Exploring Challenges Associated with Employing SmartNICs as General-Purpose HPC Accelerators

Brody Williams
Texas Tech University
brody.williams@ttu.edu

Yong Chen
Texas Tech University
yong.chen@ttu.edu

Wendy Poole
Los Alamos National Laboratory
wkpoole@lanl.gov

Steve Poole
Los Alamos National Laboratory
swpoole@lanl.gov

***Abstract*—Galvanized by waning general-purpose CPU performance improvements, large-scale system architectures have shifted towards increasingly heterogeneous designs in recent years. Systems devised under this regime are constructed using the principles of hardware/software codesign and incorporate domain specific accelerators in order to optimize system performance for targeted use cases. In this context, smartNICs represent a class of accelerator devices currently experiencing a resurgence. In the past, power and flexibility limitations largely relegated these devices to performing only tasks associated with low-level networking operations. More recent smartNIC incarnations have addressed many of these design constraints. The full range of capabilities of these new smartNICs, however, is not well understood. Moreover, the suitability of these devices as general-purpose accelerators, particularly within the high performance computing domain, as well as any potential barriers to their more widespread adoption, remain to be seen. In this work, we detail our efforts to explore these open questions.**

***Index Terms*—SmartNIC, DPU, Accelerator, Offloading, High Performance Computing**

I. Introduction

The end of Moore's law and Dennard scaling has necessitated a fundamental shift with respect to the design of high performance computing (HPC) systems. The requirement to continue realizing performance improvements, coupled with the need to enhance energy efficiency, has accelerated the adoption of heterogeneous system architectures. Within this paradigm, specialized compute components such as general purpose graphics processing units (GPGPUs), tensor processing units (TPUs), and field-programmable gate arrays (FPGAs) are distributed throughout a given system. In turn, these distinct components are then utilized to optimize the execution of specific tasks, which often occur within larger workloads, with respect to a given metric.

Against this backdrop, a class of accelerator devices positioned at the network edge known as smart network interface cards, or smartNICs, has been experiencing a resurgence in recent years. Previous incarnations of smartNIC accelerators were primarily utilized to offload low-level networking functionality from the CPU. These devices, most often based on application-specific integrated circuit (ASIC) or FPGA designs, proved well suited to such tasks, but lacked the combination of power and flexibility necessary for more widespread adoption as general-purpose accelerators.

In contrast, the emerging generation of smartNICs were largely designed to rectify these shortcomings. These new smartNICs, also known as data processing units (DPUs) or infrastructure processing units (IPUs), couple some combination of system-on-a-chip (SoC) and FPGA based designs together and often also incorporate task-specific accelerators. SmartNICs that include SoCs featuring general purpose CPU cores, which can be programmed using high-level languages such as C and C++ rather than the hardware description languages (HDLs) necessary for FPGAs, enable the potential for new use cases for smartNICs. Examples of SoC-based smartNICs include the NVIDIA BlueField family of DPUs [18], the Marvell Octeon [17], and the Broadcom Stingray [2] among others.

Previous works have demonstrated the viability of smartNICs as accelerators for software-defined networking [6], packet inspection [11], and even secure RDMA-based communication [28]. The true breadth of use cases to which smartNICs may prove applicable, however, still remains to be seen. Moreover, the potential of smartNICs as general purpose accelerators, particularly with respect to high performance computing, as well as the challenges inherent with leveraging them effectively, are not well understood.

In this work we detail our efforts to explore these open questions using two proxy applications as case studies. In the following sections we introduce these applications and describe our methodology for injecting smartNIC-based offloading at the application level. We then provide a short evaluation of our offloaded implementations using NVIDIA BlueField 2 DPUs. Finally, we conclude with a discussion of insights gleaned from our experiments.

II. Offloading Methodology

For the experiments performed in this work, we employ a smartNIC offloading methodology that offloads work from the host at the application level. Herein, we strive to modify the application in question as little as possible and instead focus on offloading portions of the existing application semantics rather than making alterations to the application that render it more performant or otherwise amenable to offloading. We take this approach in order to emulate the sort of code changes that are likely to be feasible for the large, complex scientific codes often found in HPC environments.

We effect smartNIC offloading for our experiments by utilizing a multiple program, multiple data (MPMD) model built on MPI wherein host and smartNIC processes execute distinct code segments and are treated as true peers at the application level. This strategy enables us to minimize overheads associated with rapid prototyping and provides a portable mechanism for experimenting with different smartNICs as it does not rely on any particular communication middleware stack or other proprietary software. The resulting offloaded

application source code is, however, somewhat difficult to understand and debug. Moreover, utilizing this approach also induces performance penalties associated with performing data movement through MPI when more optimized mechanisms are available. Our current implementations are also limited in that they assume a one-to-one mapping between smartNIC and host processes.

