In-place (Parallel) Super Scalar Samplesort

ESA · September 4, 2017
Michael Axtmann, Sascha Witt, Daniel Ferizovic, Peter Sanders
Parallel External Memory Model

- $t$ CPUs
- Each CPU has a private cache of size $M$
- Cache lines of size $B$
- Data transfers in cache lines from and to main memory
- One I/O: One simultaneous cache line transfer per thread
- I/O efficient sorting:
  $\Theta \left( \frac{n}{tB} \log \frac{M}{B} \frac{n}{B} \right)$ I/Os
Super Scalar Sample Sort

$k$-way partitioning

Input

Classification

Oracle

Output

[SANDERS04]
Super Scalar Sample Sort

$k$-way partitioning

Input
Classification
Oracle
Distribution
Output

branchless $k$-way decision-tree

[1 4 2 3 2] ... 3 ... w1 w2 w3 w4

bucket 1 bucket 2 bucket 3 bucket 4

[SANDERS04]
Contribution

Reduced branch mispredictions

- Block Quicksort
- Super Scalar Sample Sort
- Quick_sort
- Samplesort
- In-place
- I/O efficient

NEW

IS$^4_o$ IPS$^4_o$
In-place Super Scalar Samplesort

$k$-way partitioning

Input

Classification

Permutation

Cleanup

$b$ – block size

$b$ – block size
In-place Super Scalar Samplesort

Classification

branchless $k$-way decision-tree
In-place Super Scalar Samplesort

Classification

Input

Flush

k buffer blocks

branchless $k$-way decision-tree

One read and one write: $2^{n/B}$ I/Os
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
- First block: fetch last unpermuted block
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket

- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain

- First block: fetch last unpermuted block
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Permuation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort

Permutation

Invariant for each bucket
- Right: empty blocks
- Middle: unpermuted blocks
- Left: blocks in target bucket

Read from and write to same block: $2 \frac{n}{B}$ I/Os
In-place Super Scalar Samplesort

IS$^4$o vs. s$^3$-sort

- Slightly larger number of classifications per level: $n + n/b$
- TLB friendly
- inplace: $O(kb)$
- Slightly smaller number of I/Os: $4 \frac{n}{B} \left(4.\times \frac{n}{B}\right)$
In-place Parallel Super Scalar Samplesort

**k-way partitioning**

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Classification**

| thread 1 |
| 

| thread t |
| 

| $t \cdot k$ buffer blocks |
| 

**Empty block movement**

| 

**Permutation**

| 

**Cleanup**

| 

Call sequential subroutines in parallel if $n \leq n_{\text{init}}/t$
In-place Parallel Super Scalar Samplesort

- Atomic read and write pointers: fetch blocks atomically
- Access with fetch-and-add operations
- Blocks of size $\Omega(t)$ avoid contention
Experiments

Machines
- $2 \times$ Xeon E5-2683 v4 16-core, $4 \times$ Intel Xeon E5-4640, AMD Ryzen 1800x

Input instances
- Uniform distribution and 8 other distributions
- Shun et. al., Edelkamp et. al., ...

Input sizes
- $2^8 - 2^{34}$ elements

Data types
- $1 \times$ 64-bit floating point key
- $1 \times$ 64-bit floating point key, $1 \times$ 64-bit floating point data
- $3 \times$ 64-bit floating point key, $1 \times$ 64-bit floating point data
- 10 byte key, 90 byte data
In-place Super Scalar Samplesort

![Graph showing running time per item count for different sorting algorithms: IS^4o, s^3-sort, BlockQ, DualPivot, and std-sort.]

- **Running time / n log_2 n [ns]**
- **Item count n**

- IS^4o
- s^3-sort
- BlockQ
- DualPivot
- std-sort
In-place Parallel Super Scalar Samplesort

Running time / $n \log_2 n$ [ns]

Item count $n$

- IPS\textsuperscript{4o}
- PBBS
- MCSTL\textsubscript{mwm}
- MCSTL\textsubscript{bq}
- MCSTL\textsubscript{ubq}
- TBB
In-place Parallel Super Scalar Samplesort

