ASPaS: A Framework for Automatic SIMDization of Parallel Sorting on x86-based Many-core Processors

Kaixi Hou, Hao Wang, Wu-chun Feng

{kaixihou,hwang121,wfeng}@vt.edu

VirginiaTech *Invent the Future*

SyNeRG synergy.cs.vt.edu

Why sorting?

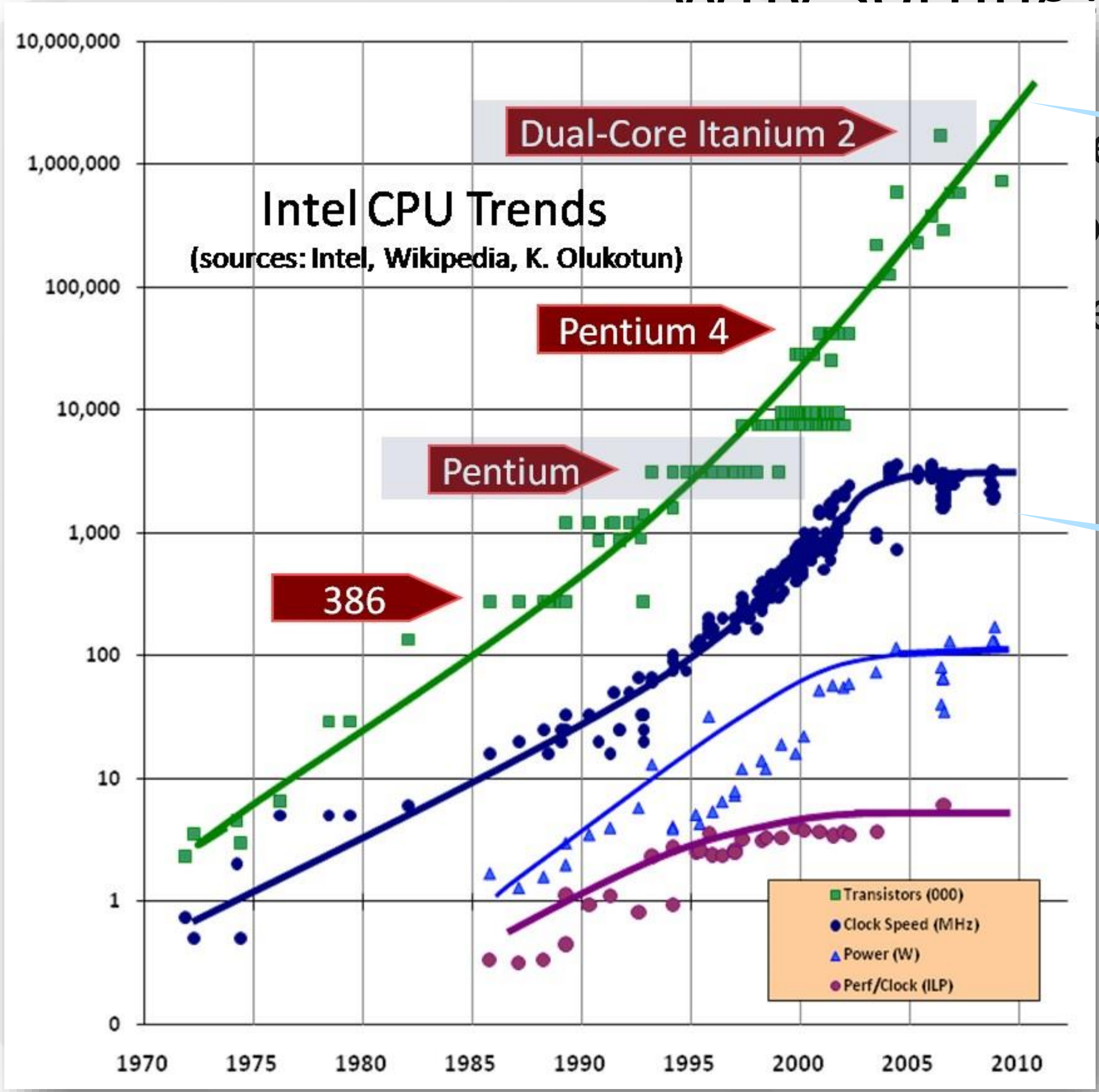
- Sorting used as a important primitive in many applications
  - Databases, computational biology, graph algorithms, etc.
- To get efficient sort, all computing resources need to be used

Why sorting?

e in many applications

h al

eso



More DLP and TLP

Cannot just rely on clock rate

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
- 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

Serial C codes

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

Ex. Interleave two arrays

AVX intrinsics on CPUs

```
__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

```
__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);
```

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

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

Ex. Interleave two arrays

AVX intrinsics on CPUs

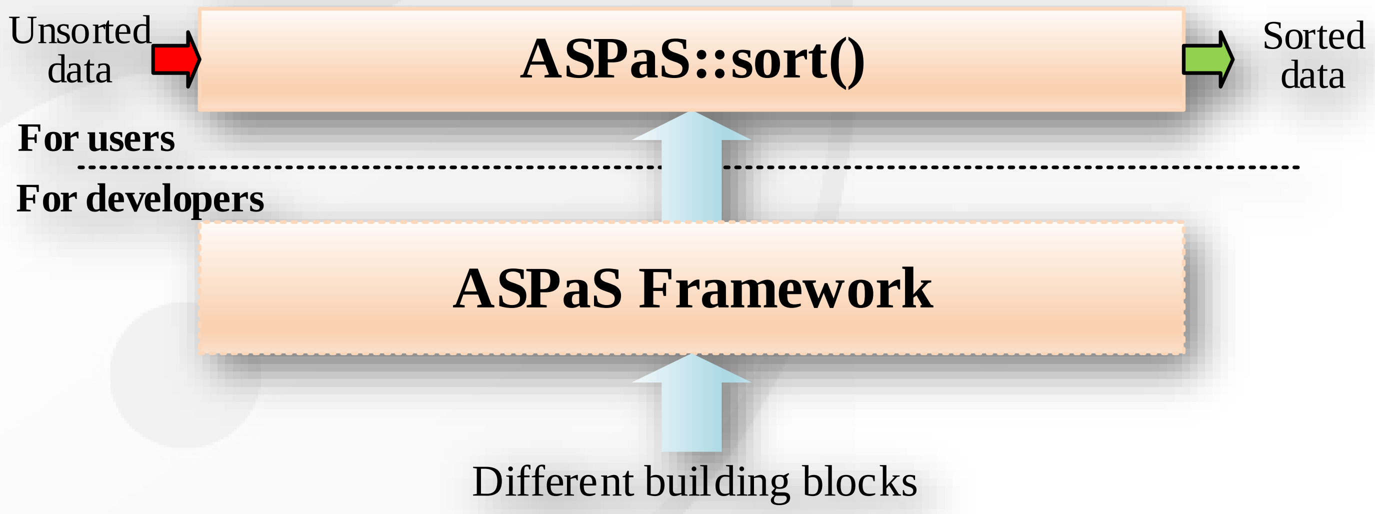
```
__mm256 v1 = _mm256_unpacklo_ps(in      inpB);
__mm256 v2 = _mm256_un     hi_p        inpB);
__mm256 trgA    mm256_            ps(v1    , 0x20);
__mm256 trgB                          v2, 0x31);
```