III. PENNANT

A. Application Overview

For our first smartNIC offloading case study, we explore the PENNANT [4], [5] application developed by Los Alamos National Laboratory. PENNANT is an unstructured mesh physics mini-app designed as both a tool for benchmarking emerging architectures as well as a direct proxy for portions of the much larger FLAG hydrodynamics code. Written in approximately 3300 lines of C++ code, PENNANT supports 2-D meshes composed of arbitrary polygons. These unstructured meshes, similar to complex graphs, are organized into geometric elements known as points, edges, and zones in zero, one, and two dimensions, respectively. At a high level, the physics calculations that take place within PENNANT are organized into two stages. The first, known as the *predictor stage*, advances a number of variables to the middle of a given hydrodynamics cycle in order to compute intermediate values. The second, or *corrector stage*, then progresses all variables associated with the simulation to the end of the timestep. The calculations performed in these stages are based on staggered grid methods wherein some variables are associated with points while others are attached to zones. Double-precision floating-point formats are utilized for both of these variable types. Due to the irregular relationships between distinct but closely related variables, frequent translations using indirection arrays are necessary during execution. As a result, PENNANT exhibits the challenging, irregular memory access patterns that characterize many scientific applications within the high performance computing domain. Although a more in depth description of PENNANT is beyond the scope of this work, for further detail we refer the reader to [4].

The reference implementation of PENNANT that we modify for our case study is parallelized using MPI and, optionally, OpenMP. Conventional domain decomposition techniques are utilized to partition the overall mesh into distinct subsets whose states are progressed by independent MPI processes. Values along mesh subset boundaries are replicated and exchanged as necessary during execution using a primary/replica communication scheme. When enabled, OpenMP is used to further subdivide each mesh subset into thread-parallel *chunks* for processing.

B. Offloading Strategy

In order to better understand the runtime behavior of the application and identify potential smartNIC offload targets, we gathered execution profiles for PENNANT when solving different problem inputs using the TAU [27] toolkit. An analysis of the resulting data revealed that, at least for relatively small scales, communication that takes place within the application represents an insignificant portion of the overall run time, and, as a result, PENNANT is a thoroughly compute-bound application. This implies that, in contrast to more conventional smartNIC use cases, any meaningful smartNIC offloading strategy for PENNANT should incorporate computational offloads.

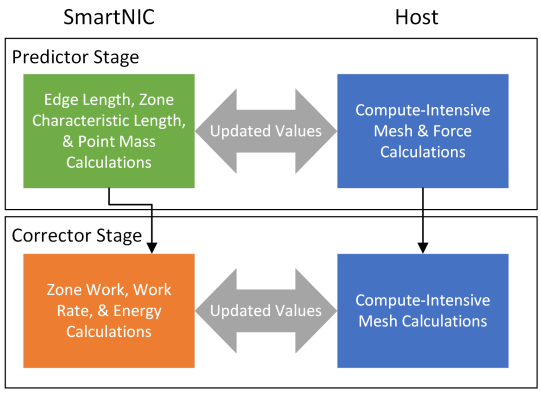

Fig. 1. Diagram illustrating offloaded functionality in PENNANT. Blue boxes represent computation performed on the host in both implementations. The green box indicates computation offloaded in both V1 and V2. The orange box contains functionality offloaded in V1, but performed on the host in V2.

Towards this end, we analyzed the source code of the hydrodynamics algorithm that constitutes the core of the application and correlated our findings together with the timing information provided by our profiling efforts in an attempt to identify functionality suitable for offloading. Herein, we discovered that, overall, the various calculations performed within the PENNANT hydro cycle are tightly coupled such that most stages of the computation are directly dependent on data computed in immediately preceding steps. As a result of these data dependencies, minimal opportunities for overlapped computation on the host and smartNIC exist.

Despite this observation, we identified a number of calculations sufficiently decoupled from the more compute intensive portions of the PENNANT hydro cycle as to be potentially viable offload candidates. The computation of edge lengths as well as zone characteristic lengths, both performed early in the predictor stage, are two such candidates. Each of these calculations requires only a single set of input variables and produce results not needed until later in the hydro cycle. Similarly, a portion of the point mass calculation, which includes both computation and communication, can be decoupled from the more compute intensive force and mesh calculations performed in the predictor stage. Within the corrector stage, the calculation of zone work, work rate, and energy values can also be largely separated from the more time consuming mesh updates performed in this stage. However, as these calculations depend on a number of variables obtained in the predictor stage, offloading these computation routines necessitates transferring a considerable amount of data between the host and smartNIC. Furthermore, the produced values in these routines are also immediately required at the beginning of the following hydro cycle.

