Automatic SIMDization of Parallel Sorting on x86-based Manycore Processors

Presenter: Kaixi Hou
Collaborator: Hao Wang
Advisor: Wu Feng
Why sorting?

• Sorting used as an important primitive in many applications
  – Databases, computational biology, graph algorithms, etc.
Why sorting?

- Sorting used as an important primitive in many applications
  - Databases, computational biology, graph algorithms, etc.
- To get efficient sort, all computing resources need to be used
Why sorting?

- Sorting is an important primitive in many applications, such as databases, computational biology, graph algorithms, etc.
- To get efficient sort, all computing resources need to be used.
Why sorting?

Sorting is used as a fundamental primitive in many applications such as databases, computational biology, graph algorithms, etc. To get efficient sorting, all computing resources need to be used.

Cannot just rely on clock rate
Why sorting?

- Sorting used as an important primitive in many applications such as databases, computational biology, graph algorithms, etc.

More DLP and TLP

Cannot just rely on clock rate
Why sorting?

• Sorting used as an important primitive in many applications such as databases, computational biology, graph algorithms, etc.

• To get efficient sort, all computing resources need to be used.

• More DLP and TLP cannot just rely on clock rate.

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
VPU: Vector Processing Units

• VPU follows the *Single Instruction, Multiple Data* (SIMD) paradigm
  – “lock-step” operations over packed data
VPU: Vector Processing Units

• VPU follows the *Single Instruction, Multiple Data* (SIMD) paradigm
  – “lock-step” operations over packed data

• CPU: AVX and AVX2
  – 256-bit
  – SandyBridge or later
VPU: Vector Processing Units

- VPU follows the *Single Instruction, Multiple Data* (SIMD) paradigm
  - “lock-step” operations over packed data
- CPU: AVX and AVX2
  - 256-bit
  - SandyBridge or later
- MIC: AVX-512 (AVX3)
  - 512-bit
  - Xeon Phi
Approaches to Data-Level Parallelism (i.e., SIMD)
Approaches to Data-Level Parallelism (i.e., SIMD)

- Compiler-based approaches
  - Compiler options
  -Pragma directives
Approaches to Data-Level Parallelism (i.e., SIMD)

• Compiler-based approaches
  – Compiler options
  –Pragma directives

Rely on the modern compilers (icc, gcc, etc.)
  – compiler options (-O2, -vec-report2)
  –pragma directives ("vector always", "ivdep", "simd")
Approaches to Data-Level Parallelism (i.e., SIMD)

- Compiler-based approaches
  - Compiler options
  - Pragma directives
Approaches to Data-Level Parallelism (i.e., SIMD)

- Compiler-based approaches
  - Compiler options
  -Pragma directives

*Issues:*
Fail to auto-vec loops, due to complex memory access, convoluted data rearrangement, etc.
Approaches to Data-Level Parallelism (i.e., SIMD)

• Compiler-based approaches
  – Compiler options
  –Pragma directives
Approaches to Data-Level Parallelism (i.e., SIMD)

- Compiler-based approaches
  - Compiler options
  -Pragma directives

- Manual optimization via ...
  - Compiler intrinsics
  - Assembly code
Approaches to Data-Level Parallelism (i.e., SIMD)

• Compiler-based approaches
  – Compiler options
  –Pragma directives

• Manual optimization via ...
  – Compiler intrinsics
  – Assembly code

Issues:
Tedious and error-prone.
Approaches to Data-Level Parallelism (i.e., SIMD)

• Compiler-based approaches
  – Compiler options
  –Pragma directives

• Manual optimization via ...
  – Compiler intrinsics
  – Assembly code
Approaches to Data-Level Parallelism (i.e., SIMD)

• Compiler-based approaches
  – Compiler options
  – Pragma directives

• Manual optimization via ...
  – Compiler intrinsics
  – Assembly code

Ex. Interleave two arrays
Approaches to Data-Level Parallelism (i.e., SIMD)

- Compiler-based approaches
  - Compiler options
  -Pragma directives
- Manual optimization via ...
  - Compiler intrinsics
  - Assembly code

