Parallel String Sorting with Super Scalar String Sample Sort

Timo Bingmann and Peter Sanders | September 4th, 2013 @ ESA’13

Institute of Theoretical Informatics – Algorithmics
Abstract

We present the currently fastest parallel string sorting algorithm for modern multi-core shared memory architectures. First, we describe the challenges posed by these new architectures, and discuss key points to achieving high performance gains. Then we give an overview of existing sequential and parallel string sorting algorithms and implementations. Thereafter, we continue by developing super scalar string sample sort ($S^5$), which is easily parallelizable and yields higher parallel speedups than all previously known algorithms.
Overview

1. Introduction and Motivation
   - Parallel Memory Bandwidth Test

2. String Sorting Algorithms
   - Radix Sort
   - Multikey Quicksort
   - Super Scalar String Sample Sort

3. Experimental Results

4. Conclusion
String Sorting Algorithms

Theoretical Parallel Algorithms

- “Optimal Parallel String Algorithms: . . .” [Hagerup ’94]

Existing Basic Sequential Algorithms

- Radix Sort [McIlroy et al. ’95]
- Multikey Quicksort [Bentley, Sedgewick ’97]
- Burstsort [Sinha, Zobel ’04]
- LCP-Mergesort [Ng, Kakehi ’08]

Existing Algorithm Library

- by Tommi Rantala (for Radix Sort [Kärkkäinen, Rantala ’09])
  http://github.com/rantala/string-sorting
String Sorting Algorithms

Theoretical Parallel Algorithms

- “Optimal Parallel String Algorithms: . . .” [Hagerup ’94]

Existing Basic Sequential Algorithms

- Radix Sort [McIlroy et al. ’95]
- Multikey Quicksort [Bentley, Sedgewick ’97]
- Burstsort [Sinha, Zobel ’04]
- LCP-Mergesort [Ng, Kakehi ’08]

Existing Algorithm Library

- by Tommi Rantala (for Radix Sort [Kärkkäinen, Rantala ’09])
  http://github.com/rantala/string-sorting

Our Contribution: Practical Parallel Algorithms

- Parallel Super Scalar String Sample Sort (pS⁵) [This Work]
Modern Multi-Core Architecture
Parallel Memory Bandwidth Test

// ScanRead64IndexUnrollLoop
for (size_t i=0; i<n; i+=16) {
    uint64_t x0 = array[i+0];
    // ... 14 times
    uint64_t x15 = array[i+15];
}

// PermRead64SimpleLoop
uint64_t p = *array;
while((uint64_t*)p != array)
    p = *(uint64_t*)p;
ScanRead64IndexUnrollLoop

1,137 GiB/s

16 GiB/s

Timo Bingmann and Peter Sanders – Parallel String Sorting with Super Scalar String Sample Sort
Institute of Theoretical Informatics – Algorithmics
September 4th, 2013
ScanRead64IndexUnrollLoop

![Graph showing bandwidth vs. area size for different parallel processes](image)

- Bandwidth [GiB/s] on the y-axis
- Area size $n$ [B] on the x-axis
- Different lines for $p = 1, 2, 4, 8, 16, 32, 64$

- $p = 1$ (blue)
- $p = 2$ (red)
- $p = 4$ (green)
- $p = 8$ (purple)
- $p = 16$ (teal)
- $p = 32$ (brown)
- $p = 64$ (cyan)

Timo Bingmann and Peter Sanders – Parallel String Sorting with Super Scalar String Sample Sort
Institute of Theoretical Informatics – Algorithmics
September 4th, 2013
ScanRead64IndexUnrollLoop

![Graph showing bandwidth vs. area size for different parallel configurations.]

- Bandwidth is measured in GiB/s.
- Area size $n$ is in bytes (B).
- Different lines represent different parallel configurations, denoted by $p$.
- Key values: 35 GiB/s and 2.6 GiB/s are highlighted.