Utilizing these insights, we develop two PENNANT implementations that incorporate smartNIC offloading. The first implementation, V1, takes an aggressive approach and offloads the computation within both the predictor and corrector stages

described above. In recognition of the fact that the corrector stage offloads may incur significant overheads, the second implementation, V2, employs a more conservative methodology and offloads only the predictor stage calculations. Figure 1 provides a simplified view of smartNIC offloaded functionality in each implementation.

IV. BigSort

A. Application Overview

We utilize the CORAL-2 BigSort benchmark [14] as the basis of our second case study on smartNIC offloading. The BigSort benchmark performs a parallel sort of a user defined number of 64-bit integer values into ascending order in a distributed environment. Parallelization is achieved through utilization of MPI to distribute sort values across available nodes, wherein one rank is used per node, while OpenMP is used to accelerate the local sort within a given node. The application is characterized by interleaved periods of computation, communication, and file I/O. User provided *DRAM_ALLOC* and *PAGE_SIZE* parameters supplied at runtime control the amount of memory utilized per rank to perform the sort, and the granularity at which the sort occurs, respectively. Although the behavior of the application may vary somewhat based on the provided parameters, in keeping with the benchmark's design of simulating scenarios wherein the total sort size exceeds the aggregated system memory capacity, the runtime is generally dominated by file I/O.

At a high level, the BigSort benchmark can be divided into three phases. In the first phase, each MPI rank reads in distinct segments of the file containing the unsorted integer values. This read data is then sorted into bins corresponding to the integer value ranges assigned to each MPI rank. MPI collective communication routines are then used to send the binned values to their proper destination where they are written to a bin file. This process is repeated until all of the unsorted values have been read from the original file and delivered to the appropriate destination rank.

In the second phase of the BigSort benchmark, each bin file constructed in phase one is sorted into ascending order locally by the associated MPI rank. Herein, segments of PAGE_SIZE are read in from the local bin files, sorted in parallel using OpenMP, and then merged together. This procedure continues in an iterative fashion until a single file per MPI rank, each containing the corresponding rank's sorted integers, is obtained. As the local sort size in this phase typically outstrips the available memory as defined by DRAM_ALLOC, a number of temporary files are utilized throughout the ongoing merge process. Files opened and written to as the output of a given iteration serve as the inputs for the subsequent cycle. Notably, the file operations performed in this phase are POSIX operations rather parallel file operations and ranks are blocked while the I/O is being performed. The final phase of the BigSort benchmark is also the simplest. Herein, parallel file operations are used to combine the individual files obtained through the local sort performed in phase two into one globally sorted file.

B. Offloading Strategy

Similar to PENNANT, we analyzed the BigSort benchmark for tasks potentially suitable for smartNIC offloading using an approach that incorporated both source code analysis and runtime profiling using TAU. From the profiles obtained with TAU, we observed that the majority of the BigSort application runtime was typically spent in the phase two local sort. However, analyzing the phase two source code revealed the existence of very little task level parallelism with respect to computation. Instead, we identified several POSIX file operations on temporary files that occur during the merging process as prospective offload targets. More specifically, within the phase two algorithm multiple temporary files are most often opened, written to, and closed during a given cycle. The rank executing these file I/O calls is blocked while the operations take place. However, as these files are not utilized again until the following cycle, the creation of the files can be safely decoupled to allow the sorting process to continue until the next cycle is reached.

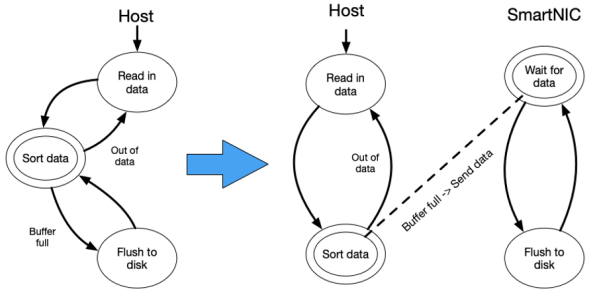

Fig. 2. State diagrams demonstrating BigSort phase two behavior for original (left) and offloaded (right) implementations.