Serial C codes

```c
for(i=0; i<2w; i++)
{
    if(i<w)
    else
        trgB[i-w]= i%2 != 0 ? inpB[i/2] : inpA[i/2];
}
```

Ex. Interleave two arrays
Approaches to Data-Level Parallelism (i.e., SIMD)

• Compiler-based approaches
  – Compiler options
  –Pragma directives

• Manual optimization via ...
  – Compiler intrinsics
  –Assembly code

Serial C codes

```c
for(i=0; i<2w; i++)
{
    if(i<w)
    else
        trgB[i-w] = i%2!=0 ? inpB[i/2] : inpA[i/2];
}
```

Ex. Interleave two arrays

AVX intrinsics on CPUs

```c
__mm256 v1 = _mm256_unpacklo_ps(inpA, inpB);
__mm256 v2 = _mm256_unpackhi_ps(inpA, inpB);
__mm256 trgA = _mm256_permute2f128_ps(v1, v2, 0x20);
__mm256 trgB = _mm256_permute2f128_ps(v1, v2, 0x31);
```
Approaches to Data-Level Parallelism (i.e., SIMD)

- Compiler-based approaches
  - Compiler options
  - Pragma directives
- Manual optimization via ...
  - Compiler intrinsics
  - Assembly code

Serial C codes

```c
for(i=0; i<2w; i++)
{
    if(i<w)
    else
        trgB[i-w] = i%2!=0 ? inpB[i/2] : inpA[i/2];
}
```

Ex. Interleave two arrays

AVX intrinsics on CPUs

```assembly
__mm256 v1 = _mm256_unpacklo_ps(inpA, inpB);
__mm256 v2 = _mm256_unpackhi_ps(inpA, inpB);
__mm256 trgA = _mm256_permute2f128_ps(v1, v2, 0x20);
__mm256 trgB = _mm256_permute2f128_ps(v1, v2, 0x31);
```

AVX512 intrinsics on MIC

```assembly
__mm512i l = _mm512_permute4f128_epi32(inpA, _MM_PERM_BDAC);
__mm512i h = _mm512_permute4f128_epi32(inpB, _MM_PERM_BDAC);
__mm512i t0 = _mm512_mask_swizzle_epi32(h, 0xcccc, l, _MM_SWIZ_REG_BADC);
__mm512i t1 = _mm512_mask_swizzle_epi32(l, 0x3333, h, _MM_SWIZ_REG_BADC);
__mm512i l = _mm512_mask_permute4f128_epi32(t1, 0x0f0f, t0, _MM_PERM_CDAB);
__mm512i h = _mm512_mask_permute4f128_epi32(t0, 0x0f0f, t1, _MM_PERM_CDAB);
__mm512i trgA = _mm512_shuffle_epi32(l, _MM_PERM_BDAC);
__mm512i trgB = _mm512_shuffle_epi32(h, _MM_PERM_BDAC);
```
Approaches to Data-Level Parallelism (i.e., SIMD)

• Compiler-based approaches
  – Compiler options
  –Pragma directives

• Manual optimization via ...
  – Compiler intrinsics
  – Assembly code

Serial C codes

```c
for(i=0; i<2w; i++)
{
    if(i<w)
    else
        trgB[i-w]= i%2!=0 ? inpB[i/2] : inpA[i/2];
}
```

Ex. Interleave two arrays

AVX intrinsic on CPUs

```c
__mm256 v1 = _mm256_unpacklo_ps(inpA, inpB);
__mm256 v2 = _mm256_unpackhi_ps(inpA, inpB);
__mm256 trgA = _mm256_permute2f128_ps(v1, v2, 0x20);
__mm256 trgB = _mm256_permute2f128_ps(v1, v2, 0x31);
```