```
__mm512i l = _mm5                                 DAC);
__mm512i h = _mm51                               _BDAC);
__mm512i t                                 SWIZ_REG_BADC);
__mm512i t1  = _                   l, _        WIZ_REG_BADC);
__mm512i l = _mm5                  x0f0f, t0 , _MM_PERM_CDAB);
__mm512i h = _                 i32(t0 , 0xf0f0, t1 , _MM_PERM_CDAB);
__mm512i trgA = _mm512          2(l, _MM_PERM_BDAC);
__mm512i trgB = _mm512     ffle_epi32(h, _MM_PERM_BDAC);
```

Can they be automatically generated?

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



Roadmap

- Introduction & Motivation
- Background
  - Sorting Networks & SIMD Processing
- ASPaS Framework
  - SIMD Sorter
  - SIMD Transposer

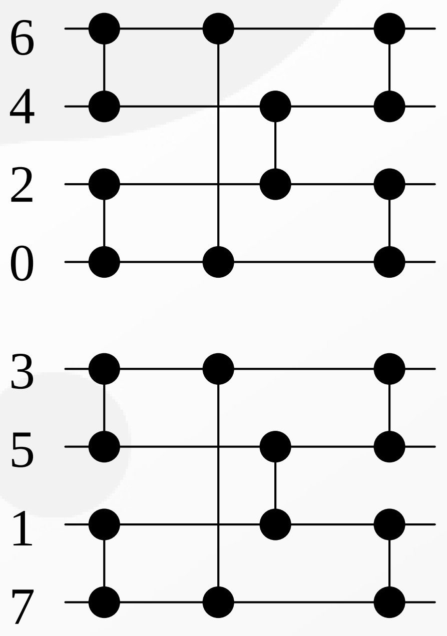
    ▷ Generate patterns for sorting the data segment by segment
  - SIMD Merger

    ▷ Generate patterns for merging the sorted data
  - SIMD Code Generator

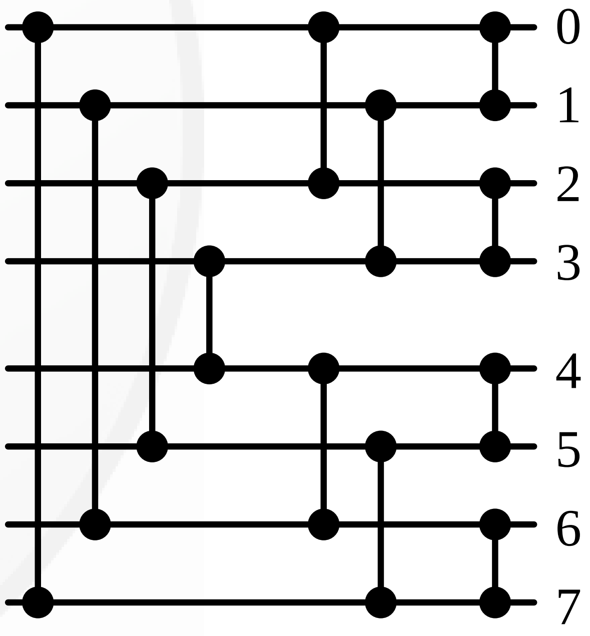
