In-place Parallel Super Scalar Samplesort (IPS4o)

Michael Axtmann, Sascha Witt, Daniel Ferizovic, and Peter Sanders

Karlsruhe Institute of Technology, Karlsruhe, Germany

Presentation by: Jessica Zhu

Motivations

- Sorting is a fundamental subroutine
  - Speed is expected
  - Memory is a constraint
- Replace quicksort, a 50-year old algorithm

Quicksort

- O(n log(n)) work
- Parallelizable
- Avoids branch mispredictions
- Cache-efficient
- Almost in-place

In-place Parallel Super Scalar Samplesort

In-place Parallel Super Scalar Samplesort

5 13 8 1 3 2 11 60 8 14 15 9

In-place Parallel Super Scalar Samplesort

5 13 8 1 3 2 11 60 8 14 15 9

1 2 5 3 8 8 13 11 9 14 60 15

Recursion!

In-place Parallel Super Scalar Samplesort

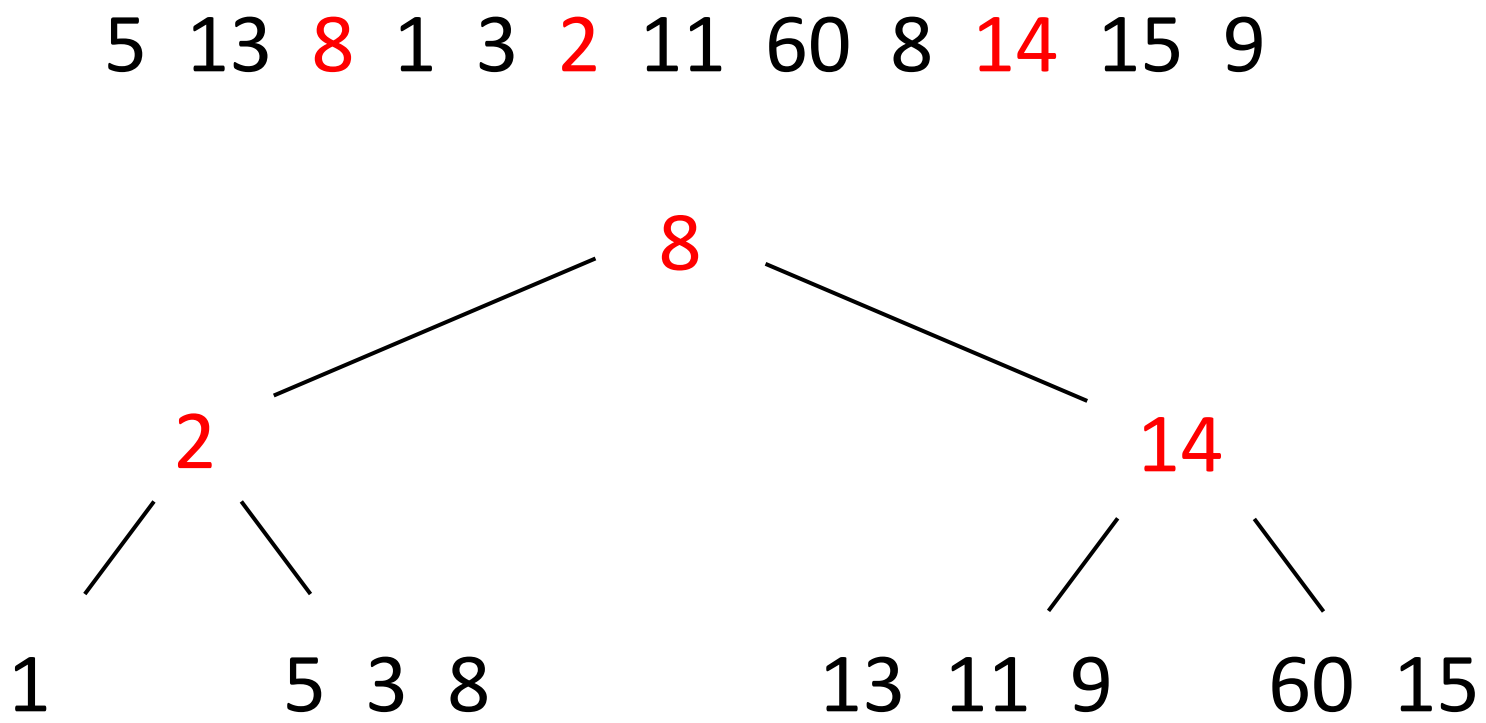
- O(n log(n)) work
- Parallelizable
- Cache-efficient