Timo Bingmann and Peter Sanders – Parallel String Sorting with Super Scalar String Sample Sort
Institute of Theoretical Informatics – Algorithmics
September 4th, 2013
PermRead64SimpleLoop

The graph shows the bandwidth (in GiB/s) as a function of the area size $n$ [B] for different parallel processing counts $p$. The bandwidth peaks for various $p$ values, with $p = 64$ showing the highest bandwidth at a certain area size, followed by $p = 32, 16, 8, 4, 2, 1$. The x-axis represents the area size in bytes, while the y-axis represents the bandwidth in GiB/s.
Parallel Memory Bandwidth Test

Scanning an Array

Random Access in an Array

Cache ($n = 16$ KiB)

<table>
<thead>
<tr>
<th>$p$</th>
<th>Scan</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35 GiB/s</td>
<td>4.4 GiB/s</td>
</tr>
<tr>
<td>16</td>
<td>567 GiB/s</td>
<td>69 GiB/s</td>
</tr>
<tr>
<td>32</td>
<td>1131 GiB/s</td>
<td>137 GiB/s</td>
</tr>
</tbody>
</table>

RAM ($n = 4$ GiB)

<table>
<thead>
<tr>
<th>$p$</th>
<th>Scan</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.6 GiB/s</td>
<td>19 MiB/s</td>
</tr>
<tr>
<td>16</td>
<td>10.3 GiB/s</td>
<td>403 MiB/s</td>
</tr>
<tr>
<td>32</td>
<td>10.6 GiB/s</td>
<td>763 MiB/s</td>
</tr>
</tbody>
</table>
Sorting Strings

array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
arcaic

⇒

abacus
alpha
arcade
arcaic
array
arrange
array
kayak
kernel
kitchen
kite
kitten
krypton
Sorting Strings

- array
- kit
- arrange
- kayak
- kernel
- kitchen
- kitten
- arcade
- kite
- abacus
- krypton
- alpha
- arcaic

- abacus
- alpha
- arcade
- arcaic
- arrange
- array
- kayak
- kernel
- kit
- kitchen
- kite
- krypton
Sorting Strings: Radix Sort

array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
arcaic

\[ 0 \quad a \quad k \quad \sigma - 1 \]

\[ 0 \quad \ldots \quad 6 \quad \ldots \quad 7 \quad \ldots \quad 0 \]
Sorting Strings: Radix Sort

array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
arcaic

0 a k σ - 1

0 ... 6 ...... 7 ...... 0

0 ... 0 6 ... 6 13 .... 13 13
Sorting Strings: Radix Sort

<table>
<thead>
<tr>
<th>array</th>
<th>0 a k σ−1</th>
</tr>
</thead>
<tbody>
<tr>
<td>kit</td>
<td>0 … 6 … 7</td>
</tr>
<tr>
<td>arrange</td>
<td>0 … 0 6 6</td>
</tr>
<tr>
<td>kayak</td>
<td>13 … 13 13</td>
</tr>
<tr>
<td>kernel</td>
<td></td>
</tr>
<tr>
<td>kitchen</td>
<td></td>
</tr>
<tr>
<td>kitten</td>
<td></td>
</tr>
<tr>
<td>arcade</td>
<td></td>
</tr>
<tr>
<td>kite</td>
<td></td>
</tr>
<tr>
<td>abacus</td>
<td></td>
</tr>
<tr>
<td>krypton</td>
<td></td>
</tr>
<tr>
<td>alpha</td>
<td></td>
</tr>
<tr>
<td>archaic</td>
<td></td>
</tr>
</tbody>
</table>
Sorting Strings: Radix Sort

array
array
arrange
arcade
abacus
alpha
arcaic
kit
kayak
kernel
kitchen
kitten
kite
krypton

0
0
a
6
ak
0

0
6
7
13
σ - 1

0
6
6
13
13
13
Sorting Strings: Radix Sort

<table>
<thead>
<tr>
<th>array</th>
<th>kit</th>
<th>arranged</th>
<th>kayak</th>
<th>kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>kitchen</td>
<td>kitten</td>
<td>arcade</td>
<td>kite</td>
<td>abacus</td>
</tr>
<tr>
<td>krypton</td>
<td>alpha</td>
<td>archaic</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccccccc}
\sigma - 1 & a & k & \sigma - 1 \\
\hline
0 & \ldots & 6 & \ldots & 7 & \ldots & 0 \\
0 & \ldots & 0 & 6 & \ldots & 6 & 13 & \ldots & 13 & 13 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
p_1 & 0 & \ldots & 2 & \ldots & 3 & \ldots & 0 \\
p_2 & 0 & \ldots & 2 & \ldots & 3 & \ldots & 0 \\
p_3 & 0 & \ldots & 2 & \ldots & 1 & \ldots & 0 \\
\end{array}
\]
Sorting Strings: Radix Sort

array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
archaic

\[
\begin{array}{cccccc}
0 & a & k & \sigma - 1 \\
0 & \cdots & 6 & \cdots & 7 & \cdots & 0 \\
0 & \cdots & 0 & 6 & \cdots & 6 & 13 & \cdots & 13 & 13 \\
\end{array}
\]

