NARMADA: Near-memory horizontal diffusion accelerator for scalable stencil computations

Gagandeep Singh[a,b], Dionysios Diamantopoulos[b], Christoph Hagleitner[b], Sander Stuijk[a], Henk Corporaal[a]
[a]Eindhoven University of Technology, Netherlands [b]IBM Research - Zurich, Switzerland
[a]{g.singh, s.stuijk, h.corporaal}@tue.nl [b]{sin, did, hle}@zurich.ibm.com

***Abstract*—Real-world weather forecasting applications consist of compound stencil kernels that do not perform well on conventional architectures. This behavior is due to their complex data access patterns, limited data reusability, and low arithmetic intensity[1]. To overcome these issues, we harness the potential of near-memory computing by offloading a horizontal diffusion kernel, which is a compound stencil kernel, from the COSMO weather prediction application to a reconfigurable fabric. We use a heterogeneous system that comprises a CPU and an FPGA with on-chip SRAM memory and on-board DRAM memory. By introducing a memory hierarchy tailored to the targeted application and using a coherent memory model, we move the computation close to the memory, which improves memory efficiency. Our hardware design on the FPGA uses high-level synthesis techniques and results in an accelerator with IBM® CAPI 2.0 (Coherent Accelerator Processor Interface) technology. We evaluate it against a tuned software implementation running on an IBM®POWER9® host system. The experimental results show that these kernels on an FPGA can outperform a complete 16-core POWER9 node (configured with 64 threads) by 3.3×. Moreover, our solution provides an 18× improvement in the active energy consumption.**

***Index Terms*—near-memory computing, FPGA, HPC, performance, energy-efficiency, CAPI**

I. Introduction

The Consortium for Small-Scale Modeling (COSMO) [1] was established to meet the high-resolution forecasting requirements of weather services. The COSMO model is a non-hydrostatic atmospheric prediction model currently being used by a dozen nations for meteorological purposes and research applications. The dynamical core (dycore) is the central part of the COSMO model, which consists of compound stencil kernels that operate on a three-dimensional grid [2]. In the literature [3]–[6], we usually come across elementary stencils. However, compound stencils consist of a collection of elementary stencils that perform a series of element-wise computations on a complete three-dimensional grid. Horizontal diffusion (hdiff) is one such compound kernel found in the dycore of the COSMO model and also in many other weather applications [7]–[9].

In the HPC domain, power consumption has become one of the significant implementation bottlenecks for these kernels. To overcome this issue, complementing general-purpose cores with specialized FPGA accelerators is a promising approach to enhance overall system performance without a significant increase in power consumption. However, taking full advantage of an FPGA for accelerating a workload is not a trivial task. First, FPGAs run at extremely low frequencies compared to the state-of-the-art CPU. Therefore, to match the CPU's performance, an FPGA must exploit at least one order of magnitude more parallelism in a target workload. Second, efficient FPGA designing requires comprehensive knowledge of the FPGA micro-architecture with sufficient FPGA programming skills, which often leads to a significant design effort. Third, CPU and FPGA are connected via low-bandwidth I/O links, which cause significant data movement overhead. To this end, IBM's CAPI technology [11] (Coherent Accelerator Processor Interface) addresses the above issues by going beyond today's I/O attached acceleration engines. This technology enables accelerators to act as coherent peers to the host processor by giving accelerator modules access to the virtual address space of a process running on the host at a high bidirectional bandwidth of 32 GB/s. This coherent memory model allows the seamless sharing of information between the host CPU and FPGA [12]. Therefore, by deploying a coherent accelerator, we can exploit the benefits of near-memory computing [13] without sacrificing the flexibility of the host CPU. The recent release of the CAPI SNAP environment[2] has further improved programmability because it provides a comprehensive hardware and software layer on top of the lower-level CAPI interface.

In this paper, we leverage the CAPI SNAP environment to

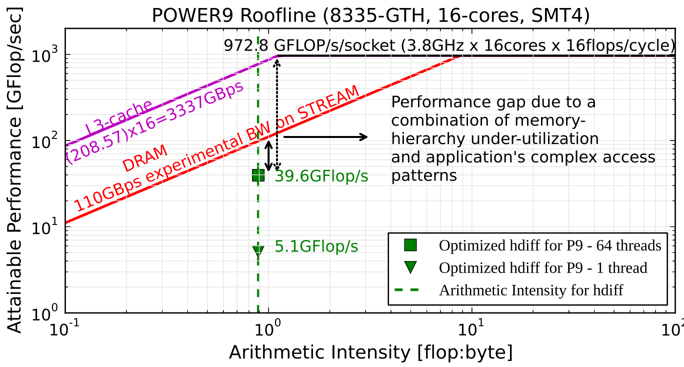


Fig. 1: Roofline [10] for POWER9 (1-socket) showing a horizontal diffusion kernel for single thread and 64 threads (one socket) implementation.

[1]The arithmetic intensity (or operational intensity) is the number of operations per byte of the memory traffic.

