Fast Parallel Suffix Array on the GPU

Leyuan Wang[1] Sean Baxter[2] John D. Owens[1]

University of California, Davis, CA, USA

D. E. Shaw Research, NY, USA

7th April 2016

Why Suffix Array?

- Suffix array is a simpler to construct, space- and cache-efficient alternative to suffix trees
- The SA data structure is used in a variety of applications, including string processing, computational biology, text indexing, and many more.

Fundamental Concepts

The Suffix Array (SA) and Inverse Suffix Array (ISA):

$$\text{SA}[i]{=}j \iff \text{ISA}[j]{=}i$$

The Burrows-Wheeler Transform (BWT):

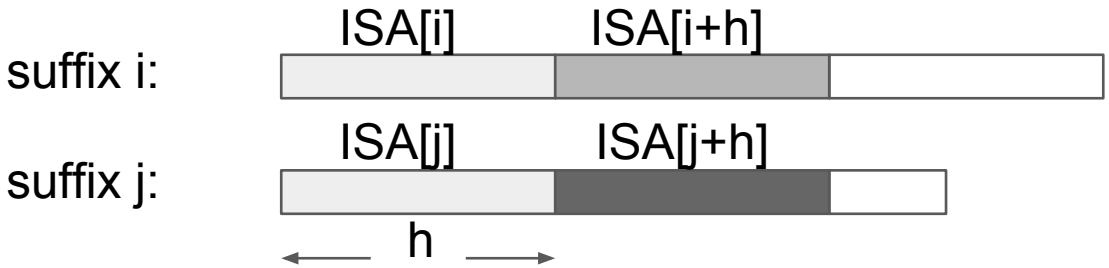
$$\text{BWT}[i] = \begin{cases} x[\text{SA}[i]-1] & \text{if SA}[i] > 0 \\ \$ & \text{if SA}[i] = 0 \end{cases}$$

input string: *banana$*

i	Suffix	Sorted Suffix	SA[i]	ISA[i]	Sorted Rotations	BWT[i]
0	banana$	$	6	4	$banana	a
1	anana$	a$	5	3	a$banan	n
2	nana$	ana$	3	6	ana$ban	n
3	ana$	anana$	1	2	anana$b	b
4	na$	banana$	0	5	banana$	$
5	a$	na$	4	1	na$bana	a
6	$	nana$	2	0	nana$ba	a

Suffix Array Construction Algorithms (SACAs)

Prefix-doubling $\mathcal{O}(n \log n)$
sorts the suffixes of a string by their prefixes, doubling the length of those prefixes every iteration.
Key idea: given an h-order of suffixes (suffixes are already sorted by their h-length prefixes), we can deduce their 2h-order in linear time.



Manber and Myers (MM)
Larsson and Sadakane (LS)
Osipov (osipov-pd) [1]

Challenges:
We can think of each iteration as producing a set of buckets that are dependent on the prefixes considered in that iteration. The number of buckets and the amount of work per bucket is irregular and data-dependent.

Suffix Array Construction Algorithms (SACAs)

Recursive Algorithms $\mathcal{O}(n)$
choose and recursively sort a subset (typically 2/3 or fewer) of the suffixes;
use the order of the sorted subset to infer the order of the remaining subset;
merge the two sorted subsets to get the order of the entire set.
Challenges: the recursion step
Kärkkäinen and Sanders (skew)
Deo and Keely (dk-amd-skew) [2]

Suffix Array Construction Algorithms (SACAs)

Induced Copying Algorithms $\mathcal{O}(n)$
is a non-recursive approach that uses already-sorted suffixes to quickly induce a complete ordering of all suffixes

