# Accelerating Weather Prediction using Near-Memory Reconfigurable Fabric

GAGANDEEP SINGH, ETH ZÃijrich, Switzerland
DIONYSIOS DIAMANTOPOULOS, IBM Research Europe, ZÃijrich Lab, Switzerland
JUAN GÓMEZ-LUNA, ETH ZÃijrich, Switzerland
CHRISTOPH HAGLEITNER, IBM Research Europe, ZÃijrich Lab, Switzerland
SANDER STUIJK, Eindhoven Univesity of Technology, The Netherlands
HENK CORPORAAL, Eindhoven Univesity of Technology, The Netherlands
ONUR MUTLU, ETH ZÃijrich, Switzerland

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 OCAPI (Open Coherent Accelerator Processor Interface) to an IBM POWER9 host system. Our experimental results show that NERO outperforms a 16-core POWER9 system by 5.3× and 12.7× when running two different compound stencil kernels. NERO reduces the energy consumption by 12× and 35× for the same two kernels over the POWER9 system with an energy efficiency of 1.61 GFLOPS/Watt and 21.01 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.

CCS Concepts: • **Hardware → Hardware-software codesign**; • **Computer systems organization → Reconfigurable computing**; *Heterogeneous (hybrid) systems.*

Additional Key Words and Phrases: FPGA, Near-Memory Computing, Weather Modeling, High-Performance Computing, Processing in Memory

## 1 Introduction

Accurate weather prediction and climate modeling using detailed weather models is essential to make weather-dependent and climate-related decisions in a timely manner. These models are based on physical laws that describe various components of the atmosphere [137]. The Consortium for Small-Scale Modeling (COSMO) [59] built one such weather model to meet the high-resolution forecasting requirements of weather services. The

Authors' addresses: Gagandeep Singh, gagan.gagandeepsingh@safari.ethz.ch, ETH ZÃijrich, Switzerland; Dionysios Diamantopoulos, did@zurich.ibm.com, IBM Research Europe, ZÃijrich Lab, Switzerland; Juan Gómez-Luna, juan.gomez@safari.ethz.ch, ETH ZÃijrich, Switzerland; Christoph Hagleitner, hle@zurich.ibm.com, IBM Research Europe, ZÃijrich Lab, Switzerland; Sander Stuijk, s.stuijk@tue.nl, Eindhoven Univesity of Technology, The Netherlands; Henk Corporaal, h.corporaal@tue.nl, Eindhoven Univesity of Technology, The Netherlands; Onur Mutlu, omutlu@ethz.ch, ETH ZÃijrich, Switzerland.


1936-7406/2022/2-ART $15.00
https://doi.org/10.1145/3501804

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 in the vertical dimension (i.e., parameters are dependent on each other at the same time instance [42]) and explicit discretization in the horizontal dimension (i.e., a solution is dependent on the previous system state [42]). The use of different discretizations leads to three computational patterns [161]: 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 [79]. *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 [97, 125, 177]. 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 [165].

Figure 1 shows the roofline plot [173] for an IBM 16-core POWER9 CPU (IC922).[1] After optimizing the `vadvc` and `hdiff` kernels for the POWER architecture[2] by following the approach in [175], 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.

Heterogeneous computing has emerged as an answer to improve the system performance in an energy-efficient way. Heterogeneous computing entails complementing processing elements with different compute capabilities, each to perform the tasks to which it is best suited. In the HPC domain, coupling specialized compute units with general-purpose cores can meet the high-performance computing demands with the ability to realize exascale systems needed to process data-intensive workloads [123]. The graphics processing unit (GPU) is one of the most popular acceleration platforms. GPUs have been used to accelerate workloads like computer graphics and linear algebra [166] because of their many-core architecture. However, GPUs are power-hungry due to high transistor density and, depending on the power constraints, may not always be the ideal platform for implementation. Recently, the use of field-programmable gate array (FPGA) in accelerating machine learning workloads with high energy efficiency has inspired researchers to explore the use of FPGAs instead of GPUs for various high-performance computing applications [48, 61]. FPGAs provide a unique combination of flexibility and performance without the cost, complexity, and risk of developing custom application-specific integrated circuits (ASICs). Modern FPGAs show four key trends.

- The advancements in the stacking technology with high-bandwidth memory (HBM) [7, 8, 10, 22, 104] blends DRAM on the same package as an FPGA. This integration allows us to implement our accelerator logic in close proximity to the memory with a lower latency and much higher memory bandwidth than the traditional DDR4-based FPGA boards. Memory-bound applications on FPGAs are limited by the relatively low DDR4 bandwidth (72 GB/s for four independent dual-rank DIMM interfaces [21]). HBM-based FPGAs can overcome this limitation with a peak bandwidth of 410 GB/s [95].

[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.

[2] We use single instruction, multiple data (SIMD) [38, 65], and simultaneous multithreading (SMT) [164] techniques to fill the hardware pipelines. We use the same tiling size for both CPU and FPGA-based designs. While compiling these kernels, we use the IBM XLC [9] 16 C/C++ compiler that is optimized for IBM POWER [134] machines with the following flags: qarch=pwr9, qtune=pwr9, O3, q64, qprefetch=aggressive, qsmp=omp, and qsimd=auto.

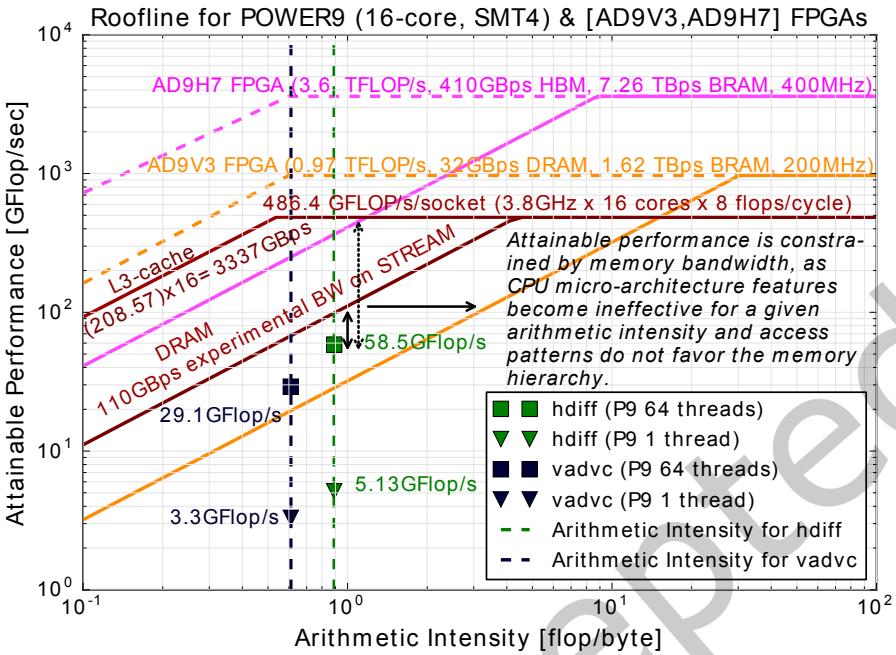


Fig. 1. Roofline [173] for POWER9 (1-socket) showing vertical advection (vadvc) and horizontal diffusion (hdiff) kernels for single-thread and 64-thread implementations. The plot shows also the rooflines of the FPGAs used in our work with peak DRAM and on-chip BRAM bandwidth.



- New cache-coherent interconnects, such as Open Coherent Accelerator Processor Interface (OCAPI) [156], Cache Coherent Interconnect for Accelerators (CCIX) [39], and Compute Express Link (CXL) [145], allow tight integration of FPGAs with CPUs at high bidirectional bandwidth (on the order of tens of GB/s). This integration reduces programming effort and enables us to coherently access the host system's memory through a pointer rather than having multiple copies of the data.
- The introduction of UltraRAM (URAM) [17] along with the BlockRAM (BRAM) that offers massive *scratch-pad*-based on-chip memory next to the logic. URAM is more denser than BRAM, but is not as distributed in the FPGA layout as the BRAM.
- FPGAs are being manufactured with an advanced technology node of 7-14nm FinFET technology [67] that offers higher performance.

These above trends suggest that modern FPGA architectures with near-memory compute capabilities can alleviate the *memory bottleneck* of real-world data-intensive applications [148]. However, a study of their advantages for real-world memory-bound applications is still missing. 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) [7, 104, 105] that are attached 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 on-package 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.

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 5.3× and 12.7× and energy reductions of 12× and 35×, respectively, with respect to optimized CPU implementations [175].

The major contributions of NERO 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 5.3× for the vertical advection (`vadvc`) and 12.7× for the horizontal diffusion (`hdiff`) kernels with energy reductions of 12× and 35×, respectively. Our design provides an energy efficiency of 1.61 GLOPS/Watt and 21.01 GFLOPS/Watt for `vadvc` and `hdiff` kernels, respectively.

This work extends our previous work [152] as follows. First, we add new results using a state-of-the-art OpenCAPI (OCAPI) interface [156]. OCAPI provides two key opportunities compared to our previous CAPI2 implementation: (1) OCAPI has double the bitwidth (1024-bit) of our previously used CAPI2 interface, and (2) the memory coherency logic has been moved to the host CPU side, which provides more area and allows us to run our design at a higher frequency. Our implementation and evaluation with state-of-the-art OCAPI improves the performance for our two main workloads from weather modeling (vadvc and hdiff) by 37% and 44%, respectively, compared to a CAPI-based HBM design [152]. All the functions in our design now operate on a 1024-bit per clock rather than 512-bit per clock, utilizing the maximum processing throughput of OCAPI (POWER9 cache line is 1024-bit). Thus, we can build a dataflow accelerator with wide AXI streams of 1024-bit. Second, we develop `HBM_multi+OCAPI`-based versions of our workloads that make use of multiple channels per processing element (PE). This multi-channel implementation allows the PEs to exploit significantly higher bandwidth. As a result, the workloads achieve an average speedup of 1.5x over the single-channel version for a single PE. Third, we provide a discussion section (Section 5) that provides various insights and takeaways while designing HBM-based FPGA accelerators, which we believe would be useful for future FPGA architects and programmers. Fourth, we implement and evaluate the `copy` stencil [161] (Figure 3), a stencil from the COSMO model to benchmark the peak performance on a platform. Fifth, we use the OC-Accel framework[3] instead of the SNAP framework for OCAPI accelerator development and deployment. Sixth, we provide a comparison of state-of-the-art works in stencil acceleration (Table 4).

[3]https://github.com/OpenCAPI/oc-accel

## 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 OC-Accel (OpenCAPI Acceleration) framework 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. In weather and climate modeling, several stencil operations are compounded together that operate on multiple input elements to generate one or more output results. 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, as shown in Figure 2a. A single *Laplacian* stencil accesses the input grid at five memory offsets, the result of which is used to calculate the *flux* stencil. hdiff has purely horizontal access patterns and does not have dependencies in the vertical dimension. Thus, it can be fully parallelized in the vertical dimension. Figure 2b shows the memory layout for the horizontal diffusion kernel. We observe that the indirect memory accesses of the input grid domain can severely impact cache efficiency on our current CPU-based systems.