    ▷ Generate codes from the patterns
- Evaluation & Discussion
- Conclusion

Background: Sorting Networks

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



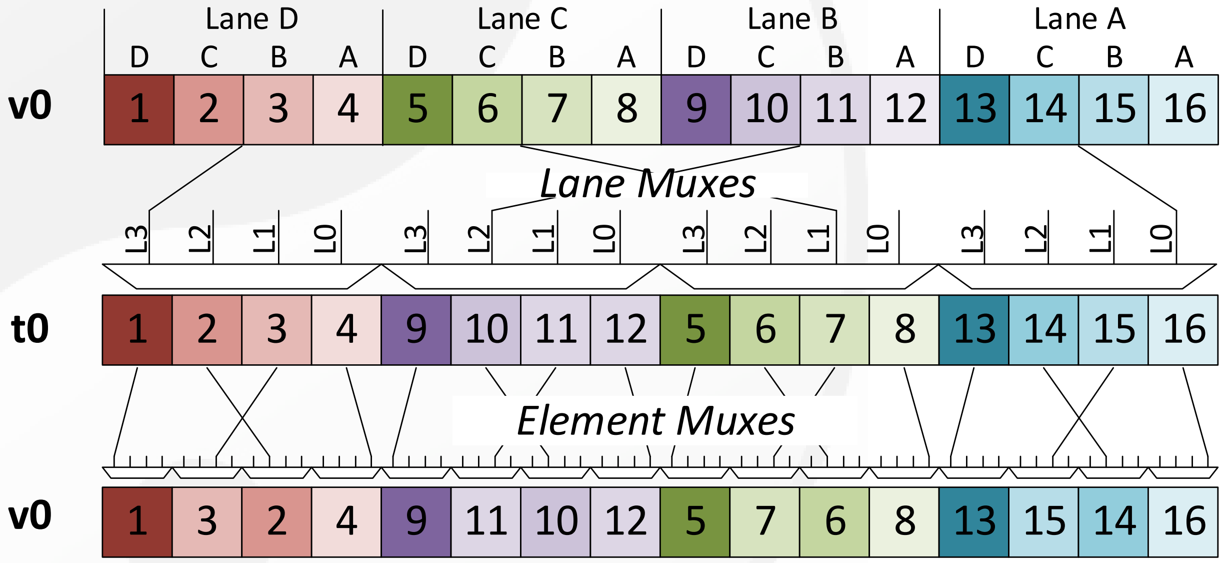
Two sorting networks



Merging network

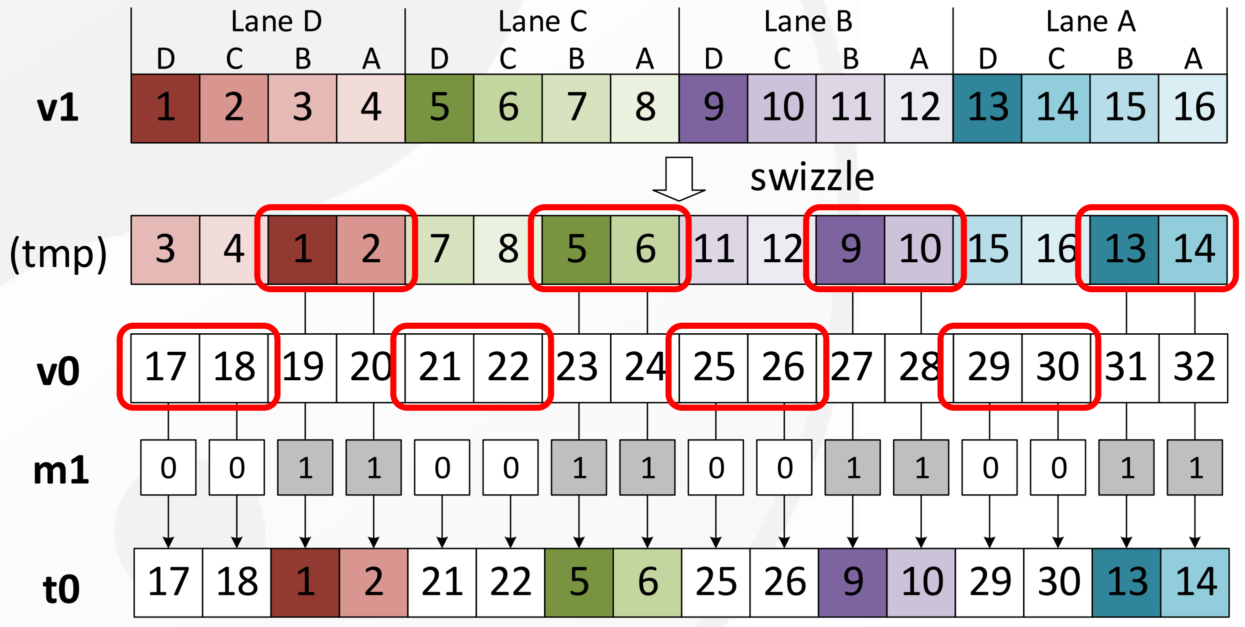
Background: SIMD for Intel Xeon Phi

- VPU Architecture
  - Manipulate one vector



Background: SIMD for Intel Xeon Phi

- VPU Architecture
  - Manipulate two vectors



Roadmap

- Introduction & Motivation
- Background
  - VPU Architecture & Sorting Networks
- ASPaS Framework
  - SIMD Sorter
  - SIMD Transposer
  - SIMD Merger
  - SIMD Code Generator
- Evaluation & Discussion
- Conclusion

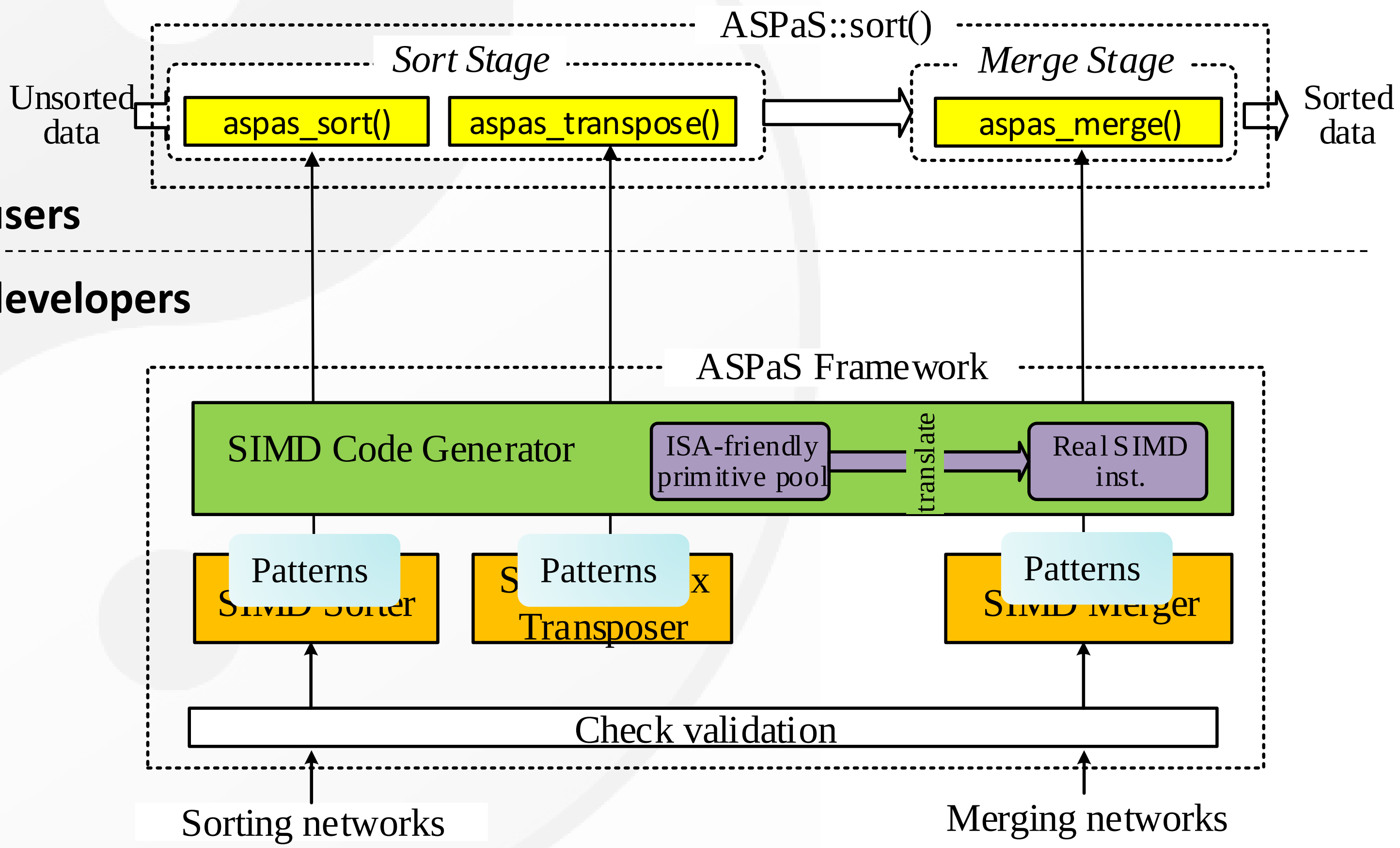
} ▷ Generate patterns for sorting the data segment by segment

▷ Generate patterns for merging the sorted data

▷ Generate codes from the patterns

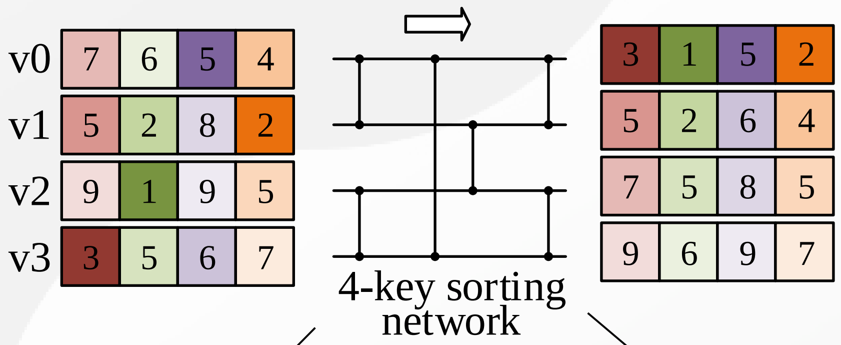
ASPaS Framework

- ASPaS Structure Overview