Two-stage induced copying

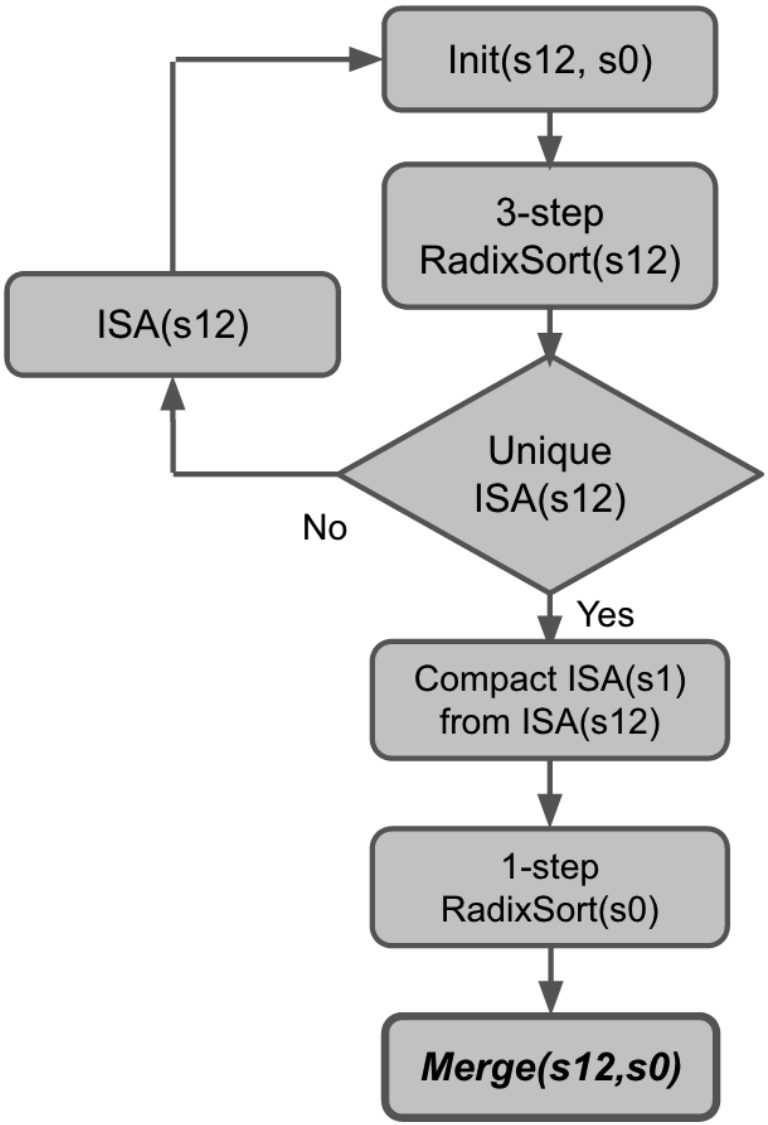
Pure induced copying (SA-IS)

Challenges: the inherent algorithmic efficiency of its CPU implementations is purely sequential, whether we can translate it into the GPU domain.

Our Contributions

- Propose and implement two massively parallel approaches on the GPU based on two classes of SACAs.
- Parallel **skew** achieves a speedup of 1.45× over Deo and Keely's work.
- A hybrid of **skew** and **prefix-doubling** (the first of its kind on the GPU) achieves a speedup of 2.3–4.4x over Osipov's prefix-doubling and 2.4–7.9x over our *skew* implementation.
- We theoretically analyze the two formulations of SACAs, show performance comparisons on a large variety of practical inputs.
- We integrate our **skew/prefix-doubling** hybrid into our GPU implementations of the Burrows-Wheeler transform (BWT) with a throughput of 132.5M characters/s and an FM-index-based pattern search application with a throughput of 77M characters/s.

Parallel Skew



Extract suffixes with starting position i where $i \bmod 3 \not\equiv 0$ (s12) and suffixes with starting position j where $j \bmod 3 \equiv 0$ (s0) from an input string;

Launch a 3-step least significant digit (LSD) radix sort (from Merrill's cub library);

Compare each triplet against its predecessor, store a flag of 1 whenever they are unequal;

Compute a prefix-sum on the list of flags to get ISA(s12);

Filter out the order of the ranks of s1 (equivalent to ISA[s1]) from ISA[s12];

Efficient Merge Primitive

Challenges

Load-balancing: divide the two sorted inputs into independent chunks of equal sized work;

Memory coalescing: ensure that the outputs of each of those chunks of work are contiguous in the final merged output.