- Allows branch mispredictions
- Not in-place

In-place Parallel Super Scalar Samplesort

5 13 8 1 3 2 11 60 8 14 15 9

In-place Parallel Super Scalar Samplesort

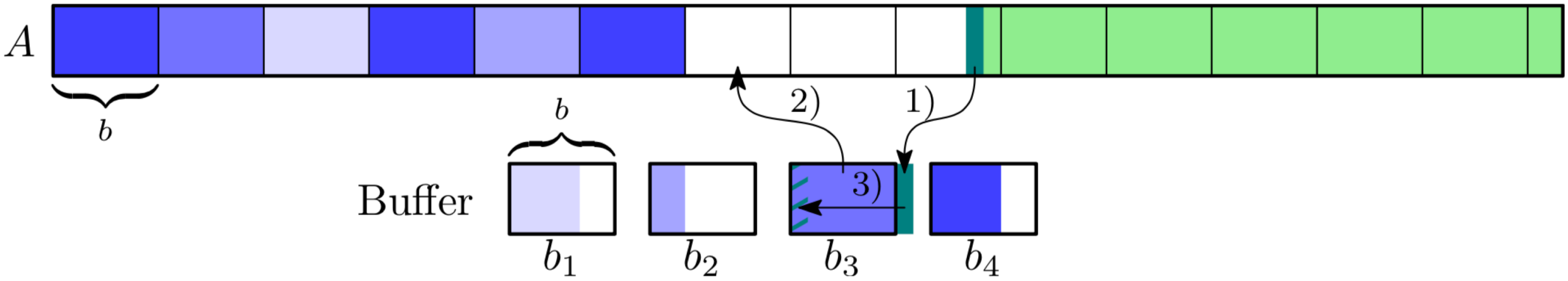


In-place Parallel Super Scalar Samplesort

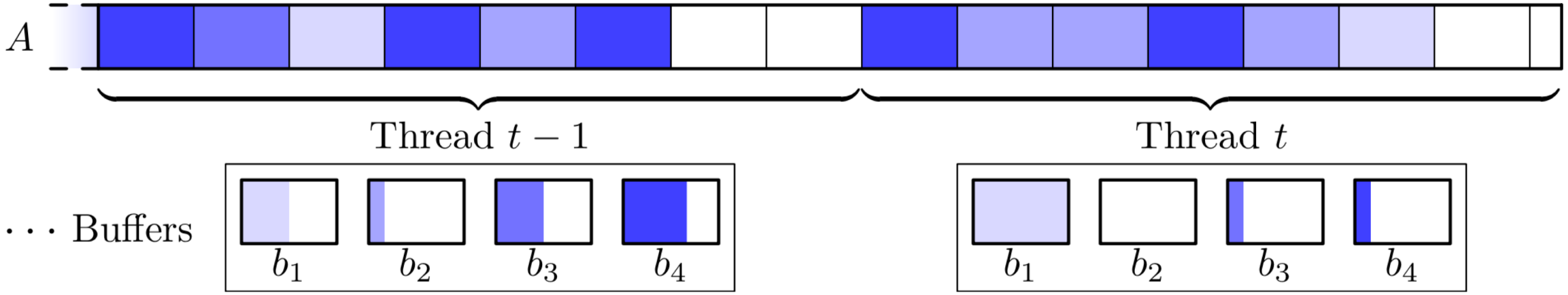
- O(n log(n)) work
- Parallelizable
- Cache-efficient
- Avoids branch mispredictions

