NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling

Gagandeep Singh[a,b,c] Dionysios Diamantopoulos[c] Christoph Hagleitner[c] Juan Gómez-Luna[b]
Sander Stuijk[a] Onur Mutlu[b] Henk Corporaal[a]
[a]Eindhoven University of Technology [b]ETH Zürich [c]IBM Research Europe, Zurich

Ongoing climate change calls for fast and accurate weather and climate modeling. However, when solving large-scale weather prediction simulations, state-of-the-art CPU and GPU implementations suffer from limited performance and high energy consumption. These implementations are dominated by complex irregular memory access patterns and low arithmetic intensity that pose fundamental challenges to acceleration. To overcome these challenges, we propose and evaluate the use of near-memory acceleration using a reconfigurable fabric with high-bandwidth memory (HBM). We focus on compound stencils that are fundamental kernels in weather prediction models. By using high-level synthesis techniques, we develop NERO, an FPGA+HBM-based accelerator connected through IBM CAPI2 (Coherent Accelerator Processor Interface) to an IBM POWER9 host system. Our experimental results show that NERO outperforms a 16-core POWER9 system by 4.2× *and* 8.3× *when running two different compound stencil kernels. NERO reduces the energy consumption by* 22× *and* 29× *for the same two kernels over the POWER9 system with an energy efficiency of 1.5 GFLOPS/Watt and 17.3 GFLOPS/Watt. We conclude that employing near-memory acceleration solutions for weather prediction modeling is promising as a means to achieve both high performance and high energy efficiency.*

1. Introduction

Accurate weather prediction using detailed weather models is essential to make weather-dependent decisions in a timely manner. The Consortium for Small-Scale Modeling (COSMO) [38] built one such weather model to meet the high-resolution forecasting requirements of weather services. The COSMO model is a non-hydrostatic atmospheric prediction model currently being used by a dozen nations for meteorological purposes and research applications.

The central part of the COSMO model (called *dynamical core* or *dycore*) solves the Euler equations on a curvilinear grid and applies implicit discretization (i.e., parameters are dependent on each other at the same time instance [26]) in the vertical dimension and explicit discretization (i.e., a solution is dependent on the previous system state [26]) in the horizontal dimension. The use of different discretizations leads to three computational patterns [96]: 1) horizontal stencils, 2) tridiagonal solvers in the vertical dimension, and 3) point-wise computation. These computational kernels are compound stencil kernels that operate on a three-dimensional grid [48]. *Vertical advection* (`vadvc`) and *horizontal diffusion* (`hdiff`) are such compound kernels found in the *dycore* of the COSMO weather prediction model. These kernels are representative of the data access patterns and algorithmic complexity of the entire COSMO model. They are similar to the kernels used in other weather and climate models [60, 79, 107]. Their performance is dominated by memory-bound operations with unique irregular memory access patterns and low arithmetic intensity that often results in <10% sustained floating-point performance on current CPU-based systems [99].

Figure 1 shows the roofline plot [104] for an IBM 16-core POWER9 CPU (IC922).[1] After optimizing the `vadvc` and `hdiff` kernels for the POWER architecture by following the approach in [105], they achieve 29.1 GFLOP/s and 58.5 GFLOP/s, respectively, for 64 threads. Our roofline analysis indicates that these kernels are constrained by the host DRAM bandwidth. Their low arithmetic intensity limits their performance, which is one order of magnitude smaller than the peak performance, and results in a fundamental memory bottleneck that standard CPU-based optimization techniques cannot overcome.

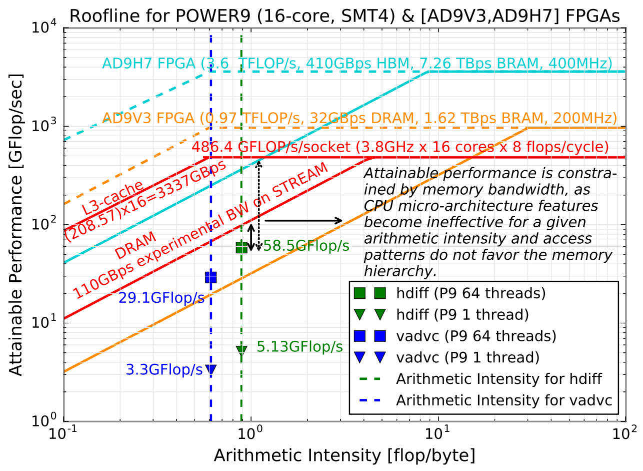


Figure 1: Roofline [104] for POWER9 (1-socket) showing vertical advection (`vadvc`) and horizontal diffusion (`hdiff`) kernels for single-thread and 64-thread implementations. The plot shows also the rooflines of the FPGAs used in our work.

In this work, our goal is to overcome the memory bottleneck of weather prediction kernels by exploiting near-memory computation capability on FPGA accelerators with high-bandwidth memory (HBM) [5, 65, 66] that are attached

[1]IBM and POWER9 are registered trademarks or common law marks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies.

978-1-7281-9902-3/20/$31.00 ©2020 IEEE
DOI 10.1109/FPL50879.2020.00014

to the host CPU. Figure 1 shows the roofline models of the two FPGA cards (AD9V3 [2] and AD9H7 [1]) used in this work. FPGAs can handle irregular memory access patterns efficiently and offer significantly higher memory bandwidth than the host CPU with their on-chip URAMs (UltraRAM), BRAMs (block RAM), and off-chip HBM (high-bandwidth memory for the AD9H7 card). However, taking full advantage of FPGAs for accelerating a workload is not a trivial task. To compensate for the higher clock frequency of the baseline CPUs, our FPGAs must exploit at least one order of magnitude more parallelism in a target workload. This is challenging, as it requires sufficient FPGA programming skills to map the workload and optimize the design for the FPGA microarchitecture.

Modern FPGA boards deploy new cache-coherent interconnects, such as IBM CAPI [93], Cache Coherent Interconnect for Accelerators (CCIX) [24], and Compute Express Link (CXL) [88], which allow tight integration of FPGAs with CPUs at high bidirectional bandwidth (on the order of tens of GB/s). However, memory-bound applications on FPGAs are limited by the relatively low DDR4 bandwidth (72 GB/s for four independent dual-rank DIMM interfaces [11]). To overcome this limitation, FPGA vendors have started offering devices equipped with HBM [6, 7, 12, 66] with a theoretical peak bandwidth of 410 GB/s. HBM-equipped FPGAs have the potential to reduce the memory bandwidth bottleneck, but a study of their advantages for real-world memory-bound applications is still missing.