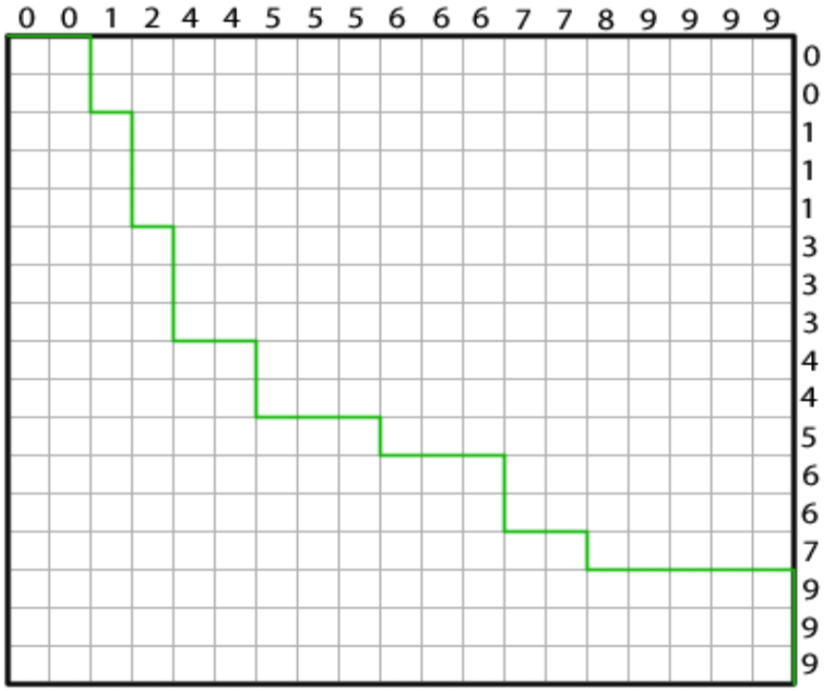
Solutions: identify split points

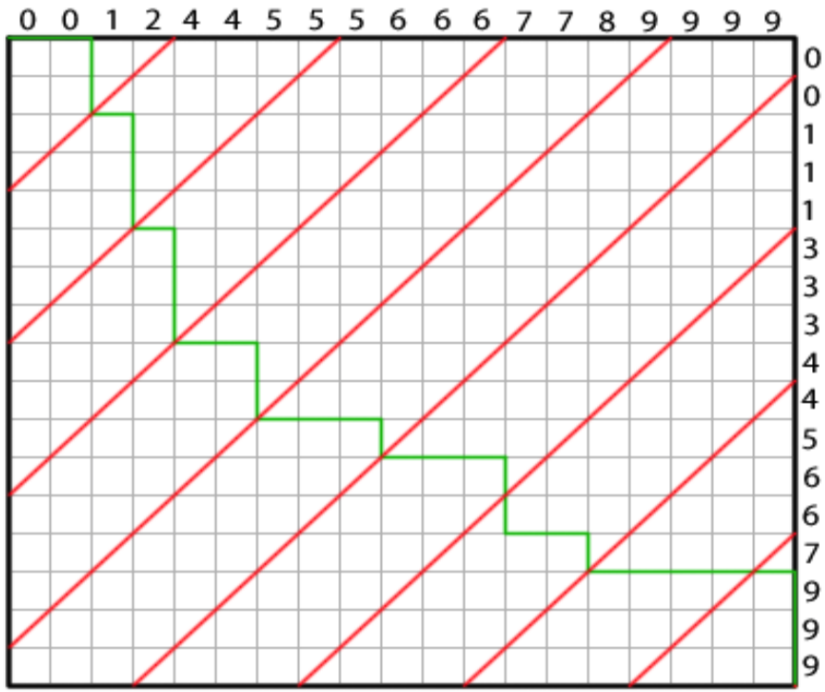
Use Merge Path [3] to transform a 2-D search to 1-D search along a diagonal that connects the two input arrays.

[0] Code is available at `http://nvlabs.github.io/moderngpu/merge.html`.

