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

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

2

1 5 3 8

8

14

13 11 9 60 15
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

Thread $t-1$

Thread $t$

$b_1$ $b_2$ $b_3$ $b_4$

$b_1$ $b_2$ $b_3$ $b_4$
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

Buffer blocks of $b_i$

Thread 1   Thread 2   $b_{i+1}$   Thread 1   Thread 2
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:

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

Additional Space:

$$O\left(kbt + \log_k \frac{n}{n_0}\right)$$
Becoming Strictly In-Place

\[ O\left(kt + \log_k \frac{n}{n_0}\right) \]

\[ i := 1 \quad \text{--- first element of current bucket} \]
\[ j := n + 1 \quad \text{--- first element of next bucket} \]

while \( i < n \) do

  if \( j - i < n_0 \) then smallSort(a, i, j - 1); \( i := j \)

  else partition(a, i, j - 1)

  \( j := \text{searchNextLargest}(A[i], A, i + 1, n) \)

  \( j := \text{searchNextLargest}(A[i], A, i + 1, n) \)

  \( i := j \)

  \( j := n + 1 \)

  -- base case

  -- partition first unsorted bucket

  -- find beginning of next bucket
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.
<table>
<thead>
<tr>
<th>Machine</th>
<th>Algo</th>
<th>Competitor</th>
<th>Uniform</th>
<th>Exponential</th>
<th>Almost</th>
<th>RootDup</th>
<th>TwoDup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel2S</td>
<td>IS(^4)o</td>
<td>both</td>
<td>1.14</td>
<td>1.23</td>
<td>0.59</td>
<td>0.97</td>
<td>1.17</td>
</tr>
<tr>
<td>Intel4S</td>
<td>IS(^4)o</td>
<td>both</td>
<td>1.21</td>
<td>1.54</td>
<td>0.77</td>
<td>1.65</td>
<td>1.44</td>
</tr>
<tr>
<td>AMD1S</td>
<td>IS(^4)o</td>
<td>both</td>
<td>1.57</td>
<td>2.02</td>
<td>0.65</td>
<td>1.37</td>
<td>1.17</td>
</tr>
<tr>
<td>Intel2S</td>
<td>IPS(^4)o</td>
<td>in-place</td>
<td>2.54</td>
<td>3.43</td>
<td>1.88</td>
<td>2.73</td>
<td>3.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-in-place</td>
<td>2.13</td>
<td>1.79</td>
<td>1.29</td>
<td>1.19</td>
<td>1.86</td>
</tr>
<tr>
<td>Intel4S</td>
<td>IPS(^4)o</td>
<td>in-place</td>
<td>3.52</td>
<td>4.35</td>
<td>3.62</td>
<td>3.19</td>
<td>2.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-in-place</td>
<td>1.75</td>
<td>1.69</td>
<td>1.84</td>
<td>1.15</td>
<td>1.19</td>
</tr>
<tr>
<td>AMD1S</td>
<td>IPS(^4)o</td>
<td>in-place</td>
<td>1.57</td>
<td>3.18</td>
<td>1.81</td>
<td>2.37</td>
<td>2.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-in-place</td>
<td>OOM</td>
<td>OOM</td>
<td>OOM</td>
<td>OOM</td>
<td>OOM</td>
</tr>
</tbody>
</table>

**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?