AVX512 intrinsics on MIC

```c
__mm512i l = _mm512_permute4f128_epi32(inpA, _MM_PERM_BDAC);
__mm512i h = _mm512_permute4f128_epi32(inpB, _MM_PERM_BDAC);
__mm512i t0 = _mm512_mask_swizzle_epi32(h, 0xcccc, l, _MM_SWIZ_REG_BADC);
__mm512i t1 = _mm512_mask_swizzle_epi32(l, 0x3333, h, _MM_SWIZ_REG_BADC);
__mm512i l = _mm512_mask_permute4f128_epi32(t1, 0x0f0f, t0, _MM_PERM_CDAB);
__mm512i h = _mm512_mask_permute4f128_epi32(t0, 0xf0f0, t1, _MM_PERM_CDAB);
__mm512i trgA = _mm512_shuffle_epi32(l, _MM_PERM_BDAC);
__mm512i trgB = _mm512_shuffle_epi32(h, _MM_PERM_BDAC);
```

Can they be automatically generated?
ASPaS Framework

- Formalize the data-reordering patterns in the parallel sorting
ASPaS Framework

- Formalize the data-reordering patterns in the parallel sorting
- Automatically generate the SIMD code
ASPaS Framework

- Formalize the data-reordering patterns in the parallel sorting
- Automatically generate the SIMD code
- Applied generally to DLP architecture, specific to x86 processors
ASPaS Framework

- Formalize the data-reordering patterns in the parallel sorting
- Automatically generate the SIMD code
- Applied generally to DLP architecture, specific to x86 processors

Unsorted data

ASPaS::sort()

Sorted data

For users

For developers

Different building blocks
Roadmap

• Introduction & Motivation
• Background
  – Sorting Networks & SIMD Processing
• ASPaS Framework
  – SIMD Sorter ▷ Generate patterns for sorting the data segment by segment
  – SIMD Transposer
  – SIMD Merger ▷ Generate patterns for merging the sorted data
  – SIMD Code Generator ▷ Generate codes from the patterns
• Evaluation & Discussion
• Conclusion
Background: Sorting Networks

- Sorting Networks
  - Comparisons can be planned out in a fixed pattern
  - Data flow is irrelevant with the value of input data
Background: Sorting Networks

- Sorting Networks
  - Comparisons can be planned out in a fixed pattern
  - Data flow is irrelevant with the value of input data
Background: Sorting Networks

• Sorting Networks
  – Comparisons can be planned out in a fixed pattern
  – Data flow is irrelevant with the value of input data

Two sorting networks
Background: Sorting Networks

• Sorting Networks
  – Comparisons can be planned out in a fixed pattern
  – Data flow is irrelevant with the value of input data

Two sorting networks
Background: Sorting Networks

- **Sorting Networks**
  - Comparisons can be planned out in a fixed pattern
  - Data flow is irrelevant with the value of input data

![Two sorting networks](image)
Background: Sorting Networks

- Sorting Networks
  - Comparisons can be planned out in a fixed pattern
  - Data flow is irrelevant with the value of input data

Two sorting networks

Merging network
### Background: SIMD for Intel Xeon Phi

- **VPU Architecture**
  - Manipulate one vector

<table>
<thead>
<tr>
<th>D</th>
<th>Lane D</th>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>Lane C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>Lane B</th>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>Lane A</th>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Lane/Element Muxes are units to do different levels of data-reordering.
Background: SIMD for Intel Xeon Phi

• VPU Architecture
  – Manipulate one vector

Lane/Element Muxes are units to do different levels of data-reordering.
Background: SIMD for Intel Xeon Phi

- **VPU Architecture**
  - Manipulate one vector

Lane Muxes

Element Muxes

Lane/Element Muxes are units to do different levels of data-reordering.
Background: SIMD for Intel Xeon Phi

- **VPU Architecture**
  - Manipulate two vectors

![Diagram of VPU Architecture](Image)

- (tmp) Certain data-reordering
Background: SIMD for Intel Xeon Phi

- VPU Architecture
  - Manipulate two vectors

\[
\begin{array}{ccccccccc}
\text{Lane D} & \text{Lane C} & \text{Lane B} & \text{Lane A} \\
\hline
v1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
(tmp) & 3 & 4 & 1 & 2 & 7 & 8 & 5 & 6 & 11 & 12 & 9 & 10 & 15 & 16 & 13 & 14 \\
\end{array}
\]