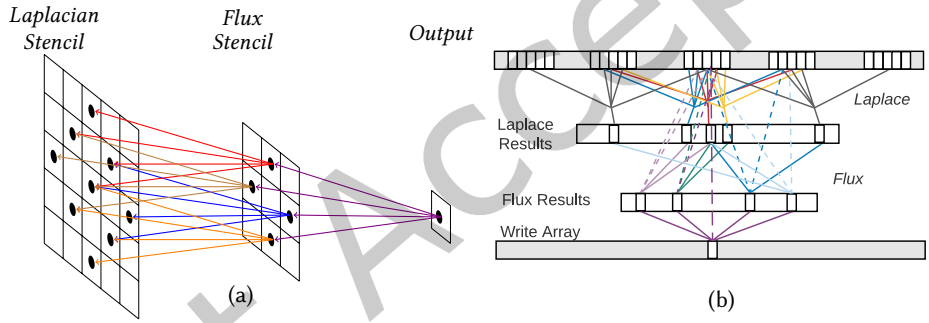


Fig. 2. (a) Horizontal diffusion kernel composition using Laplacian and flux stencils in a two dimensional plane [153]. (b) Memory layout of horizontal diffusion from 3D grid onto 1D array.

Vertical advection has a higher degree of complexity since it uses the Thomas algorithm [162] to solve a tridiagonal matrix of weather data (called *fields*, such as air pressure, wind velocity, and temperature) along the vertical axis. vadvc consists of a forward sweep that is followed by a backward sweep along the vertical dimension. vadvc requires access to the weather data, which are stored as array structures while performing forward and sweep computations. Unlike the conventional stencil kernels, vertical advection has dependencies in the vertical direction, which leads to limited available parallelism and irregular memory access patterns. For example, when the input grid is stored by *row*, accessing data elements in the *depth* dimension typically results in many cache misses [175].

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 [175] suffer from limited data locality and inefficient memory usage, as our roofline analysis in Figure 1 exposes. In Figure 3 we implement a *copy stencil* from the COSMO weather model to evaluate the performance potential of our HBM-based FPGA platform for the weather prediction application. A copy *stencil* performs an element-wise copy operation over the complete input grid. It is the simplest stencil found in the COSMO model

and, hence, serves as a benchmark to characterize the achievable peak performance on a platform for weather kernels. To implement a copy stencil, we divide the 3D grid data among the processing elements (PEs), where each PE performs an element-wise copy operation. We were able to enable only 24 HBM memory channels because adding more HBM channels leads to timing constraint violations. From the figure, we make two observations. First, as we increase the number of processing elements (PEs), we can exploit data-level parallelism because of dedicated HBM channels serving data to a PE. Second, the maximum achievable performance tends to saturate after 16 PEs. Since we implement our design in a dataflow pipeline manner, all the functions run in parallel, and the overall latency is equal to the maximum latency out of all the functions. After 16 PEs, for copy, we observe that most of the time is spent in the FPGA computation logic rather than the transfer of data from an HBM memory channel.

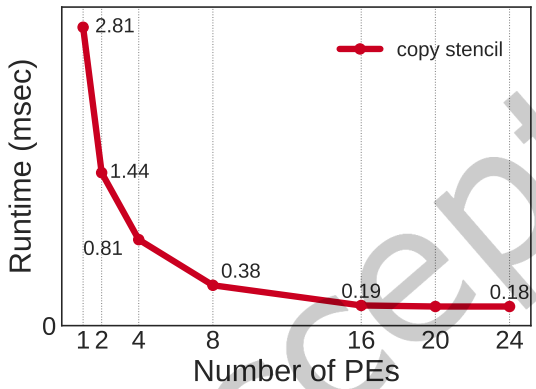


Fig. 3. Performance of copy stencil on our HBM-based FPGA platform.

## 2.2 OC-Accel Framework

The OpenPOWER Foundation Accelerator Workgroup [14] created the OC-Accel framework, an open-source environment for FPGA programming productivity. OC-Accel provides three key benefits [172]: (i) it enables an improved developer productivity for FPGA acceleration and eases the use of CAPI's cache-coherence mechanism, (ii) it places FPGA-accelerated compute engines, also known as FPGA *actions*, closer to relevant data to achieve better performance, and (iii) access to FPGA memory via user-level DMA (Direct Memory Access) semantics. OC-Accel provides a simple API to invoke an accelerated *action* and 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 [20].

The benefits of employing such cache-coherent interconnect links for attaching FPGAs to CPUs, as opposed to the traditional DMA-like communication protocols (e.g., PCIe), are not only the ultra lower-latency and the higher bandwidth of the communication, but most importantly, the ability of the accelerator to access the entire memory space of the CPU coherently, without consuming excessive CPU cycles. Traditionally, the host processor has a shared memory space across its cores with coherent caches. Attached devices such as FPGAs, GPUs, network, and storage controllers are memory-mapped and use a DMA to transfer data between local and system memory across an interconnect such as PCIe. The attached devices can not see the entire system memory but only a part of it. Communication between the host processor and attached devices requires an inefficient software stack, including user-space software, drivers, and kernel-space modules, in comparison to the communication scheme between CPU cores using shared memory. Especially when DRAM memory bandwidth

becomes a constraint, requiring extra memory-to-memory copies to move data from one address space to another is cumbersome and low performance [62, 139]. This is the driving force of the industry to push for coherency and shared memory across CPU cores and attached devices, like FPGAs. This way, the accelerators act as peers to the processor cores. Note that CAPI2 is built on top of PCIe. However, CAPI2 provides the following two advantages. First, a CAPI-attached device, unlike a PCIe device, can perform Direct Memory Access (DMA) to application memory *without* calls to a device driver or underlying operating system kernel, resulting in a reduction in latency. Avoiding these unnecessary memory calls improves performance significantly compared to the traditional PCIe I/O model [157]. Second, CAPI2 provides cache-coherent access to the CPU memory, allowing the FPGA to *directly* access the host memory. Such direct cache-coherent access reduces the FPGA developer's burden and debugging time by overcoming read after write (RAW) and write and read (WAR) dependencies, which is typically the FPGA developerâĂŹs responsibility. OCAPI is a new technology built from the ground up. It includes a faster PHY layer (BlueLink 25Gb/s x8 lanes [156]) than its CAPI predecessors, providing double the bitwidth between the host and an accelerator.

## 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 [24, 25, 43, 44, 71, 85, 86, 100, 118, 120, 121, 147, 150] that exploits near high-bandwidth memory acceleration.

Figure 4a 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 Open Coherent Accelerator Processor Interface (OCAPI). The FPGA consists of two HBM stacks[4], 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 TLx (Transaction Layer) and the DLx (Data Link Layer), which are the OCAPI endpoint on the FPGA. An AFU comprises of multiple *processing elements* (PEs) that perform compound stencil computation. Figure 5 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.

The weather data, based on the atmospheric model resolution grid, is stored in the DRAM of the host system (❶ in Figure 5). We employ the double buffering technique between the CPU and the FPGA to hide the PCIe (Peripheral Component Interconnect Express [114]) transfer latency. By configuring a buffer of 64 cache lines, between the AXI4 interface of OCAPI/TLx-DLx and the AFU, we can reach the theoretical peak bandwidth of OCAPI (i.e., 32 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 4a) handles the data placement to the appropriate memory type based on the programmer's directives.

On the FPGA, following the initial buffering (❷), the transferred grid data is mapped onto the HBM memory (❸). As the FPGA has limited resources, we propose a 3D window-based grid transfer from the host DRAM to the

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

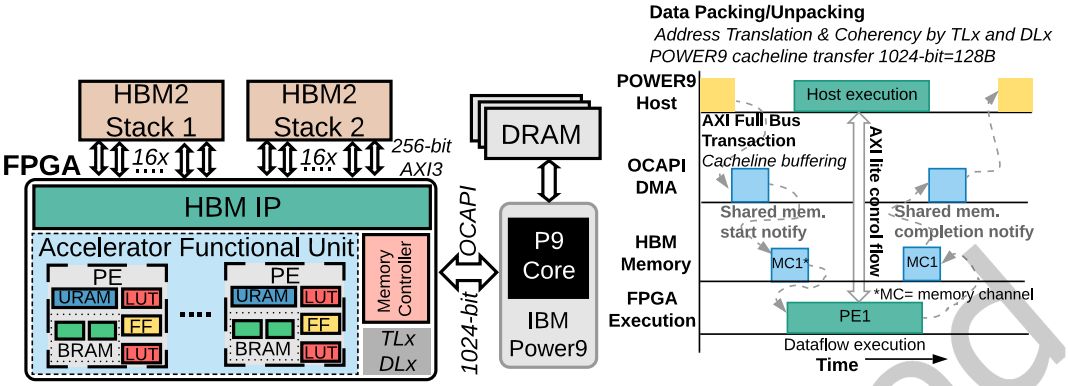


Fig. 4. (a) Heterogeneous platform with an IBM POWER9 system connected to an HBM-based FPGA board via OCAPI. We also show components of an FPGA: flip-flop (FF), lookup table (LUT), UltraRAM (URAM), and Block RAM (BRAM). (b) Execution timeline with data flow sequence from the host DRAM to the onboard FPGA memory.

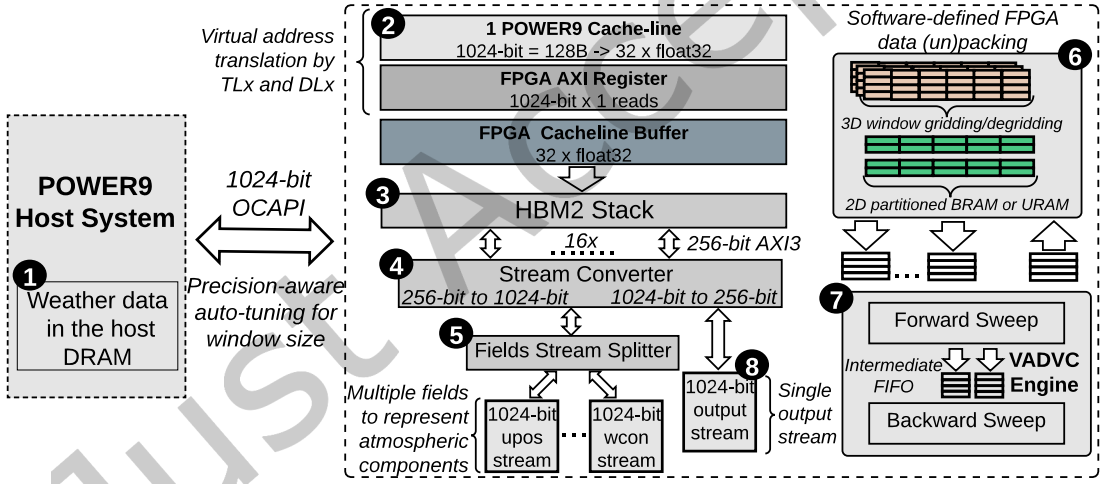


Fig. 5. Architecture overview of NERO with data flow sequence from the host DRAM to the on-board FPGA memory via POWER9 cachelines. We depict a single processing element (PE) fetching data from a dedicated HBM port. The number of HBM ports scales linearly with the number of PEs. Heterogeneous partitioning of on-chip memory blocks reduces read and write latencies across the FPGA memory hierarchy.

FPGA, facilitating a smaller, less power-hungry deployment. The window size represents the portion of the grid a processing element (PE in Figure 4a) 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 4.3) show that: (1) finding the best window size is critical in terms of the area vs. performance trade-off,

and (2) the 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 [34], which uses machine-learning techniques to guide the design-space exploration [151].

Our design consists of multiple PEs (shown in Figure 4a) 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 a quarter of the size of the OCAPI bitwidth (1024-bit). Therefore, to match the OCAPI bandwidth, we introduce a stream converter logic (❹) that converts a 256-bit HBM stream to a 1024-bit stream (OCAPI compatible) or vice versa. From HBM, a PE reads a single stream of data that consists of all the fields[5] 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 (1024-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 5). 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 [58]. 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 and cache miss latency, 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 [163].