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

build a near-memory horizontal diffusion accelerator (NARMADA), a CAPI-based hardware accelerator on an FPGA for a horizontal diffusion kernel from a real-world COSMO weather-prediction application. Owing to its complex memory pattern, this kernel on current CPUs is limited by memory bandwidth and access latency, typically resulting in about 10% or less sustained floating-point performance. Figure 1 shows the roofline [10] plot for the current state-of-the-art IBM® 16-core POWER9 CPU (G8335-GTH)[3,4]. After optimizing the hdiff kernel for the POWER architecture by following the approach in [14], we were able to achieve 39.6 GFLOP/s. This low performance is due to inefficient bandwidth utilization because of the limited locality of the data and multiple passes of data transfers.

On the other hand, NARMADA was able to outperform a complete 16-core POWER9 socket by 3.3× in terms of performance. Moreover, our solution leads to an active energy improvement of 18× with an energy efficiency of 7.2 GFLOPS/Watt. In addition, our co-designed hardware-software framework provides an optimized API to interface efficiently with the rest of the code and can be extended to other compound kernels in the COSMO model.

In summary, the major contributions of this paper are:

- First implementation and optimization of horizontal diffusion kernel from a real-world weather-prediction COSMO application on a re-configurable fabric.
- We propose a data-centric heterogeneous memory hierarchy caching scheme.
- An architecture based on IBM CAPI 2.0 on a high-end FPGA, which we compare with the state-of-the-art IBM® POWER9® CPU.
- Performance and energy evaluation with scalability analysis of our accelerator. We show that NARMADA can outperform a complete 16-core POWER9 node (configured with 64 threads) by 3.3×.

The remainder of this article is structured as follows. Section II provides details of the horizontal diffusion kernel and highlights the CAPI SNAP environment employed for building NARMADA. Section III presents the actual design with the heterogeneous memory hierarchy. Section IV describes a prediction model developed for scaling NARMADA on other FPGA devices. Section V outlines the system setup and the results of our experiments, including an efficiency analysis of our accelerator compared to an IBM® POWER9 CPU. Section VI lists related work and Section VII concludes the paper by indicating future directions.

II. Background

This section gives an overview of the horizontal diffusion kernel and the CAPI SNAP framework that has been used to connect our accelerator to a POWER9 system.

[3]This CPU belongs to the POWER9 family of processors, which fuels Summit and Sierra, which in turn are currently #1 and #2 of the TOP500.

[4]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.

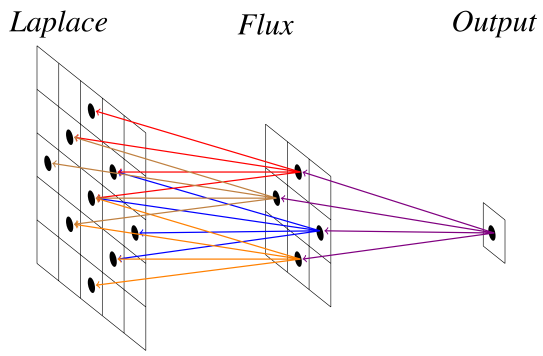


Fig. 2: Horizontal diffusion compound kernel composition in a two dimensional plane.

A. Horizontal Diffusion

Unlike stencils found in the literature [3]–[6], real-world compound stencils consist of a collection of elementary stencils that perform a sequence of element-wise computations. Algorithm 1 shows a pseudo-code for the horizontal diffusion (hdiff) kernel. This kernel iterates over a 3D grid performing *laplacian* and *flux* as depicted in Figure 2 to calculate different grid points. Such compound kernels have complex memory access patterns and low arithmetic intensity because they have a limited number of operations per loaded value. An implementation on the CPU leads to limited data locality, inefficient memory usage, and unexploited parallelism.

Algorithm 1: Pseudo-code for horizontal diffusion used by the COSMO [1] atmospheric model

```
Function horizontalDiffusion(float src, float dest)
    for $d \leftarrow 1$ to $depth$ do
        for $c \leftarrow 2$ to $column - 2$ do
            for $r \leftarrow 2$ to $row$-2 do
                /* laplacian calculate */
                $lap_{CR} = laplaceCalculate(c, r)$
                /* row-laplacian */
                $lap_{CRm} = laplaceCalculate(c, r-1)$
                $lap_{CRp} = laplaceCalculate(c, r+1)$
                /* column-laplacian */
                $lap_{CmR} = laplaceCalculate(c-1, r)$
                $lap_{CpR} = laplaceCalculate(c+1, r)$
                /* column-flux calculate */
                $flux_C = lap_{CpR} - lap_{CR}$
                $flux_{Cm} = lap_{CR} - lap_{CmR}$
                /* row-flux calculate */
                $flux_R = lap_{CRp} - lap_{CR}$
                $flux_{Rm} = lap_{CR} - lap_{CmR}$
                /* output calculate */
                $dest[d][c][r] = src[d][c][r] - c1 * (flux_{CR} - flux_{CmR}) + (flux_{CR} - flux_{CRm})$
            end
        end
    end
end
```

B. CAPI SNAP

The OpenPOWER Foundation Accelerator Workgroup created the CAPI SNAP (Storage, Network, and Analytics Programming) framework, which is an open-source environment, that establishes two technology novelties [15]: (i) it enables application programmers to embrace FPGA acceleration and CAPI's technology benefits, and (ii) it places the accelerated compute engines, or FPGA *actions*, closer to the data to achieve better performance. SNAP provides a simple API to call for an accelerated *action*, and also provides programming methods to code 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 HLS tool.

