Fast Parallel Skew and Prefix-Doubling Suffix Array on the GPU

Leyuan Wang[1] Sean Baxter[2] John D. Owens[1]
[1]University of California, Davis
[2]D. E. Shaw Research, New York

Overview

We implement two classes of suffix array construction algorithms on the GPU. The first, skew, makes algorithmic improvements to the previous work of Deo and Keely to achieve a speedup of 1.45x over their work. The second, a hybrid skew and prefix-doubling implementation, is the first of its kind on the GPU and achieves a speedup of 2.3–4.4x over Osipov's prefix-doubling and 2.4–7.9x over our skew implementation on large datasets. Our implementations rely on two efficient parallel primitives, a merge and a segmented sort. We use our implementations in a Burrows-Wheeler transform (BWT), a BWT-based lossless data compression application (bzip2) and a parallel FM index for pattern searching. [1]

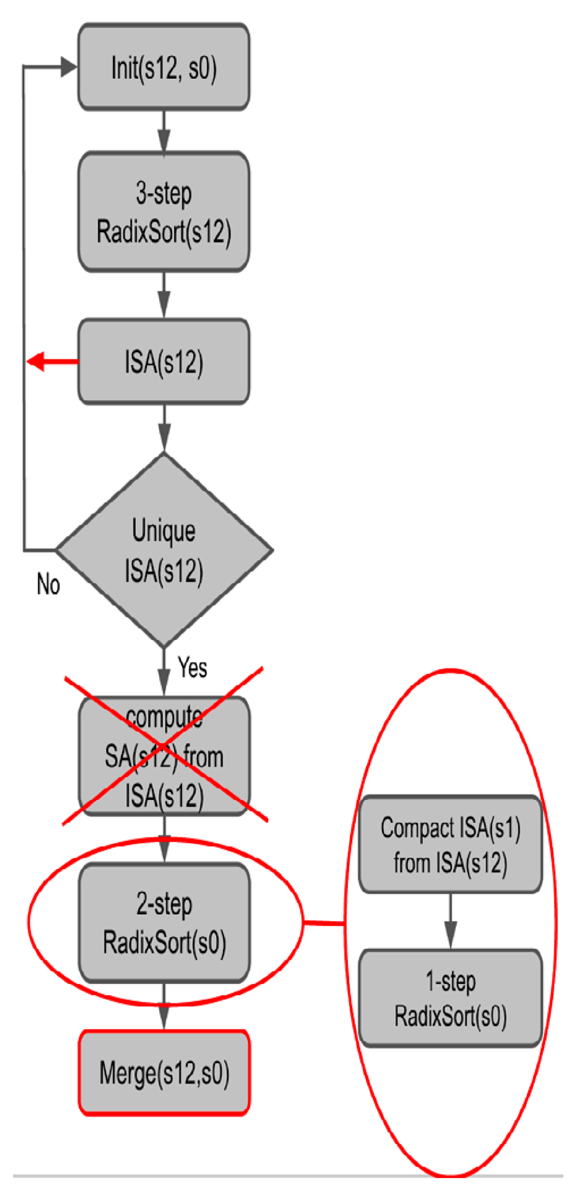
Introduction

The suffix array (SA) of a string is the sorted set of all suffixes of the string. This data structure is used in a broad spectrum of applications, including data compression, bioinformatics, and text indexing. The recent explosion in data sizes and the emergence of commodity data-parallel processors motivate efficient parallel implementations of suffix array construction algorithms (SACAs). We address the problem by designing, implementing, and comparing two different formulations of SACAs on NVIDIA GPUs. The first, skew, incorporates several algorithmic optimizations over Deo and Keely's implementation. The second, a hybrid non-recursive skew/prefix-doubling SACA overcomes the parallelization challenges identified by Deo and Keely and performs much better than Osipov's plain prefix-doubling. Comparing our two implementations, we demonstrate that a recursive doubling-like formulation can be efficiently mapped to GPUs and that our hybrid implementation in general produces the fastest SACA implementation on GPUs.

Parallel Skew Algorithm

Algorithmic Improvements:

- Only compute ranks (ISA(s12)) when the suffixes are not fully sorted, thus saves us from computing SA(s12) from ISA(s12) at the end of recursion;
- Faster 1-step radix sort + trivial compact operation replacing 2-step radix sort.



The above optimizations allow us to only use 2/3 of Deo and Keely's memory bandwidth in the recursive part and 3 fewer memory transactions in the last round.

- A more load-balanced merge primitive based on the merge path technique of Green et al.

A Hybrid of Skew/Prefix-doubling

Limitations of parallel skew:

- Recursive, cannot parallelize across iterations;
- Have to re-sort some fully sorted suffixes in order to keep the recursive routine.

A better fit for modern GPU architectures:

- Keep the first step and the final merge of skew;
- After reducing the string size by 2/3, we transition to prefix-doubling;
- Sort by $(ISA[SA[i]+\delta], ISA[SA[i]+2\delta])$ pairs using our high-performance *segmented sort*;
- Filter out fully sorted suffixes at the end of each iteration.

Improvements over plain prefix-doubling:

- First step's 3-character radix sort is faster than a 4-character one;
- *Segmented sort* has better locality than radix sort across global memory;
- Induction step is cheaper than radix sort in sorting the remaining 1/3 suffixes.