We aim to answer the following research question: **Can FPGA-based accelerators with HBM mitigate the performance bottleneck of memory-bound compound weather prediction kernels in an energy efficient way?** As an answer to this question, we present NERO, a near-HBM accelerator for weather prediction. We design and implement NERO on an FPGA with HBM to optimize two kernels (vertical advection and horizontal diffusion), which notably represent the spectrum of computational diversity found in the COSMO weather prediction application. We co-design a hardware-software framework and provide an optimized API to interface efficiently with the rest of the COSMO model, which runs on the CPU. Our FPGA-based solution for `hdiff` and `vadvc` leads to performance improvements of 4.2× and 8.3× and energy reductions of 22× and 29×, respectively, with respect to optimized CPU implementations [105].

The major contributions of this paper are as follows:

- We perform a detailed roofline analysis to show that representative weather prediction kernels are constrained by memory bandwidth on state-of-the-art CPU systems.
- We propose NERO, the first near-HBM FPGA-based accelerator for representative kernels from a real-world weather prediction application.
- We optimize NERO with a data-centric caching scheme with precision-optimized tiling for a heterogeneous memory hierarchy (consisting of URAM, BRAM, and HBM).
- We evaluate the performance and energy consumption of our accelerator and perform a scalability analysis. We show that an FPGA+HBM-based design outperforms a complete 16-core POWER9 system (running 64 threads) by 4.2× for the vertical advection (`vadvc`) and 8.3× for the horizontal diffusion (`hdiff`) kernels with energy reductions of 22× and 29×, respectively. Our design provides an energy efficiency of 1.5 GLOPS/Watt and 17.3 GFLOPS/Watt for `vadvc` and `hdiff` kernels, respectively.

2. Background

In this section, we first provide an overview of the `vadvc` and `hdiff` compound stencils, which represent a large fraction of the overall computational load of the COSMO weather prediction model. Second, we introduce the CAPI SNAP (Storage, Network, and Analytics Programming) framework[2] that we use to connect our NERO accelerator to an IBM POWER9 system.

2.1. Representative COSMO Stencils

A stencil operation updates values in a structured multidimensional grid based on the values of a fixed local neighborhood of grid points. Vertical advection (`vadvc`) and horizontal diffusion (`hdiff`) from the COSMO model are two such compound stencil kernels, which represent the typical code patterns found in the *dycore* of COSMO. Algorithm 1 shows the pseudo-code for `vadvc` and `hdiff` kernels. The horizontal diffusion kernel iterates over a 3D grid performing *Laplacian* and *flux* to calculate different grid points. Vertical advection has a higher degree of complexity since it uses the Thomas algorithm [97] to solve a tri-diagonal matrix of the velocity field along the vertical axis. Unlike the conventional stencil kernels, vertical advection has dependencies in the vertical direction, which leads to limited available parallelism.

Such compound kernels are dominated by memory-bound operations with complex memory access patterns and low arithmetic intensity. This poses a fundamental challenge to acceleration. CPU implementations of these kernels [105] suffer from limited data locality and inefficient memory usage, as our roofline analysis in Figure 1 exposes.

2.2. CAPI SNAP Framework

The OpenPOWER Foundation Accelerator Workgroup [8] created the CAPI SNAP framework, an open-source environment for FPGA programming productivity. CAPI SNAP provides two key benefits [103]: (i) it enables an improved developer productivity for FPGA acceleration and eases the use of CAPI's cache-coherence mechanism, and (ii) it places FPGA-accelerated compute engines, also known as FPGA *actions*, closer to relevant data to achieve better performance. SNAP provides a simple API to invoke an accelerated *action*, and also provides programming methods to instantiate customized accelerated *actions* on the FPGA side. These accelerated *actions* can be specified in C/C++ code that is then compiled to the FPGA target using the Xilinx Vivado High-Level Synthesis (HLS) tool [10].

[2]https://github.com/open-power/snap

Algorithm 1: Pseudo-code for vertical advection and horizontal diffusion kernels used by the COSMO [38] weather prediction model.

```
Function verticalAdvection(float* ccol, float* dcol, float* wcon, float* ustage, float* upos, float* utens, float* utensstage)
    for $c \leftarrow 2$ to $column - 2$ do
        for $r \leftarrow 2$ to $row$-2 do
            Function forwardSweep(float* ccol, float* dcol, float* wcon, float* ustage, float* upos, float* utens, float* utensstage)
                for $d \leftarrow 1$ to $depth$ do
                    /* forward sweep calculation */
                end
            end
            Function backwardSweep(float* ccol, float* dcol, float* wcon, float* ustage, float* upos, float* utens, float* utensstage)
                for $d \leftarrow depth - 1$ to 1 do
                    /* backward sweep calculation */
                end
            end
        end
    end
end
Function horizontalDiffusion(float* src, float* dst)
    for $d \leftarrow 1$ to $depth$ do
        for $c \leftarrow 2$ to $column - 2$ do
            for $r \leftarrow 2$ to $row$-2 do
                /* Laplacian calculation */
                $lap_{CR} = laplaceCalculate(c, r)$ /* row-laplacian */
                $lap_{CRm} = laplaceCalculate(c, r - 1)$
                $lap_{CRp} = laplaceCalculate(c, r + 1)$
                    /* column-laplacian */
                $lap_{CmR} = laplaceCalculate(c - 1, r)$
                $lap_{CpR} = laplaceCalculate(c + 1, r)$ /* column-flux
                    calculation */
                $flux_{C} = lap_{CpR} - lap_{CR}$
                $flux_{Cm} = lap_{CR} - lap_{CmR}$
                /* row-flux calculation */
                $flux_{R} = lap_{CRp} - lap_{CR}$
                $flux_{Rm} = lap_{CR} - lap_{CmR}$
                /* output calculation */
                $dest[d][c][r] = src[d][c][r] -$
                $c1 * (flux_{CR} - flux_{CmR}) + (flux_{CR} - flux_{CRm})$
            end
        end
    end
end
```

3. Design Methodology

3.1. NERO, A Near HBM Weather Prediction Accelerator

The low arithmetic intensity of real-world weather prediction kernels limits the attainable performance on current multi-core systems. This sub-optimal performance is due to the kernels' complex memory access patterns and their inefficiency in exploiting a rigid cache hierarchy, as quantified in the roofline plot in Figure 1. These kernels cannot fully utilize the available memory bandwidth, which leads to high data movement overheads in terms of latency and energy consumption. We address these inefficiencies by developing an architecture that combines fewer off-chip data accesses with higher throughput for the loaded data. To this end, our accelerator design takes a data-centric approach [13, 14, 27, 43, 52, 53, 63, 75, 89, 91] that exploits near high-bandwidth memory acceleration.

