LITERATURE REVIEW: Histogram Sort with Sampling

Megha Agarwal
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
meghaagarwal@cmail.carleton.ca

October 5, 2021

1 Introduction

Sorting is an important task in many computing applications owing to the ease of accessing data as per application requirement. With the expansion of parallelization to almost all domains today, it is crucial that our sorting algorithms can be implemented in parallel as well. The idea behind a parallel sorting algorithm is to distribute N keys across p processors in a manner that they are sorted on the global level. A subset of this parallel sorting is the partition based parallel sorting where the system determines a set of splitter intervals to achieve a global or local balanced splitting and then redistribute the keys. So, this project throws some light on one such state-of-the-art parallel sorting algorithm which integrates histogramming and sampling to result in a low communication and low computation algorithm. Here, sampling aids the partition to be done on a representative subset of input keys and histogramming facilitates the iterative processing and evaluation for each given partition.

2 Literature Review

The chief goal of a parallel sorting algorithm is to distribute all input keys over all available processors such that for each processor k there are no keys on k that are greater than any key on processor *k+1*. We can achieve an exact split when all processors have the same number of keys, whereas an approximate split can be achieved if for any ϵ, there are $\frac{N(1+\epsilon)}{p}$ keys per processor[5]. This induces that the primary goal for any parallel sorting algorithm is to find a global partition. In these partition-based algorithms, like quick sort, it is highly beneficial to partition the data before redistributing it, in order to lower the communication cost. Two very efficient parallel sorting algorithms, Parallel Quick Sorting Algorithm (PQSA) and Merging Subarrays from Quick Sorting Algorithm (MSQSA), are proposed by Lingxiao Zeng in [12] but they perform exceptionally with a complexity of $\frac{N}{p}O\left(log\frac{N}{p}\right)$ with shared memory. However, the focus of this project is over distributed memory parallel computers.

The most practical comparison-based sorting algorithm for distributed memory computers is realized to be sample sort [4], which partitions the input data into multiple pieces to be represented by a subset of keys. Sampling on a data can be either randomized or regular [2] , [9], where random sampling shows better results in theory [5], although it provides poor scalability owing to the large sample sizes required to attain load balancing, but, it is

regular sampling that shows better performance practically by resulting in an almost perfect global balancing [6]. An alternative of deterministic sampling could also have been used [3], but they perform exceptionally in the presence of GPUs. As long as $\theta\left(plog\frac{N}{\epsilon^2}\right)$ keys are selected for a sample, a global load balance can be achieved. It is evident from [5] that it is enough to collect a number of samples that scales logarithmically with $1/\epsilon$ and almost linearly with p. The sample sort algorithm samples s keys from each processor resulting in a total of $M{=}ps$ keys at the central processor. The steps taken to perform sampling are:

1. **Sampling Phase**: Each processor samples s keys and communicates it to the central processor, where s is called the oversampling ratio.

2. **Splitter determination**: The central processor then sorts these samples and identifies *p-1* splitters producing p partitions. The interval between each of these splitters is assigned to one processor by broadcasting the splitters to all processors.

3. **Exchanging data**: Each processors partitions data according to the received splitters and exchanges them with their destination processors, requiring an all-to-all communication. When all processors have received their allocated keys, they can then use any shared memory sorting algorithm [12] to sort the local subset.

An advancement on the sample sort is probabilistic partitioning, where rather than picking splitters after just one round of sampling, a vector of splitter candidates is iteratively refined until the satisfactory load balancing is achieved [8]. Further, the processing is done on a sample of keys to pick initial guesses and then continue with our histogramming on the sample. AMS-sort with one round of histogramming provides a locally balanced partition with a sample of $O\left(p\left(logp+\frac{1}{\epsilon}\right)\right)$ [1] and when this is expanded to multiple rounds of histogramming, provides a globally balanced partition, specifically O(p) over $O\left(log\left(log\frac{p}{\epsilon}\right)\right)$ suffices [5]. Hyksort [11] derived from the quicksort for hypercube, combines partitioning the data for each processor around a randomly chosen pivot, along with histogramming, such that we are now looking k pivots that partition the data into k+1 groups. The scanning algorithm used for AMS-sort [1] gives better partitioning than HSS [5] over one round of histogramming, but HSS shows better results than AMS-sort with multiple rounds of histogramming.

For histogram sort, there are numerous ways to create a histogram [7] as demanded by the application. The fundamental process of histogram sort [cite from 1 histogram sort] is to repeatedly refine a partition by collecting histograms of the total number of input keys present in each interval induced by the most recent set of splitters. The distributed histogram sort [8] comprises of 4 basic steps:

1. **Local Sort**: Each processor sorts is local data using any shared memory sorting algorithm which has an expected time complexity of *O(n log n)*.