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 1024-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 1024-bit wide stream goes through an HBM stream converter logic (❹) that converts the stream bitwidth to HBM port size (256-bit).

Figure 4b 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 OCAPI interface. The OC-Accel 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.

### 3.2 NERO Application Framework

Figure 6 shows the NERO application framework to support our architecture. Our previous work [152, 153] describes the corresponding application framework using SNAP-CAPI2. A software-defined COSMO API (①)

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

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 OCAPI cache-coherent link, NERO acts as a peer to the CPU. This is enabled through the TLx (Transaction Layer) and the DLx (Data Link Layer) (②). OC-Accel (③) allows for seamless integration of the COSMO API with our OCAPI-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.

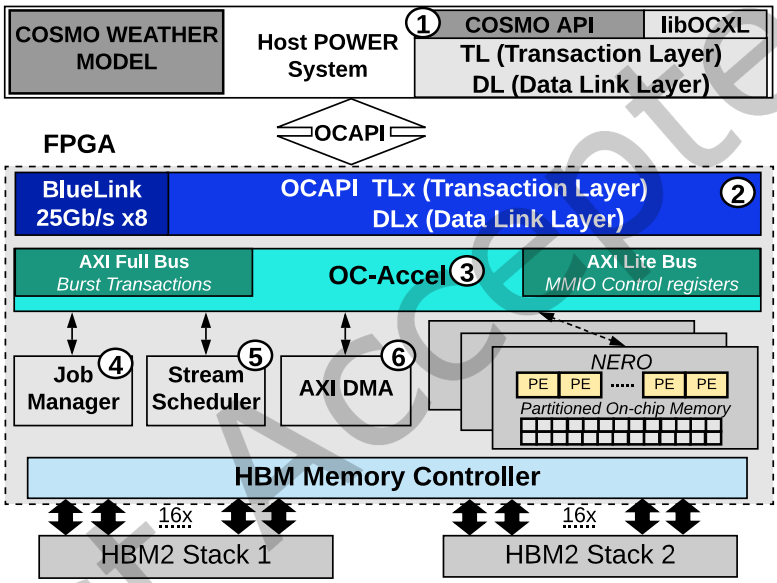


Fig. 6. NERO application framework. We co-design our software and hardware using the OC-Accel framework. COSMO API allows the host to offload kernels to our FPGA platform.



## 4 Results

### 4.1 Experimental Setup

We evaluate our accelerator designs for `vadvc`, and `hdiff` in terms of performance, energy consumption, and FPGA resource utilization on two different FPGAs, and two different external data communication interfaces between the CPU and the FPGA board. We implement our accelerator designs for `vadvc`, and `hdiff` on both 1) an Alpha-Data ADM-PCIE-9H7 card [1] featuring the Xilinx Virtex Ultrascale+ XCVU37P-FSVH2892-2-e [19] with 8GiB HBM2 [7] and 2) an Alpha-Data ADM-PCIE-9V3 card [2] featuring the Xilinx Virtex Ultrascale+ XCVU3P-FFVC1517-2-i with 8GiB DDR4 [19], connected to an IBM POWER9 host system. For the external data communication interface, we use both CAPI2 [157] and the state-of-the-art OCAPI (OpenCAPI) [156] interface. We compare these implementations to execution on a POWER9 CPU with 16 cores (using all 64 hardware threads).

Table 1 provides our system parameters. We co-design our hardware and software interface around the OC-Accel framework [13] while using the HLS design flow. Our development machine is x86 Intel® Xeon®7.9.2009 [5] distribution with GNU Compiler Collection (GCC) version 4.8.5 [6]. We use Xilinx Vivado 2019.2 [23] suite to develop our accelerator designs.

### 4.2 Performance Tuning

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`.[6] 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. We evaluate frequency values between 50-400 MHz, with an increment of 50MHz, utilizing the complete spectrum of compatible frequency configurations supported by the OC-Accel framework [13]. Figure 7 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.

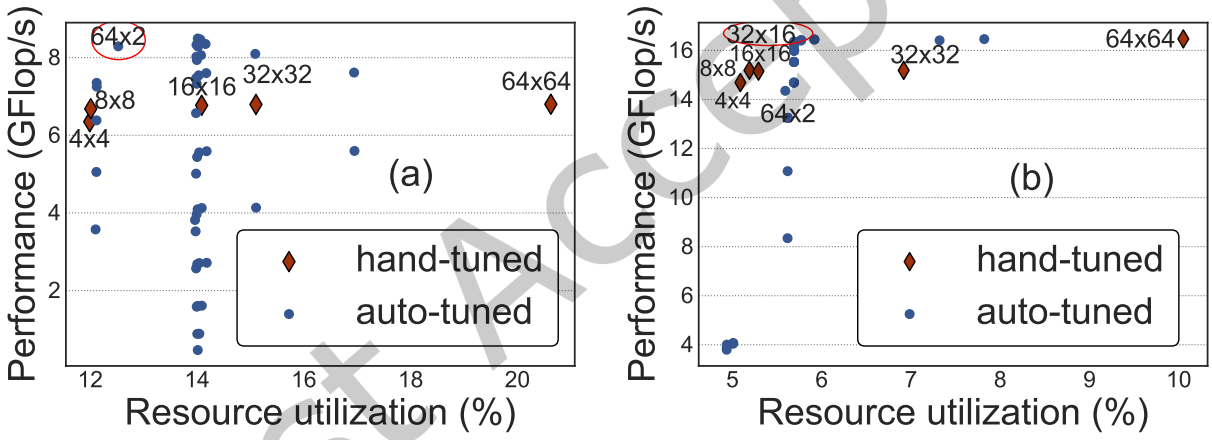


Fig. 7. Performance and FPGA resource utilization of single `vadvc` PE, as a function of tile-size, using hand-tuning and auto-tuning for (a) single-precision (32-bit) and (b) half-precision (16-bit). We highlight the Pareto-optimal solution that we use for our `vadvc` accelerator (with a red circle). 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. However, as shown in Figure 7, such hand-tuned implementations lead to sub-optimal results in terms of either resource utilization or performance.

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

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.

### 4.3 Performance Analysis

Figure 8 shows single-precision performance results for the (a) vertical advection (vadvc) and (b) horizontal diffusion kernels (hdiff). For both kernels, we implement our design on an HBM- and a DDR4-based FPGA board. For the DDR4-based design, we use CAPI2 (DDR4+CAPI2 in Figure 8. For the HBM-based design, we use CAPI2 (HBM+CAPI2) and OCAPI. We evaluate two versions of the HBM-based design with OCAPI: (1) one with a single channel per PE (HBM+OCAPI), and (2) one with multiple channels (i.e., 4 HBM pseudo channels) per PE (HBM_multi+OCAPI). To compare the performance of these four versions, we scale the number of PEs and analyze the change in execution time. We also tested different domain sizes, varying from $64 \times 64 \times 64$-point to $1024 \times 1024 \times 64$-point and observe that the runtime scales linearly and the overall performance (GLOP/s) remain constant. This shows the scalability of our accelerator design.

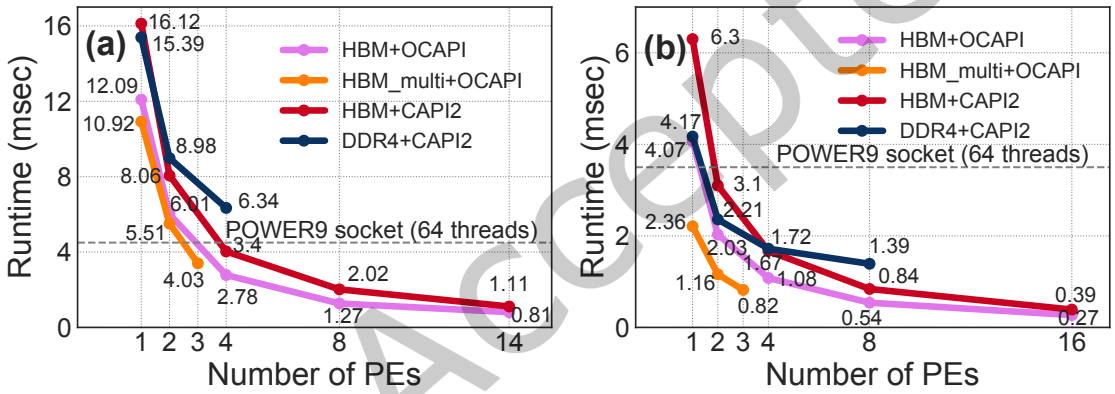


Fig. 8. Single-precision performance for (a) vadvc and (b) hdiff, as a function of accelerator PE count on the HBM- and DDR4-based FPGA boards. We also show the single socket (64 threads) performance of an IBM POWER9 host system for both vadvc and hdiff. For HBM-based design, we implement our accelerator with both the CAPI2 interface and the state-of-the-art OpenCAPI (OCAPI) interface (with both single channel and multiple channels per PE).

We draw five observations from the figure.

First, the maximum number of PEs that we can fit on the FPGA boards varies for different versions of our design. For the DDR4-based design, we can accommodate only 4 PEs/8 PEs for vadvc/hdiff on the 9V3 board. For the HBM-based design, we can fit 14 PEs/16 PEs for vadvc/hdiff for both HBM+CAPI2 and HBM+OCAPI versions before exhausting the on-board resources. The HBM_multi+OCAPI version can only fit 3 PEs (i.e., 12 HBM channels) for both vadvc and hdiff because adding more HBM channels leads to timing constraint violations.

Second, the full-blown HBM+OCAPI versions (i.e., with the maximum number of PEs) of vadvc and hdiff outperform the 64-thread IBM POWER9 CPU version by 5.3×, and 12.7×, respectively. We achieve 37% and 44% higher performance for vadvc and hdiff, respectively, with HBM+OCAPI than HBM+CAPI2 due to the following two reasons: (1) OCAPI provides double the bitwidth (1024-bit) of the CAPI2 interface (512-bit), which provides a higher bandwidth to the host CPU, i.e., 22.1/22.0 GB/s R/W versus 13.9/14.0 GB/s; and (2) with OCAPI, memory coherency logic is moved onto the IBM POWER CPU, which provides more FPGA area and allows us to run our accelerator logic at a higher clock frequency (250MHz for OCAPI versus 200MHz for CAPI2). We observe that

when implementing our accelerator designs with targets above those frequencies, the respective timing report results in the worst negative slack (WNS) higher than 200ps, which OC-Accel developers regard as dangerous for system stability. At lower frequencies, we achieved lower performance, regardless of the number of PEs. Our single-precision HBM+OCAPI-based FPGA implementations provide 157.1 GFLOP/s and 608.4 GFLOP/s for vadvc and hdiff, respectively. For half-precision, if we use the same amount of PEs as in single precision, our accelerator reaches a performance of 329.9 GFLOP/s for vadvc (2.1× the single-precision performance) and 1.5 TFLOP/s for hdiff (2.5× the single-precision performance).

Third, for a single PE, DDR4-CAPI2 is faster than HBM-CAPI2 for both vadvc and hdiff. This higher performance is because the HBM-based design uses one HBM channel per PE, and the bus width of the DDR4 channel (512 bits) is larger than that of an HBM channel (256 bits). Therefore, the HBM channel has a lower transfer rate of 0.8-2.1 GT/s (Gigatransfers per second) than a DDR4 channel (2.1-4.3 GT/s), resulting in a theoretical bandwidth of 12.8 GB/s and 25.6 GB/s per channel, respectively. One way to match the DDR4 bus width is to have a single PE fetch data from multiple HBM channels in parallel. In Figure 8, our multi-channel setting (HBM_multi+OCAPI) uses 4 HBM pseudo channels per PE to match the bitwidth of the OCAPI interface. We observe that by fetching more data from multiple channels, compared to the single-channel-single PE design (HBM+OCAPI), HBM_multi+OCAPI achieves 1.2× and 1.8× performance improvement for vadvc and hdiff, respectively.