- Not in-place

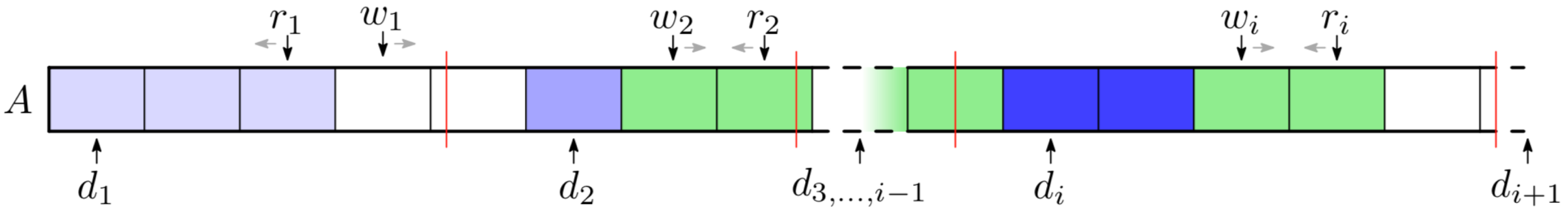
In-place Parallel Super Scalar Samplesort



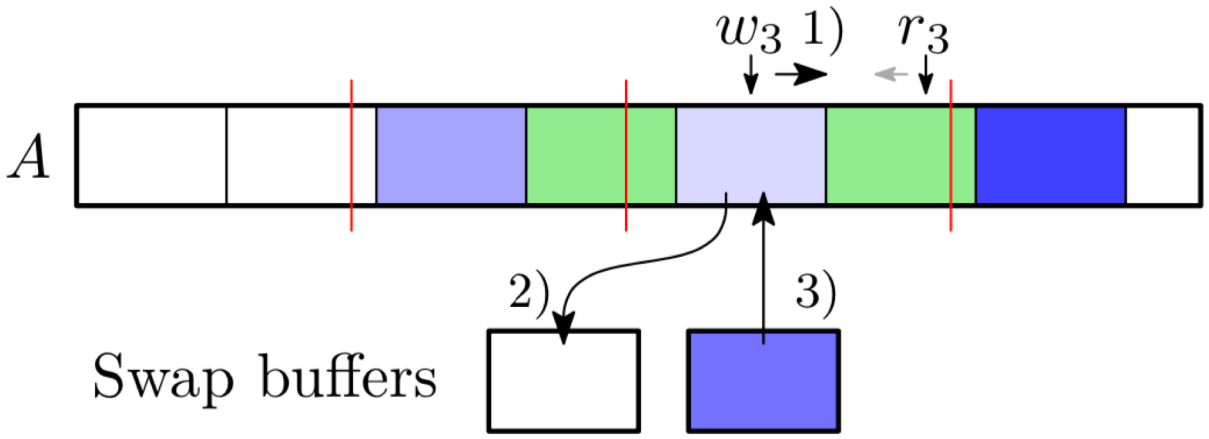
In-place Parallel Super Scalar Samplesort