2. **Splitting**: Generalizing the distributed selection algorithms, like median of partition strategy with weighted medians, to distributed multi-selection, the local array is partitioned into P subsequences [8], [10]. Rather than determining one pivot, multiple pivots are selected in each iteration, for each active range. When a pivot matching a specific rank is found, that range is discarded and pivots for all other ranges is examined from each of the two subranges. Here, each splitter is represented as a tuple containing the splitter upper and lower bounds and the splitter value. So, any splitter S_i is successfully determined if the lower bound L_i and upper bound U_i satisfy

the condition $L_i(S_i) < K_{i+1} \leq U_i(S_i)$, where K denotes the rank of splitter and i ϵ {1,2,...,P}. It is here that histogramming is performed, as mentioned in [8]. All splitters are initialized with a minimum and maximum for the global range. Then, the splitters are iteratively determined by converging the minimum and maximum for each local histogram which is combined to form a global histogram. The complexity for this is $O(p(logp) + p(logn_i))$ per iteration.

3. **Data exchange**: All processors exchange their locally sorted sequences with all other processors after determining all splitters. A permutation matrix is created where that helps identify the send displacements for each processor i [8].

4. **Local Merge**: Finally, each processor locally merges the received sorted sequence. There are two options possible here, either to sort the full array with a complexity of $O\left(\frac{N}{p}log\frac{N}{p}\right)$ using a fast shared memory algorithm or merge it out of place in $O\left(\frac{N}{p}logp\right)$ time using a binary merge algorithm [8].

The analysis for the algorithm in [8] is done using Charm++ with MPI-3 and OpenMP, and DASH. It was evident that histogramming proves to be the bottleneck in the case where number of processors was increased. Between Charm++ and DASH, it was also seen that the algorithm in [5], which uses Charm++ demonstrated volatility, and resulted in performance degradation. There was also an issue observed with the scalability of the all to all communication [8], which may stem from MPI being able to support small data transfer in all to all messages and not such big chunks as performed for this algorithm. The cost for local sorting is calculated to be $O\left(\frac{N}{p}log\frac{N}{p}\right)$, broadcasting the splitters takes O(p) time and the final data movement needs all data to be sent to all processors incurring a cost of O(N/p). A global histogram is computed by reducing all local histograms with O(S) communication and computation. So, it is safe to say that communication and computation cost of histogramming are proportional to the sample size [5].HSS determines all splitters in $O\left(log\frac{(logp)}{\epsilon}\right)$ which is way better than that of HykSort. It can be expected for a single-staged AMS-sort to take approximately 3 times the time for splitting phase as compared to HSS[5]. When examining the algorithm over various input distributions, it was also observed that the algorithm was not much affected by it[5].

References

[1] Michael Axtmann, Timo Bingmann, Peter Sanders, and Christian Schulz. Practical massively parallel sorting. *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, 2015.

[2] G. E. Blelloch and B. M. Maggs C. E. Leiserson. An experimental analysis of parallel sorting algorithms. *Theory of Computing Systems*, 31(2):135–167, 1998.

[3] Frank Dehne and Hamidreza Zaboli. Deterministic sample sort for gpus. *Parallel Processing Letters*, 22(03):1250008, 2012.

[4] W. D. Frazer and A. C. Mckellar. Samplesort: A sampling approach to minimal storage tree sorting. *Journal of the ACM*, 17(3):496–507, 1970.

[5] Vipul Harsh, Laxmikant Kale, and Edgar Solomonik. Histogram Sort with Sampling. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 201–212, Phoenix AZ USA, June 2019. ACM.

[6] David R. Helman, Joseph Jájá, and David A. Bader. A new deterministic parallel sorting algorithm with an experimental evaluation. *ACM Journal of Experimental Algorithmics*, 3:4, 1998.

[7] Wookeun Jung, Jongsoo Park, and Jaejin Lee. Versatile and scalable parallel histogram construction. *Proceedings of the 23rd international conference on Parallel architectures and compilation*, 2014.

[8] Roger Kowalewski, Pascal Jungblut, and Karl Furlinger. Engineering a distributed histogram sort. *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019.

[9] Xiaobo Li, Paul Lu, Jonathan Schaeffer, John Shillington, Pok Sze Wong, and Hanmao Shi. On the versatility of parallel sorting by regular sampling. *Parallel Computing*, 19(10):1079–1103, 1993.

[10] E. L. G. Saukas and S. W. Song. A note on parallel selection on coarse-grained multicomputers. *Algorithmica*, 24(3-4), 1999.

[11] Hari Sundar, Dhairya Malhotra, and George Biros. Hyksort. *Proceedings of the 27th international ACM conference on International conference on supercomputing - ICS 13*, 2013.

[12] Lingxiao Zeng. Two parallel sorting algorithms for massive data. *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2021.