\[
\begin{array}{cccccc}
p_1 & 0 & \cdots & 2 & \cdots & 3 & \cdots & 0 \\
p_2 & 0 & \cdots & 2 & \cdots & 3 & \cdots & 0 \\
p_3 & 0 & \cdots & 2 & \cdots & 1 & \cdots & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
p_1 & 0 & \cdots & 0 & 6 & \cdots & 6 & 13 & \cdots & 13 & 13 \\
p_2 & 0 & \cdots & 2 & 6 & \cdots & 9 & 13 & \cdots & 13 \\
p_3 & 0 & \cdots & 4 & 6 & \cdots & 12 & 13 & \cdots & 13 \\
\end{array}
\]
Sorting Strings: Multikey Quicksort

[Bentley, Sedgewick ’97]

array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
arcane
Sorting Strings: Multikey Quicksort

[ Bentley, Sedgewick ’97 ]

array
arrange
kayak
kernel
arcade
abacus
alpha
arcaic

kit
kitchen
kitten
kite
krypton
Sorting Strings: Multikey Quicksort

[Bentley, Sedgewick ’97]

array
arrange
kayak
kernel
arcade
abacus
alpha
arcaic
kit
kitchen
kitten
kite
krypton

=[arah<aro<ar]<k]<k<ar<ak<array=k
Sorting Strings: Multikey Quicksort

[Bentley, Sedgewick '97]

array
arrange
kayak
kernel
arcade
abacus
alpha
archaic
kit
kitchen
kitten
kite
krypton
Sorting Strings: Multikey Quicksort

[Bentley, Sedgewick ’97]

partition by $w = 8$ characters

[Rantala ’??]

- cache characters
  - fewer random accesses

- fastest sequential algorithm
Sorting Strings: Multikey Quicksort

[Bentley, Sedgewick ’97]

array
arrange
kayak
kernel
arcade
abacus
alphabet
arcaic

[Rantala ’??]
- partition by $w = 8$ characters
- cache characters
  - fewer random accesses
- fastest sequential algorithm

[This Work]
- parallelize using blocks
### Sorting Strings: Multikey Quicksort

<table>
<thead>
<tr>
<th>array</th>
<th>arrange</th>
<th>kayak</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel</td>
<td>arcade</td>
<td>abacus</td>
</tr>
<tr>
<td>alpha</td>
<td>archaic</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kit</th>
<th>kitchen</th>
<th>kitten</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kite</td>
<td>krypton</td>
</tr>
</tbody>
</table>

- $p_1 < p_2 < p_3$
Sorting Strings: Multikey Quicksort

array

arrange

kayak

kernel

arcade

abacus

alpha

arcaic

kit

kitchen

kitten

kite

krypton


p1 < = ? > ?

p2 < ? = ? >

p3 < ? = > ?
Sorting Strings: Multikey Quicksort

array
arrange
kayak
kernel
arcade
abacus
alpha
arcaic

kit
kitchen
kitten
kite
krypton

Output

\[ \begin{array}{c}
\text{array} \\
\text{arrange} \\
kayak \\
kernel \\
arcade \\
abacus \\
alpaha \\
arcaic \\
kit \\
kitchen \\
kitten \\
kite \\
krypton \\
\end{array} \]

\[ \begin{array}{c}
\text{<} \\
\text{<} \\
\text{<} \\
\text{=} \\
\text{=} \\
\text{>} \\
\text{>} \\
\text{=} \\
\text{=} \\
\text{>} \\
\end{array} \]
Sorting Strings: Multikey Quicksort

array
arrange
kayak
kernel
arcade
abacus
alpha
arcaic

kit
kitchen
kitten
kite

krypton

Output

p₁ < ? = ? >
p₂ < >
p₃ < ? = ? >
Sorting Strings: Multikey Quicksort

array
array
kayak
kernel
arcade
abacus
alpha
archaic
kit
kitchen
kitten
kite
krypton

Output

<  <  <
=
>
>

<  <  <  <
=
>
>

p1 < ? = ? ? 0
p2 0 = ? > ?
p3 < ? = ? 0

Timo Bingmann and Peter Sanders – Parallel String Sorting with Super Scalar String Sample Sort
Institute of Theoretical Informatics – Algorithmics
September 4th, 2013
Sorting Strings: Multikey Quicksort

array
arrange
kayak
kernel
arcade
abacus
alpha
archaic

<

kit
kitchen
kitten
kite
krypton

More Output

p1
<0 =0 >0

p2
<0 =0 >0

p3
<0 =0 >0

Output

< < <

= =

> >
Super Scalar String Sample Sort ($S^5$)

array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
archaic
Super Scalar String Sample Sort ($S^5$)

array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
archaic

\[
\begin{array}{c}
\text{ar} \\
\text{ab} \\
\text{ki} \\
\text{b}_0 \quad \text{b}_1 \quad \text{b}_2 \quad \text{b}_3 \quad \text{b}_4 \quad \text{b}_5 \quad \text{b}_6
\end{array}
\]