Based on this finding, we offload the POSIX *open()*, *write()*, and *close()* operations associated with the creation of these files to the smartNIC for our experiments. In order to do so, we replicate the loop logic for the cycles within the phase two local sort in the smartNIC source code. A circular buffer of size DRAM_ALLOC is also allocated on the smartNIC and divided into DRAM_ALLOC/PAGE_SIZE elements. At the beginning of each cycle, the smartNIC opens the temporary files (up to a predetermined maximum number of files) needed for the given iteration and then waits to receive sorted data values from the host into its current buffer. As the sort progresses on the host, it fills its own PAGE_SIZE buffer with sorted values in the same manner as in the original implementation. However, when its buffer becomes full, rather than writing the data to the temporary file directly, the host instead sends the sorted values to the smartNIC and increments an internal counter corresponding to the next buffer element on the smartNIC. In this manner, the host is able to avoid blocking on file I/O operations and continue sorting data until the smartNIC runs out of available buffer space or the cycle is completed. When the smartNIC receives a series of sorted values from the host, it first performs the ordering verification that was originally performed on the host. The smartNIC then writes the data to the appropriate file. If this is the final write to the given file, the smartNIC also closes the file. Finally, the smartNIC sends a signal back to the host indicating that the given buffer can be reused. Figure 2 provides a simplified comparison of the phase two behavior for the original and offloaded BigSort implementations.

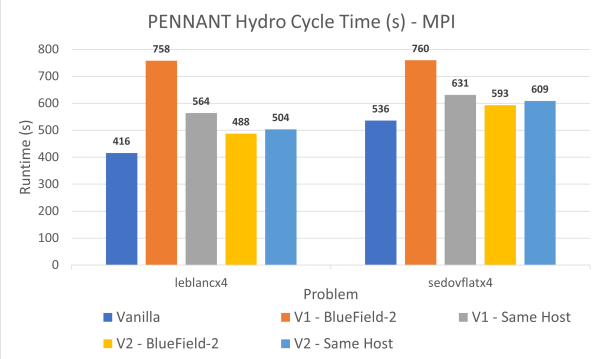

Fig. 3. Hydro cycle runtime for the PENNANT mini-app with only MPI-level parallelism

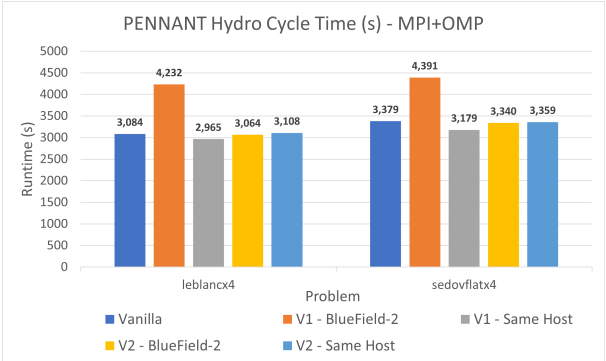

Fig. 4. Hydro cycle runtime for the PENNANT mini-app with MPI and OpenMP parallelism

V. Evaluation

A. Platform

In this section, we evaluate the efficacy of our smartNIC offloaded implementations of the PENNANT and BigSort applications. In order to do so, we perform a number of experiments to measure the performance of both the original applications and our smartNIC offloaded variants in a variety of scenarios. We then compare and analyze the associated results. For the experiments performed in this work, we utilize the HPC-AI Advisory Council [10] Thor cluster which pairs conventional host nodes together with NVIDIA BlueField-2 DPUs. We utilize Open MPI built upon UCX [26], [32] for communication. Further details of the Thor cluster and our software stack are given in Table I.

TABLE I
Thor Configuration

	Host	BlueField-2
CPU	Intel Xeon E5-2697A v4	Cortex-A72
Frequency	2.60 GHz	2.80 GHz
Cores	16	8
Sockets	2	1
Memory	256 GB	16 GB
Kernel	4.18.0-425.10.1	5.4.0-1049
Interconnect	InfiniBand HDR100	
Operating System	Rocky Linux 8.7	
Compiler	GCC 8.5.0	
UCX	v1.13.1	
MPI	Open MPI v4.1.4	

B. PENNANT Results

We evaluate our two smartNIC offloaded implementations of the PENNANT mini-app using 8 BlueField-2 enabled nodes of the Thor cluster. Performance data for the original, V1, and V2 implementations is gathered and compared when using MPI level parallelism exclusively as well as when utilizing MPI in conjunction with OpenMP. For each of our offloaded implementations, we measure performance using two different offload configurations in order to observe any changes in behavior. In the first offload configuration, each host resident rank performs offloading to a paired process on the associated BlueField-2 DPU. For our second offload configuration, host ranks instead perform offloads to processes pinned to a distinct socket within the same host node. The *leblancx4* and *sedovflatx4* problem inputs provided with PENNANT are utilized across experiments.