Efficient Merge Primitive

Merge Path

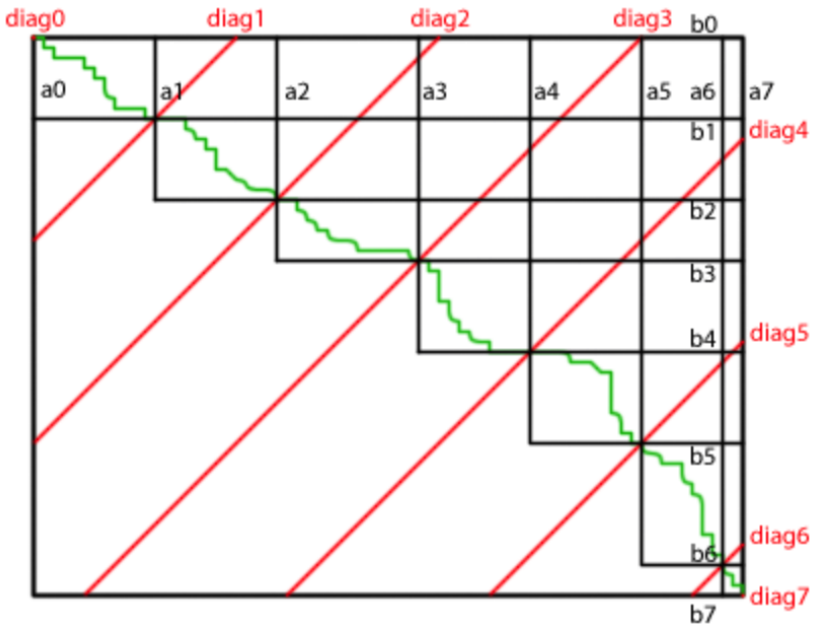


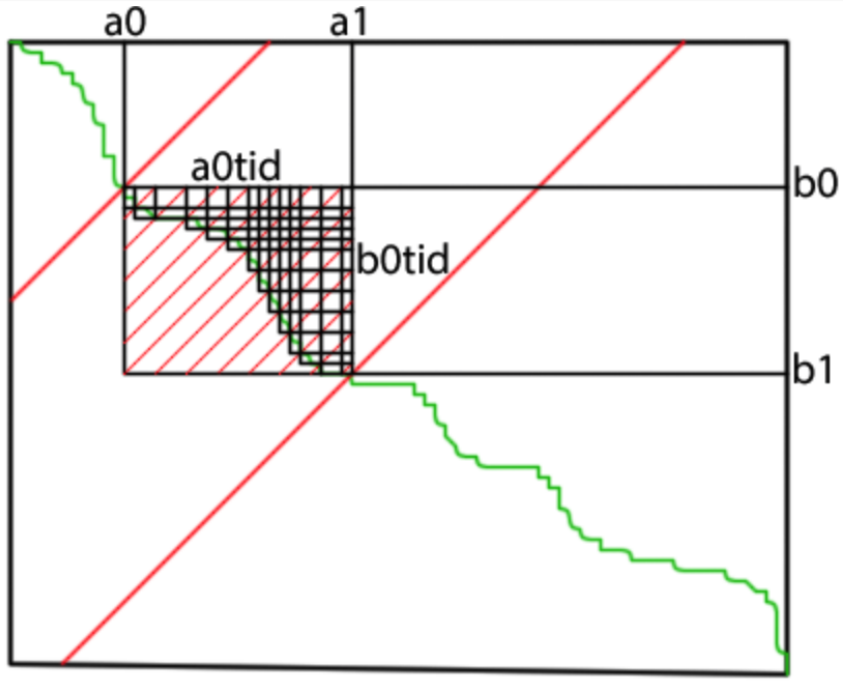


[0]Images obtained from `https://nvlabs.github.io/moderngpu/bulkinsert.html`.

Efficient Merge Primitive

Merge Path





[0]Images obtained from `https://nvlabs.github.io/moderngpu/merge.html`.

Limitations of Parallel Skew

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

Parallel Prefix-doubling

- Keep the first step of skew: reduce the string size by 2/3 and do 25-bit radix sort on 3-character substrings;
- Prefix-doubling: sort by (ISA[SA[i]+δ], ISA[SA[i]+2δ]) pairs using high-performance *segmented sort* and filter out suffixes that are fully sorted at the end of each iteration;
- Use the induction step of skew to induce the order of remaining 1/3 suffixes;
- Final merge of two sorted sequences.

Challenges: prefix-doubling has an irregular, data-dependent number of unsorted groups across phases; sort efficiently within each segment, even though the number of segments and their sizes are non-uniform and not known at compile time.

Segmented Sort [1]

Input: segments of unsorted items
Output: same lists of segments within which items are sorted
Challenges: variation in the size and number of segments; leverage the presence of segments but also work on all segments simultaneously.
Naive methods:
1. sort each segment one at a time
2. a full sort over all items
3. maintain segment IDs as the most significant bits of the key (to maintain segment stability) while choosing an appropriate sorting method for each individual segment.