![Graph showing speedup vs. core count for different libraries and libraries with different settings. The x-axis represents the core count, the y-axis represents the speedup. The graph compares the performance of IPS^4o, PBBS, MCSTLmwm, MCSTLbq, MCSTLubq, and TBB. The y-axis is labeled as Speedup \( t_{par} / t_{IS} \). The graph displays performance data for 2^{30} elements.]
In-place (Parallel) Super Scalar Samplesort

<table>
<thead>
<tr>
<th>Machine</th>
<th>Algo</th>
<th>Competitor</th>
<th>Uniform</th>
<th>AlmostSorted</th>
<th>RootDuplicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel2S</td>
<td>IS⁴₀</td>
<td>both</td>
<td>1.14</td>
<td>0.59</td>
<td>0.97</td>
</tr>
<tr>
<td>Intel4S</td>
<td>IS⁴₀</td>
<td>both</td>
<td>1.21</td>
<td>0.77</td>
<td>1.65</td>
</tr>
<tr>
<td>AMD1S</td>
<td>IS⁴₀</td>
<td>both</td>
<td>1.57</td>
<td>0.65</td>
<td>1.37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Machine</th>
<th>Algo</th>
<th>Competitor</th>
<th>Uniform</th>
<th>AlmostSorted</th>
<th>RootDuplicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel2S</td>
<td>IPS⁴₀</td>
<td>in-place</td>
<td>2.54</td>
<td>1.88</td>
<td>2.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-in-place</td>
<td>2.13</td>
<td>1.29</td>
<td>1.19</td>
</tr>
<tr>
<td>Intel4S</td>
<td>IPS⁴₀</td>
<td>in-place</td>
<td>3.52</td>
<td>3.62</td>
<td>3.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-in-place</td>
<td>1.75</td>
<td>1.84</td>
<td>1.15</td>
</tr>
<tr>
<td>AMD1S</td>
<td>IPS⁴₀</td>
<td>in-place</td>
<td>1.57</td>
<td>1.81</td>
<td>2.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-in-place</td>
<td>1.57</td>
<td>out of memory</td>
<td></td>
</tr>
</tbody>
</table>

Speedup of I(P)S⁴₀ to fastest competitor – 2³² elements

<table>
<thead>
<tr>
<th>Machine</th>
<th>Processors</th>
<th>Type</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel2S</td>
<td>2</td>
<td>Intel Xeon E5-2683 v4 16-core</td>
<td>512 GiB</td>
</tr>
<tr>
<td>Intel4S</td>
<td>4</td>
<td>Intel Xeon E5-4640 8-core</td>
<td>512 GiB</td>
</tr>
<tr>
<td>AMD1S</td>
<td>1</td>
<td>AMD Ryzen 1800x 8-core</td>
<td>32 GiB</td>
</tr>
</tbody>
</table>
### Conclusion and Further Work

**k-way partitioning**

<table>
<thead>
<tr>
<th></th>
<th>Quicksort</th>
<th>BlockQ</th>
<th>Samplesort</th>
<th>s³-sort</th>
<th>IS⁴o</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch mispred.</td>
<td>$n \log k$</td>
<td>$\frac{n}{b} \log k$</td>
<td>$n \log k$</td>
<td>$\frac{n}{b}$</td>
<td>$\frac{n}{b}$</td>
</tr>
<tr>
<td>I/O $\frac{n}{B} \times$</td>
<td>$2 \log_2 k$</td>
<td>$2 \log_2 k$</td>
<td>3</td>
<td>4 $x$</td>
<td>4</td>
</tr>
<tr>
<td>Add. space</td>
<td>$\log n$ (NEW)</td>
<td>$\log n + b$ (b NEW)</td>
<td>$n$</td>
<td>$n$</td>
<td>$kb$</td>
</tr>
</tbody>
</table>

Parallelized | yes | no | yes | no | yes |
Add. space     | $t + \log n$ (1 NEW) | $t + \log n$ (1 NEW) | $n$ | $n$ | $tkb$ |

