FESIA: A Fast and SIMD-Efficient Set Intersection Approach on Modern CPUs

Jiyuan Zhang (jiyuanz@andrew.cmu.edu), Daniele G. Spampinato, Franz Franchetti

CMU ECE

Carnegie Mellon University

Electrical & Computer ENGINEERING

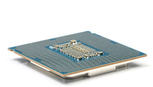
Background

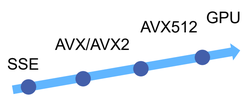
Set intersection is a functional primitive used in:

Data query processing

Social network analytics

Data parallelism is a common feature in the architectures of modern processors. And it has the trend of becoming wider and wider.





Approaches for set intersection

Example: hash, tree-based data structures for set intersections

✓ Good runtime complexity

✖ Hard to leverage vectorization features on modern processors, therefore cannot achieve speedups in practice

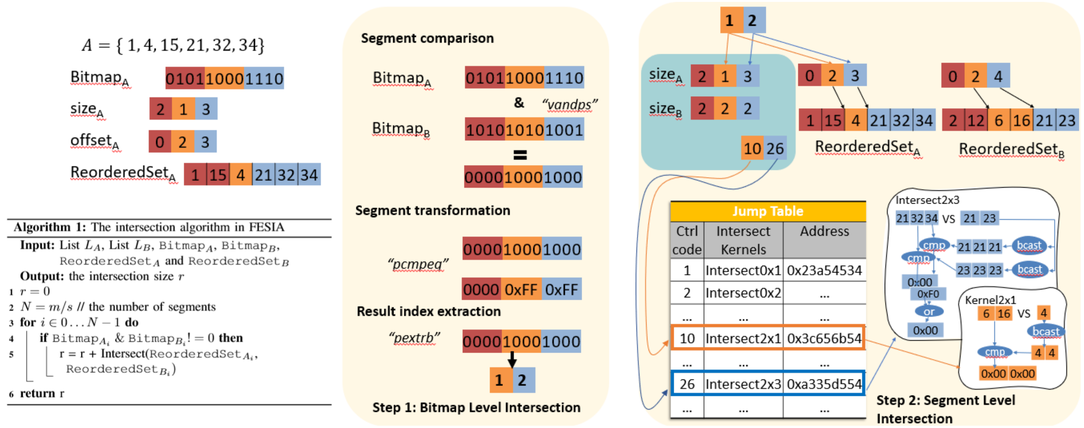
Vectorizations for set intersection

Merge-based intersection has irregular patterns of computations. It is non-trivial on how to leverage SIMD acceleration.

```
1  int scalar_merge_intersection(int L1[],
2              int n1, int L2[], int n2) {
3    int i = 0, j = 0, r = 0;
4    while (i < n1 && j < n2) {
5      if (L1[i] < L2[j]) {
6        i++;
7      } else if (L1[i] > L2[j]) {
8        j++;
9      } else {
10       i++; j++; r++;
11     }
12   }
13   return r;
14 }
```

Listing 1. A code example of scalar merge-based set intersection

FESIA: A Fast and SIMD-Efficient Set Intersection Approach



A segmented-bitmap data structure, and a two-step intersection approach: (1) the bitmaps are used to filter out unmatched elements, and (2) a segment-by-segment comparison is conducted to compute the final set intersection using specialized SIMD kernels.

Step 1:Bitmap-level intersection

(1) Bitwise-AND on bitmaps

e.g., *vandps*

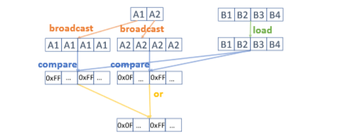
(2) Segment transformatiosn

- ❑ e.g, *pcmpeqw ,pcmpeqq,* with s = segment size
- ❑ the output grouped by the segment size, e.g., 0xFFFF (s=16)

(3) Non-zero segment index extraction

e.g., *pextrb*

Step 2: Segment-level intersection




```
vb = _mm_load_si128(B);
v1 = _mm_set1_epi32(A[0]);
c1 = _mm_cmp_epi32(v1, vb);

v2 = _mm_set1_epi32(A[1]);
c2 = _mm_cmp_epi32(v2, vb);

v3 = _mm_set1_epi32(A[2]);
c3 = _mm_cmp_epi32(v3, vb);

v4 = _mm_set1_epi32(A[3]);
c4 = _mm_cmp_epi32(v4, vb);

cmp =
_mm_or_si128(_mm_or_si128(c1,
c2), _mm_or_si128(c3, c4));
mask = _mm_movemask_ps(cmp);
count += _mm_popcnt_32(mask);
return count;
```

