15 – Vectorized Execution

1. Background

   A. Vectorization
   converting algorithm's scalar implementation to vector implementation that processes one operation on multiple pairs of operands at once
   take advantage of parallelization with single thread

   B. SIMD
   single instruction multiple data
   CPU instruction that allow the processor to perform a operation on multiple data

      i. Tradeoffs

         1. Performance gains, resource utilization improves

         2. SIMD implementation is manual process.
         SIMD may have restriction on data alignments
         store/load instruction may be inefficient

   C. Vectorization method

      i. Automatic vectorization
      compiler detect loop, and make the basic operation to vectorized code
      difficult because, at compile time, we doesn't know that code can be vectorized.
      can be done with compiler hints.

      ii. Explicit vectorization
      use CPU intrinsics to manually use SIMD registers and SIMD instruction
      mostly not portable.

   D. Vectorization direction

      i. Horizontal
      perform operation on all elements together within single vector

      ii. Vertical
      perform operation in an elementwise manner on elements of each vector

2. Vectorized Algorithms

    A. Fundamental operations

        i. selective load, store
        using mask to specify load/store target
        load/store target data in/to memory location sequentially

        ii. selective gather, scatter
        using index vector to specify gather/scatter index
        load/store target data in/to memory location specified by index vector
        actually not executed in parallel because L1 cache doesn't allow multiple access at once

    B. Selection scans

        i. Since vectorized code cannot do branch, it should be implemented in branchless version

        ii. Using selective load and selective store

        iii. Improves performance a lot

    C. Hash tables probing

        i. Parallely hash elements of vectors

        ii. Using SIMD compare to match keys

        iii. If some of elements are matched, mask that elements.
        then see next index of unmatched element

        iv. Performance extremely decrease when hash table size gets bigger, because of cache size
        its performance will be almost same with scalar algorithm

    D. Partitioning/Histograms

        i. Use scatter and gathers to increments counts
        replicate the histogram to handle collisions