Fourth, as we increase the number of PEs, we divide the workload evenly across PEs. As a result, we observe linear scaling in the performance of HBM-based designs, where each PE reads and writes through a dedicated HBM channel. For multi-channel designs, we observe that the best-performing multi-channel-single PE design (i.e., using 3 PEs with 12 HBM channels for both workloads) has 4.7× and 3.1× lower performance than the best-performing single-channel-single PE design (i.e., 14 PEs for vadvc and 16 PEs for hdiff, respectively). This observation shows that there is a tradeoff between (1) enabling more HBM pseudo channels to provide each PE with more bandwidth, and (2) implementing more PEs in the available area. For both vadvc and hdiff, data transfer and computation take a comparable amount of time. Therefore, we are able to achieve a linear execution time reduction with the number of PEs.

Fifth, the performance of the DDR-based designs scales non-linearly for vadvc and hdiff with the number of PEs, as all PEs access memory through the same channel. Multiple PEs compete for a single memory channel, which causes frequent memory stalls due to contention in the memory channel.

### 4.4 Energy Efficiency Analysis

We compare the energy consumption of our accelerator to a 16-core POWER9 host 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. For vadvc and hdiff on the HBM- and DDR4-based designs, Figure 9 and Figure 10 shows the active power consumption and the energy efficiency (GFLOPS per Watt), respectively.

We make five observations from Figure 9 and Figure 10.

First, the full-blown HBM+OCAPI designs (i.e., 14 PEs for vadvc and 16 PEs for hdiff) achieve energy efficiency values of 1.61 GFLOPS/Watt and 21.01 GFLOPS/Watt for vadvc and hdiff, respectively. These represent improvements of 12× and 35× compared to the IBM POWER9 system for vadvc and hdiff, respectively.

Second, the DDR4-CAPI2 designs for vadvc and hdiff are slightly more energy-efficient (1.1× to 1.5×) than the HBM-CAPI2 designs when the number of PEs is small. This observation is in line with our discussion about

[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.

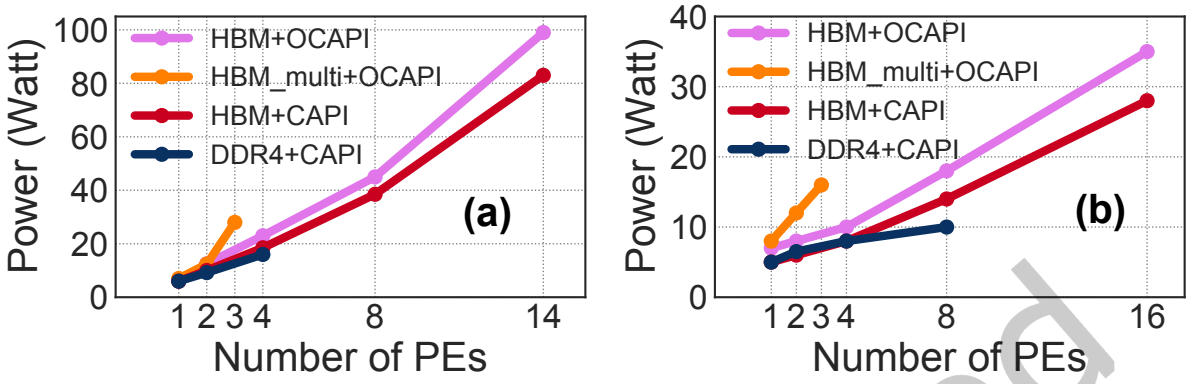


Fig. 9. Active Power Consumption for (a) vadvc and (b) hdiff on HBM- and DDR4-based FPGA boards. For HBM-based design, we implement our accelerator with both the CAPI2 interface and the state-of-the-art OpenCAPI (OCAPI) interface (with both single channel and multiple channels per PE).

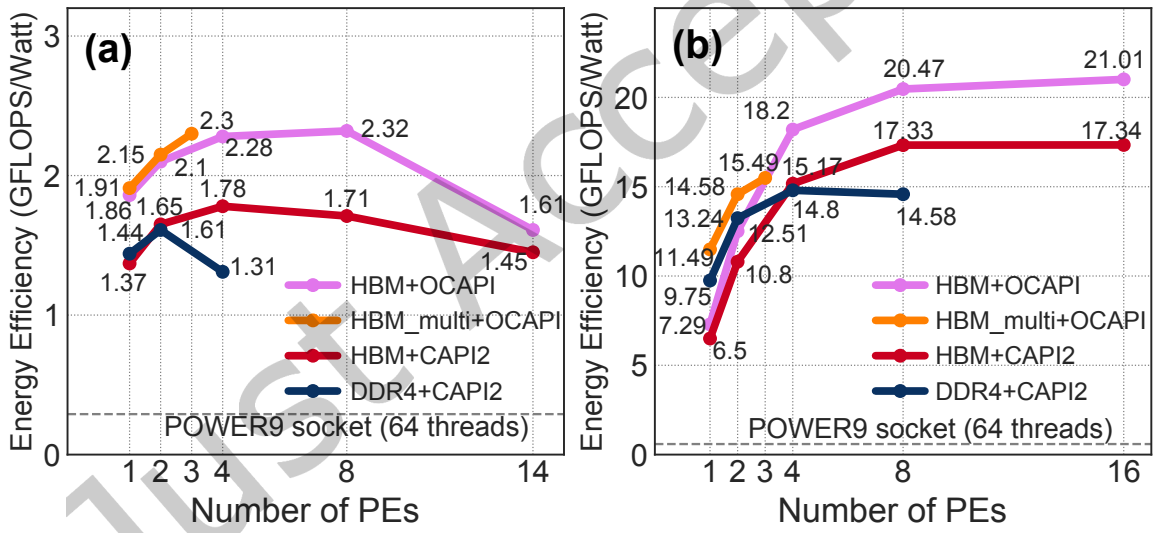


Fig. 10. Energy efficiency for (a) vadvc and (b) hdiff on HBM- and DDR4-based FPGA boards. We also show the single socket (64 threads) energy efficiency of an IBM POWER9 host system for both vadvc and hdiff. For HBM-based design, we implement our accelerator with both the CAPI2 interface and the state-of-the-art OpenCAPI (OCAPI) interface (with both single channel and multiple channels per PE).

performance with small PE counts in Section 4.3. However, as we increase the number of PEs, the HBM-CAPI2 designs provide higher energy efficiency since they make use of multiple HBM channels.

Third, the energy efficiency of the HBM-based designs (HBM+CAPI2, HBM+OCAPI) for hdiff increases with the number of PEs until a saturation point (8 PEs). This trend is because every additional HBM channel increases

power consumption by ~1 Watt (for the HBM AXI3 interface operating at 250MHz with a logic toggle rate of ~12.5%).

Fourth, `HBM+OCAPI`, `HBM+CAPI2`, and `DDR4+CAPI2` versions of `vadvc` achieve their highest energy efficiency at a number of PEs that is smaller than the maximum possible. There is a large amount of control flow in `vadvc`, which leads to large resource utilization. As a result, as shown in Figure 9, increasing the PE count increases power consumption dramatically, causing lower energy efficiency.

Fifth, the multi-channel-single PE designs (`HBM_multi+OCAPI`) are more energy-efficient than the single-channel-single PE designs (`HBM+OCAPI`) for the same number of PEs. However, `HBM+OCAPI` designs achieve higher energy efficiency for higher numbers of PEs, which are not affordable for `HBM_multi+OCAPI` designs.

### 4.5 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 up to 16 memory channels.

## 5 Discussion and Key Takeaways

A wide range of application domains have emerged with the ubiquity of computing platforms in every aspect of our daily lives. These modern workloads (e.g., machine learning, graph processing, and bioinformatics) demand high compute capabilities within strict power constraints [71]. However, today's computing systems are getting constrained by current technological capabilities, making them incapable of delivering the required performance. This paper presents our recent efforts to leverage near-memory computing capable FPGA-based accelerators to accelerate two major kernels from the weather prediction application in an energy-efficient way. We summarize the most important insights and takeaways as follows.

First, our evaluation shows that High-Bandwidth Memory-based near-memory FPGA accelerator designs can improve performance by 5.3×-12.7× and energy efficiency by 12×-35× over a single-socket high-end 16-core IBM POWER9 CPU.

Second, our HBM-based FPGA accelerator designs employ a dedicated HBM channel per PE. Such a design avoids memory access congestion, which is typical in DDR4-based FPGA designs, and ensures memory bandwidth scaling with the number of PEs. As a result, in most of the data-parallel applications, performance scales linearly with the number of PEs. Therefore, HBM provides an attractive solution for scale-out computation.

Third, the data needs to be adequately mapped to each HBM channel's address space. A data mapping scheme should map data in such a way that the data required by the processing unit is readily available in the vicinity (data and code co-location). An inefficient data mapping mechanism can severely hamper the benefits of processing close to memory.

Fourth, we make use of OCAPI in a coarse-grained way, since we offload the entire application to the FPGA. In this case, OCAPI ensures that the FPGA accelerators access the entire CPU memory with the minimum number of memory copies between the host and the FPGA, e.g., avoiding the intermediate buffer copies that a traditional PCIe-based DMA invokes [52]. However, depending on the application, the CAPI protocol can be employed in finer-grained algorithm-hardware co-design, like the *ExtraV* [106], where the authors aggressively utilize the fine-grained communication capability of OCAPI to boost graph analytics performance.

Fifth, the maximum performance of our HBM-based design is reached using the maximum PE count that we can fit in the reconfigurable fabric, with each PE having a dedicated HBM channel. However, adding more PEs could lead to timing constraint violations for HBM-based designs. As shown with our multi-channel setting (Section 4.3), where we can fit only 3 PEs for both `vadvc` and `hdiff`, enabling more HBM channels leads to timing constraint violations. HBM-based FPGAs consist of multiple super-logic regions (SLRs) [18], where an SLR represents a single FPGA die. All HBM channels are connected only to SLR0, while other SLRs have indirect connections to the HBM channels. Therefore, if a PE is implemented in a non-SLR0 region for a large design, it might make timing closure difficult. A possible way to alleviate timing issues is by running the AFU at a lower frequency, which eases the place and route.

Sixth, the energy efficiency of our HBM-based designs tends to saturate (or even reduces) as we increase the number of PEs beyond some point. The highest energy efficiency is achieved with a PE count that is smaller than the highest-performing PE count. The major reason for a decrease in the energy efficiency is the increase in power consumption with every additional HBM channel.

Seventh, the emerging cache-coherent interconnects standards like CXL [145], CCIX [39], and OCAPI [156] could be vital in improving the performance and energy efficiency of big data workloads running on FPGA-based devices because they avoid having multiple data copies. However, a very small number of works, such as [106], leverage the *coherency* aspect of these interconnects. More quantitative exploration is required to analyze the advantages and disadvantages of using these interconnects.

Eighth, we are witnessing an enormous amount of data being generated across multiple application domains [123, 151] like weather prediction modeling, radio astronomy, bioinformatics, material science, chemistry, health sciences, etc. The processing of the sheer amount of generated data is one of the biggest challenges to overcome. In this paper, we demonstrate the capabilities of near-memory reconfigurable accelerators in the domain of weather prediction, however, there are many other high-performance computing applications where such near-memory architectures can alleviate the data movement bottleneck.