Specialized 2x4 code

```
1  Function: Intersect4by5(A, B)
2    v_a = _mm_load_si128(A)
3    v_1 = _mm_set1_epi32(B_1)
4    c_1 = _mm_cmp_epi32(v_1, v_a)
5    v_2 = _mm_set1_epi32(B_2)
6    c_2 = _mm_cmp_epi32(v_2, v_a)
7    v_3 = _mm_set1_epi32(B_3)
8    c_3 = _mm_cmp_epi32(v_3, v_a)
9    v_4 = _mm_set1_epi32(B_4)
10   c_4 = _mm_cmp_epi32(v_4, v_a)
11   /* v_5 and c_5 are omitted */
12   t_1 = _mm_or_si128(c_1, c_2)
13   t_2 = _mm_or_si128(c_3, c_4)
14   t_3 = _mm_or_si128(t_2, c_5)
15   cmp = _mm_or_si128(t_1, t_3)
16   mask = _mm_movemask_ps(cmp)
17   count = _mm_popcnt_32(mask)
18   return count
```

Specialized Intersect Kernels

The summary of FESIA VS. state-of-the-art intersection approaches

Methods	Complexity	Small Intersect	SIMD	Multicore	$n_1 \ll n_2$	k-way intersection	Portable
FESIA	$n/\sqrt{w}+r$	✓	✓	✓	$\min(n_1, n_2)$	$kn/\sqrt{w}+r$	✓
BMiss [1]	n_1+n_2	✓	✓		n_1+n_2	$n_1\cdots+n_k$	✓
Galloping [2]	$n_1 \log n_2$		✓		$n_1 \log n_2$	$n_1(\log n_2+\cdots+\log n_k)$	
Hiera [3]	n_1+n_2		✓		n_1+n_2	$n_1\cdots+n_k$	
Fast [4]	$n/\sqrt{w}+r$	✓			$n/\sqrt{w}+r$	$n/\sqrt{w}+kr$	

w indicates the SIMD width, and n and r indicate the size of the input set and the intersection size

```
1  /* the dispatch control code */
2  int ctrl = ((Sa << 3) | Sb);
3  /* jump to a specialized intersection kernel */
4  switch(ctrl & 0x7F){
5  /* Each specialized kernel is a macro */
6      case 0: break; //kernel0x0
7      case 1: break; //kernel0x1
8      case 2: break; //kernel0x2
9      /* case 3 to 62 are omitted .... */
10     case 63: Kernel7x7; break;
11     default: GeneralIntersection(); break;
12 }
```

Listing 2. The jump table for specialized kernels

Runtime dispatch

Small-size intersect kernel speedups

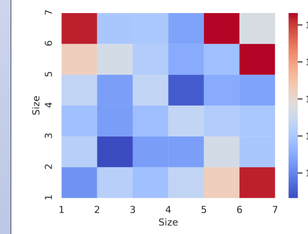


Fig. 4. Speedups of SSE kernels

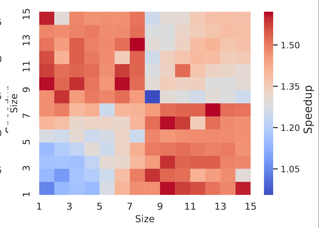


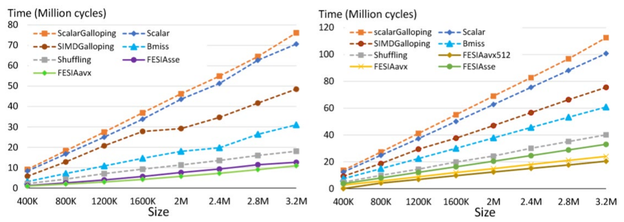
Fig. 5. Speedups of AVX kernels

Performance on varying input size

Results are shown in two Intel architectures: Haswell and Skylake.

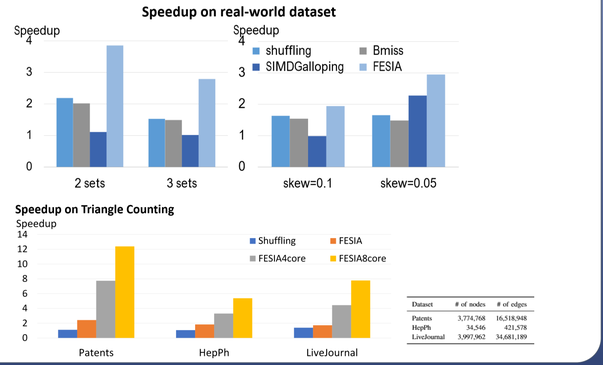
The relative performance of these methods remains consistent as we increase the input size.

FESIA outperforms other scalar and SIMD set intersection methods.



Real-world datasets

We study the performance on two real-world tasks: (1) a database query task, and (2) a triangle counting task in graph analytics



Dataset	# of nodes	# of edges
Patents	3,774,768	16,518,948
HepPh	34,546	421,578
LiveJournal	3,997,962	34,681,189