III. Design Methodology

A. Design Challenge

On our current HPC systems based on multi-core architectures, the horizontal diffusion kernel is not able to reach optimal performance due to complex memory patterns and its inefficiency to exploit a rigid cache hierarchy, see Figure 1. Moreover, owing to the low locality of the data, this kernel is not able to fully utilize the available memory bandwidth, which leads to a high data-transfer overhead in terms of latency and energy consumption. This inefficiency necessitates for an architecture that leads to fewer off-chip data movements with higher throughput for the loaded data. Therefore, our accelerator design can benefit from a data-centric approach rather than a compute-centric approach. Such architectures have proven to increase system performance by effectively managing the data [13].

B. NARMADA, a near-memory horizontal diffusion accelerator

Figure 3 shows a high-level overview of our integrated system. The FPGA is connected to a server system based on an IBM® POWER9 processor using a coherent accelerator processor interface 2.0 (CAPI 2.0). NARMADA consists of accelerator functional unit (AFU) that interact with the host system through the power service layer (PSL), which is the CAPI endpoint on the FPGA. Figure 4a shows the flow of data from the host memory to the FPGA memory (the mapping of data on different FPGA memories is shown in Figure 3). The collected weather data, which is based on the atmospheric model resolution grid, is stored in the DRAM of the host system (❶ in Figure 4a). As FPGAs have limited resources, we propose 3D window-based grid transfer from the host DRAM to the FPGA. We employ a double buffering technique (❸) between the CPU and the FPGA to hide the PCIe overhead. By configuring a buffer of 64 cachelines, between the AXI4 interface of CAPI2/PSL and the AFU, we can reach the theoretical peak bandwidth of CAPI2/PCIe (16 GB/s). We create a specialized memory hierarchy from the heterogeneous FPGA memories. By using a greedy algorithm, we determine the best-suited hierarchy for our kernel. The heterogeneous memory controller (shown in Figure 3) handles the data placement to the appropriate memory type.

On the FPGA, following the initial buffering, the 3D window-based grid data is mapped onto the FPGA DRAM (❹). The window size represents the portion of the grid an AFU will process. As the hdiff kernel has limited spatial locality, we cache only certain parts of data around the *row* dimension into the register file (❻) (LUTs and FFs). To bridge the latency gap between DRAM and the register file, we introduce another memory level (❺) in between, which is made up of BRAM (B-Hierarchy). This intermediate hierarchy consists of 3D window data decomposed (or sliced) into a 2D grid. However, to exploit the available FPGA resources further, we also introduce another level of URAM-based memory between the DRAM and BRAM (U-Hierarchy). Unlike traditionally fixed CPU memory hierarchies, that usually makes poor use of cache-hierarchies, an algorithm-specific hierarchy has been proven to improve the energy-delay product (EDP) by tailoring the cache-levels and cache-size to applications' access patterns [16]. In our case, we experimented with two cache setups, i.e., B-RAM and U-RAM. In every setup we place sequential data accesses, corresponding to a single hdiff step (lines 5-14 of Algorithm 1), in a single 2D memory space, composed of BRAMs or URAMs respectively. This 2D memory space is configured with multiple parallel BRAMs or URAMs using on-chip memory reshaping techniques that allow for bandwidth maximization [17]. Figure 4b shows the NARMADA framework. Each NARMADA AFU employs an array of processing elements (PEs) to solve the horizontal diffusion computation. Additionally, to achieve a massively parallel design, we exploit the immediately available spatial parallelism by pipelining our PEs. NARMADA AFUs act as peers to the CPU by accessing the main memory through a high-performance cache-coherent link enabled by PSL. A software-defined API handles offloading jobs ("*actions*") to NARMADA with an interrupt-based queuing mechanism that allows minimal CPU usage (and hence, power) during FPGA use. The job manager dispatches jobs to streams managed in the stream scheduler. Each stream controls the execution of a job and determines which data is to be read from the host memory and sent to the PE array through DMA transfers. The pool of heterogeneous on-chip memory is used to store the input data from the main memory and the intermediate data generated by an AFU. HDIFF SNAP, as depicted in Figure 4b, allows *actions* on an FPGA device to access the virtual memory space on the host in addition to resources such as the FPGA on-board DRAM memory. This access is achieved through the "*actions*" wrapper, which provides the interface