Certain data-reordering
Background: SIMD for Intel Xeon Phi

- **VPU Architecture**
  - Manipulate two vectors

```
  v1
  [D C B A][D C B A][D C B A]
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
  (tmp)
  3 4 1 2 7 8 5 6 11 12 9 10 15 16 13 14

  v0
  [D C B A][D C B A][D C B A]
  17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

  m1
  [0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1]

  t0
  [17 18 1 2 21 22 5 6 25 26 9 10 29 30 13 14]
```

Certain data-reordering

Kaixi@VT

8/21/2015
Roadmap

• Introduction & Motivation
• Background
  – VPU Architecture & Sorting Networks
• ASPaS Framework
  – SIMD Sorter
  – SIMD Transposer
  – SIMD Merger
  – SIMD Code Generator
    ▶ Generate patterns for sorting the data segment by segment
    ▶ Generate patterns for merging the sorted data
    ▶ Generate codes from the patterns
• Evaluation & Discussion
• Conclusion
ASPaS Framework

- ASPaS Structure Overview

For users

```
Unsorted data

ASPaS::sort()

Sorted data
```

Kaixi@VT
8/21/2015
ASPaS Framework

- ASPaS Structure Overview

Unsorted data → **Sort Stage**

- aspas_sort()
- aspas_transpose()

ASPaS::sort() → Sorted data

For users
**ASPaS Framework**

- **ASPaS Structure Overview**

  ![Diagram of ASPaS framework showing Sort Stage and Merge Stage with functions aspas_sort, aspas_transpose, and aspas_merge]

  **For users**
ASPaS Framework

- ASPaS Structure Overview

```
Unsorted data

Sort Stage
  aspas_sort()
  aspas_transpose()

Merge Stage
  aspas_merge()

Sorted data
```

For users

For developers
ASPaS Framework

- ASPaS Structure Overview

For users

For developers

ASPaS Framework

Unsorted data

Sorted data

Sort Stage

Merge Stage

aspas_sort()

aspas_transpose()

aspas_merge()
ASPaS Framework

• ASPaS Structure Overview

For users

For developers

ASPaS Framework

Unsorted data

Sort Stage

aspas_sort()

aspas_transpose()

Merge Stage

aspas_merge()

Sorted data

ASPaS::sort()

Sort Stage

aspas_sort()

aspas_transpose()

Merge Stage

aspas_merge()
ASPaS Framework

- ASPaS Structure Overview

For users
- Unsorted data
- aspas_sort()
- aspas_transpose()
- ASPaS::sort()
- Merge Stage
- aspas_merge()
- Sorted data

For developers
- Sort Stage
- Merge Stage
- ASPaS Framework
- SIMD Sorter
- SIMD Matrix Transposer
- SIMD Merger

Unsorted data

For users

For developers

ASPaS Framework

8/21/2015

Kaixi@VT
ASPaS Framework

• ASPaS Structure Overview

For users

Sort Stage

Unsorted data

aspas_sort() aspas_transpose() ASPaS::sort()

Merge Stage

aspas_merge()

Sorted data

For developers

ASPaS Framework

SIMD Sorter SIMD Matrix Transposer SIMD Merger

SIMD Code Generator

ISA-friendly primitive pool translate Real SIMD inst.

Unsorted data

Sort Stage

Merge Stage

Sorted data

12
ASPaS Framework

- ASPaS Structure Overview

```
Sort Stage
Unsorted data
aspas_sort() -> aspas_transpose()

Merge Stage
aspas::sort() -> aspas_merge()
```

For users
For developers

```
Sort Stage
Unsorted data
aspas_sort() -> aspas_transpose()

Merge Stage
aspas::sort() -> aspas_merge()
```

SIMD Code Generator
ISA-friendly primitive pool
Real SIMD inst.
translate

Patterns
SIMD Sorter
Patterns
SIMD Transposer
Patterns
SIMD Merger
ASPaS Framework

