SAPCo Sort: Optimizing Degree-Ordering for Power-Law Graphs

Mohsen Koohi Esfahani
0000-0002-7465-8003

Peter Kilpatrick
0000-0003-0818-8979

Hans Vandierendonck
0000-0001-5868-9259

{mkooohiesfahani01, p.kilpatrick, h.vandierendonck}@qub.ac.uk

Queen’s University Belfast, Northern Ireland, UK
https://blogs.qub.ac.uk/GraphProcessing

Abstract—We introduce the Structure-Aware Parallel Counting (SAPCo) Sort algorithm that optimizes performance of degree-ordering, a key operation in graph analytics. SAPCo leverages the skewed degree distribution to accelerate sorting. The evaluation for graphs of up to 3.6 billion vertices shows that SAPCo sort is, on average, 1.7–33.5 times faster than state-of-the-art sorting algorithms such as counting sort, radix sort, and sample sort.

Index Terms—High Performance Computing, Graph Algorithms, Degree-Ordering, Sorting Algorithms, Real-World Graphs, Structure-Aware Algorithms

I. INTRODUCTION

In degree-ordering, vertices of a graph are ordered based on their degrees. Degree-ordering is a basic tool in several graph algorithms such as [1], [2], [3], [4], [5], [6], [7] and its efficiency plays an important role in processing large and fast-growing real-world graphs.

Many real-world graphs derived from bioinformatics, social networks, and the world-wide web show a skewed degree distribution, following a power-law distribution: a small fraction of vertices are connected to a disproportionately large fraction of other vertices.

Several sorting algorithms with optimized complexities and implementations such as [8], [9], [10], [11], [12], [13], [14], [15], [16] have been introduced; however, they are not well-adjusted for real-world graphs. The parallel algorithms that work based on sample sort [9] and radix sort [8], move elements several times until they are accommodated in their final places. On the other hand, counting sort [8] makes advantage of writing elements directly in their final places and has a complexity of \( O(n) \) (while comparison-based sorting algorithms have a complexity of \( O(n \log n) \)); but its parallelization is restricted by the range of values.

In this paper, we introduce the Structure-Aware Parallel Counting (SAPCo) Sort algorithm that exploits the skewed degree distribution of real-world graphs to accelerate degree-ordering. The evaluation of SAPCo in comparison to state-of-the-art sample sort and radix sort algorithms shows that SAPCo is 1.7–4.0 times faster.

II. BACKGROUND: COUNTING SORT

For sorting an input array containing \( n \) integer values in range \([0, R]\), sequential counting sort performs 3 steps:

Step 1. The input array is read and a counters array of length \( R \) is used to count the number of times different unique values occur in the input array.

Step 2. To specify the insertion point of the first occurrence of unique values in the output array, the prefix sum of counters is calculated and stored in the Insertion Points (IP) array. If value \( v \) appears \( r = \text{counters}_v \) times in the input array, \( IP \) reserves space for all \( r \) repetitions of \( v \) as \( IP_{v+1} = IP_v + \text{counters}_v \).

Step 3. The input array is read again and values are placed in the output array using \( IP \): After reading an element with value \( v \), it is written on an index of the output array that is identified by the insertion point, \( IP_v \), and \( IP_v \) is incremented to be ready for the next value.

As the counters array is not needed after Step 2, its allocated memory is used for \( IP \); however, we use different names to mention distinct usages and contents.

Parallel counting sort, is performed in two ways:

I. Shared IP: Threads read partitions of the input array and atomically increment the shared counters (Step 1), \( IP \) is calculated by parallel prefix sum (Step 2), and threads read the input array and use atomic memory accesses to get an insertion point from the shared \( IP \) (Step 3). To accelerate Step 1, per-thread counters can be used to avoid atomic memory accesses.

II. Private IP: The input array is divided into partitions and per-partition counters arrays are allocated. Then, partitions are read by threads and their private counters are set (Step 1). A global counters array is accumulated by private counters, and the global \( IP \) is identified by parallel prefix sum. The global \( IP \) and the partitions’ counters are used to identify the private \( IP \) of each partition (Step 2). The input array is read again and private \( IP \) are used to identify the index required for writing to the output array (Step 3).

The first approach, shared IP, suffers from a great number of atomic memory accesses during Step 3.

The applicability of the second approach, private IP, depends on the number of partitions (which is affected by number of cores and also affects the load balance) and the range of values, \( R \). For \( p \) partitions, the memory complexity is \( O(Rp) \).

© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

978-1-6654-5954-9/22 ©2022 IEEE
https://doi.org/10.1109/ISPASS55109.2022.00015
TABLE I: Evaluation of sorting algorithms: counting sort with Shared IP (“Cnt. Sh.”) and Private IP (“Cnt. Pr.”), IPS\(^2\)Ra (radix sort), IPS\(^4\) o (sample sort), and SAPCo - “Memory Accesses” and “HW Instructions” are divided by number of elements (|V|) - “Memory Accesses” are load and store instructions - Failed attempts are shown by dash.

| Dataset         | Type | | \(|V|\) | Max. Degree | Performance (Milliseconds) | Memory Accesses | HW Instructions |
|-----------------|------|---|--------|------------|---------------------------|-----------------|-----------------|
| GB Roads        | RN   | 7 | 7      | 1.334      | 10.9 25.6 22.5 11.0      | 13.8 16.7 12.1 | 52.0 47.2 34.6 |
| US Roads        | RN   | 23.9 | 8 | 1,334 | 10.9 25.6 22.5 11.0      | 13.8 16.7 12.1 | 48.9 46.1 34.3 |
| Pokec           | SN   | 1.6 | 13.7 K | 1,130 | 166 55.9 21.4 18.7      | 32.4 23.4 12.4 | 83.6 65.7 36.2 |
| War Wikipedia   | KG   | 2.1 | 1.14 M | 119 | 573 12.6 4.3 4.8      | 38.3 30.4 16.0 | 99.9 80.2 52.6 |
| LiveJournal Links | SN | 5.2 | 15.0 K | 210 | 19.2 10.0 6.4 5.5      | 29.2 27.4 13.1 | 76.9 74.6 39.6 |
| LiveJournal     | SN   | 7.5 | 1.05 M | 315 | 799 23.1 9.8 8.9      | 35.4 33.8 13.2 | 90.6 87.0 39.8 |
| Twitter 2010    | SN   | 21.3 | 422 K | 1,130 | 166 55.9 21.4 18.7      | 32.4 23.4 12.4 | 83.6 65.7 36.2 |
| Twitter         | SN   | 28.5 | 278 K | 1,324 | 213 73.1 28.6 20.5      | 34.6 29.3 12.3 | 87.7 76.7 35.6 |
| Twitter-MPI     | SN   | 41.7 | 770 K | 1,687 | 422 103 40.8 37.5      | 30.9 28.3 12.3 | 81.9 75.2 35.7 |
| SK-Domain       | WG   | 50.6 | 8.56 M | 2,286 | 6,120 130 50.1 33.8      | 31.8 26.3 12.6 | 82.3 70.8 36.4 |
| Friendster      | SN   | 65.6 | 3,615 | 2,765 | 39.4 122 65.0 35.7      | 30.9 29.0 12.1 | 81.0 77.6 34.7 |
| Web-CC12        | WG   | 89.1 | 2.33 M | 4,226 | 1,369 228 82.5 55.9      | 32.6 25.5 12.2 | 83.6 69.5 35.2 |
| UK-Domain       | WG   | 105.2 | 975 K | 2,280 | 629 266 100 57.4      | 37.4 28.9 12.1 | 92.9 76.9 34.6 |
| UK-Delis        | WG   | 109.5 | 1.26 M | 4,649 | 984 276 109 56.7      | 33.0 27.8 12.1 | 85.1 73.8 34.6 |
| WebBase-2001    | WG   | 118.1 | 816 K | 5,591 | 783 296 117 54.4      | 30.1 24.6 12.1 | 79.8 66.0 34.4 |
| UK-Union        | WG   | 133.6 | 6.37 M | 5,511 | 3,478 335 134 66.6      | 35.3 31.7 12.2 | 89.3 81.1 34.8 |
| GSH 2015        | WG   | 988.5 | 58.8 M | 31,541 | 24,175 2,948 936 467      | 37.1 32.0 12.2 | 94.7 82.4 34.5 |
| ClueWeb09       | WG   | 1,685 | 6.44 M | 86,988 | 5,336 4,203 1,725 781      | 28.7 25.1 12.1 | 78.6 66.6 34.4 |
| WDC 2014        | WG   | 1,725 | 45.7 M | 87,792 | 27,643 5,744 1,732 679      | 36.9 25.2 12.1 | 95.3 67.0 34.3 |
| WDC 2012        | WG   | 3,564 | 95.0 M | 151,382 | 11,021 12.0 9.9 8.9      | 43.4 30.7 12.1 | 106.5 79.3 34.3 |

small \( R, p \) can be large enough to keep all processors busy; however, that is not the case for degree-ordering of real-world graphs where \( R \) may reach 95 million (Section IV). Moreover, Step 2 (merging private counters and calculating private IP) has a time complexity of \( O(Rp) \).