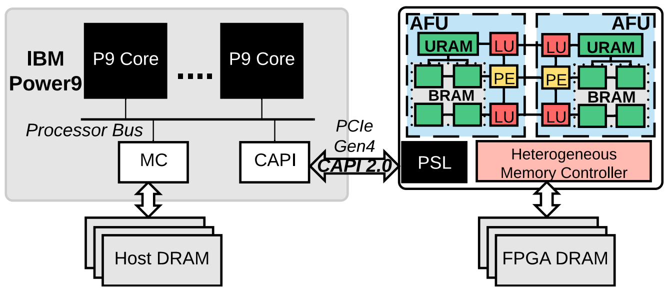
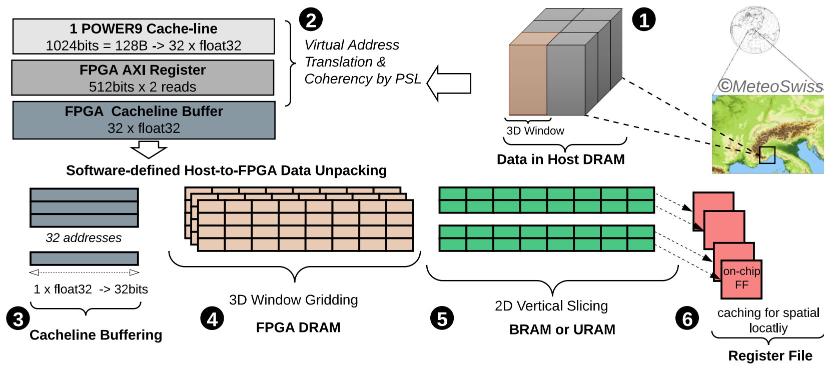
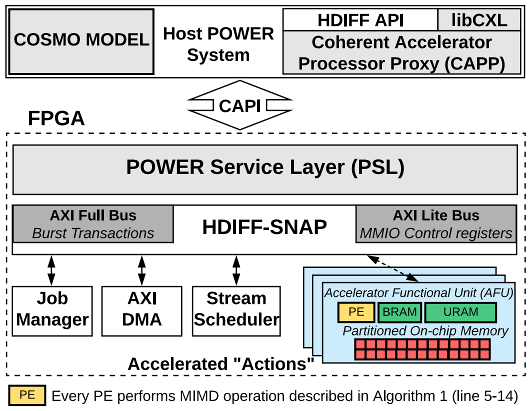


Fig. 3: Heterogeneous platform with an IBM® POWER9® host system connected to an FPGA accelerator platform via CAPI 2.0.



(a)



COSMO MODEL
Host POWER System
HDIFF API
libCXL
Coherent Accelerator Processor Proxy (CAPP)
CAPI
FPGA
POWER Service Layer (PSL)
AXI Full Bus
Burst Transactions
HDIFF-SNAP
AXI Lite Bus
MMIO Control registers
Job Manager
AXI DMA
Stream Scheduler
Accelerator Functional Unit (AFU)
PE
BRAM
URAM
Partitioned On-chip Memory
Accelerated "Actions"
PE Every PE performs MIMD operation described in Algorithm 1 (line 5-14)

(b)

Fig. 4: (a) Data flow sequence from the host DRAM in the form of a 3D window to the on-board FPGA memory via POWER9 cachelines. Heterogeneous memory partitioning of on-chip memory blocks enable minimal R/W latency across the FPGA memory hierarchy. (b) NARMADA framework: NARMADA acts as a peer to the CPU by accessing the main memory through a high-performance cache-coherent link, enabled by PSL. A software-defined API handles offloading jobs ("*actions*") to NARMADA with an interrupt-based queuing mechanism that allows minimal CPU usage—and hence, power—during the FPGA use.

for the user logic to the power service layer module. Data is transferred between the POWER9 and FPGA at the granularity of cachelines of 128 bytes (❷ in Figure 4a). A CAPI link can transfer 512 bits of data per cycle, and hence we need two reads for a complete cacheline. Assume that the total number of cachelines required to represent an entire grid domain of dimension $I \times J \times K$ is represented by ζ_{Total}. Let ζ_{window} represent the amount of data processed by a single window in terms of cachelines. Then the total number of sweeps required by an AFU to process the entire domain is given by $\lceil \frac{\zeta_{\text{Total}}}{\zeta_{\text{window}}} \rceil$. However, increasing the number of AFUs (μ) decreases the number of sweeps by a factor of μ. This is because each AFU is assigned its own ζ_{window}. Hence, the amount of on-chip storage space required per AFU depends heavily on the 3D window dimensions and the memory hierarchy used. The maximum available resources on an FPGA, heavily constrain the maximum number of AFUs ($\mu_{\max}$) that can be instantiated on a chip, which is given by

$$\mu_{\max} = \min \left\lfloor \frac{\overrightarrow{\eta_{\max}}}{\overrightarrow{\eta_{\text{AFU}}}} \right\rfloor \qquad (1)$$

where $\overrightarrow{\eta_{\max}}$ represents a vector containing the maximum available resources on a single FPGA board, being a collection of BRAM, DSP, FF, LUT, and URAM, and where the term $\overrightarrow{\eta_{\text{AFU}}}$ represents a vector of FPGA resources used by a single AFU.