Our proposed two classes of parallel SACAs

```
skew-SA (int* T, int* SA, int length)
Initialize Mod12() // form triplet s12, s0
RadixSort (s12) // LSD radix sort 1st char
RadixSort (s12) // LSD radix sort 2nd char
RadixSort (s12) // LSD radix sort 3rd char
if (!allUniqueRanks) then
  lexicRankOfTriplets (s12)
  skew-SA ()// Recurse
  storeUniqueRanks()
Compact (ISA[12]) // compact out the order of
    ISA[1]
RadixSort (s0)
Merge (s0,s12)
```

```
Initialize Mod12() // form triplets
RadixSort (s12) // 25-bit radix sort
ComputeRanks ISA[12]
size = 2/3 n, h = 6
while size > 0 do
  SegmentedSort
  (ISA[SA[i]+h],ISA[SA[i]+2h])
  Update ISA_2h and Compact SA_2h
  size = size of SA_2h
  h = h*2
Compact (ISA[12]) // compact out ISA[1]
RadixSort (s0)
Merge (s0,s12)
```

Figure 1: Left, our parallel skew description; right, our skew/prefix-doubling.

Results

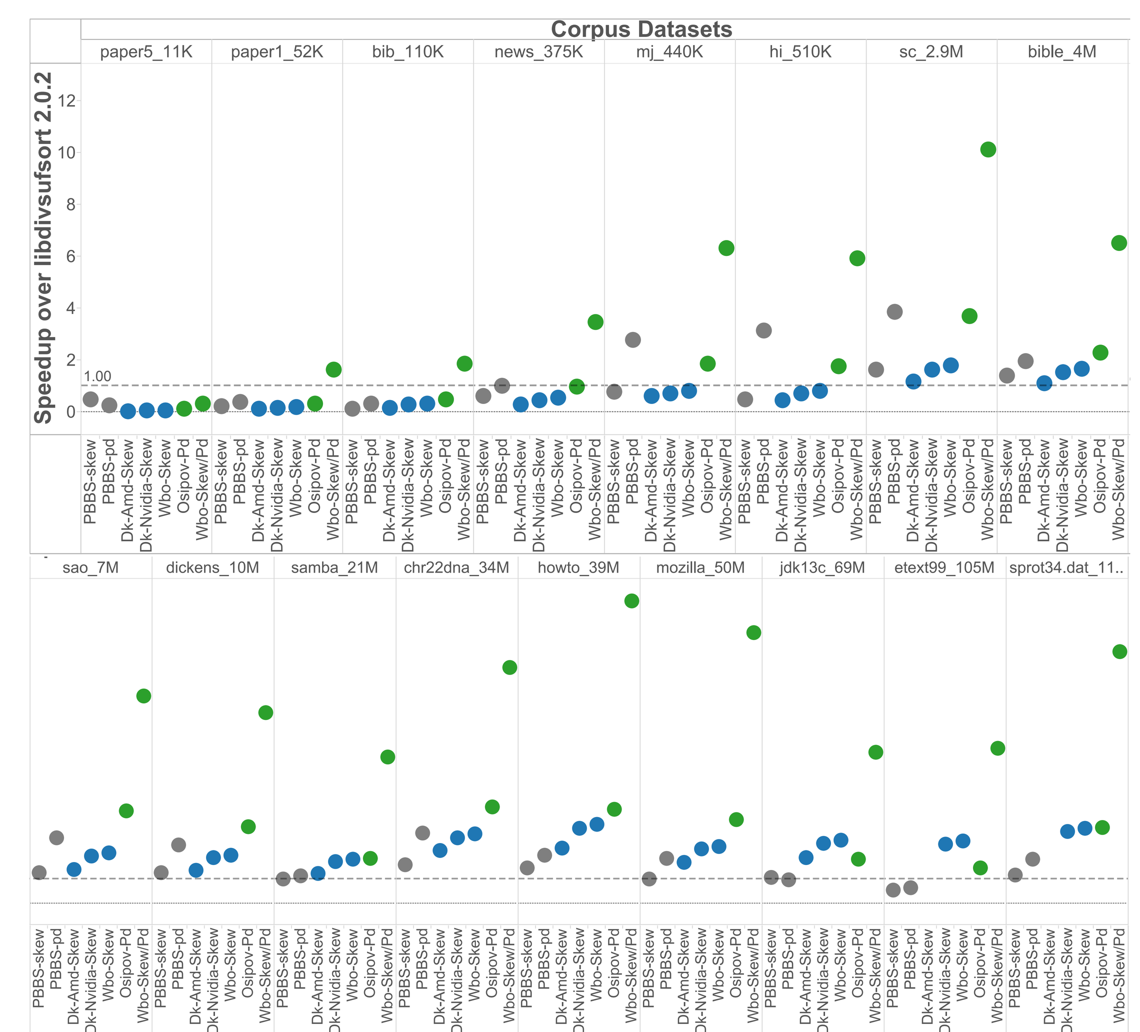


Figure 2: Speedups of seven suffix array construction implementations over corpus datasets. The CPU implementation libdivsufsort 2.0.2 is the baseline.

References

[1] L. Wang, S. Baxter, and J. Owens, "Fast parallel suffix array on the gpu," in *Euro-Par 2015: Parallel Processing*, 2015, vol. 9233, pp. 573–587.

Contact Information

Author's Email: leywang@ucdavis.edu

UC DAVIS ELECTRICAL AND COMPUTER ENGINEERING