Figure 2 shows a high-level overview of our integrated system. An HBM-based FPGA is connected to a server system based on an IBM POWER9 processor using the Coherent Accelerator Processor Interface version 2 (CAPI2). The FPGA consists of two HBM stacks[3], each with 16 *pseudo-memory channels* [3]. A channel is exposed to the FPGA as a 256-bit wide port, and in total, the FPGA has 32 such ports. The HBM IP provides 8 memory controllers (per stack) to handle the data transfer to and from the HBM memory ports. Our design consists of an *accelerator functional unit* (AFU) that interacts with the host system through the power service layer (PSL), which is the CAPI endpoint on the FPGA. An AFU comprises of multiple *processing elements* (PEs) that perform compound stencil computation. Figure 3a shows the architecture overview of NERO. As vertical advection is the most complex kernel, we depict our architecture design flow for vertical advection. We use a similar design for the horizontal diffusion kernel.

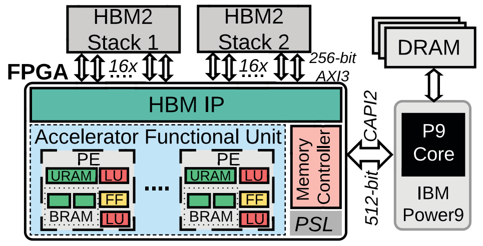
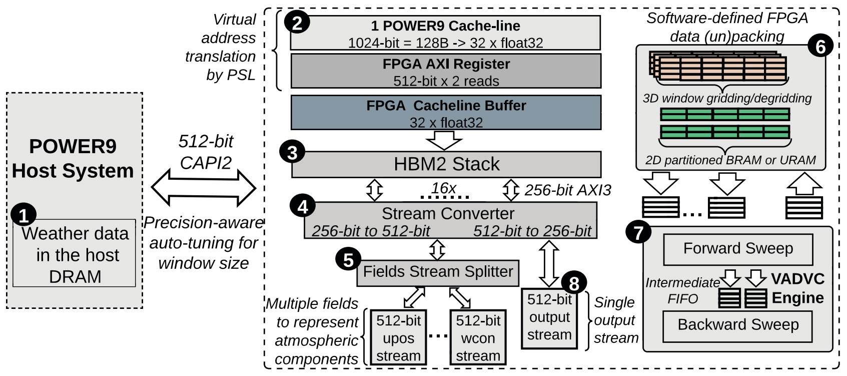


Figure 2: Heterogeneous platform with an IBM POWER9 system connected to an HBM-based FPGA board via CAPI2.

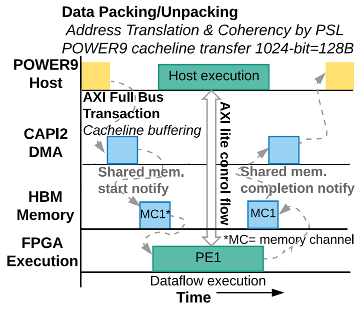
The weather data, based on the atmospheric model resolution grid, is stored in the DRAM of the host system (➊ in Figure 3a). We employ the double buffering technique between the CPU and the FPGA to hide the PCIe (Peripheral Component Interconnect Express [72]) transfer latency. By configuring a buffer of 64 cache lines, between the AXI4 interface of CAPI2/PSL and the AFU, we can reach the theoretical peak bandwidth of CAPI2/PCIe (i.e., 16 GB/s). We create a specialized memory hierarchy from the heterogeneous FPGA memories (i.e., URAM, BRAM, and HBM). By using a greedy algorithm, we determine the best-suited hierarchy for our kernel. The memory controller (shown in Figure 2) handles the data placement to the appropriate memory type based on programmer's directives.

On the FPGA, following the initial buffering (➋), the transferred grid data is mapped onto the HBM memory (➌). As the FPGA has limited resources we propose a 3D window-based grid transfer from the host DRAM to the FPGA, facilitating a smaller, less power-hungry deployment. The window size represents the portion of the grid a processing element (PE in Figure 2) would process. Most FPGA developers manually optimize for the right window size. However, manual optimization is tedious because of the huge design space, and it requires expert guidance. Further, selecting an inappropriate window size leads to sub-optimal results. Our experiments (in Section 4.2) show that: (1) finding the best window size is critical in terms of the area vs. performance trade-off, and (2) the

[3]In this work, we enable only a single stack based on our resource and power consumption analysis for the vadvc kernel.



(a)



(b)

Figure 3: (a) Architecture overview of NERO with data flow sequence from the host DRAM to the on-board FPGA memory via POWER9 cachelines. We depict a single processing element (PE) fetching data from a dedicated HBM port. The number of HBM ports scales linearly with the number of PEs. Heterogeneous partitioning of on-chip memory blocks reduces read and write latencies across the FPGA memory hierarchy. (b) Execution timeline with data flow sequence from the host DRAM to the onboard FPGA memory.

best window size depends on the datatype precision. Hence, instead of pruning the design space manually, we formulate the search for the best window size as a multi-objective auto-tuning problem taking into account the datatype precision. We make use of OpenTuner [20], which uses machine-learning techniques to guide the design-space exploration.

Our design consists of multiple PEs (shown in Figure 2) that exploit data-level parallelism in COSMO weather prediction kernels. A dedicated HBM memory port is assigned to a specific PE; therefore, we enable as many HBM ports as the number of PEs. This allows us to use the high HBM bandwidth effectively because each PE fetches from an independent port. In our design, we use a switch, which provides the capability to bypass the HBM, when the grid size is small, and map the data directly onto the FPGA's URAM and BRAM. The HBM port provides 256-bit data, which is half the size of the CAPI2 bitwidth (512-bit). Therefore, to match the CAPI2 bandwidth, we introduce a stream converter logic (❹) that converts a 256-bit HBM stream to a 512-bit stream (CAPI compatible) or vice versa. From HBM, a PE reads a single stream of data that consists of all the fields[4] that are needed for a specific COSMO kernel computation. The PEs use a fields stream splitter logic (❺) that splits a single HBM stream to multiple streams (512-bit each), one for each field.

To optimize a PE, we apply various optimization strategies. First, we exploit the inherent parallelism in a given algorithm through hardware pipelining. Second, we partition on-chip memory to avoid the stalling of our pipelined design, since the on-chip BRAM/URAM has only two read/write ports. Third, all the tasks execute in a dataflow manner that enables task-level parallelism. `vadvc` is more computationally complex than `hdiff` because it involves forward and backward sweeps with dependencies in the z-dimension. While `hdiff` performs only Laplacian and flux calculations with dependencies in the x- and y-dimensions. Therefore, we demonstrate our design flow by means of the `vadvc` kernel (Figure 3a). Note that we show only a single port-based PE operation. However, for multiple PEs, we enable multiple HBM ports.