We first examine the results of our PENNANT trials conducted using only MPI level parallelism. For these experiments, 8 host ranks were utilized per node for a total of 64 host ranks. In the case of our offloaded implementations, this number is supplemented by an additional 64 ranks as appropriate to each offload configuration. Figure 3 demonstrates the results of these experiments. As shown, when running with the specified parameters, both of our smartNIC offloaded implementations demonstrated degraded performance as compared to the original implementation regardless of offload configuration. Our V1 implementation showcased the worst performance, particularly when performing offloading to the BlueField-2 DPUs. Here, the hydro cycle run time of the leblancx4 and sedovflatx4 problems increased by approximately 82% and 42%, respectively. While less egregious, V2 also decreased performance by approximately 11-21% across configurations. Based on these results, we performed further analysis with TAU in order to better understand this behavior. Herein, we discovered that the problem size operated on by each rank within these experiments had diminished such that the overhead of communication, even across shared memory, precluded the realization of any performance improvements. The more pronounced performance decrease for V1 as opposed to V2, which transfers far more data during the corrector stage, reinforces this conclusion.

We also conducted experiments with PENNANT using both MPI and OpenMP level parallelism. In these trials, a single rank on each of our 8 host nodes was utilized. Each of these ranks was paired with a single offloading process located either on the associated BlueField-2 DPU or remaining host socket as appropriate for the offload configuration. The host and offload processes utilized 16 and 8 threads, respectively. The results of these experiments, as shown in Figure 4, are somewhat orthogonal to those demonstrated by our MPI exclusive PENNANT trials. Here, our V1 implementation again demonstrates significant performance decreases when offloading to the BlueField-2 DPUs. In the case of leblancx4, an approximately 37% decrease is exhibited as compared to approximately 30% for sedovflatx4. However, when offloading is performed to a different socket within the same host, modest

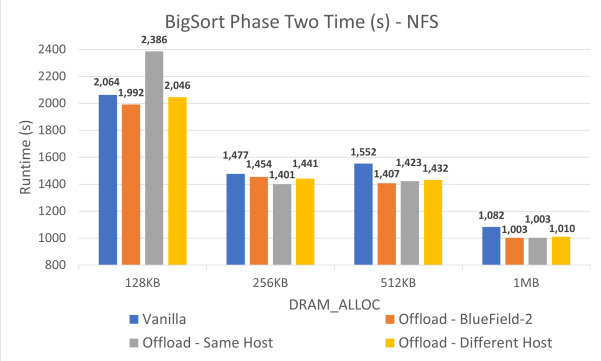

Fig. 5. Phase two runtime for the BigSort benchmark with different offload types using an NFS filesystem

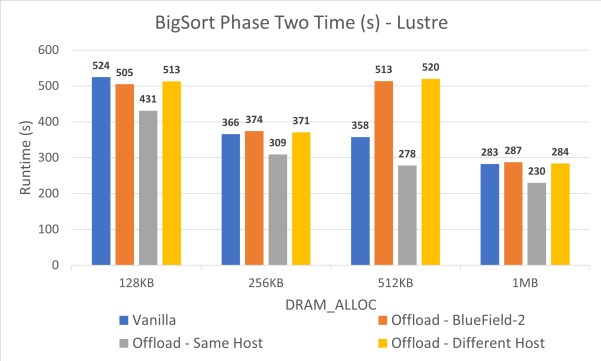

Fig. 6. Phase two runtime for the BigSort benchmark with different offload types using a Lustre filesystem

improvements of approximately 4% and 6% are realized, respectively. This suggests that, for the given problem size per rank, the computation decoupling strategy employed in V1 itself has some merit. However, the penalty of moving large amounts of data between devices significantly outweighs the benefits of the computation offloading. In contrast to V1, and our previous MPI exclusive results, the performance of our V2 implementation here differs minimally from the baseline. Across both problem inputs and offload configurations, performance differences of -0.8% - +1.2% are observed. In this implementation, offloading to the BlueField-2 DPUs slightly outperformed the host based offload scheme. Overall, results for the V2 implementation with these parameters suggest that while excessive data movement penalties were not introduced, neither was a sufficient amount of computation offloaded as to be useful.

C. BigSort Results

As with our PENNANT experiments, trials of our smartNIC accelerated BigSort implementation are conducted using 8 BlueField-2 enabled nodes. For each BigSort experiment, the host portion of the benchmark is run using 1 MPI rank per node and 32 OpenMP threads. Each host rank offloads to a single offload process that does not utilize multithreading. For trials of our offloaded implementation, we reuse the two distinct offload configurations utilized in our PENNANT experiments. We also add a third configuration wherein each host resident process offloads to a simulated smartNIC process on a distinct host node. Tests are performed using a total of 16 GB of unsorted integer values, which equates to a 2 GB phase two local sort per host rank, and an 8 KB PAGE_SIZE. The preceding parameters are kept constant across trials while the DRAM_ALLOC is varied from 128 KB to 1 MB. Notably, we test using small DRAM_ALLOC sizes in relation to the problem size in order to ensure I/O bound behavior.