## 6 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 real-world weather modeling stencils. We exploit the near-memory capabilities of such FPGAs to accelerate important weather prediction kernels. Exploiting the high-bandwidth memory in FPGAs, we answer the following questions with our work. First, do real-world weather prediction applications benefit from HBM-based FPGAs? Second, how can we scale the processing in terms of not only run-time but also energy efficiency? Third, what does the system look like regarding computation and data movement with an HBM-enabled FPGA when integrating accelerators for real-world weather prediction workloads?

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 [43, 44, 71, 118–121]. This frequent data movement has a severe impact on overall system performance and energy efficiency. For example, in the domain of climate and weather modeling, there is a data avalanche due to large atmospheric simulations [137]. Major efforts are currently underway towards refining the resolution grid of climate models that would generate *zettabytes* of data [137]. These high-resolution simulations are useful to predict and address events like severe storms. However, the sheer amount of generated data is one of the biggest challenges to overcome. We find another relevant example in radio astronomy. The first phase of the Square Kilometre Array (SKA) aims to process over 100 terabytes of raw data samples per second, yielding of the order of 300 petabytes of SKA data produced annually [91, 149]. Recent biological disciplines such as genomics have also emerged as one of the most data-intensive workloads across all different sciences wherein just a single human genome sequence produces

hundreds of gigabytes of raw data. With the rapid advancement in sequencing technology, the data volume in genomics is projected to surpass the data volume in all other application domains [124].

A way to alleviate this *data movement bottleneck* [43, 44, 71, 80, 118–121, 121, 144, 150] 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 [7, 43, 99, 102, 104, 105, 108, 118, 121, 130], and also by cache-coherent interconnects [39, 145, 156], which allow close integration of processing units and memory units. Depending on the applications and systems of interest (e.g., [24–26, 33, 36, 37, 40, 43, 44, 46, 47, 50, 63, 64, 68, 69, 73, 75, 76, 81, 82, 86, 93, 96, 98, 107, 109, 111, 113, 117, 118, 121, 121, 122, 126, 128, 138, 140–143, 146, 158, 174]), prior works propose different types of near-memory processing units, such as general-purpose CPU cores [24, 27, 44–46, 60, 75, 78, 101, 109, 112, 123, 126, 132, 136], GPU cores [72, 85, 129, 176], reconfigurable units [70, 89, 92, 153], or fixed-function units [25, 43, 78, 81, 82, 86, 100, 112, 122].

FPGA accelerators are promising to enhance overall system performance with low power consumption. Past works [28–32, 49, 57, 74, 87, 90, 92, 94, 106] show that FPGAs can be employed effectively for a wide range of applications. FPGAs provide a unique combination of flexibility and performance without the cost, complexity, and risk of developing custom application-specific integrated circuits (ASICs). The researchers at CERN, for example, are using FPGAs to accelerate physics workload in CERN's exploration of dark matter [61]. Microsoft's Project Catapult [48] is another example of how FPGAs can be used in the data center infrastructure. Driven by Catapult's promising research results, Microsoft further deployed the architecture on the Azure cloud marketplace [116]. Such integration for certain workloads can even offer more energy efficiency than CPU or GPU-based systems. 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 [7, 104] has been explored in many-core processors [72, 131] and GPUs [72, 178]. Recent benchmarking works [95, 171] show the potential of HBM for FPGAs.

NERO is the first work to accelerate a real-world HPC weather prediction application using the FPGA+HBM fabric. Compared to a previous work [153] 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. Our current work differs from [152] in the following aspects. First, we design and evaluate both horizontal diffusion and vertical advection stencils. Vertical advection is the most complex stencil in the entire COSMO application. Second, we integrate and implement our accelerator design with an HBM-based FPGA. The bus width of the DDR4 channel (512 bits) is larger than that of an HBM channel (256 bits). Therefore, the HBM channel has a lower transfer rate of 0.8-2.1 GT/s (Gigatransfers per second) than a DDR4 channel (2.1-4.3 GT/s), resulting in a theoretical bandwidth of 12.8 GB/s and 25.6 GB/s per channel, respectively. However, HBM exposes 32 memory channels that provide 4x more bandwidth (410 GB/s for HBM [95]) compared to traditional DDR4 bandwidth (72 GB/s for four independent dual-rank DIMM interfaces [21]). Therefore, the use of HBM imposes an architectural shift. We evaluate and demonstrate the use of HBM for scaling an accelerator design with different channels provided by HBM. Third, we use an auto-tuning framework to find the right window size (Figure 6) that demonstrates the importance of finding the right window size. Fourth, we provide new results using a state-of-the-art OpenCAPI (OCAPI) interface with the OC-Accel framework. OCAPI provides two key opportunities compared to our previous CAPI2 implementation: (1) OCAPI has double the bitwidth of our previously used CAPI2 interface, (2) a major component of the memory coherency logic is moved to the host CPU side, which provides more FPGA area and enables designs with higher frequency. Due to the above optimizations, we improve the performance for horizontal diffusion by 1.2x on a DDR4-based board and 4.7x on an HBM-based board compared to our previous work NARMADA [153].

Enabling higher performance for stencil computations has been a subject of optimizations across the whole computing stack [35, 51, 53–55, 66, 77, 79, 83, 115, 135, 155, 160, 168, 169, 179]. Stencil computation is essential

for numerical simulations of finite difference methods (FDM) [127] and is applied in iterative solvers of linear equation systems. We use stencil computation in a wide range of applications, including computational fluid dynamics [88], image processing [84], weather prediction modeling [59], etc.

Unlike stencils found in the literature [51, 55, 56, 135, 154, 168, 169], real-world compound stencils consist of a collection of stencils that perform a sequence of element-wise computations with complex interdependencies. Such compound kernels have complex memory access patterns and low arithmetic intensity because they have limited operations per loaded value. Our work is the first work to accelerate both horizontal diffusion and vertical advection stencils, which are representative of data access patterns and the algorithmic complexity found in the entire COSMO weather model.

Table 4 lists recent works (including NERO) that use FPGA to accelerate stencil-based application. We also mention works that accelerate elementary stencils (`7-point`, `25-point` Jacobi, `Hotspot`, and `Diffusion`). We make the following three observations. First, the elementary stencils can achieve much higher performance on comparable FPGA devices than complex weather stencils (such as `hdiff`) even without using HBM. This high performance is because elementary stencils have a higher arithmetic intensity than weather stencils. Due to their data-parallel nature, these elementary stencils can further take advantage of the increased bandwidth provided by HBM in an energy-efficient way. Second, weather stencils can reach only 2%-17% of the peak theoretical performance of an FPGA board. This low peak performance is because weather stencils have several elementary stencils cascaded together with data interdependencies that lead to complex memory access patterns. Third, compared to NARMADA [153], which uses a DDR4-based design, our HBM-based design achieves 4.7× performance improvement by exploiting the high bandwidth provided by the HBM.

Szustak *et al.* accelerate the MPDATA advection scheme on multi-core CPU [159] and computational fluid dynamics kernels on FPGA [133]. Singh *et al.* [154] explore the applicability of different number formats and exhaustively search for the appropriate bit-width for memory-bound stencil kernels to improve performance and energy efficiency with minimal loss in the accuracy. Bianco *et al.* [41] optimize the COSMO weather prediction model for GPUs. Thaler *et al.* [161], in a collaboration work between the Swiss National Supercomputing Centre (CSCS) and the Federal Institute of Meteorology and Climatology (MeteoSwiss), discuss the importance of horizontal diffusion and vertical advection kernels in the entire COSMO model. These kernels together represent the algorithmic diversity of the entire COSMO weather prediction model [41, 79, 161]. They port COSMO to a many-core system. Compared to their Intel KNL [11] (or NVIDIA P100 [12]) implementation, we observe that our FPGA-based `vadvc` and `hdiff` design provides 1.5× (or 1.4×) and 3.2× (or 2.1×) performance improvements, respectively. Several works [56, 103, 110, 170] propose frameworks for generating optimized stencil code for FPGA-based platforms. Wahib *et al.* [167] develop an analytical performance model for choosing an optimal GPU-based execution strategy for various scientific applications, including COSMO. Gysi *et al.* [79] provide guidelines for optimizing stencil kernels for CPU–GPU systems.

## 7 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. We use a heterogeneous system comprising of IBM POWER9 CPU with field-programmable gate array (FPGA) as our target platform. We create a heterogeneous domain-specific memory hierarchy using on-chip URAMs and BRAMs, and on-package HBM on an FPGA. Unlike conventional 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 both energy and latency by tailoring the cache levels and cache sizes to an application's memory access patterns.

NERO's implementations of `vadvc` and `hdiff` outperform the optimized software implementations on a 16-core POWER9 with 4-way multithreading by 5.3× and 12.7×, with 12× and 35× 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 support with the 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 and OC-Accel 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 to the SAFARI Research Group.

**Algorithm 1:** Pseudo-code for vertical advection and horizontal diffusion kernels used by the COSMO [59] weather prediction model.

```
1  Function verticalAdvection(float* ccol, float* dcol, float* wcon, float* ustage, float* upos, float* utens,
     float* utensstage)
2      for c ← 2 to column − 2 do
3          for r ← 2 to row-2 do
4              Function forwardSweep(float* ccol, float* dcol, float* wcon, float* ustage, float* upos, float*
               utens, float* utensstage)
5                  for d ← 1 to depth do
                       /* forward sweep calculation */
6                  end
7              end
8              Function backwardSweep(float* ccol, float* dcol, float* wcon, float* ustage, float* upos, float*
               utens, float* utensstage)
9                  for d ← depth − 1 to 1 do
                       /* backward sweep calculation */
10                 end
11             end
12         end
13     end
14 end
15 Function horizontalDiffusion(float* src, float* dst)
16     for d ← 1 to depth do
17         for c ← 2 to column − 2 do
18             for r ← 2 to row-2 do
                   /* Laplacian calculation */
19                 lap_CR = laplaceCalculate(c, r) /* row-laplacian */
20                 lap_CRm = laplaceCalculate(c, r − 1)
21                 lap_CRp = laplaceCalculate(c, r + 1) /* column-laplacian */
22                 lap_CmR = laplaceCalculate(c − 1, r)
23                 lap_CpR = laplaceCalculate(c + 1, r) /* column-flux calculation */
24                 flux_C = lap_CpR − lap_CR
25                 flux_Cm = lap_CR − lap_CmR
                   /* row-flux calculation */
26                 flux_R = lap_CRp − lap_CR
27                 flux_Rm = lap_CR − lap_CmR
                   /* output calculation */
28                 dest[d][c][r] = src[d][c][r] − c1 ∗ (flux_CR − flux_CmR) + (flux_CR − flux_CRm)
29             end
30         end
31     end
32 end
```

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

| | |
|---|---|
| **Host CPU** | 16-core IBM POWER9 AC922 [134] @3.2 GHz, 4-way SMT [164] |
| **Cache-Hierarchy** | 16×32 KiB L1-I/D, 256 KiB L2, 10 MiB L3 |
| **System Memory** | 32GiB RDIMM DDR4 2666 MHz [15] |
| **HBM-based FPGA Board** | Alpha Data ADM-PCIE-9H7 [1] Xilinx Virtex Ultrascale+ XCVU37P-2 [19] 8GiB (HBM2 [7]) with PCIe Gen4 x8 [114] |
| **DDR4-based FPGA Board** | Alpha Data ADM-PCIE-9V3 [2] Xilinx Virtex Ultrascale+ XCVU3P-2 [19] 8GiB (DDR4) with PCIe Gen4 x8 [114] |
| **OS details** | Ubuntu 20.04.3 LTS [16], GNU Compiler Collection (GCC) version 9.3.0 [6], IBM XL C/C++ 16 [9] |