We make use of memory reshaping techniques to configure our memory space with multiple parallel BRAMs or URAMs [37]. We form an intermediate memory hierarchy by decomposing (or slicing) 3D window data into a 2D grid. This allows us to bridge the latency gap between the HBM memory and our accelerator. Moreover, it allows us to exploit the available FPGA resources efficiently. Unlike traditionally-fixed CPU memory hierarchies, which perform poorly with irregular access patterns and suffer from cache pollution effects, application-specific memory hierarchies are shown to improve energy and latency by tailoring the cache levels and cache sizes to an application's memory access patterns [98].

The main computation pipeline (❼) consists of a forward and a backward sweep logic. The forward sweep results are stored in an intermediate buffer to allow for backward sweep calculation. Upon completion of the backward sweep, results are placed in an output buffer that is followed by a degridding logic (❻). The degridding logic converts the calculated results to a 512-bit wide output stream (❽). As there is only a single output stream (both in `vadvc` and `hdiff`), we do not need extra logic to merge the streams. The 512-bit wide stream goes through an HBM stream converter logic (❹) that converts the stream bitwidth to HBM port size (256-bit).

Figure 3b shows the execution timeline from our host system to the FPGA board for a single PE. The host offloads the processing to an FPGA and transfers the required data via DMA (direct memory access) over the CAPI2 interface. The SNAP framework allows for parallel execution of the host and our FPGA PEs while exchanging control signals over the AXI lite interface [4]. On task completion, the AFU notifies the host system via the AXI lite interface and transfers back the results via DMA.

[4]Fields represent atmospheric components like wind, pressure, velocity, etc. that are required for weather calculation.

3.2. NERO Application Framework

Figure 4 shows the NERO application framework to support our architecture. A software-defined COSMO API (①) handles offloading jobs to NERO with an interrupt-based queuing mechanism. This allows for minimal CPU usage (and, hence, power usage) during FPGA operation. NERO employs an array of processing elements to compute COSMO kernels, such as vertical advection or horizontal diffusion. Additionally, we pipeline our PEs to exploit the available spatial parallelism. By accessing the host memory through the CAPI2 cache-coherent link, NERO acts as a peer to the CPU. This is enabled through the Power-Service Layer (PSL) (②). SNAP (③) allows for seamless integration of the COSMO API with our CAPI-based accelerator. The job manager (④) dispatches jobs to streams, which are managed in the stream scheduler (⑤). The execution of a job is done by streams that determine which data is to be read from the host memory and sent to the PE array through DMA transfers (⑥). The pool of heterogeneous on-chip memory is used to store the input data from the main memory and the intermediate data generated by each PE.

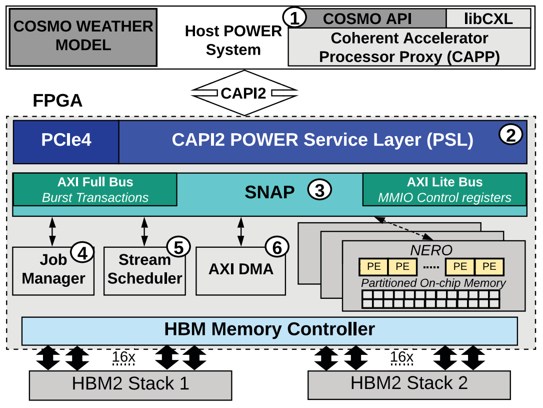


Figure 4: NERO application framework. We co-design our software and hardware using the SNAP framework. COSMO API allows the host to offload kernels to our FPGA platform.

4. Results

4.1. System Integration

We implemented our design on an Alpha-Data ADM-PCIE-9H7 card [1] featuring the Xilinx Virtex Ultrascale+ XCVU37P-FSVH2892-2-e [9] and 8GiB HBM2 (i.e., two stacks of 4GiB each) [5] with an IBM POWER9 as the host system. The POWER9 socket has 16 cores, each of which supports four-thread simultaneous multi-threading. We compare our HBM-based design to a conventional DDR4 DRAM [2] based design. We perform the experiments for the DDR4-based design on an Alpha-Data ADM-PCIE-9V3 card featuring the Xilinx Virtex Ultrascale+ XCVU3P-FFVC1517-2-i [9].

Table 1 provides our system parameters. We have co-designed our hardware and software interface around the SNAP framework while using the HLS design flow.

Table 1: System parameters and hardware configuration for the CPU and the FPGA board.

Host CPU	16-core IBM POWER9 AC922 @3.2 GHz, 4-way SMT
Cache-Hierarchy	32 KiB L1-I/D, 256 KiB L2, 10 MiB L3
System Memory	32GiB RDIMM DDR4 2666 MHz
HBM-based FPGA Board	Alpha Data ADM-PCIE-9H7 Xilinx Virtex Ultrascale+ XCVU37P-2 8GiB (HBM2) with PCIe Gen4 x8
DDR4-based FPGA Board	Alpha Data ADM-PCIE-9V3 Xilinx Virtex Ultrascale+ XCVU3P-2 8GiB (DDR4) with PCIe Gen4 x8

4.2. Evaluation

We run our experiments using a $256 \times 256 \times 64$-point domain similar to the grid domain used by the COSMO weather prediction model. We employ an auto-tuning technique to determine a Pareto-optimal solution (in terms of performance and resource utilization) for our 3D window dimensions. The auto-tuning with OpenTuner exhaustively searches for every tile size in the x- and y-dimensions for `vadvc`.[5] For `hdiff`, we consider sizes in all three dimensions. We define our auto-tuning as a multi-objective optimization with the goal of maximizing performance with minimal resource utilization. Section 3 provides further details on our design. Figure 5 shows hand-tuned and auto-tuned performance and FPGA resource utilization results for `vadvc`, as a function of the chosen tile size. From the figure, we draw two observations.

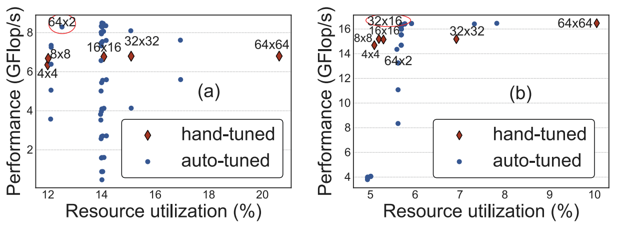


Figure 5: Performance and FPGA resource utilization of single `vadvc` PE, as a function of tile-size, using hand-tuning and auto-tuning for (a) single-precision (32-bit) and (b) half-precision (16-bit). We highlight the Pareto-optimal solution that we use for our `vadvc` accelerator (with a red circle). Note that the Pareto-optimal solution changes with precision.