ASPaS Framework

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



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

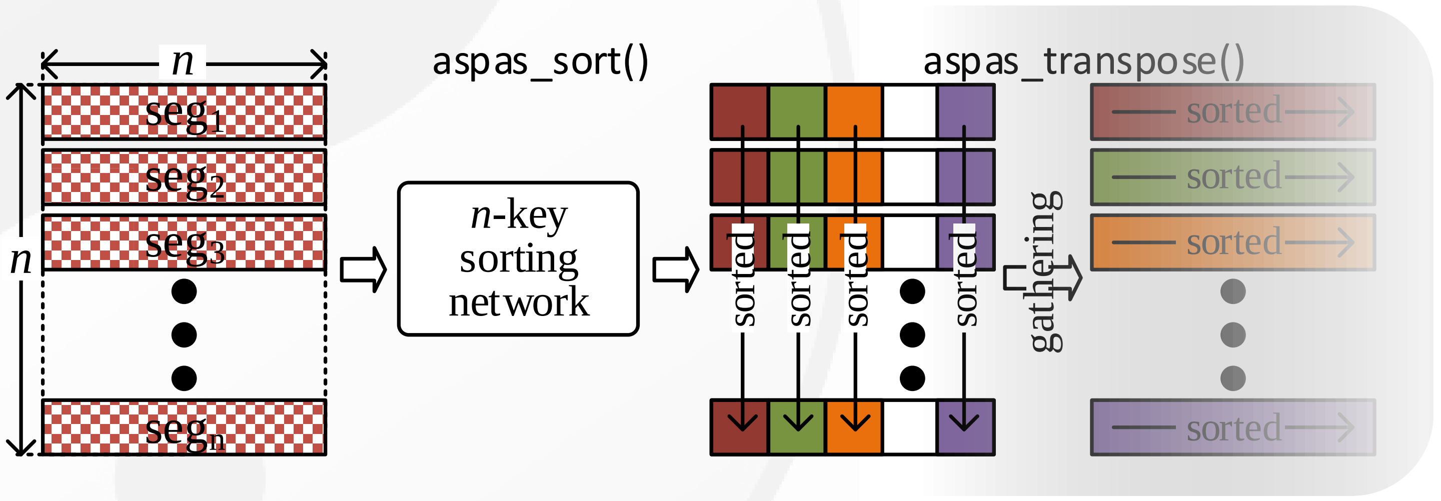
Input Macros

0. $S_2(0, 1)$; $S_2(2, 3)$;
1. $S_2(0, 3)$; $S_2(1, 2)$;
2. $S_2(0, 1)$; $S_2(2, 3)$;

Minimized Grouping

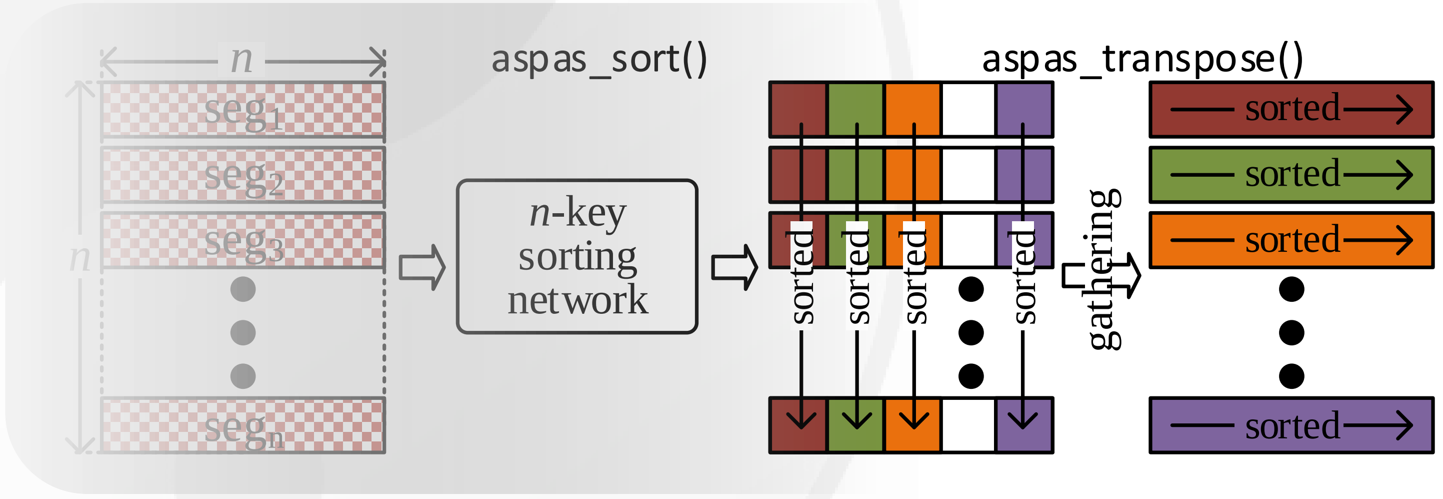
ASPaS Framework

- SIMD Transposer
  - Why need the transpose?



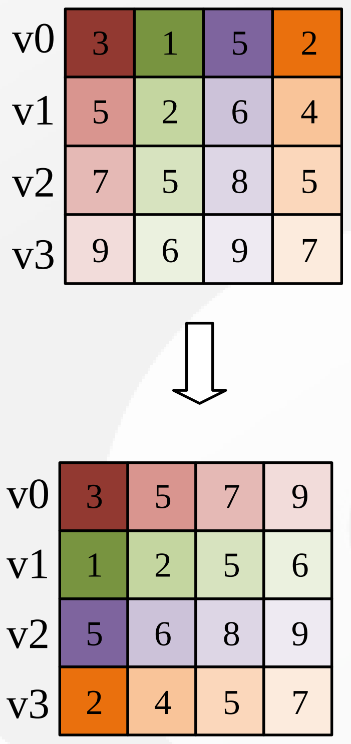
ASPaS Framework

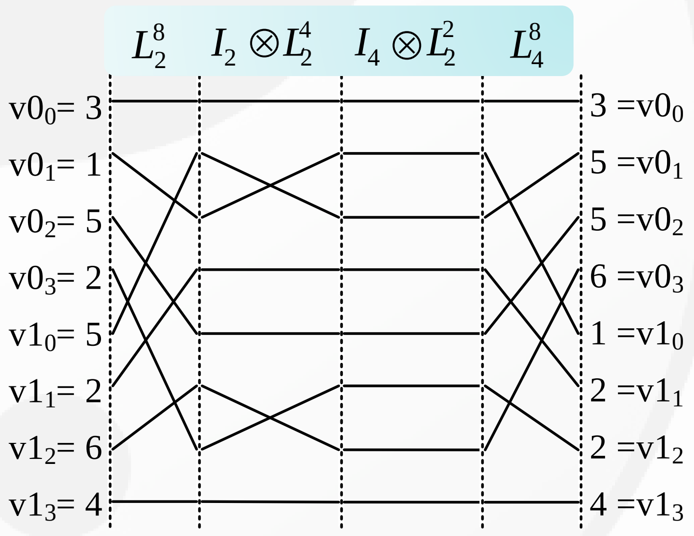
- SIMD Transposer
  - Why need the transpose?



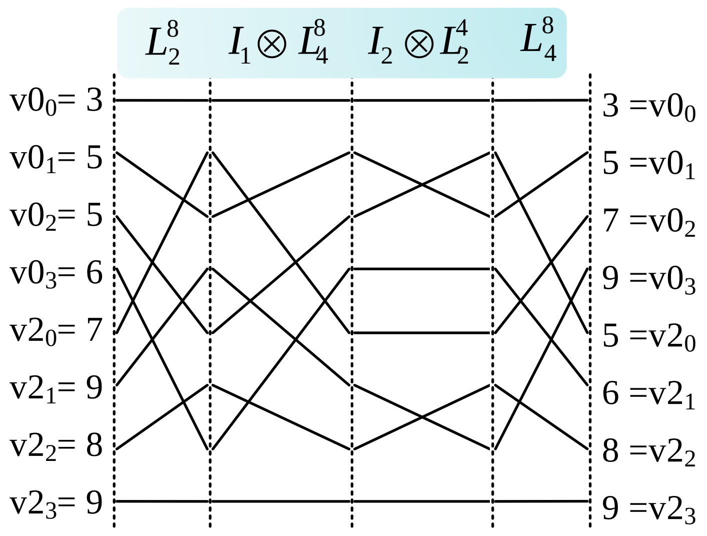
ASPaS Framework

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





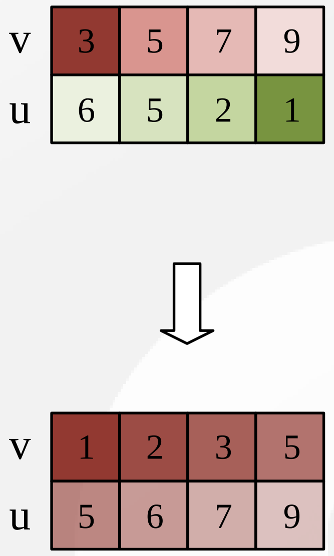
Same pattern applies on v2 and v3

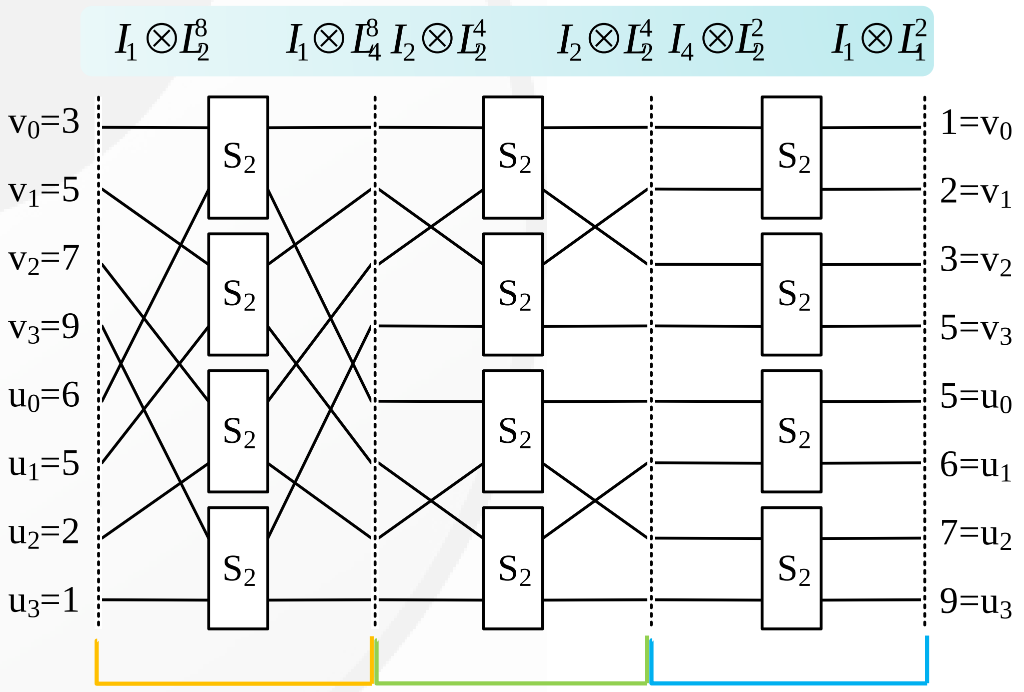


Same pattern applies on v1 and v3

ASPaS Framework

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

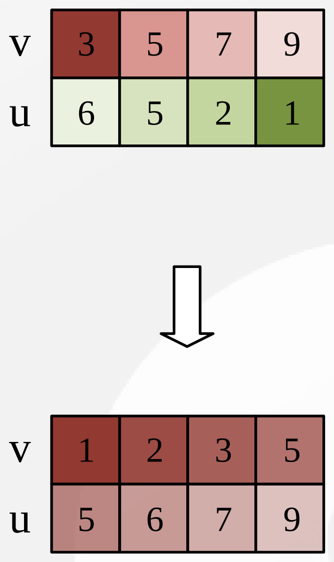


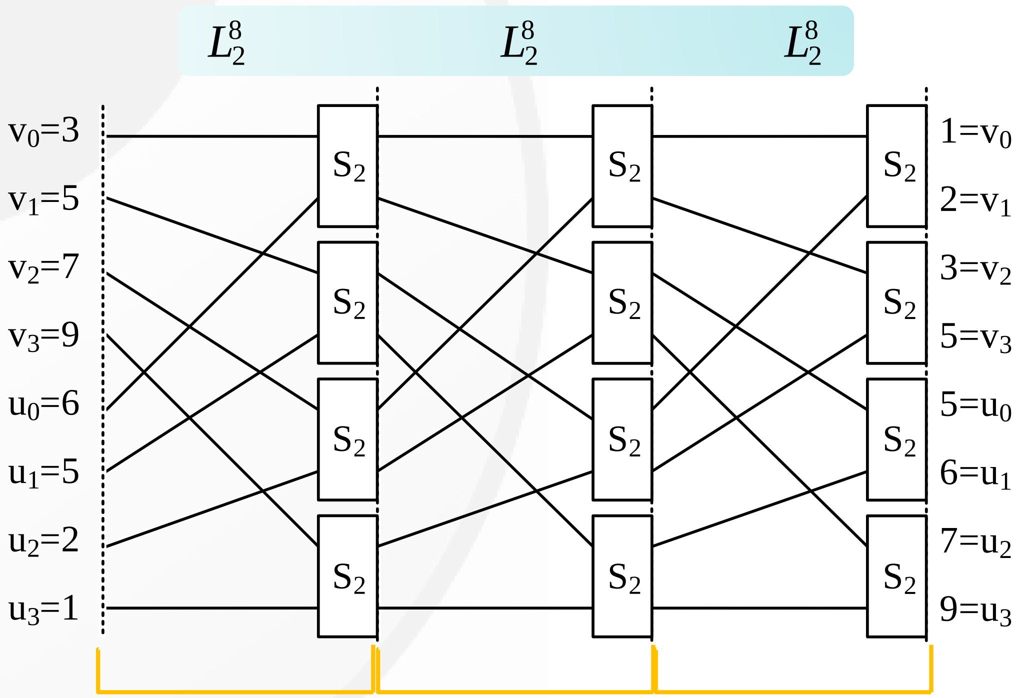