III. MOTIVATION

In power-law graphs, the number of low-degree vertices are exponentially greater than high-degree vertices. Consequently, in degree-ordering of these graphs, the input array has a very small number of High-Degree Vertices (HDVs) and a huge number of Low-Degree Vertices (LDVs).

As a result, when traversing the input array, the very small indices of counters or IP are accessed frequently; but, the greater indices are rarely accessed.

Since HDVs are rare and have a wide range of values, it is more efficient to save memory and time by allocating a shared memory array for HDVs and using atomic memory accesses to protect it from concurrent accesses of threads processing different partitions. In contrast, LDVs are frequent and in a short range. So, it is more efficient to assign per-partition private memory for them to accelerate their accesses that form almost all of the memory accesses.

IV. SAPCo SORT ALGORITHM

Step 1. We identify the maximum degree of the graph to set \( R = max\_degree + 1 \). We set a threshold between LDVs and HDVs: \( tsld = min(1000, 0.5 * R) \). We set the number of partitions to \( 64 * \#threads \) and assign a private counters (\( pcounters \)) array of size \( tsld \) for each partition. We also create a global counters (\( gcounters \)) array of size \( R \).

Step 2. Threads process elements in each partition of the input array. For an element with value \( v \), if \( v < tsld \), \( pcounters_v \) is incremented; otherwise, \( gcounters_v \) is atomically incremented.

Step 3. For each value \( 0 \leq v < tsld \), the sum of \( pcounters_v \) of different partitions is calculated and stored in the \( gcounters_v \). By applying prefix sum on the \( gcounters \), the Global Insertion Points (GIP) array is identified. Then, by using GIP and \( pcounters \), Private Insertion Points (PIP) arrays of LDVs of partitions are identified.

Step 4. The final pass over partitions of the input array is performed by threads. When reading a value \( v \), if \( v \) is a LDV, PIPs of the partition identifies the insertion point in the output and PIPs is incremented. If \( v \) is a HDV, the GIP identifies the insertion point in the output array and atomically is increased by one.

V. EVALUATION

Table I shows the real-world graph datasets from [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], and [28]. Graph types are Road Network (RN), Social Network (SN), Web Graph (WG), and Knowledge Graph (KG). Column 3 of Table I shows the numbers of vertices of graph (|V|) in millions (which specifies the number of elements in the input array, \( n \)). Column 4 of Table I “Max. Degree”, shows the maximum in-degree of graphs (which specifies the value of \( R \) in Section IV).

We use a machine with 2 Intel Xeon Gold 6126 sockets; in total, 24 cores, 24 threads, and 1.5TB memory.
We implemented SAPCo in the C language using the OpenMP API [31], libnuma, and papi [32] libraries. The gcc-9.2 was used as compiler with -o3 flag.

We evaluate SAPCo in comparison to counting sort, IPS^2Ra radix sort (commit 18795bb), and IPS^4o sample sort (commit d7a74ab).

Table I shows that SAPCo is, on average, 1.7× faster than IPS^o, 4.0× faster than IPS^2Ra, 33.5× faster than counting sort with private IP, and 71.5× faster than counting sort with a shared IP. Table II also shows that SAPCo, on average, performs 12.6 memory accesses per vertex while, IPS^o requires 27.4 accesses. Moreover, SAPCo requires 37.1 hardware instructions per vertex, on average while, IPS^4o requires 72.8 instructions.

VI. CONCLUSION

In this paper, we introduced the SAPCo sort algorithm that optimizes degree-ordering of real-world graphs with power-law degree distribution. SAPCo dedicates per-partition private arrays for low values (i.e., low-degree vertices) that are frequent while, using a global shared array for higher values (i.e., high-degree vertices) that are rare. In this way, SAPCo provides 1.7–4.0 times speedup in comparison to state-of-the-art sample sort and radix sort algorithms.

CODE AVAILABILITY

Source code repository and further discussions relating to this paper are available online in https://blogs.qub.ac.uk/GraphProcessing/SAPCO-Sort-Optimizing-Degree-Ordering-For-Power-Law-Graphs/.

ACKNOWLEDGEMENTS

This work is partially supported by the High Performance Computing center of Queen’s University Belfast and the Kelvin-2 supercomputer (EPSRC grant EP/T022175/1) and by DiPET (CHIST-ERA project CHIST-ERA-18-SDCDN-002, EPSRC grant EP/T022345/1). First author is supported by a scholarship of the Department for the Economy, Northern Ireland and Queen’s University Belfast.

REFERENCES