First, by using the auto-tuning approach and our careful FPGA microarchitecture design, we can get Pareto-optimal results with a tile size of $64 \times 2 \times 64$ for single-precision `vadvc`, which gives us a peak performance of 8.49 GFLOP/s. For half-precision, we use a tile size of $32 \times 16 \times 64$ to achieve a peak performance of 16.5 GFLOP/s. We employ a similar strategy for `hdiff` to attain a single-precision performance of 30.3 GFLOP/s with a tile size of $16 \times 64 \times 8$ and a half-precision performance of 77.8 GFLOP/s with a tile size of $64 \times 8 \times 64$.

Second, in FPGA acceleration, designers usually rely on expert judgement to find the appropriate tile-size and often adapt the design to use homogeneous tile sizes. How-

[5] `vadvc` has dependencies in the z-dimension; therefore, it cannot be parallelized in the z-dimension.

ever, as shown in Figure 5, such hand-tuned implementations lead to sub-optimal results in terms of either resource utilization or performance.

We conclude that the Pareto-optimal tile size depends on the data precision used: a good tile-size for single-precision might lead to poor results when used with half-precision.

Figure 6 shows single-precision performance results for the (a) vertical advection and (b) horizontal diffusion kernels. For both kernels, we implement our design on an HBM- and a DDR4-based FPGA board. To compare the performance results, we scale the number of PEs and analyze the change in execution time. For the DDR4-based design, we can accommodate only 4 PEs on the 9V3 board, while for the HBM-based design, we can fit 14 PEs before exhausting the on-board resources. We draw four observations from the figure.

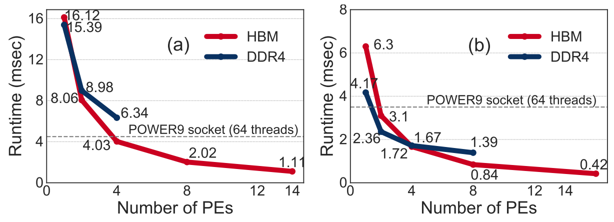


Figure 6: Single-precision performance for (a) `vadvc` and (b) `hdiff`, as a function of accelerator PE count on the HBM- and DDR4-based FPGA boards. We also show the single socket (64 threads) performance of an IBM POWER9 host system for both `vadvc` and `hdiff`.

First, our full-blown HBM-based `vadvc` and `hdiff` implementations provide 120.7 GFLOP/s and 485.4 GFLOP/s performance, which are 4.2× and 8.3× higher than the performance of a complete POWER9 socket. For half-precision, if we use the same amount of PEs as in single precision, our accelerator reaches a performance of 247.9 GFLOP/s for `vadvc` (2.1× the single-precision performance) and 1.2 TFLOP/s for `hdiff` (2.5× the single-precision performance). Our DDR4-based design achieves 34.1 GFLOP/s and 145.8 GFLOP/s for `vadvc` and `hdiff`, respectively, which are 1.2× and 2.5× the performance on the POWER9 CPU.

Second, for a single PE, which fetches data from a single memory channel, the DDR4-based design provides higher performance than the HBM-based design. This is because the DDR4-based FPGA has a larger bus width (512-bit) than an HBM port (256-bit). This leads to a lower transfer rate for an HBM port (0.8-2.1 GT/s[6]) than for a DDR4 port (2.1-4.3 GT/s). One way to match the DDR4 bus width would be to have a single PE fetch data from multiple HBM ports in parallel. However, using more ports leads to higher power consumption (∼1 Watt per HBM port).

Third, as we increase the number of PEs, we observe a linear reduction in the execution time of the HBM-based design. This is because we can evenly divide the computation between multiple PEs, each of which fetches data from a separate HBM port.

[6]Gigatransfers per second.

Fourth, in the DDR4-based design, the use of only a single channel to feed multiple PEs leads to a congestion issue that causes a non-linear run-time reduction. As we increase the number of accelerator PEs, we observe that the PEs compete for a single memory channel, which causes frequent stalls. This phenomenon leads to worse performance scaling characteristics for the DDR4-based design as compared to the HBM-based design.

4.3. Energy Analysis

We compare the energy consumption of our accelerator to a 16-core POWER9 host system. For the POWER9 system, we use the AMESTER[7] tool to measure the active power[8] consumption. We measure 99.2 Watts for `vadvc`, and 97.9 Watts for `hdiff` by monitoring built power sensors in the POWER9 system.

By executing these kernels on an HBM-based board, we reduce the energy consumption by 22× for `vadvc` and 29× for `hdiff` compared to the 16-core POWER9 system. Figure 7 shows the energy efficiency (GFLOPS per Watt) for `vadvc` and `hdiff` on the HBM- and DDR4-based designs. We make three major observations from the figure.

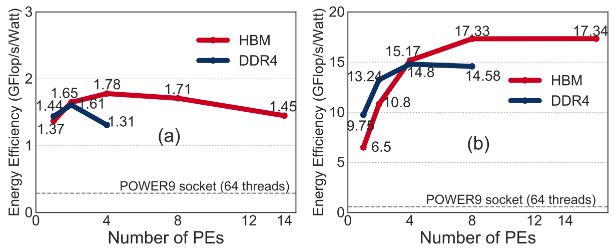


Figure 7: Energy efficiency for (a) `vadvc` and (b) `hdiff` on HBM- and DDR4-based FPGA boards. We also show the single socket (64 threads) energy efficiency of an IBM POWER9 host system for both `vadvc` and `hdiff`.

First, with our full-blown HBM-based designs (i.e., 14 PEs for `vadvc` and 16 PEs for `hdiff`), we achieve energy efficiency values of 1.5 GFLOPS/Watt and 17.3 GFLOPS/Watt for `vadvc` and `hdiff`, respectively.

Second, the DDR4-based design is more energy efficient than the HBM-based design when the number of PEs is small. This observation is inline with our discussion about performance with small PE counts in Section 4.2. However, as we increase the number of PEs, the HBM-based design provides better energy efficiency for memory-bound kernels. This is because more data can be fetched and processed in parallel via multiple ports.

Third, kernels like `vadvc`, with intricate memory access patterns, are not able to reach the peak computational power of FPGAs. The large amount of control flow in `vadvc` leads to large resource consumption. Therefore, when increasing the PE count, we observe a high increase in power consumption with low energy efficiency.