Inconsistent patterns

ASPaS Framework

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



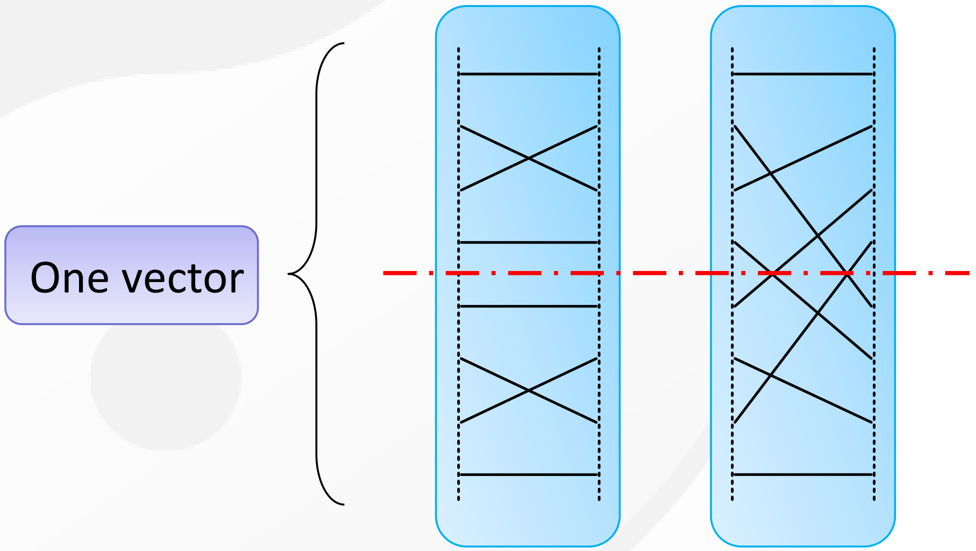


Consistent patterns

ASPaS Framework

- SIMD Code Generator
  - Generate SIMD codes based on the received patterns
  - Primitive Pool Building

① **Permute Primitives** <*Unique* and *symmetric* data-reordering>



ASPaS Framework

- SIMD Code Generator
  - Generate SIMD codes based on the received patterns
  - Primitive Pool Building

① **Permute Primitives** <*Unique* and *symmetric* data-reordering>

Suppose there are 4 units in vector

Decreasing # of Permute Primitives

4^4=256 possible permutations

Permutations w/o repetition => 4!=24

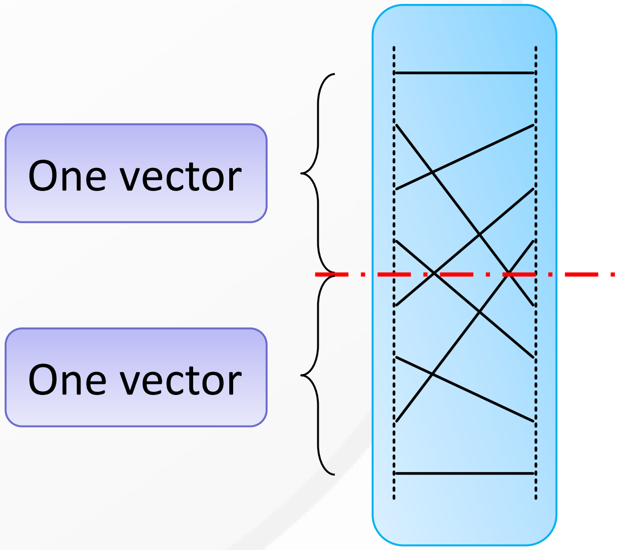
Symmetric permutations => 8 DCBA(original order), DBCA, CDAB, BDAC, BADC, CADB, ACBD, and ABCD

ASPaS Framework

- SIMD Code Generator
  - Generate SIMD codes based on the received patterns
  - Primitive Pool Building

① Permute Primitives