Implicit big $O$ notation
## Conclusion and Further Work

### $k$-way partitioning

<table>
<thead>
<tr>
<th></th>
<th>Quicksort</th>
<th>BlockQ</th>
<th>Samplesort</th>
<th>$s^3$-sort</th>
<th>IS$^4$o</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch mispred.</td>
<td>$n \log k$</td>
<td>$\frac{n}{b} \log k$</td>
<td>$n \log k$</td>
<td>$\frac{n}{b}$</td>
<td>$\frac{n}{b}$</td>
</tr>
<tr>
<td>I/O $\frac{n}{B} \times$</td>
<td>$2 \log_2 k$</td>
<td>$2 \log_2 k$</td>
<td>$3$</td>
<td>$4 \times$</td>
<td>$4$</td>
</tr>
<tr>
<td>Add. space</td>
<td>$\log n$ (NEW)</td>
<td>$\log n + b$ (b NEW)</td>
<td>$n$</td>
<td>$n$</td>
<td>$kb$</td>
</tr>
<tr>
<td>Parallelized</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Add. space</td>
<td>$t + \log n$ (1 NEW)</td>
<td>$t + \log n$ (1 NEW)</td>
<td>$n$</td>
<td>$n$</td>
<td>$tkb$</td>
</tr>
</tbody>
</table>

Implicit big $O$ notation

---

**Matters for large elements?**

**Compare function contains branch mispredictions?**
## Conclusion and Further Work

### $k$-way partitioning

<table>
<thead>
<tr>
<th></th>
<th>Quicksort</th>
<th>BlockQ</th>
<th>Samplesort</th>
<th>$s^3$-sort</th>
<th>IS$^4$o</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch mispred.</td>
<td>$n \log k$</td>
<td>$\frac{n}{b} \log k$</td>
<td>$n \log k$</td>
<td>$\frac{n}{b}$</td>
<td>$\frac{n}{b}$</td>
</tr>
<tr>
<td>I/O</td>
<td>$\frac{n}{B} \times 2 \log_2 k$</td>
<td>$2 \log_2 k$</td>
<td>$2 \log_2 k$</td>
<td>3</td>
<td>4.x</td>
</tr>
<tr>
<td>Add. space</td>
<td>$\log n \ (1)$</td>
<td>$\log n + b \ (b)$</td>
<td>$\log n + b \ (b)$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Parallelized</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Add. space</td>
<td>$t + \log n \ (1)$</td>
<td>$t + \log n \ (1)$</td>
<td>$n$</td>
<td>$n$</td>
<td>$tkb$</td>
</tr>
</tbody>
</table>

Implicit big $O$ notation

Matters for large elements?

Compare function contains branch mispredictions?

- Load-balancing, NUMA
- Formal verification: [https://github.com/SaschaWitt/ips4o](https://github.com/SaschaWitt/ips4o)
- Standard library
BlockQuicksort

Goals

- Partially decoupling control flow from data flow
- Avoid conditional branches
- In-place: $O(b)$ additional space
BlockQuicksort

Goals

- Partially decoupling control flow from data flow
- Avoid conditional branches
- In-place: $O(b)$ additional space

$b$ Elements
BlockQuicksort

Goals

- Partially decoupling control flow from data flow
- Avoid conditional branches
- In-place: $O(b)$ additional space
BlockQuicksort

Goals
- Partially decoupling control flow from data flow
- Avoid conditional branches
- In-place: $O(b)$ additional space
BlockQuicksort

Goals
- Partially decoupling control flow from data flow
- Avoid conditional branches
- In-place: $O(b)$ additional space

![Diagram of BlockQuicksort]

- Pivot
- $b$ Elements
BlockQuicksort

Goals
- Partially decoupling control flow from data flow
- Avoid conditional branches
- In-place: $O(b)$ additional space
BlockQuicksort

Goals

- Partially decoupling control flow from data flow
- Avoid conditional branches
- In-place: $O(b)$ additional space
BlockQuicksort

Goals
- Partially decoupling control flow from data flow
- Avoid conditional branches
- In-place: $O(b)$ additional space

Drawbacks
- $O\left(\frac{n}{b} \log_2 \frac{n}{n_0}\right)$ block transfers
In-place Super Scalar Samplesort

Goals
- Partially decoupling control flow from data flow
- Avoid conditional branches
- $k$-way distribution
- Cache/IO-efficient
  - $O\left(\frac{n}{tb \log k} \frac{n}{n_0}\right)$ block transfers
- In-place: $O(kb)$ additional space
- Easy to parallelize
**In-place Super Scalar Samplesort**

Block Permutation

**Invariant of each bucket**

- Left: correctly placed blocks
- Middle: unpermuted blocks
- Right: empty blocks
In-place Super Scalar Samplesort