[7]https://github.com/open-power/amester

[8]Active power denotes the difference between the total power of a complete socket (including CPU, memory, fans, I/O, etc.) when an application is running compared to when it is idle.

We conclude that enabling many HBM ports might not always be beneficial in terms of energy consumption because each HBM port consumes $\sim$1 Watt of power consumption. However, data-parallel kernels like `hdiff` can achieve much higher performance in an energy efficient manner with more PEs and HBM ports.

4.4. FPGA Resource Utilization

Table 2 shows the resource utilization of `vadvc` and `hdiff` on the AD9H7 board. We draw two observations. First, there is a high BRAM consumption compared to other FPGA resources. This is because we implement input, field, and output signals as `hls::streams`. In high-level synthesis, by default, streams are implemented as FIFOs that make use of BRAM. Second, `vadvc` has a much larger resource consumption than `hdiff` because `vadvc` has higher computational complexity and requires a larger number of fields to perform the compound stencil calculation. Note that for `hdiff`, we can accommodate more PEs, but in this work, we make use of only a single HBM stack. Therefore, we use 16 PEs because a single HBM stack offers 16 memory ports.

Table 2: FPGA resource utilization in our highest-performing HBM-based designs for `vadvc` and `hdiff`.

Algorithm	BRAM	DSP	FF	LUT	URAM
`vadvc`	81%	39%	37%	55%	53%
`hdiff`	58%	4%	6%	11%	8%

5. Related Work

To our knowledge, this is the first work to evaluate the benefits of using FPGAs equipped with high-bandwidth memory (HBM) to accelerate stencil computation. We exploit near-memory capabilities of such FPGAs to accelerate important weather prediction kernels.

Modern workloads exhibit limited locality and operate on large amounts of data, which causes frequent data movement between the memory subsystem and the processing units [27, 43, 75, 76]. This frequent data movement has a severe impact on overall system performance and energy efficiency. A way to alleviate this *data movement bottleneck* [27, 43, 75, 76, 89] is *near-memory computing* (NMC), which consists of placing processing units closer to memory. NMC is enabled by new memory technologies, such as 3D-stacked memories [5, 62, 65, 66, 81], and also by cache-coherent interconnects [24, 88, 93], which allow close integration of processing units and memory units. Depending on the applications and systems of interest (e.g., [13, 14, 15, 22, 23, 27, 29, 31, 40, 42, 49, 50, 53, 58, 61, 68, 69, 71, 74, 77, 87]), prior works propose different types of near-memory processing units, such as general-purpose CPU cores [13, 16, 27, 28, 29, 39, 64, 69, 78, 83, 86], GPU cores [44, 52, 80, 106], reconfigurable units [41, 55, 57, 90], or fixed-function units [14, 47, 49, 50, 53, 63, 70, 77].

FPGA accelerators are promising to enhance overall system performance with low power consumption. Past works [17, 18, 19, 30, 36, 45, 54, 56, 57, 59, 67] show that FPGAs can be employed effectively for a wide range of applications. The recent addition of HBM to FPGAs presents an opportunity to exploit high memory bandwidth with the low-power FPGA fabric. The potential of high-bandwidth memory [5, 66] has been explored in many-core processors [44, 82] and GPUs [44, 108]. A recent work [102] shows the potential of HBM for FPGAs with a memory benchmarking tool. NERO is the first work to accelerate a real-world HPC weather prediction application using the FPGA+HBM fabric. Compared to a previous work [90] that optimizes only the horizontal diffusion kernel on an FPGA with DDR4 memory, our analysis reveals that the vertical advection kernel has a much lower compute intensity with little to no regularity. Therefore, our work accelerates both kernels that together represent the algorithmic diversity of the entire COSMO weather prediction model. Moreover, compared to [90], NERO improves performance by 1.2$\times$ on a DDR4-based board and 37$\times$ on an HBM-based board for horizontal diffusion by using a dataflow implementation with auto-tuning.

Enabling higher performance for stencil computations has been a subject of optimizations across the whole computing stack [21, 32, 33, 34, 35, 46, 48, 51, 73, 85, 92, 95, 101]. Szustak *et al.* accelerate the MPDATA advection scheme on multi-core CPU [94] and computational fluid dynamics kernels on FPGA [84]. Bianco *et al.* [25] optimize the COSMO weather prediction model for GPUs while Thaler *et al.* [96] port COSMO to a many-core system. Wahib *et al.* [100] develop an analytical performance model for choosing an optimal GPU-based execution strategy for various scientific applications, including COSMO. Gysi *et al.* [48] provide guidelines for optimizing stencil kernels for CPU–GPU systems.

6. Conclusion

We introduce NERO, the first design and implementation on a reconfigurable fabric with high-bandwidth memory (HBM) to accelerate representative weather prediction kernels, i.e., vertical advection (`vadvc`) and horizontal diffusion (`hdiff`), from a real-world weather prediction application. These kernels are compound stencils that are found in various weather prediction applications, including the COSMO model. We show that compound kernels do not perform well on conventional architectures due to their complex data access patterns and low data reusability, which make them memory-bounded. Therefore, they greatly benefit from our near-memory computing solution that takes advantage of the high data transfer bandwidth of HBM.

NERO's implementations of `vadvc` and `hdiff` outperform the optimized software implementations on a 16-core POWER9 with 4-way multithreading by 4.2$\times$ and 8.3$\times$, with 22$\times$ and 29$\times$ less energy consumption, respectively. We conclude that hardware acceleration on an FPGA+HBM fabric is a promising solution for compound stencils present in weather prediction applications. We hope that our reconfigurable near-memory accelerator inspires developers of different high-performance computing applications that suffer from the memory bottleneck.

Acknowledgments

This work was performed in the framework of the Horizon 2020 program for the project "Near-Memory Computing (NeMeCo)". It is funded by the European Commission under Marie Sklodowska-Curie Innovative Training Networks European Industrial Doctorate (Project ID: 676240). Special thanks to Florian Auernhammer and Raphael Polig for providing access to IBM systems. We appreciate valuable discussions with Kaan Kara and Ronald Luijten. We would also like to thank Bruno Mesnet and Alexandre Castellane from IBM France for help with the SNAP framework. This work was partially supported by the H2020 research and innovation programme under grant agreement No 732631, project OPRECOMP. We also thank Google, Huawei, Intel, Microsoft, SRC, and VMware for their funding support.