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

| Algorithm | BRAM | DSP | FF | LUT | URAM |
|---|---|---|---|---|---|
| vadvc | 94% | 39% | 37% | 55% | 53% |
| hdiff | 96% | 4% | 10% | 15% | 8% |

Table 3. Prediction results for different CAPI 2.0 enabled boards across FPGA families.

| 2*FPGA Board | 2*FPGA Device | Utilization% | | | | | 2*Window | 2*AFU | 2*Performance (GFLOP/s) | 2*Energy Impr. |
|---|---|---|---|---|---|---|---|---|---|---|
| | | BRAM | DSP | FF | LUT | URAM | | | | |
| ADM-PCIE-9V3* | XCVU3P-2 | 71 | 66 | 49 | 79 | 58 | $8 \times 8 \times 8$ | 16 | 120.1 | 18.1× |
| ADM-PCIE-KU3* | XCKU3P-2 | 70 | 33 | 61 | 96 | 70 | $16 \times 16 \times 16$ | 4 | 48.9 | 9.3× |
| Semptian NSA121B | XCKU060 | 55 | 49 | 73 | 79 | - | $8 \times 8 \times 8$ | 8 | 83.5 | 20.6× |
| ADM-PCIE-8K5 | XCKU115 | 59 | 58 | 81 | 77 | - | $8 \times 8 \times 8$ | 16 | 162.7 | 19.2× |

[para,flushleft]
*URAM memory is available only in Ultrascale+ families

Table 4. Overview of the state-of-art stencil implementations on FPGAs. For each work, we mention the technology node (Tech. node), DRAM memory technology (Mem. Tech.), theoretical peak floating-point performance (Peak Perf. (TFLOPS)), available peak memory bandwidth (Peak B/W (GB/s)), frequency of the accelerator logic (Freq. (MHz)), overall logic utilization (Logic Util.), overall memory utilization (Mem. Util.), achieved performance (Perf. (GOp/s)), and the percentage of achieved peak roofline performance (Ach. Roof.).

| Stencil | Work | Year | Device | Tech. node | Mem. Tech. | Peak Perf. (TFLOPS) | Peak B/W (GB/s) | Freq. (MHz) | Logic Util. | Mem. Util. | Perf. (GOp/s) | Ach. Roof. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Diffusion 3D | [168] | 2019 | Arria 10 | TSMC 20nm | DDR3 | 1.4 | 34 | 276 | 32% | 47% | 628.0 | 44.9% |
| Hotspot 3D | [168] | 2019 | Arria 10 | TSMC 20nm | DDR3 | 1.4 | 34 | 240 | 34% | 81% | 630 | 45.0% |
| 7-point 3D | [154] | 2019 | XCVU3P | TSMC 16FF+ | DDR4 | 0.97 | 25.6 | 180 | 23.5% | 39% | 228.4 | 23.7% |
| 25-point 3D | [154] | 2019 | XCVU3P | TSMC 16FF+ | DDR4 | 0.97 | 25.6 | 190 | 49% | 39% | 327.7 | 34.1% |
| 3D Jacobi | [56] | 2021 | Stratix 10 | Intel 14nm FinFet | DDR4 | 9.2 | 76.8 | 292-317 | - | - | 568.2 | 6.2% |
| hdiff | [153] | 2019 | XCVU3P | TSMC 16FF+ | DDR4 | 0.97 | 25.6 | 200 | 64.5% | 64.1% | 129.9 | 13.3% |
| hdiff | [56] | 2021 | Stratix 10 | Intel 14nm FinFet | DDR4 | 9.2 | 76.8 | 292-317 | 26.0% | 20% | 145.0 [513.0†] | 1.6% [5.5%] |
| hdiff | [Ours] | 2021 | XCVU37P | TSMC 16FF+ | HBM | 3.6 | 204.8§ [410] | 250 | 12.5% | 52% | 608.4 | 16.9% |

† When simulated using an infinite memory bandwidth. § Note that we use only a single HBM stack due to resource limitations.

## References


[1] [n.d.]. *ADM-PCIE-9H7-High-Speed Communications Hub, https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9h7*.
[2] [n.d.]. *ADM-PCIE-9V3-High-Performance Network Accelerator, https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9v3*.
[3] [n.d.]. *AXI High Bandwidth Memory Controller v1.0, https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf*.
[4] [n.d.]. *AXI Reference Guide, https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf*.
[5] [n.d.]. *CentOS-7 (2009) Release Notes, https://wiki.centos.org/Manuals/ReleaseNotes/CentOS7.2009* .
[6] [n.d.]. *GCC, the GNU Compiler Collection, https://gcc.gnu.org/* .
[7] [n.d.]. *High Bandwidth Memory (HBM) DRAM (JESD235), https://www.jedec.org/document_search?search_api_views_fulltext=jesd235*.
[8] [n.d.]. *High Bandwidth Memory (HBM) DRAM, https://www.jedec.org/sites/default/files/JESD235B-HBM_Ballout.zip*.
[9] [n.d.]. *IBM XL C/C++ for Linux, https://www.ibm.com/products/xl-cpp-linux-compiler-power*.
[10] [n.d.]. *Intel Stratix 10 MX FPGAs, https://www.intel.com/content/www/us/en/products/programmable/sip/stratix-10-mx.html*.
[11] [n.d.]. *IntelÂő Xeon PhiâĎć Processor 7230 (16GB, 1.30 GHz, 64 core), https://www.intel.com/content/www/us/en/products/sku/94034/intel-xeon-phi-processor-7230-16gb-1-30-ghz-64-core/specifications.html*.
[12] [n.d.]. *NVIDIAÂő TESLAÂő P100 GPU ACCELERATOR, https://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-PCIe-datasheet.pdf*.
[13] [n.d.]. *OC-Accel, https://opencapi.github.io/oc-accel-doc/* .
[14] [n.d.]. *OpenPOWER Work Groups, https://openpowerfoundation.org/technical/working-groups*.
[15] [n.d.]. *RDIMM, https://www.micron.com/products/dram-modules/rdimm*.
[16] [n.d.]. *Ubuntu 20.04.3 LTS (Focal Fossa), https://releases.ubuntu.com/20.04/*.
[17] [n.d.]. *UltraScale Architecture Memory Resources, https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascale-memory-resources.pdf*.
[18] [n.d.]. *Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance, https://www.xilinx.com/support/documentation/white_papers/wp485-hbm.pdf*.
[19] [n.d.]. *Virtex UltraScale+, https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html*.
[20] [n.d.]. *Vivado High-Level Synthesis, https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html*.
[21] [n.d.]. *Xilinx VCU1525, https://www.xilinx.com/products/boards-and-kits/vcu1525-a.html*.
[22] [n.d.]. *Xilinx Virtex UltraScale+, https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html*.
[23] [n.d.]. *Xilinx Vivado, https://www.xilinx.com/support/download.html*.
[24] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In *ISCA*.
[25] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture. In *ISCA*.
[26] Berkin Akin, Franz Franchetti, and James C Hoe. 2015. Data Reorganization in Memory Using 3D-stacked DRAM. In *ISCA*.
[27] M. Alian, S. W. Min, H. Asgharimoghaddam, A. Dhar, D. K. Wang, T. Roewer, A. McPadden, O. O'Halloran, D. Chen, J. Xiong, D. Kim, W. Hwu, and N. S. Kim. 2018. Application-Transparent Near-Memory Processing Architecture with Memory Channel Network. In *MICRO*.
[28] Mohammed Alser, Zülal Bingöl, Damla Senol Cali, Jeremie Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. 2020. Accelerating Genome Analysis: A Primer on an Ongoing Journey. In *IEEE Micro*.
[29] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. 2019. Shouji: A Fast and Efficient Pre-Alignment Filter for Sequence Alignment. In *Bioinformatics*.
[30] Mohammed Alser, Hasan Hassan, Hongyi Xin, OÄ§uz Ergin, Onur Mutlu, and Can Alkan. 2017. GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping. In *Bioinformatics*.
[31] Mohammed Alser, Jeremy Rotman, Kodi Taraszka, Huwenbo Shi, Pelin Icer Baykal, Harry Taegyun Yang, Victor Xue, Sergey Knyazev, Benjamin D Singer, Brunilda Balliu, et al. 2020. Technology Dictates Algorithms: Recent Developments in Read Alignment. In *Genome Biology*.
[32] Mohammed Alser, Taha Shahroodi, Juan Gomez-Luna, Can Alkan, and Onur Mutlu. 2020. SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs. In *Bioinformatics*.
[33] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. 2019. AlignS: A Processing-In-Memory Accelerator for DNA Short Read Alignment Leveraging SOT-MRAM. In *DAC*.
[34] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. 2014. OpenTuner: An Extensible Framework for Program Autotuning. In *PACT*.
[35] Adrià Armejach, Helena Caminal, Juan M Cebrian, Rekai González-Alberquilla, Chris Adeniyi-Jones, Mateo Valero, Marc Casas, and Miquel Moretó. 2018. Stencil Codes on a Vector Length Agnostic Architecture. In *PACT*.

[36] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems. In *MICRO*.
[37] Oreoluwatomiwa O Babarinsa and Stratos Idreos. 2015. JAFAR: Near-Data Processing for Databases. In *SIGMOD*.
[38] George H Barnes, Richard M Brown, Maso Kato, David J Kuck, Daniel L Slotnick, and Richard A Stokes. 1968. The ILLIAC IV Computer. In *TC*.
[39] Brad Benton. 2017. CCIX, Gen-Z, OpenCAPI: Overview and Comparison. In *OFA*.
[40] Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, et al. 2021. SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems. In *MICRO*.
[41] M Bianco, T Diamanti, O Fuhrer, T Gysi, X Lapillonne, C Osuna, and T Schulthess. 2013. A GPU Capable Version of the COSMO Weather Model. In *ISC*.
[42] Luca Bonaventura. 2000. A Semi-implicit Semi-Lagrangian Scheme using the Height Coordinate for a Nonhydrostatic and Fully Elastic Model of Atmospheric Flows. In *JCP*.
[43] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. 2021. Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks. In *PACT*.
[44] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. 2018. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *ASPLOS*.
[45] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T Malladi, Hongzhong Zheng, et al. 2019. CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators. In *ISCA*.
[46] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T Malladi, Hongzhong Zheng, and Onur Mutlu. 2016. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. In *CAL*.
[47] Damla Senol Cali, Gurpreet S. Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gómez Luna, Amirali Boroumand, Anant Nori, Allison Scibisz, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu. 2020. GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis. In *MICRO*.
[48] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger. 2016. A Cloud-Scale Acceleration Architecture. In *MICRO*.
[49] Li-Wen Chang, Juan Gómez-Luna, Izzat El Hajj, Sitao Huang, Deming Chen, and Wen-mei Hwu. 2017. Collaborative Computing for Heterogeneous Integrated Systems. In *ICPE*.
[50] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In *ISCA*.
[51] Yuze Chi, Jason Cong, Peng Wei, and Peipei Zhou. 2018. SODA: Stencil with Optimized Dataflow Architecture. In *ICCAD*.
[52] Young-kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. 2016. A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms. In *DAC*.
[53] Matthias Christen, Olaf Schenk, and Helmar Burkhart. 2011. PATUS: A Code Generation and Autotuning Framework for Parallel Iterative Stencil Computations on Modern Microarchitectures. In *IPDPS*.
[54] Kaushik Datta, Shoaib Kamil, Samuel Williams, Leonid Oliker, John Shalf, and Katherine Yelick. 2009. Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors. In *SIAM review*.
[55] Johannes de Fine Licht, Michaela Blott, and Torsten Hoefler. 2018. Designing scalable FPGA architectures using high-level synthesis. In *PPoPP*.
[56] Johannes de Fine Licht, Andreas Kuster, Tiziano De Matteis, Tal Ben-Nun, Dominic Hofer, and Torsten Hoefler. 2021. StencilFlow: Mapping large stencil programs to distributed spatial computing systems. In *CGO*.
[57] Dionysios Diamantopoulos, Heiner Giefers, and Christoph Hagleitner. 2018. ecTALK: Energy Efficient Coherent Transprecision Accelerators âĂŞ The Bidirectional Long Short-Term Memory Neural Network Case . In *COOL CHIPS*.
[58] Dionysios Diamantopoulos and Christoph Hagleitner. 2018. A System-Level Transprecision FPGA Accelerator for BLSTM Using On-chip Memory Reshaping. In *FPT*.
[59] G Doms and U Schättler. 1999. The Nonhydrostatic Limited-Area Model LM (Lokal-model) of the DWD. Part I: Scientific Documentation. In *DWD, GB Forschung und Entwicklung*.
[60] Mario Drumond, Alexandros Daglis, Nooshin Mirzadeh, Dmitrii Ustiugov, Javier Picorel, Babak Falsafi, Boris Grot, and Dionisios Pnevmatikatos. 2017. The Mondrian Data Engine. In *ISCA*.
[61] Javier Duarte, Song Han, Philip Harris, Sergo Jindariani, Edward Kreinar, Benjamin Kreis, Jennifer Ngadiuba, Maurizio Pierini, Ryan Rivera, Nhan Tran, and Z Wu. 2018. Fast inference of deep neural networks in FPGAs for pinproceedings physics. In *JINST*.