- ASPaS Structure Overview

For users

For developers

ASPaS Framework

SIMD Code Generator

SIMD Sorter

SIMD Matrix Transposer

SIMD Merger

Sort Stage

Unsorted data

aspas_sort()

aspas_transpose()

Merge Stage

Sorted data

aspas_merge()
ASPaS Framework

- ASPaS Structure Overview

For users

```
Unsorted data
```

```
Sort Stage
aspas_sort() aspas_transpose()
```

```
Merge Stage
aspas_merge()
```

```
Sorted data
```

For developers

```
Check validation
```

```
SIMD Code Generator
ISA-friendly primitive pool
translate
Real SIMD inst.
```

```
SIMD Sorter
SIMD Matrix Transposer
```

```
SIMD Merger
```

```
Sorting networks
```

```
Merging networks
```

---

Kaixi@VT
8/21/2015
Now, let’s work on an example.
Now, let’s work on an example.
Now, let’s work on an example.

We’d like to sort this array in the SIMD style.
ASPaS Framework

• SIMD Sorter
  – Generate regrouped comparison patterns
  – Accept any kinds of sorting networks
ASPaS Framework

• SIMD Sorter
  – Generate regrouped comparison patterns
  – Accept any kinds of sorting networks

v0 7 6 5 4
v1 5 2 8 2
v2 9 1 9 5
v3 3 5 6 7

4-key sorting network
**ASPaS Framework**

- **SIMD Sorter**
  - Generate regrouped comparison patterns
  - Accept any kinds of sorting networks

```
- v0: 7 6 5 4
- v1: 5 2 8 2
- v2: 9 1 9 5
- v3: 3 5 6 7
```

4-key sorting network

```
CMP(0, 1); CMP(2, 3); CMP(0, 3);
CMP(1, 2); CMP(0, 1); CMP(2, 3);
```

Input Macros
ASPaS Framework

• SIMD Sorter
  – Generate regrouped comparison patterns
  – Accept any kinds of sorting networks

Some Math Patterns of Comparisons

Minimized Grouping

Input Macros

v0
7 6 5 4
v1
5 2 8 2
v2
9 1 9 5
v3
3 5 6 7
ASPaS Framework

- **SIMD Sorter**
  - Generate regrouped comparison patterns
  - Accept any kinds of sorting networks

```
7 6 5 4
5 2 8 2
9 1 9 5
3 5 6 7
```

```
3 1 5 2
5 2 6 4
7 5 8 5
9 6 9 7
```

**Input Macros**

- CMP(0, 1); CMP(2, 3); CMP(0, 3); CMP(1, 2); CMP(0, 1); CMP(2, 3);

**Some Math Patterns of Comparisons**

**Minimized Grouping**

8/21/2015
ASPaS Framework

- SIMD Transposer
  - Why need the transpose?
ASPaS Framework

- SIMD Transposer
  - Why need the transpose?
ASPaS Framework

• SIMD Transposer
  – Generalize the patterns required in the in-register transpose
ASPaS Framework

- SIMD Transposer
  - Generalize the patterns required in the in-register transpose

```
v0  3  1  5  2
v1  5  2  6  4
v2  7  5  8  5
v3  9  6  9  7
```
ASPaS Framework

• SIMD Transposer
  – Generalize the patterns required in the in-register transpose

Same pattern applies on
v2 and v3
ASPaS Framework

• SIMD Transposer
  – Generalize the patterns required in the in-register transpose

Same pattern applies on v2 and v3

Same pattern applies on v1 and v3
ASPaS Framework

• SIMD Transposer
  – Generalize the patterns required in the in-register transpose

Some data-reordering patterns

<table>
<thead>
<tr>
<th>v0</th>
<th>3</th>
<th>1</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>v2</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>v3</td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>

Same pattern applies on v2 and v3

<table>
<thead>
<tr>
<th>v0</th>
<th>3 = v00</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>5 = v01</td>
</tr>
<tr>
<td>v2</td>
<td>5 = v02</td>
</tr>
<tr>
<td>v3</td>
<td>6 = v03</td>
</tr>
</tbody>
</table>

