A RISC-V VPU for Very Long and Sparse Vectors

Gopinath Mahale, Karim Charfi, Tejas Limbasiya, Teresa Cervero, John Davis
{gopinath.mahale, karim.charfi, tejas.limbasiya, teresa.cervero, john.davis}@bsc.es
Barcelona Supercomputing Center

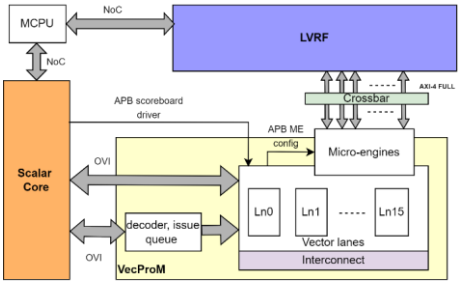

Figure 1: VecProM interfaced with Scalar core and LVRF

Operations on Vector Processing Unit (VPU) have advantages of reduced instruction memory, reduced address translations and scope for deeper pipelines leading to overall improvement in compute performance. These advantages are hard to be extended to very long vectors since the supported vector lengths on VPUs are limited by register space provided by Vector Register Files (VRF), which is difficult to be scaled indefinitely due to energy constraints. Another challenge in VPUs has been to achieve energy-efficiency in sparse-vector computations, which generally are not cache-friendly.

The ACME accelerator core designed at MEEP [1] project consists of a RISC-V VPU, termed *VecProM*, and two systolic arrays as co-processors connected to a RISC-V scalar core. VecProM is a modified version of a baseline RISC-V VPU [2], with the enhancements aligned towards support for very long vectors and sparse vectors. Highlight of VecProM is a specialized high-bandwidth memory path that bypasses the cache hierarchy and connects VPU to a scratchpad memory, termed Long Vector Register File (LVRF). ACME disaggregates memory and arithmetic instruction execution, thereby improving performance and energy benefits for selected vector types.

VecProM can operate in two operand-dependent modes: Mode_sd, suitable for short dense vectors, i.e., when the dense operand vector can fit on the register file, and Mode_ldsp, suitable for very long dense vectors and sparse vector operands. In Mode_sd, the scalar core issues the vector instructions over an OVI interface to the VecProM. The loads and stores are performed over a dedicated OVI interface, and the instructions maintain the same memory and data paths used in the baseline VPU. In Mode_ldsp, hardware strip-mining is implemented on the vector operands stored in LVRF. LVRF acts as a virtual VRF for the physical VRF on VecProM, and maintains 32 vector registers as per the RISC-V ISA specifications. The management of loads and stores on LVRF from VecProM is managed by DMA engines termed Micro-engines (MEs). MEs are configured on-the-fly with operand vector length and source addresses of the issued instruction. In Mode_ldsp, the scalar core offloads the load and store instructions to a RISC-V processor, termed *MemoryCPU* (MCPU). MCPU is connected to the ACME accelerator over an NoC [1], and it is responsible for loads and stores to the LVRF. MCPU notifies completion of load operation to the scalar core over the NoC, which in turn notifies the VecProM VRF scoreboard over an APB interface. The arithmetic instruction in the issue queue of VecProM executes once the scoreboard signals availability of operands in the LVRF. This also triggers the MEs to start loading and storing, overlapped with computations, the corresponding operands and result vector elements. In this process, multiple MEs read and write data from banks of LVRF through a Cross bar over an AXI-4 full interface. In case of sparse vectors operations, load operations in MCPU gathers non-zero elements from memory, and the resultant dense vector is loaded into the LVRF. This reduces the traffic over the NoC, gives performance benefits in processing, and also may give energy benefits by bypassing cache hierarchy, not very suitable for sparse data.

The enhancements in VecProM are introduced with minimal changes to the existing data path of baseline, thus retaining functionality and benefits of the baseline. Also, the modes of operation, mainly differing in memory path, provide flexibility to switch between the modes based on potential gains for given operand vectors.

References:

[1] Fell, A., Mazure, D. J., Garcia, T. C., Perez, B., Teruel, X., Wilson, P., & Davis, J. D. (2021). The MareNostrum Experimental Exascale Platform (MEEP). *Supercomputing Frontiers and Innovations*, *8*(1), 62–81. https://doi.org/10.14529/jsfi210105

[2] https://www.bsc.es/research-and-development/projects/epi-european-processor-initative-epi