During the initial stages of this research, platform limitations restricted us to the utilization of an NFS filesystem for our experiments. As such, we first demonstrate the performance of our solution using the NFS filesystem attached to the Thor cluster and then compare against results gathered using a high performance Lustre filesystem. The runtimes of phase two within the BigSort benchmark across different scenarios when using an NFS filesystem are given by Figure 5. As shown, when offloading was performed to the BlueField-2 DPUs, our solution was able to improve application performance by approximately 1.6% - 9.4%, where increased improvements were realized for the larger DRAM_ALLOC values. When offloading to another rank within the same host, performance decreased by 15.6% for a 128KB DRAM_ALLOC, but improved by 5.2% at 256KB. For the larger two sizes, results for same host offloading mirror those of the other offload configurations. Finally, offloads performed to a rank on a different host also improved performance, but generally did so to a slightly lesser degree than other offload configurations. Overall, our smartNIC offloaded implementation of the BigSort benchmark was able to achieve modest performance improvements over the original implementation when paired with an NFS filesystem.

In contrast to the above, results gathered using a more HPC appropriate Lustre filesystem, as demonstrated in Figure 6, tell a different story. Here, we see that both the smartNIC offload configuration and the different host offload configuration are only able to improve performance when using a 128 KB DRAM_ALLOC. Moreover, in this case they do so only by a relatively small 3.7% and 2.2%, respectively. For 256 KB and 1 MB DRAM_ALLOC values, these configurations result in a slight decrease in performance while a significant slowdown is observed for both configurations at 512 KB. Orthogonally, offloading to a process within the same host uniformly improves performance. In this configuration, improvements of 15.5% - 22.2% are demonstrated across different DRAM_ALLOC sizes. Based on this behavior, we conclude that the Lustre filesystem is most likely able to optimize repeated file operations to the same files when those operations originate from the same device. Such an optimization would explain why only offloads performed to the same host were generally beneficial for BigSort when utilizing the Lustre filesystem.

VI. Insights

Throughout the course of our research we have identified a number of insights that we believe will prove critical to efforts to harness smartNICs as general purpose HPC accelerators. We summarize these insights below in the hope they will prove useful in guiding future endeavors.

1) **SmartNIC offloads must be carefully chosen and coordinated.** As the compute capabilities of most smart-

NICs are less extensive than those of their associated hosts, naively offloading computationally intensive workloads is unlikely to prove beneficial. Instead, smartNIC offloads should largely be restricted to task parallel workloads that can be effectively overlapped with other host-resident work. If insufficient work exists to occupy the host for the duration of a proposed offload, the offload is most often inadvisable.

2) **Mechanisms that assist in determining where smartNIC offloading may be advantageous are needed.** Successfully leveraging smartNIC offloading as described above currently necessitates a painstaking analysis of the application and system in question by the end user. The ability to effectively conduct such an analysis, however, is largely limited to subject matter experts and not without potential pitfalls. Moreover, such an approach is time consuming and not scalable as a general solution. As such, tools that can help quantify the performance characteristics of different smartNICs and their platforms, as well as provide insight into how applications map to these characteristics, will be critical for smartNIC offloading. The OpenHPCA benchmark [29], [30] and Clara [22], [23] represent prominent examples of efforts in this domain.
3) **SmartNIC acceleration requires an accessible, performant interface.** At present, minimal support exists for enabling generalized smartNIC offloading. In this work, we utilized a MPMD programming model based on MPI to perform our experiments. Although this approach provided a solution agnostic of any device specific dependencies, it also compelled a significant amount of code refactoring. The resulting applications are also difficult to both debug and understand. Further, utilization of this high level abstraction incurred overheads that might have been avoided with a more optimal solution. Adoption of emerging smartNIC interfaces such as OpenSNAPI [31] and DOCA [19], which provide simple yet optimized APIs, will therefore be critical to the success of smartNICs.
4) **SmartNIC offloading necessitates an efficient channel for data sharing between the host and smartNIC** As described above, smartNIC offloading must be thoughtfully coordinated with respect to the associated host. As part of this paradigm, and in contrast to GPGPU-based acceleration, the host and smartNIC are often utilized in tandem to progress independent tasks within a larger workload. Data dependencies between devices are therefore common. As such, if a sufficiently high performance methodology for enabling communication between the devices is not present, costs associated with data movement may become insurmountable.