IV. Scaling Prediction Model

The implementation of a design on an FPGA device can provide different gains when implemented on another FPGA device. However, HLS tools are extremely slow for effective design-space exploration because, even for a single design point implementation, run times could take hours. This inhibits designers from making appropriate cost-value decisions [18], [19]. Additionally, back-of-the-envelope calculations cannot accurately predict complex design behavior. For instance, in our design, memory partitioning with a bigger window size could lead to a complex network of address multiplexers. Therefore, we built a model using a best-fit empirical polynomial equation to explore rapidly the design space of NARMADA on different FPGA devices. Algorithm 2 explains our two-step approach to estimate the performance of NARMADA on a new FPGA device.

To construct our model, we use implementation data from an FPGA device from a family and use the constructed model to make predictions for other devices across that family. The polynomial coefficients $(a_n, a_{n-1}, ..., a_0)$ are estimated at design time using a least-squares regression. In Algorithm 2, the prediction of scaled performance $\hat{\rho}$ refers to performance with μ AFUs using a window of ω. In a similar way, we make a model for all the FPGA resources.

Algorithm 2: AFU scaling prediction model

```
   Result: Pareto optimal $(\omega, \mu)$
 1 initialization
 2 while $FPGA_{device}$ do
 3     Step 1: Empirical characterization per AFU
       Input  : $\omega = \omega_{min}, \mu = \mu_{min}$
       Output: $\overrightarrow{\eta_{AFU}}$, $\rho_{AFU}$
 4     High-level-synthesis with Vivado HLS &
 5     FPGA implementation with Vivado
 6     Step 2: Performance Modeling
       Input  : $\overrightarrow{\eta_{AFU}}, \rho_{AFU}$
       Output: Pareto optimal $(\omega, \mu)$
 7     for $\omega = \omega_{min} : \omega_{max}$ do
 8         while $\eta_{\{BRAMorDSPorFForLUT\}}$ != $\eta_{max\{BRAMorDSPorFForLUT\}}$ do
 9             $\hat{\rho} = a_n \times \mu^n(\omega) + a_{n-1} \times \mu^{n-1}(\omega) + ... + a_1 \times \mu(\omega) + a_0$
10             $\mu \leftarrow \mu + 1$
11         end
12         maximize $(\hat{\rho})$
13     end
14 end
```

TABLE I: Considered system parameters and hardware configuration for the CPU and the FPGA board

Host CPU	16-cores IBM® POWER9® AC922 @3.2 GHz, 4 SMT
Cache-Hierarchy	32 KiB L1-I/D, 256 KiB L2, 10 MiB L3
System Memory	32GiB RDIMM DDR4 2666 MHz
FPGA Board	Alpha Data ADM-PCIE-9V3 PCI card
FPGA Family	Xilinx Virtex® Ultrascale+™ XCVU3P-2
Device Memory	8GiB (DDR4)
CAPI Interface	PCIe Gen4 x8

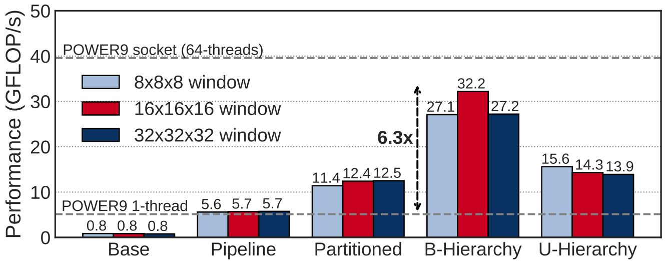


Fig. 5: Performance for a single NARMADA AFU using various optimization strategies. Also, shown is the single thread and single socket (64-threads) performace of the IBM POWER9.

V. Results

A. System Integration

We implemented our design on an Alpha-Data ADM-PCIE-9V3 card featuring the Xilinx Virtex® Ultrascale+™ XCVU3P-FFVC1517-2-i with an IBM® POWER9 as the host system. The POWER9 socket has 16 cores, each of which supports four-thread simultaneous multi-threading. Table I provides complete details of our system parameters. We have co-designed our hardware and software interface around the SNAP platform while using the HLS design flow.

B. Evaluation

Similar to the grid domain used by the COSMO model, we ran our experiments using a $256 \times 256 \times 64$-point input domain. We varied the AFU window size from $8 \times 8 \times 8$ to $32 \times 32 \times 32$. Figure 5 shows performance results for a single AFU with various optimization strategies. First, to increase the throughput, we pipelined our design to exploit the inherent parallelism in the algorithm. However, the on-chip BRAM has only two read/write ports, so multiple read accesses to the same BRAM caused the entire pipeline to stall. To overcome this, we incremented our design by partitioning the data into different memory blocks, which nearly doubled the performance.

To achieve further performance gains, we carefully tuned the memory subsystem to obtain a heterogeneous memory hierarchy tailored to our algorithm. For this tuning, we apply the memory scheme as discussed in Section III.

