Preparata [2] proposed an algorithm with O(log n) time and O(nlog n) processors under CREW PRAM, based on an efficient merging procedure [3].

Later, Cole [4] came up with a O(log n) time algorithm using only linear number of processors. To improve CPU utilization, Jeon and Kim [5] gave an load-balanced algorithm that achieved $(P - 1)\log log\, P$ speedup, where P is the number of processors.

Bilardi and Nicolau [6] gave an implementation of bitonic sort under EREW PRAM with time complexity O(log^2 n) and n / log n number of processors. The constant coefficient in the time complexity is very small.

Our project includes implementation of one or more of these algorithms and integration with Dask.

5. Proposed Solution

In this project, first we are going to implement the single-core sequential merge sort. Then we will implement a baseline parallel merge sort. This baseline algorithm divides the input array into subarrays, scatter subarrays to different processing units, and merge them recursively. After that, we will try to implement the more advanced O(log n) algorithm by Richard Cole. All algorithms will be implemented with Python. In addition, Dask library will be integrated into the algorithms. The evaluation will be comprehensive, by means of varying algorithm parameters, input array size, etc. The built-in Python sort will also be added into comparison.

6. Evaluation

The evaluation of this project is comprised by the following:

1. Time: The amount of time the algorithm takes to sort a given set of data.
2. Space: The amount of memory used by the algorithm to sort the data.
3. Stability: A sorting algorithm is considered stable if it preserves the relative order of elements with equal values.
4. Ease of Implementation: How easy it is to understand and implement the algorithm.
5. Use Cases: Some algorithms may be more suitable for specific use cases, such as sorting large datasets or real-time sorting.

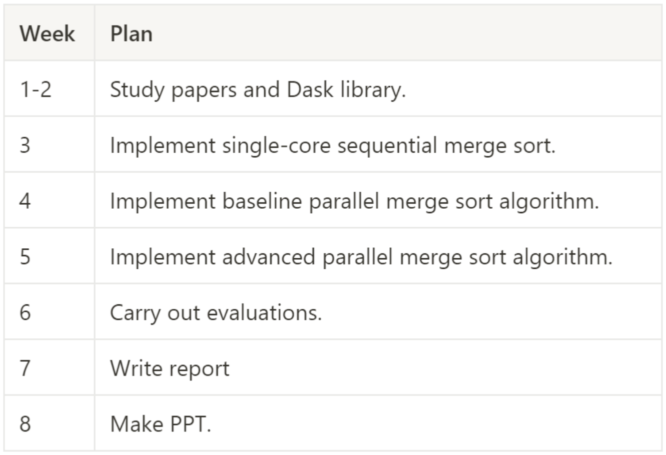
7. Conclusions

This project aims to implement and compare some parallel sorting algorithms in Python with the help of Dask and determine the best algorithm under different conditions. The results of the project will provide insights into the best parallel sorting algorithm in Python, which will be useful for researchers and practitioners working in the field of data analysis and information retrieval. The findings of the project will also contribute to the understanding of the performance limitations of parallel algorithms in Python and help to inform future research in this area.

Overall, this project will provide valuable insights into the performance of parallel sorting algorithms in Python, and help to inform the choice of algorithms for sorting large datasets. The results of the project will contribute to the body of knowledge in the field of computer science, and provide a foundation for future research in this area..

8. Additional Resources

8.a) Timeline



Week	Plan
1-2	Study papers and Dask library.
3	Implement single-core sequential merge sort.
4	Implement baseline parallel merge sort algorithm.
5	Implement advanced parallel merge sort algorithm.
6	Carry out evaluations.
7	Write report
8	Make PPT.

8.b) Deliverables

1. One report in PDF.
2. PPT presentation slides.
3. Source code.

References

[1] Rocklin, Matthew. "Dask: Parallel computation with blocked algorithms and task scheduling." *Proceedings of the 14th python in science conference*. Vol. 130. Austin, TX: SciPy, 2015.

[2] Preparata, Franco P. "New parallel-sorting schemes." *IEEE transactions on Computers* 27.07 (1978): 669-673.

[3] Valiant, Leslie G. "Parallelism in comparison problems." *SIAM Journal on Computing* 4.3 (1975): 348-355.

[4] Cole, Richard. "Parallel merge sort." *SIAM Journal on Computing* 17.4 (1988): 770-785.

[5] Jeon, Minsoo, and Dongseung Kim. "Parallel merge sort with load balancing." *International Journal of Parallel Programming* 31 (2003): 21-33.

[6] Bilardi, Gianfranco, and Alexandru Nicolau. "Adaptive bitonic sorting: An optimal parallel algorithm for shared-memory machines." *SIAM Journal on Computing* 18.2 (1989): 216-228.