[1] Code is available at `http://nvlabs.github.io/moderngpu` and described in `http://nvlabs.github.io/moderngpu/merge.html`.

Segmented Sort

1. Divide the input into equal-sized "blocks";
2. Launch "blocksorts" to sort within each block while maintaining segment order;
3. Use a sequence of iterative merge operations to get the final result.

Core: efficient merge in presence of segments

Key insight: During a merge of two contiguous lists, the only segment that is affected by the merge is one that spans the boundary between two blocks.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
m	***m***	***i***	***i***	***s***	*s*	*i*	*i*	***s***	***s***	***i***	***i***	***p***	*p*	*i*	*i*
(0	1	2	3)	(4	5	6	7)	(8	9	10	11)	(12	13	14	15)
(***i***	***i***	***m***	***m***)	(***s***	*i*	*i*	*s*)	(***i***	***i***	***s***	***s***)	(***p***	*i*	*i*	*p*)
(0	1	2	3	4	5	6	7)	(8	9	10	11	12	13	14	15)
(***i***	***i***	***m***	***m***	***s***	*i*	*i*	*s*)	(***i***	***i***	***p***	***s***	***s***	*i*	*i*	*p*)
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
i	***i***	***m***	***m***	***s***	*i*	*i*	*s*	***i***	***i***	***p***	***s***	***s***	*i*	*i*	*p*

Skew vs prefix-doubling

- Skew is essentially a "prefix tripling" technique, tripling the pace at which it samples its ranks each round;
- 2-integer segmented sort of prefix-doubling is much faster than the 3-integer radix sort of skew;
- In its radix sort, skew uses the most significant digit simply to get the suffix back in its original segment, which comes for free with prefix-doubling's segmented sort;
- Skew cannot drop fully-sorted suffixes for it needs to transform their ranks into the new coordinate system in which they will be sampled by the remaining unsorted suffixes, but with prefix-doubling, suffixes are ranked in the same coordinate system throughout the computation;
- Skew has a solid reduction ratio of 0.67, regardless of the data while prefix-doubling has a worst-case reduction ratio of 1.0 but has a more favorable reduction ratio on real-world text.