In Figure 5, we attribute B-Hierarchy and U-Hierarchy to our heterogeneous memory schemes. The B-Hierarchy does not use the on-board URAM, whereas the U-Hierarchy has an additional rank of URAM-based memory. Using the B-Hierarchy, we were able to achieve 32.2 GFLOP/s for a single AFU, which is 6.3× faster than the single-thread performance

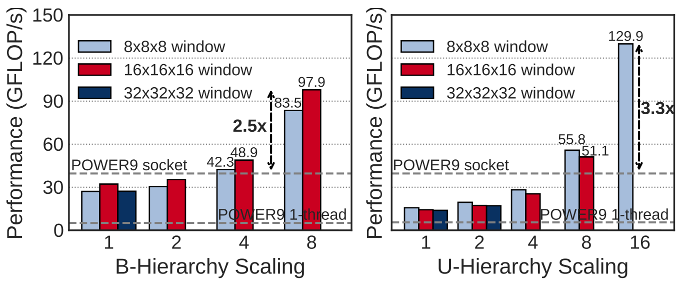


Fig. 6: Scaling the number of AFUs for B and U-based heterogeneous memory hierarchy with different window size. B-Hierarchy and U-hierarchy are able to achieve 2.5× and 3.3× performance compared to a POWER9 socket.

of the state-of-the-art POWER9 CPU with a rigid memory hierarchy. For the case of a single AFU, adding an extra layer of URAM does not improve performance compared to using a pure BRAM-based hierarchy. Moreover, it leads to a loss of performance. This is because URAM is denser (72-bits per address) and is not as distributed in the FPGA layout as the BRAM. Furthermore, bigger window size does not necessarily enhance performance because it takes longer to move data through different memories.

C. AFU Scaling

Figure 6 shows the scaling of AFUs for both B and U-Hierarchies. Note, we only show the results that we were able to implement before the on-board resources were exhausted. In the case of $32 \times 32 \times 32$ window size, BRAM exhaustion was the reason we were not able to synthesize more accelerators for both B and H-hierarchy. Upon scaling the B-hierarchy, we were able to implement a maximum of eight AFUs, which achieved a performance gain of 2.5× compared to a complete 16-core POWER9 socket configured with SMT4 (64-threads). However, by adding an extra level of URAM, we were able to decrease the number of BRAMs per AFU, thus reaching a total of 16 AFUs with a window size of $8 \times 8 \times 8$. This design yielded a performance of 129.9 GFLOP/s, which is 3.3× better than that of a complete POWER9 socket.

URAM thereby offers the resources to construct a more heterogeneous domain-specific cache hierarchy. This tailored memory hierarchy offers significantly more loading and eviction freedom than the hardware-managed caches in CPUs or GPUs. Moreover the U-Hierarchy scheme allows us to use the available resources more effectively, as shown in Figure 7.

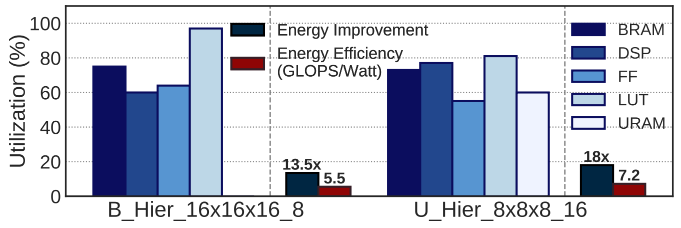


Fig. 7: FPGA resource utilization for AFUs with the highest performance achieved using B and U-based heterogeneous memory hierarchy. Also shown is the energy improvement and energy efficiency compared to an IBM® POWER9 system.

TABLE II: Prediction results for different CAPI 2.0 enabled boards across FPGA families.

FPGA Board	FPGA Device	Utilization%					Window	AFU	Performance (GFLOP/s)	Energy Impr.
		BRAM	DSP	FF	LUT	URAM				
ADM-PCIE-9V3*	XCVU3P-2	71	66	49	79	58	$8\times8\times8$	16	120.1	18.1×
ADM-PCIE-KU3*	XCKU3P-2	70	33	61	96	70	$16\times16\times16$	4	48.9	9.3×
Semptian NSA121B	XCKU060	55	49	73	79	-	$8\times8\times8$	8	83.5	20.6×
ADM-PCIE-8K5	XCKU115	59	58	81	77	-	$8\times8\times8$	16	162.7	19.2×

*URAM memory is available only in Ultrascale$^+$ families

As mentioned in Section IV partitioning a bigger window size leads to a complex network of address multiplexers. As a result, we see high usage of LUTs for the implemented designs.

D. Efficiency Analysis

We also evaluated the energy utilization and energy efficiency (GLOPS per Watt), see Figure 7, for our proposed FPGA solution. The execution time and energy consumption of each implemented configuration are shown in Figure 8. For the POWER9 node, we used the AMESTER[5] tool to measure the active power[6] consumption of 97.9 Watt by monitoring built-in power sensors on our system. With a multi-AFU B-Hierarchy-based solution, the active energy improvement of NARMADA is up to 13.5× compared to a POWER9 system. This configuration has an energy efficiency of 5.5 GFLOPS/Watt. By adding another layer of URAM-based memory, we were able to increase the energy improvement to 18× with 7.2 GFLOPS/Watt of energy efficiency.