[62] Jian Fang, Yvo T. B. Mulder, Jan Hidders, Jinho Lee, and H. Peter Hofstee. 2020. In-memory database acceleration on FPGAs: a survey. In *VLDB*.
[63] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim. 2015. NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules. In *HPCA*.
[64] Ivan Fernandez, Ricardo Quislant, Eladio Gutiérrez, Oscar Plata, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, and Onur Mutlu. 2020. NATSA: A Near-Data Processing Accelerator for Time Series Analysis. In *ICCD*.
[65] Michael J Flynn. 1966. Very High-Speed Computing Systems. In *Proceedings of the IEEE*.
[66] Haohuan Fu and Robert G Clapp. 2011. Eliminating the memory bottleneck: an FPGA-based solution for 3D reverse time migration. In *FPGA*.
[67] Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, and Trevor Bauer. 2019. Xilinx Adaptive Compute Acceleration Platform: VersalâĎć Architecture. In *FPGA*.
[68] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. 2019. ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs. In *MICRO*.
[69] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. 2015. Practical Near-Data Processing for In-Memory Analytics Frameworks. In *PACT*.
[70] M. Gao and C. Kozyrakis. 2016. HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing. In *HPCA*.
[71] Saugata Ghose, Amirali Boroumand, Jeremie S Kim, Juan Gómez-Luna, and Onur Mutlu. 2019. Processing-in-memory: A workload-driven perspective. In *IBM JRD*.
[72] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. 2019. Demystifying Complex Workload-DRAM Interactions: An Experimental Study. In *POMACS*.
[73] Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. 2021. SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures. In *HPCA*.
[74] Heiner Giefers, Raphael Polig, and Christoph Hagleitner. 2015. Accelerating arithmetic kernels with coherent attached FPGA coprocessors. In *DATE*.
[75] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. 2021. Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture. In *arXiv*.
[76] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F Oliveira, and Onur Mutlu. 2021. Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-in-Memory Hardware. In *CUT*.
[77] José González and Antonio González. 1997. Speculative Execution via Address Prediction and Data Prefetching. In *ICS*.
[78] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. 2016. Biscuit: A Framework for Near-data Processing of Big Data Workloads. In *ISCA*.
[79] Tobias Gysi, Tobias Grosser, and Torsten Hoefler. 2015. MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures. In *SC*.
[80] Nastaran Hajinazar, Geraldo F Oliveira, Sven Gregorio, João Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez Luna, and Onur Mutlu. 2021. SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM. In *ASPLOS*.
[81] Milad Hashemi, Eiman Ebrahimi, Onur Mutlu, Yale N Patt, et al. 2016. Accelerating Dependent Cache Misses with an Enhanced Memory Controller. In *ISCA*.
[82] Milad Hashemi, Onur Mutlu, and Yale N Patt. 2016. Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads. In *MICRO*.
[83] Tom Henretty, Kevin Stock, Louis-Noël Pouchet, Franz Franchetti, J Ramanujam, and P Sadayappan. 2011. Data Layout Transformation for Stencil Computations on Short-Vector SIMD Architectures. In *CC*.
[84] Txomin Hermosilla, E Bermejo, A Balaguer, and Luis A Ruiz. 2008. Non-linear fourth-order image interpolation for subpixel edge detection and localization. In *IMAVIS*.
[85] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W Keckler. 2016. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. In *ISCA*.
[86] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu. 2016. Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation. In *ICCD*.
[87] Sitao Huang, Li-Wen Chang, Izzat El Hajj, Simon Garcia de Gonzalo, Juan Gómez-Luna, Sai Rahul Chalamalasetti, Mohamed El-Hadedy, Dejan Milojicic, Onur Mutlu, Deming Chen, and Wen-mei Hwu. 2019. Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures. In *ICPE*.

[88] HT Huynh, Zhi J Wang, and Peter E Vincent. 2014. High-order methods for computational fluid dynamics: A brief review of compact differential formulations on unstructured grids. In *Computers & Fluids*.
[89] Zsolt István, David Sidler, and Gustavo Alonso. 2017. Caribou: Intelligent Distributed Storage. In *VLDB*.
[90] Jiantong Jiang, Zeke Wang, Xue Liu, Juan Gómez-Luna, Nan Guan, Qingxu Deng, Wei Zhang, and Onur Mutlu. 2020. Boyi: A Systematic Framework for Automatically Deciding the Right Execution Model of OpenCL Applications on FPGAs. In *FPGA*.
[91] R. Jongerius, S. Wijnholds, R. Nijboer, and H. Corporaal. 2014. An End-to-End Computing Model for the Square Kilometre Array. In *Computer*.
[92] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, et al. 2015. BlueDBM: An Appliance for Big Data Analytics. In *ISCA*.
[93] Yangwook Kang, Yang-suk Kee, Ethan L Miller, and Chanik Park. 2013. Enabling Cost-effective Data Processing with Smart SSD. In *MSST*.
[94] Kaan Kara, Dan Alistarh, Gustavo Alonso, Onur Mutlu, and Ce Zhang. 2017. FPGA-accelerated Dense Linear Machine Learning: A Precision-Convergence Trade-off. In *FCCM*.
[95] Kaan Kara, Christoph Hagleitner, Dionysios Diamantopoulos, Dimitris Syrivelis, and Gustavo Alonso. 2020. High Bandwidth Memory on FPGAs: A Data Analytics Perspective. In *FPL*.
[96] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C. Wu, M. Hempstead, and X. Zhang. 2020. RecNMP: Accelerating personalized recommendation with near-memory processing. In *ISCA*.
[97] Scott Kehler, John Hanesiak, Michelle Curry, David Sills, and Neil Taylor. 2016. High Resolution Deterministic Prediction System (HRDPS) Simulations of Manitoba Lake Breezes. In *Atmosphere-Ocean*.
[98] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *ISCA*.
[99] J. Kim, C. S. Oh, H. Lee, D. Lee, H. R. Hwang, S. Hwang, B. Na, J. Moon, J. Kim, H. Park, J. Ryu, K. Park, S. K. Kang, S. Kim, H. Kim, J. Bang, H. Cho, M. Jang, C. Han, J. LeeLee, J. S. Choi, and Y. Jun. 2012. A 1.2 V 12.8 GB/s 2 Gb Mobile Wide-I/O DRAM With 4×128 I/Os Using TSV Based Stacking. In *JSSC*.
[100] Jeremie S Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. 2018. GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies. In *BMC Genomics*.
[101] Gunjae Koo, Kiran Kumar Matam, Te I, H. V. Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. 2017. Summarizer: Trading Communication with Computing Near Storage. In *MICRO*.
[102] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, Sang-Hyuk Kwon, Je Min Ryu, Jong-Pil Son, Seongil O, Hak-Soo Yu, Haesuk Lee, Soo Young Kim, Youngmin Cho, Jin Guk Kim, Jongyoon Choi, Hyun-Sung Shin, Jin Kim, BengSeng Phuah, HyoungMin Kim, Myeong Jun Song, Ahn Choi, Daeho Kim, SooYoung Kim, Eun-Bong Kim, David Wang, Shinhaeng Kang, Yuhwan Ro, Seungwoo Seo, JoonHo Song, Jaeyoun Youn, Kyomin Sohn, and Nam Sung Kim. 2021. A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications. In *ISSCC*.
[103] Yi-Hsiang Lai, Yuze Chi, Yuwei Hu, Jie Wang, Cody Hao Yu, Yuan Zhou, Jason Cong, and Zhiru Zhang. 2019. HeteroCL: A Multi-Paradigm Programming Infrastructure for Software-Defined Reconfigurable Computing. In *FPGA*.
[104] Donghyuk Lee, Saugata Ghose, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. 2016. Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. In *ACM TACO*.
[105] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong. 2014. 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*.
[106] Jinho Lee, Heesu Kim, Sungjoo Yoo, Kiyoung Choi, H. Peter Hofstee, Gi-Joon Nam, Mark R. Nutter, and Damir Jamsek. 2017. ExtraV: Boosting Graph Processing near Storage with a Coherent Accelerator. In *VLDB*.
[107] Joo Hwan Lee, Jaewoong Sim, and Hyesoon Kim. 2015. BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models. In *PACT*.
[108] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, Seongil O, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021. Hardware Architecture and Software Stack for FIM Based on Commercial DRAM Technology. In *ISCA*.
[109] Vincent T Lee, Amrita Mazumdar, Carlo C del Mundo, Armin Alaghi, Luis Ceze, and Mark Oskin. 2018. Application Codesign of Near-Data Processing for Similarity Search. In *IPDPS*.
[110] Jiajie Li, Yuze Chi, and Jason Cong. 2020. HeteroHalide: From image processing DSL to efficient FPGA acceleration. In *FPGA*.
[111] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. 2016. Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories. In *DAC*.