VII. Related Work

Alongside the emergence of this new generation of smartNICs, considerable research efforts have been undertaken to discover the full range of capabilities of these devices as well as novel use cases. Grant et al. provide a primer on this new class of devices and detail scenarios in which specific features may prove advantageous [7]. sPIN [9] defines a programming model for smartNIC packet processing, based on the concept of handler functions, that is optimized for tasks associated with data movement. The subsequent study, PsPIN [3], implements an open source sPIN SoC based on RISC-V. INCA [25] describes an in-network processing model built on smartNIC tag-matching, atomics, and triggered operations that is deadline-free and complementary to streaming data models such as sPIN. Clara [22], [23] provides an automated mechanism for predicting the suitability and performance of network function offloads to smartNICs through an approach that employs source code analysis and machine learning. Similarly, the Floem [21] framework supplies an infrastructure that seeks to simplify the process of developing smartNIC-accelerated applications. Orthogonally, the Two-Chains and Three Chains frameworks [8], [16], [20] represent novel approaches to dynamically offloading tasks to accelerators, such as smartNICs, in heterogeneous environments.

With respect to NVIDIA's BlueField-2 DPUs, Liu et al. conduct an extensive evaluation to quantify the devices' network and compute capabilities [15]. BluesMPI [1], [24] utilizes BlueField-2 DPUs to realize optimized implementations of nonblocking alltoall, allgather, and broadcast collectives built on RDMA read and write operations. In contrast, Jain et al. explore offloading computation to BlueField-2 DPUs in order to accelerate the training of deep learning models [12]. Finally, Karamati et al. perform an investigation similar to our own wherein they evaluate offloading to BlueField-2 DPUs in the context of the YASK and miniMD applications [13].

VIII. Conclusion

In this work, we explored the viability of smartNICs as general-purpose accelerators for high performance computing environments. We demonstrated a methodology for prototyping application-level smartNIC offloads built upon an MPI-based MPMD programming model that is fully agnostic of system specific requirements. We then applied this methodology to two applications as a case study for smartNIC offloading. Using the PENNANT mini-app, we attempted to offload computational kernels in the context of a tightly coupled scientific application. Herein, we found that overheads associated with data movement between devices largely inhibited any benefits from a coprocessing model. For the BigSort benchmark, we took an orthogonal approach and utilized the smartNICs as asynchronous engines for performing file I/O operations. Although this approach demonstrated some merit, unanticipated optimizations within the Lustre filesystem conflicted with our approach in the most realistic HPC scenarios. In conclusion, we believe emerging smartNIC designs possess great potential for improving performance in heterogeneous HPC environments. However, as detailed in Section VI, utilizing these devices effectively will require both an understanding of their capabilities as well as more performant and intuitive interfaces.

IX. Acknowledgements

The authors are grateful to the United States Department of Defense and Los Alamos National Laboratory for supporting this work. We would also like to thank the HPC-AI Advisory Council for use of the Thor cluster. This work is authorized for release under LA-UR-23-30100.

References

[1] M. Bayatpour, N. Sarkauskas, H. Subramoni, J. Maqbool Hashmi, and D. K. Panda, "Bluesmpi: Efficient mpi non-blocking alltoall offloading designs on modern bluefield smart nics," in *High Performance Computing*, B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief, and P. Luszczek, Eds. Cham: Springer International Publishing, 2021, pp. 18–37.

[2] Broadcom, "Broadcom Stingray SmartNIC Accelerates Baidu Cloud Services," https://www.broadcom.com/company/news/product-releases/53106, 2020, accessed 2023-05-06.

[3] S. Di Girolamo, A. Kurth, A. Calotoiu, T. Benz, T. Schneider, J. Beránek, L. Benini, and T. Hoefler, "A risc-v in-network accelerator for flexible high-performance low-power packet processing," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21. IEEE Press, 2021, p. 958–971. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00079

[4] C. R. Ferenbaugh, "Pennant: an unstructured mesh mini-app for advanced architecture research," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 4555–4572, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3422

[5] C. R. Ferenbaugh, "Performance evaluation of unstructured mesh physics on advanced architectures," in *2015 IEEE International Conference on Cluster Computing*. Chicago, IL, USA: IEEE, 2015, pp. 721–728.

[6] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure accelerated networking: SmartNICs in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 51–66. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/firestone

[7] R. E. Grant, W. Schonbein, and S. Levy, "Radd runtimes: Radical and different distributed runtimes with smartnics," in *2020 IEEE/ACM Fourth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware (IPDRM)*, 2020, pp. 17–24.

[8] M. Grodowitz, L. E. Peña, C. Dunham, D. Zhong, P. Shamis, and S. Poole, "Two-chains: High performance framework for function injection and execution," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 377–387.

[9] T. Hoefler, S. Di Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "Spin: High-performance streaming processing in the network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126970

[10] HPC-AI Advisory Council, "Hpc-ai advisory council website," https://www.hpcadvisorycouncil.com/index.php, 2023, accessed 2023-05-08.

