

# Special instructions: Predicated instructions, SIMD

- [SW04] P. Sanders and S. Winkel. *Super Scalar Sample Sort*.  
12th Annual European Symposium on Algorithms (ESA), LNCS 3221, 784-796, 2004.
- [SGL09] Benjamin Schlegel, Rainer Gemulla, Wolfgang Lehner. *k-ary search on modern processors*.  
5th International Workshop on Data Management on New Hardware (DaMoN), 52-60, 2009.
- [K+10] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A.D. Nguyen, T. Kaldewey, V.W. Lee, S.A. Brandt, and P. Dubey.  
*FAST: fast architecture sensitive tree search on modern CPUs and GPUs*.  
2010 ACM SIGMOD International Conference on Management of data, 339-350, 2010.

# Conditional instruction: cmovge

```
for (int i=0; i<100; i++)  
    if (X[i]>50) large+=7;
```

L6:

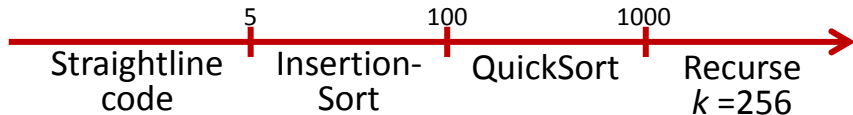
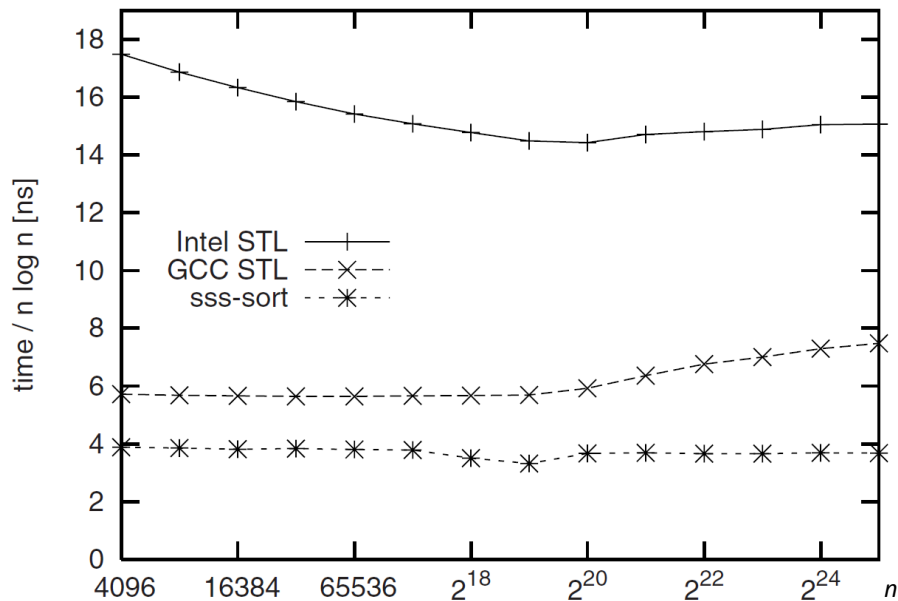
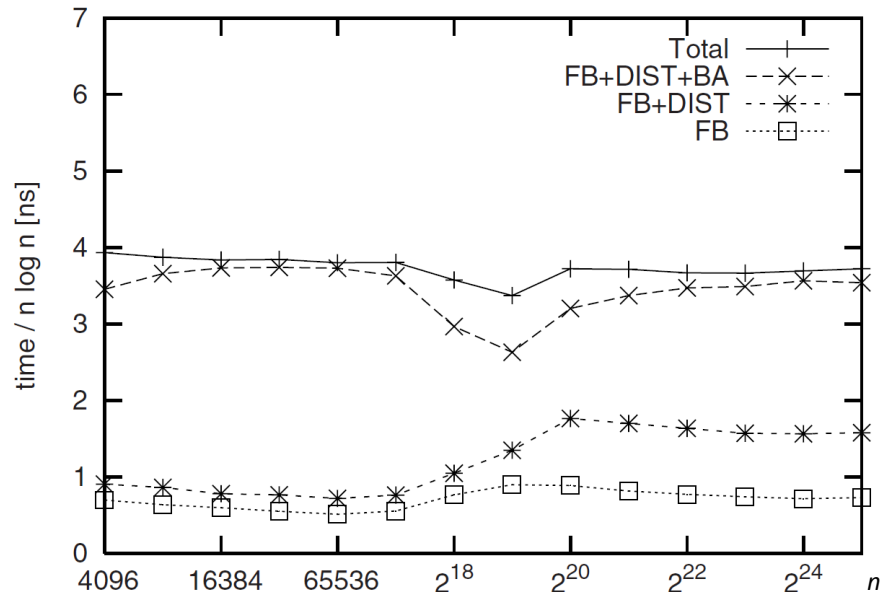
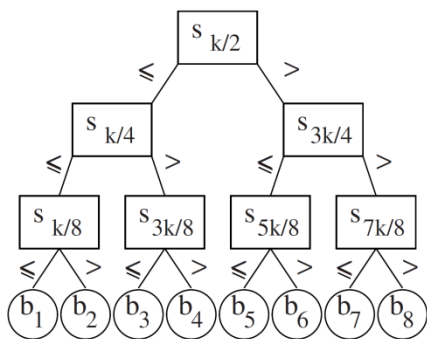
```
    cmpl  $51, (%edi,%eax,4)  
    leal  7(%edx), %ecx  
    cmovge %ecx, %edx  
    addl  $1, %eax  
    cmpl  $100, %eax  
    jne   L6
```