References

[1] "ADM-PCIE-9H7-High-Speed Communications Hub, https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9h7."
[2] "ADM-PCIE-9V3-High-Performance Network Accelerator, https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9v3."
[3] "AXI High Bandwidth Memory Controller v1.0, https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf."
[4] "AXI Reference Guide, https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf."
[5] "High Bandwidth Memory (HBM) DRAM (JESD235), https://www.jedec.org/document_search?search_api_views_fulltext=jesd235."
[6] "High Bandwidth Memory (HBM) DRAM, https://www.jedec.org/sites/default/files/JESD235B-HBM_Ballout.zip."
[7] "Intel Stratix 10 MX FPGAs, https://www.intel.com/content/www/us/en/products/programmable/sip/stratix-10-mx.html."
[8] "OpenPOWER Work Groups, https://openpowerfoundation.org/technical/working-groups."
[9] "Virtex UltraScale+, https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html."
[10] "Vivado High-Level Synthesis, https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html."
[11] "Xilinx VCU1525, https://www.xilinx.com/products/boards-and-kits/vcu1525-a.html."
[12] "Xilinx Virtex Ultrascale+, https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html."
[13] J. Ahn *et al.*, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[14] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[15] B. Akin *et al.*, "Data Reorganization in Memory Using 3D-stacked DRAM," *ISCA*, 2015.
[16] M. Alian *et al.*, "Application-Transparent Near-Memory Processing Architecture with Memory Channel Network," in *MICRO*, 2018.
[17] M. Alser *et al.*, "Shouji: a fast and efficient pre-alignment filter for sequence alignment," *Bioinformatics*, 2019.
[18] M. Alser *et al.*, "GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping," *Bioinformatics*, 2017.
[19] M. Alser *et al.*, "SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs," *arXiv preprint*, 2019.
[20] J. Ansel *et al.*, "OpenTuner: An Extensible Framework for Program Autotuning," in *PACT*, 2014.
[21] A. Armejach *et al.*, "Stencil Codes on a Vector Length Agnostic Architecture," in *PACT*, 2018.
[22] H. Asghari-Moghaddam *et al.*, "Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems," in *MICRO*, 2016.
[23] O. O. Babarinsa and S. Idreos, "JAFAR: Near-Data Processing for Databases," in *SIGMOD*, 2015.
[24] B. Benton, "CCIX, Gen-Z, OpenCAPI: Overview and Comparison," in *OFA*, 2017.
[25] M. Bianco *et al.*, "A GPU Capable Version of the COSMO Weather Model," *ISC*, 2013.
[26] L. Bonaventura, "A Semi-implicit Semi-Lagrangian Scheme using the Height Coordinate for a Nonhydrostatic and Fully Elastic Model of Atmospheric Flows," *JCP*, 2000.
[27] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," *ASPLOS*, 2018.
[28] A. Boroumand *et al.*, "CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators," in *ISCA*, 2019.
[29] A. Boroumand *et al.*, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *CAL*, 2016.
[30] L.-W. Chang *et al.*, "Collaborative Computing for Heterogeneous Integrated Systems," in *ICPE*, 2017.
[31] P. Chi *et al.*, "PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory," 2016.
[32] Y. Chi *et al.*, "SODA: Stencil with Optimized Dataflow Architecture," in *ICCAD*, 2018.
[33] M. Christen *et al.*, "PATUS: A Code Generation and Autotuning Framework for Parallel Iterative Stencil Computations on Modern Microarchitectures," in *IPDPS*, 2011.
[34] K. Datta *et al.*, "Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors," *SIAM review*, 2009.
[35] J. de Fine Licht *et al.*, "Designing scalable FPGA architectures using high-level synthesis," in *PPoPP*, 2018.
[36] D. Diamantopoulos *et al.*, "ecTALK: Energy efficient coherent transprecision accelerators - The bidirectional long short-term memory neural network case," in *COOL CHIPS*, 2018.
[37] D. Diamantopoulos and C. Hagleitner, "A System-Level Transprecision FPGA Accelerator for BLSTM Using On-chip Memory Reshaping," in *FPT*, 2018.
[38] G. Doms and U. Schättler, "The Nonhydrostatic Limited-Area Model LM (Lokal-model) of the DWD. Part I: Scientific Documentation," *DWD, GB Forschung und Entwicklung*, 1999.
[39] M. Drumond *et al.*, "The Mondrian Data Engine," in *ISCA*, 2017.
[40] A. Farmahini-Farahani *et al.*, "NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules," in *HPCA*, 2015.
[41] M. Gao and C. Kozyrakis, "HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing," in *HPCA*, 2016.
[42] M. Gao *et al.*, "Practical Near-Data Processing for In-Memory Analytics Frameworks," in *PACT*, 2015.
[43] S. Ghose *et al.*, "Processing-in-memory: A workload-driven perspective," *IBM JRD*, 2019.
[44] S. Ghose *et al.*, "Demystifying Complex Workload-DRAM Interactions: An Experimental Study," *POMACS*, 2019.
[45] H. Giefers *et al.*, "Accelerating arithmetic kernels with coherent attached FPGA coprocessors," in *DATE*, 2015.
[46] J. González and A. González, "Speculative Execution via Address Prediction and Data Prefetching," in *ICS*, 1997.
[47] B. Gu *et al.*, "Biscuit: A Framework for Near-data Processing of Big Data Workloads," *ISCA*, 2016.
[48] T. Gysi *et al.*, "MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures," in *SC*, 2015.
[49] M. Hashemi *et al.*, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," in *ISCA*, 2016.
[50] M. Hashemi *et al.*, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," in *MICRO*, 2016.
[51] T. Henretty *et al.*, "Data Layout Transformation for Stencil Computations on Short-Vector SIMD Architectures," in *CC*, 2011.
[52] K. Hsieh *et al.*, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.
[53] K. Hsieh *et al.*, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.
[54] S. Huang *et al.*, "Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures," in *ICPE*, 2019.
[55] Z. István *et al.*, "Caribou: Intelligent Distributed Storage," *VLDB*, 2017.
[56] J. Jiang *et al.*, "Boyi: A Systematic Framework for Automatically Deciding the Right Execution Model of OpenCL Applications on FPGAs," in *FPGA*, 2020.
[57] S.-W. Jun *et al.*, "BlueDBM: An Appliance for Big Data Analytics," in *ISCA*, 2015.
[58] Y. Kang *et al.*, "Enabling Cost-effective Data Processing with Smart SSD," in *MSST*, 2013.
[59] K. Kara *et al.*, "FPGA-accelerated Dense Linear Machine Learning: A Precision-Convergence Trade-off," in *FCCM*, 2017.
[60] S. Kehler *et al.*, "High Resolution Deterministic Prediction System (HRDPS) Simulations of Manitoba Lake Breezes," *Atmosphere-Ocean*, 2016.
[61] D. Kim *et al.*, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," 2016.
[62] J. Kim *et al.*, "A 1.2 V 12.8 GB/s 2 Gb Mobile Wide-I/O DRAM With 4×128 I/Os Using TSV Based Stacking," *JSSC*, 2012.
[63] J. S. Kim *et al.*, "GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC Genomics*, 2018.
[64] G. Koo *et al.*, "Summarizer: Trading Communication with Computing Near Storage," in *MICRO*, 2017.
[65] D. U. Lee *et al.*, "25.2 A 1.2V 8Gb 8-Channel 128GB/s High-Bandwidth Memory (HBM) Stacked DRAM with Effective Microbump I/O Test Methods Using 29nm Process and TSV," in *ISSCC*, 2014.
[66] D. Lee *et al.*, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," *ACM TACO*, 2016.
[67] J. Lee *et al.*, "ExtraV: Boosting Graph Processing Near Storage with a Coherent Accelerator," 2017.
[68] J. H. Lee *et al.*, "BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models," in *PACT*, 2015.
[69] V. T. Lee *et al.*, "Application Codesign of Near-Data Processing for Similarity Search," in *IPDPS*, 2018.