E. Performance Prediction

We evaluate the accuracy of our model in terms of relative error ϵ_i to indicate how close the predicted value y_i' is to the actual value y_i. The training consists of fitting upto third order polynomials and selecting the best one in terms of mean relative error (MRE). We calculate the MRE for each FPGA family with Equation 2.

$$MRE\ (\epsilon_i) = \frac{|y_i' - y_i|}{y_i} \tag{2}$$

On average for considered CAPI-enabled FPGA devices (from Ultrascale and Ultrascale+ families), the predicted values are lower than the actual ones, with the highest error rate of 15.2% for the number of DSPs, whereas for the performance and the other resources it is below 11.6%. Figure 9 shows actual and predicted values using our approach, as described in Section IV.

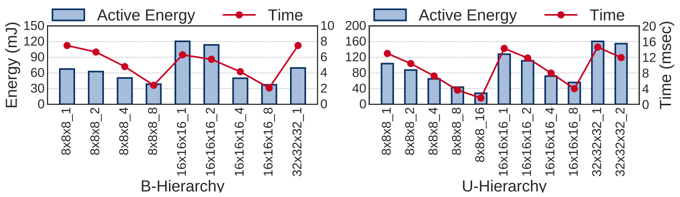


Fig. 8: Active energy consumption and execution time for different B and U-based hierarchies.

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

[6]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.

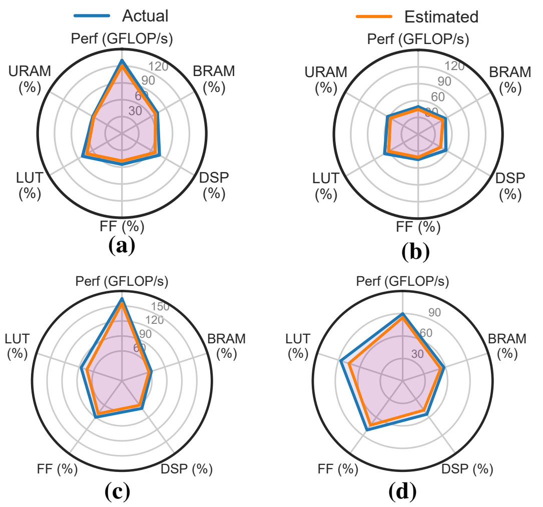


Fig. 9: Actual and predicted values for (a) 9V3 with $8\times8\times8$ (b) KU3 with $16\times16\times16$ (c) 8K5 with $8\times8\times8$ (d) NSA with $8\times8\times8$.

Table II shows the prediction of our accelerator across different high-end FPGA devices. KU3 is from the Ultrascale+ family and has half the resources as 9V3. NSA121B has more DSPs and BRAMs but fewer LUTs, which is the most expensive resource in our accelerator design. Although 8K5 has twice the resources as 9V3, and it achieves only 1.25× the performance compared to our best solution. This minimal performance gain is due to the absence of URAM. URAM thereby offers the resources to construct a more heterogeneous, domain-specific cache hierarchy.

Note that this design-space exploration (DSE) is not exhaustive. It was performed to evaluate the performance achieved by our implemented design across different CAPI-enabled FPGA devices. The DSE could be extended to different window size, more processing units, etc.

VI. Related Work

The IBM Coherent Accelerator Processor Interface (CAPI) removes significant device driver overhead and provides a high bandwidth link between the host processor and CAPI-enabled devices. IBM has leveraged this and shown the applicability of CAPI 1.0 to accelerate various algorithms such as arithmetic kernels [20] and graph processing [12]. Recently, Diamantopolous *et al.* [21] used the CAPI SNAP framework to build a trans-precise neural network co-processor.

Bianco *et al.* [22] optimized the COSMO Model for GPUs. Wahib *et al.* [23] developed an analytical performance model for choosing an optimal GPU-based execution strategy for various scientific applications, including COSMO. Gysi *et al.* [2] provided guidelines for optimizing stencil kernels for

CPU–GPU systems. To our knowledge, NARMADA is the first work to accelerate a compound stencil kernel from a real-world weather-prediction application on a reconfigurable fabric.

VII. Conclusion

We have presented the first design and implementation to accelerate horizontal diffusion (hdiff) kernel on a reconfigurable fabric. This kernel is a compound stencil found in various real-world weather-forecasting applications, including the COSMO prediction model. Compound kernels do not perform well on conventional architectures due to their complex data access patterns and low data reusability.

We used the IBM SNAP framework to build NARMADA, a hardware accelerator based on CAPI 2.0. Our experimental results showed that the hdiff kernel on an FPGA outperforms a highly optimized software implementation on a 16-core POWER9 configured with SMT4 (64-threads) by 3.3×. Moreover, our solution improves the active energy consumption by 18× with an energy efficiency of 7.2 GFLOPS/Watt. For future studies, we aim to optimize the window size and are experimenting with the precision tolerance of this kernel. Additionally, we will extend our hardware–software framework to include emerging memory options for commercial FPGAs, e.g., high-bandwidth memory (HBM), which offers much higher bandwidth with a low-energy footprint. Moreover, we would explore the applicability of our framework to other compound kernels in the COSMO model.