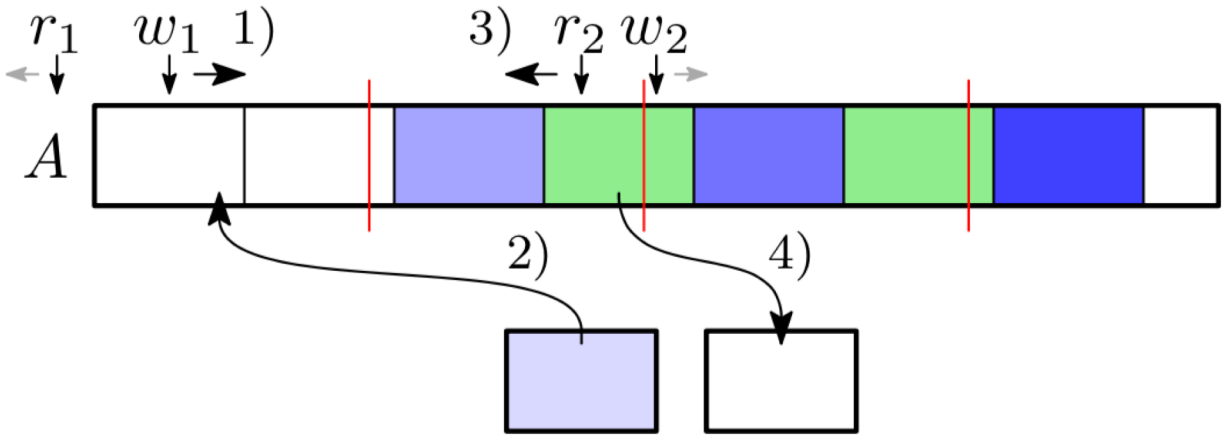
In-place Parallel Super Scalar Samplesort



In-place Parallel Super Scalar Samplesort

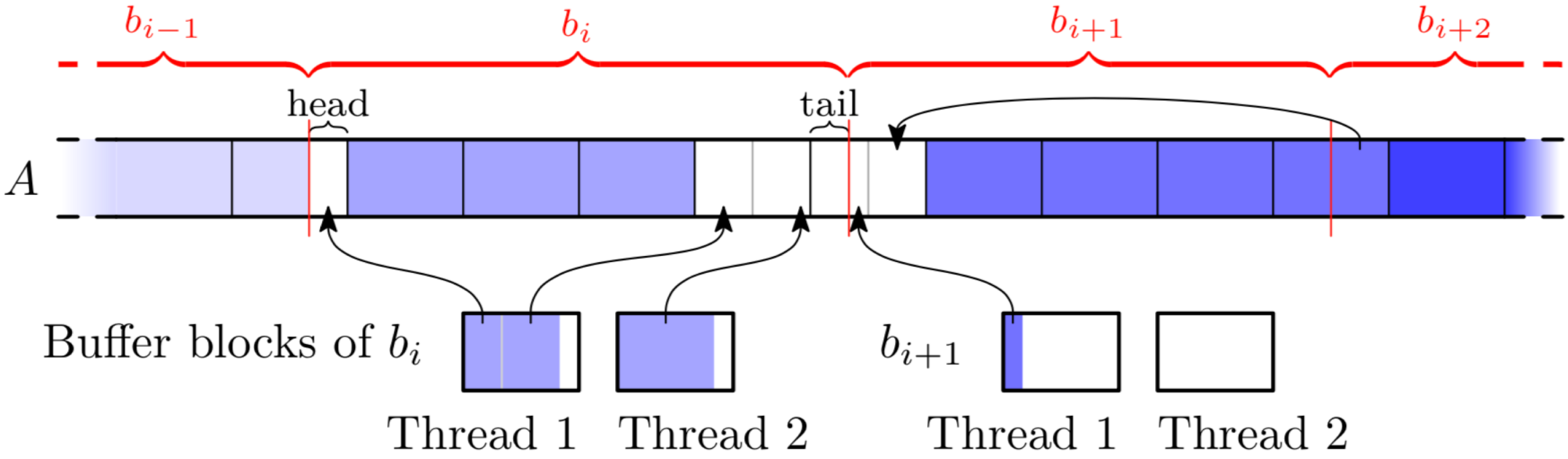


(a) Swapping a block into its correct position.



(b) Moving a block into an empty position, followed by refilling the swap buffer.