② **Blend Primitives** <*Symmetric* and *equal* data-blending>



ASPaS Framework

- SIMD Code Generator
  - Generate SIMD codes based on the received patterns
  - Primitive Pool Building

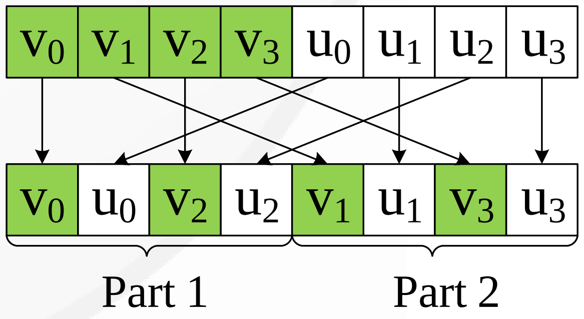
① Permute Primitives

② **Blend Primitives** <*Symmetric* and *equal* data-blending>

Suppose there are 4 elements in vector

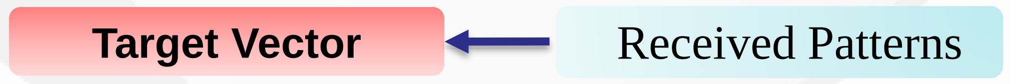
Only need 2 blend primitives to select every 1, 2 (*1 to log(W)*) elements from two input vectors respectively

- E.g. 1010


v_0 v_1 v_2 v_3 u_0 u_1 u_2 u_3
v_0 u_0 v_2 u_2 v_1 u_1 v_3 u_3
Part 1
Part 2

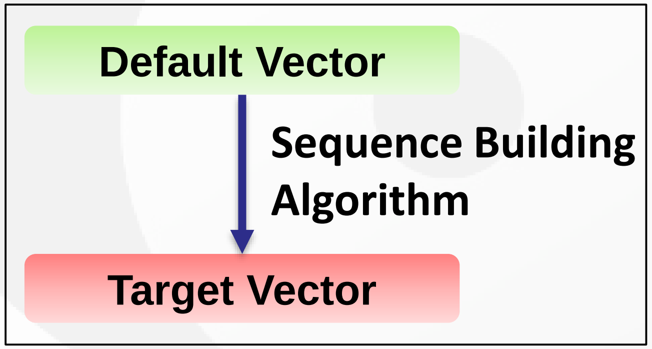
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm



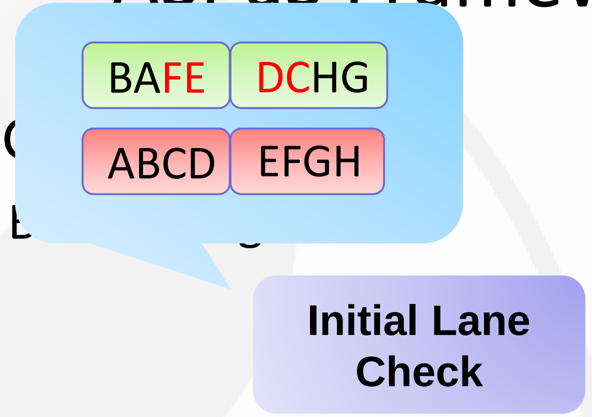
ASPaS Framework

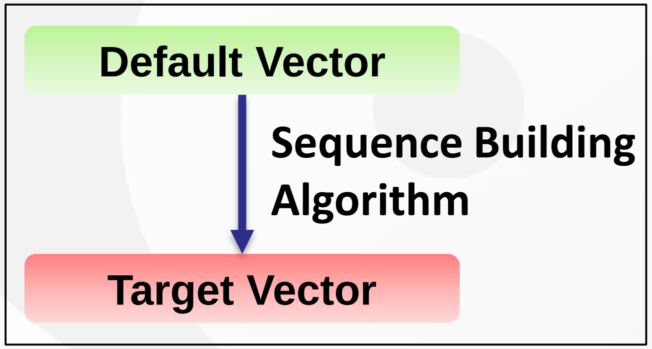
- SIMD Code Generator
  - Sequence Building Algorithm



ASPaS Framework

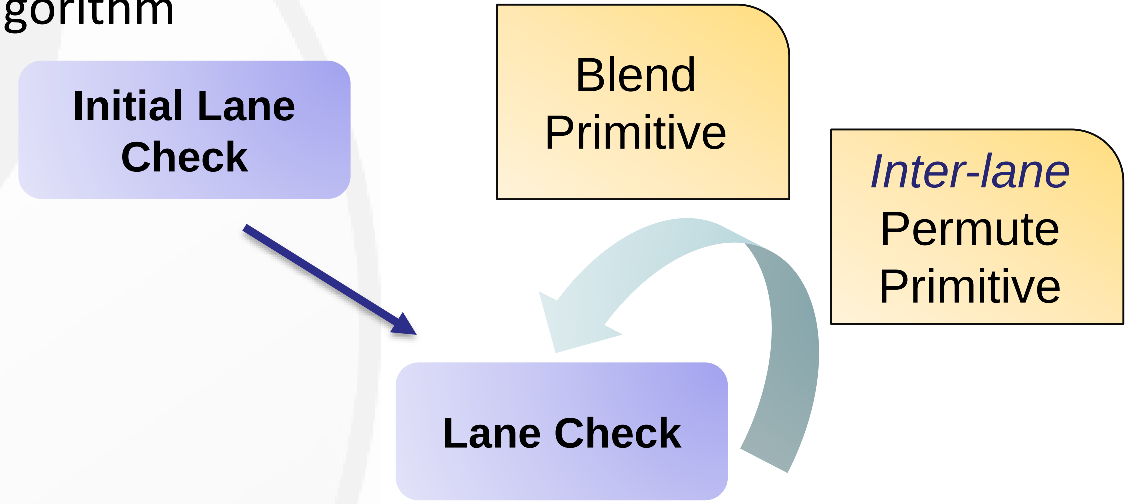
- SIMD Code (
  - Sequence E