Acknowledgement

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). We would also like to thank Martino Dazzi for his valuable remarks. This work was partially supported by the H2020 research and innovation programme under grant agreement No 732631, project OPRECOMP.

References

[1] G. Doms and U. Schättler, "The Nonhydrostatic Limited-Area Model LM (Lokal-model) of the DWD. Part I: Scientific Documentation," *DWD, GB Forschung und Entwicklung*, 1999.

[2] T. Gysi, T. Grosser, and T. Hoefler, "MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures," in *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015, pp. 177–186.

[3] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 695–705, 2014.

[4] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "OpenCL-Based FPGA-Platform for Stencil Computation and Its Optimization Methodology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1390–1402, May 2017.

[5] Y. Chi, J. Cong, P. Wei, and P. Zhou, "SODA: Stencil with Optimized Dataflow Architecture," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[6] J. de Fine Licht, M. Blott, and T. Hoefler, "Designing scalable FPGA architectures using high-level synthesis," *ACM SIGPLAN Notices*, vol. 53, no. 1, pp. 403–404, 2018.

[7] S. Kehler, J. Hanesiak, M. Curry, D. Sills, and N. Taylor, "High Resolution Deterministic Prediction System (HRDPS) Simulations of Manitoba Lake Breezes," *Atmosphere-Ocean*, vol. 54, no. 2, pp. 93–107, 2016.

[8] R. B. Neale, C.-C. Chen, A. Gettelman, P. H. Lauritzen, S. Park, D. L. Williamson, A. J. Conley, R. Garcia, D. Kinnison, J.-F. Lamarque *et al.*, "Description of the NCAR community atmosphere model (CAM 5.0)," *NCAR Tech. Note NCAR/TN-486+ STR*, vol. 1, no. 1, pp. 1–12, 2010.

[9] J. A. Zhang, F. D. Marks, J. A. Sippel, R. F. Rogers, X. Zhang, S. G. Gopalakrishnan, Z. Zhang, and V. Tallapragada, "Evaluating the Impact of Improvement in the Horizontal Diffusion Parameterization on Hurricane Prediction in the Operational Hurricane Weather Research and Forecast (HWRF) Model," *Weather and Forecasting*, vol. 33, no. 1, pp. 317–329, 2018. [Online]. Available: https://doi.org/10.1175/WAF-D-17-0097.1

[10] Y. Lo, S. Williams, B. Straalen, T. Ligocki, M. Cordery, N. Wright, M. Hall, and L. Oliker, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, vol. 8966, pp. 129–148, 2015.

[11] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel, "CAPI: A Coherent Accelerator Processor Interface," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 7–1, 2015.

[12] J. Lee, H. Kim, S. Yoo, K. Choi, H. P. Hofstee, G.-J. Nam, M. R. Nutter, and D. Jamsek, "ExtraV: Boosting Graph Processing Near Storage with a Coherent Accelerator," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1706–1717, Aug. 2017. [Online]. Available: https://doi.org/10.14778/3137765.3137776

[13] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "A Review of Near-Memory Computing Architectures: Opportunities and Challenges," in *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, 2018, pp. 608–617.

[14] J. Xu, H. Fu, W. Shi, L. Gan, Y. Li, W. Luk, and G. Yang, "Performance Tuning and Analysis for Stencil-Based Applications on POWER8 Processor," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, p. 41, 2018.

[15] L. Wenzel, R. Schmid, B. Martin, M. Plauth, F. Eberhardt, and A. Polze, "Getting Started with CAPI SNAP: Hardware Development for Software Engineers," in *European Conference on Parallel Processing*. Springer, 2018, pp. 187–198.

[16] P.-A. Tsai *et al.*, "Jenga: Software-Defined Cache Hierarchies," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 652–665.

[17] D. Diamantopoulos and C. Hagleitner, "A System-Level Transprecision FPGA Accelerator for BLSTM Using On-chip Memory Reshaping," in *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2018, pp. 338–341.

[18] K. O'Neal, M. Liu, H. Tang, A. Kalantar, K. DeRenard, and P. Brisk, "HLSPredict: Cross Platform Performance Prediction for FPGA High-Level Synthesis," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[19] G. Singh *et al.*, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: ACM, 2019, pp. 27:1–27:6. [Online]. Available: http://doi.acm.org/10.1145/3316781.3317867

[20] H. Giefers, R. Polig, and C. Hagleitner, "Accelerating arithmetic kernels with coherent attached FPGA coprocessors," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 1072–1077.

[21] D. Diamantopoulos, H. Giefers, and C. Hagleitner, "ecTALK: Energy efficient coherent transprecision accelerators - The bidirectional long short-term memory neural network case," in *2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, April 2018, pp. 1–3.

[22] M. Bianco, T. Diamanti, O. Fuhrer, T. Gysi, X. Lapillonne, C. Osuna, and T. Schulthess, "A GPU Capable Version of the COSMO Weather Model," *ser. ISC*, vol. 13, pp. 34–35, 2013.

[23] M. Wahib and N. Maruyama, "Scalable Kernel Fusion for Memory-Bound GPU Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 191–202.