[112] Jiawen Liu, Hengyu Zhao, Matheus A Ogleari, Dong Li, and Jishen Zhao. 2018. Processing-in-Memory for Energy-efficient Neural Network Training: A Heterogeneous Approach. In *MICRO*.
[113] Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu. 2017. Concurrent Data Structures for Near-Memory Computing. In *SPAA*.
[114] David Mayhew and Venkata Krishnan. 2003. PCI Express and Advanced Switching: Evolutionary Path to Building Next Generation Interconnects. In *HOTI*.
[115] Jiayuan Meng and Kevin Skadron. 2011. A Performance Study for Iterative Stencil Loops on GPUs with Ghost Zone Optimizations. In *IJPP*.
[116] Microsoft. [n.d.]. *Deploy ML models to field-programmable gate arrays (FPGAs) with Azure Machine Learning, https:// docs.microsoft.com/ en-us/azure/ machine-learning/how-to-deploy-fpga-web-service*.
[117] Amir Morad, Leonid Yavits, and Ran Ginosar. 2015. GP-SIMD Processing-in-Memory. In *ACM TACO*.
[118] Onur Mutlu. 2021. Intelligent Architectures for Intelligent Computing Systems. In *DATE*.
[119] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2019. Enabling Practical Processing in and near Memory for Data-Intensive Computing. In *DAC*.
[120] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2019. Processing Data Where It Makes Sense: Enabling In-Memory Computation. In *MicPro*.
[121] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2021. A Modern Primer on Processing in Memor. In *Emerging Computing: From Devices to Systems-Looking Beyond Moore and Von Neumann. Springer*.
[122] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim. 2017. GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks. In *HPCA*.
[123] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C. . Cher, C. H. A. Costa, J. Doi, C. Evangelinos, B. M. Fleischer, T. W. Fox, D. S. Gallo, L. Grinberg, J. A. Gunnels, A. C. Jacob, P. Jacob, H. M. Jacobson, T. Karkhanis, C. Kim, J. H. Moreno, J. K. O'Brien, M. Ohmacht, Y. Park, D. A. Prener, B. S. Rosenburg, K. D. Ryu, O. Sallenave, M. J. Serrano, P. D. M. Siegl, K. Sugavanam, and Z. Sura. 2015. Active Memory Cube: A Processing-in-Memory Architecture for Exascale Systems. In *IBM JRD*.
[124] Fábio CP Navarro, Hussein Mohsen, Chengfei Yan, Shantao Li, Mengting Gu, William Meyerson, and Mark Gerstein. 2019. Genomics and data science: an application within an umbrella. In *BMC*.
[125] Richard B Neale, Chih-Chieh Chen, Andrew Gettelman, Peter H Lauritzen, Sungsu Park, David L Williamson, Andrew J Conley, Rolando Garcia, Doug Kinnison, Jean-Francois Lamarque, et al. 2010. Description of the NCAR Community Atmosphere Model (CAM 5.0). In *NCAR Tech. Note*.
[126] Geraldo Francisco Oliveira, Juan GÃşmez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, and Onur Mutlu. 2021. DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks. In *IEEE Access*.
[127] M Necati Özişik, Helcio RB Orlande, Marcelo J Colaço, and Renato M Cotta. 2017. *Finite difference methods in heat transfer*. CRC Press.
[128] Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn. 2021. TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory. In *MICRO*.
[129] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K Mishra, Mahmut T Kandemir, Onur Mutlu, and Chita R Das. 2016. Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities. In *PACT*.
[130] J. T. Pawlowski. 2011. Hybrid Memory Cube (HMC). In *HCS*.
[131] Constantin Pohl, Kai-Uwe Sattler, and Goetz Graefe. 2019. Joins on high-bandwidth memory: a new level in the memory hierarchy. In *VLDB*.
[132] Seth H Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. 2014. NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads. In *ISPASS*.
[133] Krzysztof Rojek et al. 2019. CFD Acceleration with FPGA. In *H2RC*.
[134] Satish Kumar Sadasivam, Brian W Thompto, Ron Kalla, and William J Starke. 2017. IBM POWER9 Processor Architecture. In *IEEE Micro*.
[135] Kentaro Sano, Yoshiaki Hatsuda, and Satoru Yamamoto. 2014. Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth. In *TPDS*.
[136] Paulo C Santos, Geraldo F Oliveira, Diego G Tomé, Marco AZ Alves, Eduardo C Almeida, and Luigi Carro. 2017. Operand Size Reconfiguration for Big Data Processing in Memory. In *DATE*.
[137] Christoph Schär, Oliver Fuhrer, Andrea Arteaga, Nikolina Ban, Christophe Charpilloz, Salvatore Di Girolamo, Laureline Hentgen, Torsten Hoefler, Xavier Lapillonne, David Leutwyler, Katherine Osterried, Davide Panosetti, Stefan Rudishli, Linda Schlemmer, Thomas C. Schulthess, Michael Sprenger, Stefano Ubbiali, and Heini Wernli. 2020. Kilometer-scale Climate Models: Prospects and Challenges. In *BAMS*.
[138] Vivek Seshadri, Kevin Hsieh, Amirali Boroum, Donghyuk Lee, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2015. Fast Bulk Bitwise AND and OR in DRAM. In *CAL*.

[139] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, et al. 2013. RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization. In *MICRO*.
[140] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2016. Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM. In *arXiv*.
[141] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2017. Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology. In *MICRO*.
[142] Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. 2015. Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses. In *MICRO*.
[143] Vivek Seshadri and Onur Mutlu. 2017. Simple operations in memory to reduce data movement. In *Advances in Computers*.
[144] Vivek Seshadri and Onur Mutlu. 2019. In-DRAM bulk bitwise execution engine. In *arXiv*.
[145] DD Sharma. 2019. Compute Express Link. In *CXL Consortium White Paper*.
[146] William Andrew Simon, Yasir Mahmood Qureshi, Marco Rios, Alexandre Levisse, Marina Zapater, and David Atienza. 2020. BLADE: An in-Cache Computing Architecture for Edge Devices. In *TC*.
[147] Gagandeep Singh et al. 2019. NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning. In *DAC*.
[148] Gagandeep Singh, Mohammed Alser, Damla Senol Cali, Dionysios Diamantopoulos, Juan Gómez-Luna, Henk Corporaal, and Onur Mutlu. 2021. FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications. In *IEEE Micro*.
[149] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. 2018. A Review of Near-Memory Computing Architectures: Opportunities and Challenges. In *DSD*.
[150] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. 2019. Near-Memory Computing: Past, Present, and Future. In *MicPro*.
[151] Gagandeep Singh, Dionysios Diamantopolous, Juan Gómez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal. 2021. Modeling FPGA-Based Systems via Few-Shot Learning. In *FPGA*.
[152] Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gómez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal. 2020. NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling. In *FPL*.
[153] Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Sander Stuijk, and Henk Corporaal. 2019. NARMADA: Near-memory horizontal diffusion accelerator for scalable stencil computations. In *FPL*.
[154] Gagandeep Singh, Dionysios Diamantopoulos, Sander Stuijk, Christoph Hagleitner, and Henk Corporaal. 2019. Low Precision Processing for High Order Stencil Computations. In *Springer LNCS*.
[155] Robert Strzodka, Mohammed Shaheen, Dawid Pajak, and Hans-Peter Seidel. 2010. Cache Oblivious Parallelograms in Iterative Stencil Computations. In *ICS*.
[156] Jeffrey Stuecheli et al. 2018. IBM POWER9 opens up a new era of acceleration enablement: OpenCAPI. In *IBM JRD*.
[157] Jeffrey Stuecheli, Bart Blaner, CR Johns, and MS Siegel. 2015. CAPI: A Coherent Accelerator Processor Interface. In *IBM JRD*.
[158] B. Sukhwani, T. Roewer, C. L. Haymes, K. Kim, A. J. McPadden, D. M. Dreps, D. Sanner, J. V. Lunteren, and S. Asaad. 2017. ConTutto âĂŞ A Novel FPGA-based Prototyping Platform Enabling Innovation in the Memory Subsystem of a Server Class Processor. In *MICRO*.
[159] Lukasz Szustak, Krzysztof Rojek, and Pawel Gepner. 2013. Using Intel Xeon Phi Coprocessor to Accelerate Computations in MPDATA Algorithm. In *PPAM*.
[160] Yuan Tang, Rezaul Alam Chowdhury, Bradley C Kuszmaul, Chi-Keung Luk, and Charles E Leiserson. 2011. The Pochoir Stencil Compiler. In *SPAA*.
[161] Felix Thaler, Stefan Moosbrugger, Carlos Osuna, Mauro Bianco, Hannes Vogt, Anton Afanasyev, Lukas Mosimann, Oliver Fuhrer, Thomas C Schulthess, and Torsten Hoefler. 2019. Porting the COSMO Weather Model to Manycore CPUs. In *PASC*.
[162] Llewellyn Thomas. 1949. Elliptic Problems in Linear Differential Equations over a Network. In *Watson Sci. Comput. Lab. Rept., Columbia University*.
[163] Po-An Tsai et al. 2017. Jenga: Software-Defined Cache Hierarchies. In *ISCA*.
[164] Dean M Tullsen, Susan J Eggers, and Henry M Levy. 1995. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *ISCA*.
[165] Jan van Lunteren, Ronald Luijten, Dionysios Diamantopoulos, Florian Auernhammer, Christoph Hagleitner, Lorenzo Chelini, Stefano Corda, and Gagandeep Singh. 2019. Coherently Attached Programmable Near-Memory Acceleration Platform and its application to Stencil Processing. In *DATE*.
[166] Vasily Volkov and James W Demmel. 2008. Benchmarking GPUs to tune dense linear algebra. In *SC*.
[167] Mohamed Wahib and Naoya Maruyama. 2014. Scalable Kernel Fusion for Memory-Bound GPU Applications. In *SC*.
[168] Hasitha Muthumala Waidyasooriya and Masanori Hariyama. 2019. Multi-FPGA accelerator architecture for stencil computation exploiting spacial and temporal scalability. In *IEEE Access*.

[169] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama. 2017. OpenCL-Based FPGA-Platform for Stencil Computation and Its Optimization Methodology. In *TPDS*.
[170] Shuo Wang and Yun Liang. 2017. A comprehensive framework for synthesizing stencil algorithms on FPGAs using OpenCL model. In *DAC*.
[171] Zeke Wang, Hongjing Huang, Jie Zhang, and Gustavo Alonso. 2020. Shuhai: Benchmarking High Bandwidth Memory on FPGAs. In *FCCM*.
[172] Lukas Wenzel, Robert Schmid, Balthasar Martin, Max Plauth, Felix Eberhardt, and Andreas Polze. 2018. Getting Started with CAPI SNAP: Hardware Development for Software Engineers. In *Euro-Par*.
[173] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore architectures. In *CACM*.
[174] Lingxi Wu, Rasool Sharifi, Marzieh Lenjani, Kevin Skadron, and Ashish Venkat. 2021. Sieve: Scalable In-situ DRAM-based Accelerator Designs for Massively Parallel k-mer Matching. In *ISCA*.
[175] Jingheng Xu, Haohuan Fu, Wen Shi, Lin Gan, Yuxuan Li, Wayne Luk, and Guangwen Yang. 2018. Performance Tuning and Analysis for Stencil-Based Applications on POWER8 Processor. In *ACM TACO*.
[176] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. 2014. TOP-PIM: Throughput-Oriented Programmable Processing in Memory. In *HPDC*.
[177] Jun A. Zhang, Frank D. Marks, Jason A. Sippel, Robert F. Rogers, Xuejin Zhang, Sundararaman G. Gopalakrishnan, Zhan Zhang, and Vijay Tallapragada. 2018. Evaluating the Impact of Improvement in the Horizontal Diffusion Parameterization on Hurricane Prediction in the Operational Hurricane Weather Research and Forecast (HWRF) Model. In *Weather and Forecasting*.
[178] Maohua Zhu, Youwei Zhuo, Chao Wang, Wenguang Chen, and Yuan Xie. 2018. Performance Evaluation and Optimization of HBM-Enabled GPU for Data-intensive Applications. In *VLSI*.
[179] Hamid Reza Zohouri, Artur Podobas, and Satoshi Matsuoka. 2018. Combined spatial and temporal blocking for high-performance stencil computation on FPGAs using OpenCL. In *FPGA*.