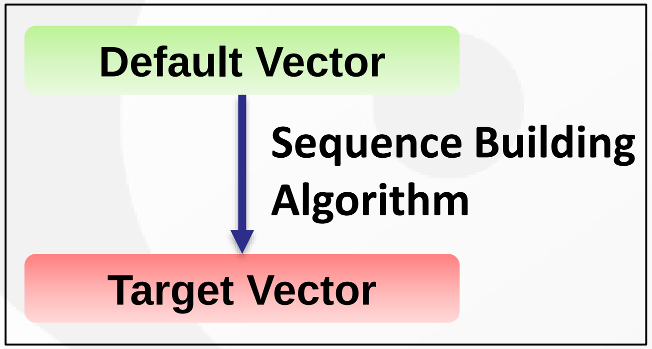




ASPaS Framework

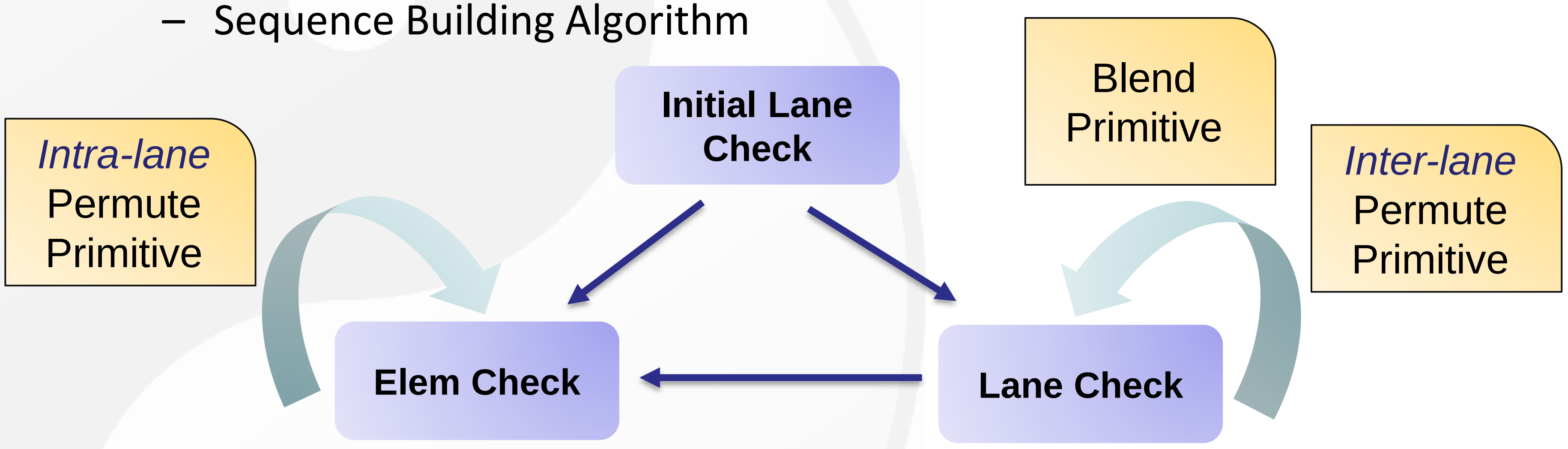
- SIMD Code Generator
  - Sequence Building Algorithm

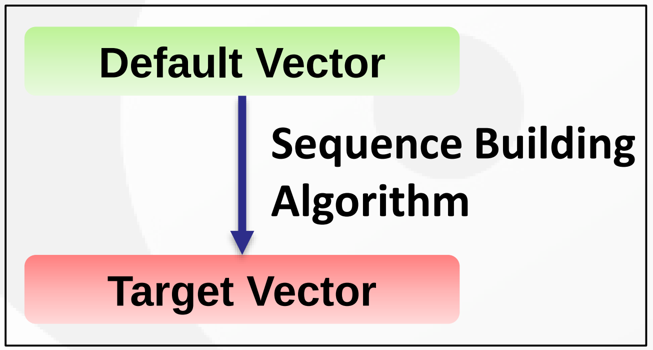




ASPaS Framework

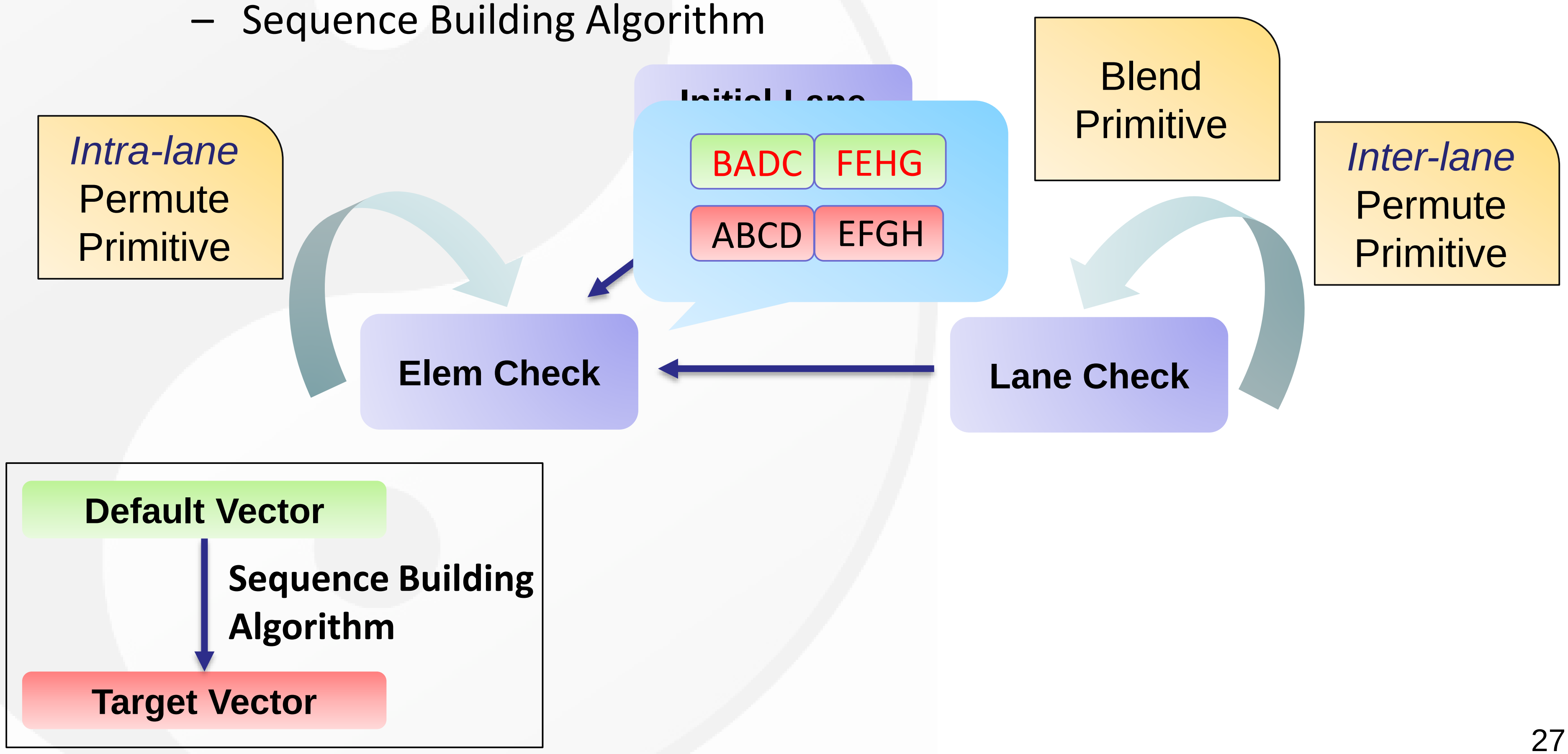
- SIMD Code Generator
  - Sequence Building Algorithm


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



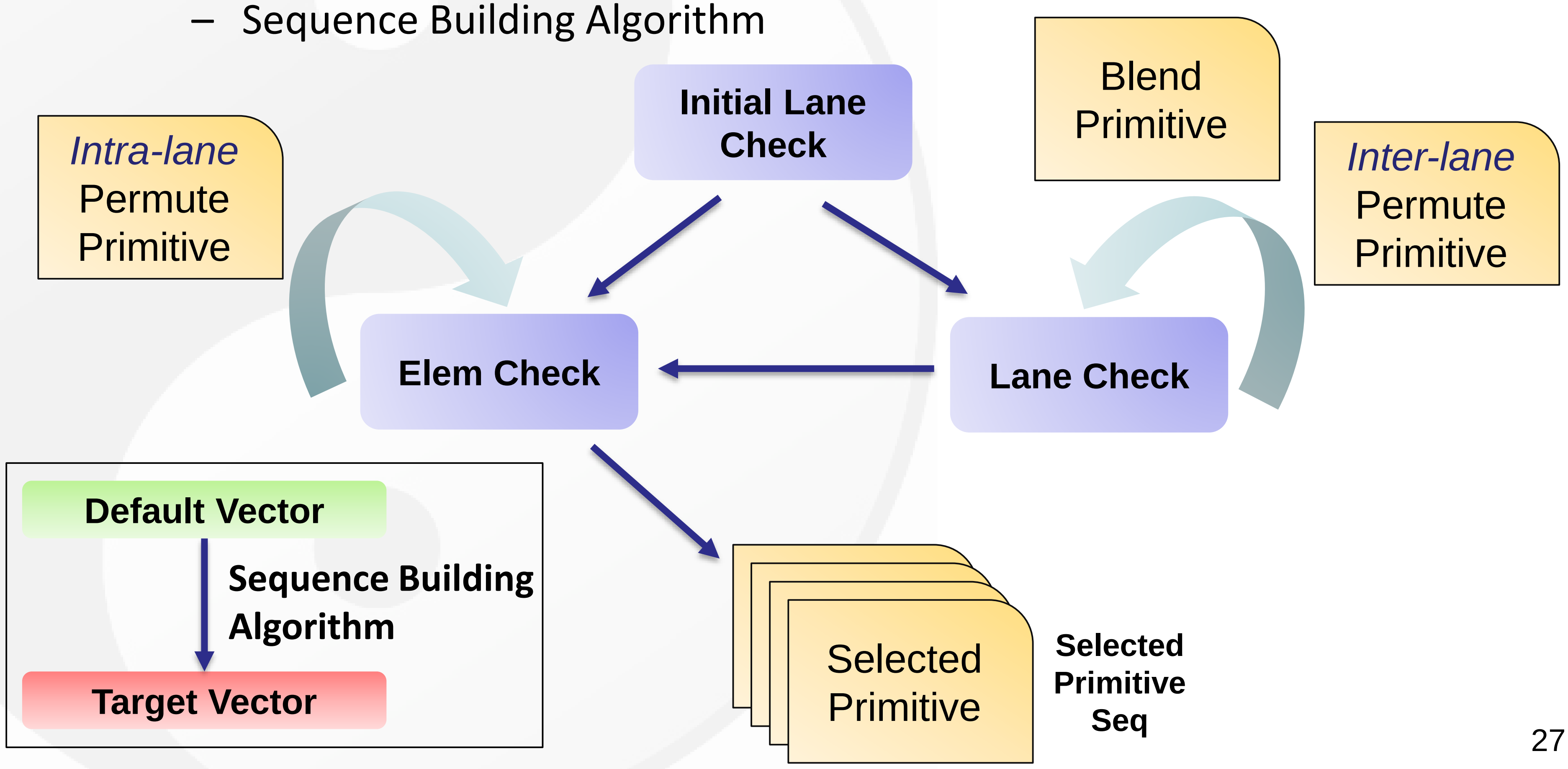
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm



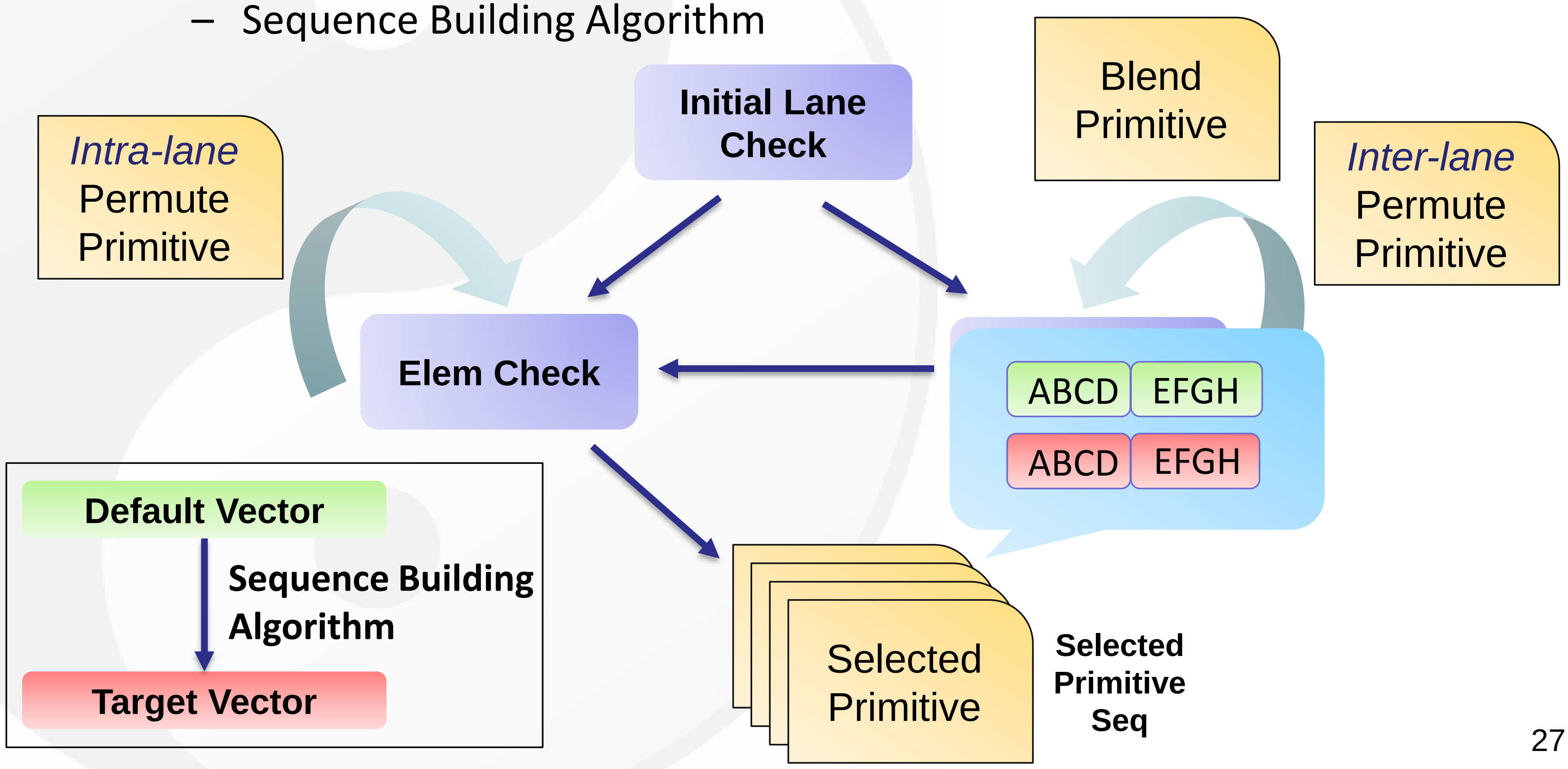
ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm



ASPaS Framework

- SIMD Code Generator
  - Sequence Building Algorithm



ASPaS Framework

- SIMD Code Generator
  - 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

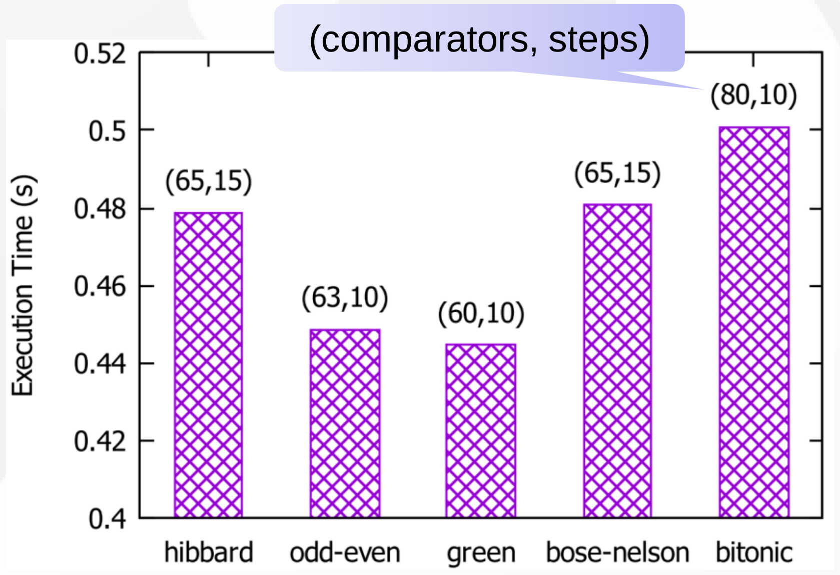
Evaluation & Discussion

- Experiment Setup

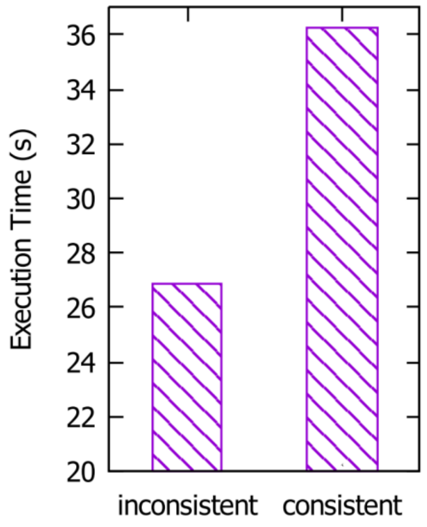
Parameter	Value
MIC	Intel Xeon Phi 5110P
Code Name	Knights Corner
# of Cores	60
Clock Rate	1.05 GHz