Timo Bingmann and Peter Sanders – Parallel String Sorting with Super Scalar String Sample Sort
Institute of Theoretical Informatics – Algorithmics
September 4th, 2013
Super Scalar String Sample Sort ($S^5$)

array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
arcade

< \[=\] >
< \[=\] >
< \[=\] >

$S^5$
**Super Scalar String Sample Sort (S⁵)**

```
array
kit
arrange
kayak
kernel
kitchen
kitten
arcade
kite
abacus
krypton
alpha
archaic
```

```
ab
<
≤
>
ki
<
≤
>
```

```
0 0 1 4 8 9 12
0 1 1 5 8 12 12
0 1 2 6 8 12 13
```
Super Scalar String Sample Sort ($S^5$)

- abacus
- alpha
- array
- arrange
- arcade
- arcacic
- kayak
- kernel
- kit
- kitchen
- kitten
- kite
- krypton

```
< = >
```

0 0 1 1 4 8 8 9 12 12
0 1 1 5 8 12 12
0 1 2 6 8 12 13
Super Scalar String Sample Sort ($S^5$)

Increase prefix by LCP of splitters or key size.
Super Scalar String Sample Sort ($S^5$)

<table>
<thead>
<tr>
<th>abacuss</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphaa</td>
<td>1</td>
</tr>
<tr>
<td>arraya</td>
<td>2</td>
</tr>
<tr>
<td>arrange</td>
<td>0</td>
</tr>
<tr>
<td>arcade</td>
<td>0</td>
</tr>
<tr>
<td>arcaic</td>
<td>2</td>
</tr>
<tr>
<td>kayak</td>
<td>0</td>
</tr>
<tr>
<td>kernel</td>
<td>0</td>
</tr>
<tr>
<td>kite</td>
<td>2</td>
</tr>
<tr>
<td>kitchen</td>
<td>2</td>
</tr>
<tr>
<td>kitten</td>
<td>2</td>
</tr>
<tr>
<td>kitem</td>
<td>0</td>
</tr>
<tr>
<td>krypton</td>
<td>0</td>
</tr>
</tbody>
</table>

partitions by $w = 8$ chars

easy parallelization

increase prefix by LCP of splitters or key size.

prefixes:

- abacuss
- alphaa
- arraya
- arrange
- arcade
- arcaic
- kayak
- kernel
- kite
- kitchen
- kitten
- kitem
- krypton

LCP = Longest Common Prefix

Timo Bingmann and Peter Sanders – Parallel String Sorting with Super Scalar String Sample Sort
Institute of Theoretical Informatics – Algorithmics
September 4th, 2013
Super Scalar String Sample Sort ($S^5$)

- partitions by $w = 8$ chars
- easy parallelization
- 256 KiB L2 cache: 13 levels

- predicated instructions

$$1 \quad 2 \quad 3$$

$$\begin{array}{ccc}
    \text{ar} & \text{ab} & \text{ki} \\
\end{array}$$

$$i := 2i + 0/1$$
Super Scalar String Sample Sort ($S^5$)

- partitions by $w = 8$ chars
- easy parallelization
- 256 KiB L2 cache: 13 levels

- predicated instructions
  
  $i := 2i + 0/1$

- equality checking:
  1. at each node
  2. after full descent
Super Scalar String Sample Sort ($S^5$)

- partitions by $w = 8$ chars
- easy parallelization
- 256 KiB L2 cache: 13 levels
- predicated instructions
  
  \[
  \begin{array}{ccc}
  1 & 2 & 3 \\
  \text{ar} & \text{ab} & \text{ki}
  \end{array}
  \]

  \[
  i := 2i + 0/1
  \]

- equality checking:
  1. at each node
  2. after full descent

- **interleave** tree descents:
  classify 4 strings at once
  \[\Rightarrow\] super scalar parallelism
Parallel $S^5$ – Sub-Algorithms

- $|S| \geq \frac{n}{p}$ fully parallel $S^5$
- $\frac{n}{p} > |S| \geq 2^{16}$ sequential $S^5$
- $2^{16} > |S| \geq 64$ caching multikey quicksort
- $64 > |S|$ insertion sort

