MEDEA: Improved Memory-Level Parallelism in a decoupled execute/access vector accelerator

Umair Riaz*, Luis A. Plana, Peter Wilson and John D. Davis

Barcelona Supercomputing Center

Abstract

The performance of Machine learning and graph-based applications is hampered by the inefficient use of memory bandwidth caused by their sparse access patterns. In order to improve bandwidth utilization in a RISC-V based vector accelerator, the Memory Engine for Decoupled Execute/Access (MEDEA) combines a hardware engine that handles vector loads and stores efficiently with a dedicated core that supports memory-intensive operations such as spinlocks and memcpy. Although work on MEDEA is still in progress, simulations show that it can have a large impact on the performance of sparse-memory applications such as SpMV.

Introduction

Efficient use of memory bandwidth is well documented as a key factor in High Performance Computing (HPC) [1]. It is critical in applications with sparse memory accesses, given that HPC systems are optimized for access patterns that exhibit spatial and temporal locality. The Accelerated Compute and Memory Engine (ACME) [2] is a RISC-V based high-performance architecture targeting both dense and sparse workloads. ACME is a chiplet-based, 2D tiled design targeting Exascale-class systems.

ACME increases memory-level parallelism (MLP) by shifting memory-accessing responsibilities from compute tiles to specialized *memory* tiles, connected as shown in Figure 1. These Memory Engine for Decoupled Execute/Access (MEDEA) tiles improve within- and across-thread concurrency through local buffering and efficient vector data processing.

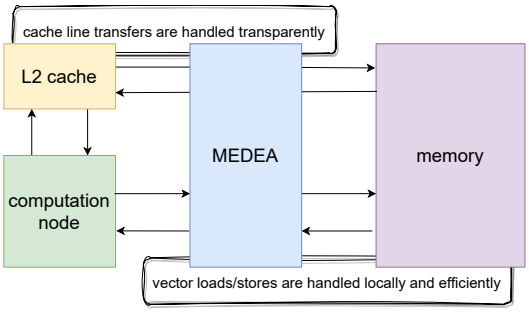

Figure 1: *MEDEA: decoupled memory-access engines.*

An ACME chiplet contains a 16 row × 4 column mesh of compute tiles, one MEDEA tile in each row, 2 High-Bandwidth Memories (HBM) and 2 Non-Volatile RAM (NVRAM) devices. The compute and memory tiles are connected through a NoC and there is a crossbar interconnecting all the MEDEA tiles and memories. The interfaces to HBM and NVRAM in each MEDEA are seen only by its memory controller and are governed by published standards.

Each ACME compute tile contains a number of compute nodes, each comprising a scalar core and its associated Vector Processing Unit (VPU), which can operate concurrently. The ACME VPU capabilities are extended to support vector operations with sparse and very long vector operands. VPUs are well-known accelerators due to their capability to exploit data-level parallelism (DLP) through single-instruction multiple-data operation. By decoupling the access and execution engines, ACME adds capabilities near the memory controller to increase MLP. Figure 1 shows a memory path through MEDEA that bypasses cache memories. This path is used by the VPU to complete vector loads and stores. MEDEA transfers vector data between memory and the VPU Long Vector Register File (LVRF). Indexed vector loads and stores, which are the major latency-inducing steps in sparse operations, are handled by MEDEA close to memory. Vector elements are packed as a dense vector representation and then transferred to the LVRF, saving power by bypassing the cache hierarchy but also reducing NoC traffic. The MEDEA path to memory is most effective in streaming, non-unit stride and sparse accesses.

MEDEA

A MEDEA tile services requests from compute tiles. It handles all requests from a given compute node in strict order. There are primarily four requests: L2 cache misses, TLB misses, vector loads/stores and memory-intensive operations (such as memcpy or spinlocks).

*Corresponding author: umair.riaz@bsc.es

The MEEP project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 946002. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Croatia, Turkey

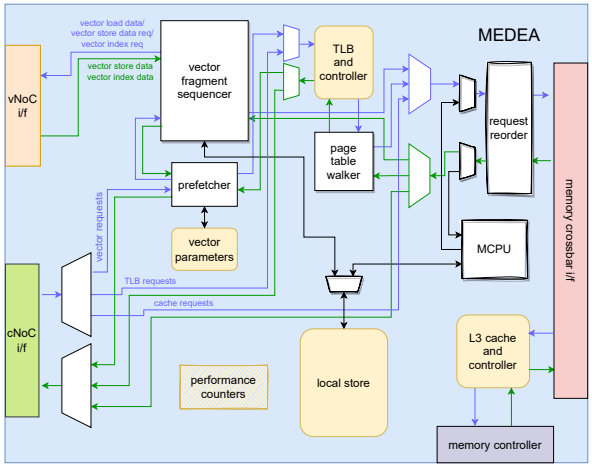

Figure 2: *MEDEA microarchitecture and interfaces.*

Figure 2 shows the MEDEA microarchitecture, which can maintain multiple in-flight requests. It is organized as three parallel elastic pipelines that handle TLB, cache and vector requests. The pipelines merge together to drive the memory interface.