L1/L2 Cache	32 KB/ 512 KB
Memory	8 GB GDDR5
Compiler	icpc 13.0.1
Compiler Options	-mmic -O3
Random Number Range	[0, DATA_SIZE]

Evaluation & Discussion

- Performance of Different Sorting Networks



aspas_sort()



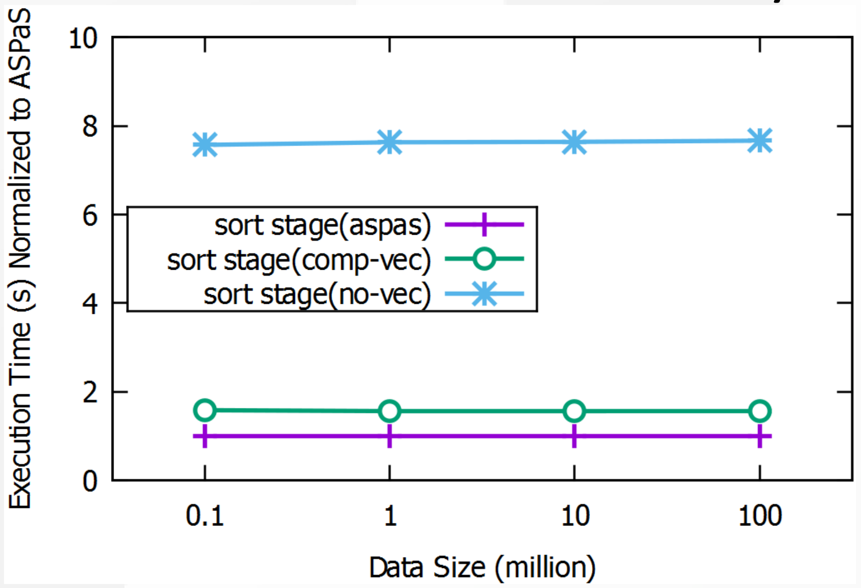
aspas_merge()

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

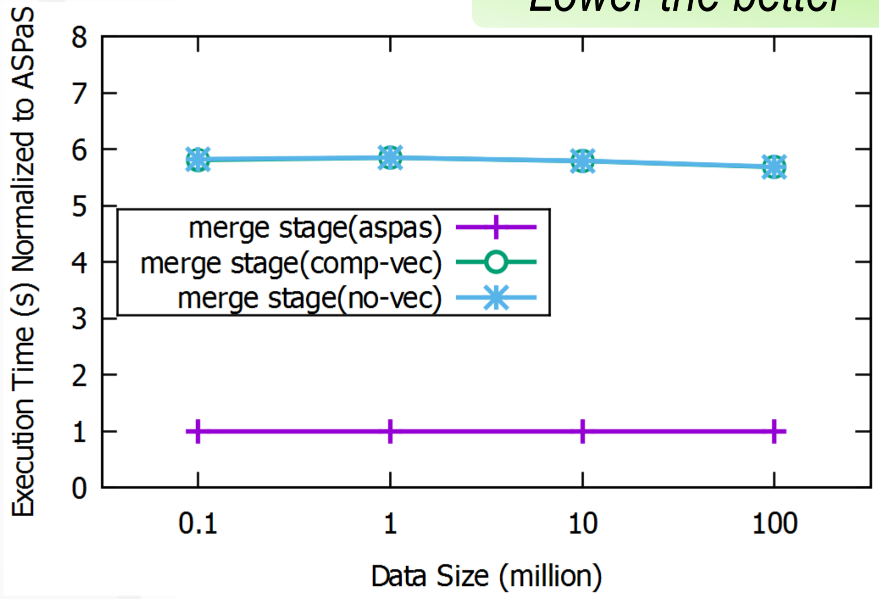
Evaluation & Discussion

- Vectorization Efficiency

Lower the better



Sort stage



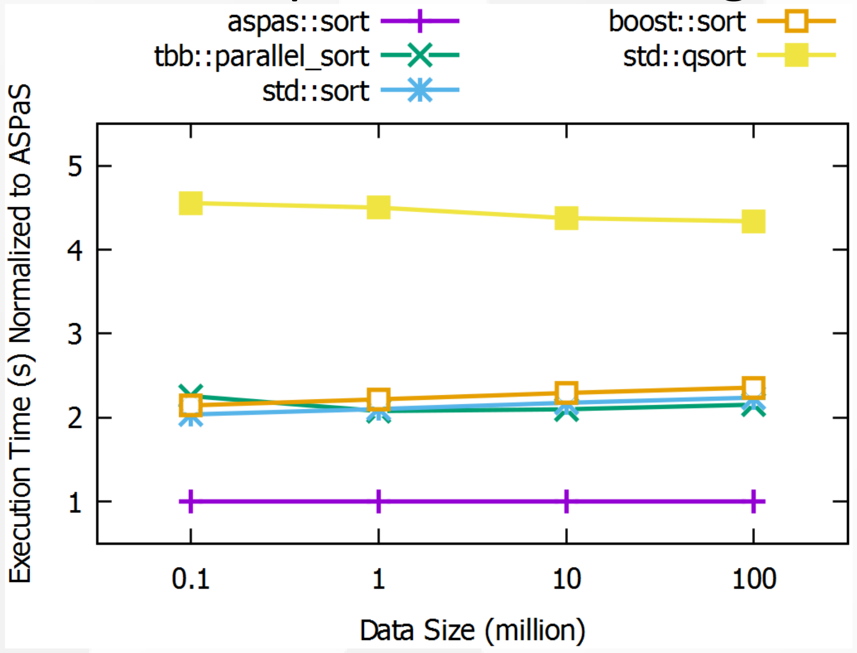
Merge stage

– 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

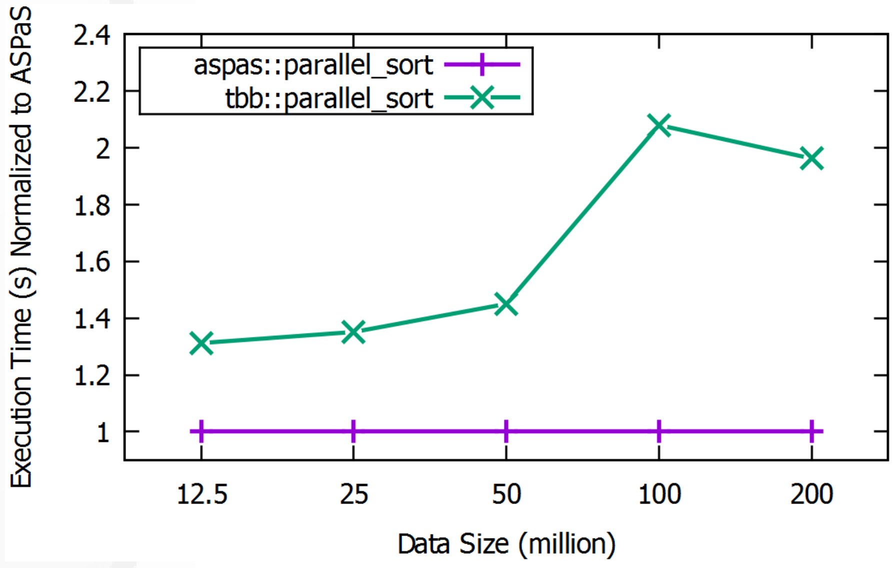
Evaluation & Discussion

- Comparison to Sorting from Libraries

Lower the better



Single-threaded sorts



Multi-threaded sorts

– ASPaS sort outperforms other sorting tools from widely-used libraries

Discussion

- 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

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!

More info: http://synergy.cs.vt.edu