There and back again ...

From lists of irregular data structures to AoSoA and back

April 17, 2019 | Andreas Beckmann | Jülich Supercomputing Centre

Motivation

Simplified problem: Sequence of Sparse Matrix-Vector Multiplications

$$\vec{y}_i \quad += \quad \mathbf{A}_j \cdot \vec{x}_k$$

- Special sparse matrix structure
- Few different matrices, many input/result vectors

Motivation

Vectorization

- Single SpMV is hard to vectorize work-optimal
- Ideal: vectorize over one matrix and multiple vectors
- But: storage layout does not support this

Motivation

Storage layout

Input and result vectors cannot be stored effectively in array-of-Struct-of-Array (AoSoA) format.

- Computation of the input vectors, consumption of the result vectors cannot be vectorized in a similar fashion
- Each matrix is only multiplied to a subset of all input vectors
- SpMV involving different matrices have different (but partially overlapping) sets of input/result vectors
- Global AoSoA conversion does not help: only fits use for a single matrix – recompute for each matrix or replicate
- Single use of the converted data - no other steps could profit from such a layout

Temporary conversion to/from AoSoA

- Process batches of SIMD_WIDTH input vectors that get multiplied to the same matrix
- Temporarily convert them to AoSoA format
- Perform the SpMV, producing a temporary AoSoA result
- Split the result into individual vectors and accumulate to the result vectors in memory
- Continue with the next batch
- Temporary batchwise conversion will fit more easily in the cache than full conversion

Building block

Transpose

Matrix-Transpose within SIMD registers

- Let s be the SIMD width, e.g. for AVX512 $s = 16$ (SP) or $s = 8$ (DP)
- Input: s SIMD-registers containing a $s \times s$ matrix, either loaded from memory or as result from a preceding computation step
- Output: s SIMD-registers with the transposed matrix, either stored to memory or consumed in successive computation
- Implemented with a combination of shuffle intrinsics
- Same implementation for conversion to and from AoSoA

Building block

Load/Store

- Decouple storage layout from transpose implementation
- C++ adapter classes that present both a set of separate vectors or a consecutive AoSoA as a vector of $s \times s$ tiles
- Tiles present an array of s elements of the underlying SIMD type (e.g. __m512), supporting Load/Store

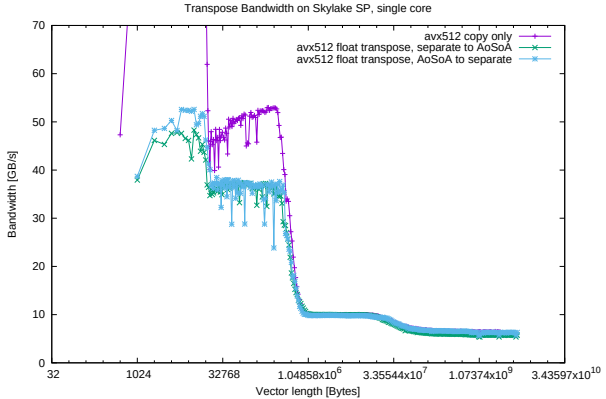
Caveats

- Conversion from/to separately stored vectors requires s general purpose registers, or another level of indirection.
- Not a big concern for e.g. $s = 4$, can use transpose on-the-fly without temporary storage
- Problematic for $s = 16$ and 16 GP registers, uses indirection and requires temporary storage

Performance

Comparing against copy without transpose

- For vector sizes fitting in L1, performance is comparable to L2 accesses
- For larger vectors, the shuffling hides well behind the memory latency



A C++ SIMD transpose library?

What we have today:

- C++ classes presenting $s \times s$ matrices while hiding the actual memory layout and providing SIMD load/store access
- Routines in intrinsics for AVX (128-bit, 256-bit), AVX512 (512-bit) doing the transpose operation

Is this helpful for problems you are facing?

What do you need?

- More platforms?
- Different interfaces?