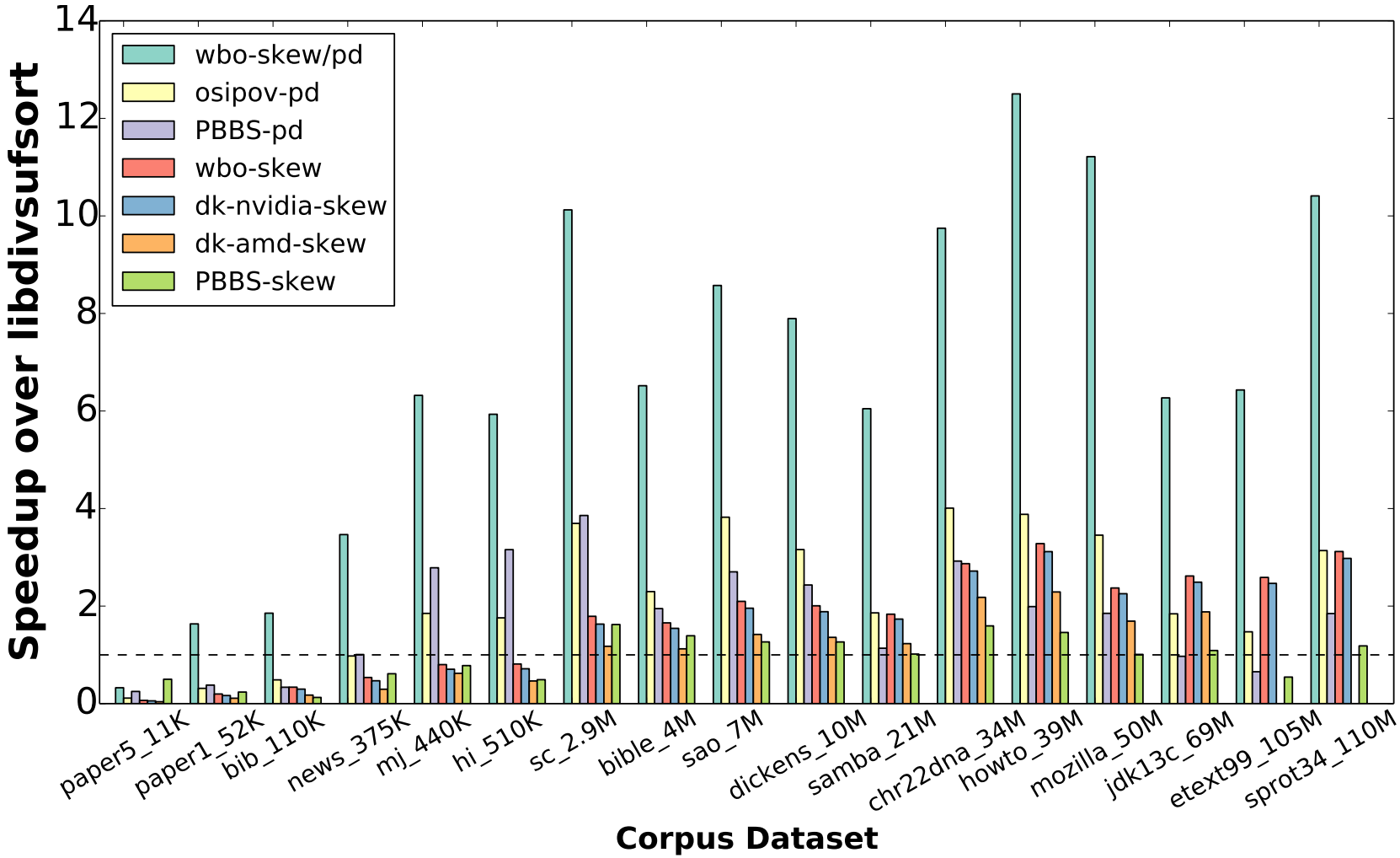


Figure 1: Speedup comparisons of five GPU suffix array construction implementations with two Intel Cilk Plus accelerated CPU implementations. Skew-based 2×, osipov-pd 4× and our wbo-skew/pd's 6-11× and 2.3–9.8× over PBBS-pd. The same SACA class track each other. wbo-skew 1.45× over dk-amd-skew, 1.1× over dk-nvidia-skew and 1.1–4.8× over PBBS-skew.

[1]Baseline: Mori Y. libdivsufsort 2.0.2. `https://github.com/y-256/libdivsufsort` 2015

Conclusions

- Of the three classes of suffix array construction algorithms, skew is perhaps the most suitable for brute-force methods when we began our work;
- The merge and segmented sort implementations in this paper make the difference between an SACA that is uncompetitive vs. an SACA that is best in class.
- Prefix-doubling with the efficient segmented sort primitive turns out to be a better fit for modern GPU architectures.
- We expect that the next frontier in GPU SACAs will be tackling the third class of SACAs—induced copying.

Original paper [4]:

http://link.springer.com/chapter/10.1007/978-3-662-48096-0_44

Acknowledgments

We acknowledge Mrinal Deo, Vitaly Osipov and Jacopo Pantaleoni for sharing their original data for comparisons.

Thanks Owens group for good advice on implementation and feedback on early drafts of the paper.

We appreciate the funding support of the National Science Foundation under grants OCI-1032859 and CCF-1017399, and UC Lab Fees Research Program Award 12-LR-238449.

[2] Our skew implementation, BWT and BWT-based lossless data compression can be found in current version of CUDA Data Parallel Primitives Library (CUDPP2.2) http://cudpp.github.io/

[3] Contact Info: leywang@ucdavis.edu

References


Osipov V. Parallel suffix array construction for shared memory architectures. *Proceedings of the 19th International Conference on String Processing and Information Retrieval*, SPIRE'12, Springer-Verlag, 2012; 379–384, doi:10.1007/978-3-642-34109-0_40.


Deo M, Keely S. Parallel suffix array and least common prefix for the GPU. *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, 2013; 197–206, doi:10.1145/2442516.2442536.


Green O, McColl R, Bader DA. GPU merge path: A GPU merging algorithm. *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, 2012; 331–340, doi:10.1145/2304576.2304621.


Wang L, Baxter S, Owens JD. Fast parallel suffix array on the GPU. *Euro-Par 2015: Parallel Processing*, *Lecture Notes in Computer Science*, vol. 9233, Träff JL, Hunold S, Versaci F (eds.). Springer Berlin Heidelberg, 2015; 573–587, doi:10.1007/978-3-662-48096-0_44.