Cache requests are essentially transferred transparently to memory and the result sent back to the requesting compute node. TLB requests cause a TLB lookup to translate the virtual address to a physical address. On a miss a page table walk is executed and the TLB updated. Vector request, on the other hand, are where most of the interesting action takes place.

MEDEA supports long vector operations. For this reason, the Vector Fragment Sequencer (VFS) breaks vectors into fragments, according to memory crossbar data width and burst length. Each fragment generates a separate memory transaction. The data of the different transactions is put together in the LVRF.

RISC-V vector operations support three different addressing modes: unit stride, strided and indexed. Unit stride requests are managed as dense memory accesses. On the other hand, strided and indexed requests are sparse accesses. In these cases, MEDEA fragments accesses based on the stride or the index, which may result in fragments that contain a single vector element. These elements are collected and packed locally, and trasferred to the LVRF as a dense vector with the corresponding energy and NoC traffic savings.

The prefetcher module can non-speculatively prefetch vector data based on the difference between the application vector length (AVL). requested by the application, and the Granted Vector Length (GVL), which is constrained by the physical limits of the LVRF. Once an initial vector request has been made, the tile can prefetch the next GVL worth of vector elements knowing that the request will eventually arrive.

Memory operation requests are handled by the Memory CPU (MCPU), a scalar processor with a tightly-coupled memory and a low-latency interface to the memory controller. The MCPU provides a collection of *library* memory-intense functions that can be accessed remotely by the compute tiles. The functions are executed locally, close to memory, with the corresponding performance, energy and NoC traffic improvements.

Discussion

Different hardware-based approaches, *e.g.*, [3], have been used to improve memory bandwidth utilization for sparse workloads. In some cases a HW/SW codesign approach, *e.g.*, [4], presents the user/compiler with an API to manage the associated hardware. MEDEA, on the other hand, contains a hardware engine that handles vector memory accesses efficiently and a dedicated core that supports memory-intensive operations.

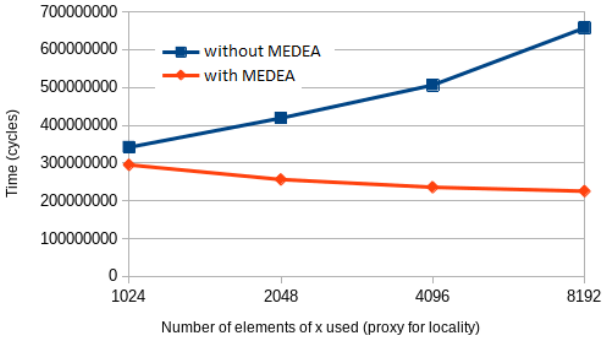

Figure 3: *SpMV execution times.*

Sparse Matrix Vector (SpMV) is a benchmark commonly used to evaluate performance for sparse memory access patterns. Figure 3 shows SpMV execution times, in clock cycles, of two different simulated systems, one with MEDEA (WM) and one without it (NM). The graph shows that, as the number of different vector elements used grows, NM simulated times grow linearly while WM times remain essentially constant due to the increased MLP provided by MEDEA.

References

[1] Ivy Peng, Roger Pearce, and Maya Gokhale. "On the Memory Underutilization: Exploring Disaggregated Memory on HPC Systems". In: *Intern. Symp. on Computer Arch. and High Performance Computing*. 2020, pp. 183–190.

[2] Alexander Fell et al. "The MareNostrum Experimental Exascale Platform (MEEP)". In: *Supercomputing Frontiers and Innovations* 8.1 (Apr. 2021), pp. 62–81.

[3] Michael Sung, Ronny Krashinsky, and Krste Asanović. "Multithreading Decoupled Architectures for Complexity-Effective General Purpose Computing". In: *SIGARCH Comput. Archit. News* 29.5 (2001), pp. 56–61.

[4] Marcelo Orenes-Vera et al. "Tiny but Mighty: Designing and Realizing Scalable Latency Tolerance for Manycore SoCs". In: *Proc. International Symposium on Computer Architecture*. 2022, pp. 817–830.