In-place Parallel Super Scalar Samplesort



In-place Parallel Super Scalar Samplesort

Edge Case: Many Identical Keys

Equality buckets

- Introduced if an element appears more than n/k times
- Skipped during recursion
- Implemented with only one extra comparison

In-place Parallel Super Scalar Samplesort

- O(n log(n)) work
- Parallelizable
- Cache-efficient
- Avoids branch mispredictions
- In-place

Theoretical Analysis

I/O Complexity with high probability:

$$\mathrm{O}\left(\frac{n}{tB}\log_k \frac{n}{n_0}\right)$$

Additional Space:

$$\mathrm{O}\left(kbt + \log_k \frac{n}{n_0}\right)$$

Becoming Strictly In-Place

$$O\left(kbt + \log_k \frac{n}{n_0}\right)$$

```
i := 1                                                    -- first element of current bucket
j := n + 1                                                -- first element of next bucket
while i < n do
    if j − i < n_0 then smallSort(a, i, j − 1);   i := j  -- base case
    else partition(a, i, j − 1)                           -- partition first unsorted bucket
    j := searchNextLargest(A[i], A, i + 1, n)             -- find beginning of next bucket
```

Experimental Results

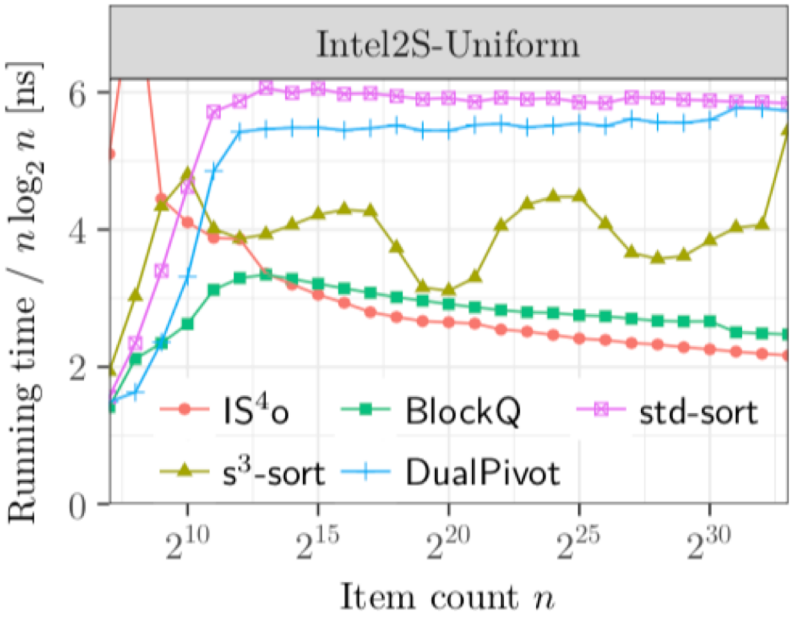


Figure 6 Running times of sequential algorithms on input distribution Uniform executed on machine Intel2S.