Same pattern applies on v1 and v3

<table>
<thead>
<tr>
<th>v0</th>
<th>3 = v00</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>5 = v01</td>
</tr>
<tr>
<td>v2</td>
<td>7 = v02</td>
</tr>
<tr>
<td>v3</td>
<td>9 = v03</td>
</tr>
</tbody>
</table>

Kaixi@VT
8/21/2015
ASPaS Framework

- SIMD Transposer
  - Generalize the patterns required in the in-register transpose

Some data-reordering patterns

- Same pattern applies on v2 and v3
- Same pattern applies on v1 and v3
ASPaS Framework

- SIMD Merger
  - Generalize the patterns required in the in-register merge
ASPaS Framework

- SIMD Merger
  - Generalize the patterns required in the in-register merge
ASPaS Framework

- SIMD Merger
  - Generalize the patterns required in the in-register merge
ASPaS Framework

• SIMD Merger
  – Generalize the patterns required in the in-register merge

\[
\begin{align*}
  v_0 &= 3 \\
  v_1 &= 5 \\
  v_2 &= 7 \\
  v_3 &= 9 \\
  u_0 &= 6 \\
  u_1 &= 5 \\
  u_2 &= 2 \\
  u_3 &= 1 \\
\end{align*}
\]
ASPaS Framework

- SIMD Merger
  - Generalize the patterns required in the in-register merge
ASPaS Framework

- SIMD Merger
  - Generalize the patterns required in the in-register merge

Inconsistent patterns

Some data-reordering patterns
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm
ASPaS Framework

• SIMD Code Generator
  – Sequence Building Algorithm
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm

Default Vector

Sequence Building Algorithm

Target Vector

Initial Lane Check

BAFE
DCHG
ABCD
EFGH
ASPaS Framework

• SIMD Code Generator
  – Sequence Building Algorithm

Initial Lane Check

Default Vector

Sequence Building Algorithm

Target Vector
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm

- Initial Lane Check
- Lane Check
- Blend Primitive
- Inter-lane Permute Primitive
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm

**Intra-lane**
Permute Primitive

**Initial Lane Check**

**Elem Check**

**Lane Check**

**Blend Primitive**

**Inter-lane**
Permute Primitive

**Default Vector**
Sequence Building Algorithm

**Target Vector**
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm

**Intra-lane**
Permute Primitive

**Elem Check**

**Initial Lane**

- BADC
- FEHG
- ABCD
- EFGH

**Blend Primitive**

**Inter-lane**
Permute Primitive

**Default Vector**

**Sequence Building Algorithm**

**Target Vector**

**Lane Check**
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm

- Intra-lane Permute Primitive
- Initial Lane Check
- Elem Check
- Lane Check
- Blend Primitive
- Inter-lane Permute Primitive

- Sequence Building Algorithm
- Default Vector
- Target Vector

Kaixi@VT
8/21/2015
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm

Intra-lane Permute Primitive

Initial Lane Check

Elem Check

Blend Primitive

Inter-lane Permute Primitive

Default Vector

Sequence Building Algorithm

Target Vector

Selected Primitive

Selected Primitive Seq

ABCDEF

ABCDEF
ASPaS Framework

• SIMD Code Generator
  – Sequence Building Algorithm
ASPaS Framework

- SIMD Code Generator
  - Translate: selected primitive sequence to real codes
ASPaS Framework

• SIMD Code Generator
  – Translate: selected primitive sequence to real codes
    • Intra-lane permute primitive => _mm512_shuffle
    • Inter-lane permute primitive => _mm512_permute4f128
    • Blend primitive => mask integrated to bond shuffle/permute instructions
ASPaS Framework

• SIMD Code Generator
  – **Translate**: selected primitive sequence to real codes
    • Intra-lane permute primitive => `_mm512_shuffle`
    • Inter-lane permute primitive => `_mm512_permute4f128`
    • Blend primitive => mask integrated to bond shuffle/permute instructions
  – Towards TLP
ASPaS Framework

