Some thoughts on sparsity support for RISC-V matrix extensions

José Moreira
IBM Research

Warning

- These are very preliminary thoughts on how we could/should support sparse matrices on RISC-V matrix extensions (probably applicable to both IMF and AMF)
- These thoughts are inspired by a paper I recently read (Sparse Fine-tuning for Inference Acceleration of Large Language Models) and by the fact that modern GPUs (from AMD and NVIDIA, that I know of) have some limited form of sparsity support
- Much work remains before we can claim to have an approach or a roadmap for dealing with sparse matrices in RISC-V matrix extensions – but I want to put the problem out there so that the RISC-V Vector SIG members can collectively discuss and think about it

Sparse matrix-dense matrix multiplication (SpMM)

- We want to compute

$$\mathbf{R} \leftarrow \mathbf{R} + \mathbf{SD}$$

  using the algorithm

$$\mathbf{R} \leftarrow \mathbf{R} + \left(\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{D}_k \right)$$

- Where:
  - $\mathbf{R}$ is a $M \times N$ dense matrix
  - $\mathbf{S}$ is a $M \times K$ sparse matrix, and $\mathbf{S}^k$ is the k^{th} column of $\mathbf{S}$
  - $\mathbf{D}$ is a $K \times N$ dense matrix, and $\mathbf{D}_k$ is the k^{th} row of $\mathbf{D}$
- $\mathbf{S}$ has a sparsity (fraction of zeroes) of approximately 50-90%
- These conditions arise in modern DL models, where $\mathbf{S}$ is the weights matrix and $\mathbf{D}$ is the input
- Sparse matrices in scientific/engineering computing have much higher sparsity – not clear that solutions will be common to DL models, so let us focus on DL first

A more concrete example

- Let VLEN = 128 bits, 4×32-bit elements
- Let the architecture define a set of accumulators of size $4 \times 4 \times 32$-bit elements – for now, it does not matter if those are new registers or if they reuse vector register state
- The basic operation of the hardware is

$$A \leftarrow A + uv^{\mathrm{T}}$$

where A is an accumulator, u and v are vector registers

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \leftarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} \begin{bmatrix} v_0 & v_1 & v_2 & v_3 \end{bmatrix}$$

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \leftarrow \begin{bmatrix} a_{00} + u_0 v_0 & a_{01} + u_0 v_1 & a_{02} + u_0 v_2 & a_{03} + u_0 v_3 \\ a_{10} + u_1 v_0 & a_{11} + u_1 v_1 & a_{12} + u_1 v_2 & a_{13} + u_1 v_3 \\ a_{20} + u_2 v_0 & a_{21} + u_2 v_1 & a_{22} + u_2 v_2 & a_{23} + u_2 v_3 \\ a_{30} + u_3 v_0 & a_{31} + u_3 v_1 & a_{32} + u_3 v_2 & a_{33} + u_3 v_3 \end{bmatrix}$$

Going sparse

- Now, let there be an 8-element column of the sparse matrix, out of which at most 4 elements are nonzero – these can be represented by a 4-element vector u and an 8-bit mask, such as $[1,0,0,1,0,0,1,1]$
- In that case, we want to compute

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \\ a_{40} & a_{41} & a_{42} & a_{43} \\ a_{50} & a_{51} & a_{52} & a_{53} \\ a_{60} & a_{61} & a_{62} & a_{63} \\ a_{70} & a_{71} & a_{72} & a_{73} \end{bmatrix} \leftarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \\ a_{40} & a_{41} & a_{42} & a_{43} \\ a_{50} & a_{51} & a_{52} & a_{53} \\ a_{60} & a_{61} & a_{62} & a_{63} \\ a_{70} & a_{71} & a_{72} & a_{73} \end{bmatrix} + \begin{bmatrix} u_0 \\ 0 \\ 0 \\ u_1 \\ 0 \\ 0 \\ u_2 \\ u_3 \end{bmatrix} \begin{bmatrix} v_0 & v_1 & v_2 & v_3 \end{bmatrix}$$

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{30} & a_{31} & a_{32} & a_{33} \\ a_{60} & a_{61} & a_{62} & a_{63} \\ a_{70} & a_{71} & a_{72} & a_{73} \end{bmatrix} \leftarrow \begin{bmatrix} a_{00} + u_0 v_0 & a_{01} + u_0 v_1 & a_{02} + u_0 v_2 & a_{03} + u_0 v_3 \\ a_{30} + u_1 v_0 & a_{31} + u_1 v_1 & a_{32} + u_1 v_2 & a_{33} + u_1 v_3 \\ a_{60} + u_2 v_0 & a_{61} + u_2 v_1 & a_{62} + u_2 v_2 & a_{63} + u_2 v_3 \\ a_{70} + u_3 v_0 & a_{71} + u_3 v_1 & a_{72} + u_3 v_2 & a_{73} + u_3 v_3 \end{bmatrix}$$

Some observations

- In both the dense and sparse cases, the outer product uv^{T} is the same
- In both the dense and sparse cases, one accumulator-worth of data (4 rows) are updated
- The only difference between the two cases are the origin and destination of the updated data
  - In the dense case, all 4 rows come from one accumulator register
  - In the sparse case, 2 rows come from one accumulator register and 2 rows from another register
- Maybe we need different types of instructions
  - At most 4 nonzero in 4 elements (0% sparse – update 1 accumulator)
  - At most 4 nonzero in 8 elements (50% sparse – update 2 accumulators)
  - At most 4 nonzero in 16 elements (75% sparse – update 4 accumulators)
  - At most 4 nonzero in 32 elements (88% sparse – update 8 accumulators)