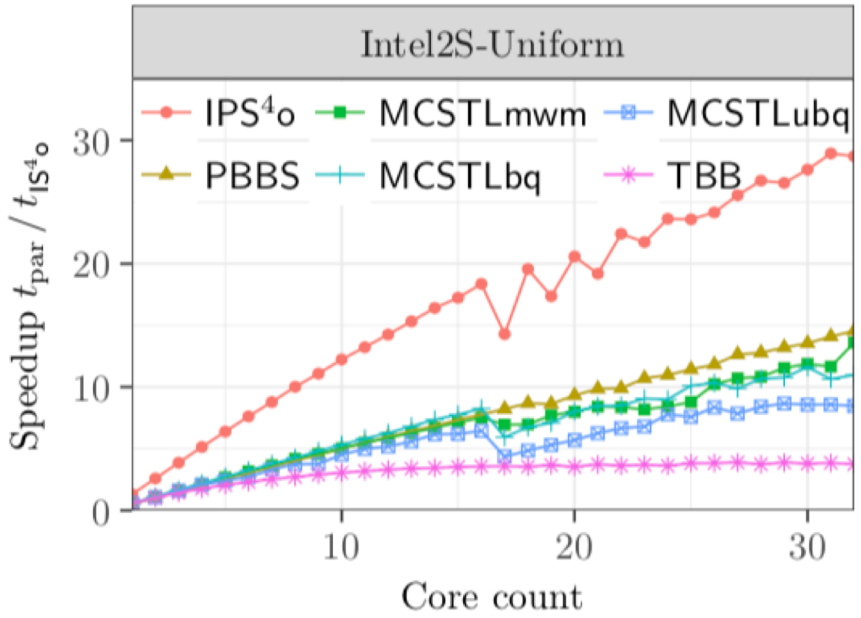
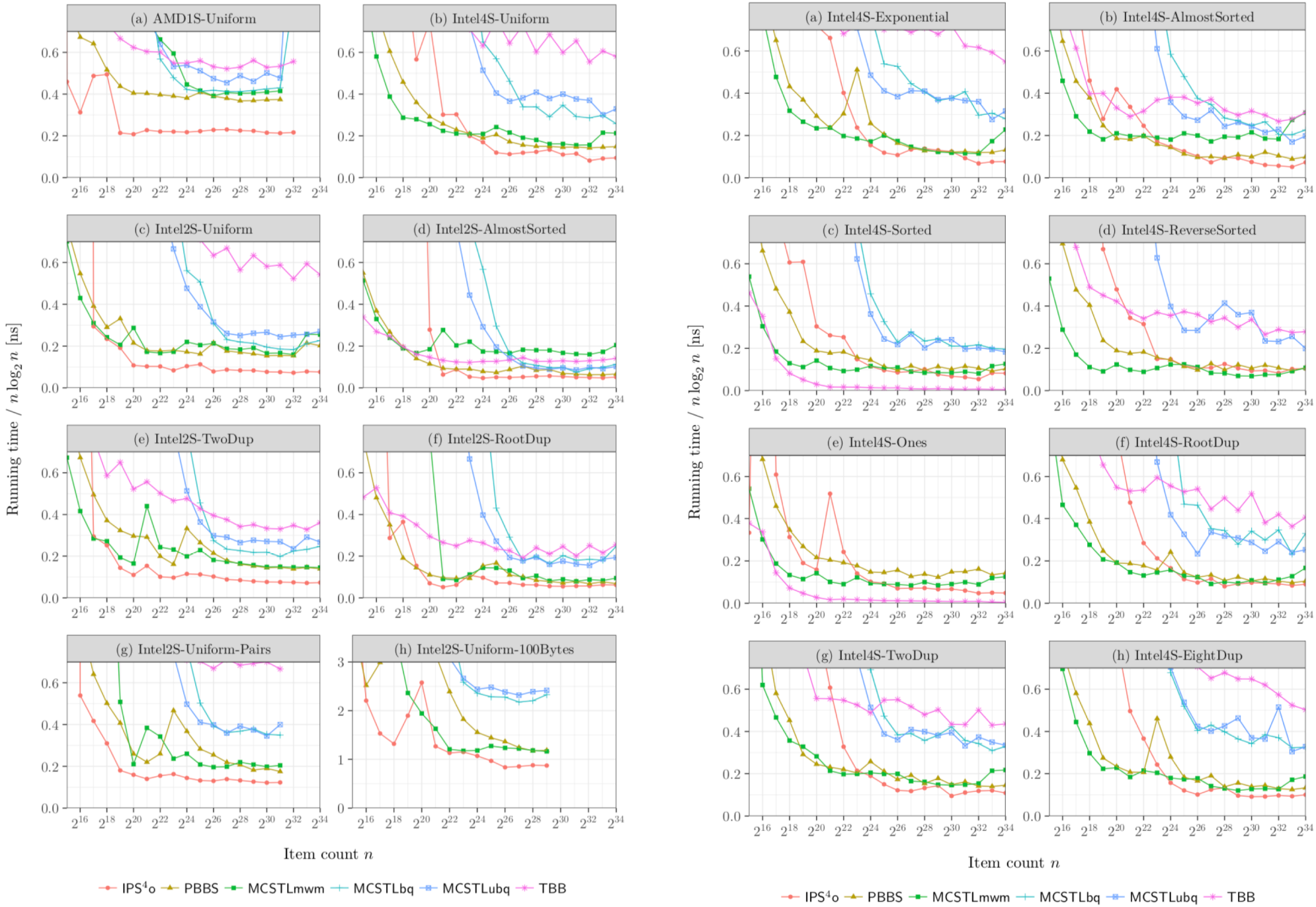