• SIMD Code Generator
  – **Translate**: selected primitive sequence to real codes
    • Intra-lane permute primitive => _mm512_shuffle
    • Inter-lane permute primitive => _mm512_permute4f128
    • Blend primitive => mask integrated to bond shuffle/permute instructions
  – Towards TLP
    • Threads sort their own parts (aspas::sort())
    • Half of them merge the adjacent parts (aspas::merge())
      – Continues until only one thread left
Evaluation & Discussion

• Experiment Setup

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIC</td>
<td>Intel Xeon Phi 5110P</td>
</tr>
<tr>
<td>Code Name</td>
<td>Knights Corner</td>
</tr>
<tr>
<td># of Cores</td>
<td>60</td>
</tr>
<tr>
<td>Clock Rate</td>
<td>1.05 GHz</td>
</tr>
<tr>
<td>L1/L2 Cache</td>
<td>32 KB/ 512 KB</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB GDDR5</td>
</tr>
<tr>
<td>Compiler</td>
<td>icpc 13.0.1</td>
</tr>
<tr>
<td>Compiler Options</td>
<td>-mmic -O3</td>
</tr>
<tr>
<td>Random Number Range</td>
<td>[0, DATA_SIZE]</td>
</tr>
</tbody>
</table>
Evaluation & Discussion

- Performance of Different Sorting Networks

![Graph showing performance comparison between different sorting networks: aspas_sort() and aspas_merge().](image)
Evaluation & Discussion

- Performance of Different Sorting Networks

- ASPaS_sort(): More comparators, worse performance
Evaluation & Discussion

- Performance of Different Sorting Networks

- ASPaS_sort(): More comparators, worse performance
- ASPaS_merge(): “Consistent” variant consists of the SIMD-unfriendly interleaving data-reordering
Evaluation & Discussion

- Vectorization Efficiency

**Sort stage**

![Graph showing performance comparison between different stages and data sizes for the sort stage.](image1)

**Merge stage**

![Graph showing performance comparison between different stages and data sizes for the merge stage.](image2)

*Lower the better*
Evaluation & Discussion

- Vectorization Efficiency
  - **Sort stage**: ASPaS still can outperform the auto-vec version, thanks to its contiguous memory access
  - **Merge stage**: Lower the better
Evaluation & Discussion

- Vectorization Efficiency
  
  - **Sort stage**: ASPaS still can outperform the auto-vec version, thanks to its contiguous memory access.
  
  - **Merge stage**: the complex data dependency prevents the compiler from auto-vectorizing the loops.
Evaluation & Discussion

- Comparison to Sorting from Libraries

**Single-threaded sorts**

**Multi-threaded sorts**

*Lower the better*
Evaluation & Discussion

• Comparison to Sorting from Libraries

Single-threaded sorts
- ASPaS sort outperforms other sorting tools from widely-used libraries

Multi-threaded sorts

Lower the better
Conclusion

• ASPaS: a framework for the Automatic SIMDization of Parallel Sorting code generation
  – Formalizes the data-reordering operations
  – Fast and efficiently build the real instruction sequences
  – Can be applied to CPU as well

• Various parallel sorting codes generated with ASPaS
  – Significant vectorization efficiency
  – Can outperform tools from STL, Boost, and Intel TBB
Conclusion

• ASPaS: a framework for the Automatic SIMDization of Parallel Sorting code generation
  – Formalizes the data-reordering operations
  – Fast and efficiently build the real instruction sequences
  – Can be applied to CPU as well

• Various parallel sorting codes generated with ASPaS
  – Significant vectorization efficiency
  – Can outperform tools from STL, Boost, and Intel TBB

THANK YOU!

Paper in ACM’s ICS’15 and http://synergy.cs.vt.edu
Backup

• Portability
  – Easily ported to other x-86 multi-core CPU architectures
  – Only need to change the part of “Translate: primitives to real codes” in the SIMD Code Generator
    • Permute primitives => _mm256_shuffle/permute2f128
    • Blend primitives => e.g. _mm256_unpacklo/unpackhi