# Super Scalar Sample Sort

```

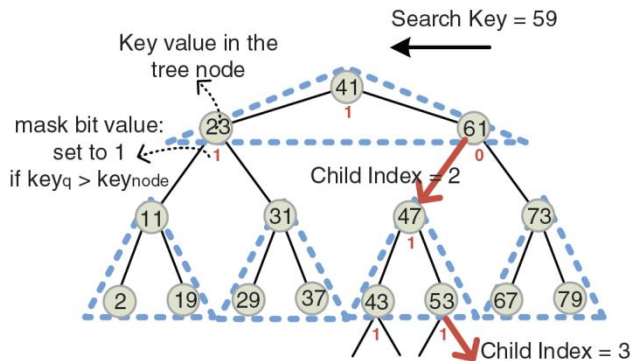
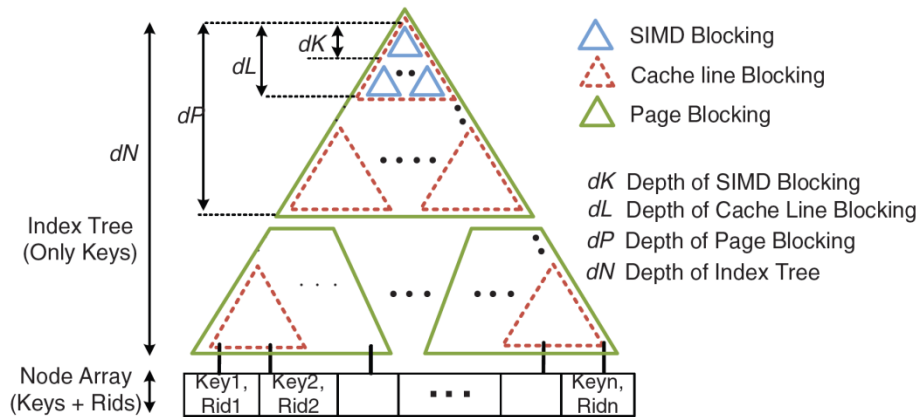
t := ⟨sk/2, sk/4, s3k/4, sk/8, s3k/8, s5k/8, s7k/8, ...⟩
for i := 1 to n do // locate each element
  j := 1 // current tree node := root
  repeat log k times // will be unrolled
    j := 2j + (ai > tj) // left or right?
  j := j - k + 1 // bucket index
  |bj|++ // count bucket size
  o(i) := j // remember oracle

for i := 1 to n do a'_{B[o(i)]} := a_i
  
```



# Multiway Comparisons

## SIMD (Single instruction, multiple data)



Use mask value as index

| Lookup Index | Child Index |
|--------------|-------------|
| 000          | 0           |
| 100          | N/A         |
| 010          | 1           |
| 110          | 2           |
| 001          | N/A         |
| 101          | N/A         |
| 011          | N/A         |
| 111          | 3           |

```

/*
TΔ: starting address of a tree
page_address: starting address offset of a particular page blocking sub-tree
page_offset: starting address offset of a particular cache line blocking sub-tree
cache_offset: starting address offset of a particular SIMD blocking sub-tree
*/

```

```

__m128i xmm_key_q = _mm_load1_ps(key_q);
/* xmm_key_q : vector register Vkeyq. Splat a search key (keyq) in Vkeyq */

for (i=0; i<number_of_accessed_pages_within_tree; i++) {
    page_offset = 0;
    page_address = Compute_page_address(child_offset);
    for (j=0; j<number_of_accessed_cachelines_within_page; j++) {
        /* Handle the first SIMD blocking sub-tree (=2 levels of the tree)*/

        __m128i xmm_tree = _mm_loadu_ps(TΔ + page_address + page_offset);
        /* xmm_tree: vector register Vtree. Load four tree nodes in Vtree*/

        __m128i xmm_mask = _mm_cmpgt_epi32(xmm_key_q, xmm_tree);
        /* xmm_mask: mask register Vmask. Set the mask register Vmask*/

        index = _mm_movemask_ps(_mm_castsi128_ps(xmm_mask));
        /* Convert mask register into index*/

        child_index = Lookup[index];

        /* Likewise, handle the second SIMD blocking sub-tree (=2 levels of the tree)*/
        xmm_tree = _mm_loadu_ps(TΔ + page_address + page_offset + Nk*child_index);
        xmm_mask = _mm_cmpgt_epi32(xmm_key_q, xmm_tree);
        index = _mm_movemask_ps(_mm_castsi128_ps(xmm_mask));
        cache_offset = child_index*4 + Lookup[index];
        page_offset = page_offset*16 + cache_offset;
    }
    child_offset = child_offset*(2^dp) + page_offset;
}

/* child_offset is the offset into the input (Key, Rid) tuple (T) */
While (T[child_offset].key <= key_q2)
    child_offset++

```