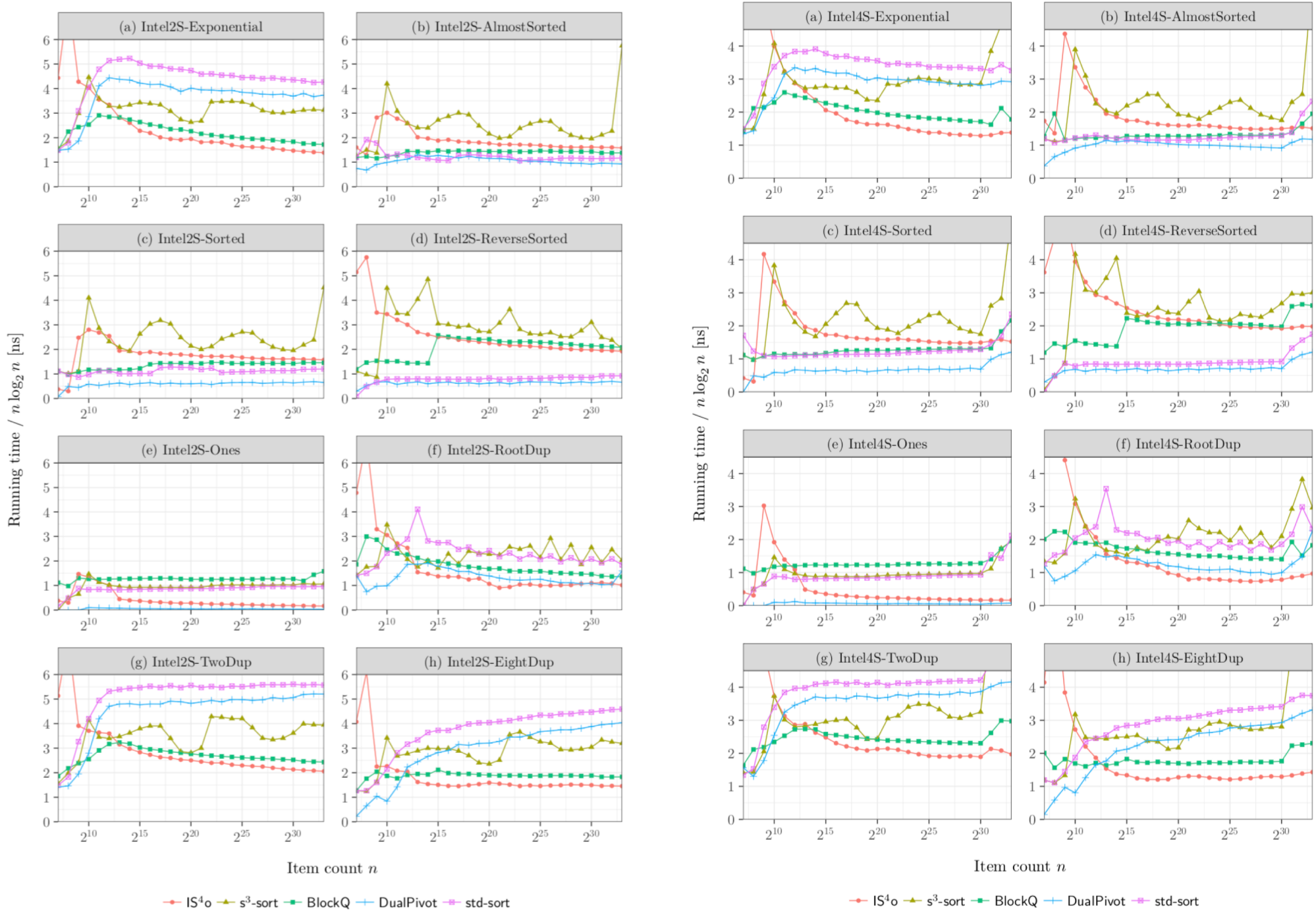