[11] J. Hypolite, J. Sonchack, S. Hershkop, N. Dautenhahn, A. DeHon, and J. M. Smith, "Deepmatch: Practical deep packet inspection in the data plane using network processors," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 336–350. [Online]. Available: https://doi.org/10.1145/3386367.3431290

[12] A. Jain, N. Alnaasan, A. Shafi, H. Subramoni, and D. K. Panda, "Accelerating cpu-based distributed dnn training on modern hpc clusters using bluefield-2 dpus," in *2021 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2021, pp. 17–24.

[13] S. Karamati, J. Young, T. Conte, K. S. Hemmert, R. Grant, C. Hughes, and R. Vuduc, "Computational offload with bluefield smart nics," 10 2021. [Online]. Available: https://www.osti.gov/biblio/1832297

[14] Lawrence Livermore National Laboratory, "Coral-2 benchmarks," https://asc.llnl.gov/coral-2-benchmarks, 2023, accessed 2023-05-08.

[15] J. Liu, C. Maltzahn, C. D. Ulmer, and M. L. Curry, "Performance characteristics of the bluefield-2 smartnic," 5 2021. [Online]. Available: https://www.osti.gov/biblio/1783736

[16] W. Lu, L. E. Pena, P. Shamis, V. Churavy, B. Chapman, and S. Poole, "Bring the bitcode-moving compute and data in distributed heterogeneous systems," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2022, pp. 12–22. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CLUSTER51413.2022.00017

[17] Marvell, "Marvell Octeon 10 DPU Platform Product Brief," https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-octeon-10-dpu-platform-product-brief.pdf, 2021, accessed 2023-05-06.

[18] NVIDIA, "BlueField Data Processing Units," https://www.nvidia.com/en-us/networking/products/data-processing-unit/, 2023, accessed 2023-05-06.

[19] NVIDIA, "Nvidia doca sdk," https://developer.nvidia.com/networking/doca, 2023, accessed 2023-05-06.

[20] L. E. Peña, W. Lu, P. Shamis, and S. Poole, "Ucx programming interface for remote function injection and invocation," in *OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Exascale and Smart Networks*, S. Poole, O. Hernandez, M. Baker, and T. Curtis, Eds. Cham: Springer International Publishing, 2022, pp. 144–159.

[21] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for nic-accelerated network applications," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 663–679.

[22] Y. Qiu, Q. Kang, M. Liu, and A. Chen, "Clara: Performance clarity for smartnic offloading," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, ser. HotNets '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 16–22. [Online]. Available: https://doi.org/10.1145/3422604.3425929

[23] Y. Qiu, J. Xing, K.-F. Hsu, Q. Kang, M. Liu, S. Narayana, and A. Chen, "Automated smartnic offloading insights for network functions," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 772–787. [Online]. Available: https://doi.org/10.1145/3477132.3483583

[24] N. Sarkauskas, M. Bayatpour, T. Tran, B. Ramesh, H. Subramoni, and D. K. Panda, "Large-message nonblocking mpi_iallgather and mpi ibcast offload via bluefield-2 dpu," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. Bengaluru, India: IEEE, 2021, pp. 388–393.

[25] W. Schonbein, R. E. Grant, M. G. F. Dosanjh, and D. Arnold, "Inca: In-network compute assistance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356153

[26] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss *et al.*, "Ucx: an open source framework for hpc network apis and beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, IEEE. Santa Clara, CA, USA: IEEE, 2015, pp. 40–43.

[27] S. S. Shende and A. D. Malony, "The tau parallel performance system," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, p. 287–311, may 2006. [Online]. Available: https://doi.org/10.1177/1094342006064482

[28] K. Taranov, B. Rothenberger, A. Perrig, and T. Hoefler, "sRDMA – efficient NIC-based authentication and encryption for remote direct memory access," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. Virtual: USENIX Association, Jul. 2020, pp. 691–704. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/taranov

[29] Unified Communication Framework Consortium, "Open high performance compute availability benchmark," https://ucfconsortium.org/projects/hpca-benchmark/, 2023, accessed 2023-05-08.

[30] Unified Communication Framework Consortium, "Open high performance compute availability benchmark repository," https://github.com/openucx/openhpca, 2023, accessed 2023-05-08.

[31] Unified Communication Framework Consortium, "Open smart network api," https://ucfconsortium.org/projects/opensnapi/, 2023, accessed 2023-05-08.

[32] Unified Communication Framework Consortium, "The Unified Communication X Library," https://openucx.org, 2023, accessed 2023-05-06.