Important: dynamic load balancing with voluntary work freeing
URLs – 1.1 G Lines, 70.7 GiB

http://algo2.iti.kit.edu/index.php
http://algo2.iti.kit.edu/english/index.php
http://algo2.iti.kit.edu/1483.php
http://algo2.iti.kit.edu/1484.php
http://www.kit.edu/
http://algo2.iti.kit.edu/286.php
http://algo2.iti.kit.edu/1294.php
http://algo2.iti.kit.edu/research.php
http://algo2.iti.kit.edu/publications.php
http://algo2.iti.kit.edu/members.php
http://algo2.iti.kit.edu/lehre.php
http://algo2.iti.kit.edu/1844.php
http://algo2.iti.kit.edu/294.php
http://algo2.iti.kit.edu/basic-toolbox-page.php
http://map.project-osrm.org?dest=49.0137004,8.419307&destname=%22Am%20Fa...
http://www.uni-karlsruhe.de/fs/Uni/info/campusplan/index.php?id=50.34
http://www.informatik.kit.edu/1158.php
http://algo2.iti.kit.edu/emailform.php?id=eea49c752e4c1329710cba2efae10511
http://algo2.iti.kit.edu/routeplanning.php
http://algo2.iti.kit.edu/sanders.php
http://www.mwk.baden-wuerttemberg.de/forschung/forschungsfoerderung/land...
http://algo2.iti.kit.edu/1996.php
URLs – Speedup on 32-core Intel E5

The diagram shows the speedup of different sorting algorithms as a function of the number of threads. The algorithms compared are:

- pS^5
- pMultikeyQuicksort
- pRadixSort

The speedup increases with the number of threads up to a certain point, after which it levels off. The pS^5 algorithm shows the highest speedup across all thread counts, followed by pMultikeyQuicksort and then pRadixSort.
The ""Karlsruhe Institute of Technology"" (""KIT""), is one of the largest research and educations institution in Germany, resulting from a merger of the university (""Universität Karlsruhe (TH)"") and the research center (""Forschungszentrum Karlsruhe"")&lt;ref&gt;[[Federal Ministry of Education and Research (Germany)]]: http://www.bmbf.de/pub/eckpunktepapier_kit.pdf&lt;/ref&gt; of the city of [[Karlsruhe]]. The university, also known as ""Fridericiana"", was founded in 1825. In 2009, it merged with the former national nuclear research center founded in 1956 as the ""Kernforschungszentrum Karlsruhe (KfK)"".
BACKGROUND INFORMATION. The Forest Ecosystem Dynamics (FED) Project is concerned with modeling and monitoring ecosystem processes and patterns in response to natural and anthropogenic effects. The project uses coupled ecosystem models and remote sensing models and measurements to predict and observe ecosystem change. The overall objective of the FED project is to link and use models of forest dynamics, soil processes, and canopy energetics to understand how ecosystem response to change affects patterns and processes in northern and boreal forests and to assess the implications for global change. See Conceptual Diagram for model schematic.

The Forest Ecosystem Dynamics World-Wide-Web server has been online since July 1994. The FED server was created for the dissemination of project information, to archive numerous spatial and scientific data sets, and demonstrate the linking of ecosystem and remote sensing models.
GOV2 – Speedup on 32-core Intel E5

![Graph showing speedup vs number of threads for different algorithms]

- **pS^5**
- **pMultikeyQuicksort**
- **pRadixSort**

Number of threads: 1, 8, 16, 32, 48, 64

Speedup: 0, 2, 4, 6, 8, 10, 12
stereosto tocellular cellular and portable well as discussion of computer security issues in general. While we welcome all manner of question about encrypted messages, how does PGP encrypt a message? What is a public key; encrypted messages themselves are off-topic. Host Camelot Message File BBS Culver City CA List Graphics General Graphics /one.prop/five.prop Discussions about all types of computer graphics and language conference for the exchange of ideas. Learn more about the Spanish culture while communicating in the language. Host List Fantasy Sci Fi Fantasy.
Words – Speedup on 4-core Intel i7

number of threads

speedup

1 2 3 4 5 6 7 8

pS^5
pMultikeyQuicksort
pRadixSort

Timo Bingmann and Peter Sanders – Parallel String Sorting with Super Scalar String Sample Sort
Institute of Theoretical Informatics – Algorithmics
September 4th, 2013
Future Work

- Non-uniform memory architecture (NUMA) effects
- Distributed string sorting algorithms
- Distributed text index construction and query
  - High-performance middleware?
Thank you for your attention!

Questions?