Figure 7 Speedup of parallel algorithms with different number of cores relative to our sequential implementation IS^{4}o on Intel2S, sorting 2^{30} elements of input distribution Uniform.


(a) AMD1S-Uniform
(b) Intel4S-Uniform
(c) Intel2S-Uniform
(d) Intel2S-AlmostSorted
(e) Intel2S-TwoDup
(f) Intel2S-RootDup
(g) Intel2S-Uniform-Pairs
(h) Intel2S-Uniform-100Bytes
(a) Intel4S-Exponential
(b) Intel4S-AlmostSorted
(c) Intel4S-Sorted
(d) Intel4S-ReverseSorted
(e) Intel4S-Ones
(f) Intel4S-RootDup
(g) Intel4S-TwoDup
(h) Intel4S-EightDup
Running time / $n \log_2 n$ [ns]
Item count n
IPS^{4}o
PBBS
MCSTLmwm
MCSTLbq
MCSTLubq
TBB


(a) Intel2S-Exponential
(b) Intel2S-AlmostSorted
(c) Intel2S-Sorted
(d) Intel2S-ReverseSorted
(e) Intel2S-Ones
(f) Intel2S-RootDup
(g) Intel2S-TwoDup
(h) Intel2S-EightDup
(a) Intel4S-Exponential
(b) Intel4S-AlmostSorted
(c) Intel4S-Sorted
(d) Intel4S-ReverseSorted
(e) Intel4S-Ones
(f) Intel4S-RootDup
(g) Intel4S-TwoDup
(h) Intel4S-EightDup
Running time / $n \log_2 n$ [ns]
Item count n
IS^{4}o
s^3-sort
BlockQ
DualPivot
std-sort

Machine	Algo	Competitor	Uniform	Exponential	Almost	RootDup	TwoDup
Intel2S	IS^{4}o	both	1.14	1.23	0.59	0.97	1.17
Intel4S	IS^{4}o	both	1.21	1.54	0.77	1.65	1.44
AMD1S	IS^{4}o	both	1.57	2.02	0.65	1.37	1.17
Intel2S	IPS^{4}o	in-place	2.54	3.43	1.88	2.73	3.02
		non-in-place	2.13	1.79	1.29	1.19	1.86
Intel4S	IPS^{4}o	in-place	3.52	4.35	3.62	3.19	2.89
		non-in-place	1.75	1.69	1.84	1.15	1.19
AMD1S	IPS^{4}o	in-place	1.57	3.18	1.81	2.37	2.02
		non-in-place	OOM	OOM	OOM	OOM	OOM

Table 1 The first three rows show the speedups of IS^{4}o relative to the fastest sequential in-place and non-in-place competitor on different input types executed on machine Intel2S, Intel4S, and AMD1S for $n = 2^{32}$. The last rows show the speedups of IPS^{4}o relative to the fastest parallel in-place and non-in-place competitor on different input types executed on different machine instances for $n = 2^{32}$. Measurements in cells labeled with OOM ran out of memory.

Strengths and Weaknesses

- Thorough comparisons of IPS4o to other sorting algorithms on different machines, inputs, input sizes, memory limitations
- Well structured paper that explained the algorithm clearly
- Appendix helpful for extra data and proofs
- Results seem promising
- Pseudocode would be helpful for implementation details
- Theoretical bounds rely on tight constraints to be valid
- Complex algorithm that has yet to be verified

Discussion Questions

- Will IPS4o replace Quicksort in certain situations? If not, what ultimately will?
- Can taking care of the edge case of many identical keys be applied to other sorting algorithms to provide the same speed up?