Block Permutation

Invariant of each bucket

- Left: correctly placed blocks
- Middle: unpermuted blocks
- Right: empty blocks

Swap buffers
In-place Super Scalar Samplesort

Block Permutation

Invariant of each bucket
- Left: correctly placed blocks
- Middle: unpermuted blocks
- Right: empty blocks

![Diagram showing block permutation process with left, middle, and right sections indicating correct placement, unpermuted, and empty blocks respectively, with swap buffers at the beginning marked as 1 and 2.}
In-place Super Scalar Samplesort
Block Permutation

Invariant of each bucket
- Left: correctly placed blocks
- Middle: unpermuted blocks
- Right: empty blocks

Swap buffers in cache
Read from and write to same location: $2 \frac{n}{B} \text{ I/Os}$
$n/b$ classifications
### Overview

<table>
<thead>
<tr>
<th>$k$-way distr.</th>
<th>QuickS</th>
<th>BlockQS</th>
<th>SampleS</th>
<th>SSSS</th>
<th>ISSSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch mispred.</td>
<td>many</td>
<td>*few</td>
<td>many</td>
<td>*few</td>
<td>*few</td>
</tr>
<tr>
<td>I/O in $\frac{n}{B}$</td>
<td>$2 \log_2 k$</td>
<td>$2 \log_2 k$</td>
<td>3</td>
<td>4.x</td>
<td>4</td>
</tr>
<tr>
<td>Add. space</td>
<td>1</td>
<td>$b$</td>
<td>$n$</td>
<td>$n$</td>
<td>$+kb$</td>
</tr>
<tr>
<td>Parallelized</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>inpl.</td>
</tr>
<tr>
<td>Add. space</td>
<td>1</td>
<td>–</td>
<td>$n$</td>
<td>–</td>
<td>1</td>
</tr>
</tbody>
</table>

*$O(tkb)$ for the parallel version

* slow inplace parallelizations described

$b$ Block size

$B$ Cache line size

$t$ Number of threads
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant

- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain

- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant

- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain

- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant

- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain

- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
**In-place Super Scalar Samplesort**

**Block Permutation**

Bucket invariant

- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

![Bucket invariant diagram](image)

Permutation chain

- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant

- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain

- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant

- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain

- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort

Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Permutation chain
- First block: fetch last unpermuted block
- Swap block: fetch first unpermuted block
- Last block: fetch first empty block
In-place Super Scalar Samplesort
Block Permutation

Bucket invariant
- Left: permuted blocks
- Middle: unpermuted blocks
- Right: empty blocks

Use two swap buffers
Read from and write to same block: $2 \frac{n}{B}$ I/Os
Overview

Quicksort

In-place

Samplesort

I/O efficient
Overview

Reduced branch mispredictions

Blocked Quicksort
BlockQ

Super-Scalar Samplesort
$s^3$-sort

Quicksort

Samplesort

In-place

I/O efficient
Overview

Reduced branch mispredictions

- Blocked Quicksort (BlockQ)
- Super-Scalar Samplesort ($s^3$-sort)

- Quicksort
- Samplesort

In-place

I/O efficient

NEW

$IS^4 o$  $IPS^4 o$
In-place Super Scalar Samplesort

\( k \)-way distribution

Input
In-place Super Scalar Samplesort

$k$-way distribution

Input

Classification

$b$ – block size

$b$ – block size
In-place Super Scalar Samplesort

\[ k\text{-way distribution} \]

Input

Classification

Permutation

\[ b \text{ – block size} \]
In-place Super Scalar Samplesort

$k$-way distribution

Input

Classification

Permutation

Cleanup

$b \rightarrow \text{block size}$
In-place Parallel Super Scalar Samplesort

Input

Classification

thread 1

\[ t \cdot k \text{ buffer blocks} \]

thread t
In-place Parallel Super Scalar Samplesort

Input

Classification

thread 1

\( t \cdot k \) buffer blocks

thread \( t \)

Empty block movement
In-place Parallel Super Scalar Samplesort

Input

Classification

thread 1

\[ \cdots \]

thread t

Empty block movement

Permutation
In-place Parallel Super Scalar Samplesort

Input

Classification

thread 1

\[ \cdots \]

thread t

Empty block movement

Permutation

Cleanup

Call sequential subroutines in parallel if \( n \leq n_{\text{init}}/t \)