[70] J. Liu *et al.*, “Processing-in-Memory for Energy-efficient Neural Network Training: A Heterogeneous Approach,” in *MICRO*, 2018.
[71] Z. Liu *et al.*, “Concurrent Data Structures for Near-Memory Computing,” in *SPAA*, 2017.
[72] D. Mayhew and V. Krishnan, “PCI Express and Advanced Switching: Evolutionary Path to Building Next Generation Interconnects,” in *HOTI*, 2003.
[73] J. Meng and K. Skadron, “A Performance Study for Iterative Stencil Loops on GPUs with Ghost Zone Optimizations,” *IJPP*, 2011.
[74] A. Morad *et al.*, “GP-SIMD Processing-in-Memory,” *ACM TACO*, 2015.
[75] O. Mutlu *et al.*, “Processing Data Where It Makes Sense: Enabling In-Memory Computation,” *MicPro*, 2019.
[76] O. Mutlu *et al.*, “Enabling Practical Processing in and near Memory for Data-Intensive Computing,” in *DAC*, 2019.
[77] L. Nai *et al.*, “GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks,” in *HPCA*, 2017.
[78] R. Nair *et al.*, “Active Memory Cube: A Processing-in-Memory Architecture for Exascale Systems,” *IBM JRD*, 2015.
[79] R. B. Neale *et al.*, “Description of the NCAR Community Atmosphere Model (CAM 5.0),” *NCAR Tech. Note*, 2010.
[80] A. Pattnaik *et al.*, “Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities,” in *PACT*, 2016.
[81] J. T. Pawlowski, “Hybrid Memory Cube (HMC),” in *HCS*, 2011.
[82] C. Pohl *et al.*, “Joins on high-bandwidth memory: a new level in the memory hierarchy,” *VLDB*, 2019.
[83] S. H. Pugsley *et al.*, “NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads,” in *ISPASS*, 2014.
[84] K. Rojek *et al.*, “CFD Acceleration with FPGA,” in *H2RC*, 2019.
[85] K. Sano *et al.*, “Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth,” *TPDS*, 2014.
[86] P. C. Santos *et al.*, “Operand Size Reconfiguration for Big Data Processing in Memory,” in *DATE*, 2017.
[87] V. Seshadri *et al.*, “Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses,” in *MICRO*, 2015.
[88] D. Sharma, “Compute Express Link,” *CXL Consortium White Paper*, 2019.
[89] G. Singh *et al.*, “Near-Memory Computing: Past, Present, and Future,” *MicPro*, 2019.
[90] G. Singh *et al.*, “NARMADA: Near-memory horizontal diffusion accelerator for scalable stencil computations,” in *FPL*, 2019.
[91] G. Singh *et al.*, “NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning,” in *DAC*, 2019.
[92] R. Strzodka *et al.*, “Cache Oblivious Parallelograms in Iterative Stencil Computations,” in *ICS*, 2010.
[93] J. Stuecheli *et al.*, “IBM POWER9 opens up a new era of acceleration enablement: OpenCAPI,” *IBM JRD*, 2018.
[94] L. Szustak *et al.*, “Using Intel Xeon Phi Coprocessor to Accelerate Computations in MPDATA Algorithm,” in *PPAM*, 2013.
[95] Y. Tang *et al.*, “The Pochoir Stencil Compiler,” in *SPAA*, 2011.
[96] F. Thaler *et al.*, “Porting the COSMO Weather Model to Manycore CPUs,” in *PASC*, 2019.
[97] L. Thomas, “Elliptic Problems in Linear Differential Equations over a Network,” *Watson Sci. Comput. Lab. Rept., Columbia University*, 1949.
[98] P.-A. Tsai *et al.*, “Jenga: Software-Defined Cache Hierarchies,” in *ISCA*, 2017.
[99] J. van Lunteren *et al.*, “Coherently Attached Programmable Near-Memory Acceleration Platform and its application to Stencil Processing,” in *DATE*, 2019.
[100] M. Wahib and N. Maruyama, “Scalable Kernel Fusion for Memory-Bound GPU Applications,” in *SC*, 2014.
[101] H. M. Waidyasooriya *et al.*, “OpenCL-Based FPGA-Platform for Stencil Computation and Its Optimization Methodology,” *TPDS*, 2017.
[102] Z. Wang *et al.*, “Shuhai: Benchmarking High Bandwidth Memory on FPGAs,” in *FCCM*, 2020.
[103] L. Wenzel *et al.*, “Getting Started with CAPI SNAP: Hardware Development for Software Engineers,” in *Euro-Par*, 2018.
[104] S. Williams *et al.*, “Roofline: An Insightful Visual Performance Model for Multicore architectures,” *CACM*, 2009.
[105] J. Xu *et al.*, “Performance Tuning and Analysis for Stencil-Based Applications on POWER8 Processor,” *ACM TACO*, 2018.
[106] D. Zhang *et al.*, “TOP-PIM: Throughput-Oriented Programmable Processing in Memory,” in *HPDC*, 2014.
[107] J. A. Zhang *et al.*, “Evaluating the Impact of Improvement in the Horizontal Diffusion Parameterization on Hurricane Prediction in the Operational Hurricane Weather Research and Forecast (HWRF) Model,” *Weather and Forecasting*, 2018.
[108] M. Zhu *et al.*, “Performance Evaluation and Optimization of HBM-Enabled GPU for Data-intensive Applications,” *VLSI*, 2018.